Set-1

PES1201800125 MPCA Assignment DATE PAGE

A1)

a. The diff between computer organization and computer architecture is as follows.

1) Computer architecture is mainly concerned with how hardware components are connected together whereas organization is concerned with the structure and behaviours of a computer as seen by the user.

2) The architecture is helps us understand the functionalities of the system while organization specifies the relevant interconnections and placing.

3) Architecture deals with high level design issues while orgnization deals with low-level (gate level) issues.

4) Architecture involves cache optimization, addressing modes, instruction sets etc while organization involves circuit design, adders, peripherals etc.

b.)

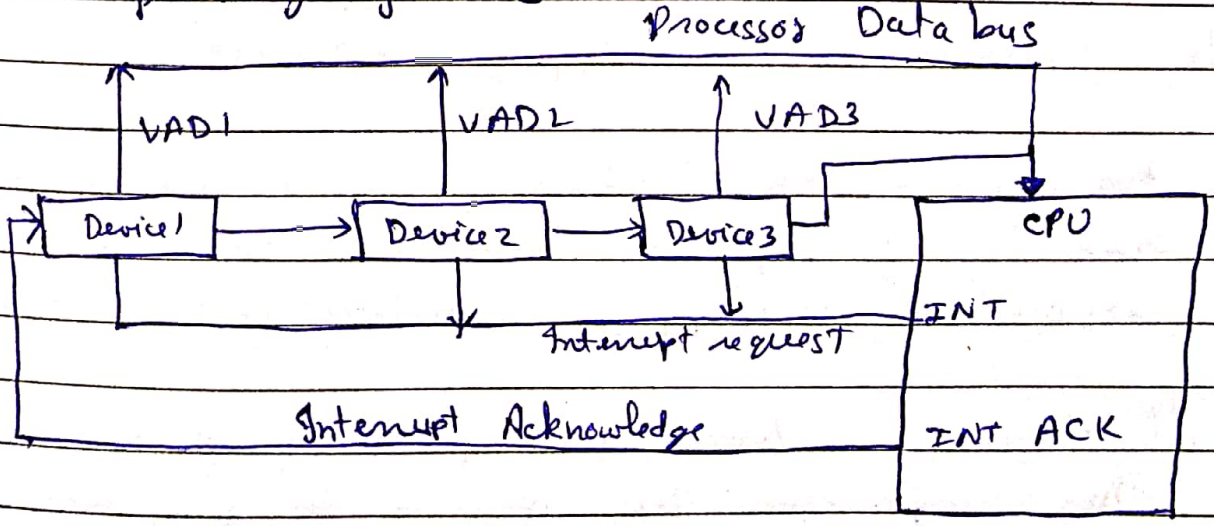
CISC	RISC
1. Emphasis on hardware	Emphasis on software.
2. Fewer registers	Uses more registers.
3. Pipelining is difficult	pipelining is easy
4. More addressing modes	Fewer addressing modes
5. Instructions take varying amount of cycle time	Instruction takes 1 cycle to execute.

6. Eg→ Motorola 68000, AMD, intel x86 CPUs	Eg→ MIPS, Intel i860, Atmel AVR etc

c.) ARM handles an exception in the following steps →

1) The Software Method - Polling
- In this method all interupts/exception are serviced by branching to the same service program. This program then checks with each device if it is the one generating interupt (acc. to priority). Once found, another device specific program is called to handle it.

2) The hardware Method - Daisy Chain
- It involves connecting all the devices that can request an interupt in a serial manner. This config. is set acc. to the priority of devices.



A2) Pipelining is needed for the following reasons→

1) To increase the CPU throughput
2) Decrease the total program execution time
3) In order to parallelize the sequential processing by overlapping the execution of instructions.

The goal of a pipeline designer is to design an effective pipeline which not only does the above stated tasks but also tries to avoid/take care of as many Data and Structural hazards as possible. Also to decide upon the type of pipelining to be employed in order to tackle the problem (3 or 5 stage).

