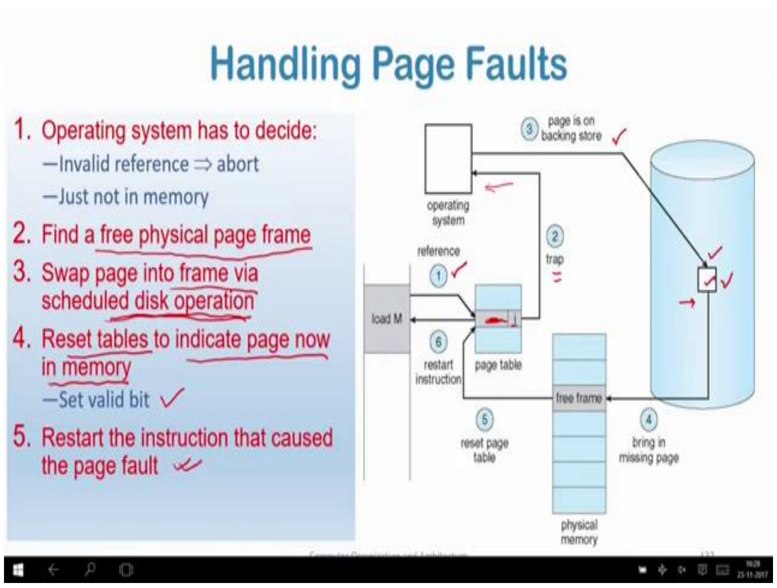
(Refer Slide Time: 23:09)



Handling Page Faults

1. Operating system has to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Find a free physical page frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   - Set valid bit ✓
5. Restart the instruction that caused the page fault

So, now we will take a deeper look into page faults. So, during a page fault when I do not have the required data in memory, I incur a page fault. So, at that time the page table entry corresponding to corresponding the page table entry corresponding to the page that I want to access shows that it is invalid. That is the valid bit is 0; that means, the corresponding physical page number is not mapped to the page table entry.

In that case what happens is that; firstly, I need to decide whether this translation that I want to do is for an invalid reference or for a valid reference. If it is for invalid reference, that means, that this particular virtual address is not present in my virtual address space itself.

This can happen for scattered virtual address spaces of a process. If this virtual address is not part of the address space of the process itself, then this is an invalid reference and we immediately abort. Otherwise we see that the valid bit is 0 and therefore, the page is just not in memory. So, therefore, it has to be brought from the disk or the secondary memory.

So, then what happens? In I have to find a physical page frame; I need to find a physical page frame. So, if you see the sequences that are happening here I try to reference in number 1, I try to reference and I see that the corresponding mapping is not there; it is it is not there. Therefore, I will have to trap the OS; I will have to trap the operating system to indicate that this is a page fault.

You know the operating system we will first find out what kind of trap has it received. So, it will then find that this is a page fault type of a trap. And therefore, then what the OS will do? It will find a physical page frame in the physical memory, it will find a physical page frame in the physical memory and then maybe through replacement of an existing page in physical memory which we will not bother right now we will do we will see bit later. Then it will swap page into this frame via scheduled disk operation.

So, now, when I have found a free page frame in physical memory I will need to bring the page that I require from the backing store or the secondary memory into the physical memory. And I said that how do we get that? In a earlier class we said that in addition to the mapping of a virtual memory corresponding to virtual memory and a physical memory, physical page number the I also the page table additionally logically maybe in a different physical table.

It also contains where in the backing store I have this page I have this page; that means, corresponding to a virtual page number the page table in addition to keeping the physical page number, it also keeps the address of where in the secondary memory this page is resident.

