By 3220104364 王晓宇

1

> *Consider the following description of a memory hierarchy.*
>
> - *Virtual address width = 45 bits*
> - *Physical memory address width = 38 bits*
> - *Page size = 4 KB*
> - *Cache capacity = 8 KB*
> - *Block size = 32 Bytes*
> - *It is a write-back 2-way associative cache.*

> ***(a)** How many bits are there in the fields of tag, index, and block offset of the physical memory address?*

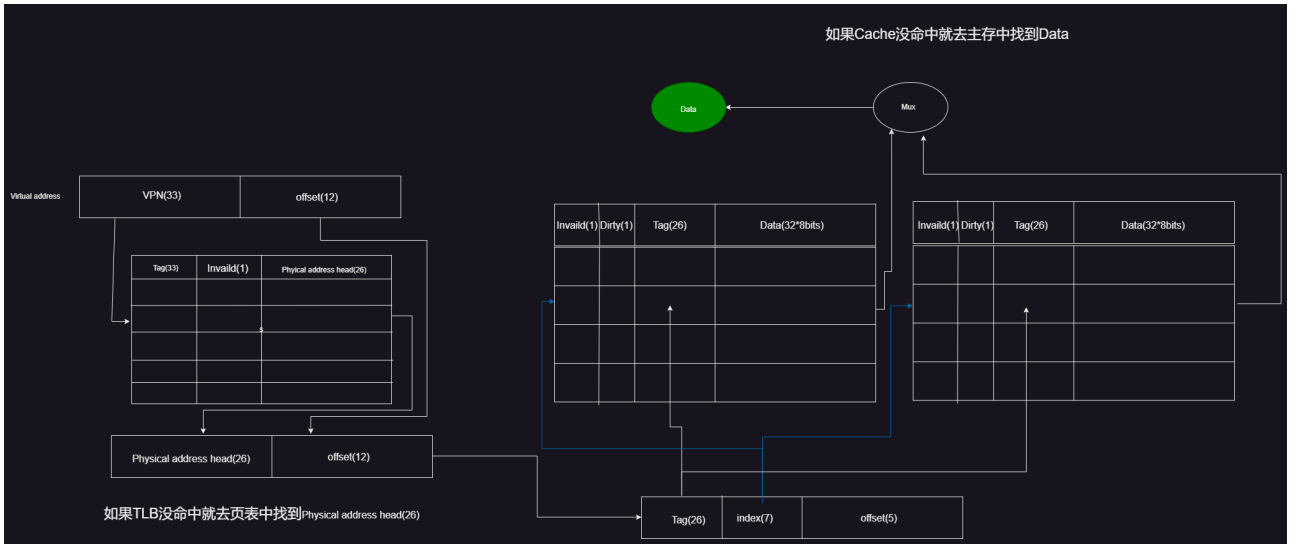
$$block\ offset = log_2(32Bytes) = 5bits$$
$$index = log_2\frac{8KB}{32Bytes} - 1 = 7bits$$
$$tag = (38 - 5 - 7)bits = 26bits$$

> ***(b)** Draw a graph to show a cache line (including tag, data, and some other control bits) in the cache.*

Invaild(1)	Dirty(1)	Tag(26)	Data(32*8bits)	Invaild(1)	Dirty(1)	Tag(26)	Data(32*8bits)

> ***(c)** Draw a graph to show if it is implemented in the way of virtually indexed and physically tagged cache. Draw both cache and TLB.*

$$vistual\ address = 45bits$$
$$page\ offset = log_2 4K = 12$$
$$VPN = 45 - 12 = 33$$



> ***(d)*** *Please describe the access procedure to the memory hierarchy in (c) when a CPU address (virtual address) is given to access the cache*

1. Virtual address of CPU
   A virtual address consists of the following two parts:

Virtual Page Number (VPN): Used to identify a page in virtual memory.

Offset: Used to identify specific bytes within a page.

2. TLB lookup

Use virtual page number (VPN) to search for the corresponding physical page number in TLB.

If the TLB hits (i.e. the corresponding physical page number is found in the TLB), the TLB returns the physical page number.

If the TLB misses (i.e. the corresponding physical page number cannot be found in the TLB), the page table needs to be accessed for address translation.

3. Page table lookup (if TLB misses)

Use virtual page number (VPN) to search for the corresponding physical page number in the page table.

After finding the physical page number (PPN), store it in the TLB for quick access next time.

4. Generate physical address
   After obtaining the physical page number, combine it with the page offset to generate a physical address.

Physical address=Physical page number+In page offset (Offset)

5. Cache lookup

Use the index portion of the virtual address to search for cache lines in the cache.

Verify the validity of cache lines using the marked portion of physical addresses.

If the cache hits (i.e. the corresponding data block is found in the cache), the data is returned.

If the cache misses (i.e. the corresponding data block cannot be found in the cache), data needs to be loaded from memory into the cache.

6. Memory access (if cache misses)

Use physical addresses to access main memory and retrieve data.

