Week 2

80x86 Processor Architecture

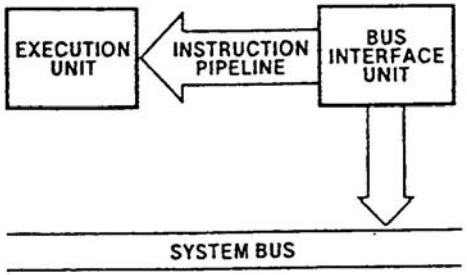
Evolution of Intel's Micoprocessors

Product	8080	8085	8086	8088	80286	80386	80486
Year	1974	1976	1978	1979	1982	1985	1989
Clock Rate	2-3	3-8	5-10	5-8	6-16	16-33	25-50
# of Trans.	4500	6500	29000	29000	130000	275000	1.2 M
Physical Memory	64K	64K	1M	1M	16M	4G	4G
Internal Data Bus	8	8	16	16	16	32	32
External Data Bus	8	8	16	8	16	32	32
Address	16	16	20	20	24	32	32
Data Type	8	8	8,16	8,16	8,16	8,16,32	8,16,32

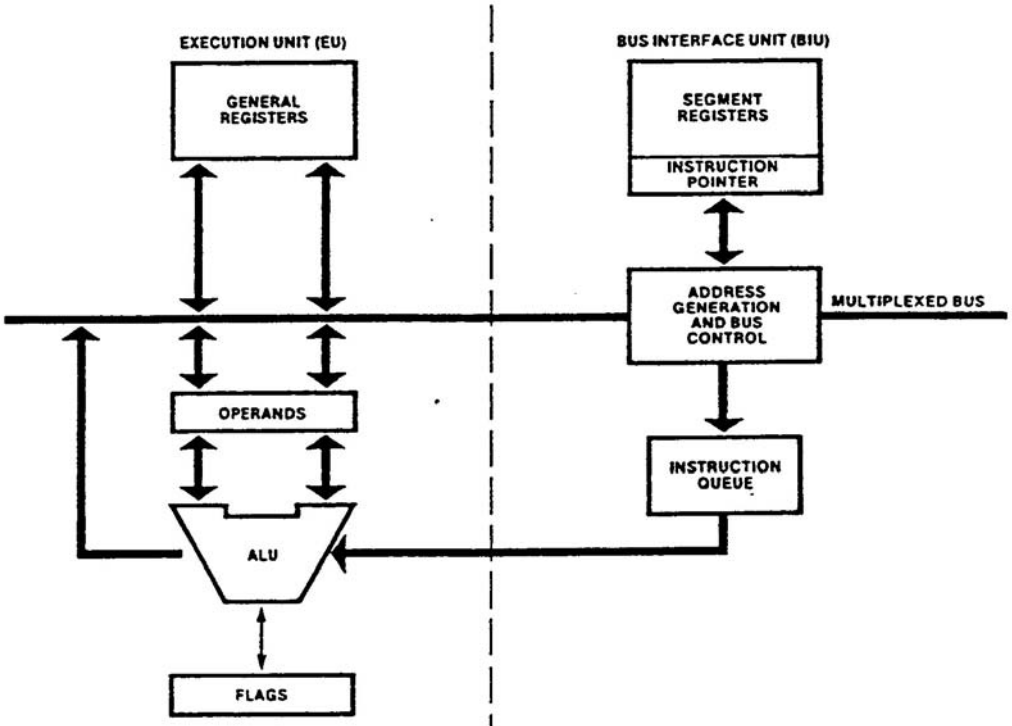
The 8086 and 8088

- The 8086 microprocessor represents the foundation upon which all the 80x86 family of processors have been built
- Intel has made the commitment that as new generations of microprocessors are developed, each will maintain software compatibility with this first generation part
- *Processor model*
  - BIU (Bus Interface Unit) provides hardware functions including generation of the memory and I/O addresses for the transfer of data between itself and the outside world
  - EU (Execution Unit) receives program instruction codes and data from the BIU executes these instructions and stores the results in the general registers.
  - EU has no connection to the system busses; it receives and outputs all its data through the BIU.

Execution and Bus Interface Units



(a)



(b)

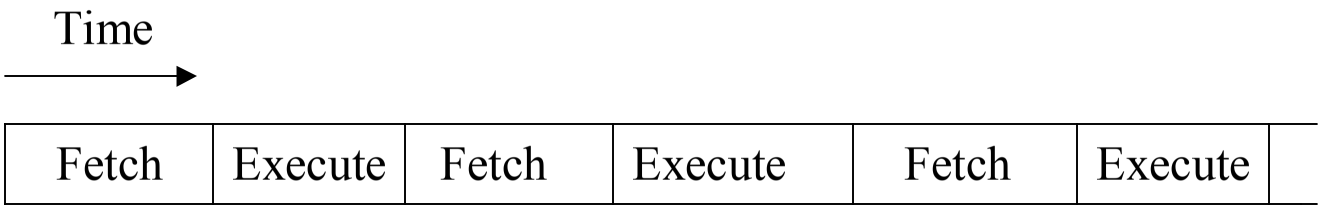
Fetch and Execute Cycle

- Fetch and execute cycles overlap
  - BIU outputs the contents of the IP onto the address bus
  - Register IP is incremented by one or more than one for the next instruction fetch
  - Once inside the BIU, the instruction is passed to the queue; this queue is a first-in-first-out register sometimes likened to a pipeline
  - Assuming that the queue is initially empty the EU immediately draws this instruction from the queue and begins execution
  - While the EU is executing this instruction, the BIU proceeds to fetch a new instruction.
    - BIU will fill the queue with several new instructions before the EU is ready to draw its next instruction
  - The cycle continues with the BIU filling the queue with instructions and the EU fetching and executing these instructions

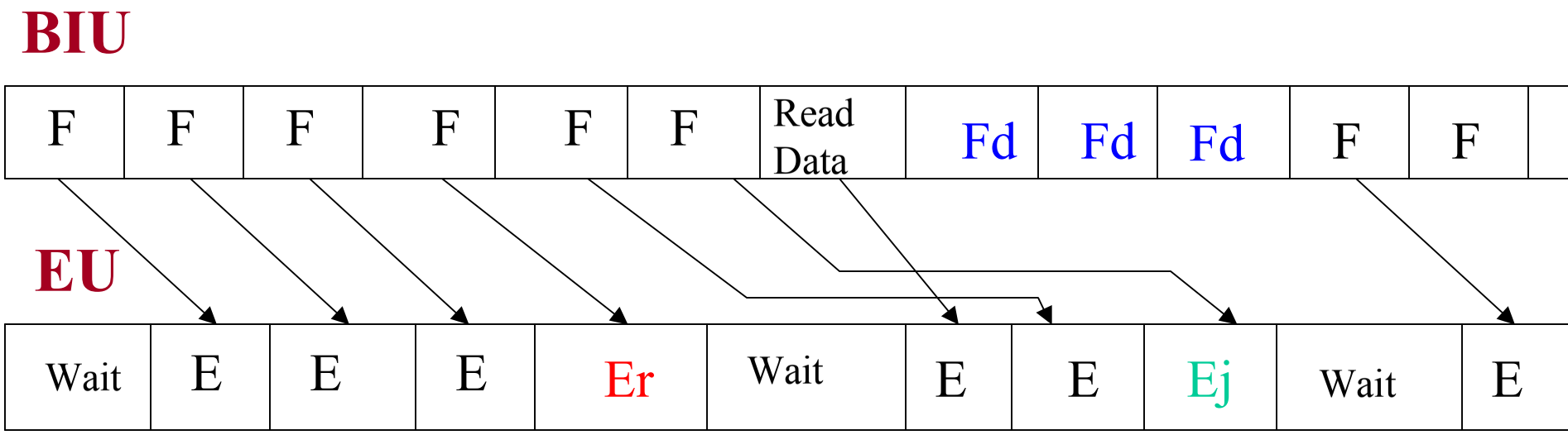
Pipelined Architecture

- Three conditions that will cause a wait mode
  - when the instruction requires access to a memory location not in the queue
  - when the instruction to be executed is a jump instruction
  - during the execution of slow instructions BIU waits
    - for example the instruction AAM (ASCII Adjust for Multiplication) requires 83 clock cycles to complete for an 8086
- 8086 vs 8088
  - BIU data bus width 8 bits for 8088, BIU data bus width 16 bits for 8086
  - 8088 instruction queue is four bytes instead of six
  - 8088 is found to be 30% slower than 8086
    - WHY?
  - Long instructions provide more time for the BIU to fill the queue

