Microprocessors

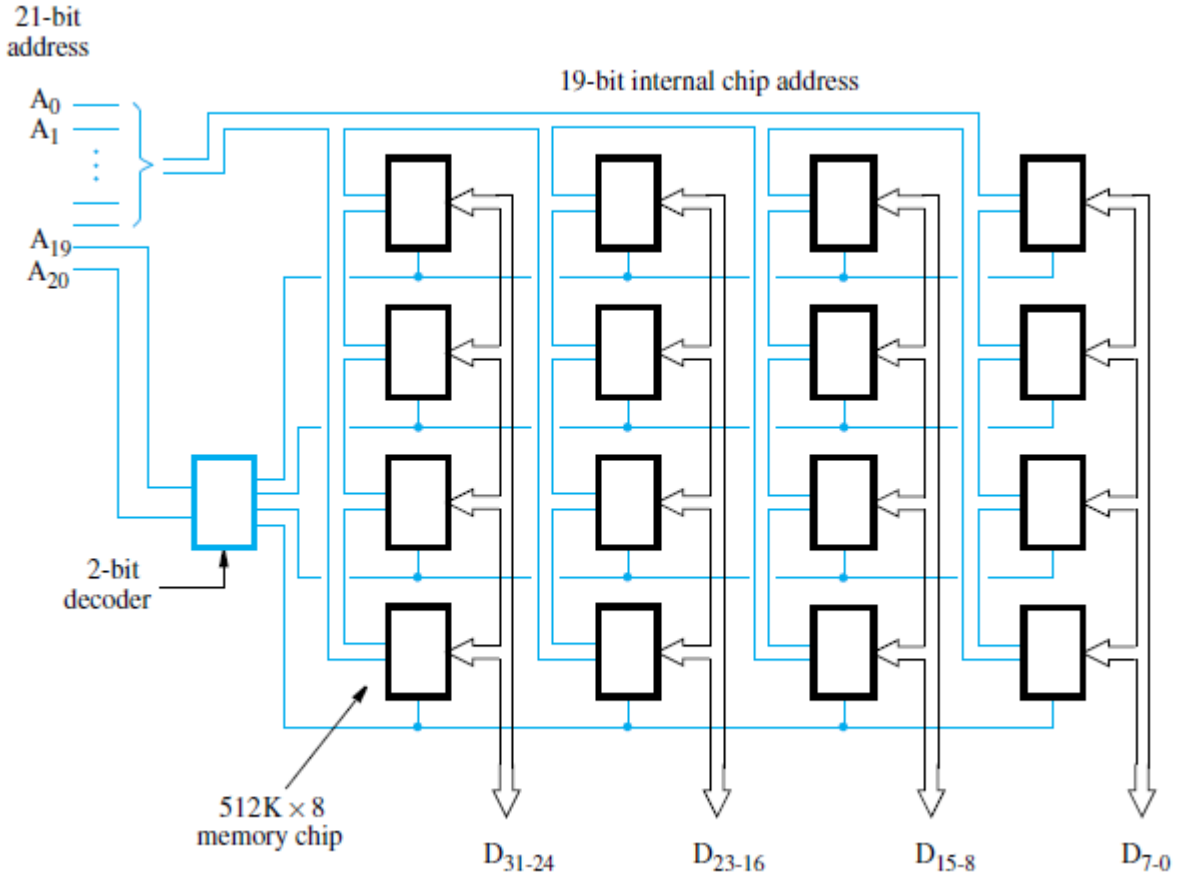
Tuba Ayhan

MEF University

Memories

CH8.6, 8.8

Structure of Larger Memories



Question: Describe a structure for an 8M × 32 memory using 512K × 8 memory chips.

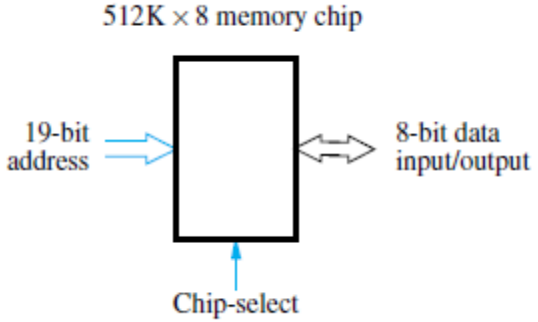


Figure 8.10 Organization of a 2M × 32 memory module using 512K × 8 static memory chips.

- 16 rows are needed, each with four 512 × 8 chips.
- Address lines *A*18–0 should be connected to all chips.
- Address lines *A*22–19 should be connected to a 4-bit decoder to select one of the 16 rows.

Cache Memories

- A small and very fast memory
- Function: make the main memory appear to the processor to be much faster than it actually is.
- Principle: *locality of reference.*
- Most of execution time is spent in routines in which many instructions are executed repeatedly:
  - a simple loop,
  - nested loops,
  - a few procedures that repeatedly call each other.

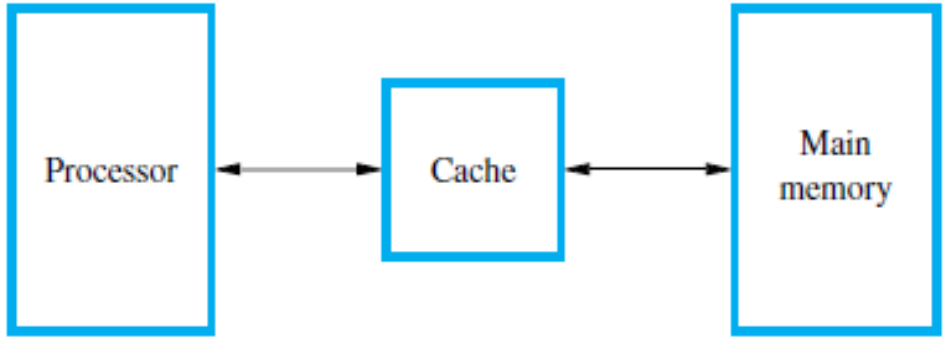


Figure 8.15 Use of a cache memory.

Cache Memories

- Principle: *locality of reference.*
- *Temporal:* recently executed instruction is likely to be executed again very soon.
- *Spatial:* instructions close to a recently executed instruction are also likely to be executed soon.

