Chapter 2

The Microprocessor and its Architecture

The Intel 8086, 80X86, and Pentium Family

Contents

- Internal architecture of the Microprocessor:
  - The programmer's model, i.e. the registers model
  - The processor (organization) model
- Memory addressing with segmentation
  - In the real mode
  - In the protected mode
- Memory addressing with paging

Objectives for this Chapter

- Describe the function and purpose of program-visible registers
- Describe the Flags register and the purpose of flag bits
- Describe how memory is accessed using segmentation in both the real mode and the protected mode
- Describe the program-invisible registers
- Describe the structures and operation of the memory paging mechanism
- Describe the organizational processor model
- Briefly review the evolution of the 80X86 architecture

Architecture

- MP is a programmable digital device.
- Designed with registers, flip-flop's (FF) & timing circuits.
- MP has a set of instructions, designed internally, to manipulate data & communicate with peripherals.

- This process of data manipulation & communication is determined by logic design of MP, called the Architecture.
- MP can be programmed to perform fun'c on given data by selecting necessary instructions from its set.

- These instructions are given to MP by writing into its memory.
- Writing (or entering) instructions & data is done through an input device (keyboard).
- MP reads or translates one instruction at a time, matches it with its instruction set, & performs data manipulation indicated by instruction.

- Result stored in memory or sent to

such output devices as LED or a CRT terminal.

- MP can respond to external signals.
- It can be interpreted, reset or asked to Wait to synchronize with slower peripherals.

- All fun'c performed by the MP can be classified in 03 general categories:

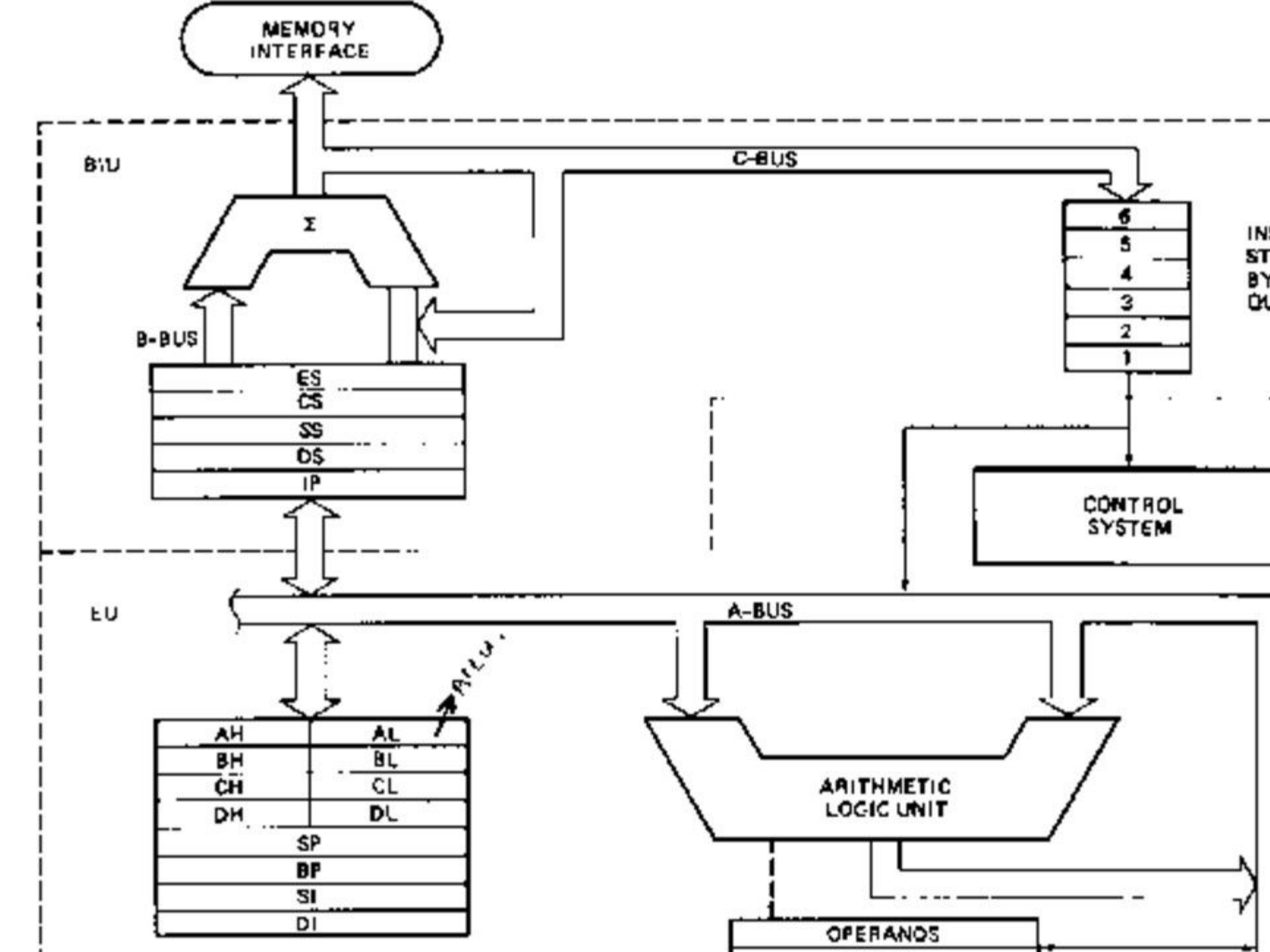
1. MP-initiated op'n
2. Internal op'n
3. Peripheral (externally initiated op'n).

8086 Internal Architecture

- 8086 MP is internally divided into 02 separate functional units:

1. Bus Interface Unit (BIU)
2. Execution Unit (EU).

-Dividing the work bet'n these units speeds up processing.


MEMORY
INTERFACE
BIU
C-BUS
Σ
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
CONTROL
SYSTEM
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ARITHMETIC
LOGIC UNIT
OPERANDS

1.BIU (Bus Interface Unit)

FUNCTIONS:

- BIU sends out address
- Fetches instruction from memory
- Read data from memory & ports.
- BIU interfaces 8086 to outside world.
- BIU handle all transfer of data & addresses on buses for EU.

2. EU (Execution Unit)

FUNCTIONS:

- EU executes instruction that have already been fetched by the BIU.
- EU tells BIU where to fetch instructions or data from, decodes instruction, & executes instructions.
- BIU & EU fun'c independently.

EU (decode & executes instructions).

1. Control circuitry, Instruction decoder & ALU:

- Control circuitry directs internal op'n.
- Decoder translates instructions fetched from memory into a series of actions which EU carries out.

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 (1978)	16	20	1M
8088	8	20	1M
80186	16	20	1M
80188 (Microcontrollers)	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 (2000)	64	32	4G
Pentium Pro–Pentium 4 (if extended addressing is enabled)	64	36	64G
	≈ Increase	Increase	

The programming model

- **Program visible:**

Programming model of 8086 through P4 is considered to be program visible because it registers are used during application programming & specified by instructions.

