EE222 – Microprocessor Systems

Real Mode Memory System

Arbab Latif

Spring 2022

Resources:
The Intel Microprocessors: Architecture, Programming, and Interfacing, Eighth Edition Barry B. Brey
(**Section 2.2**)

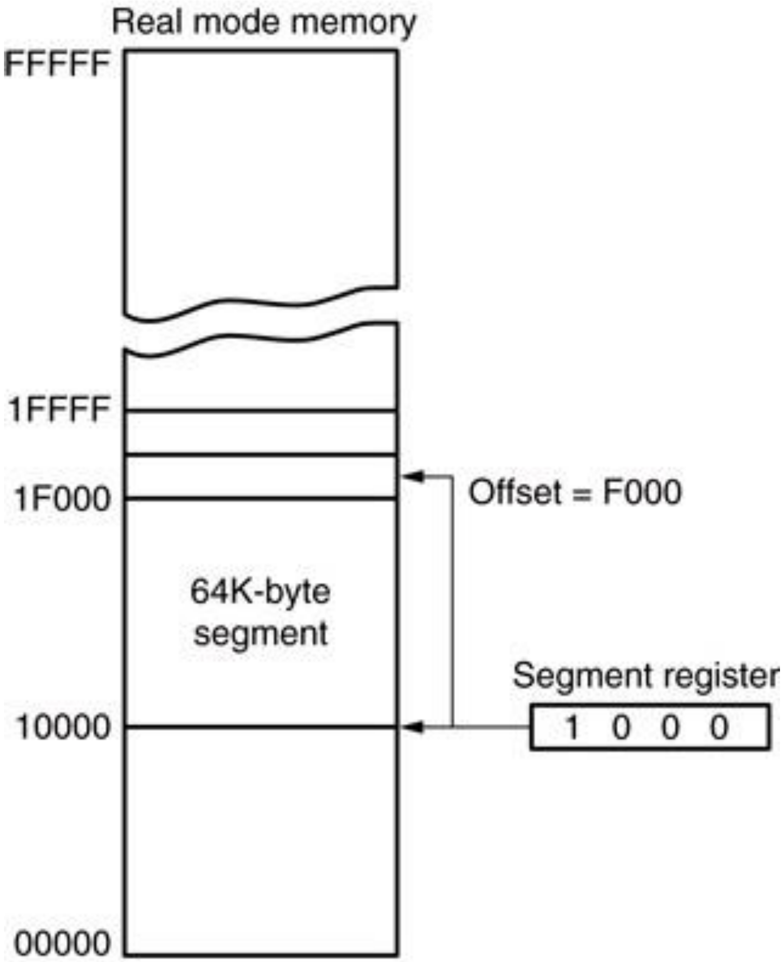
2–2 REAL MODE MEMORY ADDRESSING

- 80286 and above operate in either the real or protected mode.
- **Real mode operation** allows addressing of only the first 1M byte of memory space—even in Pentium 4 or Core2 microprocessor.
  - the first 1M byte of memory is called the **real memory**, **conventional memory**, or **DOS memory** system

Segments and Offsets

- All real mode memory addresses must consist of a segment address plus an offset address.
  - **segment address** defines the beginning address of any 64K-byte memory segment
  - **offset address** selects any location within the 64K byte memory segment
- Figure 2–3 shows how the **segment plus offset** addressing scheme selects a memory location.

Figure 2–3 The real mode memory-addressing scheme, using a segment address plus an offset.



