Microprocessor and Assembly Language

Chapter-Two

Internal Microprocessor Architecture

Outline

- Introduction
- Programming model of microprocessors
    - ✓ Multipurpose registers
    - ✓ Special purpose registers
    - ✓ Segment Registers
- 8086/8088 flags
- Memory addressing with segmentation

  - In the real mode

  - In the protected mode
- Memory management and paging

The Intel Family

Addressable Memory, bytes

$= 2^A$

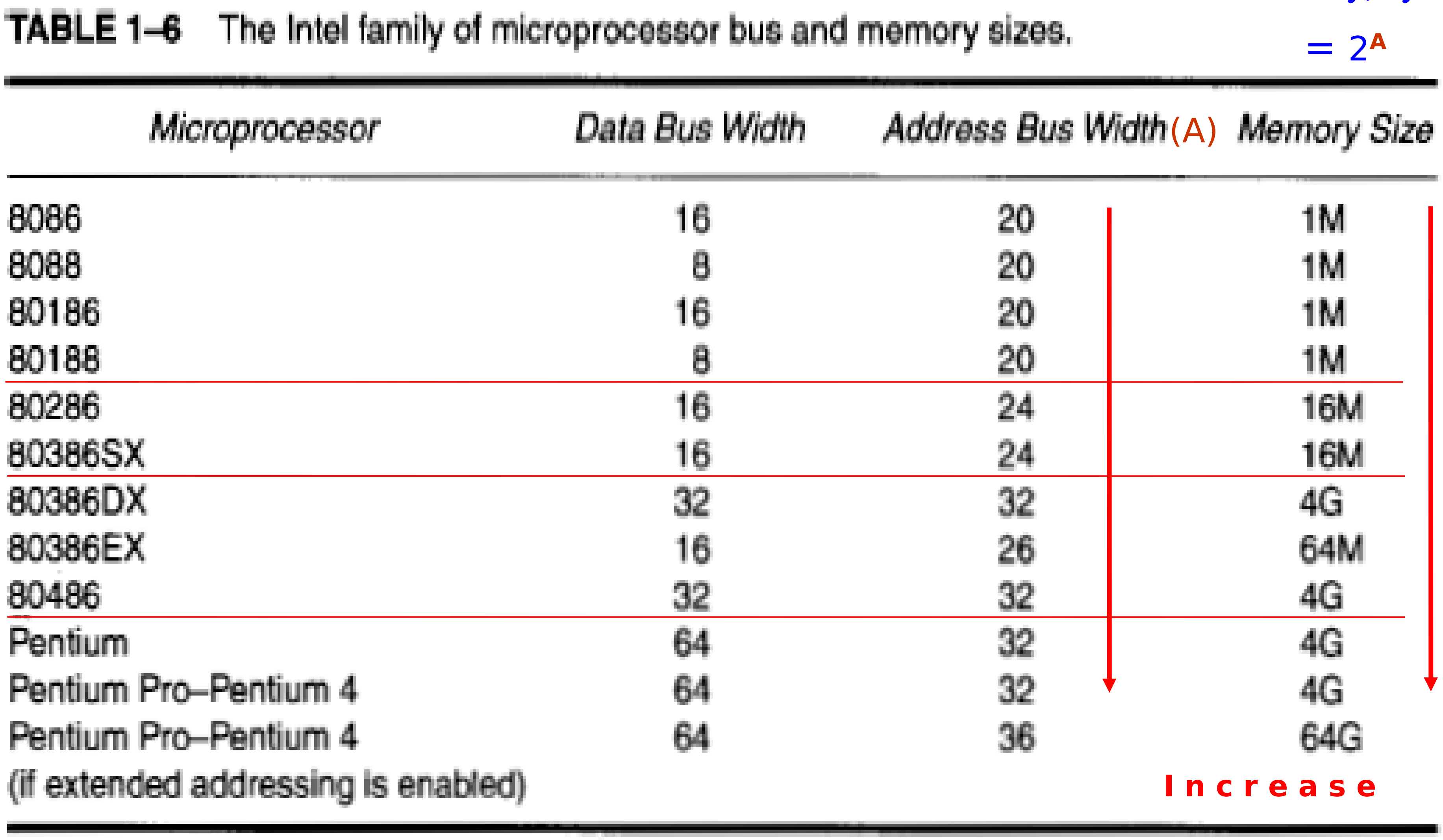


TABLE 1–6 The Intel family of microprocessor bus and memory sizes.

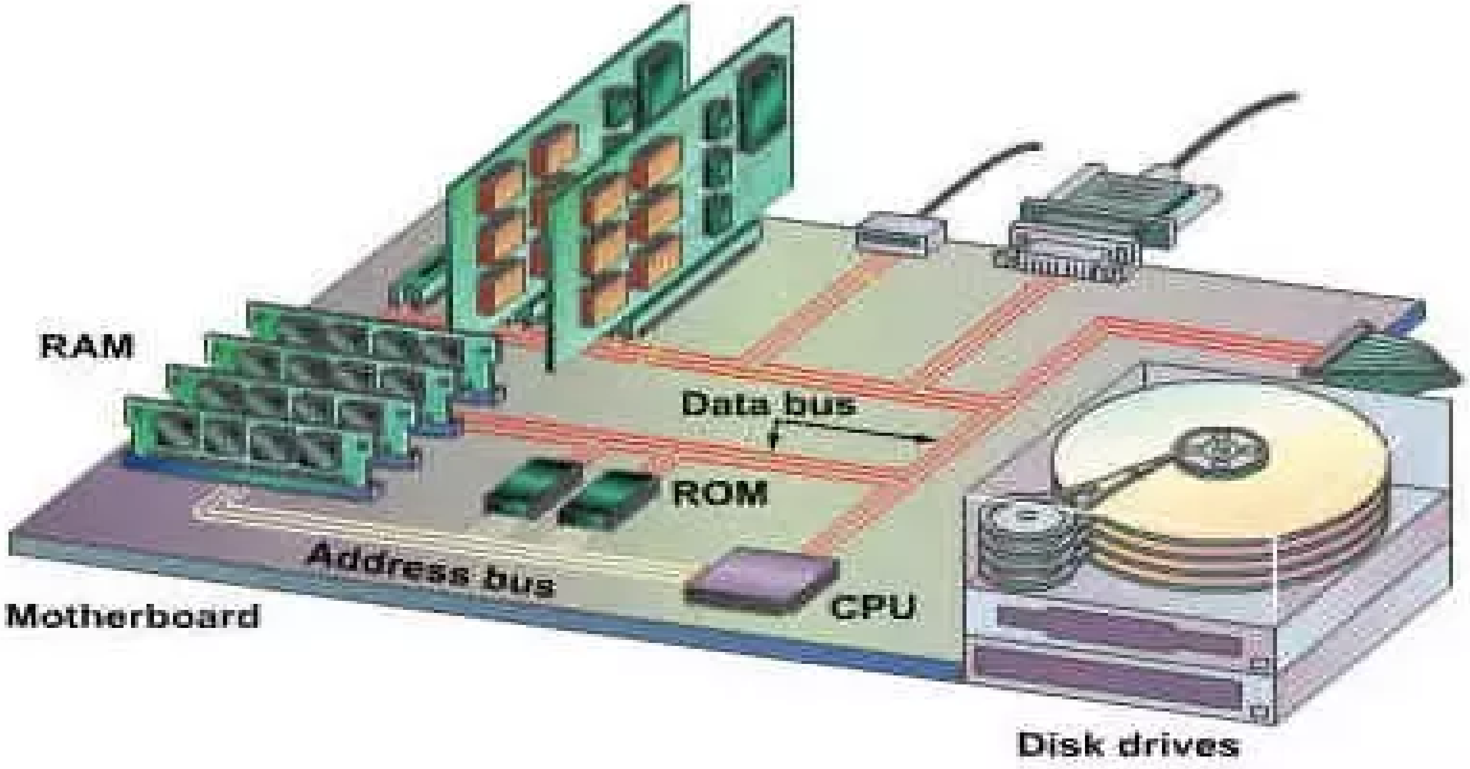
Microprocessor	*Data Bus Width*	*Address Bus Width* (A)	*Memory Size*
8086	16	20	1M
8088	8	20	1M
80186	16	20	1M
80188	8	20	1M
80286	16	24	16M
80386SX	16	24	16M
80386DX	32	32	4G
80386EX	16	26	64M
80486	32	32	4G
Pentium	64	32	4G
Pentium Pro–Pentium 4	64	32	4G
Pentium Pro–Pentium 4 (if extended addressing is enabled)	64	36	64G

Increase

Introduction

- As we have studied earlier, there are different types of microprocessor each of them having different internal architecture.
- Microprocessor is an Integrated Circuit with all the functions of a CPU however, it **cannot be used stand alone** since unlike microcontroller it has no memory or peripherals.
- That is, microprocessor does not have a RAM or ROM inside it.
- However, it has internal registers(which we will see later in this chapter) for **storing intermediate** and final results and interfaces with memory located outside it through the Bus.
- Let us first have a look at what is the bus?
- **Bus** is a communication link used to send **address**, **control** and **data** between the processor and other components.

What is Bus?



Cont...

- The computer system works based on the size of buses.
- The **address bus** is used by CPU to send the address of the memory location.
- The maximum number of memory locations that can be accessed in a system is determined by the **number of lines of an address bus**.
- Each of the lines of bus carry **either 0** or **1** at a time. And hence a group of lines together carry a group of bits at the same time.
- In general, address bus of **n lines** can be addressed at most **2^n locations**.
- That is; **16-bit address bus** can allow access to **2^{16} memory locations** or **64 K** Byte of memory(assuming that each address stores 8bit(1byte) of data).
- Therefore, we can conclude that the width of the address bus determines the amount of memory a system can address.
- The wider the address bus, the more memory a computer can use.