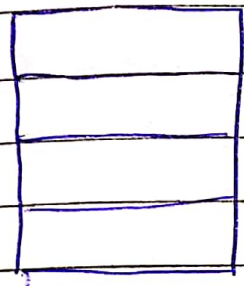
It is difficult to implement a pipeline because of the Pipelining Hazards→

1) Structural Hazards → They arise from resource conflicts when the hardware can't support all possible combinations of overlapping instructions.

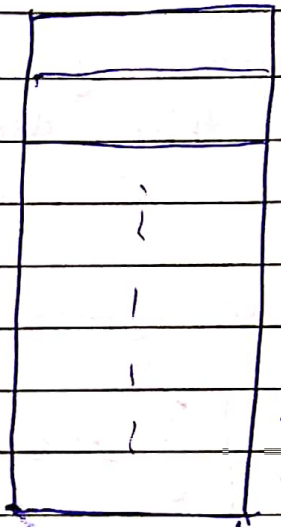
2) Data Hazards → Arise when an instruction depends on the results of a previous instruction in a way that is exposed by overlapping of instruction in pipeline.

3) Control Hazards → Arise from pipelining of branches and other instructions that change the program counter.

A3) Fully Associative mapping cache is faster than Direct mapping because

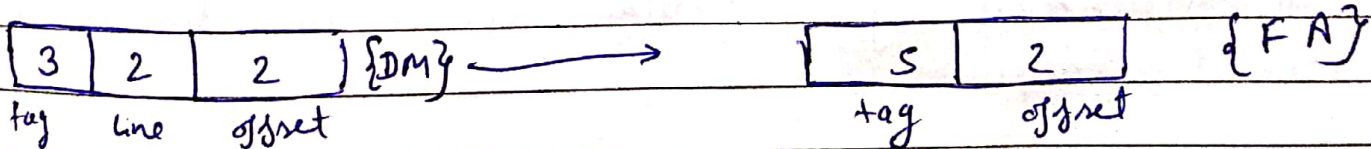


Now, assuming this case for all the example based explanation for the answers.

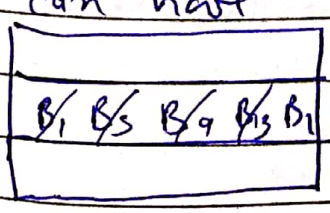


- When the cache is directly mapped then the Block say Bn is allowed only to be mapped to a particular line L~~x~~ n% size. Thus say if the CPU requests for blocks B_1, B_5, B_9, B_{13}, B_2 Then we will consecutively face cache miss where the tag when compared won't be the same. Thus we'll have to bring in the block from main memory and replace the one in cache

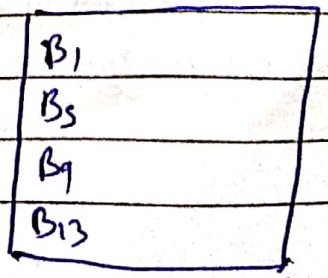
Now, when we use fully associative, in this case



Thus we can place any block anywhere, hence we can have



changed to →

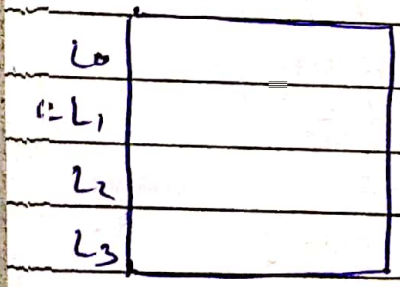


Thus increasing the cache hits.

That increase in the number of cache hits makes fully associative, faster than Direct Mapping.