- **this shows a memory segment beginning at 10000H, ending at location IFFFFH**
  - **64K bytes in length**

- **also shows how an offset address, called a displacement, of F000H selects location 1F000H in the memory**

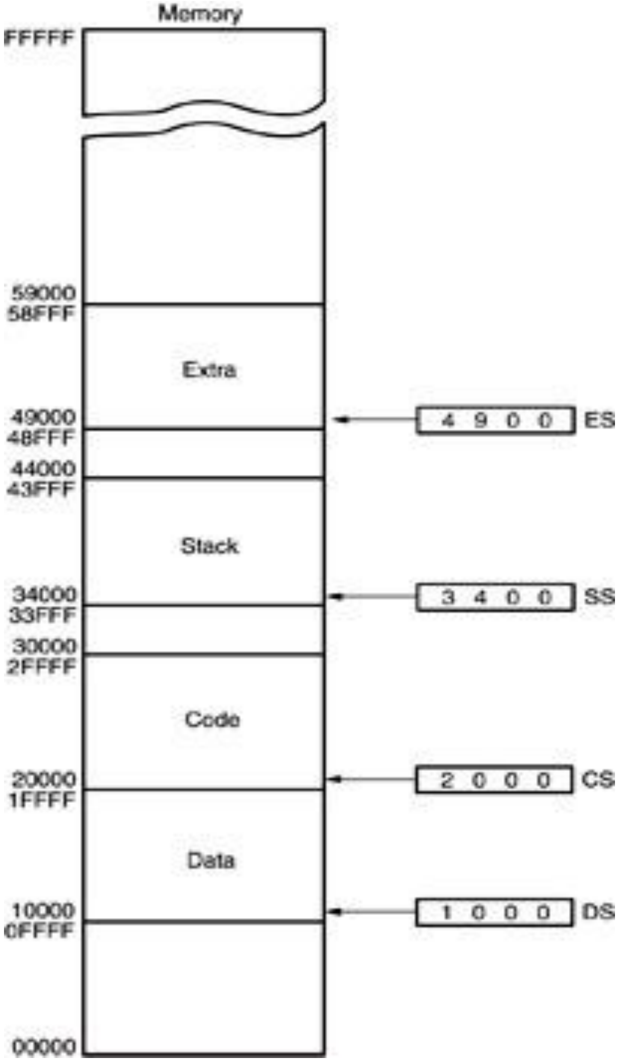
- Once the beginning address is known, the **ending address** is found by adding FFFFH.
  - because a real mode segment of memory is64K in length
- The offset address is always added to the segment starting address to locate the data.
- Segment and offset address is sometimes written as 1000:2000.
  - a segment address of 1000H; an offset of 2000H

Default Segment and Offset Registers

- The microprocessor has rules that apply to segments whenever memory is addressed.
  - these define the segment and offset register combination
- The **code segment** register defines the start of the code segment.
- The **instruction pointer** locates the next instruction within the code segment.

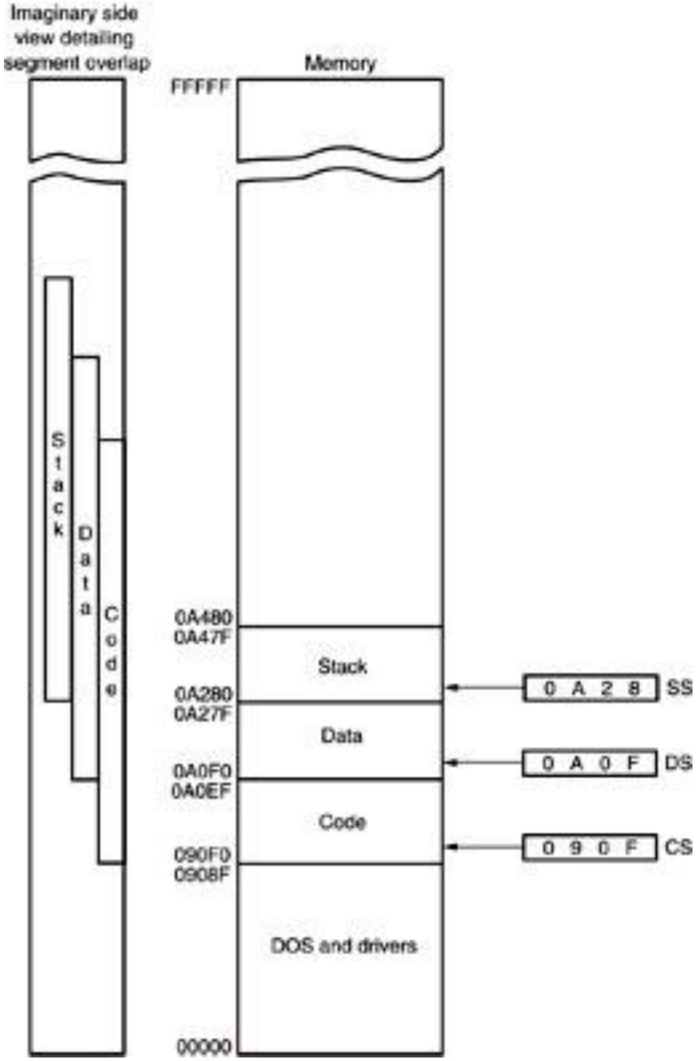
- Another of the default combinations is the **stack**.
  - stack data are referenced through the stack segment at the memory location addressed by either the stack pointer (SP/ESP) or the pointer (BP/EBP)
- Figure 2–4 shows a system that contains four memory segments.
  - a memory segment can touch or overlap if 64K bytes of memory are not required for a segment