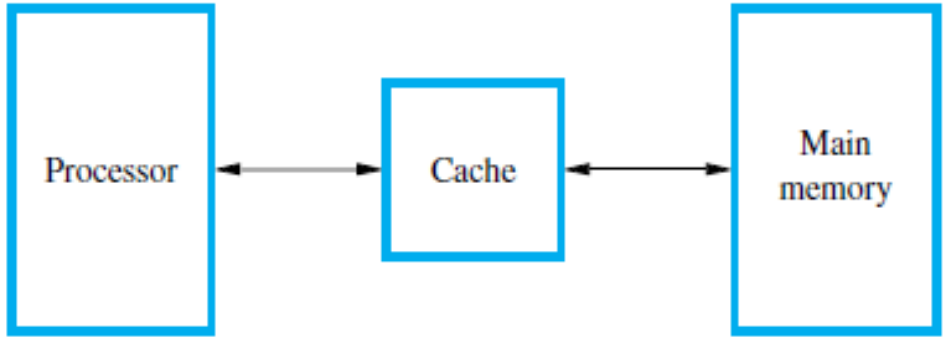


Figure 8.15 Use of a cache memory.

Cache Memories – Operation

- A *memory control circuitry* is designed:
- Temporal locality: whenever an information item is first needed, this item should be brought into the cache, because it is likely to be needed again soon.
- Spatial locality: instead of fetching just one item from the main memory to the cache, it is useful to fetch several items that are located at adjacent addresses as well.
- Cache block: A set of contiguous address locations of some size.
- Example: When the processor issues a Read request:
  - The contents of a block of memory words containing the location specified are transferred into the cache.
  - When the program references any of the locations in this block, the desired contents are read directly from the cache.

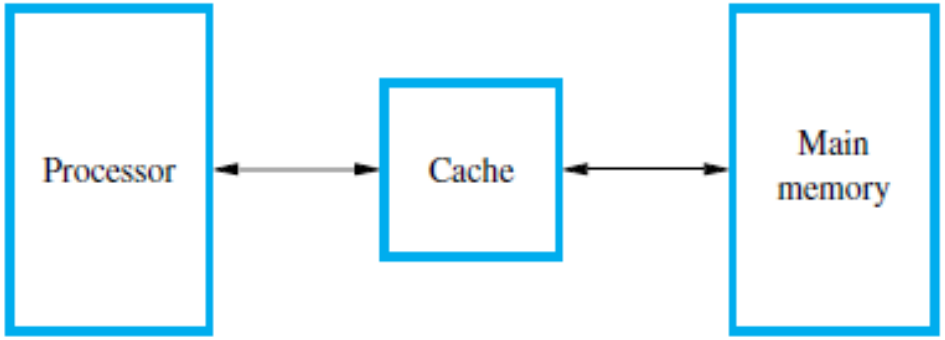


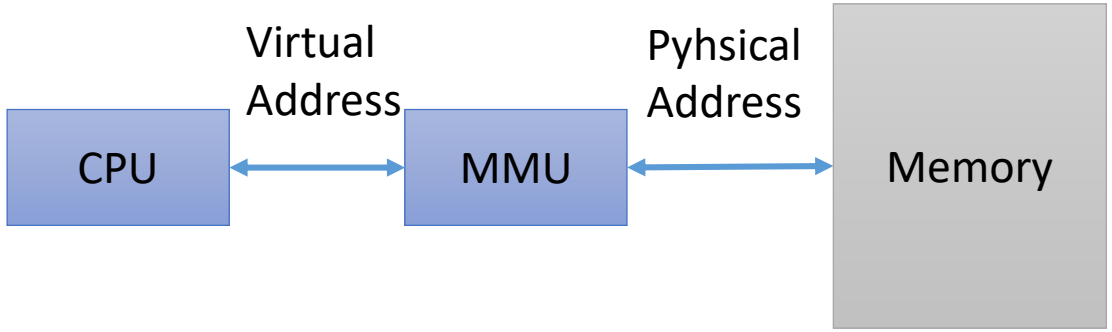
Figure 8.15 Use of a cache memory.

- The correspondence between the main memory blocks and those in the cache is specified by a *mapping function*.
- *Cache's replacement algorithm:* When the cache is full and a memory word that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word.

Virtual Memory

- In most modern computer systems, the physical main memory is not as large as the address space of the processor.
- If a program does not completely fit into the main memory, the parts of it not currently being executed are stored on a secondary storage device, typically a magnetic disk.
- As these parts are needed for execution, they must first be brought into the main memory, possibly replacing other parts that are already in the memory → using a scheme known as *virtual memory*.
- Application programmers need not be aware of the limitations imposed by the available main memory. They prepare programs using the entire address space of the processor.

Virtual Memory



- The binary addresses that the processor issues for either instructions or data are called *virtual or logical addresses*.
- These addresses are translated into physical addresses by a combination of hardware and software actions.
- A virtual address refers to program or data
  - That is currently in the physical memory → the contents of the appropriate location in the main memory are accessed immediately.
  - Otherwise → the contents of the referenced address must be brought into a suitable location in the memory before they can be used.

Virtual Memory

- The *Memory Management Unit* (MMU) keeps track of which parts of the virtual address space are in the physical memory.
- The desired data or instructions are in the main memory → the MMU translates the virtual address into the corresponding physical address. Then, the requested memory access proceeds in the usual manner.
- The desired data or instructions are not in the main memory → the MMU arranges the transfer of data from the disk to the memory. Such transfers are performed using the DMA scheme.

