# Cache memory performance overview and cache memory optimization

Presented By

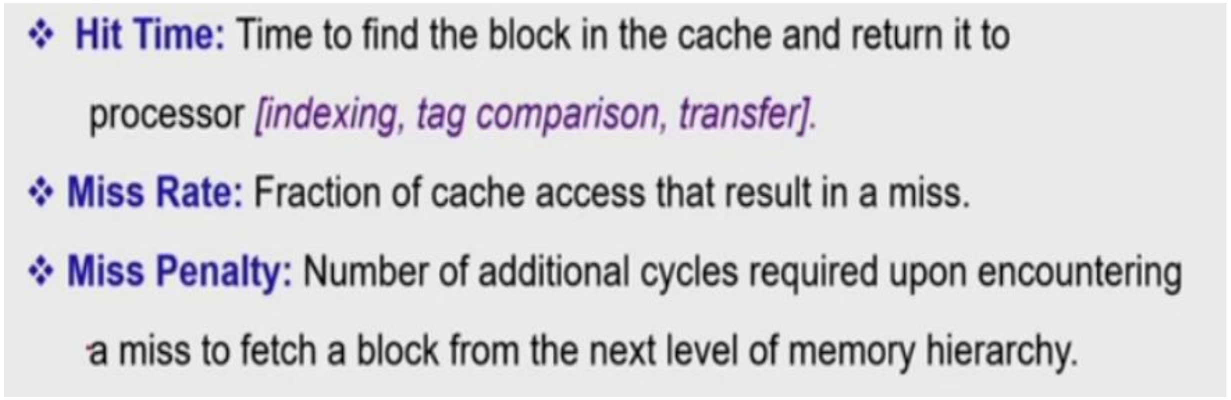
**Dr. Banchhanidhi Dash**

School of Computer Engineering
KIIT University

## How to Improve Cache Performance?

A better measure of memory hierarchy performance is the ***Average Memory Access Time* (AMAT) per instructions**

**AMAT=hit time+miss rate* miss penalty**



- **Hit Time:** Time to find the block in the cache and return it to processor *[indexing, tag comparison, transfer].*
- **Miss Rate:** Fraction of cache access that result in a miss.
- **Miss Penalty:** Number of additional cycles required upon encountering a miss to fetch a block from the next level of memory hierarchy.

**where hit time is the time to hit in the cache; we have seen the other two terms before. The components of average access time can be measured either in absolute time—say, 0.25–1.0 ns on a hit—or in the number of clock cycles that the processor waits for the memory**

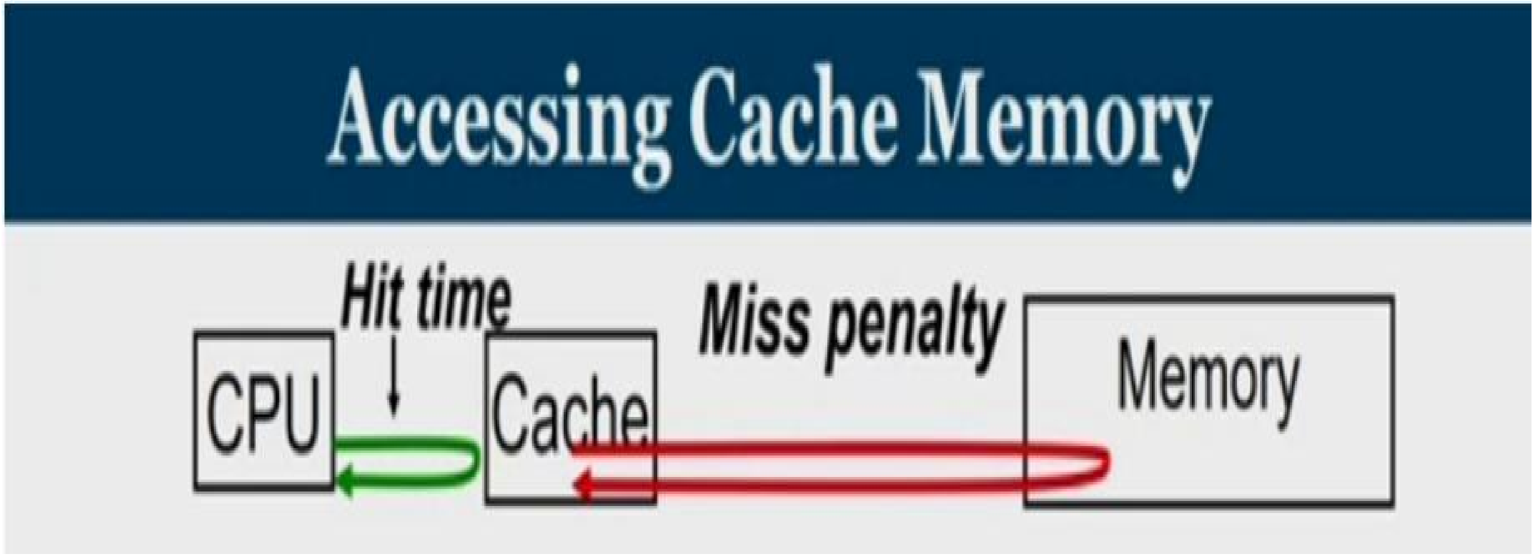
- If a direct mapped cache has a hit rate of 95%, a hit time of 4 ns, and a miss penalty of 100 ns, what is the AMAT?

