Chapter 7

The Pentium Processor

7–1 The main purpose of registers is to provide a scratch pad so that the processor can keep data on a temporary basis. For example, the processor may keep the procedure return address, stack pointer, instruction pointer, and so on. Registers are also used to keep the data handy so that it can avoid costly memory accesses. Keeping frequently accessed data in registers is a common compiler optimization technique.

7–2 Pentium supports the following three address spaces:
1. Linear address space
2. Physical address space
3. I/O address space (from discussion in Section 1.7)

7–3 In segmented memory organization, memory is partitioned into segments, where each segment is a small part of the memory. In the real mode, each segment of memory is a linear contiguous sequence of up to 64 KB. In the protected mode, it can be up to 4 GB.

Pentium supports segmentation largely to provide backward compatibility to 8086. Note that 8086 is a 16-bit processor with 20 address lines. This mismatch between the processor’s 16-bit registers and 20-bit addresses is solved by using the segmented memory architecture. This segmented architecture has been carried over to Pentium. However, in the protected mode, it is possible to consider the entire memory as a single segment; thus, segmentation is completely turned off.

7–4 In the real mode, a segment is limited to 64 KB due to the fact that 16 bits are used to indicate the offset value into a segment. This magic number 16 is due to the 16-bit registers used 8086 processor. Note that the Pentium emulates 8086 in the real mode.

7–5 In the real mode, Pentium emulates 8086 processor, which is a 16-bit processor (i.e., all its internal registers are 16 bits wide). Since 8086’s address bus is 20 bits wide but its internal registers are all 16 bits in size, a segments start address is stored in a 16-bit segment register with the assumption that the least significant four bits of the 20-bit address are zeroes. Thus, segments can only start at

1
addresses that have the least significant four bits as zero (that is, address must be a multiple of 16). For example, a segment can start at address 16, 32, 48, and so on.

7–6  (a) Since the least significant four bits (A = 1010) are not zero, a segment cannot be placed at this address.

(b) Since the least significant four bits (5 = 0101) are not zero, a segment cannot be placed at this address.

(c) Since the least significant four bits are zero, a segment can be placed at this address.

(d) Since the least significant four bits are zero, a segment can be placed at this address.

7–7  In the protected mode, a segment can be up to 4 GB.

7–8  This limitation is due to number of segment registers available in Pentium.

7–9  This is due to the 20-bit value used to specify the segment limit. If the granularity bit is zero, the segment size is interpreted in bytes. To specify larger segment sizes, the granularity is increased to 4 KB. Why? This is because the difference $32 - 20 = 12$ bits, which corresponds to 4 KB.

7–10  The Table Indicator (TI) bit indicates whether the local or global descriptor table should be used.

\[
\begin{align*}
0 &= \text{Global descriptor table}, \\
1 &= \text{Local descriptor table}.
\end{align*}
\]

The global descriptor table contains descriptors that are available to all tasks within the system. There is only one GDT in the system. Typically, the GDT contains code and data used by the operating system. The local descriptor table contains descriptors for a given program. There can be several LDTs, each of which may contain descriptors for code, data, stack, and so on. A program cannot access a segment unless there is a descriptor for the segment in either the current LDT or GDT.

7–11  Both LDT and GDT can contain up to $2^{13} = 8192$ 8-bit descriptors. The reason for this is that only 13 bits are used to select a segment descriptor as shown in the following figure:
The conversion of logical address to physical address is straightforward. This translation process is shown below:
The translation process involves adding four least significant zero bits to the segment base value and then adding the offset value. When using the hexadecimal number system, simply add a zero digit to the segment base address at the right and add the offset value. As an example, consider the logical address 1100:450H. The physical address is computed as follows:

\[
\begin{align*}
11000 & \quad \text{(add 0 to the 16-bit segment base value)} \\
+ 450 & \quad \text{(offset value)} \\
\hline
11450 & \quad \text{(physical address).}
\end{align*}
\]

7–13 In protected mode, contents of the segment register are taken as an index into a segment descriptor table to get a descriptor. The segment translation process is shown in the following figure:
Segment descriptors provide the 32-bit segment base address, its size, and access rights. To translate a logical address to the corresponding linear address, the offset is added to the 32-bit base address. The offset value can be either a 16-bit or 32-bit number.

7–14 In the real mode:

- Segment size is limited to 64 KB;
- No explicit segment size indication;
- Segments must begin on 16-byte boundaries;
- No segment descriptors;
- Segmentation cannot be turned off.

In the protected mode:

- Segment size can be up to 4 GB (limited by the memory address space).
- Segment descriptor contains explicit segment size information.
- Segments may be placed anywhere in memory. There is no restriction that they should begin on 16-bit boundaries.
- Segment descriptors provide information on the segment (including segment size, segment type, DPL, and so on)
• If the operating does not use segmentation, segmentation can be turned off. In essence, the entire memory is treated as single segment.

7–15 A processor with 16-bit addresses supports $2^{16} = 64$ KB. The first address is 0000H and the last address is FFFFH.

7–16 All numbers are in hex. The additions in the following are done in hexadecimal.
   (a) $1A2B0 + 019A = 1A44A$
   (b) $39110 + 200 = 39310$
   (c) $25910 + 10B5 = 269C5$
   (d) $11000 + ABCD = 1BBCD$