Cont...

- The **Data bus** helps to transfer data between various components.
- The data bus consists of 8, 32,64, etc. separate lines.
- The number of lines refers to the **width of the data bus**. This bus width helps to determine the **data transferring rate**.
- Therefore, the data bus width determines the system performance, but it is expensive to increase the number of lines.
- The more wires you use, the more costly the computer system will become - it's like the number of lanes in a high way system.
- **Width of the data bus** = **# wires** used in the data bus.

Effect of the width(number of wires) of the databus:

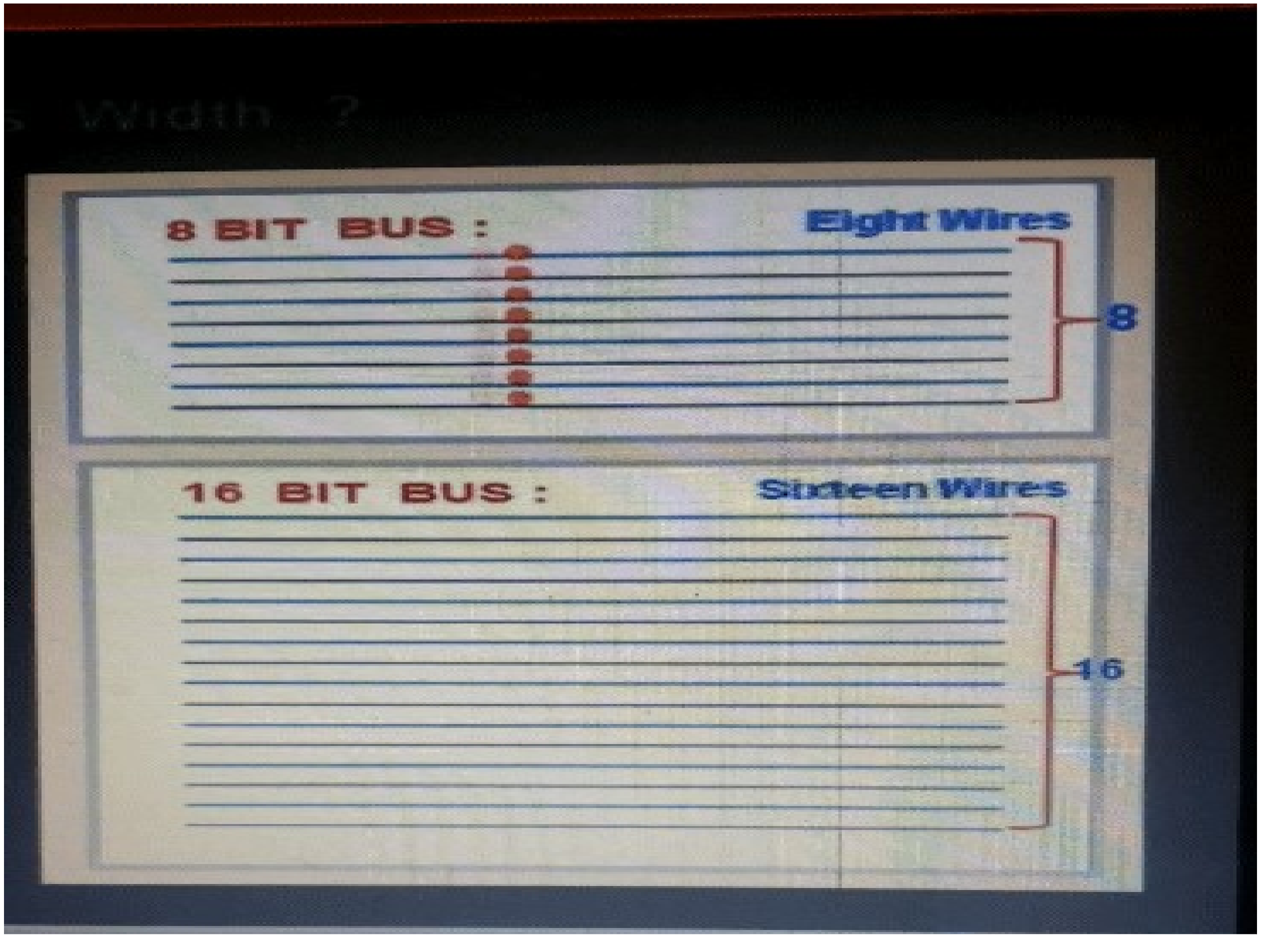
- A data bus that consists of **8 bits**, can transfer **1 byte** of data per read/write operation.
- A data bus that consists of **16 bits**, can transfer **2 bytes** of data per read/write operation.

Cont...

- A data bus that consists of **32 bits**, can transfer **4 bytes** of data per read/write operation.
- And so on...
- Therefore, we can conclude that the **width of the data bus** determines the amount of data transferred per memory transfer operation.
- That is, the wider (= more wires) the the data bus, the more data you can transfer per time unit (second).
    - This will result is a faster running computer.
- The *width of the data bus* is analogous to the **# lanes on the high way:** the more lanes (= more wires), the more cars (more bits) you can transfer per time unit).

Cont...

- In the bus with 8wires, 8bit sized data can travel at a time.
- In the bus with 16wires, 16bit sized data can travel.



The Programming Model

- Before a program is written or any instruction is investigated, the **internal configuration** of the microprocessor must be known.
- It is this programming model that shows the CPUs internal configuration.
- In a computer system, the programming model shows what the CPU has available to a programmer for the execution of computer programs.
- It covers the CPU resources such as:
  - Registers,
  - Operating modes etc.. that are used for execution of the CPU's instruction set.

Cont…

- This programming model contains two types of registers(based on the way they are accessed by the program).

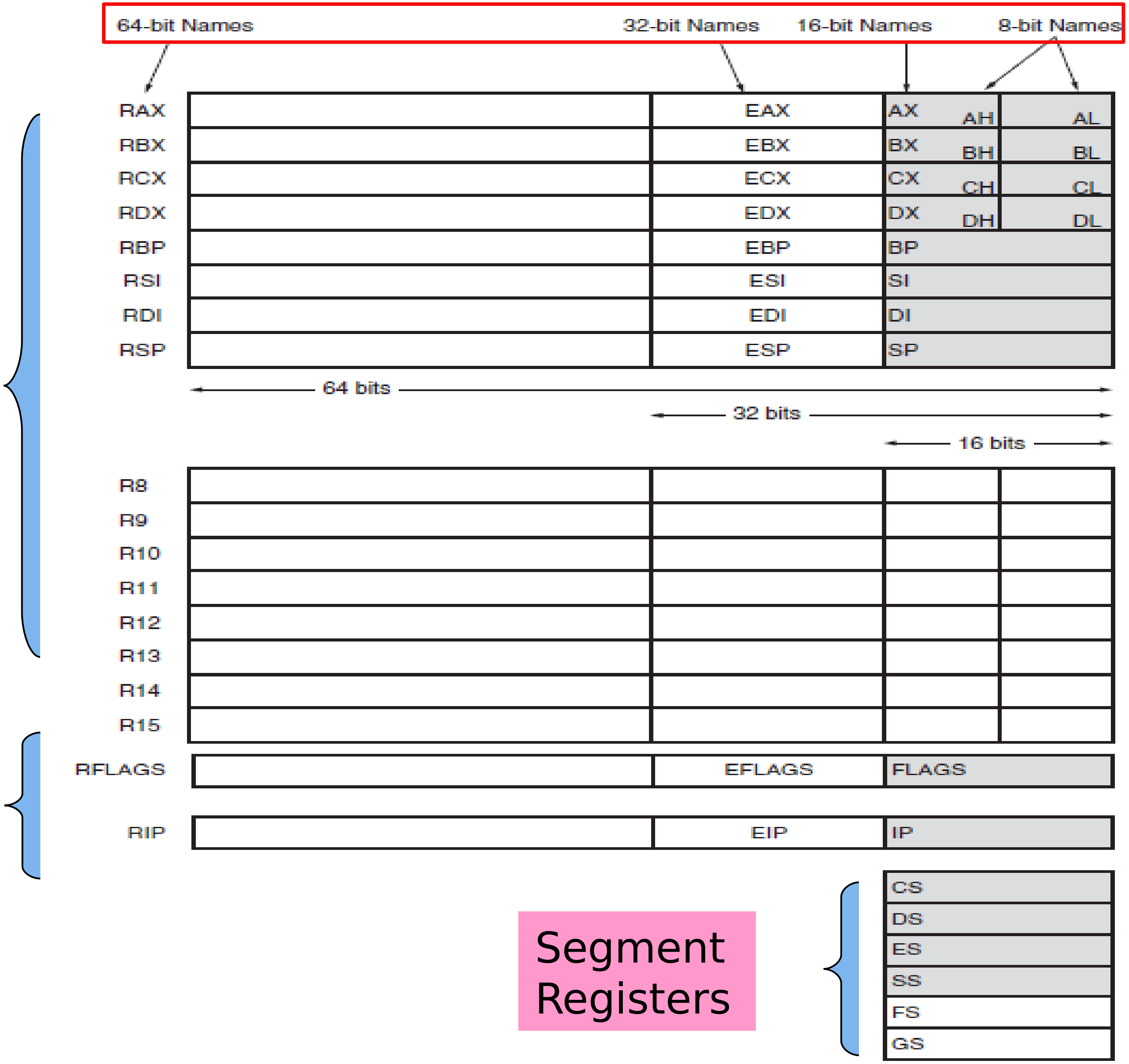
⇨ **Program visible registers:** this are registers that are used during application programming and are specified by the instructions.

