❖ AIM

Implementation of Translation Lookaside Buffer.

when the virtual to physical address translation happens, 2 memory accesses are required

1. First for the page table to get the physical frame number which is then combined with an offset (within the frame) to get the actual physical address of the memory location referenced by CPU.

2. Second for the access done to the desired memory location with the physical address computed above.

To improve upon the effective memory access time, a fast cache or lookup service is implemented in hardware -- This is known as TLB or Translation LookAside Buffer. It serves as a cache for the virtual to physical page mappings stored in page table (in DRAM). The virtual address is first presented to the TLB to see if the mapping exists.

❖ THEORY

A translation lookaside buffer (TLB) is a memory cache that stores recent translations of virtual memory to physical addresses for faster retrieval.

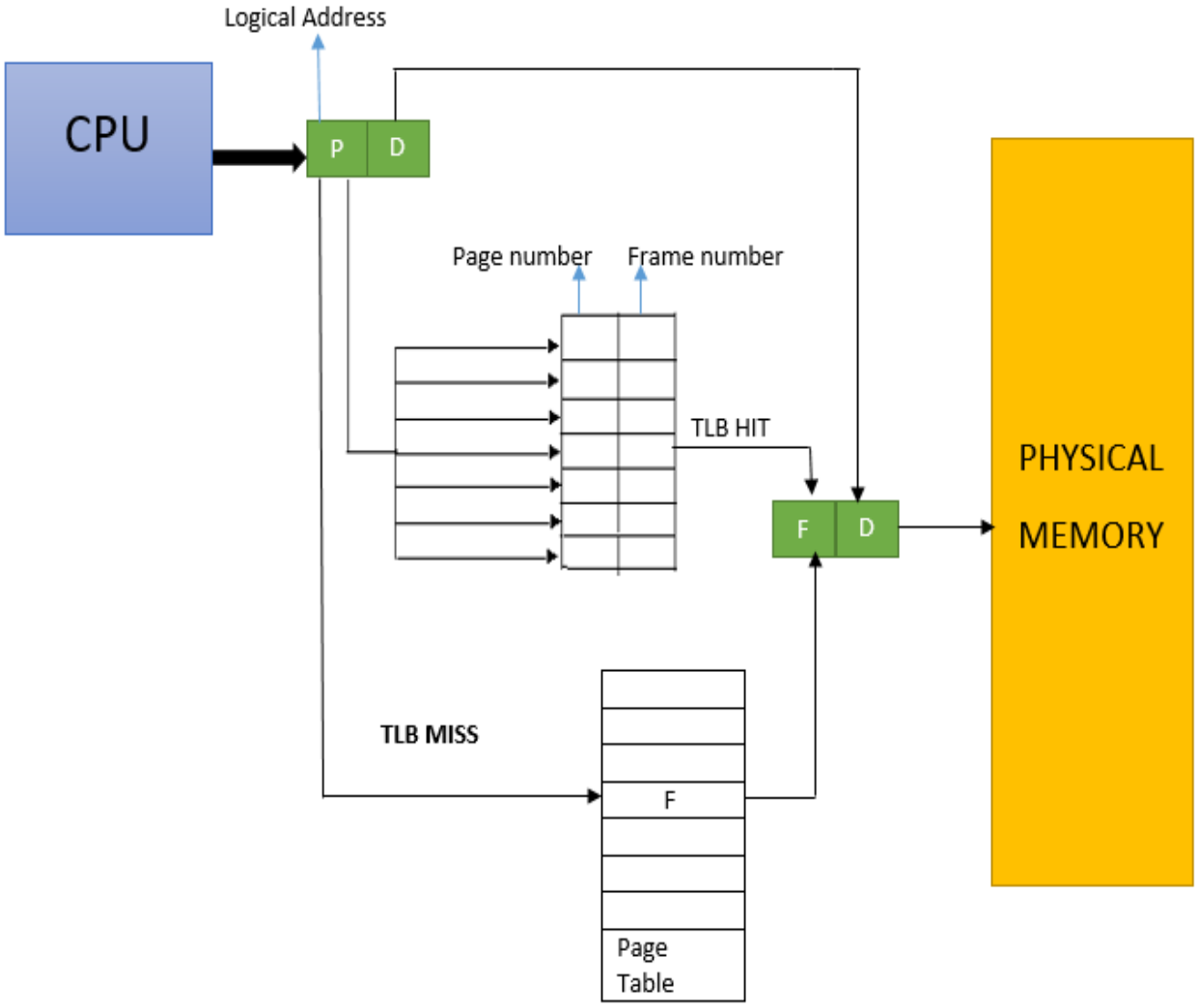
When a virtual memory address is referenced by a program, the search starts in the CPU. First, instruction caches are checked. If the required memory is not in these very fast caches, the system has to look up the memory's physical address. At this point, TLB is checked for a quick reference to the location in physical memory.