Nonpipelined vs pipelined architecture



Non-pipelined architecture



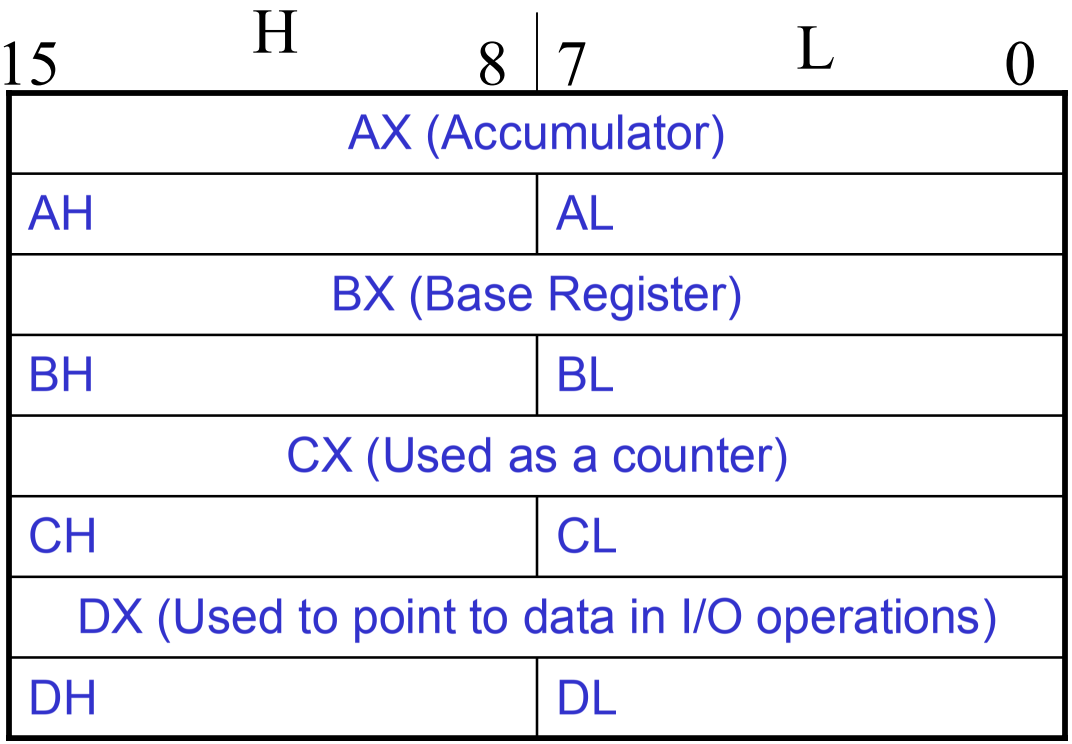
Pipelined architecture

Er: a request for data not in the queue
Ej: jump instruction occurs
Fd: Discarded

Registers of the 8086/80286 by Category

Category	Bits	Register Names
General	16	AX,BX,CX,DX
	8	AH,AL,BH,BL,CH,CL,DH,DL
Pointer	16	SP (Stack Pointer), Base Pointer (BP)
Index	16	SI (Source Index), DI (Destination Index)
Segment	16	CS(Code Segment) DS (Data Segment) SS (Stack Segment) ES (Extra Segment)
Instruction	16	IP (Instruction Pointer)
Flag	16	FR (Flag Register)

General Purpose Registers



- Data Registers are normally used for storing temporary results that will be acted upon by subsequent instructions
- Each of the registers is 16 bits wide (AX, BX, CX, DX)
- General purpose registers can be accessed as either 16 or 8 bits e.g., AH: upper half of AX, AL: lower half of AX

Data Registers

Register	Operations
AX	Word multiply, word divide, word I/O
AL	Byte multiply, byte divide, byte I/O, decimal arithmetic
AH	Byte multiply, byte divide
BX	Store address information
CX	String operations, loops
CL	Variable shift and rotate
DX	Word multiply, word divide, indirect I/O

Pointer and Index Registers

SP	Stack Pointer
BP	Base Pointer
SI	Source Index
DI	Destination Index
IP	Instruction Pointer

The registers in this group are all 16 bits wide

Low and high bytes are not accessible

These registers are used as memory pointers

- Example: MOV AH, [SI]

  Move the byte stored in memory location
  whose address is contained in register SI to register AH

IP is not under direct control of the programmer

16 bit Segment Register Assignments

Type of Memory Reference	Default Segment	Alternate Segment	Offset
Instruction Fetch	CS	none	IP
Stack Operations	SS	none	SP,BP
General Data	DS	CS,ES,SS	BX, address
String Source	DS	CS,ES,SS	SI, DI, address
String Destination	ES	None	DI

Computer Programming

- Machine Language vs Assembly Language
  - Machine language or object code is the only code a computer can execute but it is nearly impossible for a human to work with
  - E4 27 88 C3 E4 27 00 D8 E6 30 F4 the object code for adding two numbers input from the keyboard
- When programming a microprocessor, programmers often use assembly language
  - This involves 3-5 letter abbreviations for the instruction codes (mnemonics) rather than the binary or hex object codes

Address	Hex Object Code				Mnemonics		Comment
					Op-Code	Operand	
0100	E4	27			IN	AL,27H	Input first number from port 27H and store in AL
0102	88	C3			MOV	BL,AL	Save a copy of register AL in register BL
0104	E4	27			IN	AL,27H	Input second number to AL
0106	00	D8			ADD	AL,BL	Add contents of BL to AL and store the sum in AL
0107	E6	30			OUT	30H,AL	Output AL to port 30H
0109	F4				HLT		Halt the computer

Source code

Edit, Assemble, Test, and Debug Cycle

- Using an *editor*, the source code of the program is created. This means selecting the appropriate instruction mnemonics to accomplish the task
- A compiler program which examines the source code file generated by the editor and determines the object code for each instruction in the program, is then run. In assembly language programming, this is called an *assembler* (MASM (Chapter 2 of the textbook, DEBUG: Appendix A of the textbook, etc.,)
- The object code produced by the computer is loaded into the target computer's memory and is then *run.*
- *Debugging:* locating and fixing the source of error

MOV Instruction

- MOV destination,source
  - 8 bit moves
    - MOV CL,55h
    - MOV DL,CL
    - MOV BH,CL
    - Etc.
  - 16 bit moves
    - MOV CX,468Fh
    - MOV AX,CX
    - MOV BP,DI
    - Etc.

MOV Instruction

- Data can be moved among all registers but data cannot be moved directly into the segment registers (CS,DS,ES,SS).
  - To load as such, first load a value into a non-segment register and then move it to the segment register

```
MOV AX,2345h
MOV DS,AX
```

- Moving a value that is too large into a register will cause an error

```
MOV BL,7F2h        ; illegal
MOV AX,2FE456h  ; illegal
```

- If a value less than than FFh is moved into a 16 bit register. The rest of the bits are assumed to be all zeros.

```
MOV BX,5        ; BX = 0005 with BH  = 00 and BL = 05
```

MOV Instruction

• MOV AX,58FCH	√
• MOV DX,6678H	√
• MOV SI,924BH	√
• MOV BP,2459H	√
• MOV DS,2341H	x
• MOV CX,8876H	√
• MOV CS,3F47H	x
• MOV BH,99H	√

ADD Instruction

- ADD destination,source
- The ADD instruction tells the CPU to add the source and destination operands and put out the results in the destination

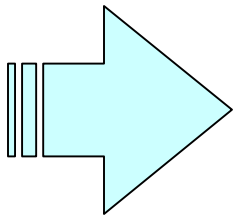
DESTINATION = DESTINATION + SOURCE

```
MOV AL,25H
MOV BL,34h
ADD AL,BL   ; (AL should read 59h once the instruction is executed)
MOV DH,25H
ADD DH,34h   ; (AL should read 59h once the instruction is executed)
```

Immediate operand

ADD Instruction

```
MOV AL,0
ADD AL, [0200]
ADD AL,[0201]
ADD AL,[0202]
ADD AL,[0203]
```



```
MOV AL,0
MOV BX,200H
ADD AL, [BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
```

DATA:
DS:0200 => 25
DS:0201 => 12
DS:0202 => 15
DS:0203 => 1F

Little Endian Convention

"Little Endian" means that the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. (The little end comes first.)
Intel uses Little Endian Convention. For example, a 4 byte LongInt

Byte3 | Byte2 | Byte1 |Byte0 will be arranged in memory as follows:
Base Address+0 Byte0

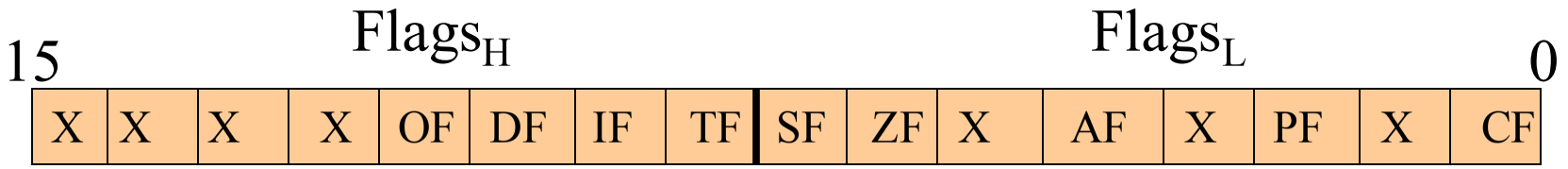
Base Address+1 Byte1

Base Address+2 Byte2

Base Address+3 Byte3

- **Adobe Photoshop** -- Big Endian
- **BMP (Windows and OS/2 Bitmaps)** -- Little Endian
- **GIF** -- Little Endian
- **IMG (GEM Raster)** -- Big Endian
- **JPEG** -- Big Endian