Example: the programming model of 8086 MP.

⇨ **Program invisible registers:** this are registers that are used to control and operate the protected memory system and other features of the microprocessor. Used in 80286 and above MP's.

- This registers are not addressable directly during application programming, but may be used indirectly during system programming.

Programming model of Microprocessor



64-bit Names	32-bit Names	16-bit Names	8-bit Names	
RAX	EAX	AX	AH	AL
RBX	EBX	BX	BH	BL
RCX	ECX	CX	CH	CL
RDX	EDX	DX	DH	DL
RBP	EBP	BP		
RSI	ESI	SI		
RDI	EDI	DI		
RSP	ESP	SP		
R8				
R9				
R10				
R11				
R12				
R13				
R14				
R15				
RFLAGS	EFLAGS	FLAGS		
RIP	EIP	IP		
		CS		
		DS		
		ES		
		SS		
		FS		
		GS		

General Purpose Registers

Special Purpose Registers

Segment Registers

- **The earlier 8086, 8088, and 80286 contain 16-bit internal architectures.**

Programming model(Cont…)

- The **shaded areas** in the above illustration represent registers that are found in early versions of the **8086**, **8088**, or **80286** microprocessors. That is,
  - They contain **16-bit** internal architectures.
- The **80386** through to **80586** microprocessors contain full **32-bit** internal architectures.
- The **8-bit** registers are AH, AL, BH, BL, CH, CL, DH, and DL and are referred to when an instruction is formed using these two-letter designations.

For example, an **ADD AL, AH** instruction adds the 8-bit contents of AH to AL.

- Only AL changes due to this instruction.

Programming model(Cont…)

- The **16-bit** registers are AX, BX, CX, DX, SP, BP, DI, SI, IP, FLAGS, CS, DS, ES, SS, FS, and GS.

☛ **Note:** The first 4 16-bit registers contain a pair of 8-bit registers. An example is AX, which contains AH and AL.

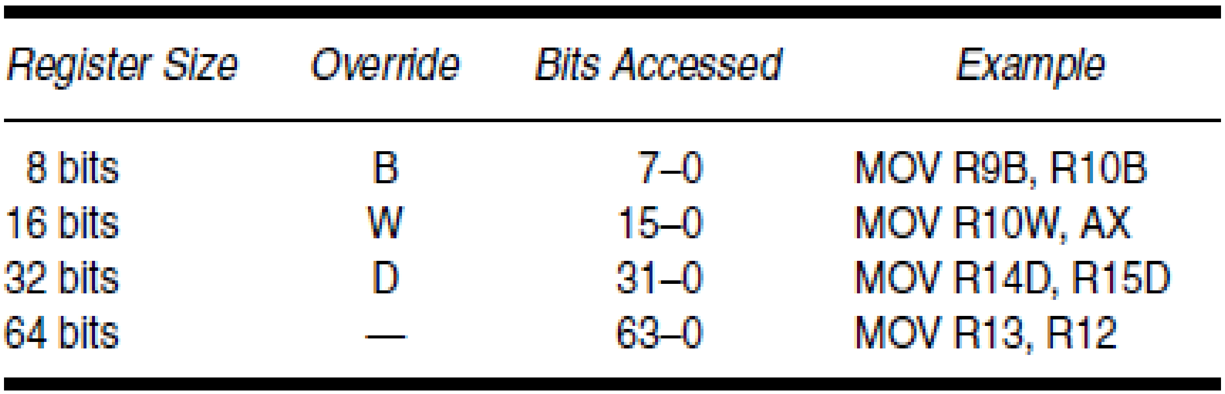
- The **extended 32-bit** registers are EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI, EIP, and EFLAGS.

☛ **Note:** These 32-bit extended registers, and 16-bit registers FS and GS, are available only in the 80386 and above.

- The **64-bit** registers are designated as RAX, RBX, RCX, RDX, and so forth. In addition to the renaming of the registers for 64-bit widths, there are also additional 64-bit registers that are called R8 through R15.

Programming model(Cont…)

- R8 through R15 have no provision for directly addressing bits 8 through 15 as a byte.
  - To access the low-order byte of the R8 register, use R8B (where B is the low-order byte).
  - Likewise, to access the low-order word of a numbered register, such as R10, use R10W in the instruction.
  - The letter **D** is used to access a **doubleword**.
- Example, MOV R11D, R8D. copies the low-order doubleword from R8 to R11.



Register Size	*Override*	*Bits Accessed*	*Example*
8 bits	B	7–0	MOV R9B, R10B
16 bits	W	15–0	MOV R10W, AX
32 bits	D	31–0	MOV R14D, R15D
64 bits	—	63–0	MOV R13, R12

Multipurpose Registers

- This type of registers are used for temporary storage of data and address of the data during manipulation. They **don't have specific purpose**. These are:

- **RAX (accumulator)** is referenced as a 64-bit register (RAX), a 32-bit register (EAX), a 16-bit register (AX), or as either of two 8-bit registers (AH and AL).

  ✓ Accumulator is used for instructions such as multiplication, division, and some of the adjustment instructions.

  Example: ADD **AX**, AX (**AX** = AX + AX)

- **RBX (base index)** is addressable as RBX, EBX, BX, BH, or BL.

  ✓ The BX register sometimes holds the offset address of a location in the memory system in all versions of the microprocessor.

  - Offset address is a logical address.

Multipurpose Registers(Cont…)

- **RCX(count)** which is addressable as RCX, ECX, CX, CH, or CL, is a general-purpose register that holds the count for various instructions.

✓ This serves as a loop counter. Program loop constructions are facilitated by it.

Example: MOV CX, 0005
LOOP

- **RDX(data)** addressable as RDX, EDX, DX, DH, or DL, holds part of the result from a multiplication or part of the dividend before a division.

Example: MUL BX (DX, AX = AX * BX)

- **RBP (base pointer)** addressable as RBP, EBP, or BP, points to data in stack segment(SS).

Multipurpose Registers(Cont…)

- **RDI (destination index)** addressable as RDI, EDI, or DI, often addresses string destination data for the string instructions.
  - ✓ Used in operations that contain String instruction (MOVS, CMPS).
- **RSI (source index)** is used as RSI, ESI, or SI. It often addresses source string data for the string instructions.
  - ✓ Used in operations that contain String instruction (MOVS, CMPS).
- **R8 through R15** only found in the Pentium 4 and Core2 if 64-bit extensions are enabled, and are *reserved* for temporary data storage.
  - ✓ Most applications will not use these registers until 64-bit processors are common.

Special-purpose registers

- This registers, unlike general purpose register, their purpose is predetermined during manufacturing process and can't be changed.
- **RIP (instruction pointer)** stores the address of the next instruction that is to be executed. Also serve as offset for code segment(CS).
  - ✓ This register is IP (16 bits) when the microprocessor operates in the real mode and EIP (32 bits) when the microprocessor operates in the protected mode.
  - ☛ **Note:** the 8086, 8088, and 80286 do not contain an EIP register.
- **RSP (stack pointer)** addresses an area of memory called the stack. (points to the current top value of the Stack). The stack memory stores data through this pointer. Used as offset for Stack Segment.
- **EFLAGS** – indicates latest conditions (state) of the microprocessor (FLAGS). Several types are there as listed below.

8086/8088 flags

- Depending upon the value of result after any arithmetic and logical operation the flag bits become **set (1)** or **reset (0)**. There are different types of flags. These are:
- **C (carry flag)** holds the carry after addition or the borrow after subtraction.

Example: Assume we want to add two 8 bit numbers and save the result in 8 bit register.

✓ If the two numbers are 255 + 9 = 264 which is more than what 8 bit register can store. So the value "8" will be saved as a carry and the **CF flag will be set**.

- **P (parity flag)** Parity is a **logic 0 for odd parity** and a **logic 1 for even parity**.
  - Parity is the count of ones in a number expressed as even or odd. *For example*, 111 has odd parity.

8086/8088 flags

- **Z (zero flag)** shows that the **result of an arithmetic** or **logic operation** is zero. That is;
  - If Z=1,the result is zero; if Z=0, the result is not zero. *Example*: 0001 – 0001 = 0000 Thus, Z=1
- **S (sign flag)** holds the **arithmetic sign of the result** after an arithmetic instruction executes. That is;
  - If S=1, the sign bit (leftmost bit of a number) is set or negative;
  - if S=0, the sign bit is cleared or positive.

8086/8088 flags(Cont…)

- **I(interrupt flag)** controls the operation of the INTR (interrupt request) input pin.
  - ✓If I=1, the INTR pin is enabled; if I=0 , the INTR pin is disabled.
- **D(direction flag)** selects either the increment or decrement mode for the DI and/or SI registers during string instructions (Controls the left to right or right to left direction processing of string instruction).
  - ✓If D=1, the registers are automatically decremented; if D=0, the registers are incremented.
- **O (overflow flag)** An overflow indicates that the result has exceeded the capacity of the machine.

Segment register

- We know that we create segments in memory (segmentation).
- To get the starting location of this segments we store the address of this segments in segment register .
- In short segment registers **store the starting location of segments** present in memory i. e segment address.
- **CS (code)** holds the code (programs and procedures) used by the microprocessor. It defines the starting address of the section of memory holding code.
- **DS (data)** It contains most data used by a program.
  - Data are accessed in the data segment by an offset address.
- **ES (extra)** is an additional data segment that is used by some of the **string instructions** to hold destination data.
- **SS (stack)** It defines the area of memory used for the stack.
  - The stack entry point is determined by **SS** and **SP** registers.

