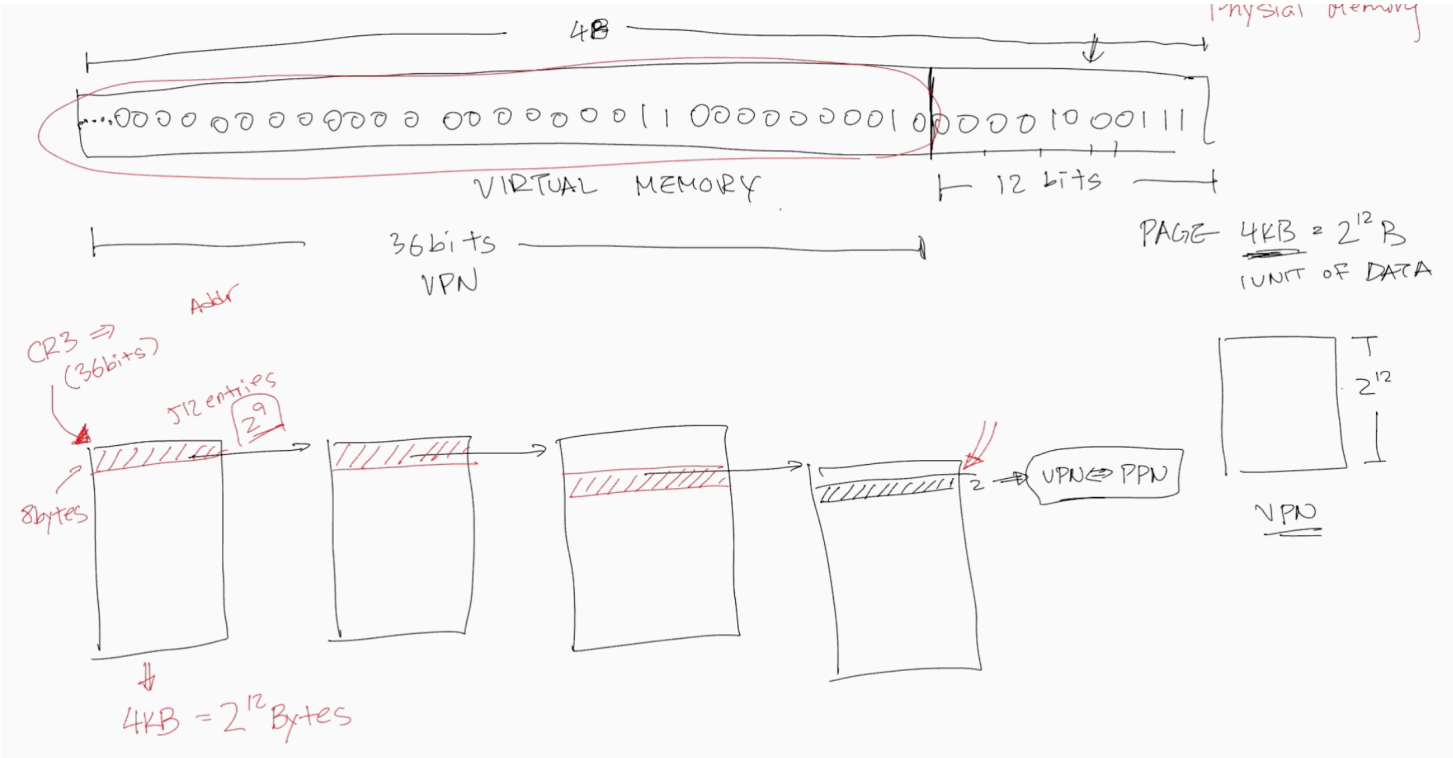
NOTES:
Intel Implementation of Virtual Memory & Cache

notes/intel-vmem-cache.md

Intel Implementation of Virtual Memory & Cache

Page Tables

- PTBR is register *CR3*
  - CR3 only stores the top 36 bits. It's guaranteed that a page table is 64KB so the the bottom 12 bits are always 0 because of the alignment
- Four levels of tree-structured page tables
  - Every time you descend a level, you must go to cache (and possibly memory) - so fetching an instruction can take up to 5 memory accesses
    - But in the case of something like `addq %rax, (%rbx)`, on the write back, `rbx` is guaranteed to be in cache so worst case 10 memory accesses
- Address is 48 bits long
  - 36 bits for Virtual Page Number
    - Virtual Page Number is split into 4 parts, one for each level of the page table
    - 9 bits for each level
      - Each page table entry is 8 bytes (for flags, address of next page table, etc)
        - 12 bottom bits for permissions, next 36 for physical address of next page table, top 16 are zeroes
      - Therefore, there are $4096 \div 8 = 2^9 = 5124096 \div 8 = 29 = 512$ entries per level
      - Therefore, we need 9 bits for each level
  - 12 bits for Page Offset (since page is $4KB = 2^{12}4\text{KB} = 212$ bytes)



Figuring out page table structure and sizing

1. Find how many bits you need for the page offset.
    - Log_2 the size of the page.
2. Find how many bits in the virtual address you need to index each page table level.
    - Divide the page size by 8 bytes (size of a page table entry), and Log_2 that.
3. Find how many levels there are (if you're given the # of bits actively used in each virtual address)
    - Subtract the # bits in page offset from the total # of bits. Divide by the # of bits you need to index each page table level.

Parameter	Description
Fundamental parameters	
$S = 2^s$	Number of sets
E	Number of lines per set
$B = 2^b$	Block size (bytes)
$m = \log_2(M)$	Number of physical (main memory) address bits
Derived quantities	
$M = 2^m$	Maximum number of unique memory addresses
$s = \log_2(S)$	Number of *set index bits*
$b = \log_2(B)$	Number of *block offset bits*
$t = m - (s + b)$	Number of *tag bits*
$C = B \times E \times S$	Cache size (bytes), not including overhead such as the valid and tag bits

Note: Large Pages

In some Linux systems, there are 2MB and 1GB pages—2^{21}221 and 2^{30}230 bytes respectively.

The size goes up by a factor of $2^9$29 each time, since an extra 9 bits are used for the page offset instead of the virtual page number.

For example, for the 2MB large page, the bottom 21 bits are the offset, and the top 27 bits are the virtual page number. The 27 bits are split into 3 parts for each of the 3 used levels of the page table.

TLB (Translation Lookaside Buffer)

- TLB takes a virtual address/virtual page number, and outputs a physical page number
- Looks something like this

- Last used is for LRU eviction policy, and is usually a number that increments every time a cache line is created; lowest number is evicted
- Data is the physical page number. Every virtual page number will have a unique cache line, so no need to store VPN

```
valid | tag | metadata | last used | data (physical page number)
```