COMPSCI 2GA3 Tutorial 11 Note

Note:

This note does NOT cover all the materials in Chapter 5 -- Only the ones rated to sample questions of this tutorial are included.

For any questions about the tutorials and courses, feel free to contact me. (Email: wangm235@mcmaster.ca)

GLHF :)

Mingzhe Wang

Dependable Memory Hierarchy (MTBF, MTTR, MTTF etc.)

Elaboration: To calculate how many bits are needed for SEC, let p be total number of parity bits and d number of data bits in $p + d$ bit word. If p error correction bits are to point to error bit ($p + d$ cases) plus one case to indicate that no error exists, we need:

$$2^p \geq p + d + 1 \text{ bits, and thus } p \geq \log(p + d + 1).$$

For example, for 8 bits data means $d = 8$ and $2^p \geq p + 8 + 1$, so $p = 4$. Similarly, $p = 5$ for 16 bits of data, 6 for 32 bits, 7 for 64 bits, and so on.

EXAMPLE

MTTF vs. AFR of Disks

Some disks today are quoted to have a 1,000,000-hour MTTF. As 1,000,000 hours is 1,000,000/(365 × 24) = 114 years, it would seem like they practically never fail. Warehouse scale computers that run Internet services such as Search might have 50,000 servers. Assume each server has 2 disks. Use AFR to calculate how many disks we would expect to fail per year.

ANSWER

One year is 365 × 24 = 8760 hours. A 1,000,000-hour MTTF means an AFR of 8760/1,000,000 = 0.876%. With 100,000 disks, we would expect 876 disks to fail per year, or on average more than 2 disk failures per day!

Service interruption is measured as *mean time to repair* (MTTR). *Mean time between failures* (MTBF) is simply the sum of MTTF + MTTR. Although MTBF is widely used, MTTF is often the more appropriate term. *Availability* is then a measure of service accomplishment with respect to the alternation between the two states of accomplishment and interruption. Availability is statistically quantified as

$$\text{Availability} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})}$$

Note that reliability and availability are actually quantifiable measures, rather than just synonyms for dependability. Shrinking MTTR can help availability as much as increasing MTTF. For example, tools for fault detection, diagnosis, and repair can help reduce the time to repair faults and thereby improve availability.

We want availability to be very high. One shorthand is to quote the number of "nines of availability" per year. For example, a very good Internet service today offers 4 or 5 nines of availability. Given 365 days per year, which is 365 × 24 × 60 = 526,000 minutes, then the shorthand is decoded as follows:

One nine:	90%	=> 36.5 days of repair/year
Two nines:	99%	=> 3.65 days of repair/year
Three nines:	99.9%	=> 526 minutes of repair/year
Four nines:	99.99%	=> 52.6 minutes of repair/year