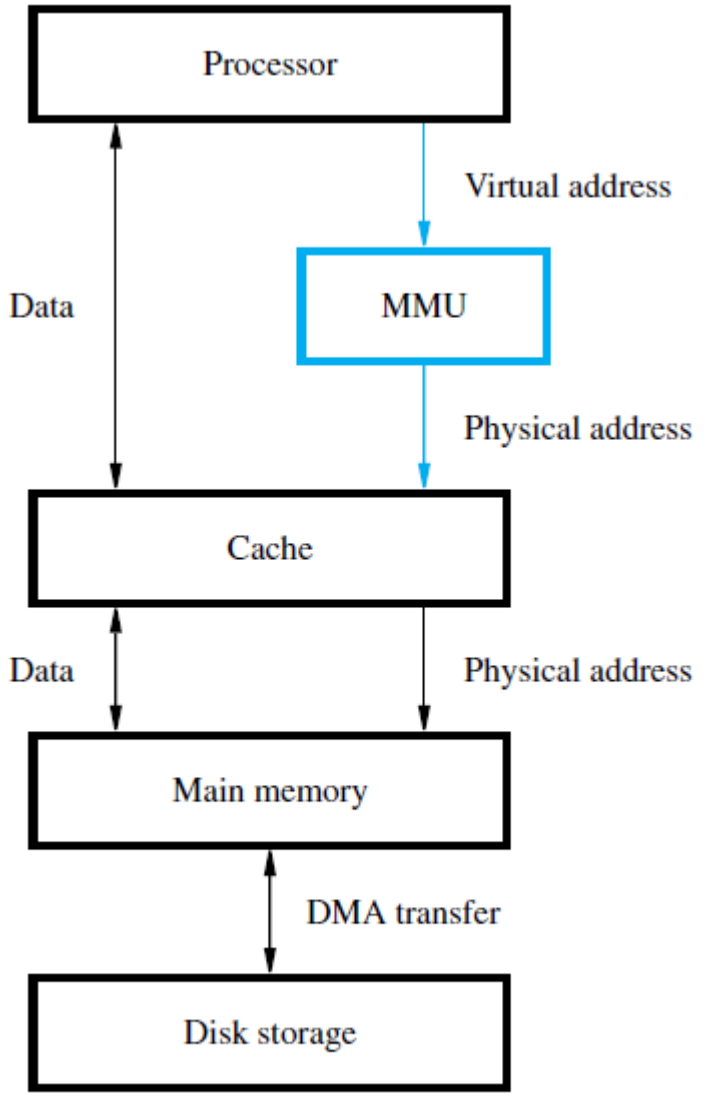


Figure 8.24 Virtual memory organization.

Page

- *Pages*: All programs and data are composed of fixed-length units, each of which consists of a block of words that occupy contiguous locations in the main memory.
- Pages commonly range from 2K to 16K bytes in length.
- Pages should not be too small →
  - The access time of a magnetic disk is much longer (several milliseconds) than the access time of the main memory. The data on the cache can be transferred at a rate of several megabytes per second.
- Pages should not be too large →
  - A substantial portion of a page may not be used, yet this unnecessary data will occupy valuable space in the main memory.

Cache and Virtual memory

- Conceptually, cache techniques and virtual-memory techniques are very similar:
- The cache bridges the speed gap between the processor and the main memory and is implemented in **hardware**.
- The virtual-memory mechanism bridges the size and speed gaps between the main memory and secondary storage and is usually implemented in **part by software techniques**.
- They differ in the details of their implementation.

Address Translation

Using Page Concept

- Each virtual address is interpreted as
  - a virtual page number (high-order bits)
  - followed by an offset (low-order bits) that specifies the location of a particular byte (or word) within a page.
- Information about the main memory location of each page is kept in a page table.
  - This info includes the main memory address where the page is stored and the current status of the page.

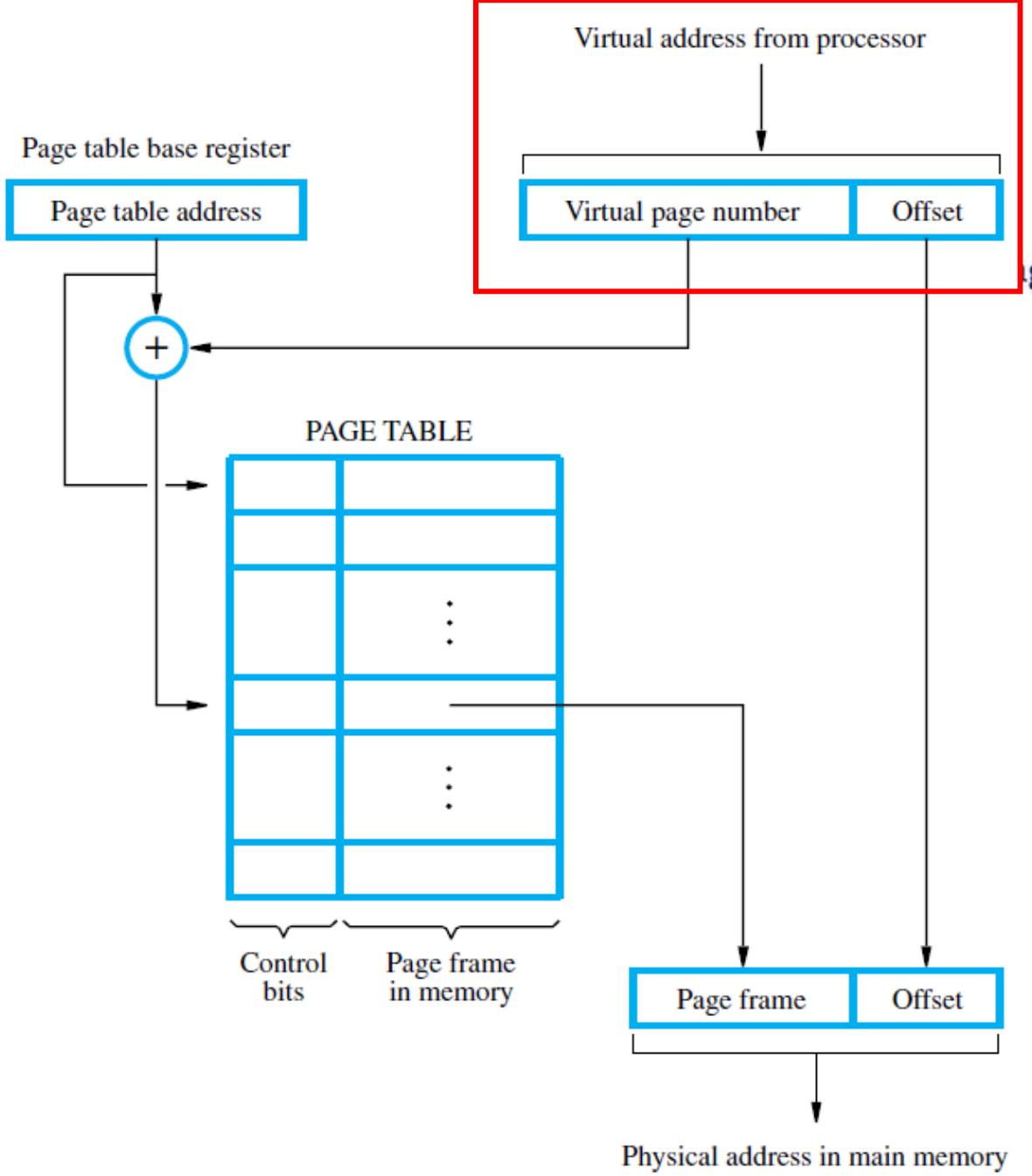


Figure 8.25 Virtual-memory address translation.

Address Translation

Using Page Concept

- An area in the main memory that can hold one page is called a *page frame*.
- *Page table base register*: The starting address of the page table.
- The virtual page number + page table base register = the starting address of the page, if that page currently resides in the main memory.