Operating modes of microprocessor

- Processor modes **controls how the processor sees and manipulates the system memory** and the tasks that use it.
- There are two modes of processor operation:

1. Real mode
2. Protected mode

- Real mode is the only mode available on the 8086/8088 MP.

  i.e only the 8086 and 8088 operate **exclusively in the real mode**.

✓ It allows the microprocessor to address only the first 1M byte of memory space(8086/8088 has 20bit address bus, hence **2^{20}=1Mb**).

✓ DOS operates in real mode.

✓ It offers memory addressability of 1 MB of physical memory.

✓ This mode handles only one task at a time, and its characterized by 20bit segmented memory address space.

Real Mode (Cont...)

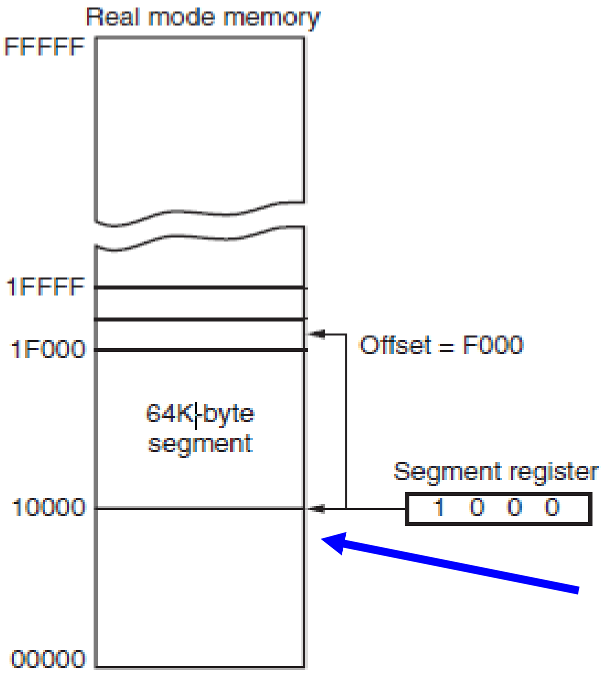
- But the 80286 and above operate in either the real or protected mode.

Segment and Offset Registers

- A combination of a **segment address** and an **offset address** accesses a memory location in the real mode.
- A segment is either:
  - 64K (2^{16}) bytes of fixed length (real mode), or
  - Up to 4G (2^{32}) bytes of variable length (protected mode).
- The segment address, located within one of the segment registers, defines the beginning address of any 64K-byte memory segment(in real mode).
- The offset address selects any location within the 64K byte memory segment.

Real Mode (Cont…)

- Figure below shows how the segment plus offset addressing scheme selects a memory location.



Segment Register	*Starting Address*	*Ending Address*
2000H	20000H	2FFFFH
2001H	20010H	3000FH
2100H	21000H	30FFFH
AB00H	AB000H	BAFFFH
1234H	12340H	2233FH

↳ Memory segment that begins at location **10000H** and end at location **1FFFFH** , 64Kb(FFFF=65535).

↳ Shows how an offset address of **F000H** selects location **1F000H** as a physical address.

↳ Segment register contains 1000,yet it addresses a starting segment at location 10000H. In the **real mode**, each segment register is appended with a **0H** on its rightmost end.

↳Once the beginning address is known, the **ending address** is found by adding **FFFFH**.

✓ Since it's real mode it offers memory addressability of 1Mb(FFFFF=1048575).

Real Mode (Effective Address calculation)

- Effective/Physical address is used to specify how the memory location would be selected/computed in real mode.

Effetive Address = Segment Register (SR) x 0H + Offset addres

- For instance on the above table we can see that the segment register(1000) and offset address(F000H) selects a memory location(1F000H) which is physical Address. That is,

EA = 1000**0H** + F000H = **1F000H**

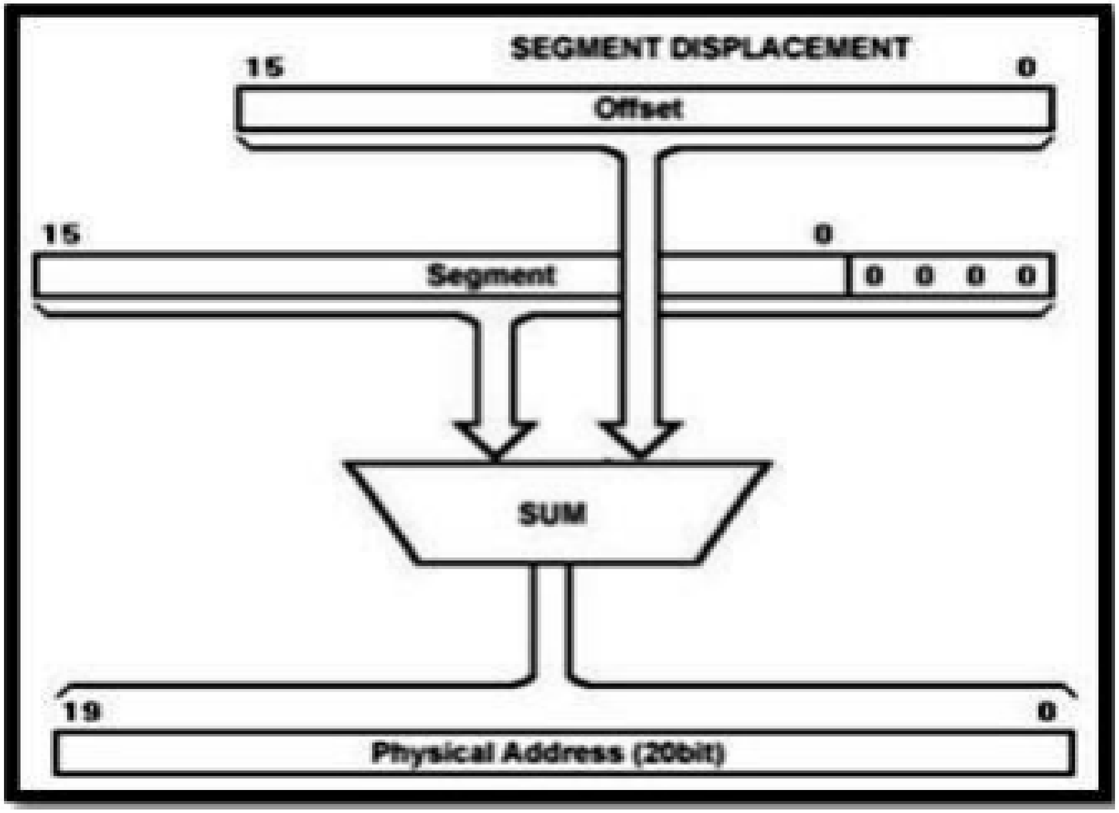
Example 2: If SR: 1000H, offset = 0023, what is EA?

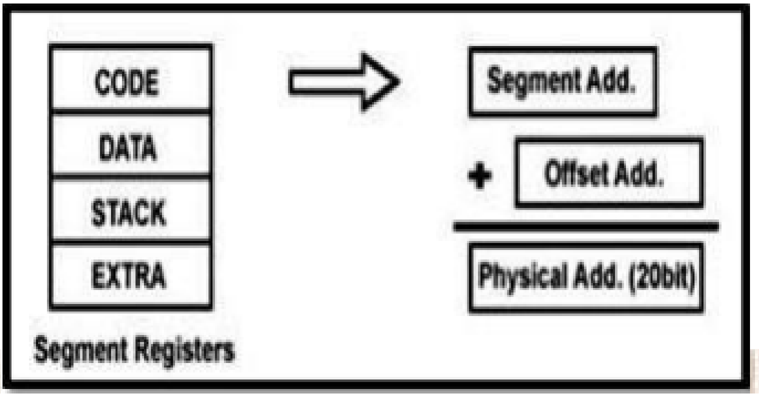
EA = 1000**0H** + 0023 = **10023H**

✓ **Offset address** is a **logical address**, 16bit in size, used to address specific location within memory.

✓ **Physical address** is a **real address**, 20bit in size, used to map directly anywhere into the 1 Mb memory space.

Real Mode (Effective Address calculation)





Default segment-offset registers combination

- The microprocessor has a set of rules, these rules, define the segment register and offset register combination.
- For example, the **code segment(CS)** register is always used with the **Instruction pointer**(IP) to address the next instruction in a program. This combination is CS:IP.
- Example,if CS=1400H and IP=1200H, the microprocessor fetches its next instruction from memory location 1400**0H**+1200H = 15200H.
- From the above example we can understand that:

✓ The segment address is stored in CS and its values is 1400H.

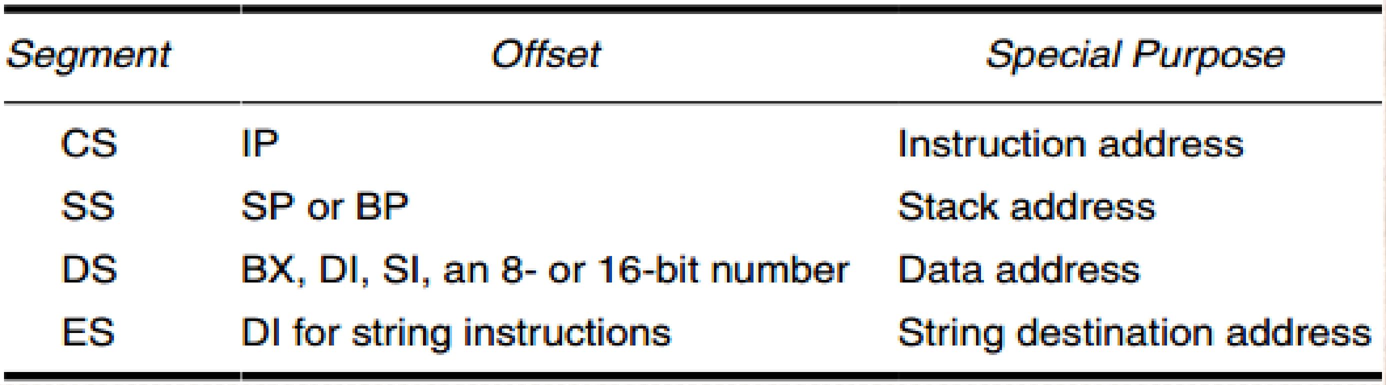
✓ The offset address is stored in IP and its value is 1200H.