More: textbook P418

SEC/DED) Hamming code

Elaboration: To calculate how many bits are needed for SEC, let p be total number of parity bits and d number of data bits in $p + d$ bit word. If p error correction bits are to point to error bit ($p + d$ cases) plus one case to indicate that no error exists, we need:
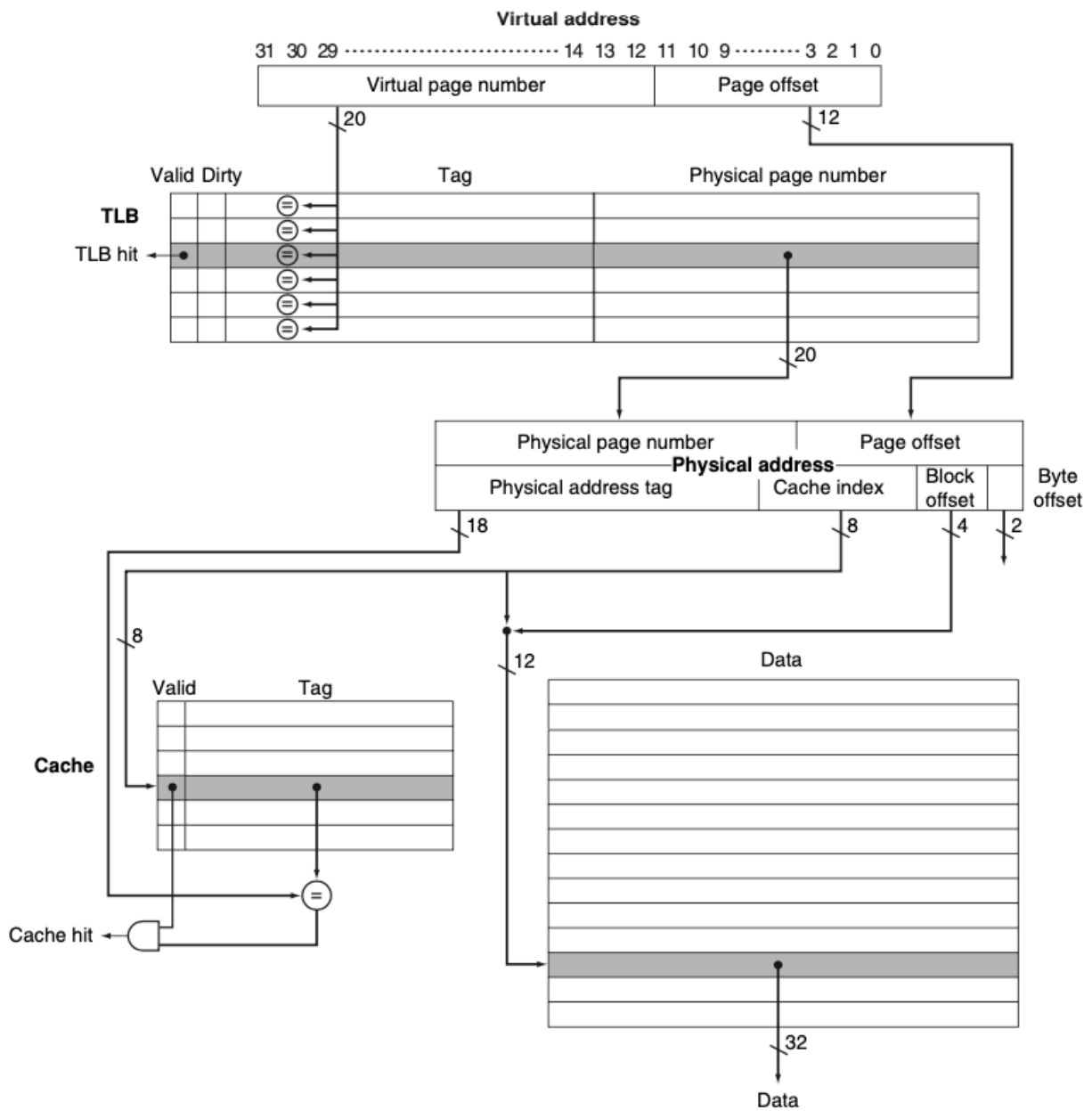
$$2^p \geq p + d + 1 \text{ bits, and thus } p \geq \log(p + d + 1).$$

For example, for 8 bits data means $d = 8$ and $2^p \geq p + 8 + 1$, so $p = 4$. Similarly, $p = 5$ for 16 bits of data, 6 for 32 bits, 7 for 64 bits, and so on.

1. Start numbering bits from 1 on the left, as opposed to the traditional numbering of the rightmost bit being 0.

2. Mark all bit positions that are powers of 2 as parity bits (positions 1, 2, 4, 8, 16, …) .

3. All other bit positions are used for data bits (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, …).

4. The position of parity bit determines sequence of data bits that it checks (Figure 5.24 shows this coverage graphically) is:

   ■ Bit 1 (0001_{two}) checks bits (1,3,5,7,9,11,...), which are bits where rightmost bit of address is 1 (0001_{two}, 0011_{two}, 0101_{two}, 0111_{two}, 1001_{two}, 1011_{two},….).
   ■ Bit 2 (0010_{two}) checks bits (2,3,6,7,10,11,14,15,…), which are the bits where the second bit to the right in the address is 1.
   ■ Bit 4 (0100_{two}) checks bits (4–7, 12–15, 20–23,...) , which are the bits where the third bit to the right in the address is 1.
   ■ Bit 8 (1000_{two}) checks bits (8–15, 24–31, 40–47,...), which are the bits where the fourth bit to the right in the address is 1.

   Note that each data bit is covered by two or more parity bits.

