

Module 3: Memory Management

Memory Management

Memory may be used for code & data (applications & system), system internals (eg disk buffer), dedicated (eg Video RAM).

- Management of Memory
 - Memory usage in Windows & Unix
 - Protection/Sharing
 - Relocation
- Virtual Memory
 - Rationale
 - Paging
 - Segmentation

Some specific case studies also.

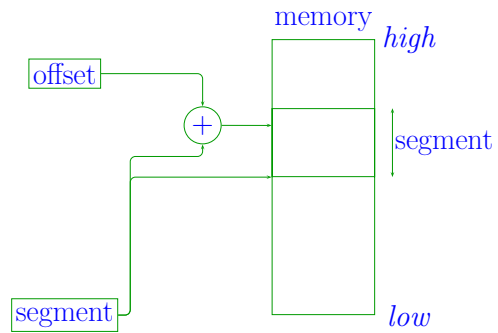
Memory Usage in Windows: DLL's

- Common code “libraries”:
 - loaded when needed (on demand)
 - shared between processes
- DLL = Dynamic Link Library
- Look under `Windows\system32` for *.dll
- Examples:
 - `comctl32.dll` for Progress Bar, Calendar Control
 - `setupapi.dll` for USB device access
- Unix: `.so` (shared object)

Memory Allocation Example: malloc/free

- `malloc()` - memory allocate, contiguous memory
- Request number of bytes, returns pointer to block or NULL
- `free()` - returns memory to OS pool
- Run performance monitor in task manager: CTRL-ALT-DEL, Performance tab
- Run `malloc.exe`
- Memory allocated > physical RAM
- Note spikes in CPU graph due to sampling
- Note dips in memory usage graph

Basic Concepts – Relocation



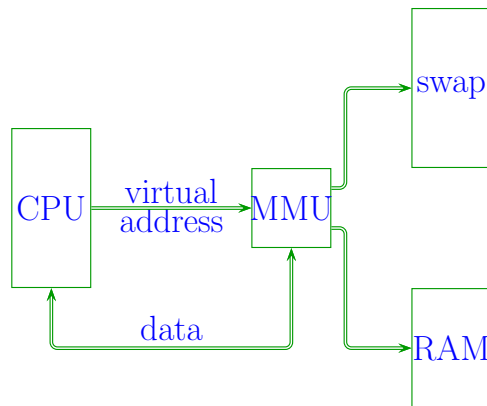
5

Basic Concepts – Relocation

- Each task has a *relocation address* or starting point.
- Physical Memory = Sum of all task's memory.

6

Basic Concepts



7

Rationale

- Allows programs whose size is greater than RAM size to be executed.
- Cost effective (disk cost per gigabyte \ll RAM cost).
- Reduces performance \rightarrow design problem.

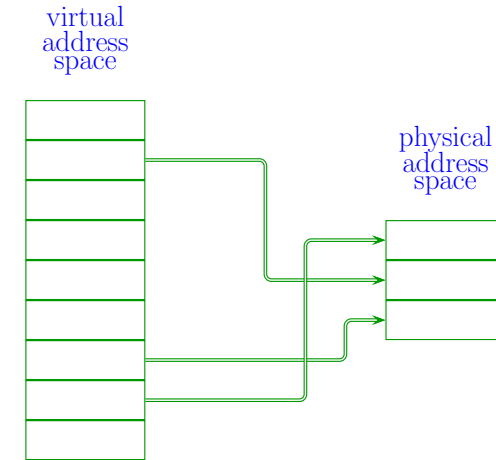
8

Operation

1. The CPU requests a read (or write) of data from (to) memory in the usual manner, by sending an address out on the address bus.
2. The MMU checks the address requested. If it happens to be in RAM, the data is returned and hence accessed quickly.
3. If the address requested is not in RAM it is guaranteed to be on disk. The MMU generates a memory or page fault interrupt, and the operating system must run a special interrupt handler to read a portion of the swap area into RAM.

