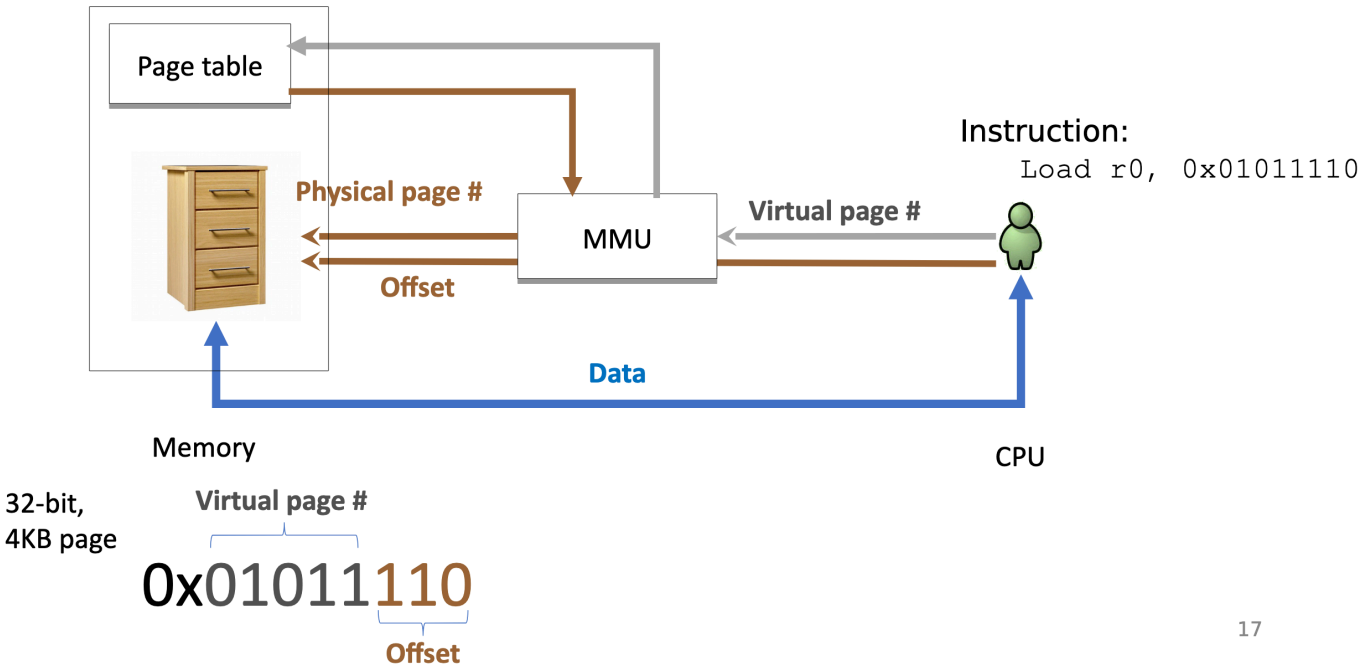
NOTES: Virtual Memory

notes/virtual-memory.md

Virtual Memory

- Virtual Memory is an abstraction/interface that allows implementation to be ported to software with different hardware specifications
- Programs reference virtual memory addresses, which are not necessarily where they are in the hardware



Paging

- Each virtual memory address (that a program addresses) needs to be translated into the physical memory address (in hardware)
  - This is done by the **Memory Management Unit** (MMU)
- However, since an address is 64 bits on 64 bit architectures, the translation table would be larger than the memory itself!
- So, we use **paging** to break the address into smaller chunks
- Pages are often 4KB in size