Status and Control Flags



- Six of the flags are status indicators reflecting properties of the last arithmetic or logical instruction.
- For example, if register AL = 7Fh and the instruction ADD AL,1 is executed then the following happen
  - AL = 80h
  - CF = 0; there is no carry out of bit 7
  - PF = 0; 80h has an odd number of ones
  - AF = 1; there is a carry out of bit 3 into bit 4
  - ZF = 0; the result is not zero
  - SF = 1; bit seven is one
  - OF = 1; the sign bit has changed
- Can be used to transfer program control to a new memory location; for example:

  ADD AL,1
  JNZ 0100h

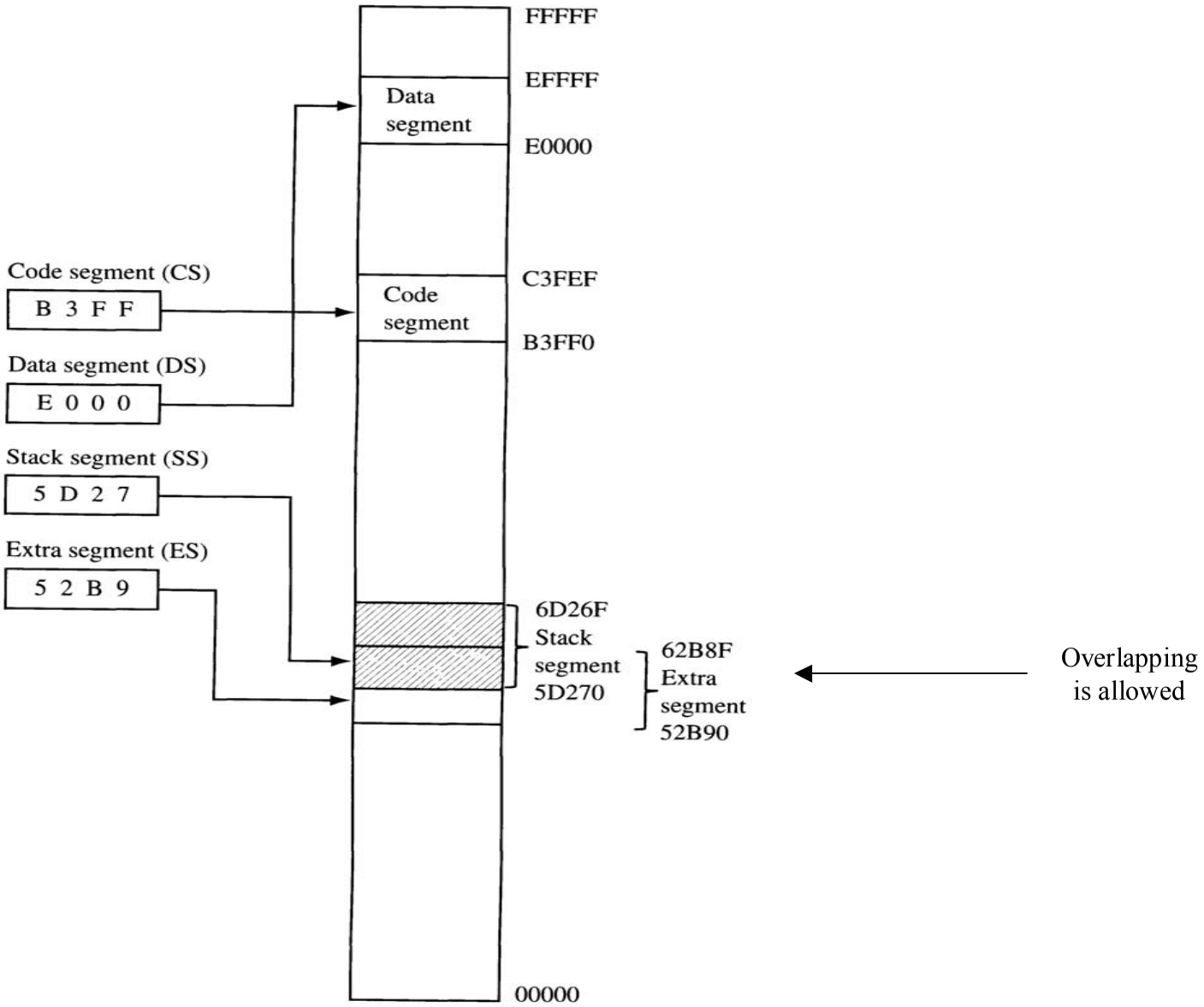
TF, IF, and DF

- Three of the flags can be set or reset directly by the programmer and are used to control the operation of the microprocessor, these are TF, IF, and DF.
- When TF (Trap Flag) is set, control is passed to a special address after each instruction is executed. Normally a program to display all the registers and flags is stored there. Single-stepping mode.
- When IF (Interrupt Flag) is set, external interrupt requests on the 8086's interrupt line INTR will be enabled.
  - For example a printer may spend several seconds printing a page of text from its internal buffer
  - When it is ready for new data, the printer control circuit drives the 8086's INTR inout line
  - The processor then suspends whatever it is doing and begins running the printer *interrupt service routine* (ISR)
  - When the routine has finished via a IRET (interrupt return) instruction, control is transferred back to the original instruction in the main program that was executing when the interrupt occurred
  - Hardware and software interrupts
- Direction Flag (DF) is used with block move instructions.
  - DF = 1 then the block memory pointer will automatically decrement (STD)
  - DF = 0, then the block memory pointer will automatically increment (CLD)

Origin and Definition of a Segment

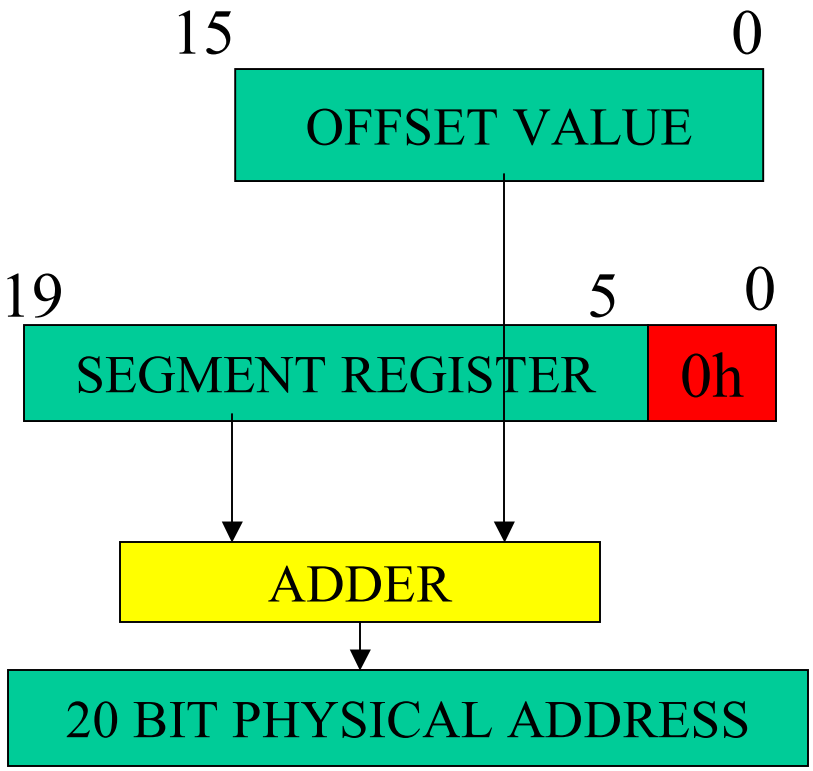
- A segment is an area of memory that includes up to 64 Kbytes and begins on an address divisible by 16 (such an address ends with an hex digit 0h)
  - 8085 could address 64Kbytes 16 address lines
- In the 8085, 64 K is for code, data, and stack
- In the 8086/88, 64 K is assigned to each category
  - Code segment
  - Data segment
  - Stack Segment
  - Extra Segment

Segment Registers



Logical and Physical Addresses

- Addresses within a segment can range from address 0 to address FFFFh. This corresponds to the 64Kbyte length of the segment
- An address within a segment is called an *offset* or *logical address*
- *Ex.* Logical address 0005h in the code segment actually corresponds to B3FF0h + 5 = B3FF5h.