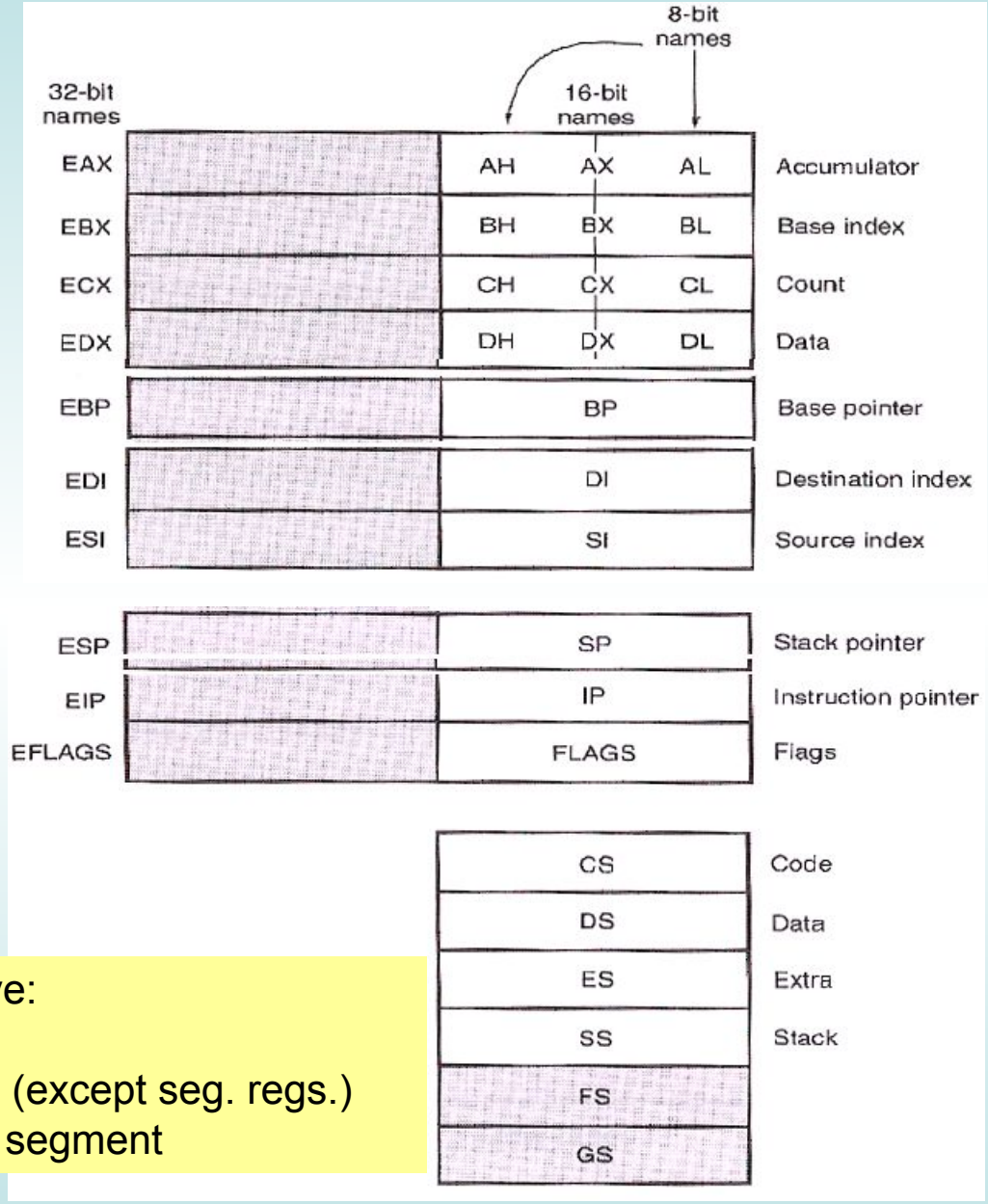
- **Program invisible:** not directly addressed by s/w.

Programming Model

General Purpose Registers

Special Purpose Registers

Segment Registers



80386 and above:

-32-bit registers (except seg. regs.)
-Two additional segment registers:F,G

General Purpose Registers:

- The EU has 08 general purpose registers.

  AH, AL, BH, BL, CH, CL, DH, DL.

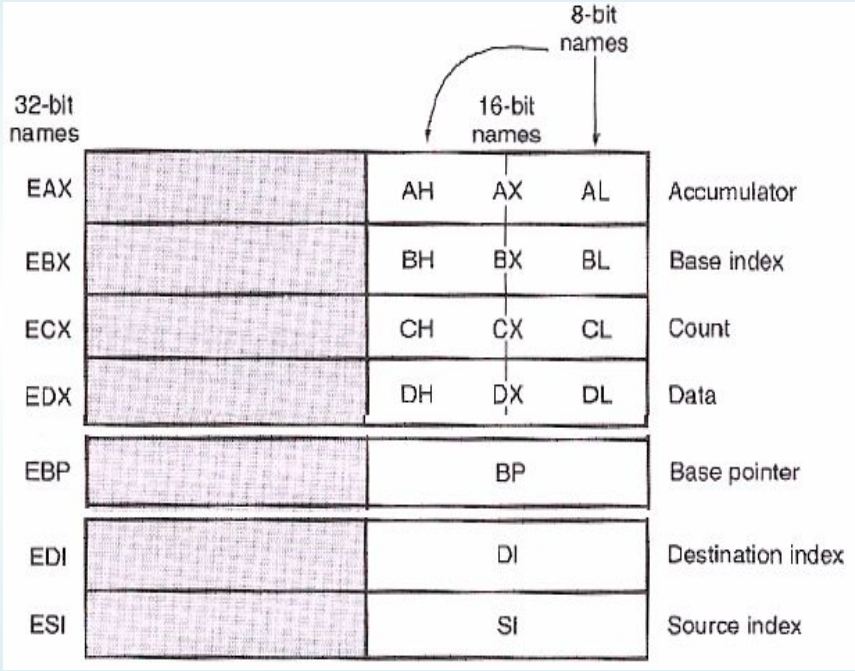
- These registers can be used individually for temporary storage of 8-bit data.

- Certain pairs of these general purpose registers can be used together to store 16-bit data words.
- The acceptable register pairs are AH & AL, BH & BL, CH& CL, DH & DL.

- AH-AL pair refereed to as AX register.
- BH-BL ……BX register.
- CH-CL……CX register.
- DH-DL……DX register.
- Adv. Of using internal register for the temporary storage of data is that, since data is already in EU, it can be accessed much more quickly than it could be accessed in external memory.