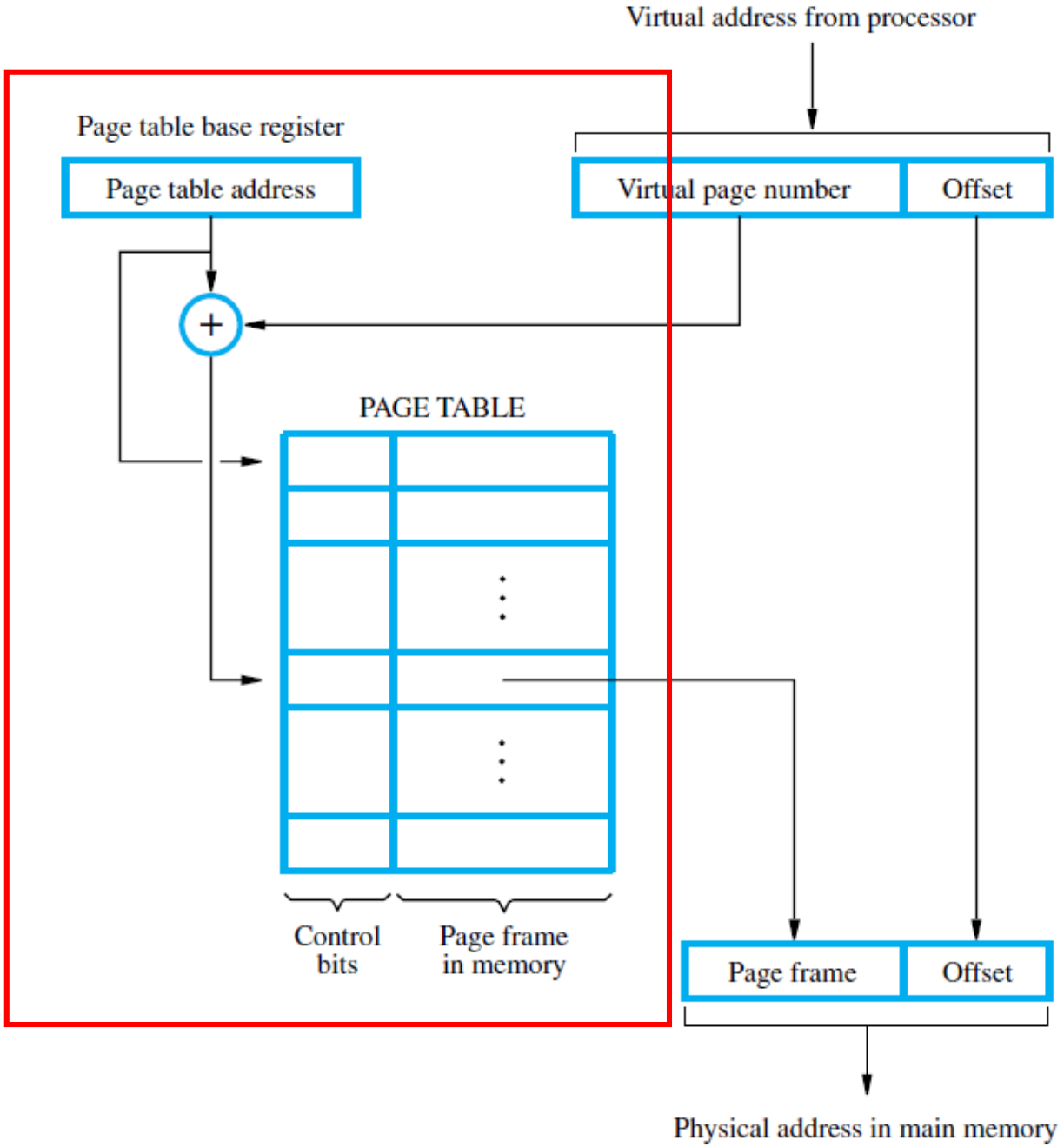


Figure 8.25 Virtual-memory address translation.

Address Translation

Using Page Concept

- Control bits of the page table describe the status of the page while it is in the main memory.
- **One bit indicates the validity of the page**: whether the page is actually loaded in the main memory. It allows the operating system to invalidate the page without actually removing it
- **One bit indicates whether the page has been modified** during its residency in the memory. As in cache memories, this information is needed to determine whether the page should be written back to the disk before it is removed from the main memory to make room for another page.
- **Other control bits indicate various restrictions** that may be imposed on accessing the page. For example, a program may be given full read and write permission, or it may be restricted to read accesses only.