Example 1:
Segment base value: 1234h
Offset: 0022h

12340h
+ 0022h

12362h is the physical 20 bit address

Two different logical addresses may correspond to the same physical address
D470h in ES 2D90h in SS or
ES:D470h and SS:2D90h

Example

•If DS=7FA2H and the offset is 438EH

a) Calculate the physical address

7FA20 + 438E = 83DAE

b) calculate the lower range

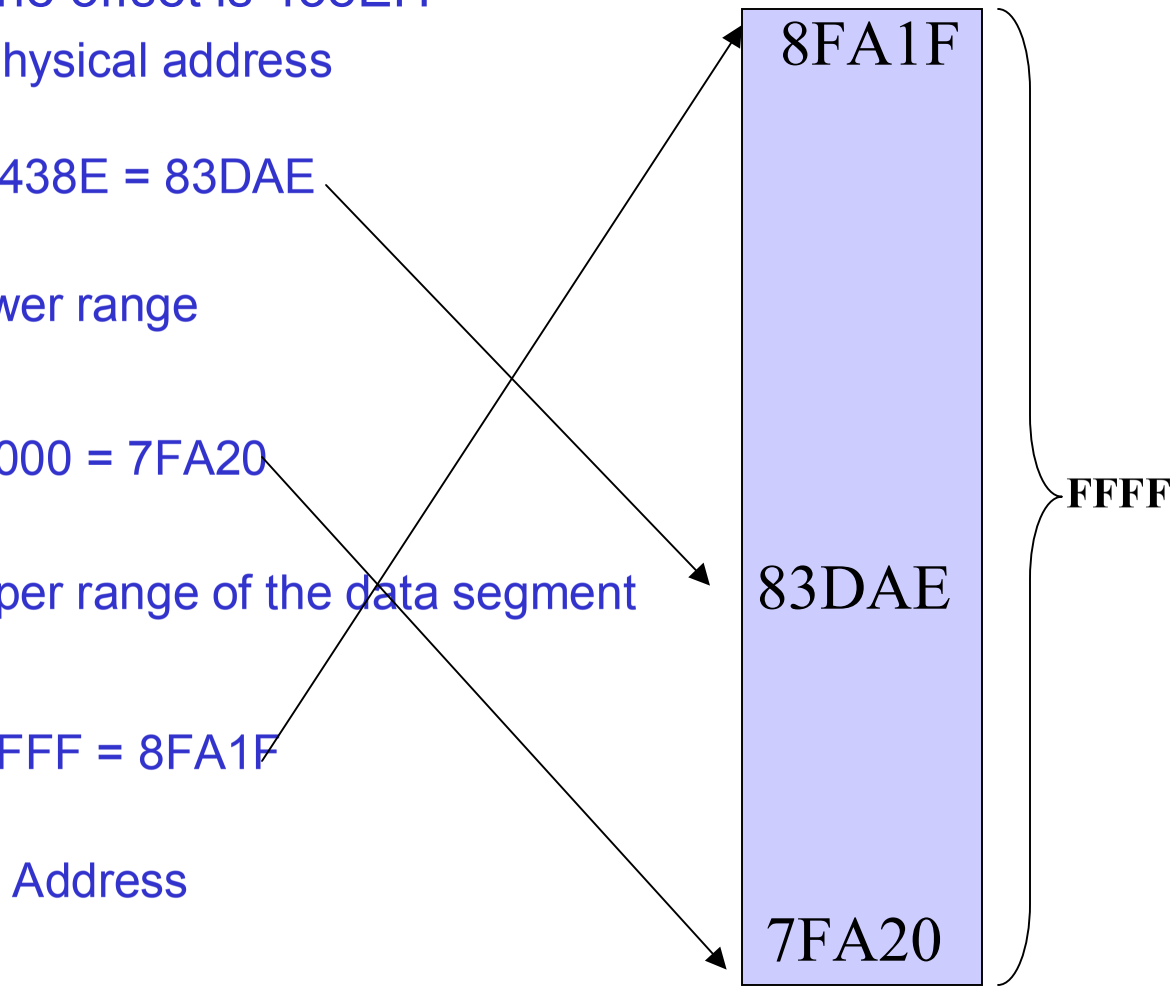
7FA20 + 0000 = 7FA20

c) Calculate the upper range of the data segment

7FA20 + FFFF = 8FA1F

d) Show the logical Address

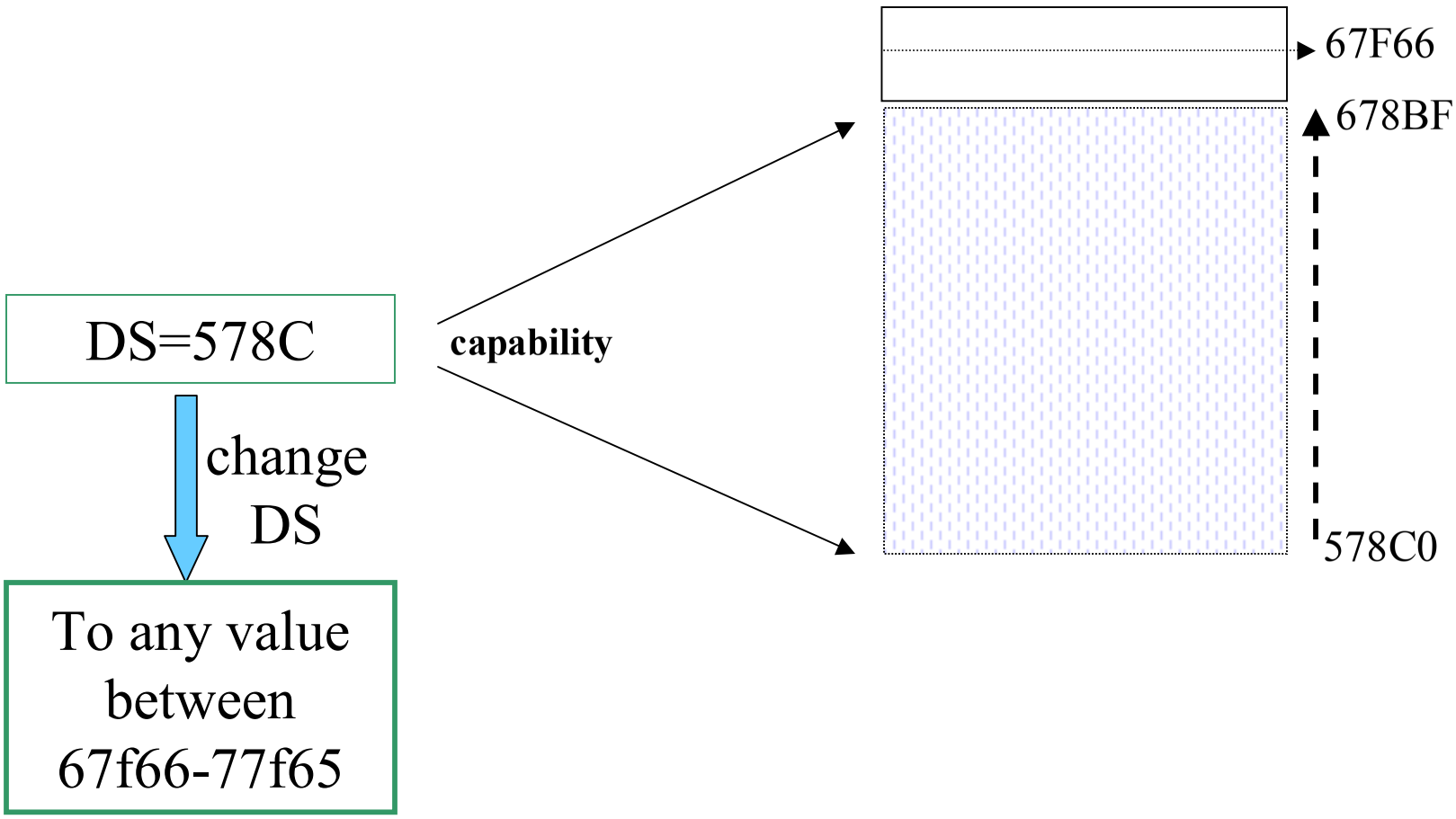
7FA2:438E



Example

Question:

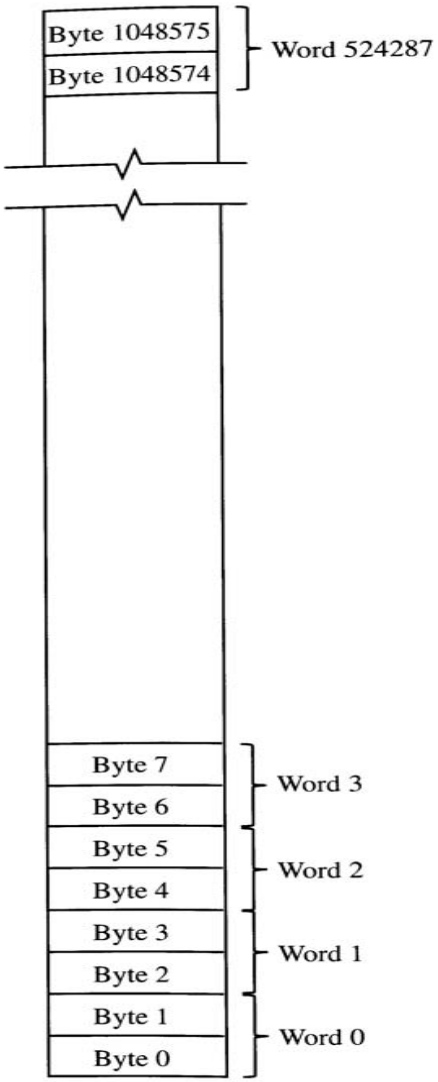
Assume DS=578C. To access a Data in 67F66 what should we do?