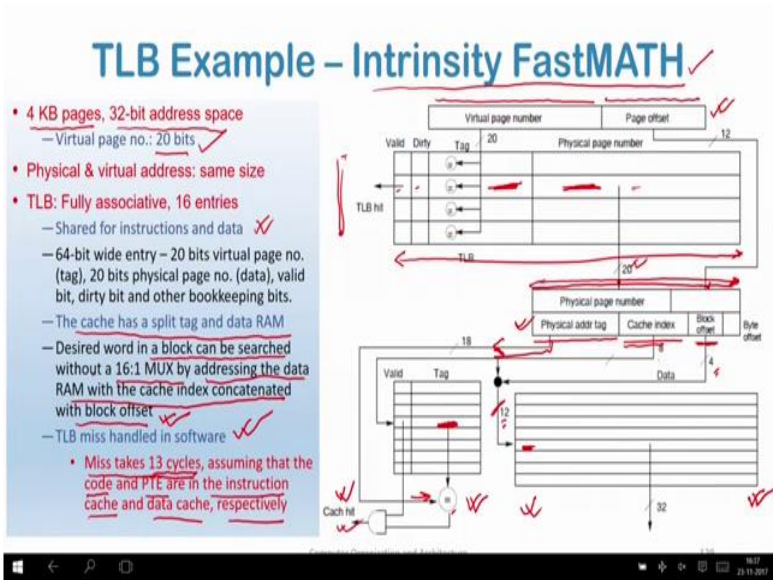
So, when I get a trap the OS finds out firstly, the physical frame where which is free and where I can give get the page into the physical memory where I can bring it and then it finds out where in the in the secondary memory the page is corresponding to this virtual page number.

Then it gets the page from the secondary memory through a scheduled disk operation. So, this schedule disk operation means that for the disk I have a queue of a of requests I have a queue of requests who want to access pages from the disk and I have to incur I have to go through this queue and get it from the secondary memory and I have to incur seek time and less latency time of the disk. And then through this disk operation I will swap in this page into the physical memory.

After I have brought in this page I know the page number. So, therefore I have to reset the tables, I have to reset the tables to indicate that the page is now in memory. So, I have to I have to update the page table entry corresponding to this page, corresponding to this virtual page, I will update the page table entry and write where what is the current physical page number corresponding to this virtual page number. I will also set the valid bit to indicate that the virtual this virtual page is currently now in physical memory ok, I will set the valid bit.

Then I will restart the instruction that caused the page fault. So, I will then restart the instruction. So, in a particular memory access I had a page fault, then I service the page fault, then I will restart the memory and then in the subsequent memory reference I will get the data that I sought for in the physical memory.

(Refer Slide Time: 29:00)



TLB Example – Intrinsity FastMATH

- 4 KB pages, 32-bit address space
  - Virtual page no.: 20 bits
- Physical & virtual address: same size
- TLB: Fully associative, 16 entries
  - Shared for instructions and data
  - 64-bit wide entry – 20 bits virtual page no. (tag), 20 bits physical page no. (data), valid bit, dirty bit and other bookkeeping bits.
  - The cache has a split tag and data RAM
  - Desired word in a block can be searched without a 16:1 MUX by addressing the data RAM with the cache index concatenated with block offset
  - TLB miss handled in software
    - Miss takes 13 cycles, assuming that the code and PTE are in the instruction cache and data cache, respectively

Now, we will take an example of a practical architecture which the TLB of a practical architecture ok. So, we will take a look at the Intrinsity FastMATH architecture and in this architecture we have a 4 KB pages and 32 bit virtual address space.

So, if we see here in this virtual page number we see that it has a 20 bit virtual page number and 12 bit page offset; which means that see this page size is 4 KB 12 bits. Why? Because right for $2^2 \times 10$ KB. So, basically I have to using 12 bits I have I have a page offset of 12 bits and so, the page size is 4 KB.

And because I have a 32 bit address space, the virtual page number is 20 bits. Now, here we see that the physical page number, the physical page number here because this one is also of 20 bits, this one is also of 20 bits; the physical page number and the physical the physical address space is also of the same size as the virtual address space. The physical page physical address is also 32 bits ok.

So, the TLB is fully associative; as we as we told what it means it is fully associative and it has 16 entries in the TLB. So, the TLB is fully associative having 16 entries and this TLB is shared for instructions and data. So, it is a translation look aside buffer for both instructions and data. Each TLB entry is 64 bits wide. So, this one is 64 bits wide and it contains the 20 bit virtual address. So, this part is 20 bits; it contains the sorry 20 bit virtual page number. So, this one is 20 bits; it contains the 20 bit physical page number. So, this one is again 20 bits; it contains the valid bit, dirty bits and other bookkeeping bits. So, the total address so, the total size of one page table entry is 64 bits.