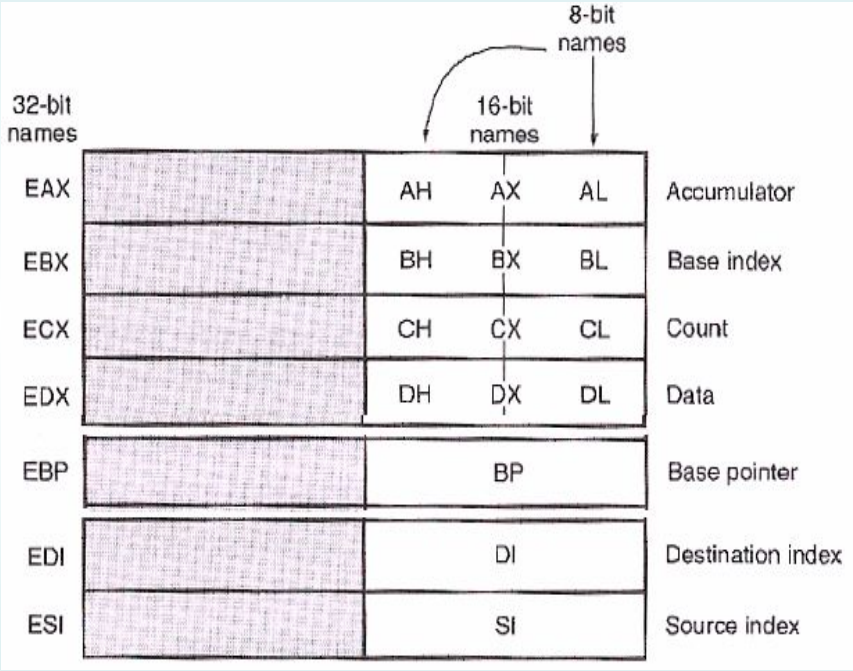
General-Purpose Registers

- The top portion of the programming model contains the general purpose registers: EAX, EBX, ECX, EDX, EBP, ESI, and EDI
- Can carry both Data & Address offsets
- Although general in nature, each has a special purpose and name:
- EAX – Accumulator
  - Used also as AX (16 bit), AH (8 bit), and AL (8 bit)
- EBX – Base Index often used to address memory (BX, BH, and BL)



General-Purpose Registers (continued)

- ECX – count, for shifts, rotates(CL),repeated string instruction (CX) and loops (CX/ECX)
- EDX – data, used with multiply and divide (DX, DH, and DL)
- EBP – base pointer used to address stack data (BP)
- ESI – source index (SI) for memory locations, e.g. with string instructions
- EDI – destination index (DI) for memory locations



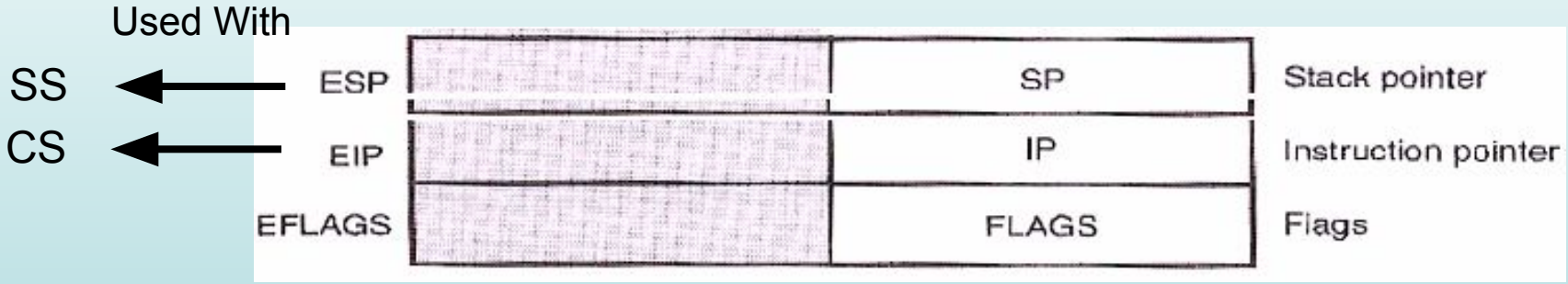
Two Index Register:

- **SI (Source Index)** Register
- **DI (Destination Index**) Register
- Used in indexed addressing.
- Instruction that process data string use SI & DI together with DS & ES, in order to distinguish bet'n source & destination address.

Special-Purpose Registers

- ESP, EIP, and EFLAGS

  Each has a specific task
  - ESP – Stack pointer: Offset to the top of the stack in the stack segment. Used with procedure calls (SP)
  - EIP – Instruction Pointer: Offset to the next instruction in a program in the code segment (IP)
  - EFLAGS – indicates latest conditions (state) of the microprocessor (FLAGS)



Registers That Form a Stack

- There are special features to handle computers where some of the “registers” form a stack. Stack registers are normally written by pushing onto the stack, and are numbered relative to the top of the stack.

- A **stack pointer** is a small register that stores the **address of the last program request in a stack**.
- A stack is a specialized buffer which stores data from the top down. As new requests come in, they "push down" the older ones. The most recently entered request always resides at the top of the stack, and the program always takes requests from the top.

- A stack (also called a pushdown stack) operates in a last-in/first-out sense.
- When a new data item is entered or "pushed" onto the top of a stack, the stack pointer increments to the next physical memory address, and the new item is copied to that address.
- When a data item is "pulled" or "popped" from the top of a stack, the item is copied from the address of the stack pointer, and the stack pointer decrements to the next available item at the top of the stack.

Two position Registers

- SP (Stack Pointer)
- BP (Base Pointer)
- Used to access data in the stack segment (SS).
- SP is used as an offset from the current SS **during execution of instruction that involve stack segment in external memory**.

- SP contents are automatically updated

(increment or decrement) due to execution of POP or PUSH instruction.

- **BP** contains an offset address in the current SS. This **offset is used by instructions utilizing based addressing mode.**

The Queue:

- -The BIU instruction queue is a first-in-first-out (FIFO) group of registers in which up to 06 bytes of instruction code are perfected from memory ahead of time

- • This is done in order to speed up prog. Execution by overlapping instruction fetch with execution.
- • When EU is ready for its next instruction, it simply reads instruction byte for the instruction from queue in the BIU.