  AMAT = Hit time + Miss rate x Miss penalty = 4 + 0.05 x 100 = 9 ns

- If replacing the cache with a 2-way set associative increases the hit rate to 97%, but increases the hit time to 5 ns, what is the new AMAT?

  AMAT = Hit time + Miss rate x Miss penalty = 5 + 0.03 x 100 = 8 ns

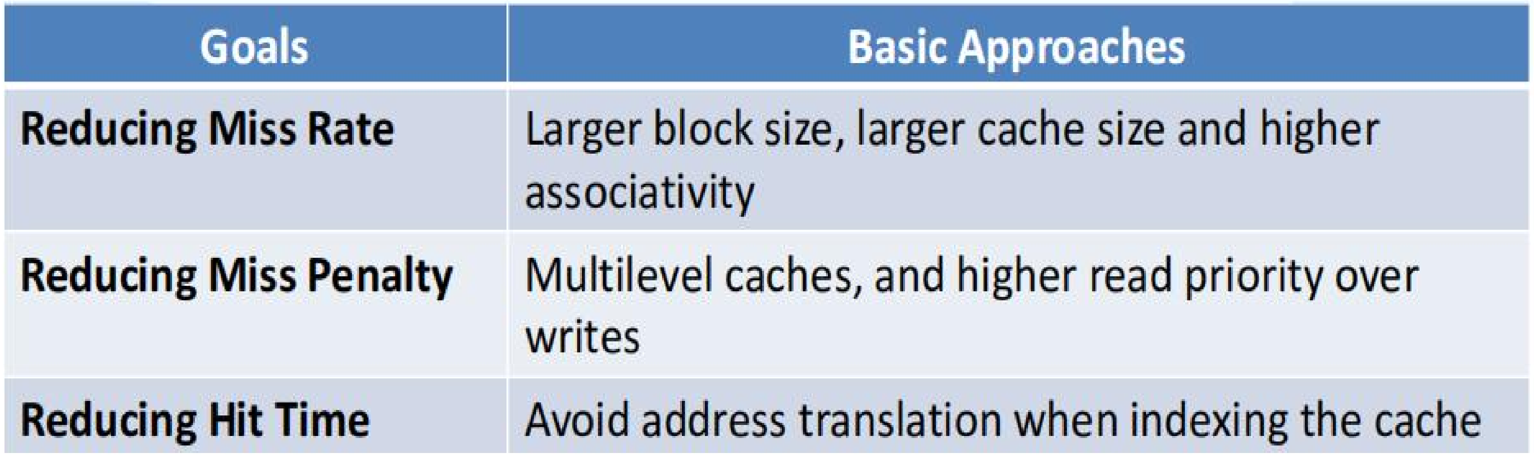
## Cache memory Optimizations



## Accessing Cache Memory

## How to Improve Cache Performance?

$$\text{Average Memory Access Time} = \boxed{\text{Hit Time}} + \boxed{\text{Miss Rate}} * \boxed{\text{Miss Penalty}}$$



| Goals | Basic Approaches |
|---|---|
| **Reducing Miss Rate** | Larger block size, larger cache size and higher associativity |
| **Reducing Miss Penalty** | Multilevel caches, and higher read priority over writes |
| **Reducing Hit Time** | Avoid address translation when indexing the cache |

## Reducing Misses

- The classical approach to improve the cache behavior is to reduce miss rates Misses: 3 Cs
  - *Compulsory*—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*.
    *(Misses in infinite cache)*
  - *Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
    *(Misses due to size of cache)*
  - *Conflict*—If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*.
    *(Misses due to associative and size of cache)*

To show the benefit of associativity, conflict misses are divided into misses caused by each decrease in associativity. Here are the four divisions of conflict misses

**Eight-way**—Conflict misses due to going from fully associative (no conflicts)
to eight-way associative

■ **Four-way**—Conflict misses due to going from eight-way associative to four-way associative

■ **Two-way**—Conflict misses due to going from four-way associative to two-way associative

■ **One-way**—Conflict misses due to going from two-way associative to one-way
associative (direct mapped)

## 6 basic cache optimizations technique

- **Reducing miss rate**
  - **Larger block size**
  - **Larger caches** size
  - **Higher associativity**
- **Reducing miss penalty**
  - **Multilevel caches**
  - **Giving priority to read misses over writes**
- **Reducing cache hit time**
  - *Avoiding address translation during indexing of the cache to reduce hit time*