Load the data into the cache for quick access next time.

2

> *Assume that we have two machines A and B. The only difference between A and B lies in their cache hierarchies:*
>
> - ***Machine A:*** *64 KB level-one data cache with an 8 ns access time and a miss rate of 8%.*
> - ***Machine B:*** *8 KB level-one data cache with a 2 ns access time and a miss rate of 15%, and a 1 MB level-two cache with a 20 ns access time and a miss rate of 10%.*
>
> *Assume that both machines have an I-cache miss rate of 0%, a main memory access time of 50 ns, and all the bus transfer time could be ignored. Which machine will have better performance in memory access (AMAT)? Why?*

$$AMAT_A = 8ns + 8\% \times 50ns = 12ns$$
$$AMAT_B = 2ns + 15\%(20ns + 10\% \times 50ns) = 5.75ns$$

So Machine B have better performance in memory access since of less AMAT.

3

> *You are building a system around a processor with in-order execution that runs at 1.0 GHz and has a CPI of 1.35 excluding memory accesses. The only instructions that read or write data from memory are loads (20% of all instructions) and stores (10% of all instructions). The memory system for this computer is composed of a split L1 cache that imposes no penalty on hits. Both the I-cache and D-cache are direct-mapped and hold 32 KB each. The I-cache has a 2% miss rate and 32-byte blocks, and the D-cache is write-through with a 5% miss rate and 16-byte blocks. There is a write buffer on the D-cache that eliminates stalls for 90% of all writes. The 512 KB write-back, unified L2 cache has 64-byte blocks and an access time of 12 ns. It is connected to the L1 cache by a 128-bit data bus that runs at 266 MHz and can transfer one 128-bit word per bus cycle. Of all memory references sent to the L2 cache in this system, 85% are satisfied without going to main memory. Also, 50% of all blocks replaced are dirty. The 128-bit-wide main memory has an access latency of 80 ns, after which any number of bus words may be transferred at the rate of one per cycle on the 128-bit-wide 133 MHz main memory bus.*

> ***Questions:***
>
> ***a.*** *What is the average memory access time for instruction accesses?*

$$T_{L_2-miss} = 80ns + \frac{1}{133MHz} \times \frac{64Bytes}{128bits} = 110ns$$
$$T_{L_1-miss} = \frac{1}{266MHz} \times \frac{32Bytes}{128bits} + 12ns + 15\% \times T_{L_2-miss} = 36ns$$
$$AMAT_{I-Cache} = 0 + 2\% \times T_{L_1-miss} = 0.72ns$$

> ***b.*** *What is the average memory access time for data reads?*

$$T_{L_2-miss} = 80ns + (\frac{1}{133MHz} \times \frac{64Bytes}{128bits}) \times (50\% \times 2 + 50\%) = 125ns$$
$$T_{L_1-miss} = \frac{1}{266MHz} \times \frac{16Bytes}{128bits} + 12ns + 15\% \times T_{L_2-miss} = 34.5ns$$
$$AMAT_{D-Cache-read} = 0 + 5\% \times T_{L_1-miss} = 1.725ns$$

> ***c.*** *What is the average memory access time for data writes?*

$$T_{L_2-miss} = 80ns + (\frac{1}{133MHz} \times \frac{64Bytes}{128bits}) \times (50\% \times 2 + 50\%) = 125ns$$

$$T_{L_1-miss} == \frac{1}{266MHz} \times \frac{16Bytes}{128bits} + 12ns + 15\% \times T_{L_2-miss} = 34.5ns$$

$$AMAT_{D-Cache-write} = 0 + 5\% \times T_{L_1-miss} = 1.725ns$$

> ***d.*** *What is the overall CPI, including memory accesses?*
>
> *提示: L1 cache 的 miss penalty 认为是替换数据从 L2 到 L1 的传输时间，忽略响应时间。*

$$CPI = 1.35 + 0.72 + 0.2 \times 1.725 + 0.1 \times 1.725 = 2.5875$$

4

> *The transpose of a matrix interchanges its rows and columns; this is illustrated below:*
>
> $$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \Rightarrow \begin{bmatrix} A_{11} & A_{21} & A_{31} & A_{41} \\ A_{12} & A_{22} & A_{32} & A_{42} \\ A_{13} & A_{23} & A_{33} & A_{43} \\ A_{14} & A_{24} & A_{34} & A_{44} \end{bmatrix}$$
>
> *Here is a simple C loop to show the transpose:*
>
> ```c
> for (i = 0; i < 3; i++) {
>     for (j = 0; j < 3; j++) {
>         output[j][i] = input[i][j];
>     }
> }
> ```
>
> *Assume that both the input and output matrices are stored in row-major order (row-major order means that the row index changes fastest). Assume that you are executing a* 256×256 ***single-precision*** *transpose on a processor with a 16KB fully associative (don't worry about cache conflicts) least recently used (LRU) replacement L1 data cache with 64-byte blocks. Assume that the L1 cache misses or prefetches require 16 cycles and always hit in the L2 cache, and that the L2 cache can process a request every two processor cycles. Assume that each iteration of the inner loop above requires four cycles if the data are present in the L1 cache. Assume that the cache has a write-allocate fetch-on write policy for write misses. Unrealistically, assume that writing back dirty cache blocks requires 0 cycles.*