Figure 2–4 A memory system showing the placement of four memory segments.



- think of segments as windows that can be moved over any area of memory to access data or code
- a program can have more than four or six segments,
  - but only access four or six segments at a time

Figure 2–5 An application program containing a code, data, and stack segment loaded into a DOS system memory.



- **a program placed in memory by DOS is loaded in the TPA at the first available area of memory above drivers and other TPA programs**
- **area is indicated by a free-pointer maintained by DOS**
- **program loading is handled automatically by the program loader within DOS**

TABLE 2–3 Default 6-bit segment and ffset 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

TABLE 2–4 Default 32-bit segment and offset combinations.

Segment	*Offset*	*Special Purpose*
CS	EIP	Instruction address
SS	ESP or EBP	Stack address
DS	EAX, EBX, ECX, EDX, ESI, EDI, an 8- or 32-bit number	Data address
ES	EDI for string instructions	String destination address
FS	No default	General address
GS	No default	General address

Segment and Offset Addressing Scheme Allows Relocation

- Segment plus offset addressing allows DOS programs to be relocated in memory.
- A **relocatable program** is one that can be placed into any area of memory and executed without change.
- **Relocatable data** are data that can be placed in any area of memory and used without any change to the program.

- Because memory is addressed within a segment by an offset address, the memory segment can be moved to any place in the memory system without changing any of the offset addresses.
- Only the contents of the segment register must be changed to address the program in the new area of memory.
- Windows programs are written assuming that the first 2G of memory are available for code and data.