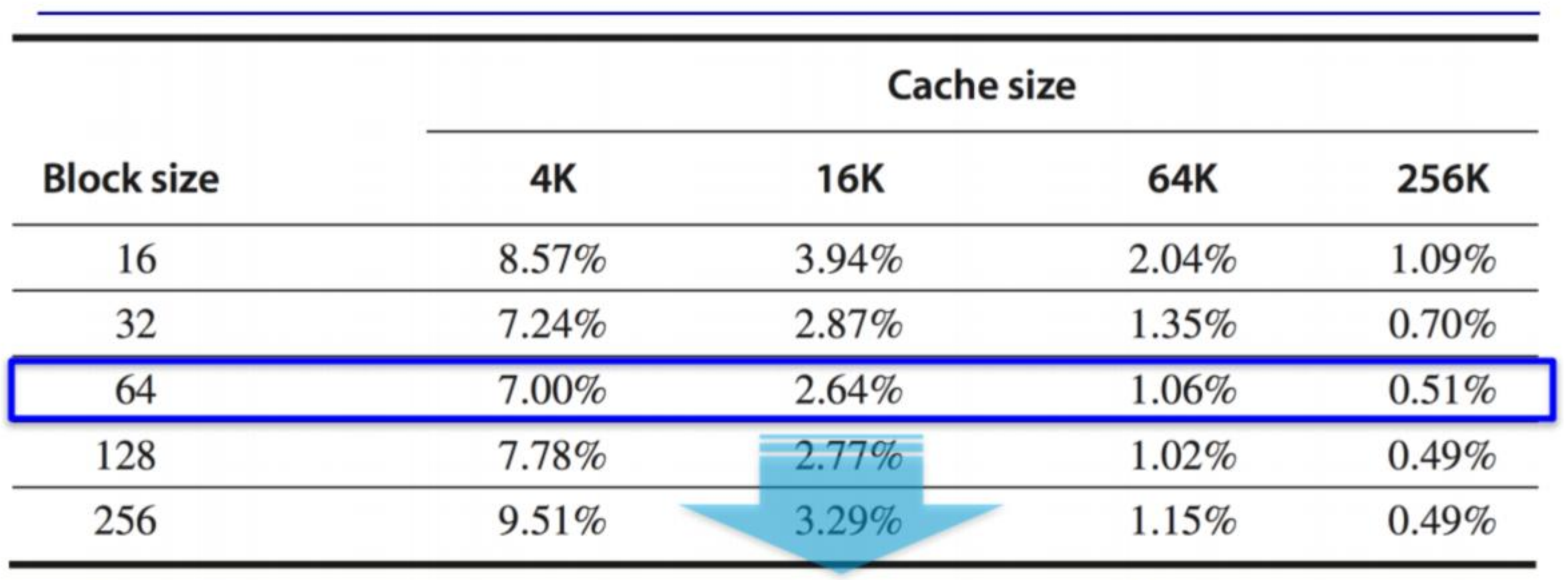
## First Optimization: Larger Block Size to Reduce Miss Rate

- The simplest way to reduce miss rate is to increase the block size. Figure shows the trade-off of block size versus miss rate for a set of programs and cache sizes. Larger block sizes will reduce also compulsory misses. This reduction occurs because the principle of locality has two components: temporal locality and spatial locality. Larger blocks take advantage of spatial locality.

- At the same time, larger blocks increase the miss penalty. Because they reduce the number of blocks in the cache, larger blocks may increase conflict misses and even capacity misses if the cache is small. Clearly, there is little reason to increase the block size to such a size that it increases the miss rate. There is also no benefit to reducing miss rate if it increases the average memory access time. The increase in miss penalty may outweigh the decrease in miss rate.

## First Optimization: Larger Block Size to Reduce Miss Rate

### Much Larger Block Size: → Increase Miss Rate



|  | Cache size |  |  |  |
|---|---|---|---|---|
| Block size | 4K | 16K | 64K | 256K |
| 16 | 8.57% | 3.94% | 2.04% | 1.09% |
| 32 | 7.24% | 2.87% | 1.35% | 0.70% |
| 64 | 7.00% | 2.64% | 1.06% | 0.51% |
| 128 | 7.78% | 2.77% | 1.02% | 0.49% |
| 256 | 9.51% | 3.29% | 1.15% | 0.49% |

**Figure** **Actual miss rate versus block size for the five different-sized caches**

Actual miss rate versus block size for the five different-sized caches Note that for a 4 KB cache, 256-byte blocks have a higher miss rate than 32-byte blocks. In this example, the cache would have to be 256 KB in order for a 256-byteblock to decrease misses.

