IJARnD INTERNATIONAL JOURNAL OF ADVANCE RESEARCH AND DEVELOPMENT

(Volume 5, Issue 5)

Available online at: www.ijarnd.com

A simulation based study of TLB misses handling

Khaing Myint, Aye Aye Chaw

University of Computer Studies, Mandalay, Myanmar

ABSTRACT

Most operating systems assign a page table for each process. The page table keeps track of where the virtual pages are saved in the physical memory. The virtual memory scheme would suffer the effect of doubling the memory access time. We can reduce the time taken to access the page table again and again by using Translation Lookaside Buffer (TLB). But, when we don't find the page frame number inside the TLB, the CPU has to access main memory for it. One problem is where the needed information itself actually is in a cache, although the information for virtual-to-physical translation is not in a TLB. A TLB miss can be more important due to the need for not just a load from main memory. The paper priority explains the concept of TLB miss handling because the translation is performed quickly without having to consult the page table. This paper aims to discuss how hardware can help us make address translation faster and how to provide MIPS R4000 architecture on TLB to translate virtual address into physical address. So, we will describe TLB Control Flow Program to avoid TLB miss as much as we can. We will explain that examine an array in a tiny address trace. Also note the role that the array access will suffer even fewer misses.

Keywords— *Memory management system, Address spaces, Computer architectures, and Translation loodaside buffer (TLB)*

1. INTRODUCTION

Memory management is the act of managing the memory of the computer. It is an essential function of the operating system. The operating system maintains the virtual address spaces and the assignment of real memory to virtual memory. In the CPU, address translation hardware often indicated to as memory management unit (MMU), automatically translates the virtual addresses to physical addresses [9]. In operating system, for each process page table will be created, which will contain Page Table Entry (PTE) in register that will tell in the main memory the actual page is reside. In this case, the processor size may be big because requiring big page table although register size is small. So registers may not hold all the PTE's of page table [7]. To overcome this small register size problem, we describe that the TLB with high speed cache is set up for page table entries. And then we studies to speed up address translation and avoid the extra memory reference. The page table entry required for conversion of virtual address to physical address is not present in TLB that become TLB miss. So, we mainly describe hardware management and software management for handling TLB misses. If PTE valid hardware fills TLB and processor never knows, TLB miss must occur. If not, after which kernel decides what to do afterwards. According to hardware exceptions, processor receives TLB miss, kernel traverses page table. If TLB valid, it fills TLB and returns from fault. If not, internally calls tarp handler. We explain the concept of commonly found in modern architectures the CR3 register on x86 that it uses in hardware management TLB and the MIPS R4000 that architecture specifies a software management TLB.

2. MEMORY MANAGEMENT SYSTEM

Most modern computers have special hardware called a memory management unit (MMU). This unit sites between the CPU and the memory unit. MMUs are used to provide virtual addressing. A virtual address is generated by the CPU and a physical address is translated by MMU [2].

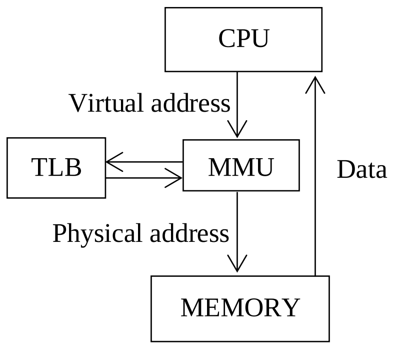

Fig. 1: Memory management unit

Most MMUs use in-memory table of items known as a page table, containing one-page table entry (PTE) per page, to map virtual page numbers to physical page numbers in main memory. The TLB that is used to avoid accessing the main memory every time when a virtual address is mapped is an associative cache of PTEs [10].

2. 1 Virtual and Physical Address Spaces

The operating system manages the physical memory and allocates portions of the available physical memory to map parts or the entire virtual address space of a process. When a process requests to access a memory location, an address translation from the virtual address space to the physical address space needs to be performed. The address translation is the function that provides that virtual-to-physical mapping; it receives as input a virtual page number, or simply a virtual address, and produces as output a physical page number, or simply a physical address [6].

2.2 Computer Architectures
Two criteria for computer architectures are possible used in modern architectures.

The x86 architecture is an instruction set architecture (ISA) series for computer processor. Developed by Intel Corporation, x86 architecture defines how a processor handles and executes different instructions passed from the operating system and software programs. The hardware is transparent to the kernel because it handled the TLB misses in the x86 architecture. The only time kernel code deals with the TLB when the contents of the TLB are to be discarded (a TLB flush) [11].