EE222 – Microprocessor Systems

Addressing

Arbab Latif

Spring 2021

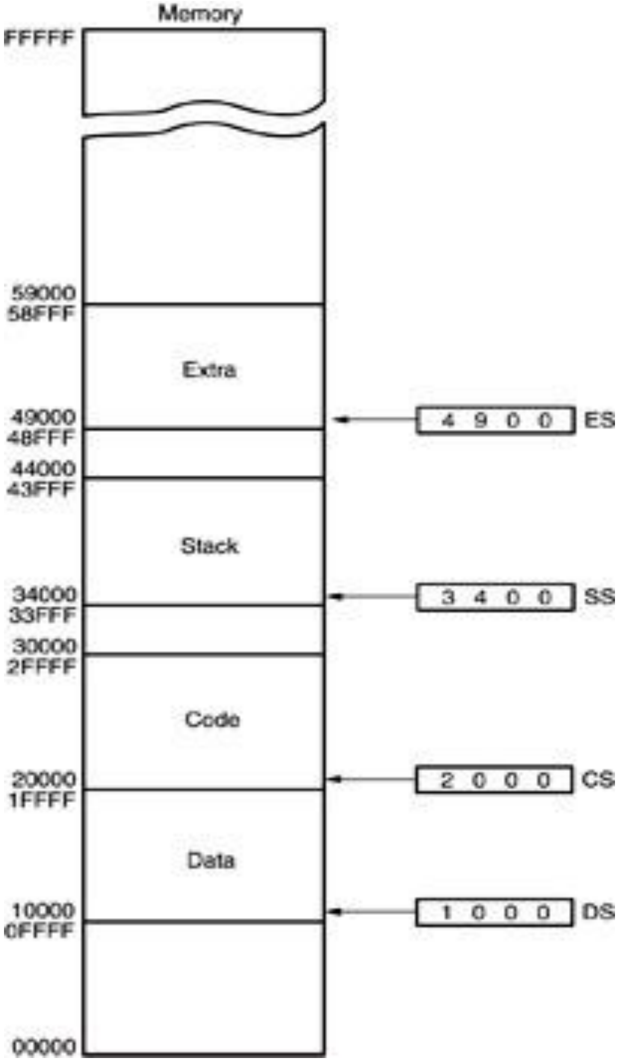
Resources:
The Intel Microprocessors: Architecture, Programming, and Interfacing, Eighth Edition Barry B. Brey
(**Section 1.2, 2.2**)

Default Segment and Offset Registers

- The microprocessor has rules that apply to segments whenever memory is addressed.
  - these define the segment and offset register combination
- The **code segment** register defines the start of the code segment.
- The **instruction pointer** locates the next instruction within the code segment.

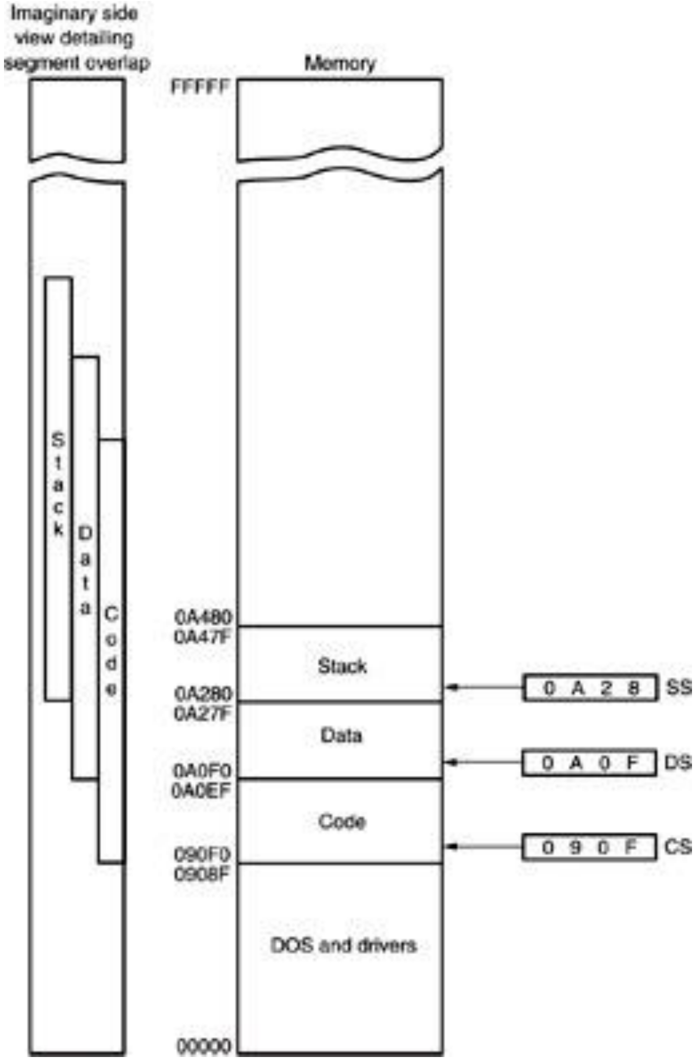
- Another of the default combinations is the **stack**.
  - stack data are referenced through the stack segment at the memory location addressed by either the stack pointer (SP/ESP) or the pointer (BP/EBP)
- Figure 2–4 shows a system that contains four memory segments.
  - a memory segment can touch or overlap if 64K bytes of memory are not required for a segment

Figure 2–4 A memory system showing the placement of four memory segments.