✓ OH is appended to the segment address likewise.

Default segment-offset… (Cont...)

- Similarly, Stack data are referenced through the stack segment at the memory location addressed by either stack pointer (SP/ESP) or the pointer (BP/EBP). These combinations are **SS:SP (SS:ESP)**, or **SS:BP (SS:EBP)**.
- For example, if SS = 2000H and BP = 3000H, the microprocessor addresses memory location: 2000**0H** + 3000H = 23000H.
- From the above example we can understand that:
  - The segment address is stored in SS and its values is 2000H.
  - The offset address is stored in BP and its value is 3000H.
  - OH is appended to the segment address likewise.
- Other defaults are shown below.

TABLE 2–3 Default 16-bit segment and offset combinations.



Segment	*Offset*	*Special Purpose*
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, an 8- or 16-bit number	Data address
ES	DI for string instructions	String destination address

Protected mode memory addressing

- Protected mode works on 80286 and above MP’s.
- It allows access to data and programs located above or within the first 1M byte of memory.
- Protected mode is where windows operates. This mode handles multiple task at time.
- Segment address, as discussed with real mode memory addressing, is no longer present in the protected mode.
- In place of the segment address, the segment register contains a selector that selects a descriptor from a descriptor table.
- The descriptor contains information about the segment, e.g., it's base address(segment location), length and access rights.

Protected mode memory addressing(Cont...)

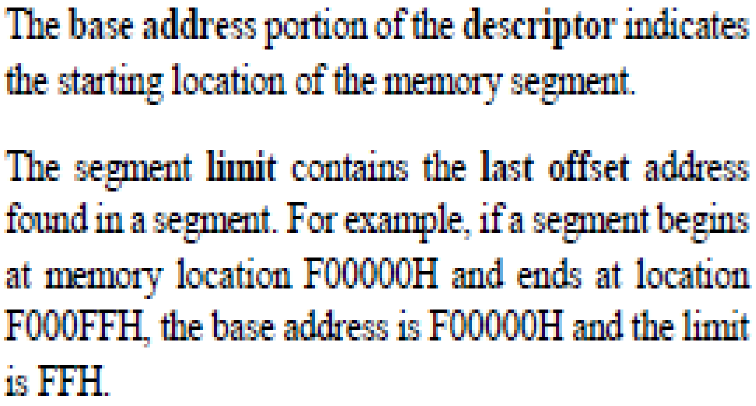
- The difference between real mode and protected modes is:
  - ✓ The way segment register is interpreted by MP to access memory segment.
  - ✓ In 80386 and above, offset address is 32-bit(of course they are backward compatible to 16-bit) but in real mode the OA is 16-bit.
- 32-bit Offset Address allows segment lengths of 4Gbytes

  i.e **2^{32} = 4Gbytes**.
- The 16-bit Offset Address allows segment lengths of 64K bytes.

  i.e **2^{16} = 64Kbytes**.

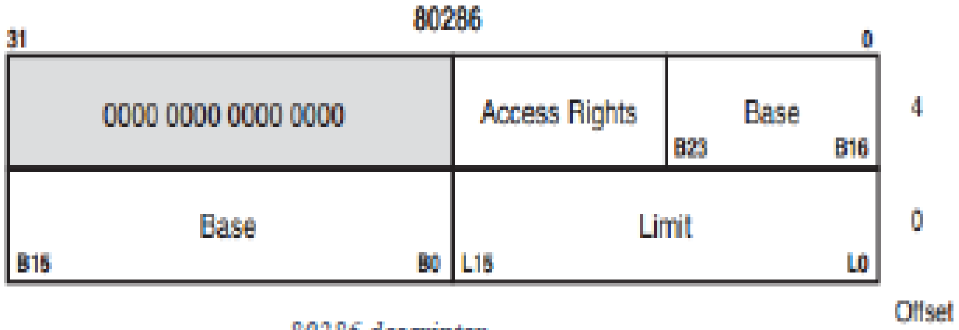
Selectors and descriptors

- The selector, located in the segment register, selects one of **8192 descriptor** from the two descriptor tables.
- There are **two descriptor tables** used with the segment registers:
  - **Global (system) descriptors** and
  - **Local (application) descriptors**.
- The global descriptors contain segment definitions that apply to all programs.
- The local descriptors are usually unique to an application.
- Each descriptor table contains 8192 descriptors, so a total of **16,384 descriptors are available** to an application at any time.
- Because the descriptor describes a memory segment, this allows up to 16,384 memory segments to be described for each application.



The base address portion of the descriptor indicates the starting location of the memory segment.

The segment limit contains the last offset address found in a segment. For example, if a segment begins at memory location F00000H and ends at location F000FFH, the base address is F00000H and the limit is FFH.



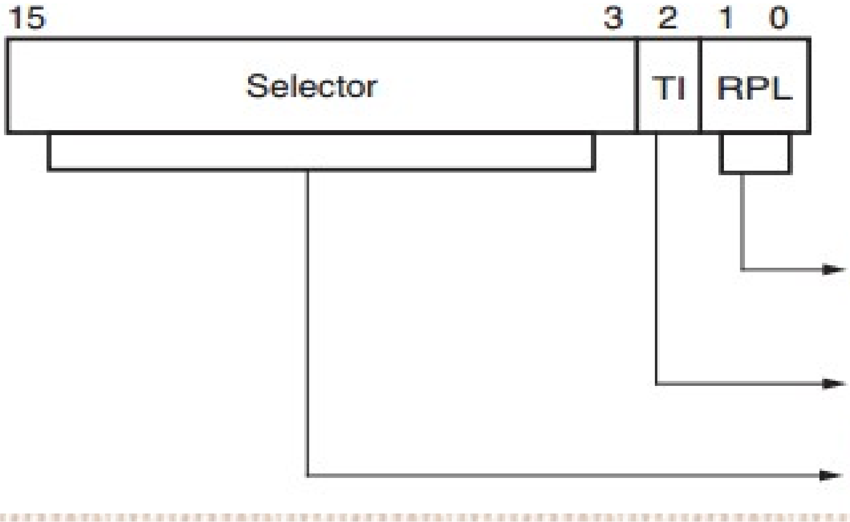
80286

31		0	Offset
0000 0000 0000 0000	Access Rights	Base B23 B16	4
Base B15 B0	Limit L15 L0		0

80286 descriptor

Selectors and descriptors(Cont...)

- The access rights byte controls access to the protected mode segment.
- The segment register contains a 13-bit selector field, a table selector bit, and a requested privilege level field.
- The **13-bit selector** chooses one of the 8192 descriptors from the descriptor table. Also called **descriptor index**.
- The **TI bit** selects descriptor table (TI=0,global or TI=1,local).
- The requested privilege level (**RPL**) requests the access privilege level of a memory segment. Highest privilege level=00, lowest is 11. If the requested privilege level matches or is higher in priority than the privilege level set by the access rights byte, access is **granted**.
- Below is the contents of a segment register of 80286(in protected mode).