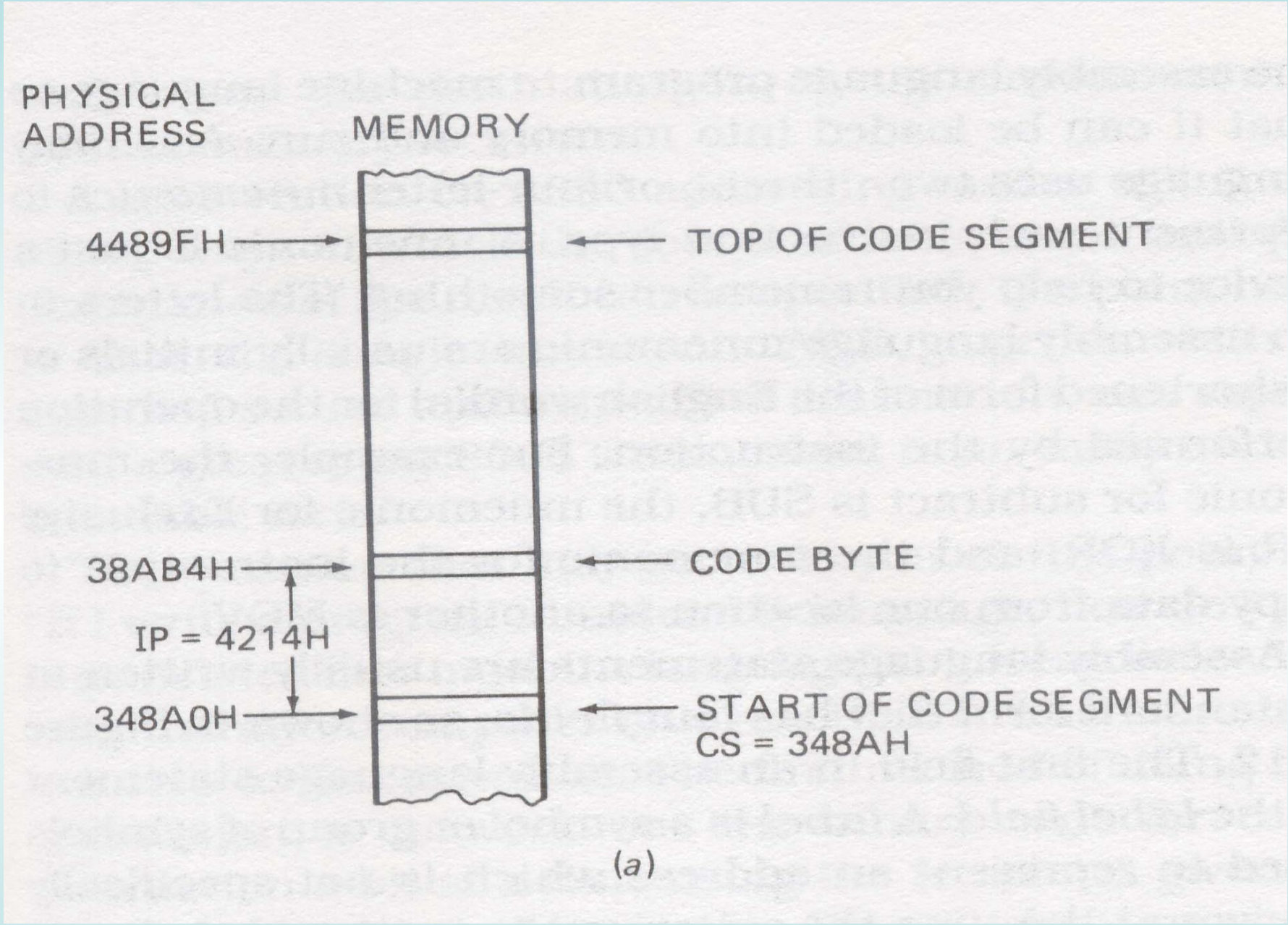
- • This is much faster than sending out an address to system memory & waiting for memory to send back to the next instruction byte/bytes.
- • Except in the cases of JMP & CALL instructions, where queue must be dumped & then reloaded starting from a new address.

- • This prefetch-&-queue scheme greatly speeds up processing.
- • Fetching next instruction while current instruction executes is called ***Pipelining.***

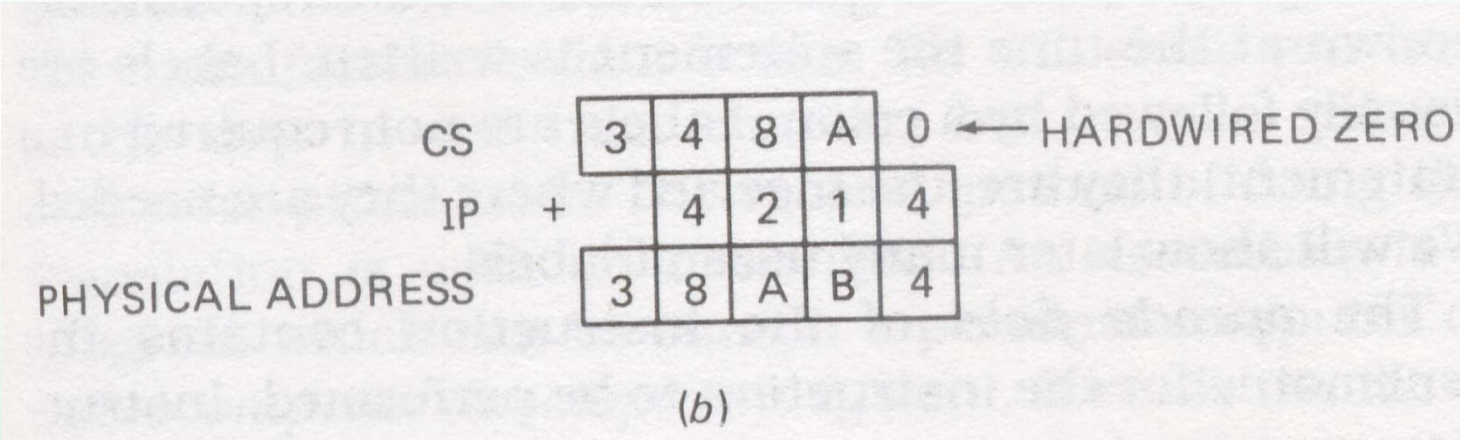
Instruction Pointer (IP):

- • CS register holds upper 16 bits of starting address of the segment from which BIU is currently fetching instruction code bytes.
- • IP Register holds 16 bits address/offset of the next code byte within this code segment.

- The value contained in IP is refereed to as an ***offset because*** this value must be offset from (added to) segment base register in CS to produce required 20-bit physical address sent out by BIU.



- • The CS register contains points to the base/start of current code segment.
- • IP contains the distance/offset from this base address to the next instruction byte to be fetched.



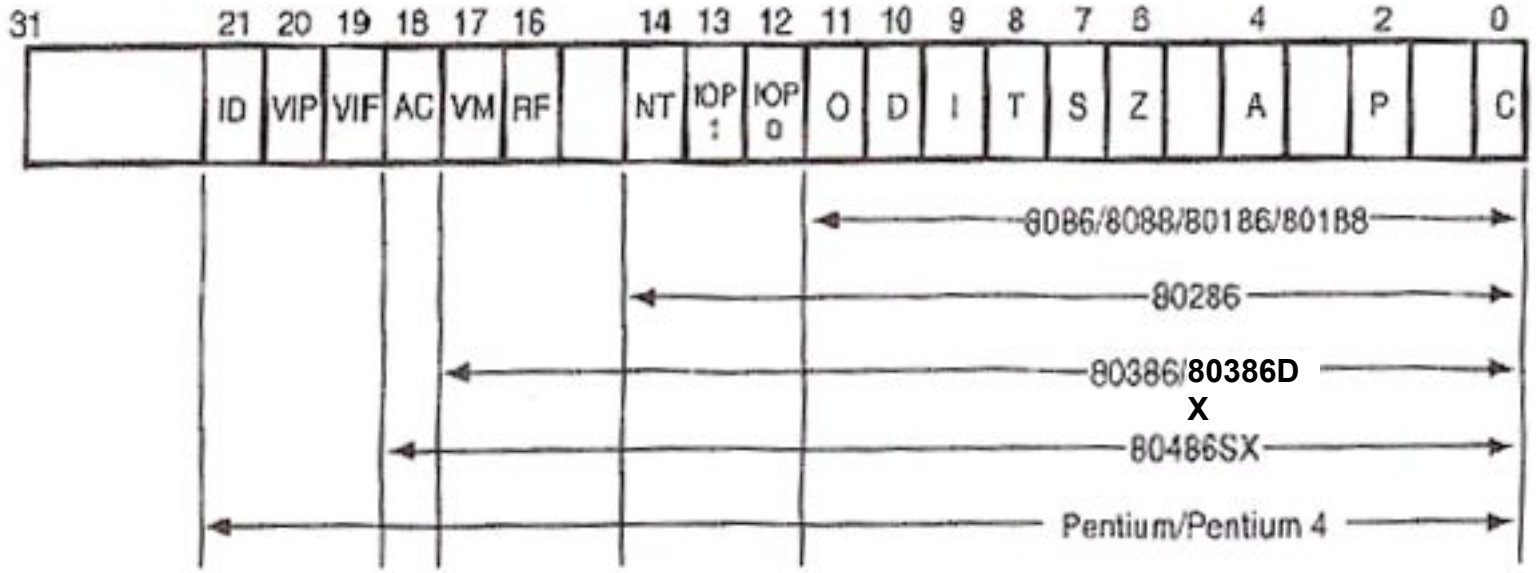
(*b*)