- think of segments as windows that can be moved over any area of memory to access data or code
- a program can have more than four or six segments,
  - but only access four or six segments at a time

Figure 2–5 An application program containing a code, data, and stack segment loaded into a DOS system memory.



- **a program placed in memory by DOS is loaded in the TPA at the first available area of memory above drivers and other TPA programs**
- **area is indicated by a free-pointer maintained by DOS**
- **program loading is handled automatically by the program loader within DOS**

TABLE 2–3 Default 6-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

TABLE 2–4 Default 32-bit segment and offset combinations.

Segment	*Offset*	*Special Purpose*
CS	EIP	Instruction address
SS	ESP or EBP	Stack address
DS	EAX, EBX, ECX, EDX, ESI, EDI, an 8- or 32-bit number	Data address
ES	EDI for string instructions	String destination address
FS	No default	General address
GS	No default	General address

Segment and Offset Addressing Scheme Allows Relocation

- Segment plus offset addressing allows DOS programs to be relocated in memory.
- A **relocatable program** is one that can be placed into any area of memory and executed without change.
- **Relocatable data** are data that can be placed in any area of memory and used without any change to the program.

- Because memory is addressed within a segment by an offset address, the memory segment can be moved to any place in the memory system without changing any of the offset addresses.
- Only the contents of the segment register must be changed to address the program in the new area of memory.
- Windows programs are written assuming that the first 2G of memory are available for code and data.