## First Optimization: Larger Block Size to Reduce Miss Rate

### Reduce Miss Rate via Larger Block Size

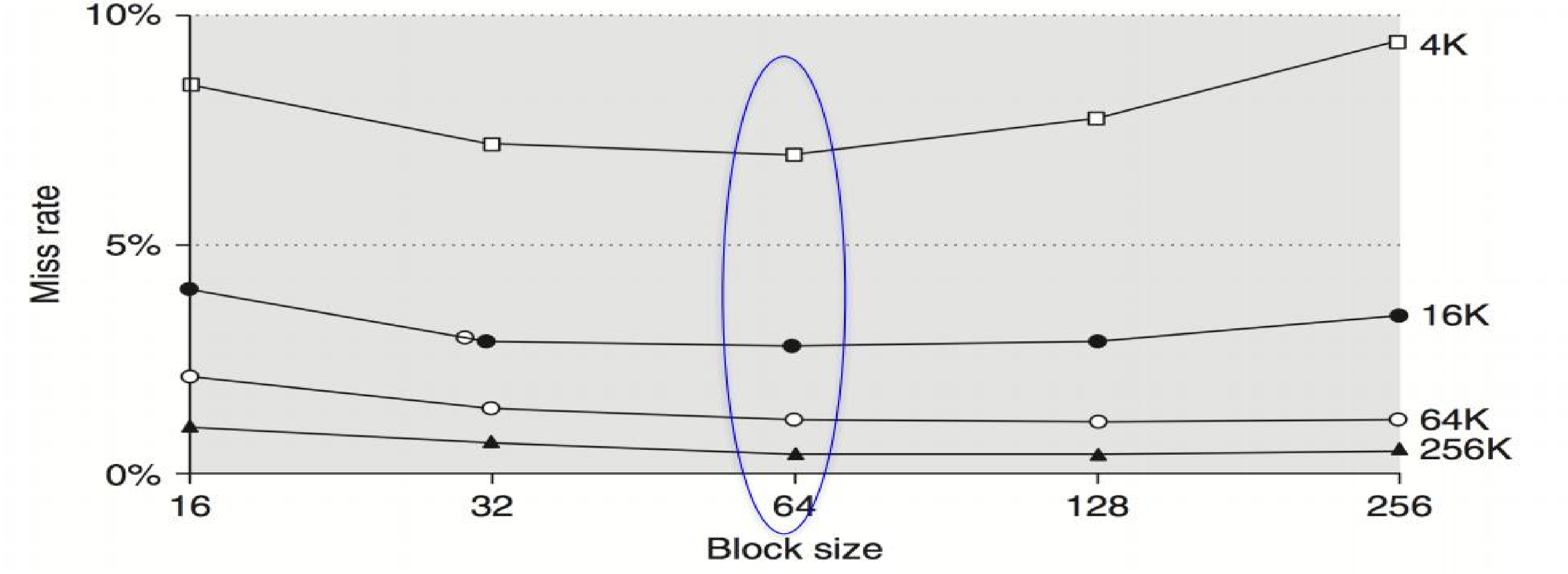


**Figure** Miss rate versus block size for five different-sized caches.

Miss rate versus block size for five different-sized caches. Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size.

Average memory access time is

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

If we assume the hit time is 1 clock cycle independent of block size, then the access time for a 16-byte block in a 4 KiB cache is

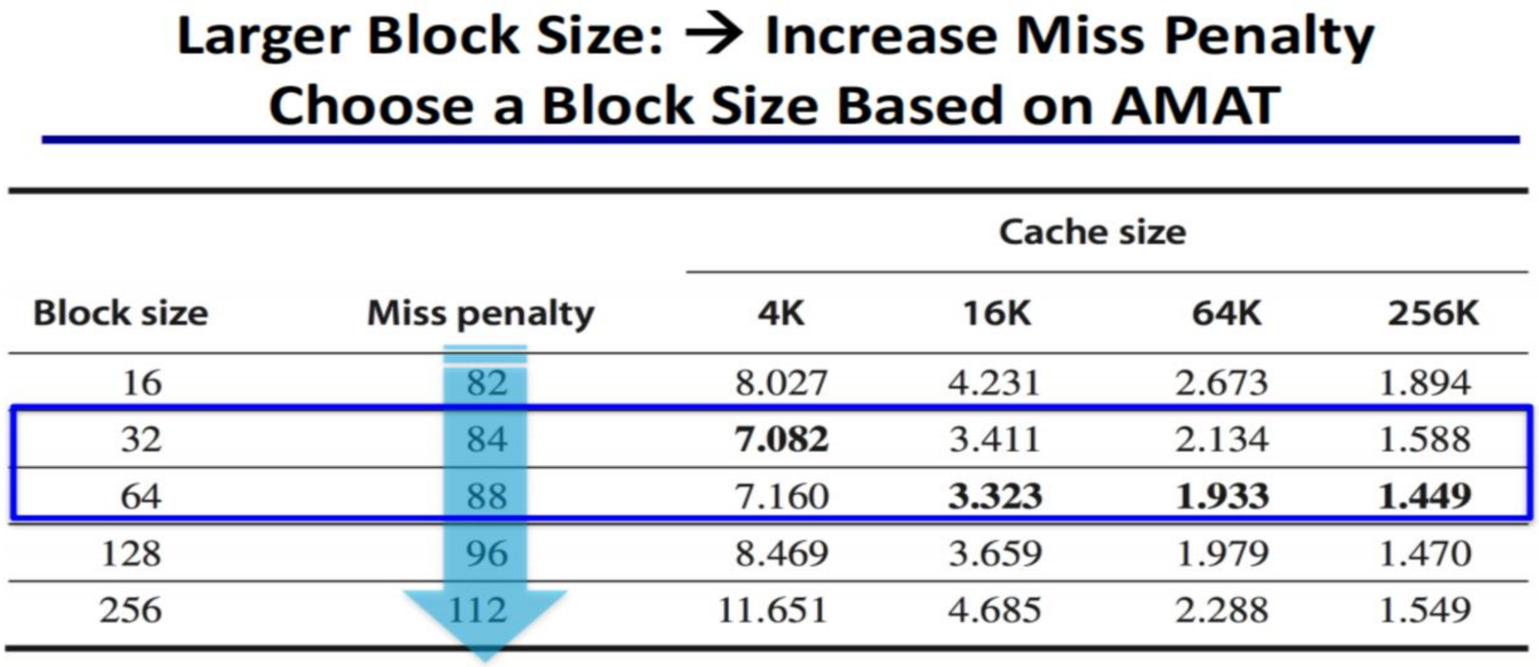
$$\text{Average memory access time} = 1 + (8.57\% \times 82) = 8.027 \text{ clock cycles}$$

and for a 256-byte block in a 256 KiB cache the average memory access time is

$$\text{Average memory access time} = 1 + (0.49\% \times 112) = 1.549 \text{ clock cycles}$$

**miss rate vs. reduce AMAT**

which  block size has the smallestAverage memory access time for each cache size??