- Fig (b) shows, how 16-bit offset in IP added to 16-bit segment base address in CS to produce 20-bit physical address.
- • Two 16-bit no. are not added directly in line, because CS register contains only upper 16 bits of the base address for code segment.

- • BIU automatically inserts zeros for lowest 04 bits of segment base address.
- • If CS register, contains 348AH, starting address for code segment 348A0H.
- • When BIU adds offset of 4214H in IP to this segment base address, result is a 20-bit physical address of 38AB4H.

- • Alternative way of representing a 20-bit physical address is:
- **Segment base : Offset form**
- **CS : IP**
- **348A : 4214**

EFLAGS



The Flags

Determined by last operation

Basic Flag Bits (8086 etc.): Output, Input bits

- C – Carry/borrow from last operation
- P – the parity flag (little used today)
- A – auxiliary flag Half-carry between bits 3 and 4, used with BCD arithmetic
- Z – zero
- S – sign
- O – Overflow
- D – direction - Determines auto increment/decrement direction for SI and DI registers with string instructions
- I – interrupt - Enables (using STI) or disables (using CLI) the processing of hardware interrupts arriving at the INTR input pin of the processor
- T – Trap - Turns trapping interrupt (for program debugging) on/off

Set/Reset explicitly by the programmer

Some flag bits can be both, e.g. the C flag

FLAG REGISTERS:

- A flag register indicates some condition produced by the execution of an instruction or controls certain op'n of EU.
- • Flag register in the EU holds the status flags typically after an ALU op'n.
- • 16-bit flag resister contains 09 active flags.

- EU has a 16-bit ALU which can arithmetic & logical op'n(addition, subtraction, OR, XOR, increment, Decrement, complement or shift binary numbers).

- 16 bits flag resister contains 09 active flags.
- • 06 flags are used to indicate some condition produced by an instruction.
- • 06 conditional flags are:

1. Carry flag (CF) 2. Parity flag (PF)

3.Auxiliary carry flag (AC) 4.zero flag(ZF) 5.Sign flag (SF) 6.Overflow flag (OF)

1. Carry flag (CF):

• CF is set, if an arithmetic op'n results in a carry. Otherwise it is reset.

2. Parity flag (PF):

• PF is set if the result has even parity, PF is zero for odd parity.

• If the result has an even number of 1, flag is set, for odd no. of 1, flag is reset.

3. Auxiliary carry flag (AF):

• AF is used by BCD arithmetic instructions.

• In an arithmetic op'n, when a carry is generated by digit 3 &passed to digit 4, the AF flag is set.

4. Zero flag (ZF):

•• ZF is set(1) if result is zero, ZF is zero(0) for nonzero result.

5. Sign flag (SF):

• SF is set if most significant bit of the result is 1(negative).

• Cleared to zero, for non-negative result.

6. Overflow flag (OF):

• OF set, if there is an arithmetic overflow, that is, if the size of the result exceeds the capacity of destination location.

- 03 remaining flag used **to control certain op'n of processor.**
- 06 **conditional flag** are set/reset by EU on the basis of the results of some arithmetic or logical op'n.
- Control flags are set/reset with specific instruction put in your prog.

7.Trap (Trace) flag (TF):

- Setting TF to one places the 8086 in the single-step mode.
- in this mode, 8086 generates an internal interrupt after execution of each instruction.
- User can write a service routine at the interrupt address vector to display the desired registers & memory location.

7. TF (Cont…)

• User can thus debug a program.

8. Interrupt flag (IF):

• Used to allow or prohibit the interruption of a program.

•-if I=1, INTR pin is enable.

•-if I=0, INTR “ “ disable.

- ***9. Direction flag (DF):***

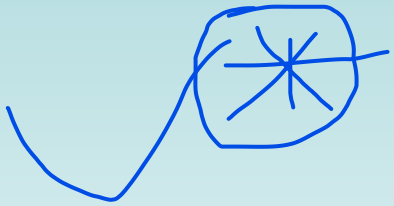
-DF select either increment or decrement mode during string instructions.

- If D =1, register automatically decrement.
- If D = 0, register “ ” incremented.

Newer Flag Bits

- IOPL – 2-bit I/O privilege level in protected mode
- NT – nested task
- RF – resume flag (used with debugging)
- VM – virtual mode: multiple DOS programs each with a 1 MB memory partition in Windows
- AC – alignment check: detects addressing memory on wrong boundary for words/double words
- VIF – virtual interrupt flag
- VIP – virtual interrupt pending
- ID = CPUID instruction is supported

  The instruction gives info on CPU version and manufacturer



Segment Registers

Each register points to the start of a segment in memory

- The segment registers are:
  - CS (code),
  - DS (data),
  - ES (extra data. used as destination for some string instructions),
  - SS (stack),
  - FS, and GS: Additional segment registers on 80386 and above
- Segment registers define the start of a section (segment) of memory for a program.
- 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).
- All code (programs) reside in a code segment.

Real Mode Memory Addressing

- Used by the DOS operating system
- The only mode available on the 8086-8088:
  20 bit address bus 1 MB, 16 bit data bus, 16 bit registers
- Real mode memory is the first 1M (2^{20}) bytes of the memory system (real, conventional, DOS memory) in later processors
- Real mode 20-bit addresses are obtained by combining a segment number (in a segment register) and an offset address (in another processor register)
- The segment register address (16-bits) is appended with a 0H or 0000_2 (or multiplied by 10H or 16d) to form a 20-bit start of segment address
- Then the effective memory address (EA) =
  this 20-bit segment start address + the 16-bit offset address in another processor register
- For the 8086, segment length is fixed @ 2^{16} = 64K bytes (determined by the size of the offset registers)