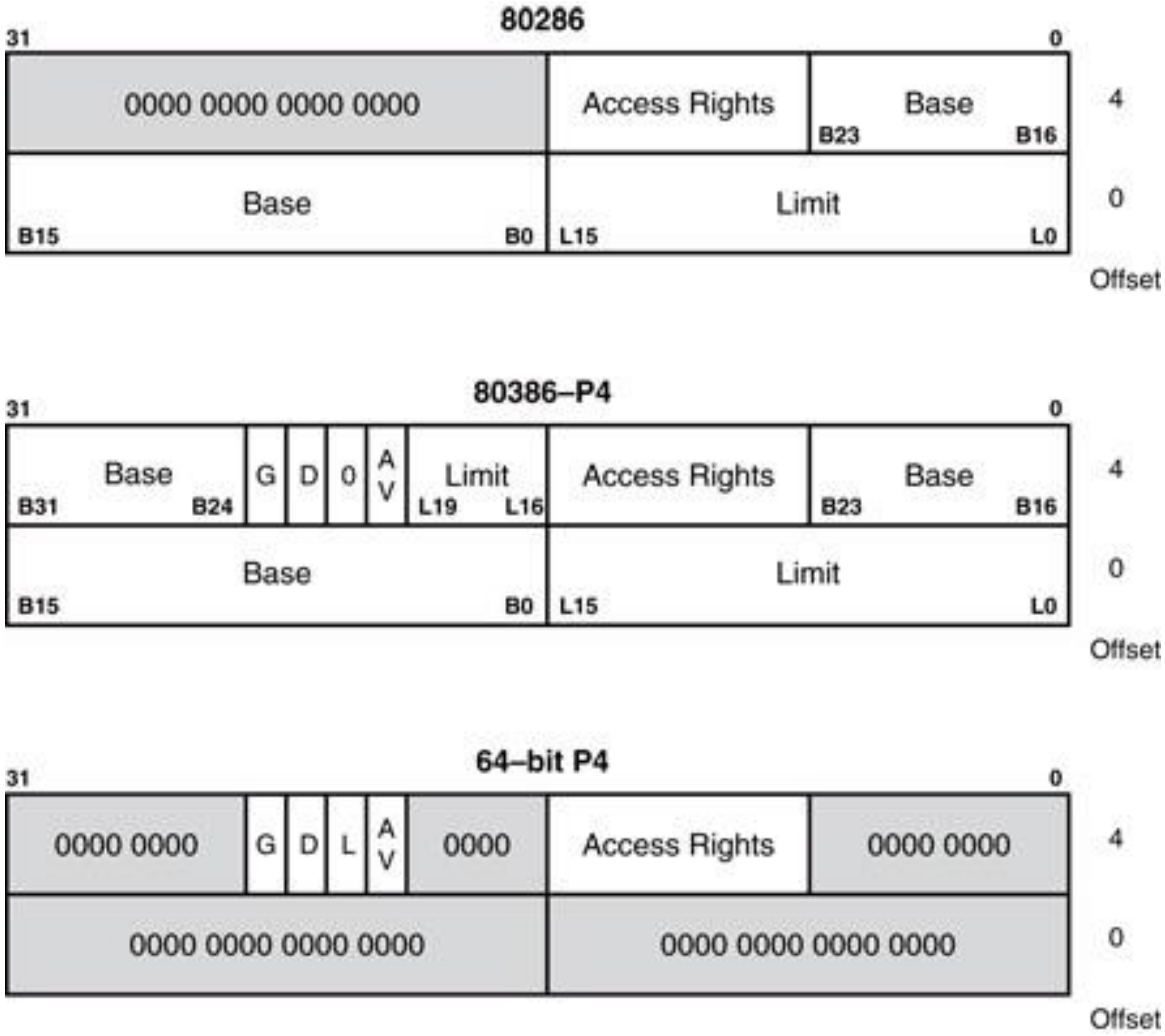
2–3 INTRO TO PROTECTED MODE MEMORY ADDRESSING

- Allows access to data and programs located within & above the first 1M byte of memory.
- **Protected mode** is where Windows operates.
- In place of a segment address, the segment register contains a **selector** that selects a descriptor from a descriptor table.
- The **descriptor** describes the memory segment’s location, length, and access rights.