One of the most widely supposed of all processor architectures is the MIPS architecture that services to help ensure speedy, safe and the cost effective of development. MIPS architecture can solve to get maximum flexibility from processor IP by microprocessor developers. In the MIPS R4000 architecture, the TLB misses are handled by software-managed [3].

3. TRANSLATION LOOKASIDE BUFFER
Translation lookaside buffer (TLB) reduces the time taken to access a user memory location. It is a part of the chip's memory management unit (MMU) and performs translation of virtual memory address into physical memory address; thus, a better name would be an address-translation cache. A TLB may reside between the CPU and the CPU cache, between CPU cache and the main memory or between the different levels of the multi-level cache [7].

The majority of desktop, laptop, and server, smartphones and other complex computation devices including one or more TLBs as a part of the memory management facilities providing virtual memory support [8]. The TLB is almost always implemented as an associative cache. It contains only a few of the page table entries. A virtual address's page number is displayed to the TLB generating the CPU it. Its frame number is immediately available when the page number is found and then access memory. When TLB miss become that is not in, a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory [1].

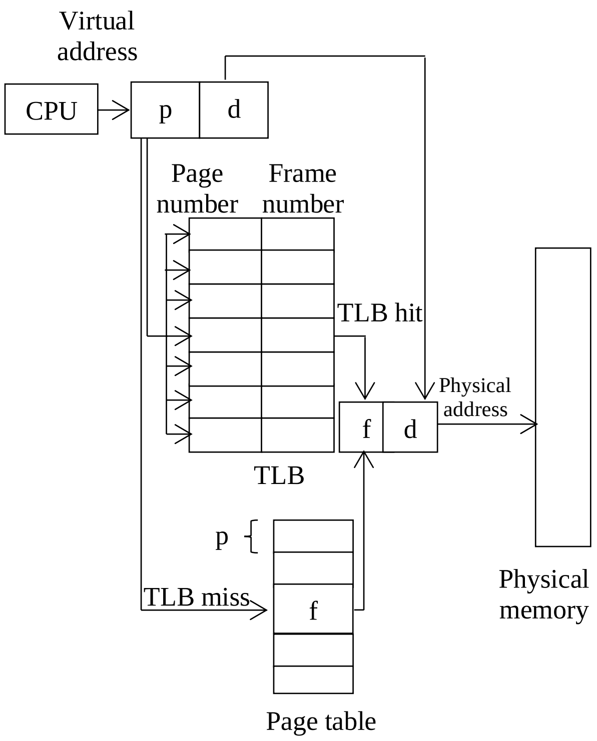

Fig. 2: Use of TLB in paging [4]

Address-space identifiers (ASIDs) are stored in each entry of the some TLB. The ASID for the currently running process matches with the virtual page when the TLB attempts to resolve virtual page numbers. The ASIDs are treated as a TLB miss when it does not match. An ASID allows the TLB to contain entries for several different processes simultaneously moreover providing address-space protection. Whenever a new page table is selected, the TLB must be flushed to ensure northing use the wrong translation information [1].

3.1 TLB Basic Algorithm
In this section, we start by looking at the basic algorithm of TLB control flow that shows a rough sketch of how hardware might handle a virtual address translation and OS handled of TLB control flow for the TLB misses.

3.1.1 TLB Control Flow Algorithm
Firstly, we studied the basic TLB control flow algorithm that the hardware follows works. That algorithm shows to handle virtual address translation by assuming a simple linear page table and a hardware-managed TLB [5].

```
1.  VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2.  (Success, TlbEntry) = TLB_Lookup (VPN)
3.  if (Success = = True)    // TLB Hit
4.      if (CanAccess (TlbEntry.ProtectBits) = = True)
5.          Offset = VirtualAddress & OFFSET_MASK
6.          PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7.          Register = AccessMemory (PhysAddr)
8.      else
9.          RaiseException (PROTECTION_FAULT)
10. else                          //TLB Miss
11.     PTEAddr = PTBR + (VPN * sizeof (PTE))
12.     PTE = AccessMemory (PTEAddr)
13.     if (PTE .Valid = = False)
14.         RaiseException (SEGMENTATION_FAULT)
15.     else if (CanAccess (PTE.ProtectBits) = = False)
16.         RaiseException (PROTECTION_FAULT)
17.     else
18.         TLB_Insert (VPN, PTE.PFN, PTE.ProtectBits)
19.         RetryInstruction ( )
```

