Virtual Memory

Originally, computers addressed memory directly.

Data was loaded from disk storage into some location of RAM, and the address of the first byte of that location was used thereafter to reference the data.

Eventually, data addresses became virtual, meaning that they were arbitrary locations unrelated to physical RAM.

How, then, are virtual addresses used to read and write data?

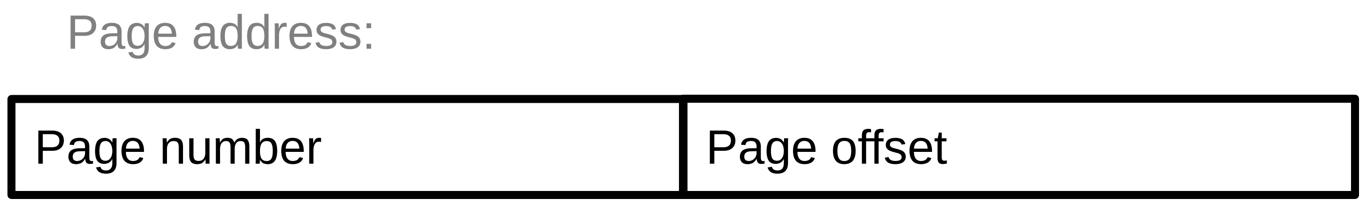
Instead of accessing data one byte at a time, data would be accessed one page at a time.

The size of a page of data would usually be 4 or 8 KB, or sometimes 4MB.

Whatever page size is chosen for a system becomes the universal size for all pages on that system. In other words, all pages ALWAYS have the same size on a given system.

This is true regardless of whether a page is virtual or physical.

The address of data is then decomposed into two parts,
first the page number, then the page offset:

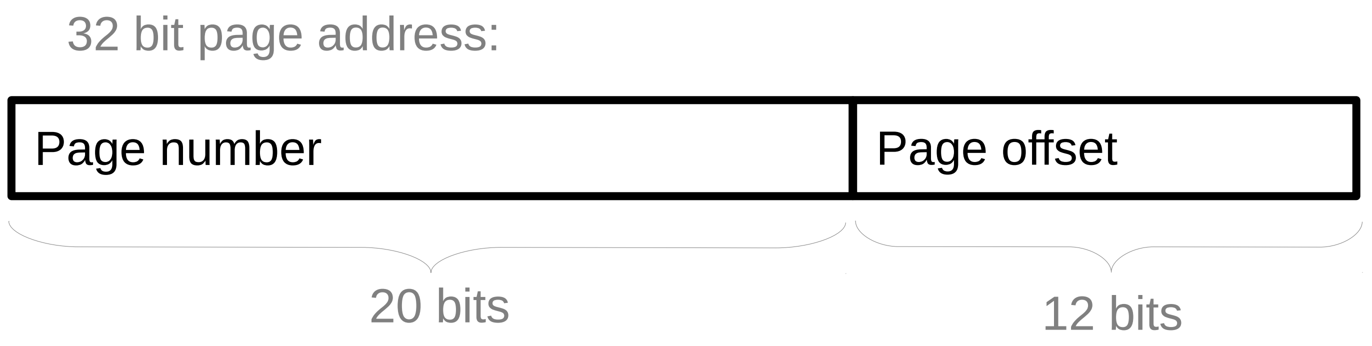


The page offset refers to the location of the data within the page.

Assuming a page size of 4 KB (4096 bytes), the page offset would be composed of 12 bits (2^12 = 4096).