Advantages of Segmented Memory

- One program can work on several different sets of data. This is done by reloading register DS to a new value.
- Programs that reference logical addresses can be loaded and run anywhere in the memory: relocatable
- Increases portability, different PC's with different addressing
- Segmented memory introduces extra complexity in both hardware in that memory addresses require two registers.
- They also require complexity in software in that programs are limited to the segment size
- Programs greater than 64 KB can be run on 8086 but the software needed is more complex as it must switch to a new segment.
- Protection among segments is provided (later in 80286)

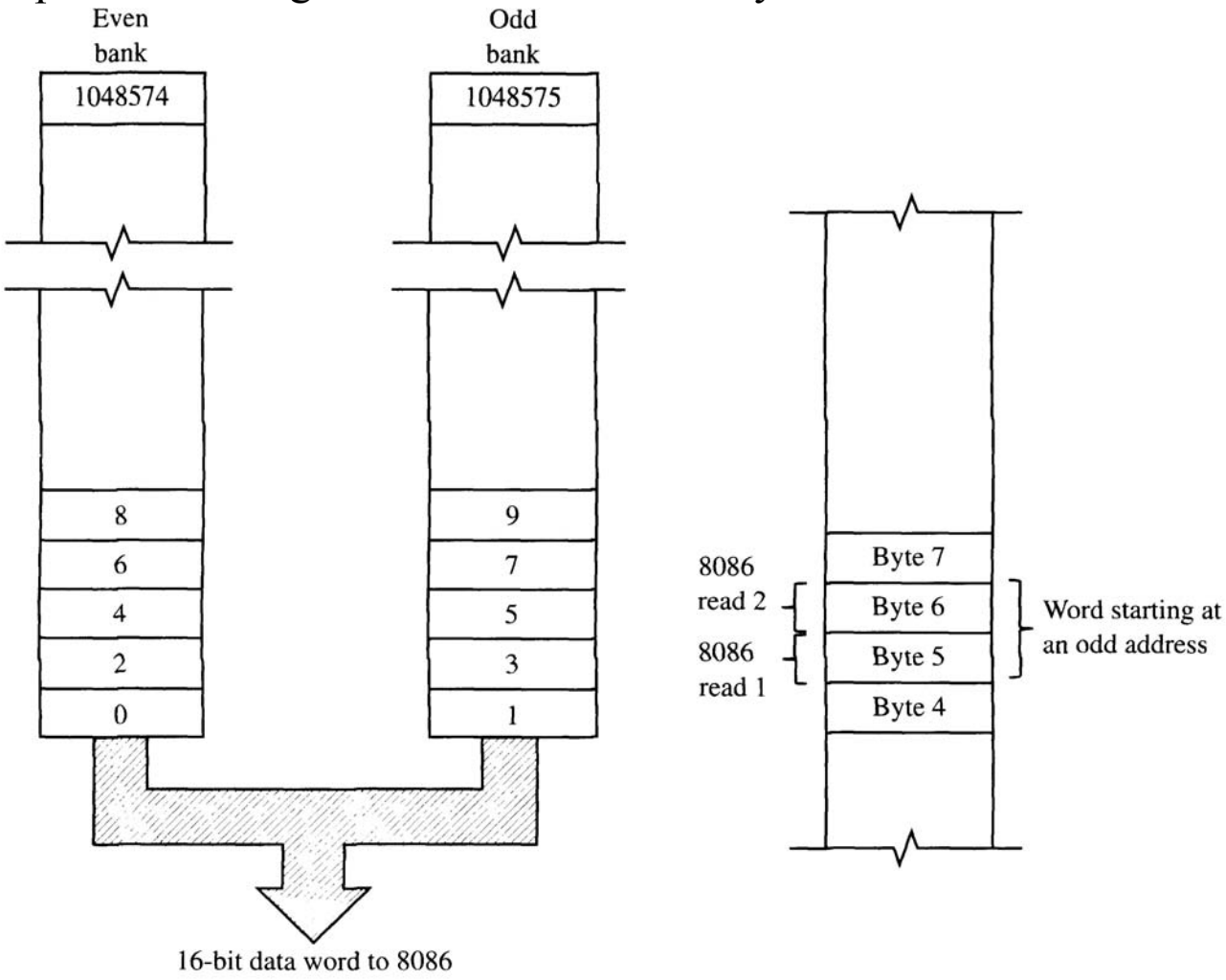
Memory Address Space and Organization



- Word
- Double Word

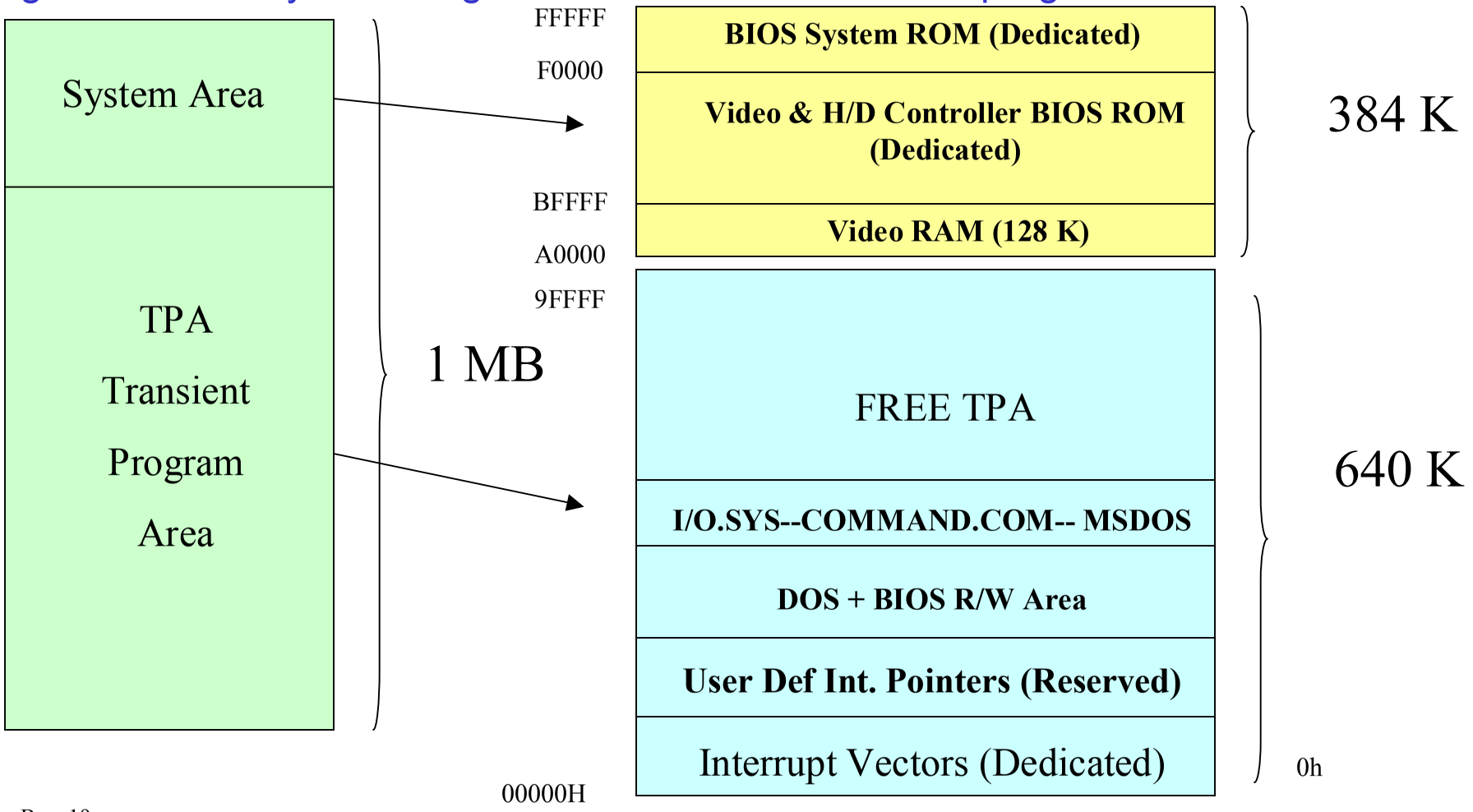
Even addressed and odd-addressed banks

If the CPU has a 16 bit data bus like 8086/286/386SX uprocessors then the address spaces are designated as odd and even bytes. BHE + A0 used to select the banks.



