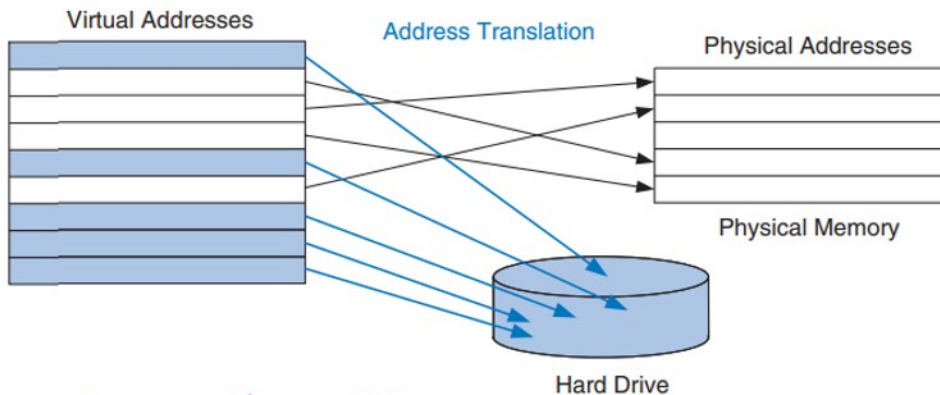
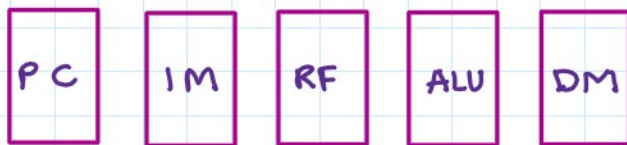


Figure 6.31 Example RISC-V memory map

RISC-V CPU



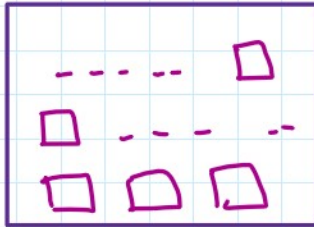
→ gives the illusion of more memory

→ each process gets its own VAS

CPU-RAM Interaction

Once an executable is run, **Virtual Address space** is created in RAM (**Physical memory**) → main memory

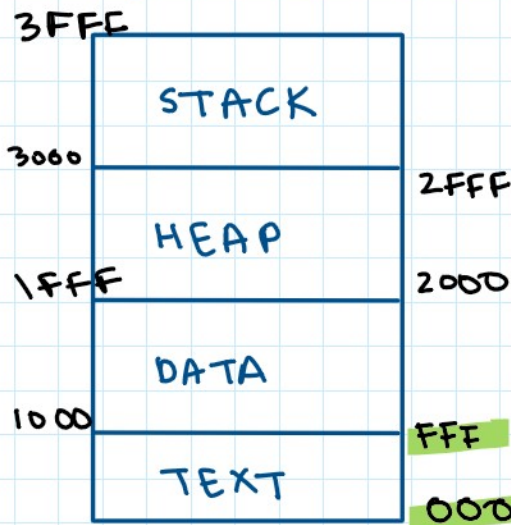
Virtual Address space comprises of Virtual pages -



2^{15} pages in RAM

Because the page size is $4 \text{ KiB} = 2^{12}$ bytes, there are $2^{31}/2^{12} = 2^{19}$ virtual pages and $2^{27}/2^{12} = 2^{15}$ physical pages. Thus, the virtual and physical page numbers are 19 and 15 bits, respectively. Physical memory can only

So, a VAS looks like



ending virtual address

starting virtual address of text page

→ Programs can access data anywhere in virtual memory, so they must use **virtual addresses** to specify location in virtual memory

RISC-V assembly code works with these virtual addresses

`lw t0, -4(sp)`

$-4 + 3FFF$

→ Virtual Address

`# sp = 3FFF`



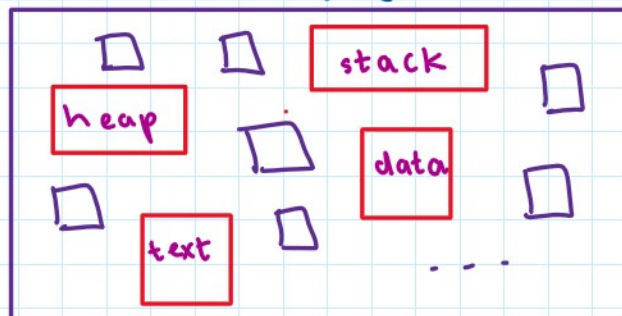
Virtual Address

We can extract virtual page number from a virtual address by picking 4th digit onwards from the LSB.