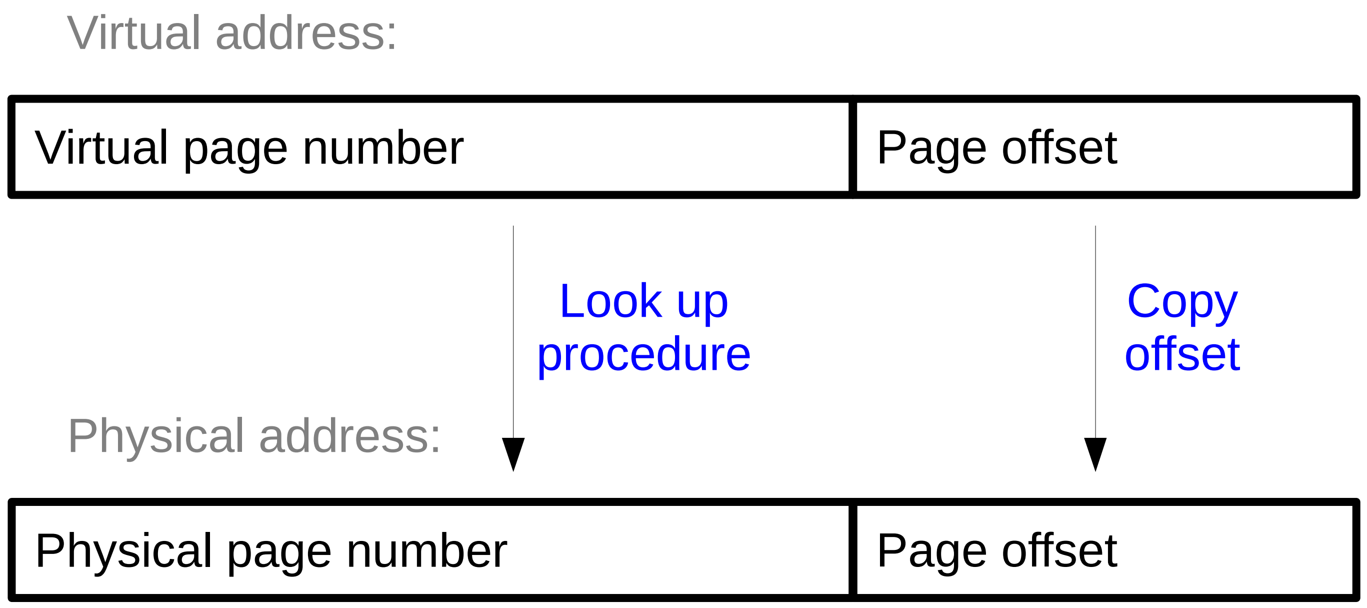
The offset value specifies where the data is located within the page.

In a 32-bit system therefore, the page number would be composed of 20 bits (32 total bits – 12 offset bits).



This division of an address into a page number and page offset applies to both virtual addresses and physical addresses.

Continuing with this example, a virtual address can be mapped to a physical address:



Note that page offset portion of the address is the same in both the virtual address and the physical address. Hence, the page offset from the virtual address is merely copied to the physical address.

Virtual page numbers are created in a few different ways.

For example, the location of data on disk could be interpreted as a virtual address.

More importantly though, virtual page numbers are created by the assignment of addresses to instructions and data in executable files. The compiler assigns these addresses.

Based on these executable file addresses, the system assigns a virtual address to the heap. Consequently, all dynamically allocated memory uses virtual addresses.

In order to correlate virtual pages with physical pages in RAM, a data structure called a page table is created.

The page table contains one entry for each possible virtual page number.

This entry contains the corresponding physical page number, which is simply the address in RAM where the physical page begins.

Example page table (simplified):

Virtual page number	Valid bit	Physical page number
PTE 0	1	0x237f
PTE 1	1	0x3438
. . .	0	
	0	
	0	
	0	
	0	
PTE 7	0	

PTE = Page Table Entry

In order to quickly look up the corresponding physical page number for a virtual page number, a device known as the Memory Management Unit (MMU) was added to the CPU.

Each process has its own page table in RAM; the MMU is provided with the address of the page table through a register that is part of the process context.

Generally then, the MMU translates the virtual address into a physical address and then accesses the data as follows:

1) The ALU issues an instruction to access data located at a certain virtual address.