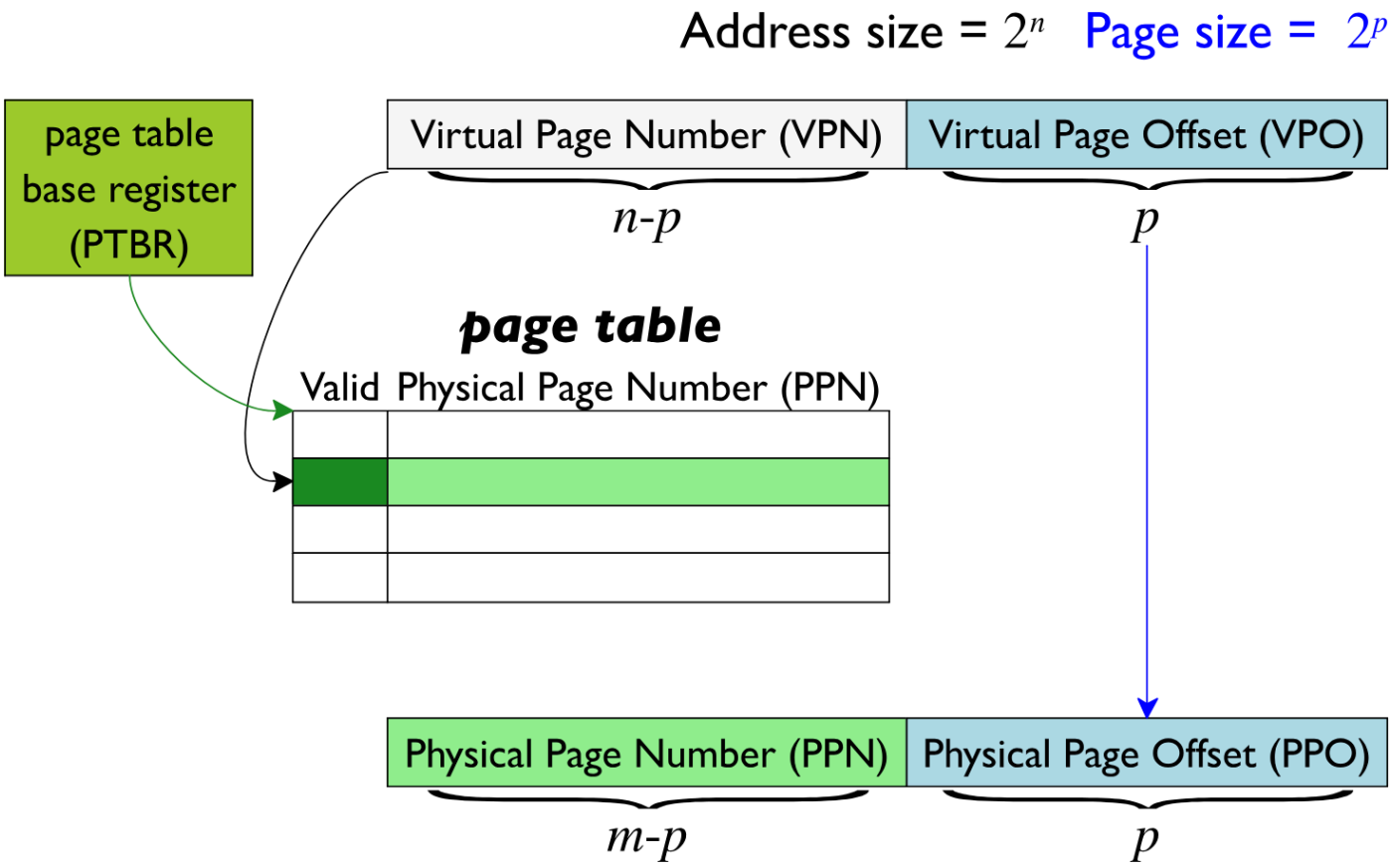
Page Numbers

- Page number is split into two parts: **virtual page number** and **page offset**
- The virtual page number is used to index into the page table, which gives the physical page number (first 5 hex digits of an address)

- The page offset is used to find the exact byte in the physical page (last 3 hex digits of an address)
- A **Page Table** is used to translate virtual page numbers to physical page numbers
  - The page table is pointed to by the **Page Table Base Register (PTBR)** (kind of like the stack base pointer)

Differing Virtual and Physical Memory Sizes

- Virtual memory can be larger than physical memory, or vice versa
- The page table has a **valid bit** to indicate whether the page is in physical memory or not



"Lazy" Memory Allocation

- When you call `malloc`, a virtual page is created, but no physical page is allocated
- When you use the memory for the first time, a page fault occurs, and the OS allocates a physical page

Moving Data to Disk (Page Swapping)

- If the working set (# of pages) for the physical memory to store, the OS will move some pages to disk

- The oldest page is marked as invalid in the page table; when a page fault is triggered, it will be reinstated from memory