9

Paging



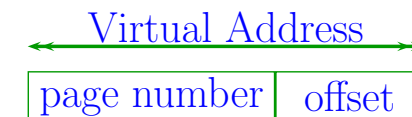
10

Paging

- Virtual Address Space much larger than Physical Address Space.
- Inefficient to “swap” entire process to disk when memory runs low, so divide segment into pages (page frames).
- CPU only ‘sees’ virtual addresses.
- MMU performs Virtual → Physical translation.
- Translation Lookaside Buffer (TLB) caches recently-used virtual-to-physical translations.
- Pentium is more complex: has a page directory table plus multiple page tables.

11

Paging



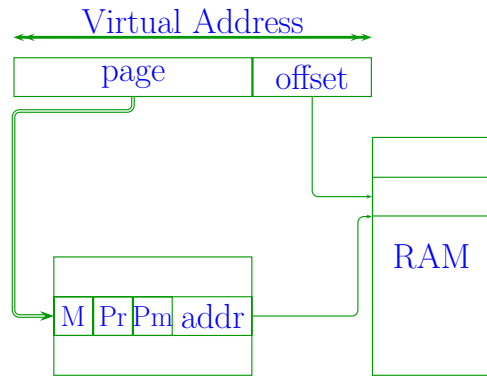
1		325
2		463
3		006
4		065

physical page number RAM virtual page space

Typical page size = 4 kbytes (12 bits for offset field).

12

Paging



Page Table composed of Page Table Entries (PTE's).

13

Paging Example

Suppose the Virtual Address space = 64G, the page size = 4kbytes. Physical RAM is 4G.

- Virtual Address = 64G \rightarrow 36 bits.
- Page = 4k \rightarrow 12 bits for displacement.
- Page Number occupies 36-12 = 24 bits.
- Page Table needs $2^{24} = 16\text{M}$ entries.
- Physical RAM = 4G \rightarrow 32 bit address.

14

Paging Example

Virtual Address space = 64G, the page size = 4kbytes. Physical RAM = 4G.
Then we have:

- There are $\frac{2^{32}}{2^{12}} = 2^{20} = 1\text{M}$ pages in RAM.
- Require 20 bits to reference RAM page.
- Each PTE needs 20 bits + P bit + M bit + permission bits (say, 32 bits)
- Page table is therefore 16M entries \times 4 bytes = 64M.

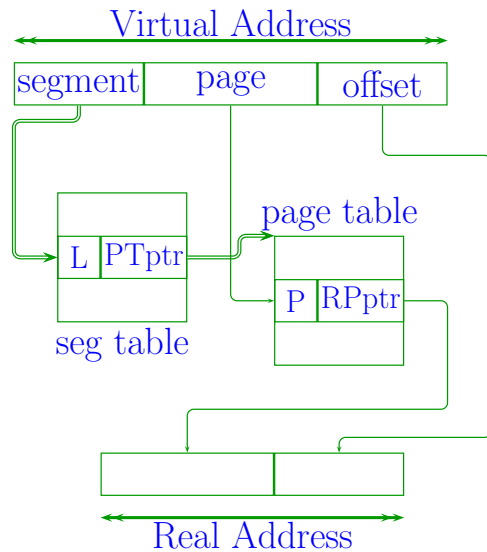
15

Paging Question

- This is quite a large page table.
- Suppose the pages were increased to 4M.
- What would the result be?
- Would this be a wise design move?

16

Segmentation



Term “swap” (Module 1) comes from swapping entire process/segment to disk if memory low.

17

Segmentation

- PTptr = page table pointer
- RPptr = real page pointer
- Segments of variable size.
- Allows swapping of page table.
- L = length (length+base of segment)
- Pentium has segments stored in “descriptor tables”. These contain segment base address, segment limit, and access rights.
- Pentium has two-level paging: one page directory plus multiple page tables.

18

Pentium Addressing – Real Mode

S_3	S_2	S_1	S_0	segment ← 4
	O_3	O_2	O_1	O_0 16-bit offset
R_4	R_3	R_2	R_1	R_0 20-bit result

Segment (16 bits) is shifted left 4 bits and added to the 16-bit offset.
Effective address is 20 bits (1 M).