In the example above, virtual page number is 3.

Likewise, VPN for Heap is 2, VPN for data is 1, and VPN for Text/code is 0.

Note that Virtual Addresses are 'congruent' i.e. 000 to FFF, 1000 to 1FFF, 2000 to 2FFF, and 3000 to 3FFF. However, in actuality, these four pages might be scattered around in RAM, anywhere in the 2¹⁵ pages, as shown below:



Question: How do we know which real physical pages are allocated to the 4 virtual pages?

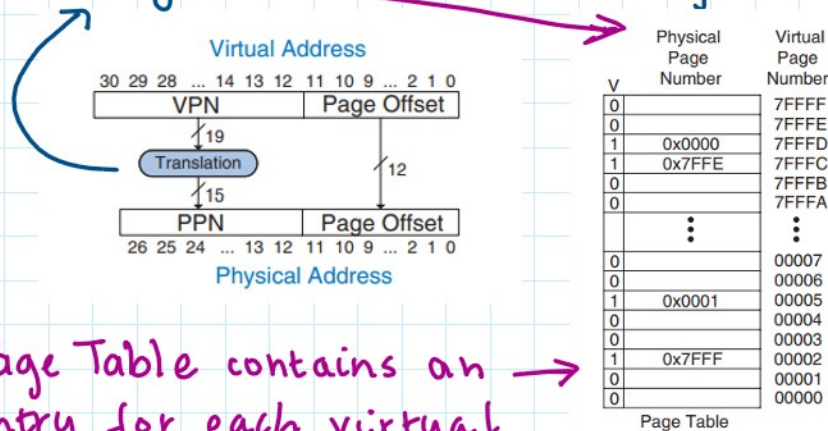
Answer: Lookup Tables

In RISC-V, lookup tables are used for virtual-to-physical translation. Primarily through page tables and TLB

Address Translation

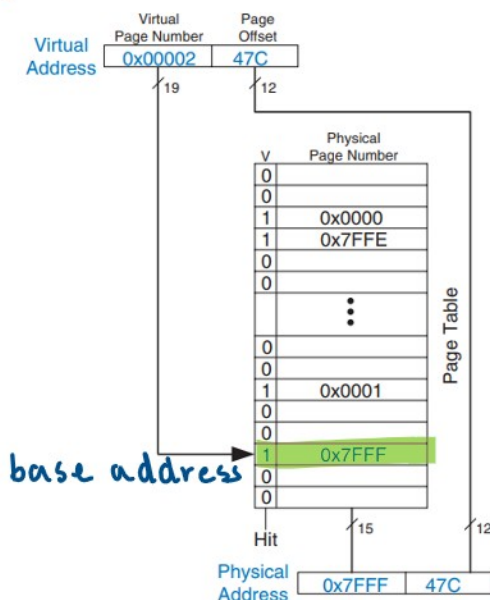
- 1) The MSB of virtual/physical address specify VPN or PPN
- 2) The LSB specify **page offset**
 ↳ word within the page

Page Tables are used for translation



Page Table contains an entry for each virtual page.

Figure 8.23 The page table for



→ Page offset remains the same

Multi-level Page Tables

To conserve physical memory page tables can be broken up into multiple levels. (usually 2).