If you note the cache is split; cache has a split tag and data part. So, basically what happens? When there is a TLB hit, when there is a TLB hit I get the corresponding let us say the TLB the virtual page number matched in this entry and I got the corresponding physical page number; the 20 bit physical page number and therefore, I generated the physical address here. After generating the physical address I divide the physical address into 3 parts, I split it. One is the physical address tag part, the cache index part and the block offset part; so to access what? To access the cache ok.

Now, here now subsequently after dividing this physical address into splitting it into three parts; the tag part, the cache index part and the block offset part I will access the cache. And this cache is implemented again as a split tag and data. So, what happens here? The tag part if you see here, the tag part is the physical tag part is matched with the tag part of the cache. The tag part of the cache is here, I match it with the tag part of the physical address, I match it with the tag part of the physical address here, I match it and then if the valid bit is on I get a cache hit ok.

Now, at the same time because I have now split the data part of the cache what do I do? Why do have I done that? I now am by splitting it I now am able to concatenate the cache index part and the block offset part as a 12 bit value and this 12 bit using these 12 bits I directly access the data date data word that I require; I directly access the word. So, if I did not concatenate this, so, what would I have to do? I would have to access the given block and within that block I have to do because the block offset is 4 contains 4 bytes; which means that I have 16 words within each block.

Now, I have to I had to compare this the 16 words; I had to compare because each I had to compare these 16 words and using a 16 × 1 multiplexer to find out which data word I actually need. Now, what happens? I have the tag part; I have the tag part and the block offset part.

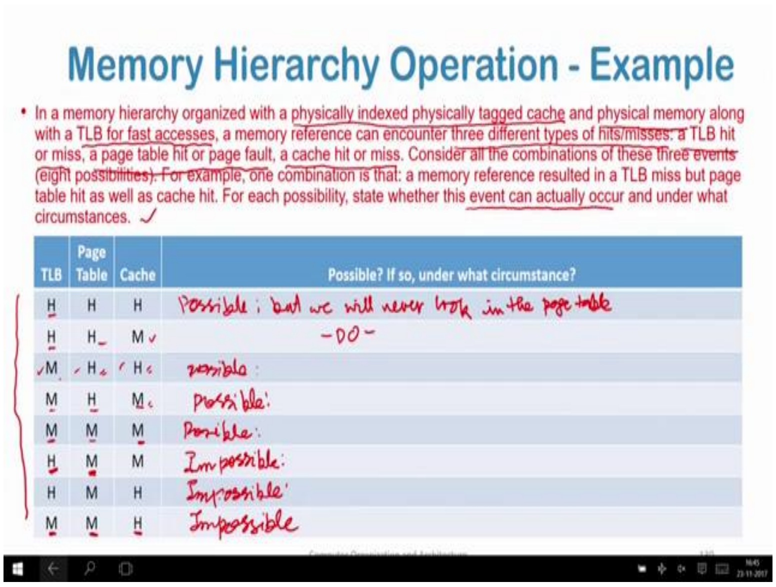
So, basically I have got the exact data word. I don’t have to do this, I don’t have to go into the block and then using a 16 × 1 MUX and the block offset, I don’t have using the using the block offset as the select line, I don’t have to use the 16 × 1 MUX with block offset as the select line to and to know which exact word within this block should I access. I don’t have to do this. I do not have to use a 16 × 1 MUX with the block offset as the select line to know which word within the block do should, I correctly access.

Instead I have concatenated the cache index part and the block offset part to generate a 12 bit value and I directly access the data I directly access this data RAM part of the cache and get the required word and I know that the required word is correct because I have a cache hit here ok. So, the desired word in a block: the cache has a split tag and a data RAM. The desired word in a block can be searched without a 16 × 1 MUX by addressing the data RAM with the cache index concatenated with block offset ok.

Now, in this architecture a TLB miss is handled in software. So, how do I handle this? If I have a TLB miss what do I do? I take the virtual page number and I save it you know hardware register. Then I trap the OS and say that I have a TLB miss. So based on this, the OS generates special instructions to go into to find the page table entry using the page table base register and the virtual page number part; virtual page number part and the page table base register combination. It gets into the page table entry and brings the required page table entry.

Now, the page in a TLB miss requires only 13 cycles in this system when we when we consider when we assume that the code and the page table entry are in the instruction and instruction cache and data cache respectively. So, this TLB miss has an overhead of only 13 cycles when the code and the TLB entry are in the instruction cache and in the data cache.