## Larger Block Size: → Increase Miss Penalty
## Choose a Block Size Based on AMAT

| | | Cache size | | | |
|---|---|---|---|---|---|
| Block size | Miss penalty | 4K | 16K | 64K | 256K |
| 16 | 82 | 8.027 | 4.231 | 2.673 | 1.894 |
| 32 | 84 | **7.082** | 3.411 | 2.134 | 1.588 |
| 64 | 88 | 7.160 | **3.323** | **1.933** | **1.449** |
| 128 | 96 | 8.469 | 3.659 | 1.979 | 1.470 |
| 256 | 112 | 11.651 | 4.685 | 2.288 | 1.549 |

Average memory access time versus block size for five different-sized caches . Block sizes of 32 and 64 bytes dominate. The smallest average time per cache size is boldfaced.

the average memory access time for all block and cache sizes between those two extremes. The boldfaced entries show the fastest block size for a given cache size: 32 bytes for 4 KB and 64 bytes for the larger caches. These sizes are, in fact, popular block sizes for processor caches today

## First Optimization: Larger Block Size to Reduce Miss Rate

- The cache designer is trying to minimize both the miss rate and the miss penalty. The selection of block size depends on both the latency and bandwidth of the lower-level memory.
- High latency and high bandwidth encourage large block size because the cache gets many more bytes per miss for a small increase in miss penalty.
- Conversely, low latency and low bandwidth encourage smaller block sizes because there is little time saved from a larger block.
- For example, twice the miss penalty of a small block may be close to the penalty of a block twice the size. The larger number of small blocks may also reduce conflict misses.

- Larger block size
  - Take advantage of spatial locality
  - Decreases compulsory misses
- 
  - May increase the miss penalty (need to get more data)
  - May increase hit time (need to read more data from cache )

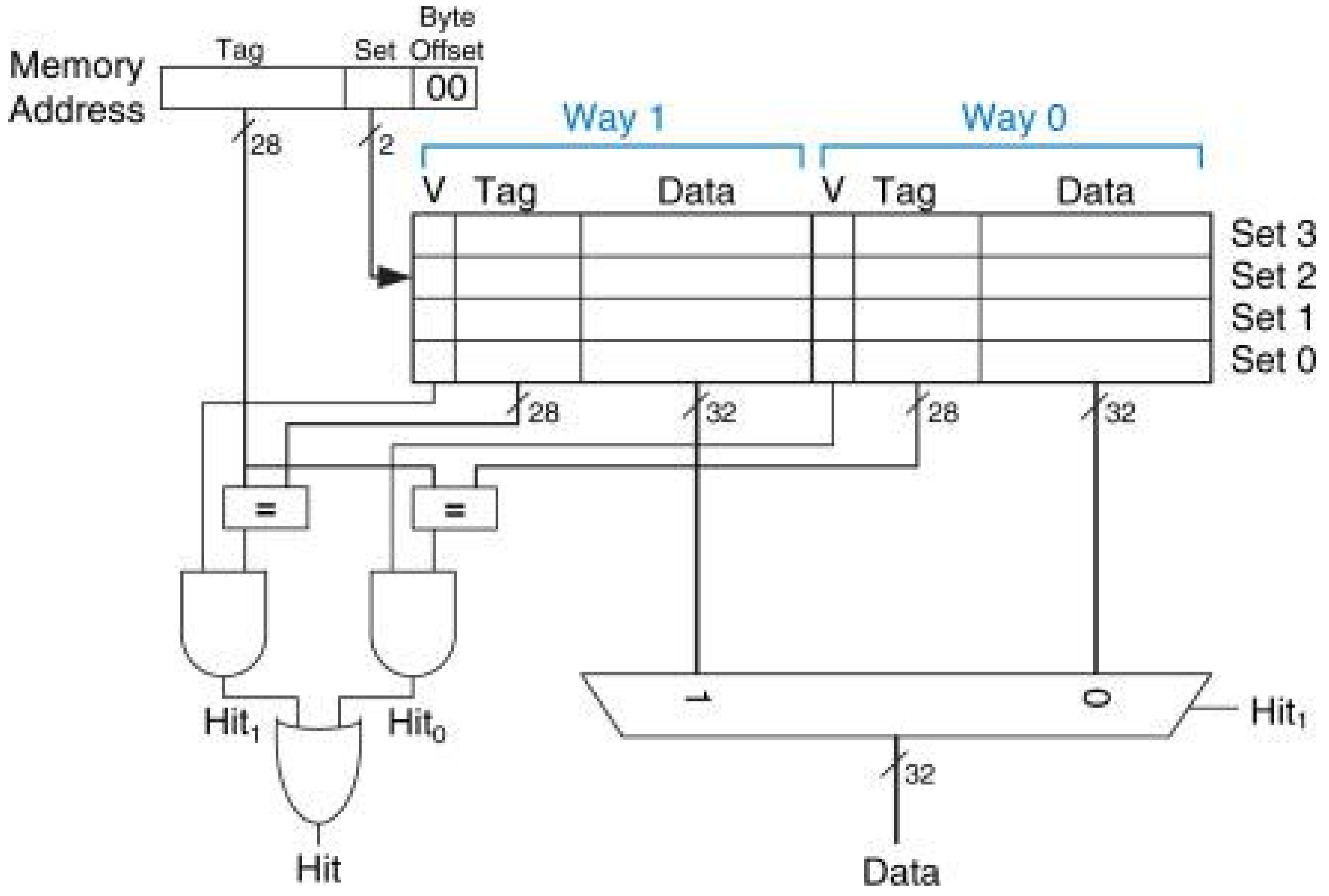
## second  Optimization: Larger cache Size to Reduce Miss Rate