Dedicated, Reserved and General Purpose Memory

•Some address locations have dedicated functions and should not be used as general memory for storage of data or instructions of a program



Brey 19
Mazidi 32

MS-DOS Functions and BIOS Services

- BIOS: usually stored in ROM
  - tells the CPU what to do at startup
  - these routines provide access to the peripheral devices of the PC, such as the keyboard, video, printer, and disk
  - To test all the devices connected to the PC and alert if error

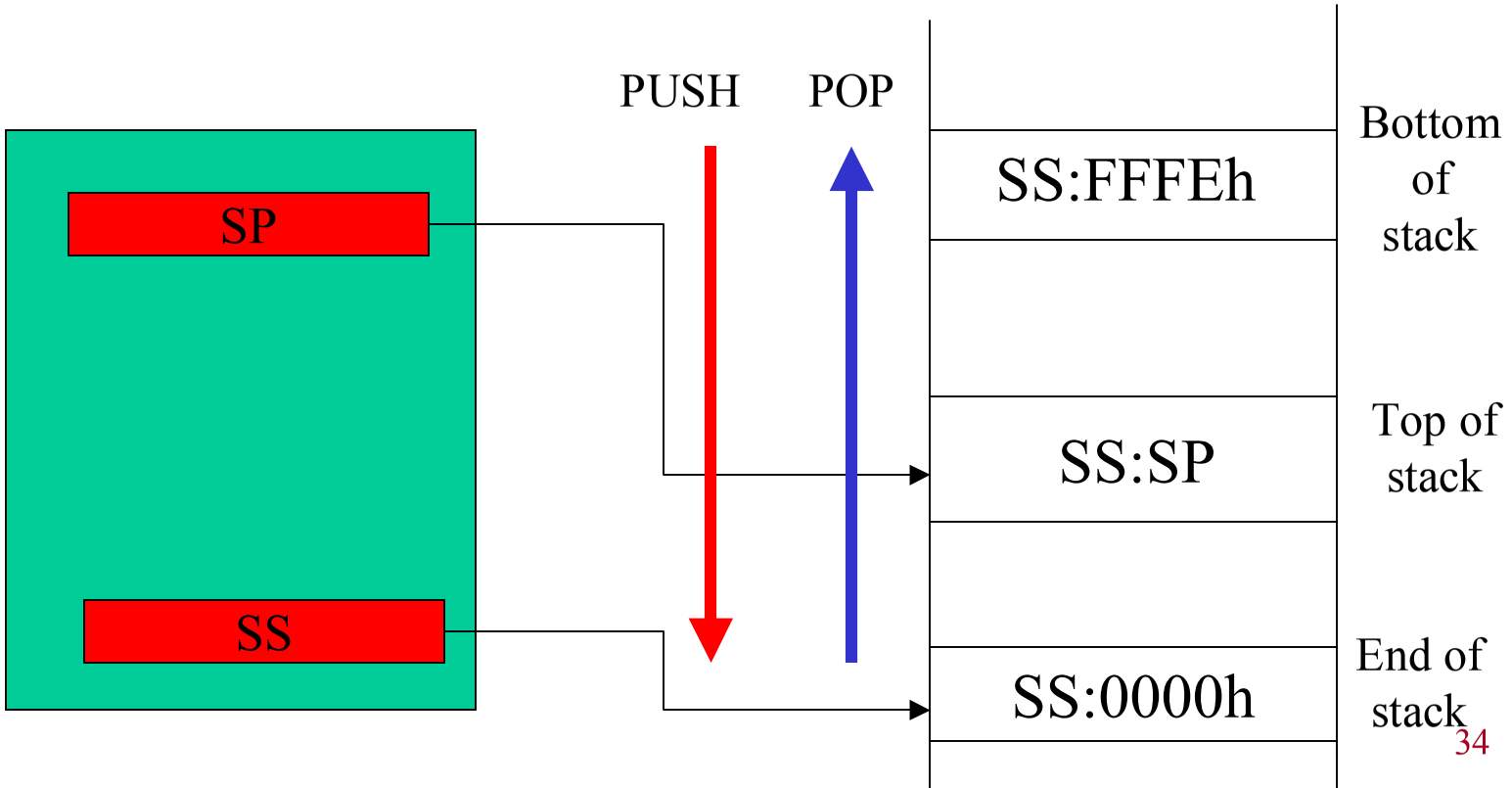
- Access to the BIOS is done through the software interrupt instruction Int *n*
- For example, the BIOS keyboard services are accessed using the instruction INT 16h
- In addition to BIOS services, DOS also provides higher level functions
  - INT 21h
  - More details later

More About RAM

- Memory management is one of the most important functions of the DOS operating systems and should be left to DOS
- Therefore, we do not assign any values for the DS,CS,SS registers; this is the job of DOS
- It is very important to remember that
  - The DS,CS, and DS values we will experiment will be different than those used by the textbook; do not worry

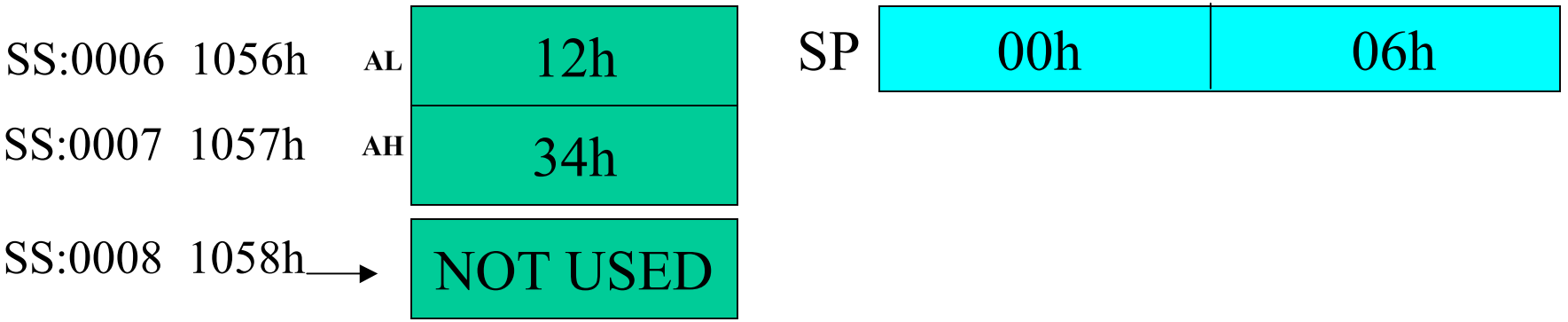
The Stack

- The stack is used for temporary storage of information such as data or addresses; for instance when a CALL is executed the 8088 automatically PUSHes the current value of CS and IP onto the stack.
- Other registers can also be pushed
- Before return from the subroutine, POP instructions can be used to pop values back from the stack into the corresponding registers



Example for PUSH

- Given
  - SS = 0105h
  - SP = 0008h
  - AX = 1234h
  - What is the outcome of the PUSH AX instruction?
- A_{BOS} = 01050 + FFFEh = 1104h
- A_{TOS} = 01050 + 0008h = 1058h
- Decrement the SP by 2 and write AX into the word location 1056h.



Example for POP

- What is the outcome of the following
  - POP AX
  - POP BX
  - if originally 1058h contained AABBh?

1056	12
1057	34
1058	BB
1059	AA

- Read into the specified register from the stack and increment the stack pointer for each POP operation
- At the first POP
  - AX = 1234h SP = 0008h
- At the second POP
  - BX = AABBh

Question? What is SP and Physical address now? →

SP = 000Ah and
Physical Address: 105A

The 80286 and above - Modes of Operation

- Real Mode
  - The address space is limited to 1MB using address lines A0-19; the high address lines are inactive
  - The segmented memory addressing mechanism of the 8086 is retained with each segment limited to 64KB
  - Two new features are available to the programmer
    - –Access to the 32 bit registers
    - –Addition of two new segments F and G

- Protected Mode
  - –Difference is in the new addressing mechanism and protection levels
  - –Each memory segment may range from a single byte to 4GB
  - –The addresses stored in the segment registers are now interpreted as pointers into a descriptor table
  - –Each segment's entry in this table is eight bytes long and identifies the base address of the segment, the segment size, and access rights
  - –In 8088/8086 any program can access the core of the OS hence crash the system. Access Rights are added in descriptor tables.