Fig. 3: TLB Control Flow Algorithm

3. 1. 2 TLB Control Flow Algorithm (OS Handled)
Secondly, we describe OS handled TLB Control Flow algorithm, if the CPU does not find the translation in the TLB. We might guess the trap handler is code within the OS that is written with the express of handling TLB misses [5].

```
1.  VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2.  (Success, TlbEntry) = TLB_Lookup (VPN)
3.  if (Success = = True)    // TLB Hit
4.      if (CanAccess (TlbEntry.ProtectBits) = = True)
5.          Offset = VirtualAddress & OFFSET_MASK
6.          PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7.          Register = AccessMemory (PhysAddr)
8.      else
9.          RaiseException (PROTECTION_FAULT)
10. else                          //TLB Miss
11.     RaiseException (TLB_MISS)
```

Fig. 4: TLB Control Flow Algorithm (OS Handled)

4. A SIMULATION FOR HANDLING TLB MISSES
The TLB contains a storage site for storing at least a part of a virtual to physical memory translation. This storage location is managed both hardware and software. When the hardware

TLB manage, the format of the TLB entries is not visible to software and can transform from CPU to CPU nothing loss of compatibility for the programs (for instance using the CR3 register on x86 architecture).The instructions that allow loading entries into any slot in the TLB can be had with the software managed TLBs. The format of the TLB entry is defined as a part of the instruction set architecture (for instance using MIPS R4000 architecture).

4.1 TLB misses Handling with Basic Algorithm

Two schemes for handling TLB misses are commonly found in modern architecture. This section discusses how to handle TLB miss. In the above algorithm from figure (3 & 4), firstly we assume protection checks success. The TLB holds the translation that has a TLB hit. So, we can now extract the page frame number (PFN) from the relevant TLB entry and then examine on the offset from the original and raise the original address and form the needed physical address and enter the memory. If the CPU does not find the translation that has a TLB miss, the hardware allows the page table to find the translation. This problem is costly because the extra memory index needed to access the page table. In this case, we assume that the virtual memory reference generated by the process is valid and accessible, the updates the TLB with translation. When the TLB is updated once, the hardware retries the instruction. At a moment, the translation is achieved in the TLB and the memory index is processed fast. If TLB miss occur often, the program will run more slowly and lead to more memory access and are quite costly. Because the hardware must have for aware definitely "where" the page tables are found in memory, the hardware would move the page table to find the correct page table entry and select the desired translation then update the TLB with the translation and retry the instruction.

The hardware raises an exception for northing have to do much on a miss. And then, the OS TLB miss handler performs the rest. So, secondly, we discuss TLB control flow algorithm (OS Handled). That algorithm shows the software-managed TLB which allows OS to use any data structure to implement the page table without requiring a change in hardware. Software-managed TLB is used in MIPS architecture. When the hardware produces an exception on a TLB miss, the software-management TLB produces the privilege level to kernel mode and starts to a trap handler. In the OS which handles TLB miss contain a code that specify the trap handler. When the code run, it will see the translation in the page table and take special the privileged instructions to update the TLB and return from the trap handler. At this point, the hardware retries the instruction that will result in a TLB hit. Besides, the hardware must resume execution that caused the trap and this retry thus allows the instruction run again, this time resulting in a TLB hit. So, the OS wants to be further careful by eliminating an infinite chain of TLB misses to occur.

Finally, we briefly describe that uses MIPS R4000 TLB entry. We assume that the MIPS R4000 supports a 32-bit address space with 4KB pages. We expect a 20-bit VPN and 12 bit offset in our typical virtual address. TLB turns out only 19 bits for VPN, the reset reserved for the kernel. The VPN translates to up a 24-bit frame number (PFN), so can support systems with up to 64 GB of main memory (2^{24} 4KB pages).

We explain a few other interesting bits in the MISP TLB. A global bit (G) is used for pages that are globally-shared among processes. The global bit is set by ignoring the ASID. In figure (5), show the 8-bit ASID which the OS can use to distinguish between the address spaces. Next, Coherence (C) bits determine how a page is cached by the hardware; a Dirty (D) bit is marked when the page has been written to; Valid (V) bit inform the hardware when there is a valid translation instant in the entry. In figure (5), some of the 64 bits are unused. MIPS TLBs usually have 32 or 64 of these entries. Most of these are used by user processes as they run. A few are reserved for the OS. Where a TLB miss would be problematic, the OS uses these reserved mappings for code and data that it wants to access during critical times. Because MIPS TLB that is software management needs to be instructions to update the TLB.

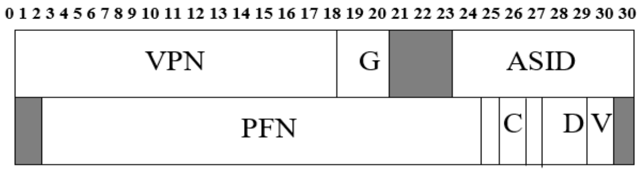

Fig. 5: A MIPS TLB Entry

4. 2 TLB Handling with a Tiny Address Space

In this section, we examine a simple virtual address trace and make the operation of a TLB to test how it can develop its performance.