Compulsory misses are independent of cache size, while capacity misses decrease as capacity increases, and conflict misses  decrease as associativity increases. The  way to reduce capacity misses  is to increase capacity of the cache.
The obvious drawback is potentially longer hit time and highercost and power. This technique has been especially popular in off-chip caches.

## third Optimization: Increasing Associativity to Reduce Miss Rate

- Increasing associativity helps reduce conflict misses
- 2:1 Cache Rule:
  - The miss rate of a direct mapped cache of size N is about equal to the miss rate of a 2-way set associative cache of size N/2
  - For example, the miss rate of a 32 Kbyte direct mapped cache is about equal to the miss rate of a 16 Kbyte 2-way set associative cache
- 
  - Need to do large number of comparisons
  - Need n-to-1 multiplexor for n-way set associative could increase hit time

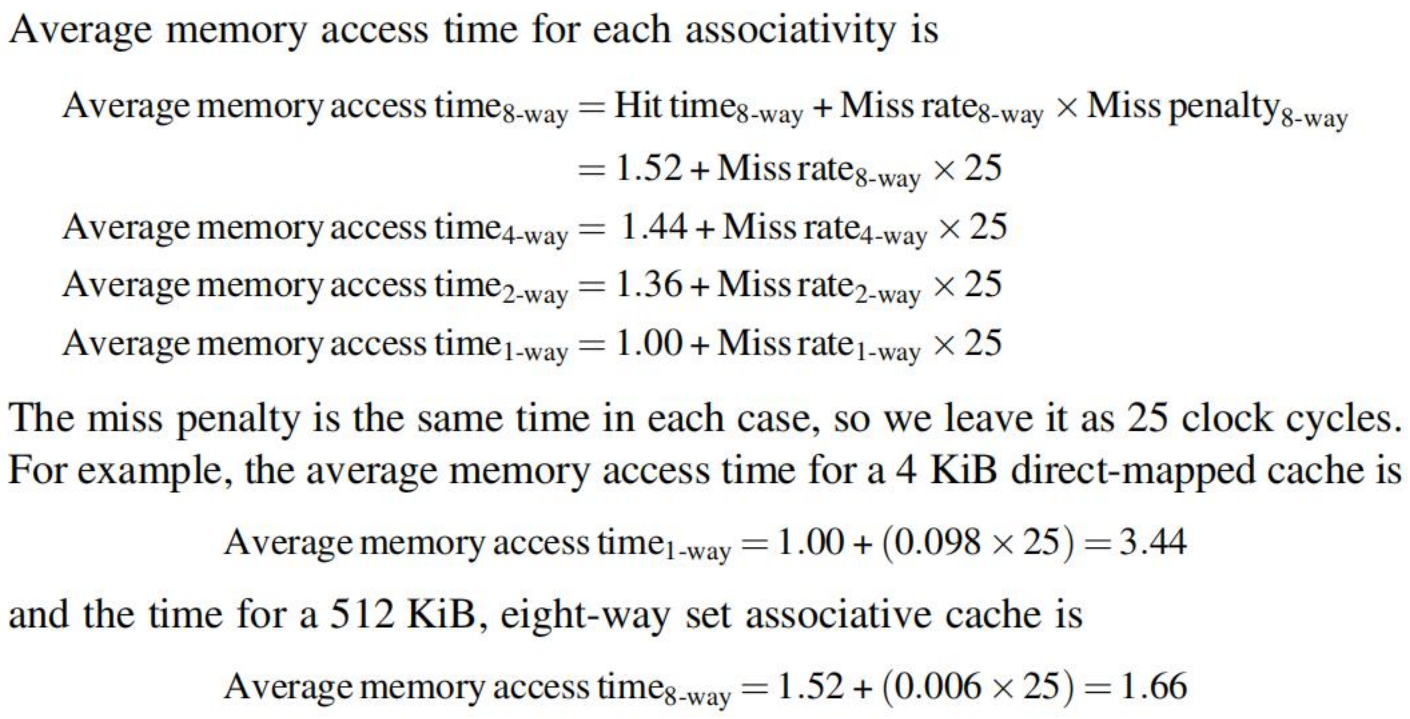
# 2 way set associative cache example



## Total miss rate for each size cache

| Cache size (KiB) | Degree associative | Total miss rate |
|---|---|---|
| 4 | 1-way | 0.098 |
| 4 | 2-way | 0.076 |
| 4 | 4-way | 0.071 |
| 4 | 8-way | 0.071 |
| 8 | 1-way | 0.068 |
| 8 | 2-way | 0.049 |
| 8 | 4-way | 0.044 |
| 8 | 8-way | 0.044 |
