Memory Mapping in 64 Bit Mode

Ray Seyfarth

July 30, 2011

Outline

1. Memory mapping basics
2. Page map level 4
3. Page directory pointer table
4. Page directory table
5. Page table
6. Large pages
7. CPU support for fast lookups

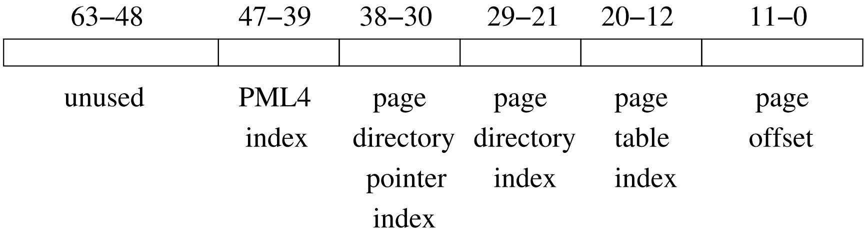
Memory mapping register

- Named “Control Register 3” or `CR3`
- Contains the physical address of the top-level of the memory mapping tables
- There are 4 levels in a hierarchy of tables for memory mapping
- Memory mapping tables are setup and then CR3 is set to the address of top table.
- The top table is called “Page Map Level 4” or `PML4`

Memory mapping pages and tables

- Each page is $2^{12} = 4096$ bytes
- An address is 8 bytes
- Each page can hold $2^9 = 512$ addresses
- A 9 bit field is needed to index the mapping tables
- In general if pages are size 2^k, then a page can hold $k - 3$ pointers
  - I think $2^{15} = 32768$ would be nice for a page size
  - We could use 12 bits for page table indices
  - 3 levels for 36 bits would be enough page hierarchy levels
  - Total memory limit would $2^{51} = 2,251,799,813,685,248$
  - 2 Petabytes of RAM might be enough for most personal computers
- Current mapping uses 48 bits, we we are limited to 2^{48} bytes which is about 262 Terabytes

Logical memory address fields



63–48	47–39	38–30	29–21	20–12	11–0
unused	PML4 index	page directory pointer index	page directory index	page table index	page offset

- Bits 47-39 are used to index the PML4 table
- Bits 38-30 are used to index the selected page directory pointer table
- Bits 29-21 are used to index the selected page directory table
- Bits 20-12 are used to index the selected page table
- Bits 11-0 are the offset into the page (for 4 KB pages)

Page map level 4

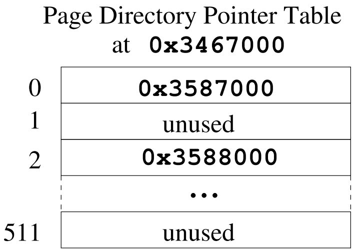
- Assume CR3 has the physical address 0x4ffff000

PML4 at **0x4ffff000**

0	**0x3466000**
1	**0x3467000**
2	**0x3468000**
	...
511	unused

- Let's translate logical address 0x80801fffa8
- Bits 47-39 = 1, so we use the second entry
- The page directory pointer table we need is at 0x3467000

Page directory pointer table



Page Directory Pointer Table
at **0x3467000**

0	**0x3587000**
1	unused
2	**0x3588000**
	...
511	unused

- We're translating logical address `0x80801fffa8`
- Bits 38-30 = 2, so we use the third entry
- The page directory table we need is at `0x3588000`

Page directory table

Page Directory Table
at **`0x3588000`**

0	**`0x3678000`**
1	**`0x3579000`**
2	unused
	…
511	unused

- We're translating logical address `0x80801fffa8`
- Bits 29-21 = 0, so we use the first entry
- The page table we need is at `0x3678000`

Page table

Page Table
at **0x3678000**

0	**0x5788000**
1	**0x5789000**
2	**0x578a000**
	...
511	**0x5799000**

- We're translating logical address `0x80801fffa8`
- Bits 20-12 = 511, so we use the last entry
- The page we need is at `0x5799000`
- So logical address `0x80801fffa8` is at physical address `0x5799fa8`

Large pages

- Using the first 3 existing levels of page tables, we can have large pages with $2^{21} = 2097152$ bytes
- This is used by Linux for the kernel
- By allocating some memory for large pages, user programs can use large pages
- By using 1 byte per page on 8 GB of data, I managed to have more need for cache for page table entries than my computer had
- Performance was significantly slower
- Large database memory regions would benefit greatly from using 2 MB pages

CPU support for fast lookups

- A CPU uses a special cache called a "Translation Lookaside Buffer" or TLB to speed up memory translation
- A TLB operates much like a hash table
- Presented with a logical address, it produces the physical address or failure in about 1/2 a clock cycle
- The Intel Core i7 has 2 levels of TLBs
    - Level 1 holds 64 small page translations (or 32 big pages)
    - Level 2 holds 512 page translations
    - Large programs with small pages will experience TLB misses which can be satisfied fairly rapidly with normal cache
    - Very large programs can crawl