2^{13}=8192

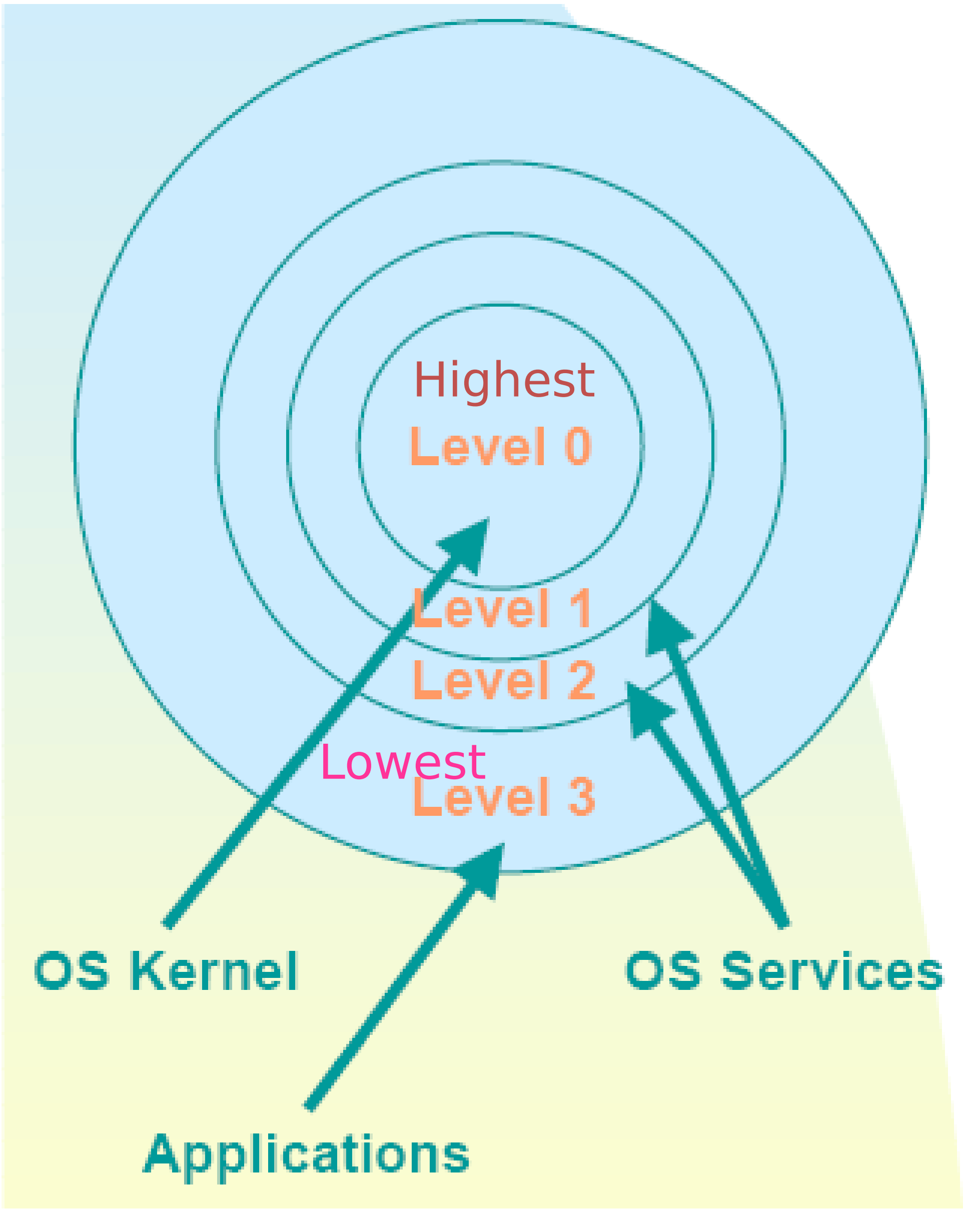
16 bits segment register = 13 bit ds + 1 bit TI + 2-bit RPL

RPL = Requested privilege level where
00 is the highest and 11 is the lowest

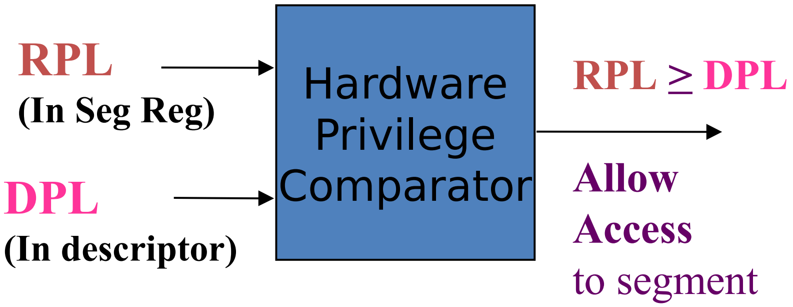
TI = 0 Global descriptor table
TI = 1 Local descriptor table

Selects one descriptor from 8192 descriptors
in either the global or the local descriptor table

Privilege Levels



00: Highest Privilege
01
10
11: Lowest Privilege



RPL
(In Seg Reg)

DPL
(In descriptor)

Hardware Privilege Comparator

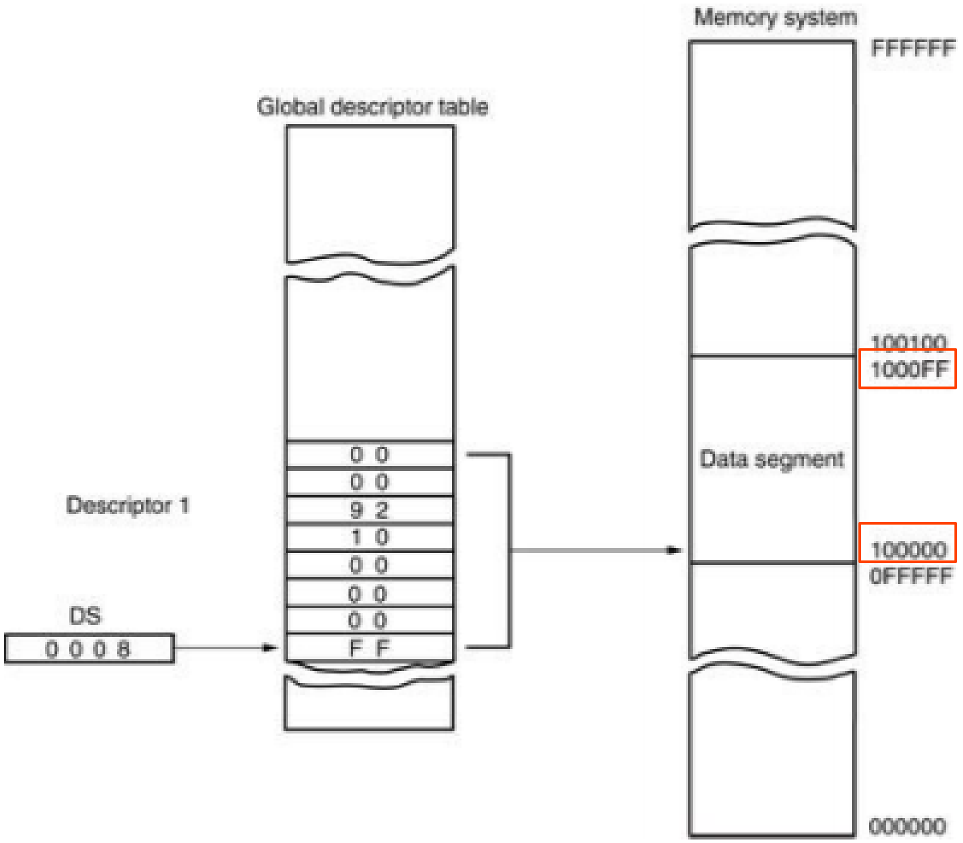
RPL ≥ DPL

Allow Access to segment

➔ Privilege Level is a protection mechanism used for memory management.

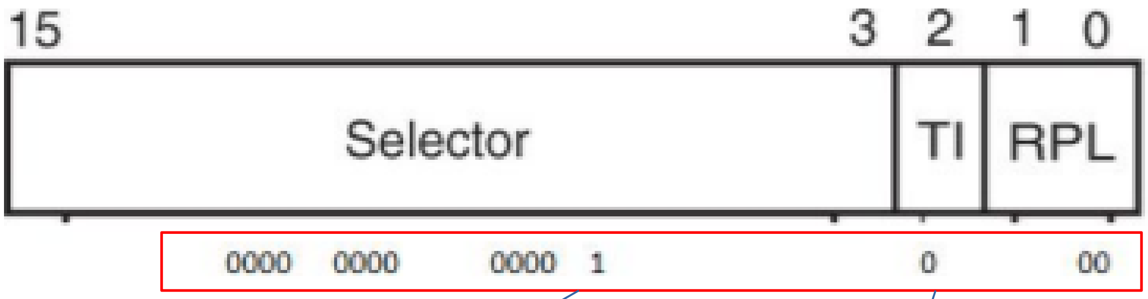
Example Protected mode scheme

- Using the segment register DS, containing a selector, to select a descriptor from the global descriptor table.



- From this DS's **0008H** value we can generate the selector table as follows: **0000 0000 0000 1000**, then….

Example Protected mode scheme(Cont...)



From this we can understood that the selector accesses the **descriptor number 1** from the **global descriptor** table using a requested privilege level of 00, highest.

- The **entry in the global descriptor** table selects a segment in the memory system.
- Assume that descriptor number 1 contains a descriptor that defines the **base address** as 00100000H with segment **limit** of 000FFH(as shown in the prev. slide).
- This means that a value of 0008H loaded into DS causes the MP to use memory locations 00100000H – 001000FFH.

Example Protected mode scheme(Cont...)

- But what will be the physical address if the offset address is 00000FC0?

Solution: since as given above the base address is 00100000H,

The Physical address is: 00100000H + 00000FC0H

= **00100FC0H**

- Show the segment start and end if the base address is 10000000H and the limit is 001FFH.

Solution:

Base Address = Start Address

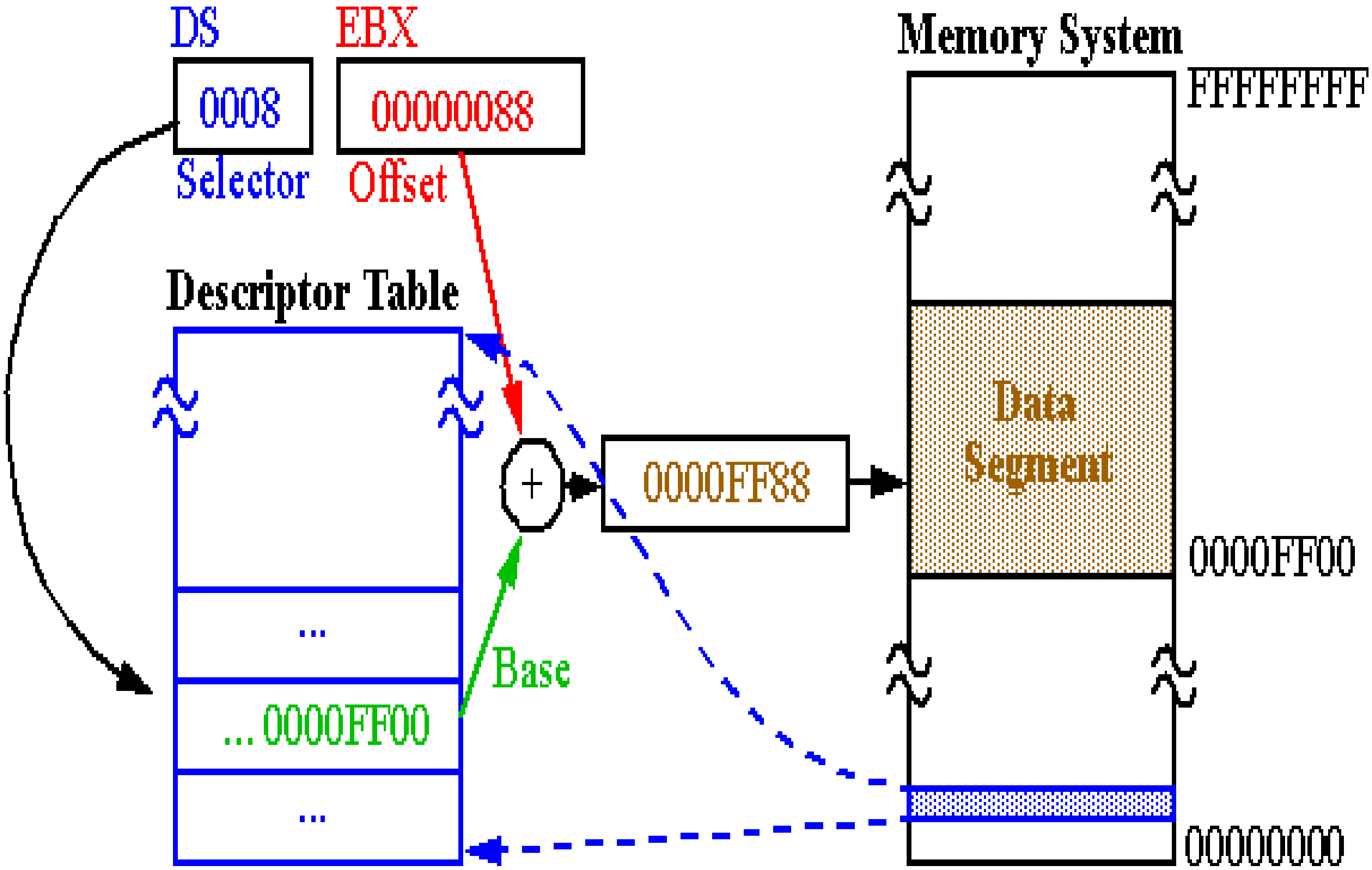
= 10000000H

End Address = Base + Limit

= 10000000H + 001FFH

= **100001FFH**

Protected mode scheme(Cont...)



Limitations of the real mode scheme

- Segment size is fixed at and limited to 64 KB.
- Real mode provides no support for multitasking.
- Principle is difficult to apply with 80286 and above with segment registers remaining 16-bits(in 80286)!

- 80286 use 16 - bit segment registers and 24 Address bus. That is, it covers up to 2^24= 16Mb memory), but above 80286 MP's use 32 bit addresses and above(also the address bus is 24 bit and above).

- No protection mechanisms: Programs can overwrite operating system code segments and corrupt them!

➔ Use memory segmentation in the protected mode.

Why Protected Mode?

- This protected mode addressing is better than the real mode addressing due to the following main reasons:

- ✓ Flexible definition of segment size.
- ✓ Protection mechanisms that prevent programs from corrupting the code and data of each other and of the operating system.

Memory Management

Paging: 80386 and above

- Paging is one of the memory management techniques used for virtual memory multitasking system.
- Thus paging mechanism provides an effective technique to manage the physical memory for multitasking systems.
- The segmentation scheme may divide the physical memory into a variable size segments but the paging divides the memory into a fixed size pages.
- The **memory paging** mechanism located within the **80386 and above** allows any physical memory location to be assigned to any linear address.

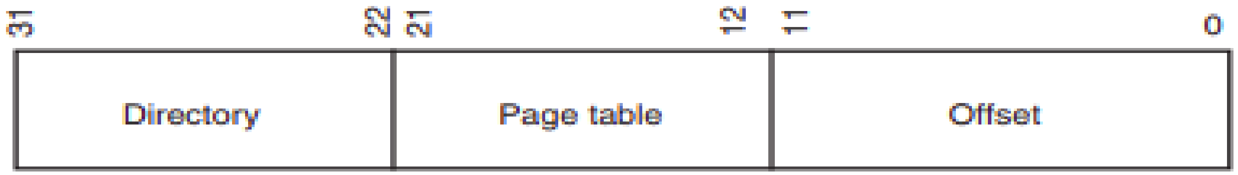
✓ That is, they translates a **logical(linear)** address generated by the program **to a physical(real)** address that accesses a storage location in a memory.

Paging: 80386 and above(Cont….)

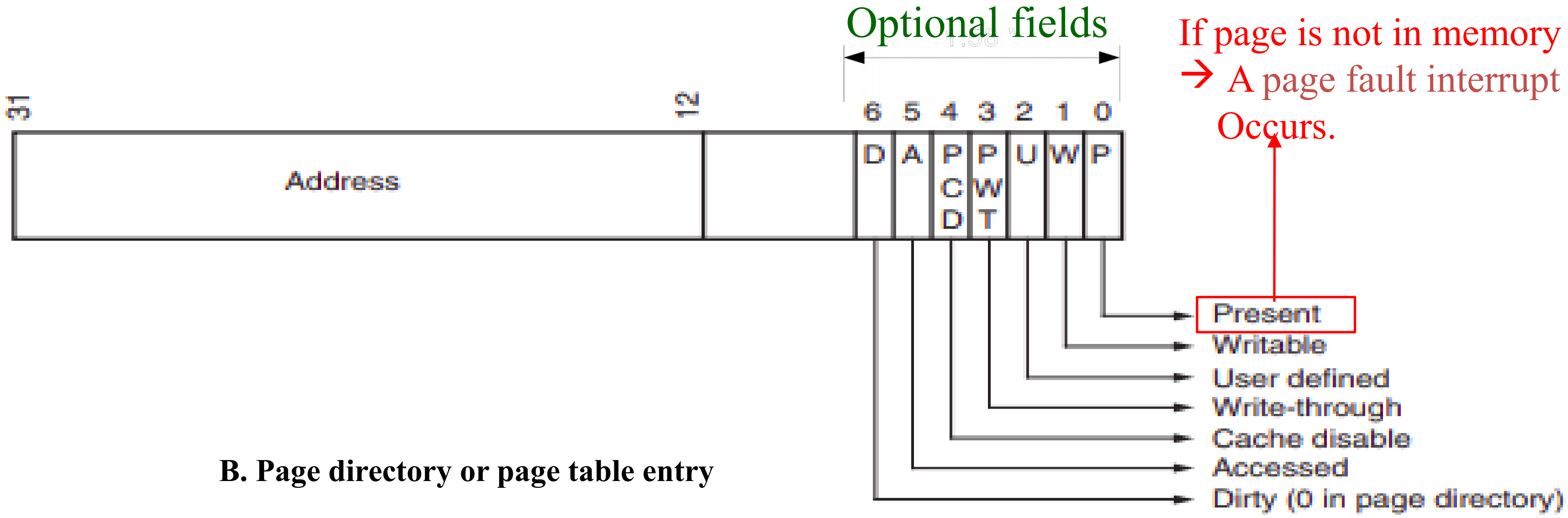
- The **linear address** is defined as the address generated by a program.
- The **physical address** is the actual memory location accessed by a program.
- Translation is done from linear to physical pages.
- Physical pages may or may not reside in physical memory.
- If page is not in memory page fault occurs.

Paging: 80386 and above (Cont...)

- The linear address, as generated by software, is broken into three sections that are used to access the **page directory entry**, **page table entry**, and **memory page offset address**.



A. The format for the linear address



B. Page directory or page table entry

If page is not in memory → A page fault interrupt Occurs.

Figure above shows the linear address and its makeup for paging.

Paging: 80386 and above (Cont...)

- 2^{32} = 4 GB address space.
- Paging uses several smaller page tables (up to 1K tables), each providing translation for 1K pages.
- Start address for each page table used is kept in one directory table.
- As indicated above, the 32-bit linear address space(as generated by the processor) is divided into three parts:

✓ **Directory**: 10 bits, determines which page table in the page table directory.

- It contains 1024 entries and each directory entry is 4byte in size and thus it is of 4Kbytes in size.
- It's the upper 10 bits of the linear address, used as an index to the corresponding page directory entry. The page directory entries **point to page tables**.