(Refer Slide Time: 37:19)



Memory Hierarchy Operation - Example

- In a memory hierarchy organized with a physically indexed physically tagged cache and physical memory along with a TLB for fast accesses, a memory reference can encounter three different types of hits/misses: a TLB hit or miss, a page table hit or page fault, a cache hit or miss. Consider all the combinations of these three events (eight possibilities). For example, one combination is that: a memory reference resulted in a TLB miss but page table hit as well as cache hit. For each possibility, state whether this event can actually occur and under what circumstances. ✓

TLB	Page Table	Cache	Possible? If so, under what circumstance?
H	H	H	Possible ; but we will never look in the page table
H	H	M ✓	-DO-
✓M	✓H	H	possible :
M	H	M	Possible:
M	M	M	Possible:
H	M	M	Impossible:
H	M	H	Impossible'
M	M	H	Impossible

Now, we will continue our discussion and take an example here to understand to illustrate as to how the memory hierarchy works in unison together. How they cooperate among each other and what happens when they work all together? So, memory hierarchy in operation, the example goes like this.

In a memory hierarchy organized with a physically indexed physically tagged cache and physical memory along with the TLB for fast accesses. So, what I have? I have a physically indexed physically tagged cache; meaning that the cache is addressed by the physical address by the physical address.

It is not using virtual addresses to address the cache. As opposed to other we will see over a technique in the next class, two techniques in the next class; where part of the full or part of the virtual address is used to address the cache. Now, in this hierarchy this does not happen. So, I generate the physical address and then from the physical address I address the cache.

So, it is a physically indexed physically tagged cache. So, I index the cache using the part of the physical address and I tag the cache using part of the physical address. So, in a I have a memory organized with a physically indexed physically tagged cache and physical memory along with. So, I have a cache and I also have a physical memory along with a TLB for a fast accesses. So, a memory reference can encounter three different, in this system a memory reference can encounter three different types of hits or misses. It can encounter a TLB hit or miss, it can encounter a page table hit or miss or and a cache hit or miss.

So, consider all the combinations of these three events. So, you have 8 possibilities, you have 8 possibilities. For example, one combination is that a memory reference resulted in a TLB miss, but a page hit and a so, I have a TLB miss, but I have a page hit and a cache hit. For each possibility we need to say whether this event can actually occur? Whether this event can actually occur?

Whether it is possible in practice that these that that are given three even that they get that an event of a combination of hit and a hit or miss will actually occur. And if so, under what circumstances can it occur? For example, can we have a hit in the TLB, a hit in the page table and a hit in the cache? Yes we can.

But obviously, when I have a hit in the TLB what happens? When I have a hit in the TLB, so this is possible, this is possible. But, but we will never look in the page table, page table. Because I have a TLB hit because I have a TLB hit we will never look in the page table ok. So, and then I have when I have got it in the page table, when I have got it in the page table I have a cache hit. Then this is possible. So, the page table hit I got and then I got the physical page number and after I got the physical page number, I got the data subsequently in cache.

Now, I have a TLB hit and a page table hit and a cache miss this is also possible; DO. But what happens? This is possible, but obviously, I will not check the page table because I have already got a hit in the cache. So, I will not go get into memory and look actually, look into the page table because I have a hit in the TLB. Now when I have a miss in the TLB, yes I can have a miss in the TLB a possible, possible. So, I can have a miss in the TLB and then I will I will go into the page table lower level of hierarchy and find out whether the page table entry is there or not.

And if I have a I then basically here it says that I have a page table hit. If I have a page table hit then basically it is in memory and if it is in memory it may or may not reside in cache. In this case I am saying that it is a hit in the cache. So, it is in the hit in the page table. Therefore, the data is in memory and it is a hit in the cache, so the data is also in the cache.

So, this is possible. The next case I have a miss in the, I have a miss in the page I have a miss in the TLB. I have a hit in the page table and I have a miss in the cache. Yes, this is also possible as we said I have a miss in the page table. So, I go to get into the next level, get the page table entry. I will find that the page; for the page table entry the day the actual page actually resides in physical memory.

