Lecture 3: Protected Mode Addressing and Paging
Protected Mode Memory Addressing

• “Real” (808x) uses 1MB only
• “Protected” mode (80286 and above)
  – Segment register contains a selector…
  – The selector selects a descriptor from a descriptor table.
    • There is a global and a local descriptor table.
    • Each table holds 8192 descriptors
    • Each descriptor is 8-bytes long
    • So…there are $2 \times 8 \times 8192 = 128$K bytes for the descriptors (64KB for each table)
  – The descriptor describes the memory location, length, and access right (and other things on the 386-P4).
    • Which means that there are as many as 128K memory segments (64K each) accessible in protected mode.
**Descriptor Formats**
(same for global or local)

- Look at 80286
  - 24-bit base address
    - Segments can start anywhere in the 16MB of memory – no paragraphing
  - 16-bit segment limit defines the upper boundary of the segment
    - Each segment therefore has a size limit between 1B and 64KB
Descriptor Formats
(same for global or local)

• Look at 80386 and beyond
  – 32-bit base address
    • segments can start anywhere in the 4GB of memory – no paragraphing
  – 20-bit segment limit defines the upper boundary of the segment
    • Each segment therefore has a size limit between 1B and 1MB (or 4K and 4GB – see granularity bit, G=1)
Descriptor Formats
(same for global or local)

• Look at 80386 – the other control bits
  – G=granularity bit:
    • G=0: limit is 1B-1MB (00000h to FFFFFFFh)
    • G=1: limit is multiplied by 4K (00000xxxh to FFFFFFFxxxh = 4KB-4GB)
  – AV=availability bit
    • AV=1: segment is available (e.g. not being accessed by another thread)
    • AV=0: segment is NOT available
  – D=memory mode
    • D=0: 16-bit instruction mode (DOS)
    • D=1: 32-bit instruction mode – accessible in a protected mode system only (e.g. Windows NT, 95, 98, ME, XP, Vista).
Access Rights Byte

- Allow control of access to the segment
- Defines how the segment will function

**Diagram:**

- P: System descriptor
- DPL: Descriptor privilege level
- S: Code or data segment descriptor
- E: Descriptor describes code segment
- ED: Segment expands upward (data segment)
- ED: Segment expands downward (stack segment)
- R/W: Data may not be written
- W: Data may be written
- A: Segment not accessed

**Legend:**

- 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.
Segment register contents

- 13-bit selector (MSBs) points to any one of the 8192 descriptors
- TI (table identifier) bit:
  - TI=0: point to global descriptor table
  - TI=1: point to local descriptor table
- RPL (requested privilege level) bits:
  - Highest is 00, lowest is 11 (hi-to-lo: 00, 01, 10, 11)
  - Compared to the DPL (descriptor privilege level) already in place in the access rights byte of the descriptor.
    e.g., if RRL=01 and DPL=10, access is granted (admin vs. power user vs. user vs. guest in Windows XP, for example)
Example (80286)...

In the DS register, we have

DS= 0008h ...or...
DS= 0000 0000 0000 1000

Selects descriptor #1 (there is no zero)

Global table

RPL=0 (highest)
In the descriptor table we have
Top two bytes (00 00) are ignored in the 80286
Access rights byte=92h

1001 0010
P=1: all is valid
DPL=00; you better have high privileges!
S=1: this is code or data (not a system routine)
E=0: compliments S, it is data (not code)
ED=0: it is data not stack data, so increment data pointers upward when applicable
W=1: Data may be written
A=0: The segment is free, go for it!
In the descriptor table we have
Top two bytes (00 00) are ignored in the 80286
Access rights byte=92h
1001 0010
Base=100000h
Start the segment here
Limit=00FFh
End the segment here.
It is only 256KB long
Program Invisible Registers

- Not directly addressable by user.
- Each time the xS register is loaded, a copy of the information is loaded into the cache for easy access during program execution (instead of accessing registers each time - faster)

Notes:
1. The 80286 does not contain FS and GS nor the program-invisible portions of these registers.
2. The 80286 contains a base address that is 24-bits and a limit that is 16-bits.
3. The 80386/80486/Pentium/Pentium Pro contain a base address that is 32-bits and a limit that is 20-bits.
4. The access rights are 8-bits in the 80286 and 12-bits in the 80386/80486/Pentium.
Memory Paging Mechanism

- 80386 and higher only.
- Maps *linear address* (generated by a program) to a *physical address* in memory banks.
  - The code (or *process*) “sees” a linear memory map but in reality, code and data can be scattered.
  - In the overall “system” of memory and hard drive swap space, the contiguous memory is also known as *virtual memory*, although bits and pieces can be scattered in memory or on the hard drive.
- Also permits DOS access to the memory block located between the video BIOS (C8000h) and the system BIOS ROMs (F0000h) via EMM386.EXE.
- Used on both REAL and PROTECTED modes.
- The **page directory** holds 1024 entries of 4 bytes each (there is only 1 page directory in the system).
- Each page directory entry points to one of 1024 page tables.
- Control registers (CP4-CP0) govern the paging mechanism.

If all directory entries are used, this means that $4K \times 1024 = 4MB$ of memory is used for the directory and tables (that’s a lot!)
How is this done?

• Page directory → page table → page offset
• For addresses in the range 00000000h-003FFFFFh (4MB), the first entry of the directory is selected (0000 0000 0011 1111…)
• Next 10 bits select the page table entry.
• The final bits select offset.

Linear address
converted for you (it is virtual, you aren’t aware of what’s happening – neither is the protected mode mechanism)

Directory/Table Entry: don’t worry about the details here
Example

- EMM386.EXE repaging of memory for DOS usage:
  - Here, page directory has four entries (each points to 4MB of physical memory)
  - 4 page tables with 1024 entries each (each pages 4KB of memory)