We assume that have a tiny address space: 8-bits, with 16-bytes pages. So a virtual address contains two components:

VPN = 4 bits =16 pages
Offset = 4 bits

Let's assume further that we have an array of 10 4-bytes integers in memory, starting at virtual address 100. Accordingly, we must use a simple loop that accesses each array element, something that would look like this in C:

```
int sum = 0;
for (i = 0; i < 10; i++) {
      sum + = a [i];
}
```

Figure (6) shows the array laid out on the 16 16-bytes pages of the system. The first's array entry a[0] will begins on VPN = 06 and Offset = 04.

Offset	00	04	08	12	16
VPN = 00					
VPN = 01					
VPN = 02					
VPN = 03					
VPN = 04					
VPN = 05					
VPN = 06		a[0]	a[1]	a[2]	
VPN = 07	a[3]	a[4]	a[5]	a[6]	
VPN = 08	a[7]	a[8]	a[9]		
VPN = 09					
VPN = 10					
VPN = 11					
VPN = 12					
VPN = 13					
VPN = 14					
VPN = 15					

Fig. 6: An Array in a Tiny Address Space

When the first array element a[0] is accessed, the CPU will load to virtual address 100. The hardware selects the VPN from VPN = 06 and serves that to make sure the TLB for a valid translation. If the program accesses the array, the result will be a TLB miss. But the next two accesses are hits because we get on the same page. If the program accesses a[3], we meet another TLB miss. But the next two entries will hit in the TLB once again because of all reside on the same page in memory. Consequently, the program access to a[7] causes one last TLB miss. Next time the hardware examines the page table to get the location of virtual page in physical memory and updates the TLB. The final two accesses receive the benefits of this update. Thus, we can access the hit rate for the array: with 7 hits and 3 misses, the hit rate is 70%. If a machine we have a tiny address space 4KB, the pages will be 256 bytes pages. So the TLB hit rate is high.

5. CONCLUSION

In this paper, we explain the concept of the TLB misses handling with its need and role in problematic. We described how hardware and software can help us make address translation faster. In this paper, we explain with a simulation that handling of TLB miss may be implemented in hardware or software. The hardware management of the TLB entry structure is transparent to the software which grants using different processors until maintain software compatibility. The hardware-managed TLBs are used in x86 architecture that is designed to operate with very low latency and completely in hardware. Software-managed of TLBs simplifies hardware design and allow OS to use it wants to implement the page table without necessitating the hardware change. However, the software-management also incurs larger penalty. The MIPS architecture enables reliable and cost effective that is use software-managed TLBs.

6. ACKNOWLEDGEMENT

I would like to describe my sincere thanks to Dr. Aye Aye Chaw, Professor, Head of Faculty of Computer Science, University of Computer Studies (Mandalay) for her helpful recommendations and suggestions. Families and friends are acknowledged for their encouragement and unyielding support. Reviewers and experts are also shown gratitude for their assessments, contributions and comments towards this paper.

7. REFERENCES

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, "Operating System Concepts: Sixth Edition", John Wiley & Sons, Inc. 605 Third Avenue, New York, 2002.
[2] D.Agrawal, Memory Management Term Paper Operating Systems CS-384, February 2, 2003.
[3] J. Heinrich, "MIPS R4000 Microprocessor User's Manual: Second Edition", MIPS Technologies, Inc., 2011 N. Shoreline Blvd., Mountain View, CA 94039-7311, 1994.
[4] M. Agrawal and M. Jailia, Effect of TLB on System Performance, ICTCS'16, Udaipur, India, and March 04-05, 2016.
[5] R H. ARPACI-DUSSEAU and A C. ARPACI-DUSSEAU, "Operating Systems: Three Easy Pieces", Univ. of Wisconsin-Madison, August, 2018 (Version 1.00).
[6] V. Karakostas, Improving the Performance and Energy-efficiency of Virtual Memory: A Range-Based Approach, April 2016.
[7] W. Stallings, "Operating Systems Internals and Design Principles: Seventh Edition", Prentice Hall, 1 Lake Street, Upper Saddle River, New Jersey, 2012.
[8] Y.I. Klimiankou, Translation Lookaside Buffer Management, UDC 004.451.3, April, 2019.
[9] http://en.wikipedia.org/wiki/Virtual_memory
[10] http://en.wikipedia.org/wiki/Memory_management_unit
[11] http://www.techopedia.com/definition/5334/x86-architecture