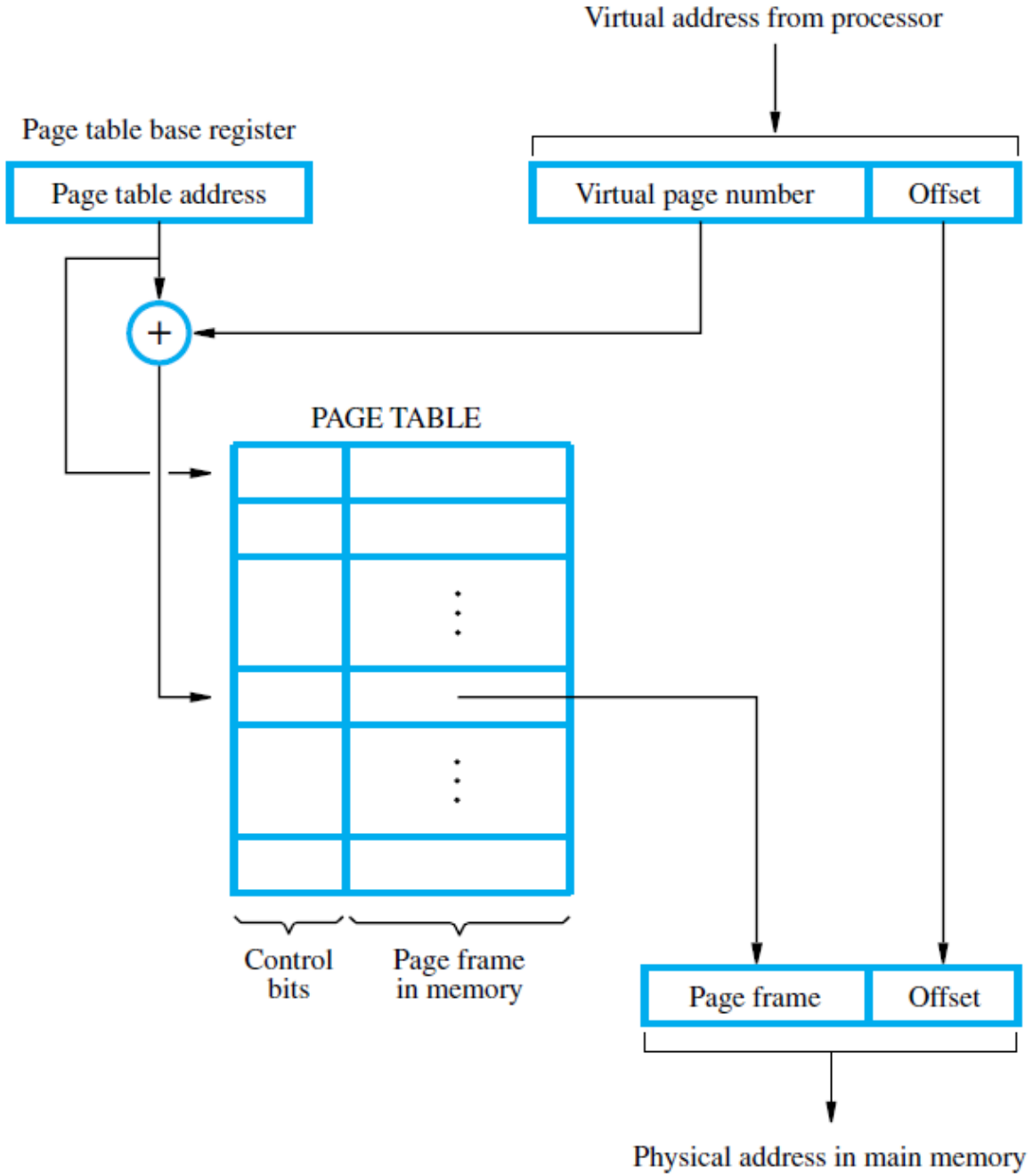


Figure 8.25 Virtual-memory address translation.

Translation Lookaside Buffer (TLB)

- The page table information is used by the MMU for every read and write access.
- Ideally, the page table should be situated within the MMU. Instead, a copy of only a small portion of the table is accommodated within the MMU, and the complete table is kept in the main memory.
- The portion maintained within the MMU consists of the entries corresponding to the most recently accessed pages. They are stored in *Translation Lookaside Buffer (TLB)*.
- The TLB functions as a cache for the page table in the main memory.
- Each entry in the TLB includes a copy of the information in the corresponding entry in the page table.
- It includes the virtual address of the page, which is needed to search the TLB for a particular page.

Microprocessors

Tuba Ayhan

MEF University

MMU – ARM

Cortex A – Programmer's Guide

MMU – ARM

- The MMU enables multiple programs to be run simultaneously at the same virtual address while being actually stored at different physical addresses.
- MMU allows us to build systems with multiple virtual address maps. Each task can have its own virtual memory map. The OS kernel places code and data for each application in physical memory, but the application itself does not need to know where that is.
- Address translation is done using using *page tables*.
  - Page tables contain a series of entries, each of which describes the physical address translation for part of the memory map.
  - Page table entries are organized by virtual address.
  - They provide access permissions and memory attributes for that page.

- Addresses generated by the processor core are [redacted]
- The MMU essentially replaces
  - the most significant bits of this virtual address with some other value, to generate [redacted]
  - the lower bits are the same in both addresses → PAGE [redacted]
- The ARM MMU supports a multi-level page table architecture with two levels of page table: level 1 (L1) and level 2 (L2).
- A single set of page tables is used to give the translations and memory attributes which apply to instruction fetches and to data reads or writes.
- *Page table walking:* The process in which the MMU accesses page tables to translate addresses.

Level 1 page tables

- Step 1: locate the page table entry associated with the virtual address.
- The L1 page table divides the full 4GB address space into ??? equally sized 1MB sections.
- The L1 page table therefore contains ??? entries, each entry being word sized.
- Each entry can either hold a pointer to the base address of a level 2 page table or a page table entries for translating a 1MB section.
- If the page table entry is translating a 1MB section, it gives the base address of the 1MB page in physical memory.
- To locate the relevant entry in the page table, we take the top 12 bits of the virtual address and use those to index to one of the ??? words within the page table.

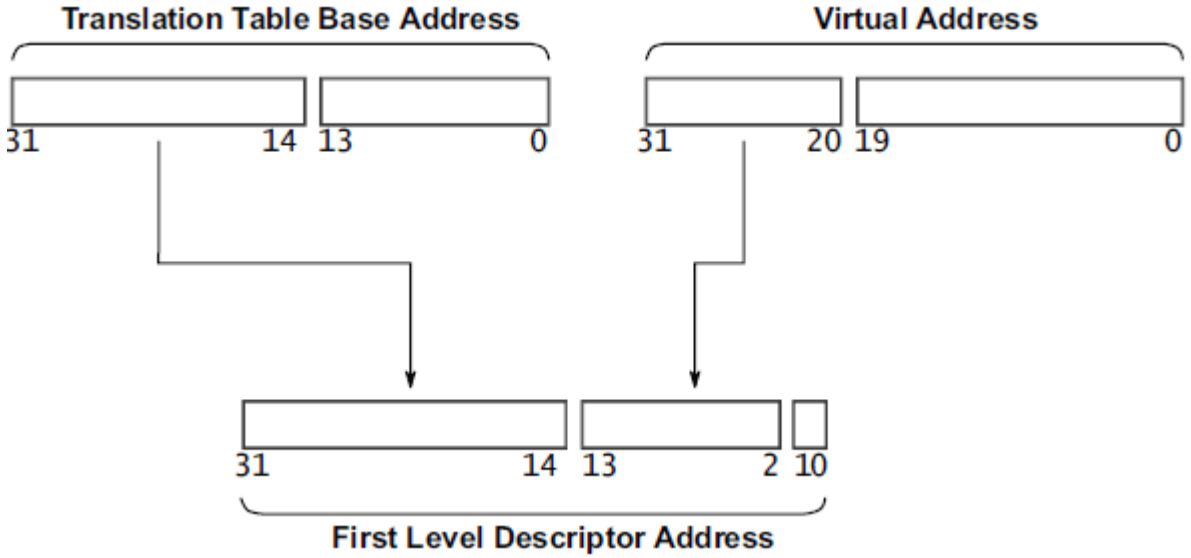


Figure 8-2 Finding the address of the level 1 page table entry

Level 1 page tables

- Step 1: locate the page table entry associated with the virtual address.
- The L1 page table divides the full 4GB address space into 4096 equally sized 1MB sections.
- The L1 page table therefore contains 4096 entries, each entry being word sized.
- Each entry can either hold a pointer to the base address of a level 2 page table or a page table entries for translating a 1MB section.
- If the page table entry is translating a 1MB section, it gives the base address of the 1MB page in physical memory.
- To locate the relevant entry in the page table, we take the top 12 bits of the virtual address and use those to index to one of the 4096 words within the page table.