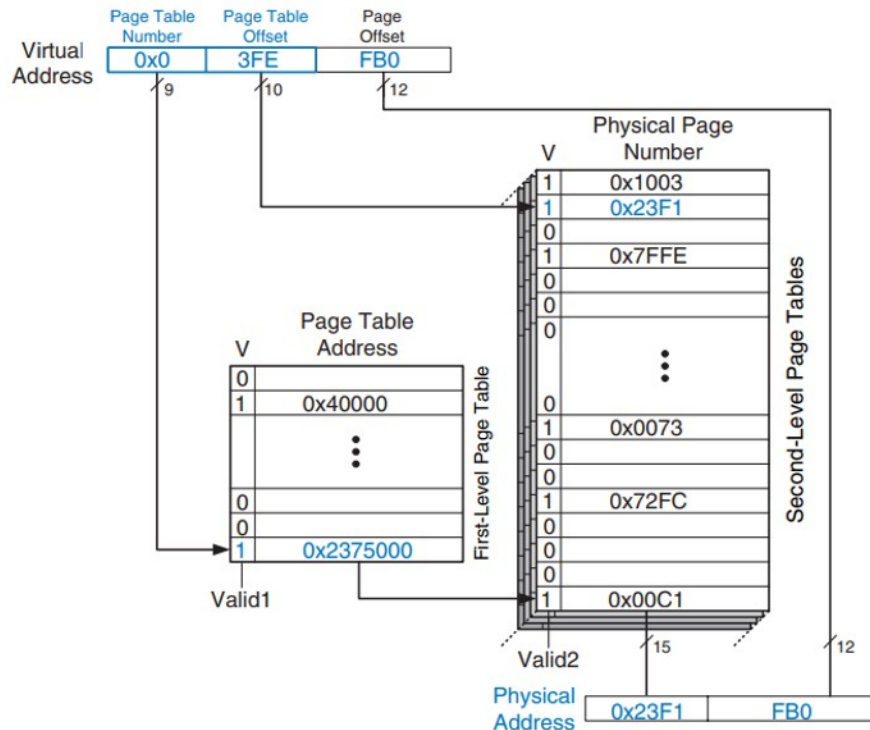


Figure 8.27 Address translation using a two-level page table

- First-level page table always kept in physical memory.
- It points to small second level page tables stored in virtual memory.
- The small second level PTs contain actual translations.

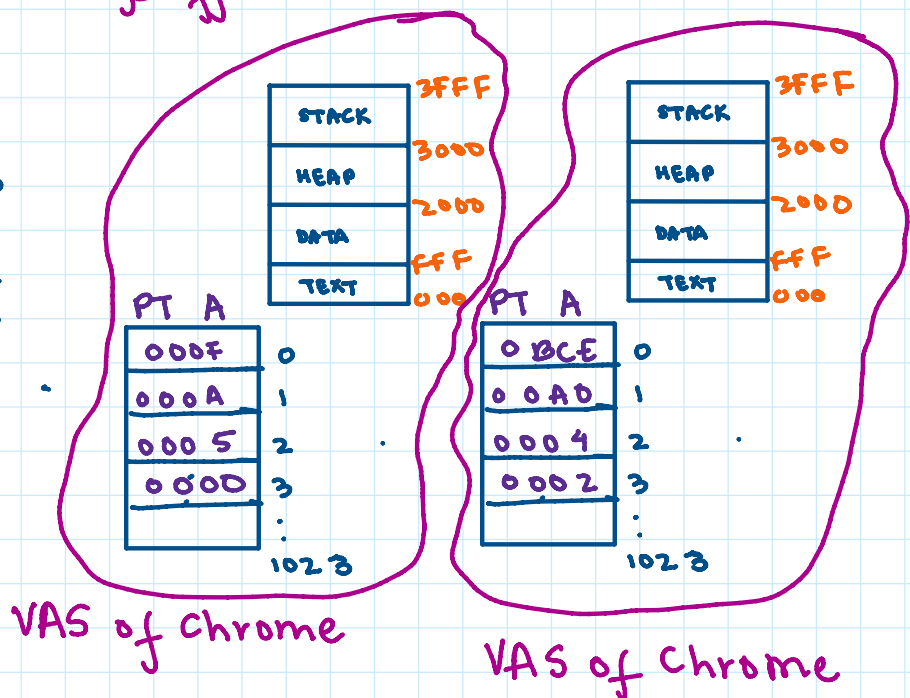
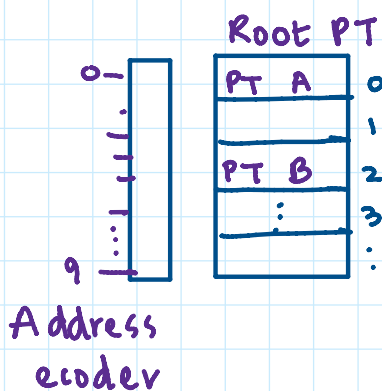
Page Table Number indexes the first-level PT (root PT)

Page Table Offset indexes the 2nd level PT

* Remaining 12 bits are page offset for a page of size 4KiB
 $2^{12} = 4 \text{ KiB}$

Imagine there are 2 executables ^{chrome { firefox} running. As a result, 2 virtual address spaces are created.

Virtual Address



Mapping

VPN \longrightarrow PPN
 00000 \longrightarrow 000F Text segment of VAS chrome
 00001 \longrightarrow 000A Data segment of VAS chrome
 00002 \longrightarrow 0005 Heap segment of VAS chrome
 00003 \longrightarrow 0000 stack segment of VAS chrome

00002

0000 - heap segment of VAS chrome

00003 → 0000 stack segment of VAS chrome

00400 → 0BCE Text segment of VAS firefox

00401 → 00A0 data segment of VAS firefox

00402 → 0004 Heap segment of VAS firefox

00403 → 0002 stack segment of VAS firefox

So different physical pages are mapped to VAS's of chrome & firefox.