5. Set parity bits to create even parity for each group.

Virtual Machines

Virtual address

31 30 29 ·························· 14 13 12 11 10 9 ········ 3 2 1 0

Virtual page number	Page offset

20

12

Valid Dirty   Tag   Physical page number

TLB

TLB hit ◄———

20

Physical page number	Page offset

Physical address

Physical address tag	Cache index	Block offset		Byte offset

18

8

4

2

8

12   Data

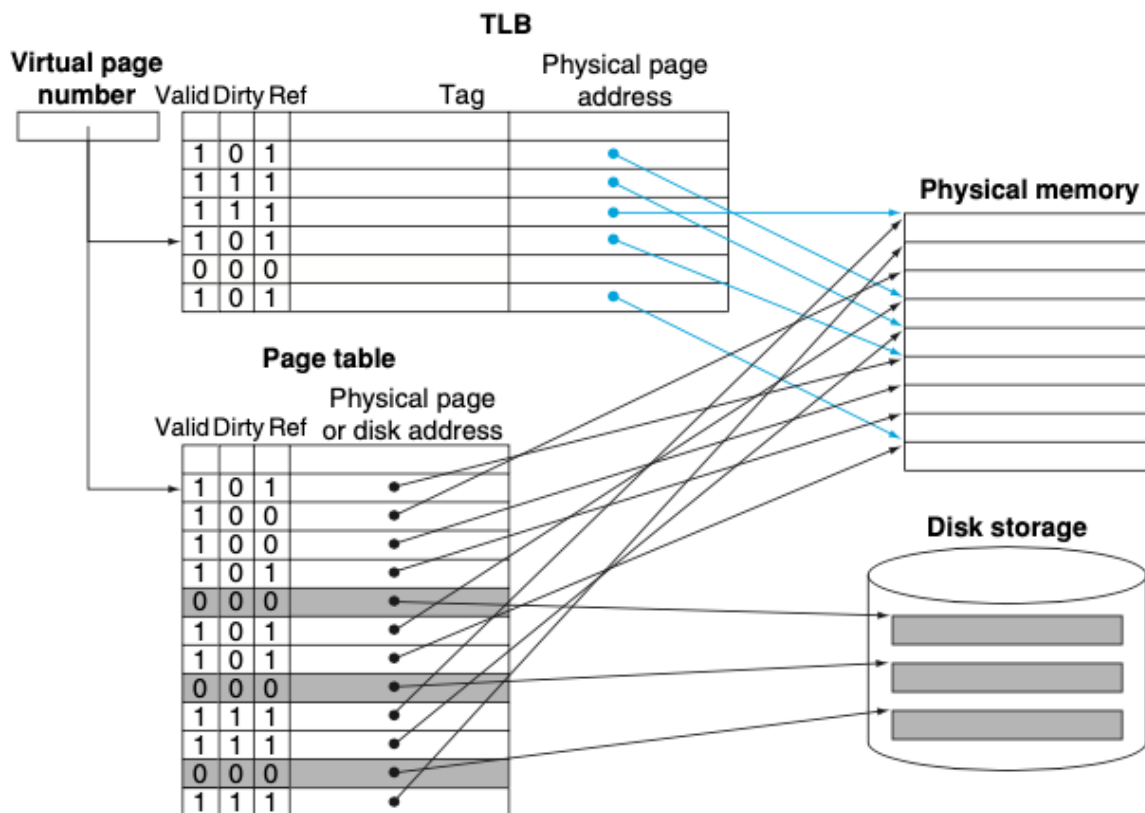Valid   Tag

Cache

Cache hit ◄———

32

Data

FIGURE 5.29   The TLB acts as a cache of the page table for the entries that map to physical pages only. The TLB contains a subset of the virtual-to-physical page mappings that are in the page table. The TLB mappings are shown in color. Because the TLB is a cache, it must have a tag field. If there is no matching entry in the TLB for a page, the page table must be examined. The page table either supplies a physical page number for the page (which can then be used to build a TLB entry) or indicates that the page resides on disk, in which case a page fault occurs. Since the page table has an entry for every virtual page, no tag field is needed; in other words, unlike a TLB, a page table is *not* a cache.

Data

FIGURE 5.30   The TLB and cache implement the process of going from a virtual address to a data item in the Intrinsity FastMATH. This figure shows the organization of the TLB and the data cache, assuming a 4 KiB page size. This diagram focuses on a read; Figure 5.31 describes how to handle writes. Note that unlike Figure 5.12, the tag and data RAMs are split. By addressing the long but narrow data RAM with the cache index concatenated with the block offset, we select the desired word in the block without a 16:1 multiplexor. While the cache is direct mapped, the TLB is fully associative. Implementing a fully associative TLB requires that every TLB tag be compared against the virtual page number, since the entry of interest can be anywhere in the TLB. (See content addressable memories in the *Elaboration* on page 408.) If the valid bit of the matching entry is on, the access is a TLB hit, and bits from the physical page number together with bits from the page offset form the index that is used to access the cache.

More: textbook p424