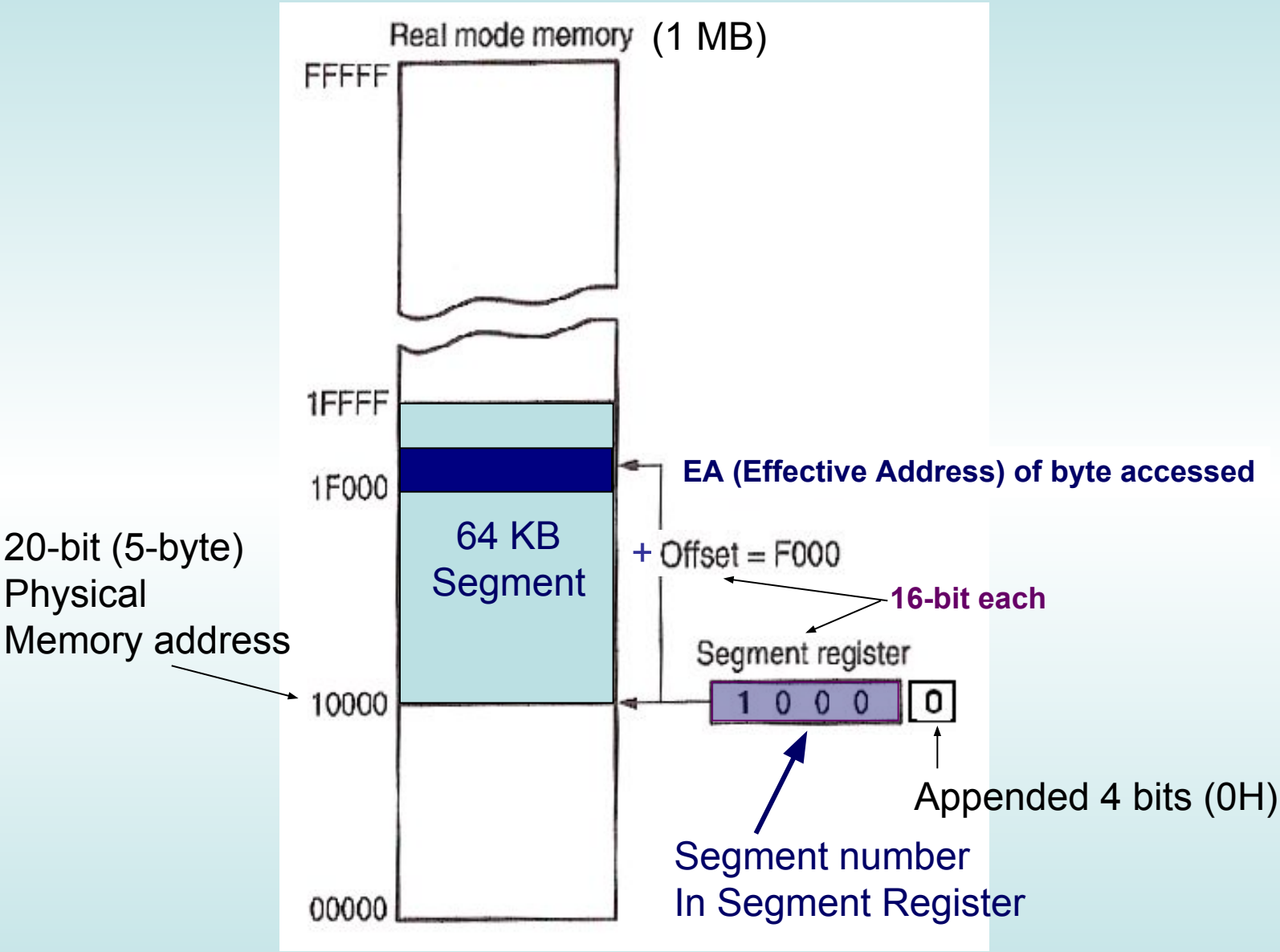
Segments & Offsets

.

- • A combination of ***segment address*** & an ***offset address*** access a memory location in real mode.
  • All real mode memory address must consists of a segment address plus an offset address

- • **Segments address** located within one of segment register, defines the beginning address of any 64KB memory segment.
- • **Offset address** selects any location within 64KB memory segment.
- • Segment in real mode always have a length of 64KB.

- • **See fig.(2.3), P-57, Brey.**
- • A memory segment begins at location 10000H & ends at location 1FFFFH-64KB in length.
- • An offset address, sometimes called a **displacement,** of F000H selects location 1F000H in memory system.



- • Offset or displacement is the distance above the start of the segment.
- • Segment register contains a 1000H, its address a starting segment at location 10000H.
- • In real mode, each segment register is internal appended with 0H on its rightmost end.

- • This form a 20 bit memory address, allowing it to access the start of a segment.
- • MP must generate a 20-bit memory address to access a location within first 1M of memory.

- •For example:
- • When a segment register contains a 1200H, it address a 64KB memory segment beginning at location 12000H.
- Likewise, if a segment register contains a 1201H, it address a memory segment beginning at location 12010H.

- • Because of internally appended 0H, real mode segments can begin only at a 16- byte boundary in memory system. This16-byte boundary often called a Paragraph.
- • Because a real mode segment of memory is 64K in length, once the beginning address is known, ending address is found by adding FFFFH.

- • Example:
- • If a segment register contains 3000H, the 1st address of the segment is 30000H.
- • Last address of segment, 30000H + FFFFH = 3FFFFH.

Example of segment addresses

-

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

- • The offset address, which is a part of address, is added to the start of the segment to address a memory location within the memory segment.
- • Example:
- • If segment address is 1000H & offset address 2000H, MP Address memory location 12000H.

- **“The offset address is always added to the starting address of the segment to locate the data”.**
- • the segment & offset address is sometimes written as 1000:2000 for a segment address of 1000H with an offset of 2000H.

Default Segment & Offset Registers

- • MP has a set of rules that apply to segments whenever memory is addressed.
- • These rules, which apply in real mode, define the segment register & offset register combination.
- • CS register is always used with IP to address next instruction in a program.

- • This combination is CS:IP.
- • CS register defines the start of the code segment & IP locates the next instruction within the code segment.
- • This combination (CS:IP) locates next instruction executed by MP.