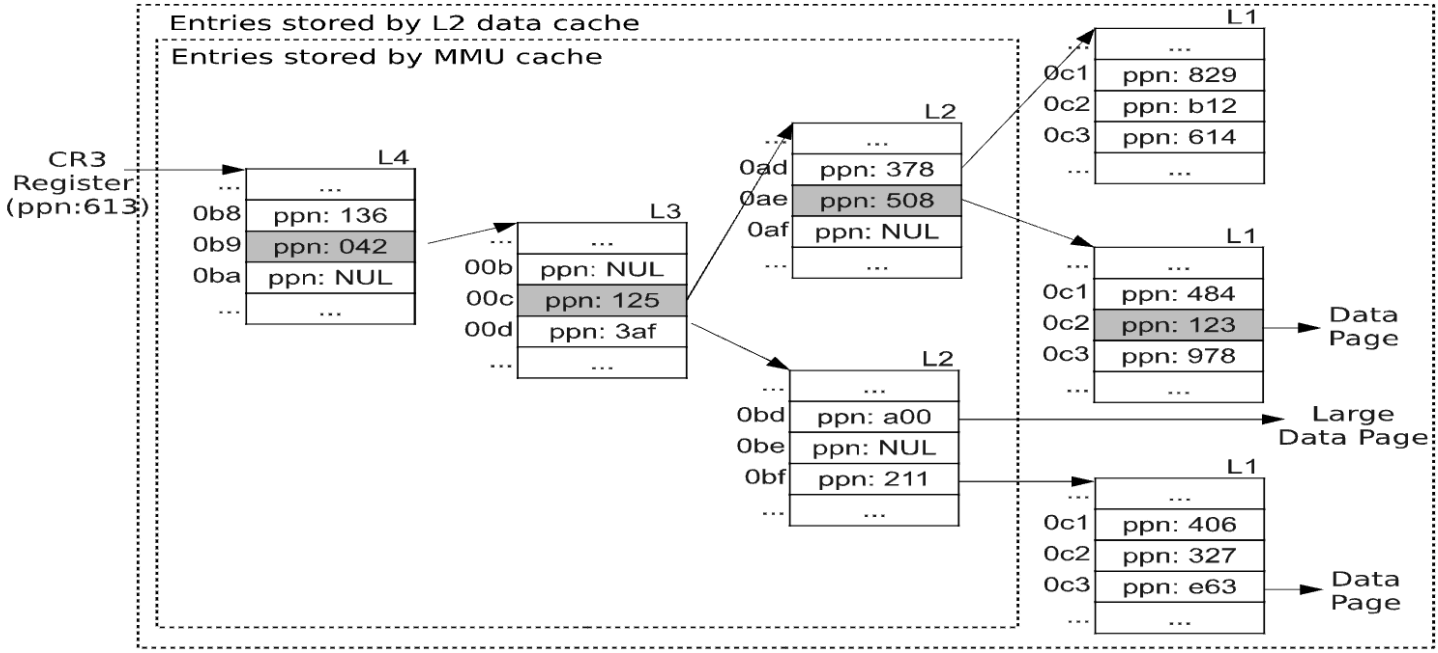
Optimizations

Translation Lookaside Buffer (TLB)

- The TLB is a cache for page table entries that reduce number of memory accesses
- The TLB comes at a cost, because you have to check the TLB before checking the page table, but it overall is an improvement

Multi-Level Page Table

- Instead of a single page table, page tables can be stored in a tree structure
- This means you don't have to store the entire physical address memory value in the page table, since a few bits of the physical memory address is stored at each level of the tree
- If an entire layer of page table is empty, it is not stored in memory, reducing the size of the total page table structure
- Additionally, the MMU cache stores the highest levels of the tree, reducing the number of memory accesses



Interaction with Processes

- The kernel-space page table is system-wide, while the user-space page table is per process
- User page tables have to be stored in kernel space to prevent user from overwriting page table

- Number of processes on a machine is limited by memory, since each process needs its own page table

Permissions

- In addition to the valid bit, there is a **read/write permission bit** and a **executable bit** that specify whether a program can read/modify/execute a page
- Using the page table to enforce permissions is a design philosophy that speeds up programs
- The downside of this is that the entire page must have the same permissions, causing **inter-page fragmentation**
- If a disallowed operation occurs, the system sends a **SIGSEGV** signal (segfault) to the process

Sharing Memory

- If two processes need the same memory (for example, a Dynamically Linked Library), their page tables can point to the same physical memory

Sharing Memory upon `fork`

- When `fork` is called, memory is not immediately copied
- There is a bit called **copy-on-write** (COW) in the page table
- When a process tries to write to a page with copy-on-write set to 1, a page fault is triggered, and the OS copies the page

Implementation in Linux

- In Linux, all threads of the same process share the same memory struct
- To differentiate between stacks, the memory struct has an `mmap` that stores each Virtual Memory Area
    - VMAs are stored in a red-black tree!