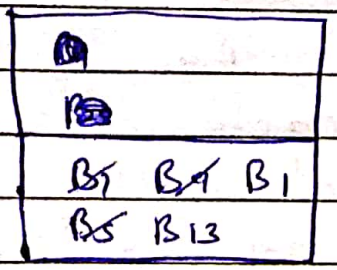
b) Now, moving further with explaining why set-assoc. is faster than direct mapping. Lets say we have 2-way set associative mapping. Then

$$\text{sets} = \frac{\text{no. of lines}}{k} = \frac{4}{2} = 2$$

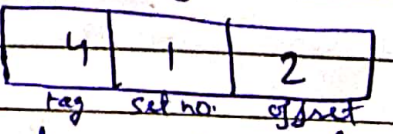


Now, for CPU call $B_1, B_5, B_4, B_{13}, B_1$

Then we have →



As we know the set is obtained by n % num sets and then the blocks are mapped just like associative mapping.

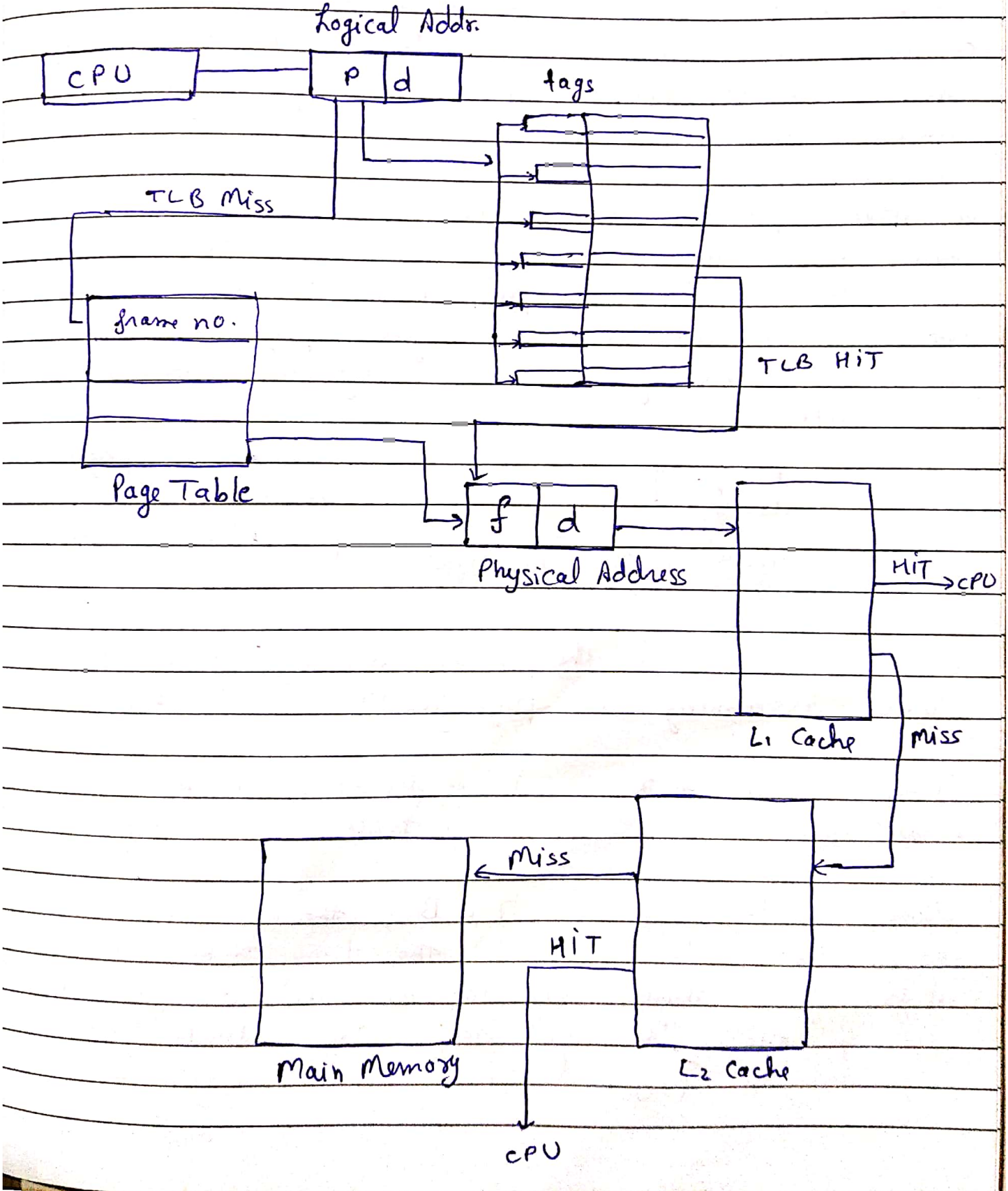


(here)

Thus as we can see the number of cache misses are lesser than the number of cache misses we obtained for similar scenarios but with direct mapping.

Since they misses are less we can say that the k-way set associative mapping is faster than direct mapping.

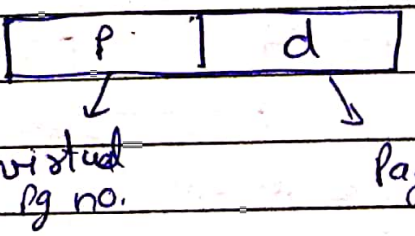
A4)



A translation look aside buffer (TLB) is a memory cache that is used to reduce the time taken to access a user memory location. The TLB stores the recent translations of virtual memory to physical memory and can be called an address-translation cache.

In ideal process we divide our process into pages and main memory into frames where the pages are stored. Now a page table is used to obtain the frame address for a given page.

Now, the CPU generates a logical address with



Now, the page as well as the page table both are present inside the main memory/ cache L_1 / L_2

Now, assuming no page fault occurs, then

Time = $x + x$ { time for accessing Pg. table + Accessing frame }

* ideal case

Now, with the use of TLB, since it is an address translation cache the time taken to access it is far lesser. The logical address generated by the CPU is sent to TLB to obtain the physical address.

The L_1 cache is checked for a match, if HIT occurs then the offset is used to read the corresponding word and that is sent to processor. In case of a miss, it is checked for in the L_2 cache. If no cache HIT occurs, the the main memory is checked for the same.
In case of a TLB miss, we refer the page table and then L_1, L_2, main memory in order which takes more time than the former method. Tags are used to check for a valid match in either case.