Effective Address Calculations

- EA = segment register (SR) x 10H + offset

  (a) SR: 1000H

  10000 + 0023 = 10023

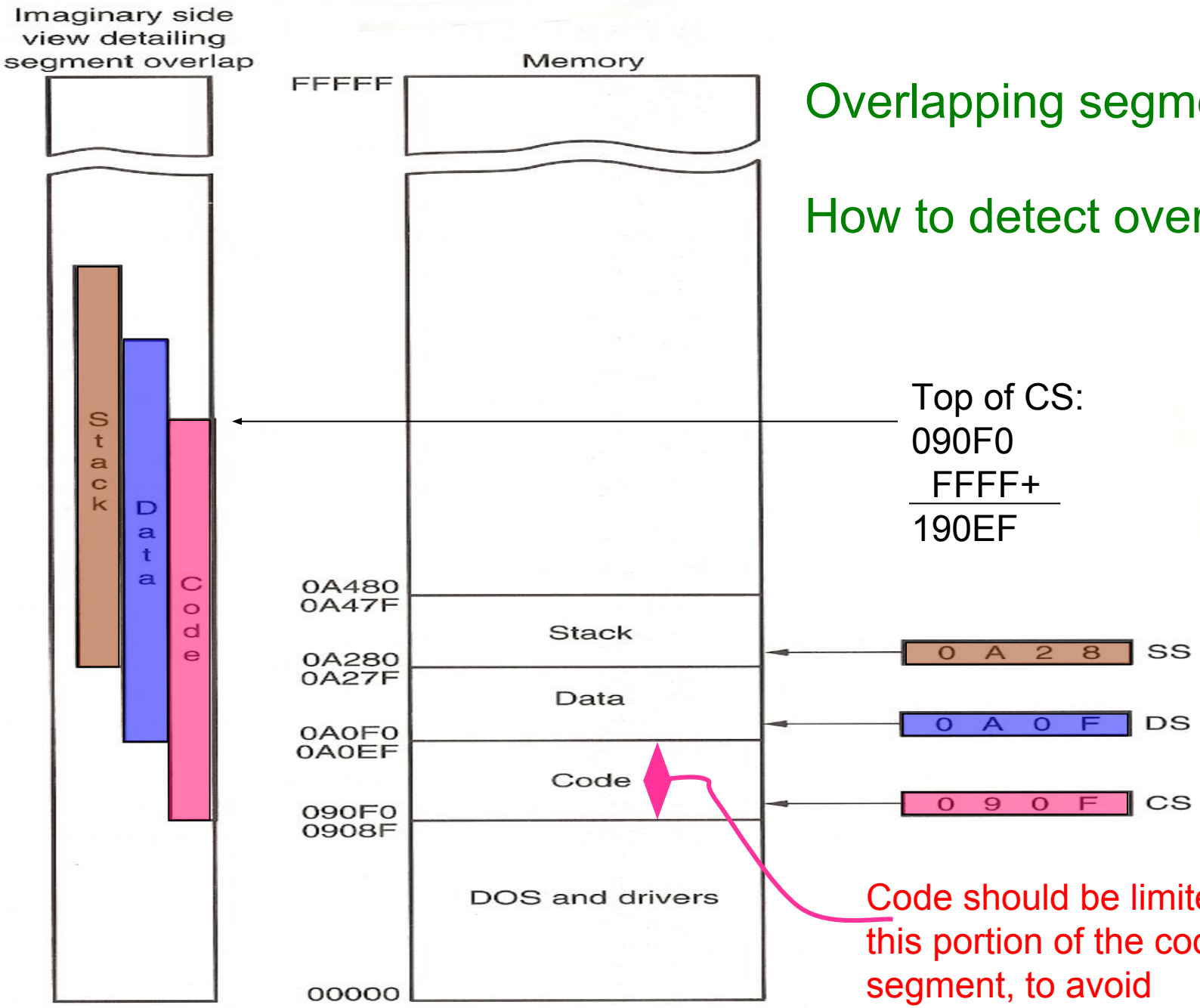
  (b) SR: AAF0H

  AAF00 + 0134 = AB034

  (c) SR: 1200H

  12000 + FFF0 = 21FF0

Q: Is 3FC81 a valid start address of a segment?

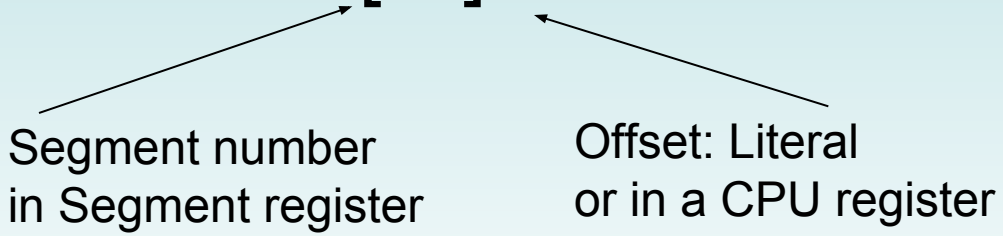


Overlapping segments

How to detect overlap?

Defaults

Convention Example: EA = CS:[IP]



- Default segment numbers in:
  - CS for program (code)
  - SS for stack
  - DS for data
  - ES for string (destination) data
- Default offset addresses that go with them:

Segment	Offset (16-bit) 8080, 8086, 80286	Offset (32-bit) 80386 and above	Purpose
CS	IP	EIP	Program
SS	SP, BP	ESP, EBP	Stack
DS	BX, DI, SI, 8-bit or 16-bit #	EBX, EDI, ESI, EAX ECX, EDX, 8-bit or 32-bit #	Data
ES	DI, with string instructions	EDI, with string instructions	String destination

Segmentation: Pros and Cons

Advantages:

- Allows easy and efficient relocation of code and data
- To relocate code or data, only the number in the relevant segment register needs to be changed

Consequences:

- A program can be located anywhere in memory without making any changes to it (addresses are not absolute, but offsets relative to start of segments)
- Program writer needs not worry about actual memory structure (map) of the computer used to execute it

Disadvantages:

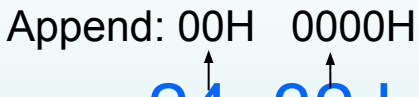
- Complex hardware and for address generation
- Address computation delay for every memory access
- Software limitation: Program size limited by segment size (64KB with the 8086)

Limitations of the above real mode segmentation scheme

- Segment size is fixed at and limited to 64 KB
- Segment can not begin at an arbitrary memory address…

  With 20-bit memory addressing, can only begin at addresses starting with 0H, i.e. at 16 byte intervals

  Principle is difficult to apply with 80286 and above, with segment registers remaining at 16-bits!



  80286 and above use 24, 32 bit addresses but still 16-bit segment registers

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

→ Use memory segmentation in the protected mode

Protected Mode Segmentation:

Protected mode is where Windows operates.

Primarily, what is needed:

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

Similarities with real mode

- When data and programs are addressed in extended memory, the **offset address** is still **used to access information** located within the memory segment.

Difference with real mode

- Segment address is no longer in the protected mode.
- In place of segment address , Segment register contains a **selector**.
- Selector selects a **descriptor** from a descriptor table.
- Descriptor describes the memory segments location, length and access rights

Selectors and Descriptors

- **Selectors:**

-Located in the segment register

-Selects one descriptors.