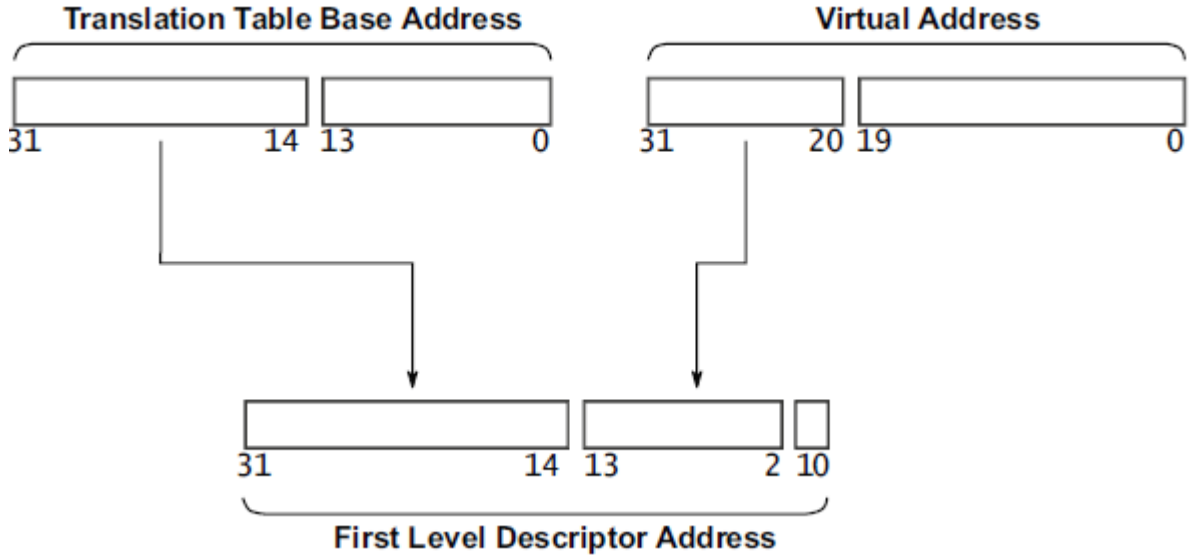


Figure 8-2 Finding the address of the level 1 page table entry

Level 1 page tables – Example

- We place our L1 page table at address 0x12300000.
- The processor core issues virtual address 0x00100000.
- The top 12 bits [31:20] define which 1MB of virtual address space is being accessed. In this case 0x001, so we need to read table entry [1]. Each entry is one word (4 bytes).
- To get the offset into the table we must multiply the entry number by entry size:
- 0x001 * 4 = Address offset of 0x004
- The address of the entry is 0x12300000 + 0x004 = 0x12300004.
- So, upon receiving this virtual address from the processor, the MMU will read the word from address 0x12300004. That word is an L1 page table entry.

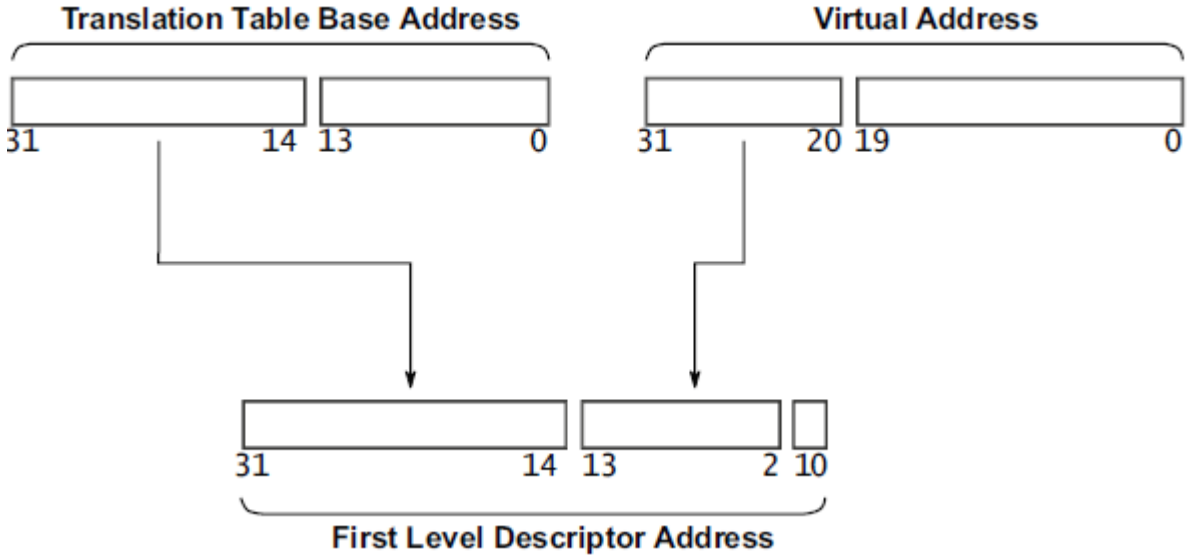


Figure 8-2 Finding the address of the level 1 page table entry

Question: L1 Page table is located at 0x12300000. (L1 page table base)
Virtual address = 0x00100000. Find the first level desc. address.
NOTE! Each enty is one word.

Level 1 – example

- In a page table entry for a 1MB section of memory, the upper 12 bits of the page table entry replace the upper 12 bits of the virtual address when generating the physical address