First Simplification →

- L_1 cache is a split cache, it requires two TLBs
- One cache and TLB is for instructions; driven from PC
- One cache and TLB is for data; driven from effective address.

Second Simplification →

All caches and TLBs are direct mapped. If associative is used, it would replicate each set of comparators, tag memory & data memory.
Connect data memories with a n:1 MUX to select a hit. If total cache size remained same, the cache index would shrink by $\log_2 n$ bits.

A5)

a. Data Parallelism → It means parallelization across multiple processors in parallel computing environments. It mainly focusses on distributing the data across different nodes, which operate on the data in parallel. It can be applied on regular data structures like arrays and matrices by working on each element in parallel.
Only one execution thread is operating on all the subsets of data thus the speed up is more. The amount of parallelization in such case is proportional to the input data size. Thus it helps in optimal load balancing on multiprocessor system. by the use of synchronous computation.

b. Task Parallelism → It means parallelization of the computer codes across multiple processors in parallel computing environments. Task parallelism focuses on distributing tasks, concurrently performed by processes or threads across different processors.
In such parallelism, different tasks may be performed on the same or different sets of data asynchronously. The amount of the parallelization is proportional to the number of independent tasks. Here the load balancing depends on the availability of hardware and job scheduling algorithms.

c. Function Parallelism → This kind of parallelism is mostly similar to the task parallelism. It is based on different functional blocks in our program / application. The application / program is split into separate processing units, that communicate with a fixed number other units in such a way that the output of one part serves as the input of another. It's more of a pipelining concept that has been adapted to reduce the computing time.