Selectors and Descriptors

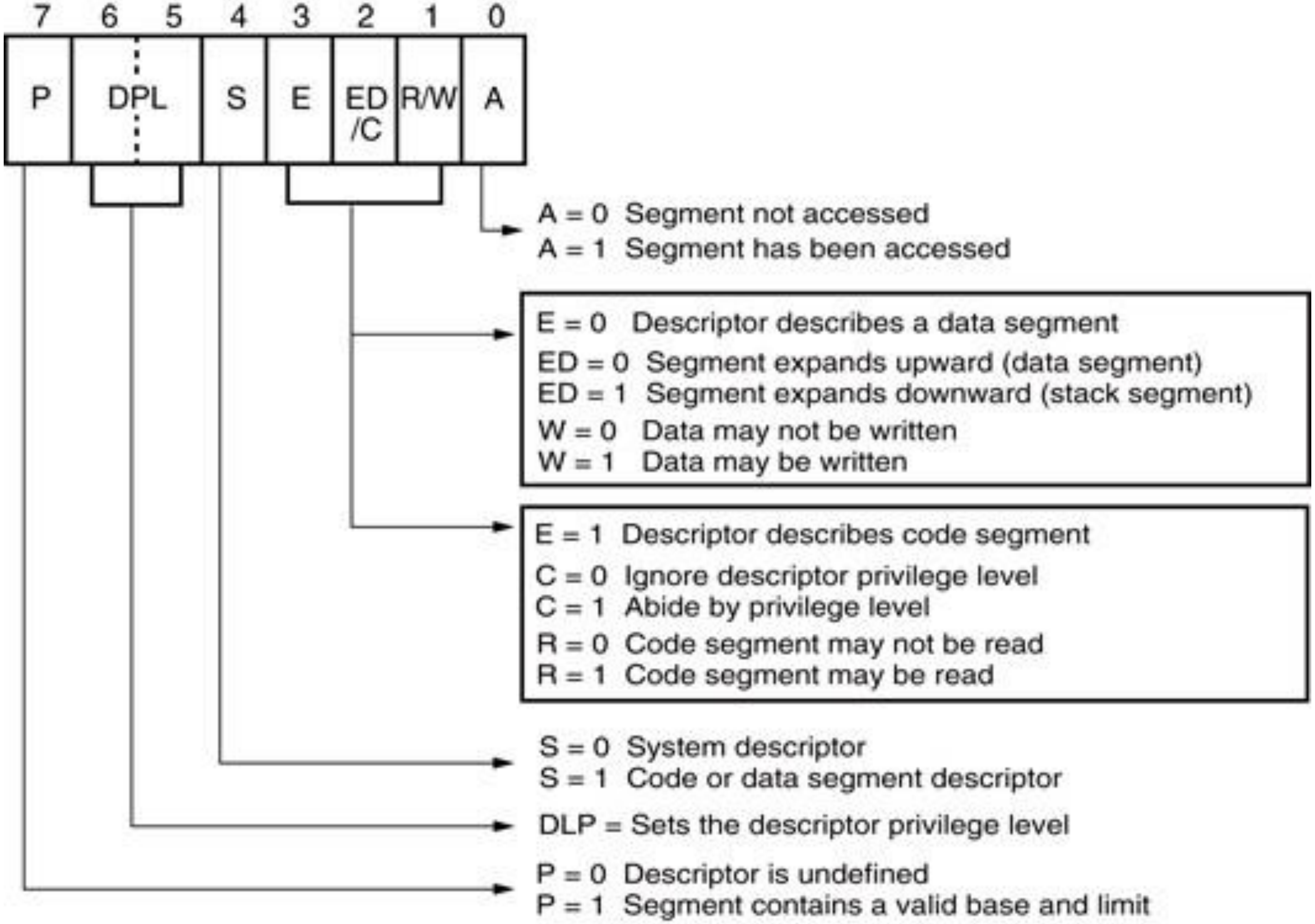
- The **descriptor** is located in the segment register & describes the location, length, and access rights of the segment of memory.
  - it selects one of 8192 descriptors from one of two tables of descriptors
- **Global descriptors** contain segment definitions that apply to all programs.
- **Local descriptors** are usually unique to an application.
  - a global descriptor might be called a **system descriptor,** and local descriptor an **application descriptor**

Figure 2–6 The 80286 through Core2 64-bit descriptors.



each descriptor is 8 bytes in length global and local descriptor tables are a maximum of 64K bytes in length

Figure 2–7 The access rights byte for the 80286 through Core2 descriptor.



A = 0 Segment not accessed
A = 1 Segment has been accessed

E = 0 Descriptor describes a data segment
ED = 0 Segment expands upward (data segment)
ED = 1 Segment expands downward (stack segment)
W = 0 Data may not be written
W = 1 Data may be written

E = 1 Descriptor describes code segment
C = 0 Ignore descriptor privilege level
C = 1 Abide by privilege level
R = 0 Code segment may not be read
R = 1 Code segment may be read