When an address is searched in the TLB and not found, the physical memory must be searched with a memory page crawl operation. As virtual memory addresses are translated, values referenced are added to TLB. When a value can be retrieved from TLB, speed is enhanced because the memory address is stored in the TLB on processor. Most processors include TLBs to increase the speed of virtual memory operations through the inherent latency-reducing proximity as well as the high-running frequencies of current CPU's.

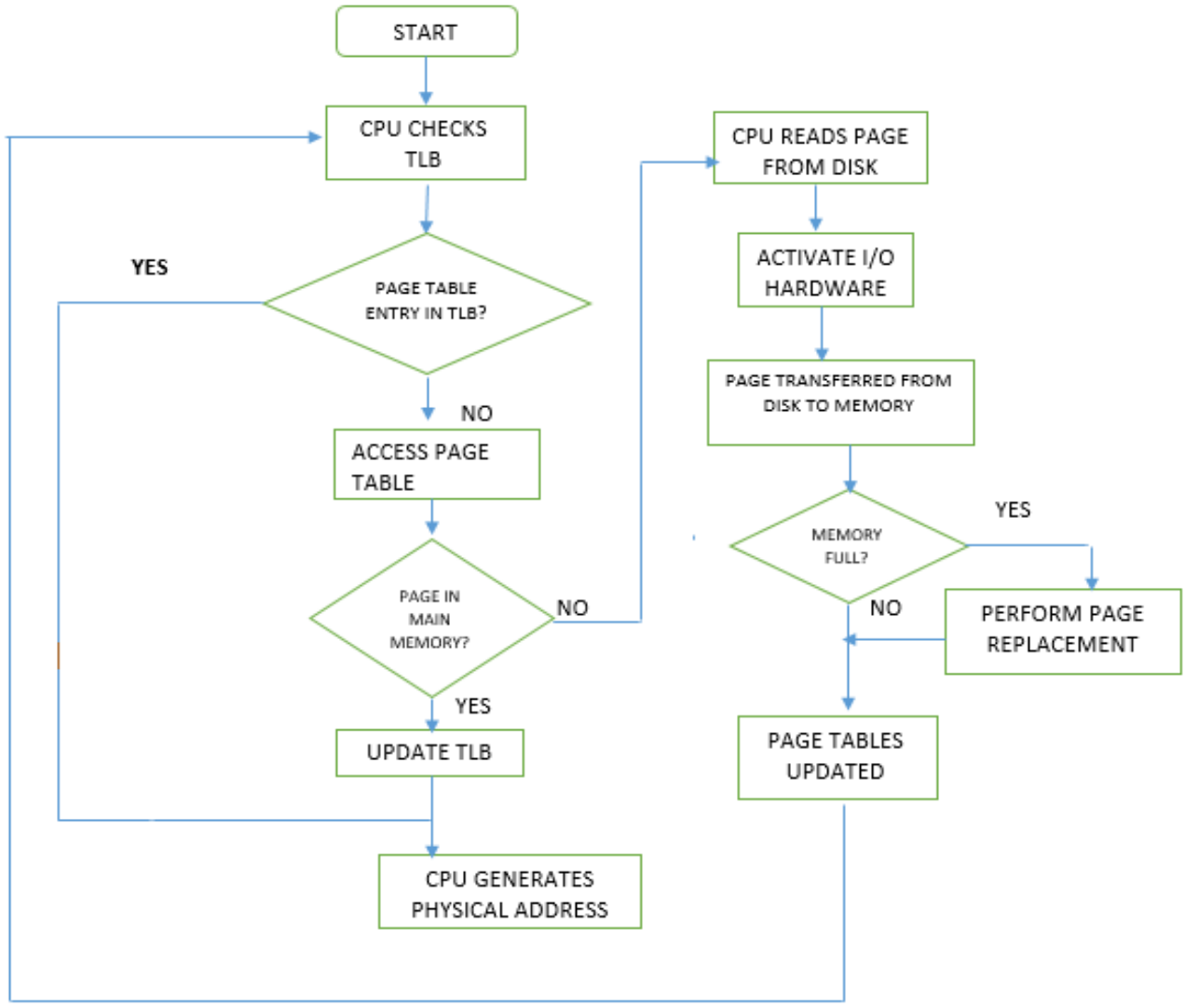
TLBs also add the support required for multi-user computers to keep memory separate, by having a user and a supervisor mode as well as using permissions on read and write bits to enable sharing.

TLBs can suffer performance issues from multitasking and code errors. This performance degradation is called a cache thrash. Cache thrash is caused by an ongoing computer activity that fails to progress due to excessive use of resources or conflicts in the caching system.