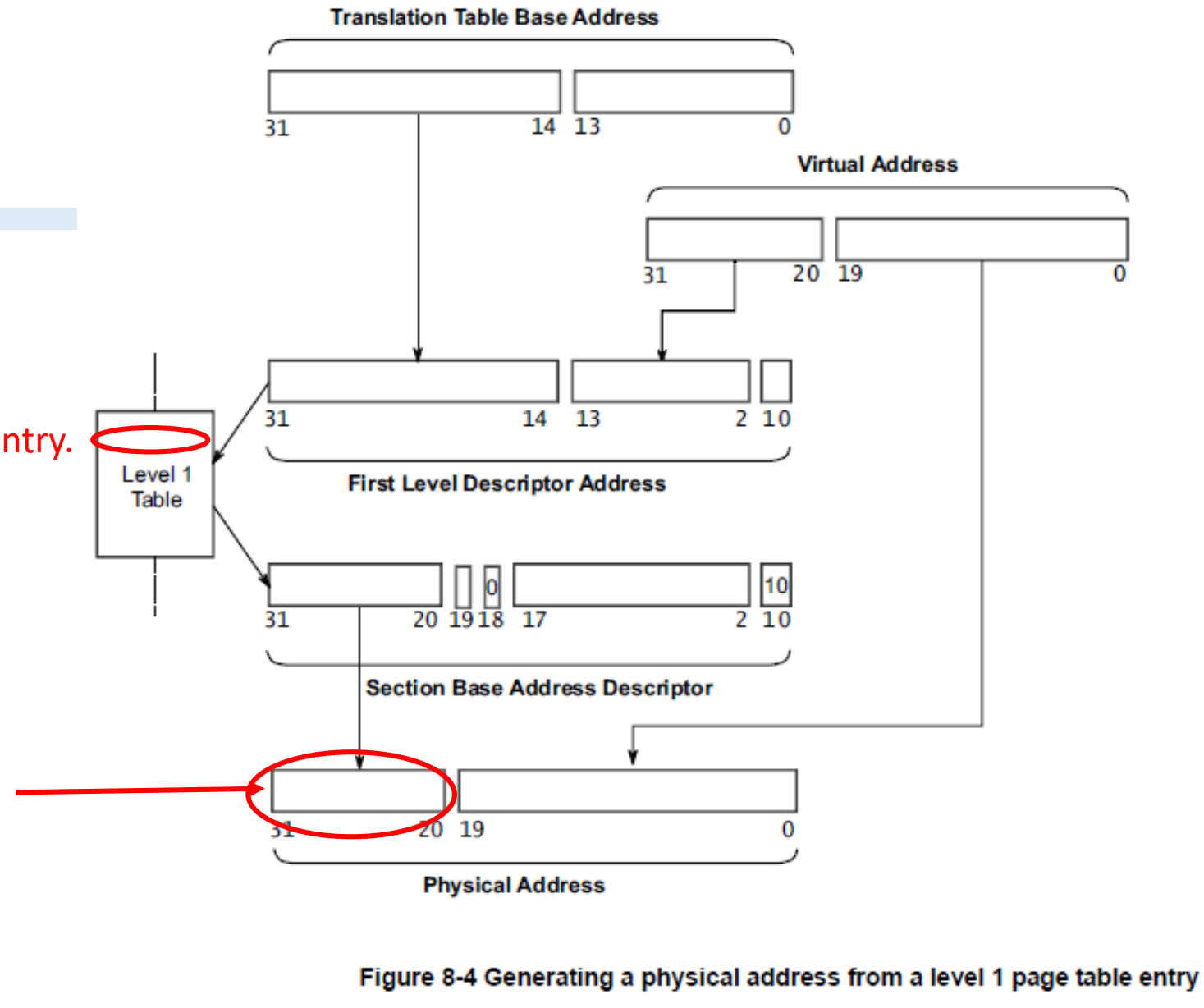


Figure 8-4 Generating a physical address from a level 1 page table entry

L1 Page table entry

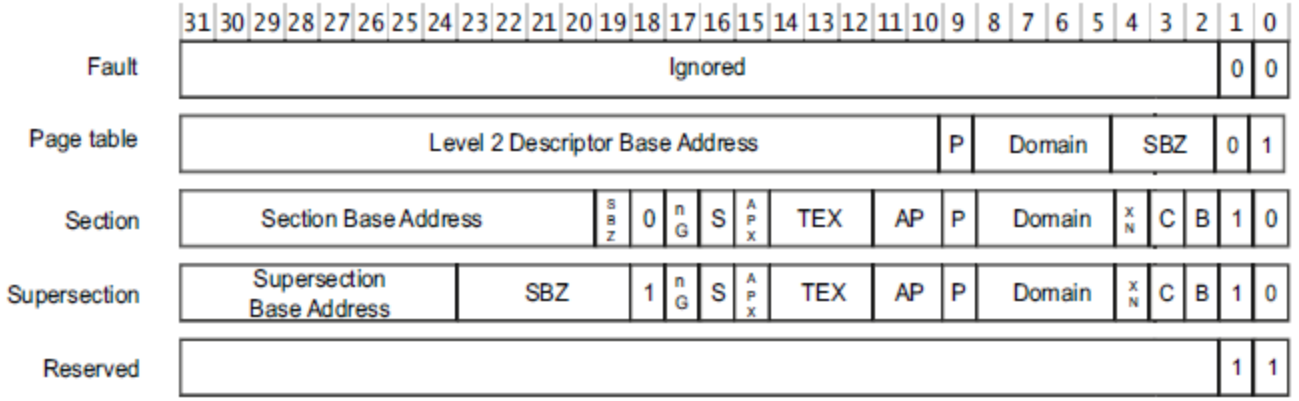


Figure 8-3 Level 1 page table entry format

1. A **fault** entry that generates an abort exception.
2. A 1MB section **translation entry**.
3. An entry that **points to an L2 page table**. This enables a 1MB piece of memory to be further sub-divided into smaller pages.
4. A 16MB **supersection**. This is a special kind of 1MB section entry, which requires 16 entries in the page table.

The least significant two bits [1:0] in the entry define which one of these the entry contains (with bit [18] being used to differentiate between a section and supersection).

Level 2 page tables

An L2 page table has 256 word-sized entries,

- requires 1KB of memory space
- and must be aligned to a 1KB boundary.
- Each entry translates a 4KB block of virtual memory to a 4KB block in physical memory.
- A page table entry can give the base address of either a 4KB or 64KB page.

Level 2 page tables

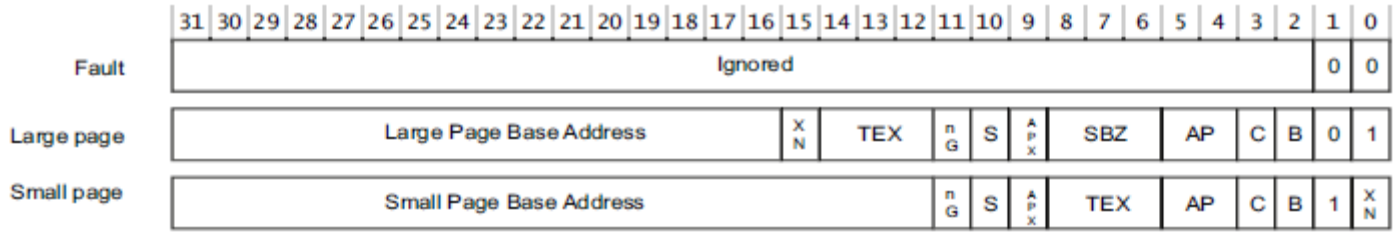


Figure 8-5 Format of a level 2 page table entry

- There are three types of entry used in L2 page tables, identified by the value in the two least significant bits of the entry:
- A **large** page entry points to a 64KB page.
- A **small** page entry points a 4KB page.
- A **fault** page entry generates an abort exception if accessed.