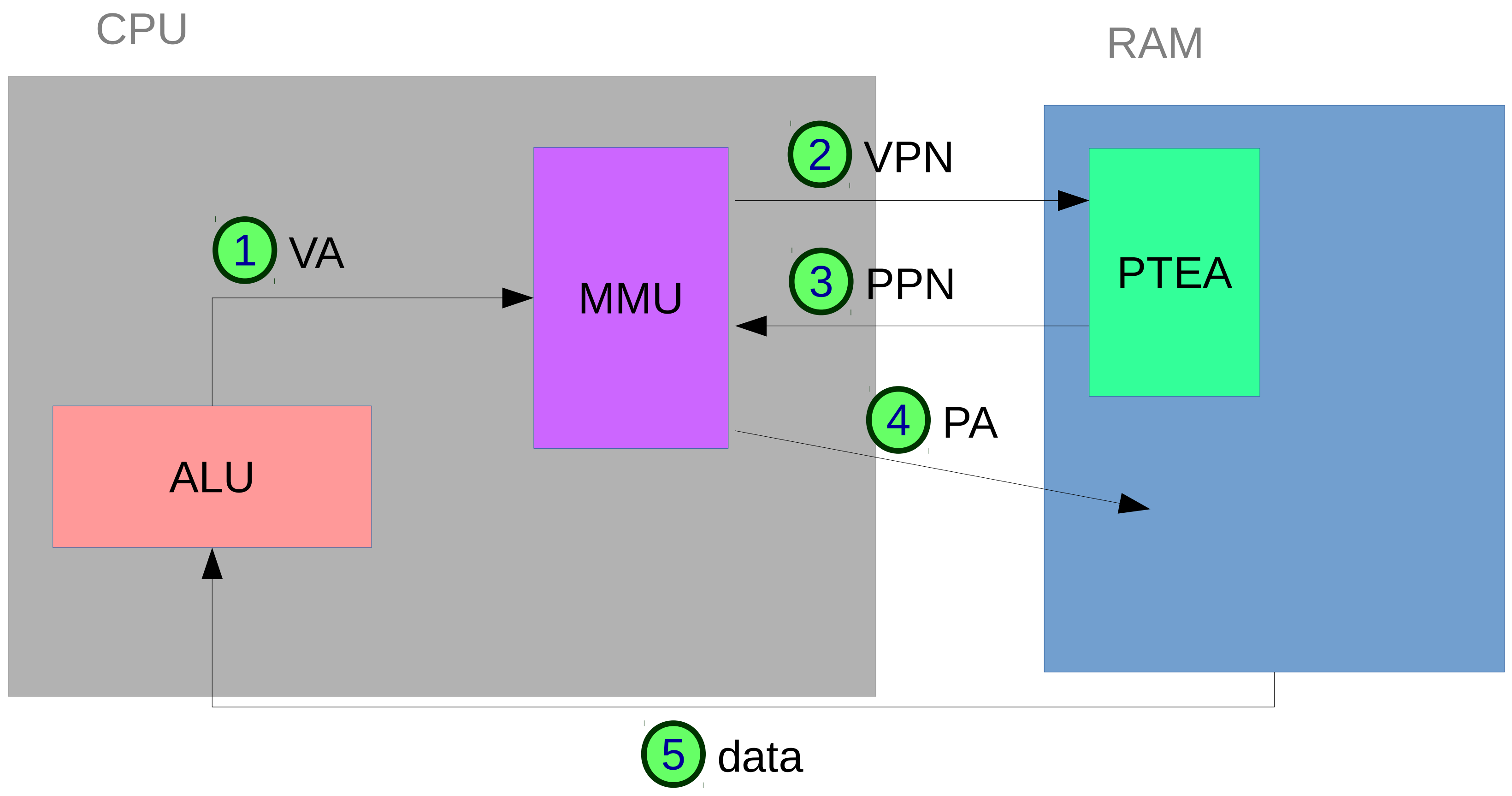
2) The MMU extracts the virtual page number from the address and uses it to find an entry in the page table.

3) The MMU retrieves the physical page number from the page table entry.

4) The MMU composes the physical address from the physical page number and the offset. It then issues a read request for the data in RAM.

5) The data is sent to the ALU.

This process is represented graphically as follows:

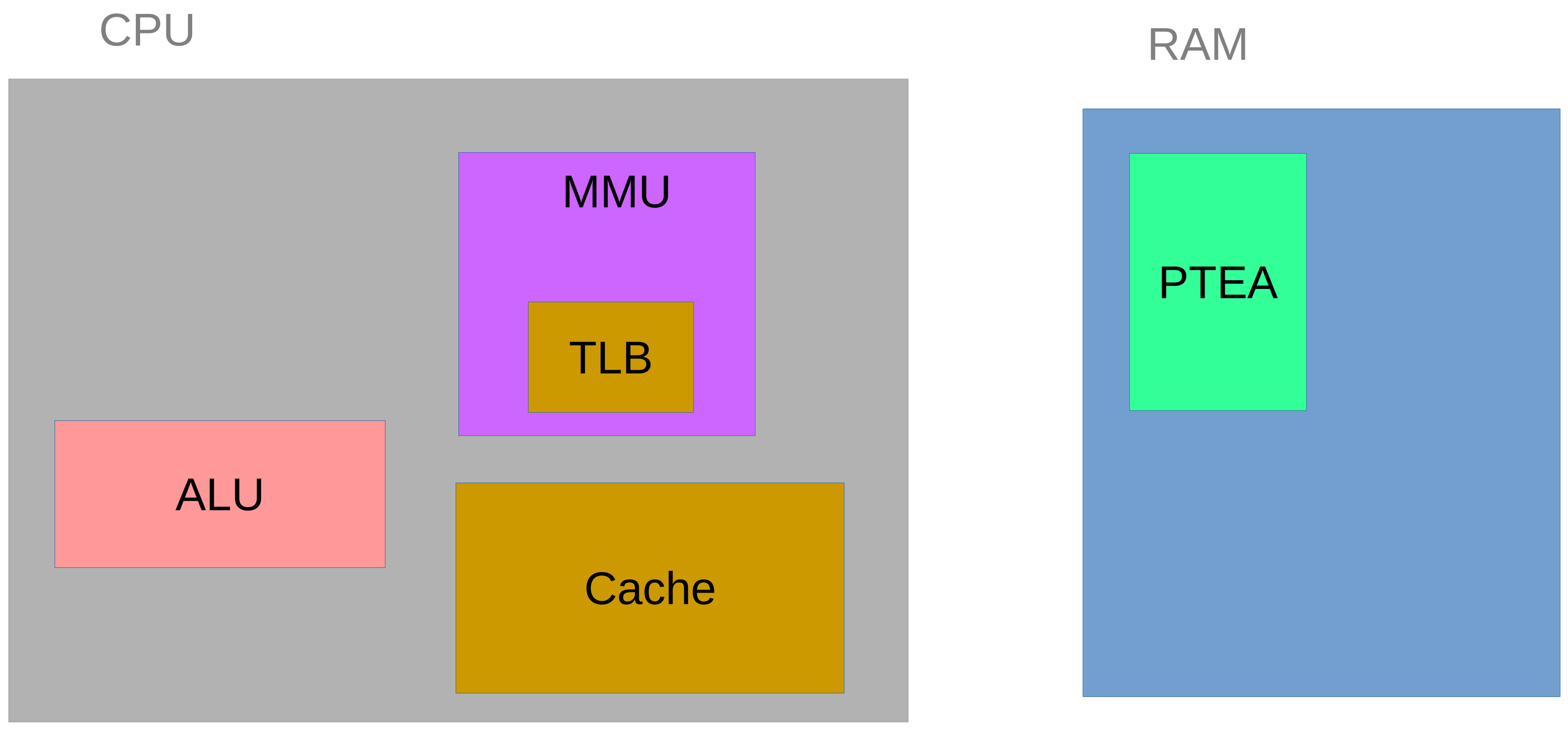


PTEA = Page Table Entry Array

In order to speed things up, page table entries are cached
so that the MMU does not always have to access RAM directly.

Furthermore, a small amount of memory is placed directly
on the MMU; it is dedicated to caching page table entries.
It is known as the Translation Lookaside Buffer (TLB).

Enhanced view of virtual memory hardware:



Hence, the process of determining the physical page number (PPN) occurs conceptually as follows:

1) The ALU issues an instruction to access data located at a certain virtual address.