However, when I generate the when I generate the physical address break it into a tag, cache index, block offset, etcetera for accessing the cache; I see that the data is not present in cache. So, therefore, I have to incur a memory reference physical memory access and get the data from physical memory, I will not get it from the cache directly. It is also possible that I will have a miss in the TLB, subsequently, I will l look into the page table and I see that the data is not in the page table.

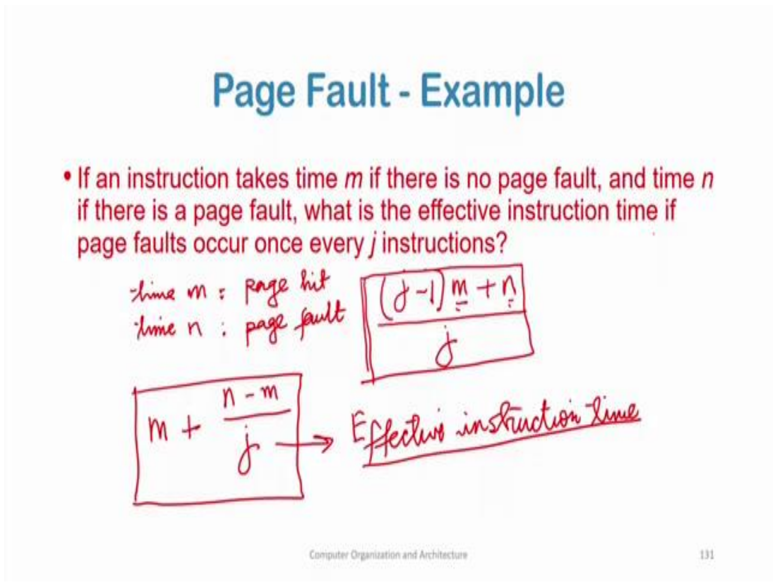
So therefore, I will basically the data does not reside in the in the main memory and I will also have a miss in the cache. So, this is also possible. This is also possible. So, I will obviously, because there is a page table miss I will also have a miss in the cache. it cannot be that my page it is there it is not there in the main memory or data is not there in the main memory, but the data is there in the cache; this cannot happen.

Now, let us see the next case. I have a hit in the page table, but I have a miss in I have a hit in the TLB, but I have a miss in the page table. Is this possible? This is not possible because if I have a hit in the TLB, TLB acts as a cache for the page table. So, if I have a hit in the TLB, the data must also be there in the page table and the entry must also be there in the page table. So, it is not possible that I have a so, I have a hit in the TLB, but I have a miss in the page table.

Similarly, this also this is also impossible. It cannot be that I have a hit in the TLB and a miss in the page table. I have a miss in the page I have a miss in the TLB, I have a miss in the page table and but I have a hit in the cache. Is this possible? This is also impossible. Why? Because; I have a miss in the page TLB fine; so, therefore, I have a miss in the page table; this part is fine.

However, I have a miss in the page table means that I the data I have a page fault. I have a miss in the page table means I have a page fault. So, the page is not resident in main memory. So, when a when a particular page is not resident in main memory, the corresponding block cannot reside in the cache. So, this is impossible. So, we will end this lecture with a small example.

(Refer Slide Time: 45:43)



Page Fault - Example

- If an instruction takes time *m* if there is no page fault, and time *n* if there is a page fault, what is the effective instruction time if page faults occur once every *j* instructions?

time m : page hit
time n : page fault

$$\boxed{\frac{(j-1)m+n}{j}}$$

$$\boxed{m+\frac{n-m}{j}} \rightarrow \underline{\text{Effective instruction time}}$$

Computer Organization and Architecture
131

If an instruction takes time m if there is no page fault and time n if there is a page fault, what is the effective instruction time if page faults occur every j instructions? So, we see that time m when page hit, I have a page hit and time n when I have a page fault ok. So now, I am saying that in a group of j instructions I have one miss and the all other are hits. So, in this system page faults occur once every j instruction.

So, for $j-1$ instructions, I have the time required is m and for 1 instruction the time required is n. So, every j instructions what happens is that for $j-1$ instructions, I take m time and for one other instruction I take n time ok. So, therefore the effective instruction time; So, what is the average instruction time? Then the average instruction time is given by this. So, this is basically m plus what? $m+\frac{n-m}{j}$. So, this is the effective instruction time.

With this we come to the end of this lecture.