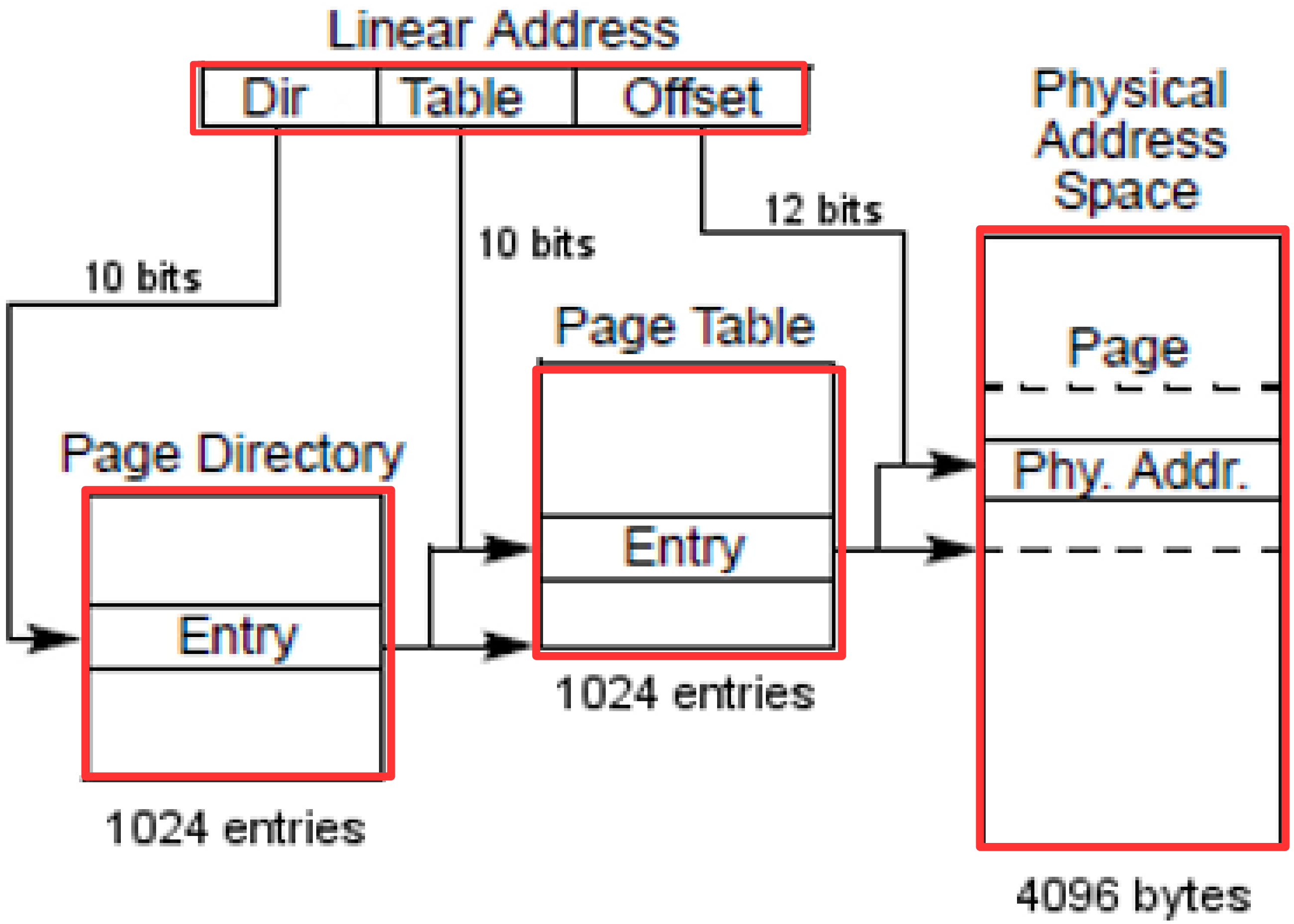
Paging: 80386 and above (Cont...)

Note: Each entry is 4 bytes in size which is due to address space representation in 32bit architecture(i.e 32 bits = 4bytes).

- ✓ **Page table**: 10 bits, determines which page in that page table.
  - Again it contain 1024 entries each of which is 4byte in size and forming the size of each page table to be 4Kbytes in size.
  - The page table entries contain the starting address of the page and the statistical information about the page.
- ✓ **Memory offset**: 12 bits, determines which byte in that page from 4G physical bytes((2^{10}*2^{10}*2^{12})bits=4G bytes).

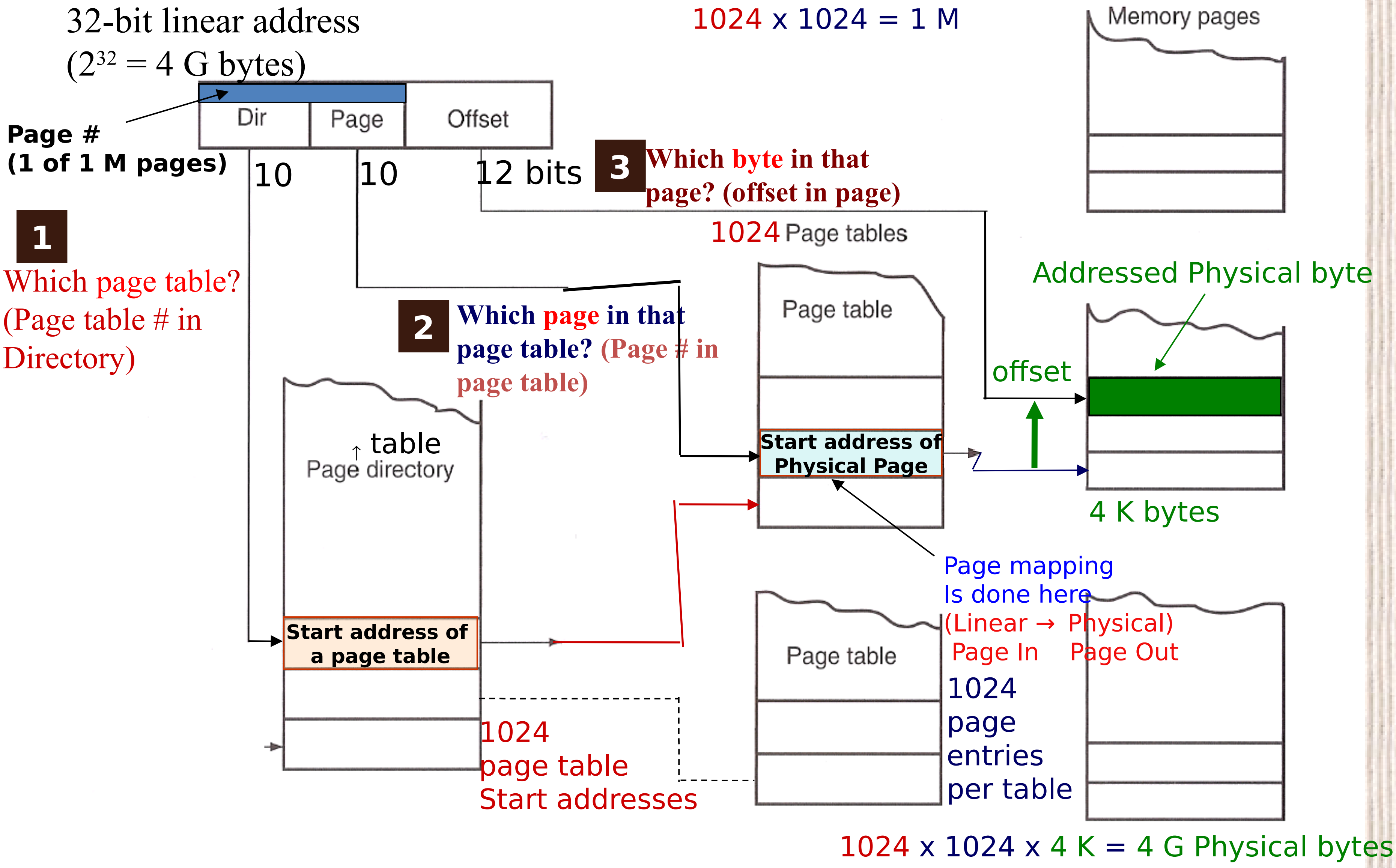
- Thus, the upper 20 bit address is combined with the lower 12 bit of the linear address.
- The address bits A12- A21 are used to select the 1024 page table entries.
- The address bits A22-A31 are used to select the 1024 page table.

Paging: 80386 and above (Cont...)



- The **page table entries** contain the starting address of the page and the statistical information about the page.

Paging: 80386 and above (Cont...)



32-bit linear address

(2^{32} = 4 G bytes)

1024 x 1024 = 1 M

Memory pages

Page #

(1 of 1 M pages)

Dir | Page | Offset

10 10 12 bits

3 Which byte in that page? (offset in page)

1 Which page table? (Page table # in Directory)

1024 Page tables

Addressed Physical byte

2 Which page in that page table? (Page # in page table)

Page table

offset

table

Page directory

Start address of Physical Page

4 K bytes

Page mapping Is done here

(Linear → Physical)

Page In Page Out

Start address of a page table

Page table

1024 page entries per table

1024 page table Start addresses

1024 x 1024 x 4 K = 4 G Physical bytes

Examples….

- For linear address 00000000H – 00000FFFH, the first **page directory** is accessed by the left most 10 bits (bit positions 22–31). Each page directory entry represents a 4M section of the memory system.
- The contents of the page directory **select a page table** that is indexed by the next 10 bits of the linear address (bit positions 12–21). This means that address 00000000H – 00000FFFH selects **page directory entry of 0** and **page table entry of 0**.
- The **offset part** of the linear address (bit positions 0 – 11) next selects a byte in the 4Kbyte memory page.
- Now let us assume the page table entry 0 contains address (starting address) of **00100000H**, then what would be the physical address range?

Examples….

Note: on the above given example(linear address range 00000000 – 000000FFF) both addresses have the same page directory and page table entries, but they differ in their offset address. I.e $\underbrace{00000}000 – \underbrace{00000}\text{FFF}$

$\longrightarrow$ are the same

- **Solution:** Now, if the page table entry 0 contains 00100000H then the physical address will be found as follows;

=> 00100000H + values of offset address(12bit in size). That is:

For the first address(00000000H) the physical address would be:

= 00100000H + its Offset Address

= 00100000H + 000

= 00100000H

Examples….

For the last address(00000FFFH) the physical address would be:

= 00100000H + its Offset Address

= 00100000H + FFF

= 00100FFFH

- The physical address range is therefore;

  <u>00100000H – 00100FFFH.</u>

- This means that when the program accesses a location between 00000000H and 00000FFFH, the microprocessor physically addresses location 00100000H—00100FFFH.