> *For the simple implementation given above, this execution order would be nonideal for the input matrix; however, applying a loop interchange optimization would create a nonideal order for the output matrix. Because loop interchange is not sufficient to improve its performance, it must be blocked instead.*

> ***Questions:***
>
> ***a.*** *What should be the minimum size of the cache to take advantage of blocked execution?*

$$single-precision\ size = 4Bytes$$

$$Element\ in\ a\ block = \frac{64Bytes}{4Bytes} = 16$$

$$minimun\ size\ of\ cache = 2 \times (16 \times 64Bytes) = 2048Bytes$$

> ***b.*** *How do the relative number of misses in the blocked and unblocked versions compare in the minimum sized cache above?*

In the blocked version,we can calculate the miss rate
$\frac{16+16}{16\times16} = \frac{2}{16} miss\ per\ element$

In the unblocked version,we can calculate the miss rate
$\frac{16+256}{256} = \frac{17}{16} miss\ per\ element$

The miss rate in unblocked version is $\frac{17}{2}$ larger than that in blocked version.

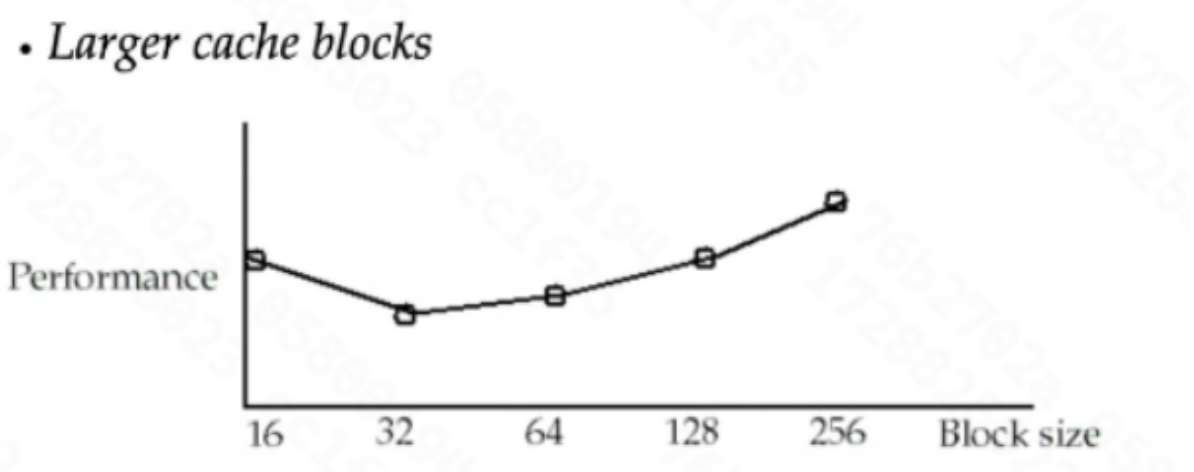
> ***c.*** *Write code to perform a transpose with a block size parameter B which uses $B \times B$ blocks.*

```
for (i = 0; i < 256; i=i+B) {
    for (j = 0; j < 256; j=j+B) {
        for(m=0; m<B; m++) {
            for(n=0; n<B; n++) {
                output[j+n][i+m] = input[i+m][j+n];
            }
        }
    }
}
```

> ***d.*** *What is the minimum associativity required of the L1 cache for consistent performance independent of both arrays' position in memory?*

2-way associativity?since when n = 1 , in the direct mapped cache blocks will be repeated

5



> *解释上图为什么会呈U型*

- **why decrease** : Increases capacity and conflict misses, increases miss penalty.Larger cache blocks may increase cache miss rates, as the amount of data loaded each time is large and may contain some data that is not currently needed, resulting in data in cache lines that are not currently needed.
- **why increase**: Reduces compulsory misses.Smaller cache blocks can reduce cache miss rates because the amount of data loaded each time is smaller, making it easier to hit the cache.

6

> *Below are various cache optimization techniques. Please match each technique with the corresponding optimization goal by choosing from the following*
>
> *options:*
>
> 1. ***Reduce the miss penalty***
> 2. ***Reduce the miss rate***
> 3. ***Reduce the miss penalty and miss rate via parallelism***
> 4. ***Reduce the time to hit in the cache***

Techniques:

1. Multilevel caches: (1)
2. Critical word first: (1)
3. Larger block size: (2)

4. Non-blocking caches: (1)
5. Hardware prefetching: (3)
6. Small and simple caches: (4)
7. Higher associativity: (2)
8. Avoiding address translation: (4)
9. Victim caches: (2)
10. Compiler optimizations: (2)
11. Way prediction and pseudo-associativity: (4)
12. Pipelined cache access: (3)