Level 2 – example

The address of the L2 page table entry that we need is calculated by taking

- the (1KB aligned) base address of the level 2 page table (given by the level 1 page table entry)
- and using 8 bits of the virtual address (bits [19:12]) to index within the 256 entries in the L2 page table.

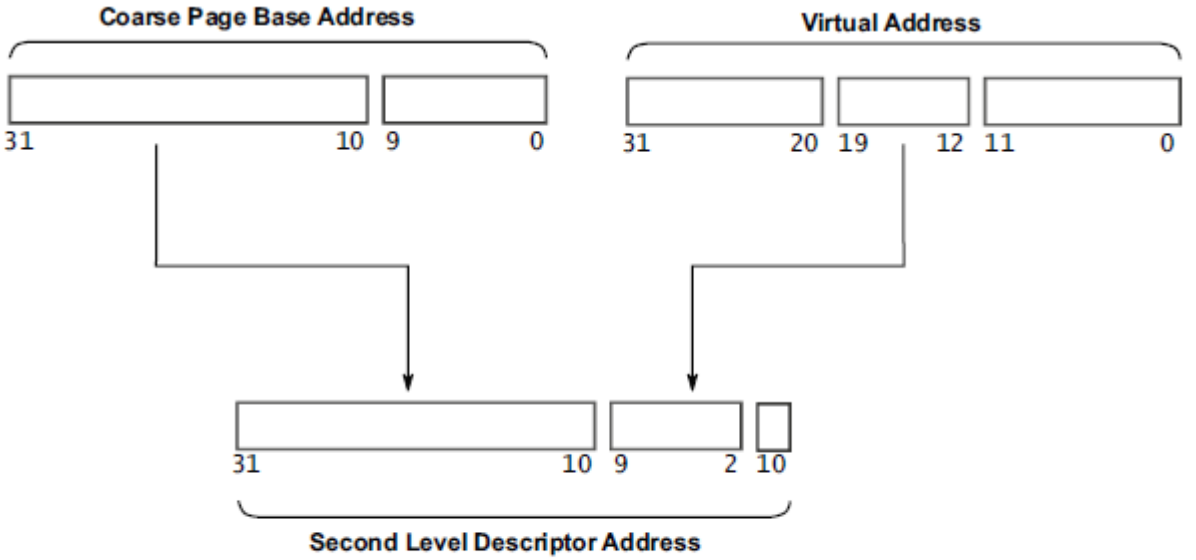


Figure 8-6 Generating the address of the level 2 page table entry

The address translation process when using 2 layers of page tables

- Bits [31:20] of the virtual address are used to index into the 4096-entry L1 page table.
- The L1 page table entry points to an L2 page table, which contains 256 entries. Bits [19:12] of the virtual address are used to select one of those entries which then gives the base address of the page.
- The final physical address is generated by combining that base address with the remaining bits of the physical address.

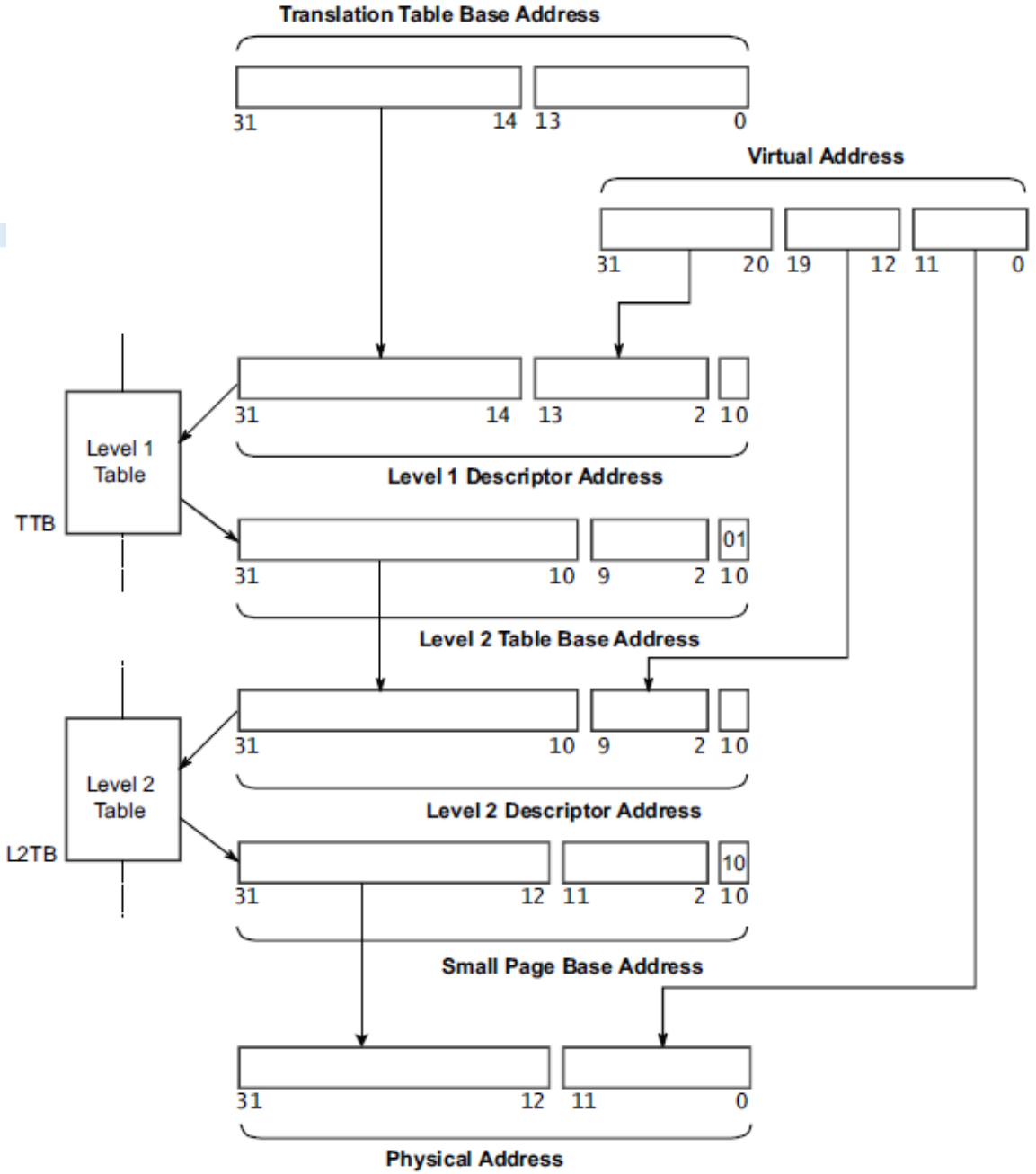


Figure 8-7 Summary of generation of physical address using the L2 page table entry