Example: `mov ax, es:[si]`

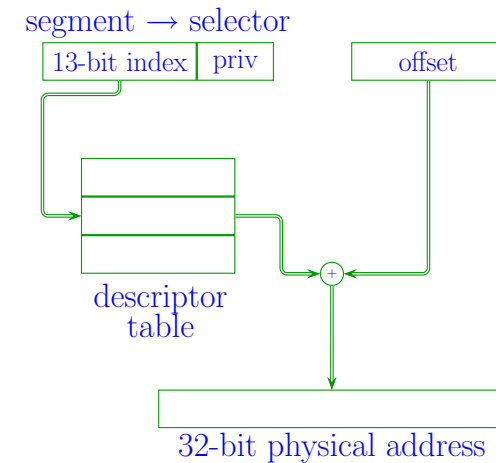
es (extra segment) is the segment register

si (source index) is the offset register

ax (accumulator) is the 16-bit destination register.

19

Pentium Addressing – Protected Mode



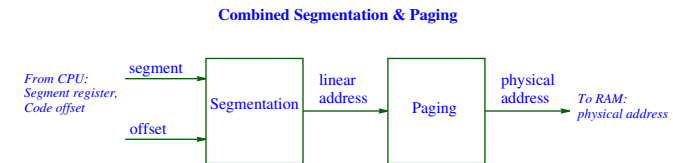
20

Pentium Addressing – Protected Mode

- Same instructions, however now ‘segment’ registers are called ‘selector’ registers .
- They are an index into a ‘descriptor table’ which describes each segment.
- 8 bytes hold Base Address, Limit Address, Privilege Level, Segment Present, Segment Accessed, Code or Data, Readable or Writable.
- Allows for relocation.