| 16 | 1-way | 0.049 |
| 16 | 2-way | 0.041 |
| 16 | 4-way | 0.041 |
| 16 | 8-way | 0.041 |
| 32 | 1-way | 0.042 |
| 32 | 2-way | 0.038 |
| 32 | 4-way | 0.037 |
| 32 | 8-way | 0.037 |
| 64 | 1-way | 0.037 |
| 64 | 2-way | 0.031 |
| 64 | 4-way | 0.030 |
| 64 | 8-way | 0.029 |
| 128 | 1-way | 0.021 |
| 128 | 2-way | 0.019 |
| 128 | 4-way | 0.019 |
| 128 | 8-way | 0.019 |
| 256 | 1-way | 0.013 |
| 256 | 2-way | 0.012 |
| 256 | 4-way | 0.012 |
| 256 | 8-way | 0.012 |
| 512 | 1-way | 0.008 |
| 512 | 2-way | 0.007 |
| 512 | 4-way | 0.006 |
| 512 | 8-way | 0.006 |

## AMAT calculation for each associatitivty



Average memory access time for each associativity is

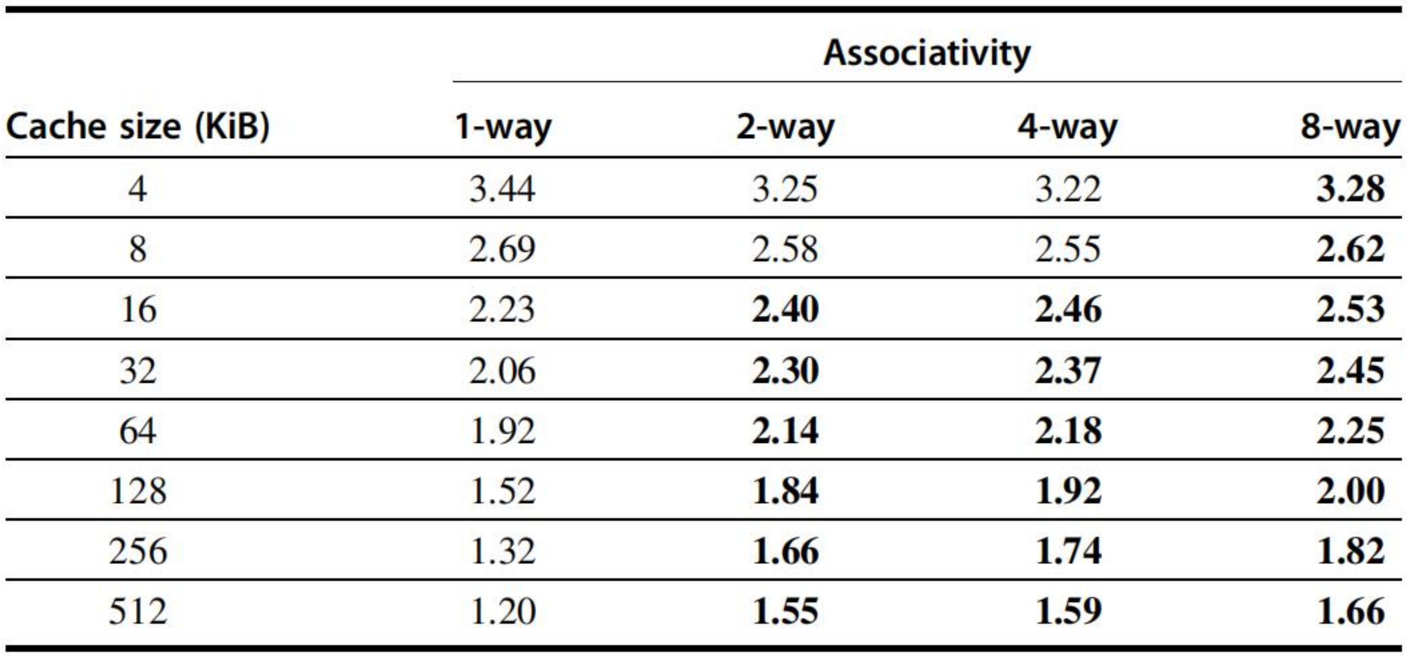
$$\begin{aligned}
\text{Average memory access time}_{\text{8-way}} &= \text{Hit time}_{\text{8-way}} + \text{Miss rate}_{\text{8-way}} \times \text{Miss penalty}_{\text{8-way}} \\
&= 1.52 + \text{Miss rate}_{\text{8-way}} \times 25 \\
\text{Average memory access time}_{\text{4-way}} &= 1.44 + \text{Miss rate}_{\text{4-way}} \times 25 \\
\text{Average memory access time}_{\text{2-way}} &= 1.36 + \text{Miss rate}_{\text{2-way}} \times 25 \\
\text{Average memory access time}_{\text{1-way}} &= 1.00 + \text{Miss rate}_{\text{1-way}} \times 25
\end{aligned}$$