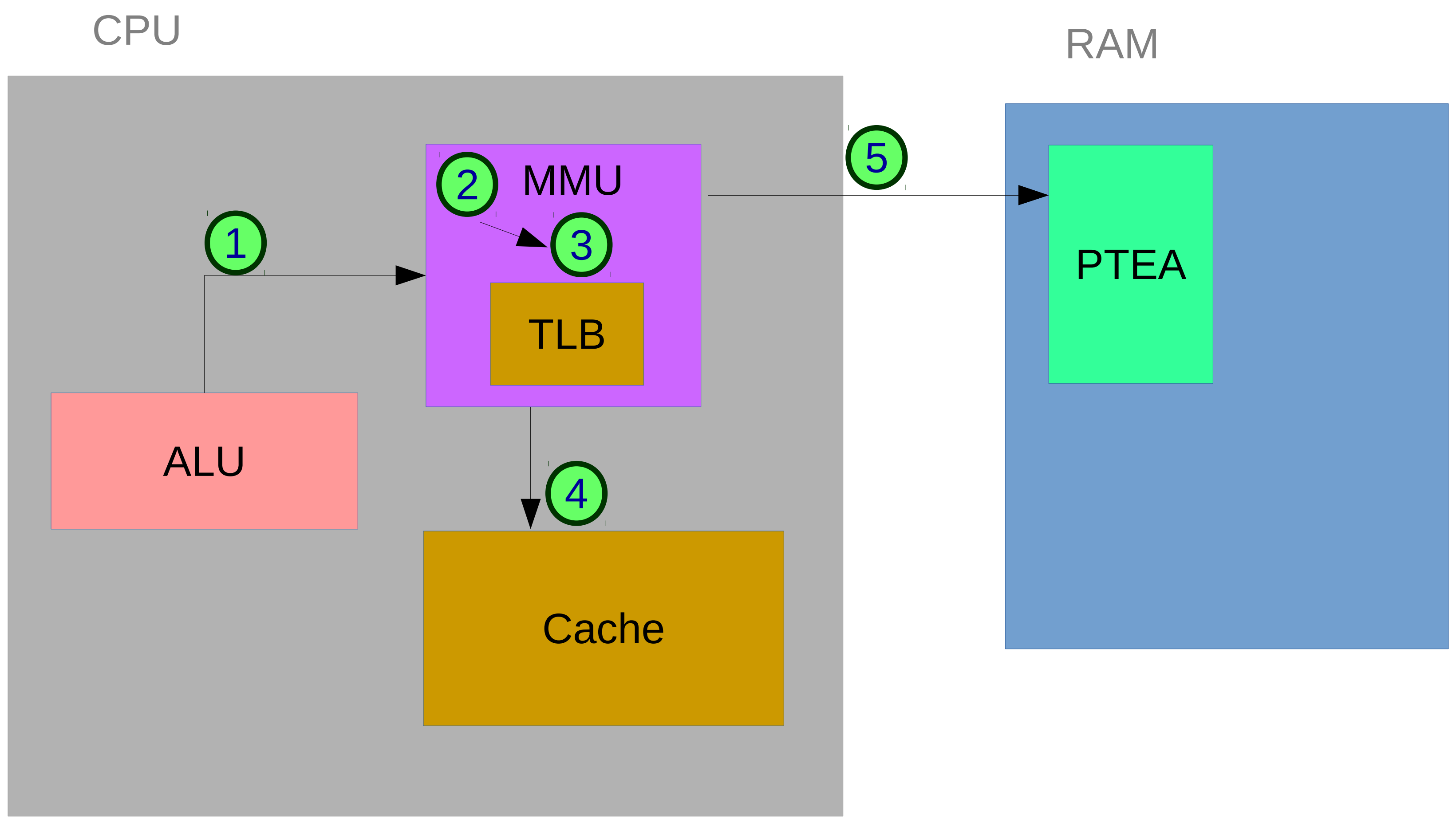
2) The MMU extracts the virtual page number from the address.

3) The MMU then checks the TLB for that virtual page number. If present, the MMU retrieves the physical page number from the TLB.

4) If not present, the MMU then checks the cache for the virtual page number. If the page table entry has been cached, the MMU retrieves the page number from the cache entry.

5) If not in the cache, the MMU must look in RAM for the page table entry that matches the virtual page number.

This procedure for obtaining the PPN can be represented graphically:



In reality, the MMU need only check the TLB for the virtual page number and, if not present, check the cache.

When the cache is checked but the entry is not present, the caching mechanism retrieves the page table entry and places it in the cache.

The cache returns the entry to the MMU, which places it in the TLB for future reference.

Once the MMU obtains the physical page number, it can then compose the physical address by combining the physical page number with the offset provided in the virtual address.

Having the physical address, the MMU generates a request for the data, which is returned to the ALU.

Virtual memory involves some overhead, but it also provides some great advantages:

- Prevents processes from accessing memory belonging to other processes and the system.
- Permits the introduction of constraints on the use of memory, such as read-only and non-executable use.
- Simplifies memory management for programmers.