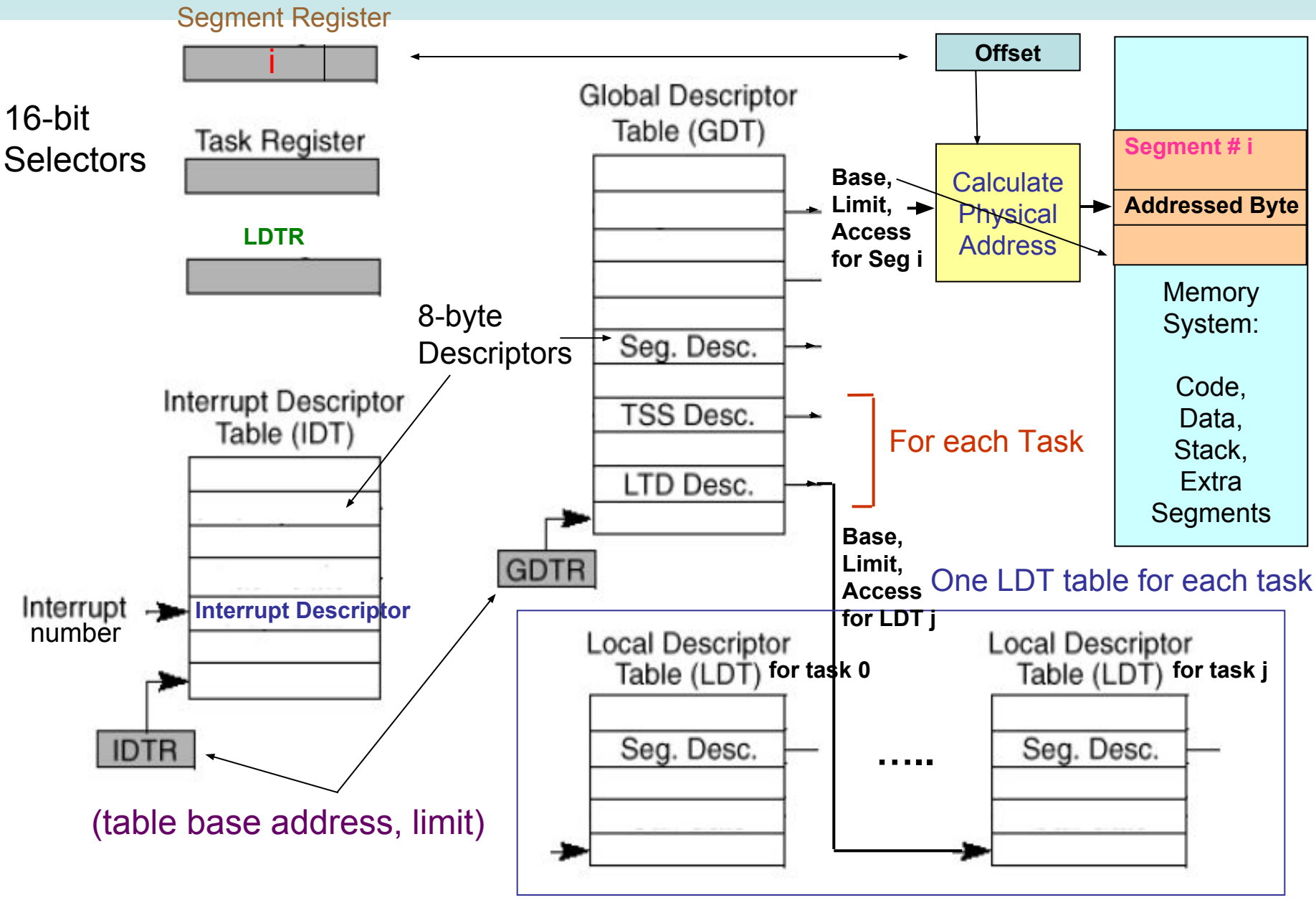
- Example:
- In real mode, CS=0008H, starting address: 00080H.
- In protected mode, the segment number can address any memory location in the entire system for the CS

- **Descriptor:**

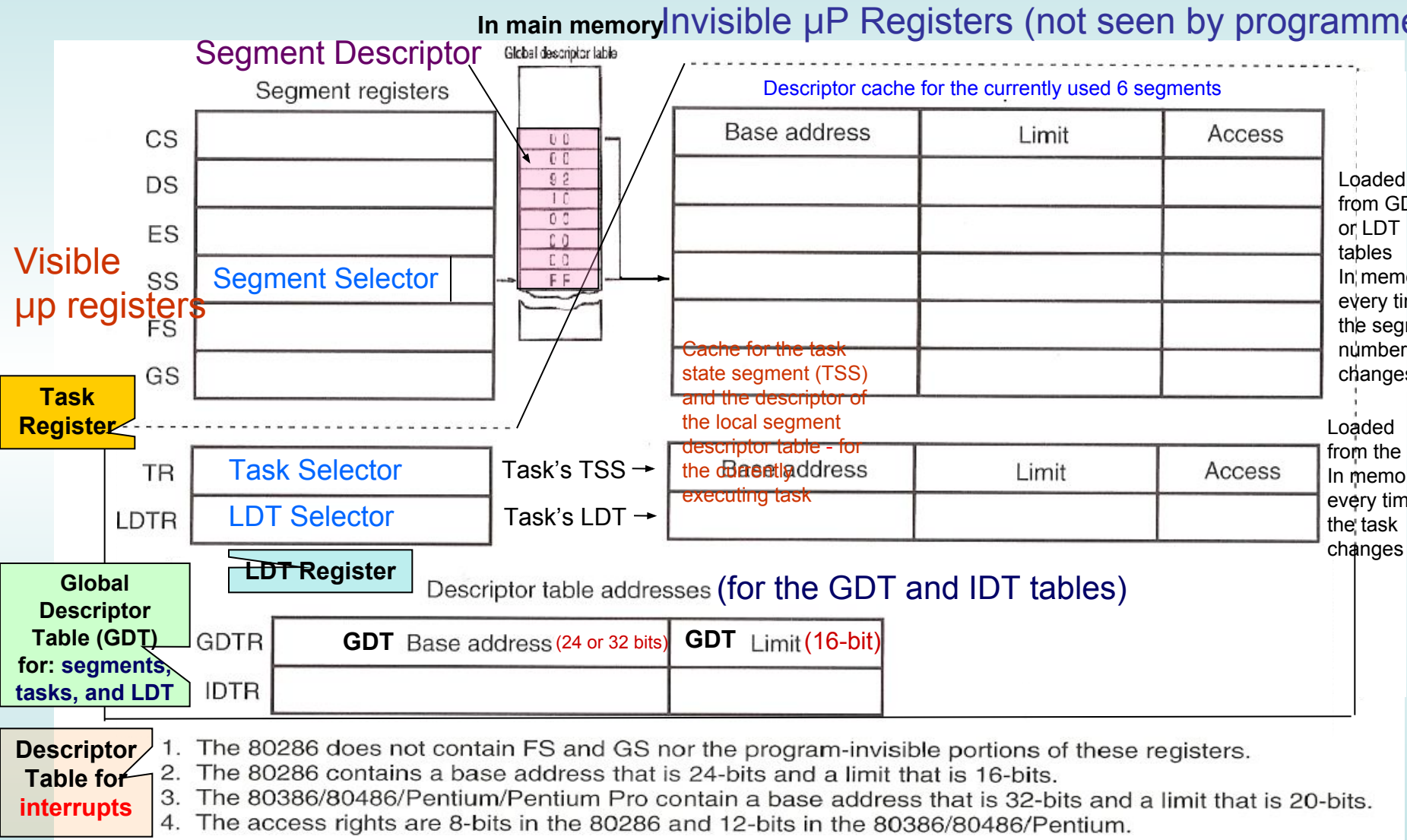
❑ Two descriptor tables.

- Global/system descriptor: contains segment definitions that apply to all program.
- Local/ application descriptor: unique to an application.
- Each table contains 8192 descriptors. Total 16384

Types of Descriptor Tables in memory



Program Invisible Registers (caches)



LDT for a task is accessed through a descriptor in the GDT

Chapter 2 Summary

- Described the μP programming model and purpose and function of program-visible registers
- Described the Flags register and the purpose of each flag bit
- Described how memory is accessed using segmentation, both in the real mode and the protected mode
- Described the program-invisible registers
- Described the structures and operation of the memory paging mechanism
- Described the organizational model of the 8086 μP
- Reviewed the evolution of the 80X86 architecture: Pipelining □ Super pipelining □ Super scalar □ Multi core