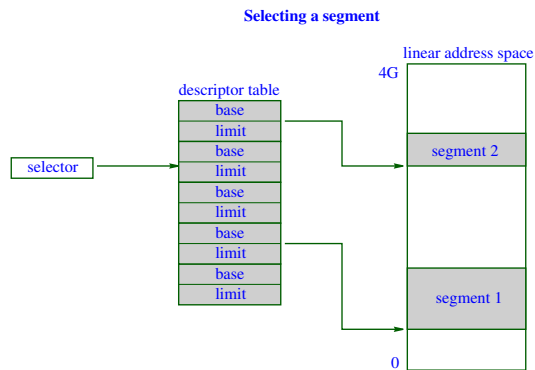
21

Pentium: Segmentation+Paging



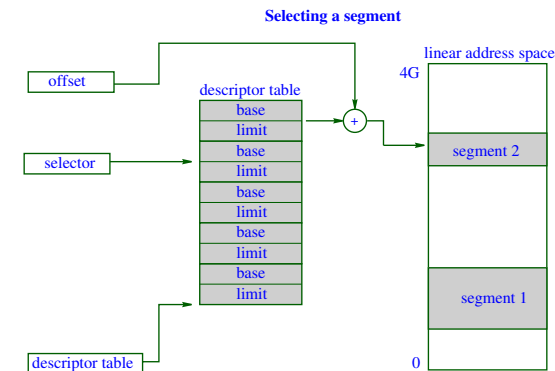
22

Pentium: Segmentation



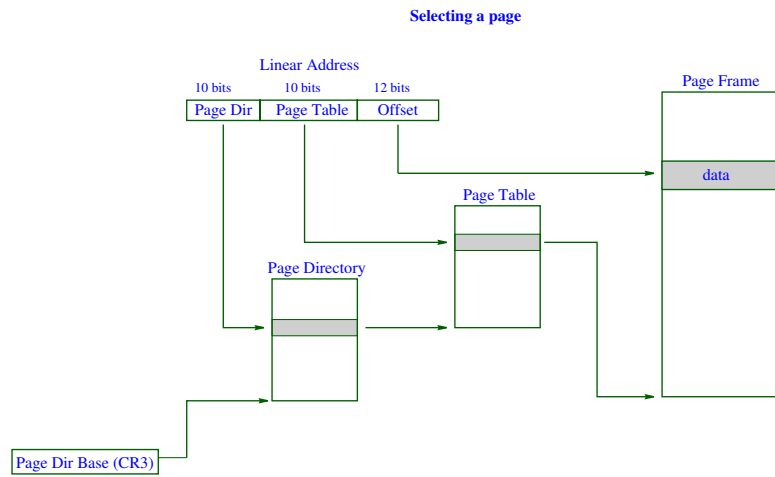
23

Pentium: Segmentation



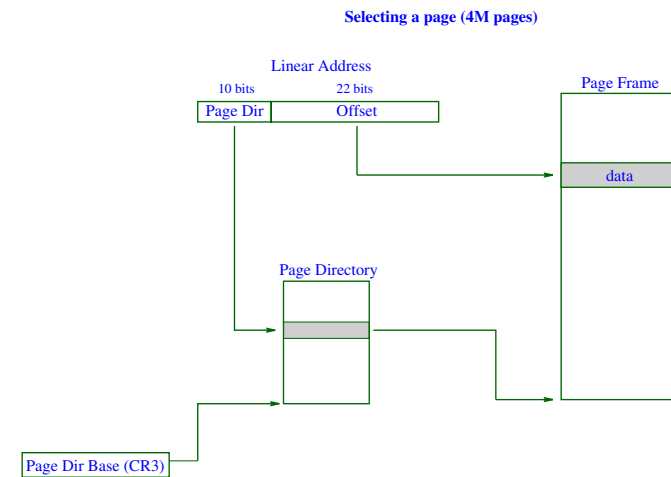
24

Pentium: Paging (2-level, 4k pages)



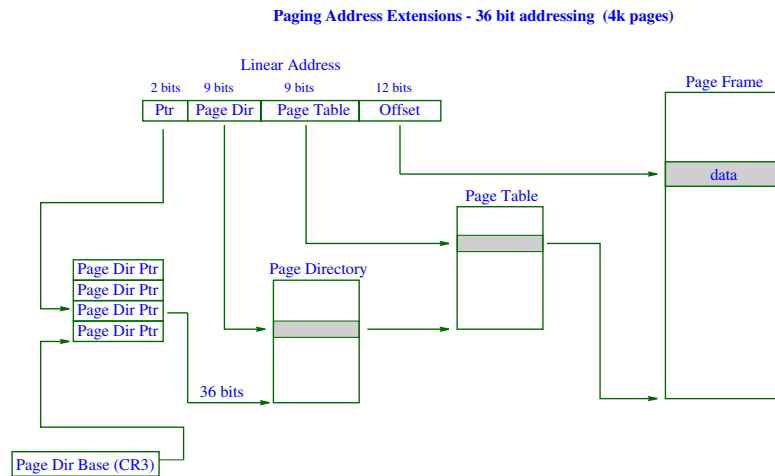
25

Pentium: Paging (4M pages)



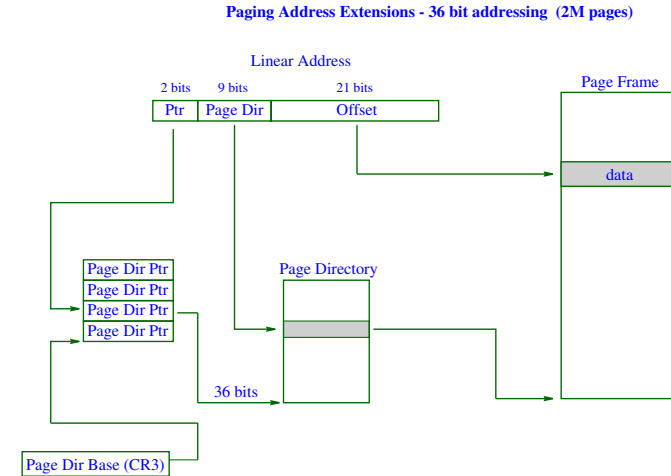
26

Pentium: 36 bit Page Address Extensions (4k pages)



27

Pentium: 36 bit Page Address Extensions (2M pages)



28

System Properties under Windows

- Command shell (Start-run, type “cmd”)
- Change to system area – either `cd c:\windows\system32` or `cd c:\WINNT\system32`
- Type `winmsd` – diagnostics
- Type `perfmon` – performance monitor, click “+” to add things like cache, memory, disk, network
- Type `control` – control panel. Click System-Advanced-Performance Options, examine virtual memory settings
- Virtual memory: `attrib c:\pagefile.sys`

Module Summary – Important Points

1. Memory use in practice
2. Virtual memory, paging, segmentation
3. Overview of recent developments