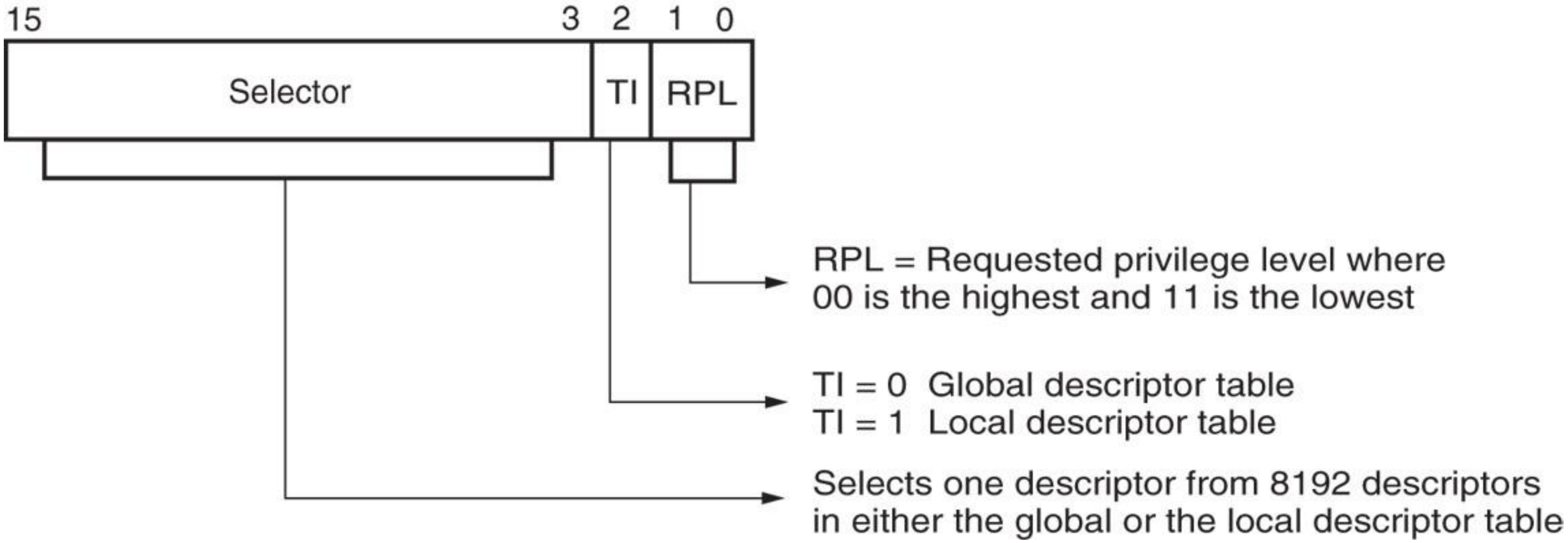
S = 0 System descriptor
S = 1 Code or data segment descriptor

DLP = Sets the descriptor privilege level

P = 0 Descriptor is undefined
P = 1 Segment contains a valid base and limit

Note: Some of the letters used to describe the bits in the access rights bytes vary in Intel documentation.

Figure 2–8 The contents of a segment register during protected mode operation of the 80286 through Core2 microprocessors.



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

Program-Invisible Registers

- Global and local descriptor tables are found in the memory system.
- To access & specify the table addresses, 80286–Core2 contain program-invisible registers.
  - not directly addressed by software
- Each segment register contains a program-invisible portion used in the protected mode.
  - often called cache memory because cache is any memory that stores information

- The GDTR (**global descriptor table register**) and IDTR (**interrupt descriptor table register**) contain the base address of the descriptor table and its limit.
  - when protected mode operation desired, address of the global descriptor table and its limit are loaded into the GDTR
- The location of the local descriptor table is selected from the global descriptor table.
  - one of the global descriptors is set up to address the local descriptor table

- To access the local descriptor table, the LDTR (**local descriptor table register**) is loaded with a selector.
  - selector accesses global descriptor table, & loads local descriptor table address, limit, & access rights into the cache portion of the LDTR
- The TR (task register) holds a selector, which accesses a descriptor that defines a task.
  - a task is most often a procedure or application
- Allows multitasking systems to switch tasks to another in a simple and orderly fashion.