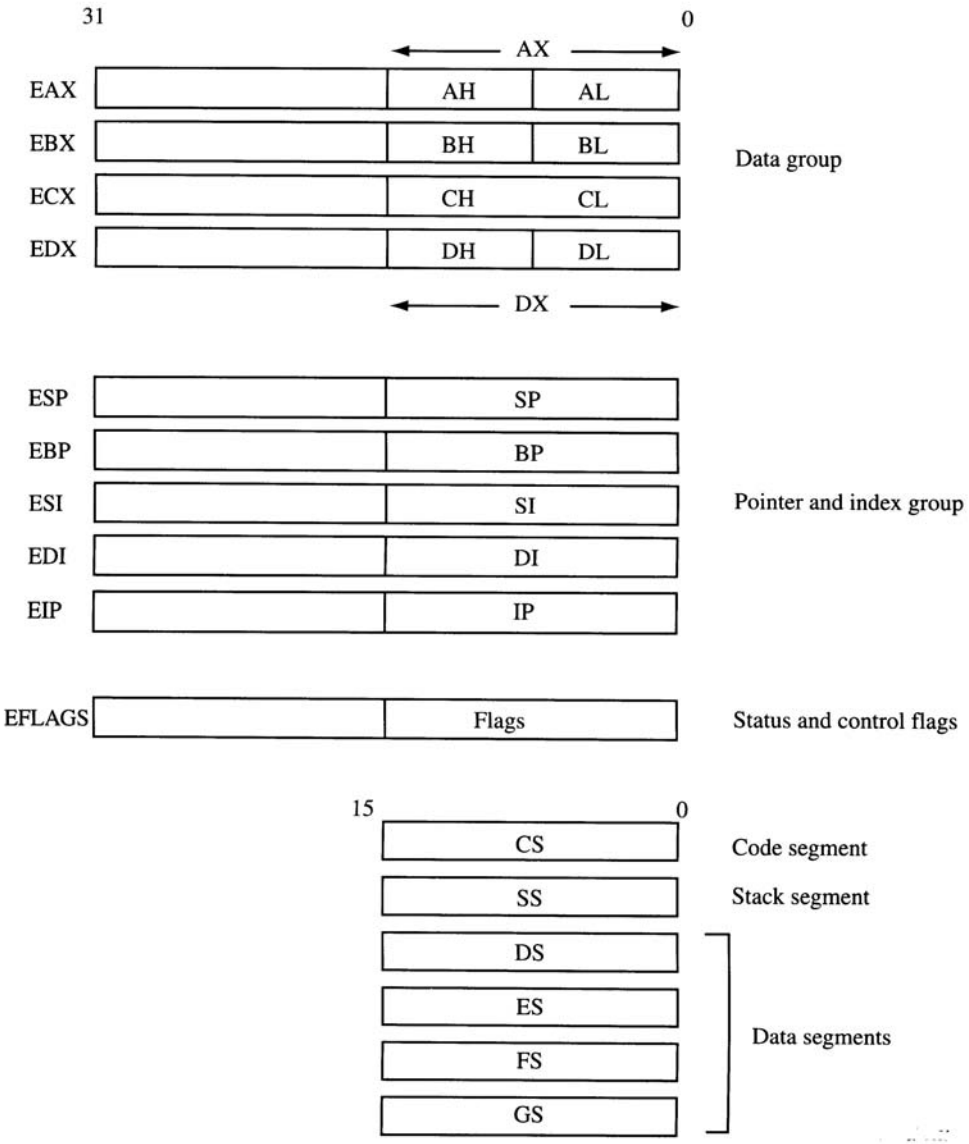
Virtual Memory

- 286 onward supported Virtual Memory Management and Protection™
- Unlimited amount of main memory assumed
- Two methods are used:
  - Segmentation
  - Paging
- Both techniques involve swapping blocks of user memory with hard disk space as necessary
  - If the program needs to access a block of memory that is indicated to be stored in the disk, the OS searches for an available memory block (typically using a least recently used algorithm) and swaps that block with the desired data on the hard drive
  - Memory swapping is invisible to the user
  - Segmentation: the block size is variable ranging up to 4GB
  - Paging: Block sizes are always 4 KB at a time.
- A final protected mode feature is the ability to assign a privilege level to individual tasks (programs). Tasks of lower privilege level cannot access programs or data with a higher privilege level. The OS can run multiple programs each protected from each other.

Virtual 8086 Mode

- Real Mode
  - Only one program can be run one time
  - All of the protection and memory management functions are turned off
  - Memory space is limited to 1MB
- Virtual 8086 Mode
  - The 386 hands each real mode program its own 1MB chunk of memory
  - Multiple 8086 programs to be run simultaneously but protected from each other (multiple MSDOS prompts)
  - Due to time sharing, the response becomes much slower as each new program is launched
  - The 386 can be operated in Protected Mode and Virtual 8086 mode at the same time.
  - Because each 8086 task is assigned the lowest privilege level, access to programs or data in other segments is not allowed thus protecting each task.
  - We'll be using the virtual 8086 mode in the lab experiments on PCs that have either Pentiums or 486s.

Programming Model



Protected Mode - Segmentation

- When operated in Protected Mode, the 386's memory segments are more flexible than when operated in Real Mode
  - Instead of a fixed 64 KB segment size, protected mode segments can be as small as a single byte or as large as 4 GB.
  - Each memory segment is assigned a privilege level
  - Although just 6 segments can be active at a given time, PM allows as many as **16384** total different segments to be defined.

RPL = Requested Privilege Level 00 highest, 11 lowest

TI = Global (0) or Local (1) Descriptor table

SELECTOR Selects one descriptor table

SELECTOR (A15-A3)	TI	RPL(A1-A0)

Protected Mode - Segmentation

–Protected Mode memory segments can readily be swapped out to disk to support virtual memory. Status bits indicate if the selected memory segment is actually present in memory or if the segment is stored on disk necessitating a memory swap

–Memory segments can be marked read-only to protect data and code

–In order to specify a MP memory segment

  –Its starting address

  –Its size

  –Its attributes

  should be known

–Segments in this case are referred to as selectors, and segment registers are used to point to the base of a descriptor table.

–Each entry or descriptor in this table is eight bytes long

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
Segment Base Address B24–B31								Byte 7
G	D	O	AVL	Segment Size Limit L16–L19				Byte 6
P	DPL		S	E	C/ED	R/W	A	Byte 5
Segment Base Address B16–B23								Byte 4
Segment Base Address B8–B15								Byte 3
Segment Base Address B0–B7								Byte 2
Segment Size Limit L8–L15								Byte 1
Segment Size Limit L0–L7								Byte 0

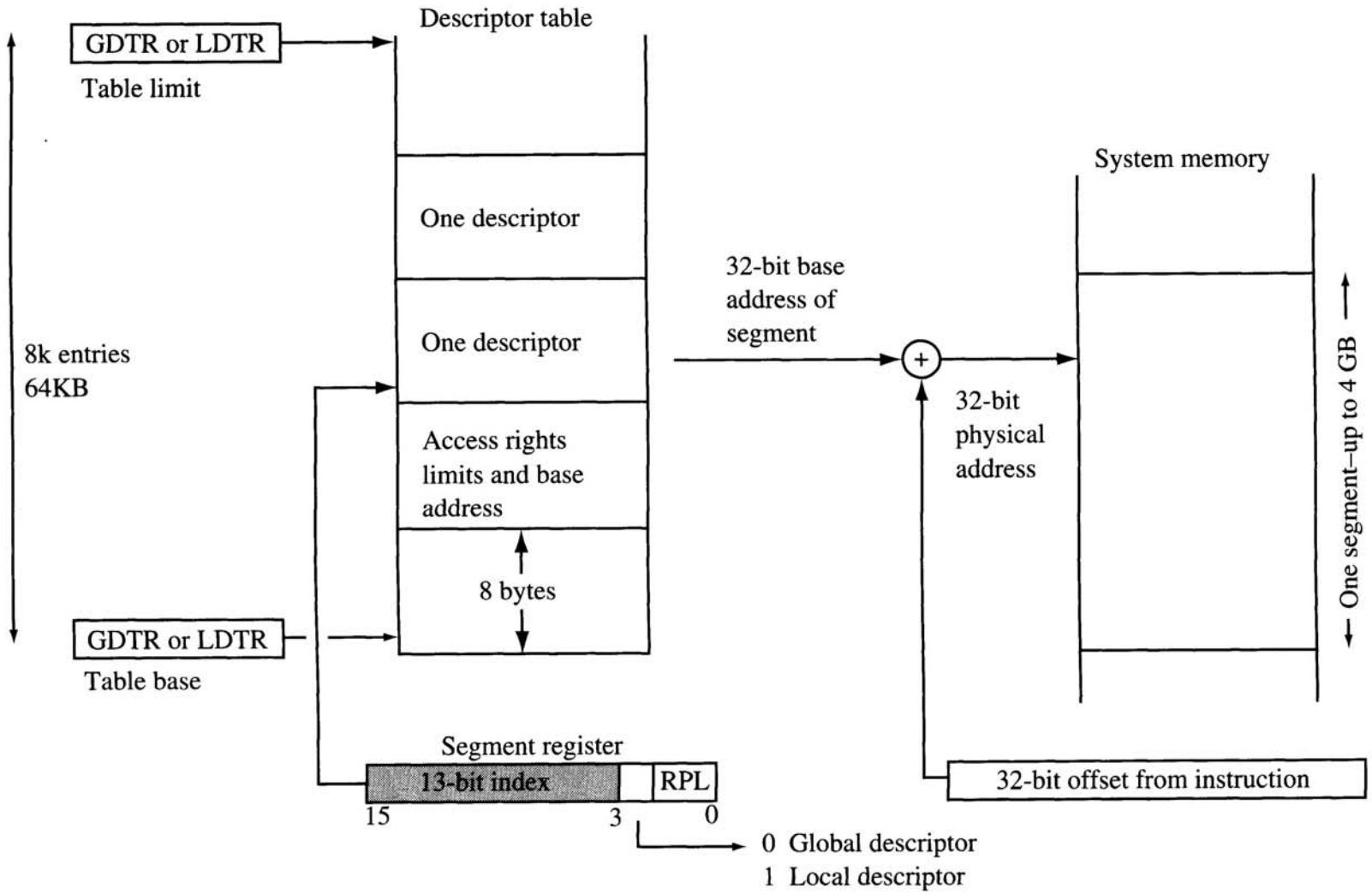
Notes:

Segment Base Address	32-bit physical starting address of the segment.
Segment Size Limit	The segment is specified in bytes with a 1 MB maximum (G = 0) or 4 KB pages with a 4 GB maximum (G = 1).
P (present bit)	The segment is present in main memory when this bit is set. If P = 0 the operating system must bring the segment into memory from the hard disk.
DPL (descriptor privilege level)	The privilege level can vary from 00 (most privileged) to 11 (least privileged).
S (segment descriptor)	If S = 0 the segment is a system descriptor. If S = 1 the segment is a code or data segment.
E (executable)	The segment is executable (E = 1) or data (E = 0).
C/ED (conforming/executable direction)	If E = 1: Then if C/ED = 1 code segment can be executed only when CPL ≥ DPL otherwise these bits are ignored. If E = 0: Then if C/ED = 1 expand up the segment otherwise if C/ED = 0 expand down the segment.
RW (read/write)	Access is read-only (RW = 0) or read/write (RW = 1)
A (accessed)	If the segment has been accessed A = 1. This bit allows the operating system to decide if a segment needs to be saved or can be overwritten (trashed).
G (granularity)	This bit sets the size of the segment (see Segment Size Limit).
D (default operation size)	When D = 0.16-bit addressing and CPU registers are assumed. When D = 1 32-bit addressing and CPU registers are assumed.
AVL	This bit is available to the operating system.

Physical Address Computation

- Four steps are involved:
  - Bit 2 of the segment register is examined to determine if the global or Local descriptor table is to be used
  - The upper 13 bits of the segment register are added to the base address in the system address registers GDTR and LDTR thereby selecting a particular descriptor
  - A fault is generated if EPL (Effective Privilege Level) is less privileged than DPL (Descriptor Privilege Level)
  - If the privilege level of the request has an equal or higher privilege level than the descriptor, the 32 bit offset from the computer instruction is added to the 32 bit base address of the segment (bytes 2,3,4, and 7 of the descriptor) to form the 32 bit physical address

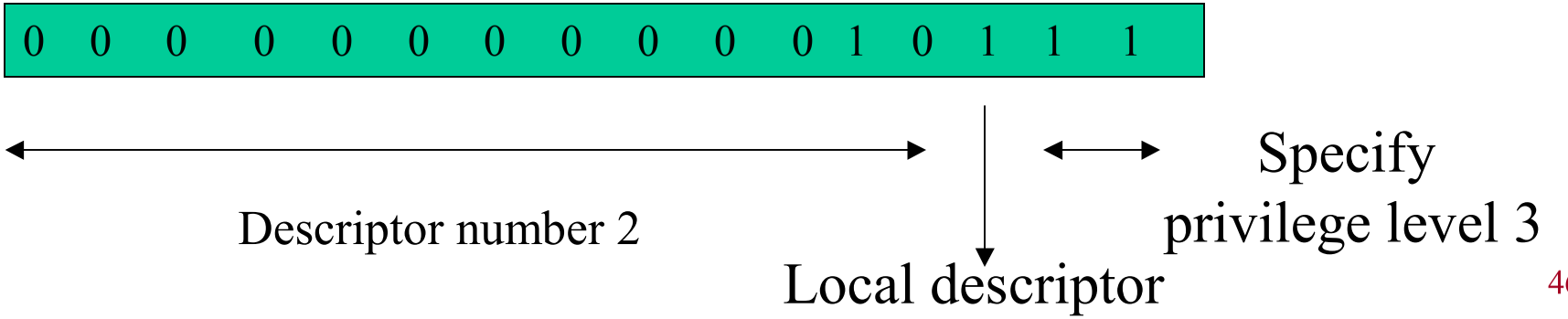
80386 Protected Mode Addressing



Example

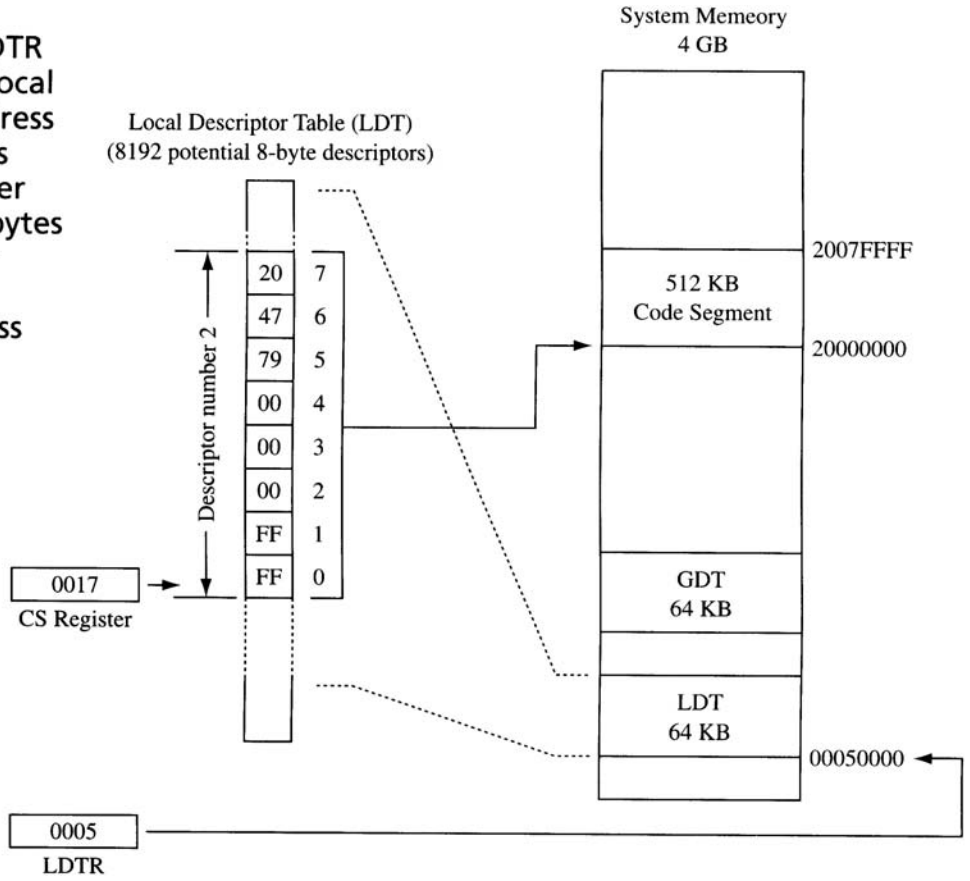
```
MOV AX, 0005h
LLDT AX                 ; copy register AX to LDTR
MOV AX, 0017h
MOV CS, AX              ; copy register AX to segment register CS
```

- The first two instructions point the LDTR register at the base of the local descriptor table
- The 32 bit pointer address is obtained by appending 4 hex digits all zero
- In this example, the base address is 00050000h
- The last two instructions load register CS with 0017h



Example

Figure 3–16.
Example showing register LDTR pointing to the base of the local descriptor table (LDT) at address 00050000H. The CS register is pointing to descriptor number two in this table. The eight bytes that make up this descriptor specify a 512 KB memory segment beginning at address 20000000H.



Example

1. Bytes 2, 3, 4, and 7 define the segment base. In this example, 20000000H.
2. Bytes 0, 1, and the low four bits of byte 6 define the segment size limit. In this example, 7FFFFH. Noting that the granularity bit (bit 7 of byte 6) is low, we interpret this number as the size of the segment in bytes (instead of pages). The size is 512 KB (7FFFFH).
3. The ending segment address is calculated by adding the segment size limit to the base address: 20000000H + 0007FFFFH = 2007FFFFH.
4. The high-order bits of byte 6 = 0100. This indicates that:
   (a) G = 0 The segment limit is interpreted in bytes.
   (b) D = 1 The CPU registers default to 32 bits wide.
   (c) AVL = 0
5. The access byte (byte 5) = 79H = 0111 1001. This indicates that:
   (a) P = 1 The segment is present in memory.
   (b) DPL = 11 The privilege level is 3 (the least trusted).
   (c) S = 1 The segment is a code or data segment.
   (d) E = 1 The segment is executable.
   (e) C/ED = 0 CPL need not be greater than DPL to execute this code.
   (f) R/W = 0 The segment is read-only.
   (g) A = 1 The segment has been previously accessed.