Block Diagram of TLB



Flow-Chart of TLB



The flowchart provided explains the working of a TLB. If it is a TLB miss, then the CPU checks the page table for the page table entry. If the present bit is set, then the page is in main memory, and the processor can retrieve the frame number from the page-table entry to form the physical address. The processor also updates the TLB to include the new page-table entry. Finally, if the present bit is not set, then the desired page is not in the main memory, and a page fault is issued. Then a page-fault interrupt is called, which executes the page-fault handling routine.

If the page working set does not fit into the TLB, then TLB thrashing occurs, where frequent TLB misses occur, with each newly cached page displacing one that will soon be used again, degrading performance in exactly the same way as thrashing of the instruction or data cache does. TLB thrashing can occur even if instruction-cache or data-cache thrashing are not occurring, because these are cached in different-size units. Instructions and data are cached in small blocks (cache lines), not entire pages, but address lookup is done at the page level. Thus even if the code and data working sets fit into cache, if the working sets are fragmented across many pages, the virtual-address working set may not fit into TLB, causing TLB thrashing. Appropriate sizing of the TLB thus requires considering not only the size of the corresponding instruction and data caches, but also how these are fragmented across multiple pages.

❖ PROPOSED SYSTEM

Virtual address is managed by Operating System whereas the physical address are associated with hardware. So say a process is running and requests to load something or store something. It will give CPU a virtual address. (It should be noted that the translation of this Virtual address to Physical address is managed by OS in form of Page Tables. These tables are stored in RAM.) Now you need to load/store the data from/to memory. But wait... we don't have physical memory, rather we have virtual memory. So should we go to main memory and look up the page tables to get the physical address corresponding to this virtual address. The answer is obviously NO. (And if you are thinking WHY, then just think of reason of implementing caches, what would happen if we go to RAM for every translation, what will be overhead)

So the smart people said "Why not make a small cache which stores the most commonly used translation mapping". So here was the birth of TLB. So now when your virtual address comes then small part of it is used to index the TLB and rest is used as tag (virtual tag) and it gives you the physical address. There are many other optimizations done to speed up operation like using VIPT - Virtually Indexed Physically Tagged Cache).

We have used Lists Data Structures and its Fuctions on Python Platform.

❖ IMPLEMENTATION

```
import pyqt file
import random as r
tlb=[]
pagetable=[]
existpage=[]
exsitinst=[]
print('Implementation of TLB for a single process without Context Switching.\nIntructions:')
print('We consider only 1 process here.\nTime taken to access memory=100 micro secs.\nTime taken to access
TLB is 20micro seconds.\nWe consider that each page has 50 instructions in it.')
print('We use Page size as 64')
n=32
timetlb=20
timemem=100
np=n*2
ni=32
time=0
print('Number of pages for the process=',np)
pagetable=[[x,r.randrange(100,800),-1] for x in range(np)]
print('\nPrinting Page Table:')
print('---------------------------------------')
for index,items,k in pagetable:
        print('Page No:',index,'\tFrame No:',items)
print('---------------------------------------')
print('Starting execution process:\n')

'''
for i in range(np):
        for j in range(ni):
                if (i,i+100) in tlb:
                        time+=timetlb
                        print('Page=',i,' presrent in TLB.(Hit)')
                        time+=timemem
                else:
                        time+=timetlb
                        if len(tlb)==n:
                                del(tlb[0])
                                print('FIFO used to dump first inserted Page')
                                print('\nUpdated TLB:')
                                print('---------------------------------------')
                                for index,items in tlb:
                                        print('Page No:',index,'\tFrame No:',items)
                                print('---------------------------------------')
                        tlb.append(pagetable[i])
                        print('Page=',i,' not present in TLB, therefore added!(Miss)')
                        print('\nUpdated TLB:')
                        print('---------------------------------------')
```

```
                        for index,items in tlb:
                                print('Page No:',index,'\tFrame No:',items)
                        print('----------------------------------------')
                        time+=timemem
                        time+=timemem



'''
while True:
        x=set([i[2] for i in pagetable])
        if len(x)==1 and 31 in x:
                break
        page=r.randrange(0,np)
        instruction=r.randrange(pagetable[page][2],32)
        if page in [x[0] for x in tlb]:
                time+=timetlb
                print('Page=',page,' present in TLB.(Hit)')
                time+=timemem
        else :
                time+=timetlb
                if len(tlb)==n:
                        del(tlb[0])
                        print('FIFO used to dump first inserted Page')
                        print('\nUpdated TLB:')
                        print('----------------------------------------')
                        for index,items,k in tlb:
                                print('Page No:',index,'\tFrame No:',items)
                        print('----------------------------------------')
                tlb.append(pagetable[page])
                print('Page=',page,' not present in TLB, therefore added!(Miss)')
                print('\nUpdated TLB:')
                print('----------------------------------------')
                for index,items,k in tlb:
                        print('Page No:',index,'\tFrame No:',items)
                print('----------------------------------------')
                time+=timemem
                time+=timemem
        for i in range(instruction):
                if pagetable[page][2] == 31:
                        break
                pagetable[page][2]+=1


avtime=time/(np*ni)
avtime_without_tbl=2*timemem
print('\n\nTotal time of execution for all instructions without TLB=',avtime_without_tbl*np*ni)
print('Average time of execution of an instruction without TLB=',avtime_without_tbl)
print('\n\nTotal time of execution for all instructions with TLB=',time)
print('Average time of execution of an instruction with TLB=',avtime)
```

```
Implementation of TLB for a single process without Context Switching.
Intructions:
We consider only 1 process here.
Time taken to access memory=100 micro secs.
Time taken to access TLB is 20micro seconds.
We consider that each page has 50 instructions in it.
We use Page size as 64
Number of pages for the process= 16

Printing Page Table:
---------------------------------------
Page No: 0      Frame No: 771
Page No: 1      Frame No: 242
Page No: 2      Frame No: 417
Page No: 3      Frame No: 454
Page No: 4      Frame No: 523
Page No: 5      Frame No: 437
Page No: 6      Frame No: 289
Page No: 7      Frame No: 309
Page No: 8      Frame No: 210
Page No: 9      Frame No: 701
Page No: 10     Frame No: 776
Page No: 11     Frame No: 392
Page No: 12     Frame No: 656
Page No: 13     Frame No: 311
Page No: 14     Frame No: 630
Page No: 15     Frame No: 471
---------------------------------------
Starting execution process:
```

```
Page= 12  not present in TLB, therefore added!(Miss)

Updated TLB:
---------------------------------------
Page No: 12     Frame No: 656
---------------------------------------
Page= 12  present in TLB.(Hit)
Page= 11  not present in TLB, therefore added!(Miss)

Updated TLB:
---------------------------------------
Page No: 12     Frame No: 656
Page No: 11     Frame No: 392
---------------------------------------
Page= 2  not present in TLB, therefore added!(Miss)

Updated TLB:
---------------------------------------
Page No: 12     Frame No: 656
Page No: 11     Frame No: 392
Page No: 2      Frame No: 417
---------------------------------------
Page= 10  not present in TLB, therefore added!(Miss)

Updated TLB:
---------------------------------------
Page No: 12     Frame No: 656
Page No: 11     Frame No: 392
Page No: 2      Frame No: 417
Page No: 10     Frame No: 776
```

```
Updated TLB:
---------------------------------------
Page No: 2      Frame No: 417
Page No: 10     Frame No: 776
Page No: 7      Frame No: 309
Page No: 11     Frame No: 392
Page No: 3      Frame No: 454
Page No: 15     Frame No: 471
Page No: 1      Frame No: 242
Page No: 5      Frame No: 437
---------------------------------------
Page= 5  present in TLB.(Hit)
Page= 3  present in TLB.(Hit)
Page= 3  present in TLB.(Hit)
FIFO used to dump first inserted Page

Updated TLB:
---------------------------------------
Page No: 10     Frame No: 776
Page No: 7      Frame No: 309
Page No: 11     Frame No: 392
Page No: 3      Frame No: 454
Page No: 15     Frame No: 471
Page No: 1      Frame No: 242
Page No: 5      Frame No: 437
---------------------------------------
Page= 4  not present in TLB, therefore added!(Miss)

Updated TLB:
---------------------------------------
```

```
Updated TLB:
---------------------------------------
Page No: 2      Frame No: 417
Page No: 10     Frame No: 776
Page No: 7      Frame No: 309
Page No: 11     Frame No: 392
Page No: 3      Frame No: 454
Page No: 15     Frame No: 471
Page No: 1      Frame No: 242
Page No: 5      Frame No: 437
---------------------------------------
Page= 5  present in TLB.(Hit)
Page= 3  present in TLB.(Hit)
Page= 3  present in TLB.(Hit)
FIFO used to dump first inserted Page

Updated TLB:
---------------------------------------
Page No: 10     Frame No: 776
Page No: 7      Frame No: 309
Page No: 11     Frame No: 392
Page No: 3      Frame No: 454
Page No: 15     Frame No: 471
Page No: 1      Frame No: 242
Page No: 5      Frame No: 437
---------------------------------------
Page= 4  not present in TLB, therefore added!(Miss)

Updated TLB:
---------------------------------------
```

❖ CONCLUSION

Hence we have successfully implemented Translation Lookaside Buffer.

❖ REFERENCE

Book:Operating Systems:Internals and Design Principles.
Authors:William Stallings.
Publisher:Pearson.

Website:*WhatIs.techtarget.com.*