The miss penalty is the same time in each case, so we leave it as 25 clock cycles. For example, the average memory access time for a 4 KiB direct-mapped cache is

$$\text{Average memory access time}_{\text{1-way}} = 1.00 + (0.098 \times 25) = 3.44$$

and the time for a 512 KiB, eight-way set associative cache is

$$\text{Average memory access time}_{\text{8-way}} = 1.52 + (0.006 \times 25) = 1.66$$



|  | Associativity |  |  |  |
|---|---|---|---|---|
| Cache size (KiB) | 1-way | 2-way | 4-way | 8-way |
| 4 | 3.44 | 3.25 | 3.22 | **3.28** |
| 8 | 2.69 | 2.58 | 2.55 | **2.62** |
| 16 | 2.23 | **2.40** | **2.46** | **2.53** |
| 32 | 2.06 | **2.30** | **2.37** | **2.45** |
| 64 | 1.92 | **2.14** | **2.18** | **2.25** |
| 128 | 1.52 | **1.84** | **1.92** | **2.00** |
| 256 | 1.32 | **1.66** | **1.74** | **1.82** |
| 512 | 1.20 | **1.55** | **1.59** | **1.66** |

Average memory access time using miss rates in slide no 25. for parameters in the example. Boldface type means that this time is higher than the number to the left, that is, higher associativity increases average memory access time.

## Associativity

+ Decreases conflict misses
– Increases $\text{latency}_{hit}$

## Block size

– Increases conflict/capacity misses (fewer entries)
+ Decreases compulsory/capacity misses (spatial locality)
- No significant effect on $\text{latency}_{hit}$

## Capacity

+ Decreases capacity misses
– Increases $\text{latency}_{hit}$

## fourth optimization:Reducing Miss penalty with multi level cache

The performance gap between processors and memory leads the architect to a question:
**Should I make the cache faster to keep pace with the speed of processors, or make the cache larger to overcome the widening gap between the processor and main memory?**

**ANS:**
One answer is, do both.
Adding another level of cache between the original cache and memory simplifies the decision. The first-level cache can be small enough to match the clock cycle time of the fast processor. Yet, the second-level cache can be large enough to capture many accesses that would go to main memory, thereby lessening the effective miss penalty.

## Reduce Miss Penalty via Multilevel Caches

- Approaches
  - Make the cache faster to keep pace with the speed of CPUs
  - Make the cache larger to overcome the widening gap
- **L1: fast hits, L2: fewer misses**
- L2 Equations

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{L1}$$

and

$$\text{Miss penalty}_{L1} = \text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}$$

so

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

- Hit $\text{Time}_{L1}$ << Hit $\text{Time}_{L2}$ << … << Hit $\text{Time}_{Mem}$
- Miss $\text{Rate}_{L1}$ < Miss $\text{Rate}_{L2}$ < …

## Multi-Level Cache: Some Definitions

- Local miss rate— misses in this cache divided by the total number of memory accesses to this cache
- Global miss rate—misses in this cache divided by the total number of memory accesses generated by the CPU
  - Local Miss Rate$_{L1}$ x Local Miss Rate$_{L2}$
- L1 Global miss rate = L1 Local miss rate

**Consider a system with two level cache having 400 memory references and there are 40 misses in the L1 cache and 20 misses in the L2 cache. Calculate the global and local miss rates for both the caches?**

**solution:**

**The miss rate (either local or global) for the (first-level cache) L1 is 40/400 or 10%.**
**The global miss rate of (second-level) L2 cache is 20/400 or 5%**
**The local miss rate of (second-level) L2 cache is 20/40=0.5**

## Example

Suppose that in 1000 memory references there are 40 misses in the first-level cache and 20 misses in the second-level cache. What are the various miss rates? Assume the miss penalty from the L2 cache to memory is 200 clock cycles, the hit time of the L2 cache is 10 clock cycles, the hit time of L1 is 1 clock cycle. What is the average memory access time .ignore the impact of writes.

## Example

Ans:
The miss rate (either local or global) for the first-level cache is 40/1000 or 4%. The
local miss rate for the second-level cache is 20/40 or 50%. The global miss rate of
the second-level cache is 20/1000 or 2%. Then
Average memory access time= Hit $\text{time}_{L1}$ + Miss $\text{rate}_{L1}$ (Hit $\text{time}_{L2}$ + Miss $\text{rate}_{L2}$ xMiss penaltyL2 )

=1+4%(10 + 50%*200) = 1+4% *110 = 5.4 clock cycles

## Multilevel Caches: Design of L2

- Size
  - Since everything in L1 cache is likely to be in L2 cache, L2 cache should be much bigger than L1
- Whether data in L1 is in L2
  - novice approach: design L1 and L2 independently
  - multilevel inclusion: L1 data are always present in L2
    - Advantage: easy for consistency between I/O and cache (checking L2 only)
    - Drawback: L2 must invalidate all L1 blocks that map onto the 2nd-level block to be replaced => slightly higher 1st-level miss rate
      - i.e. Intel Pentium 4: 64-byte block in L1 and 128-byte in L2
  - multilevel exclusion: L1 data is never found in L2
    - A cache miss in L1 results in a swap of blocks between L1 and L2
    - Advantage: prevent wasting space in L2
      - i.e. AMD Athlon: 64 KB L1 and 256 KB L2

## Split vs. Unified Cache

- **Unified cache (mixed cache): Data and instructions are stored together (von Neuman architecture)**
- **Split cache: Data and instructions are stored separately (Harvard architecture)**
- **Why do instructions caches have a lower miss ratio?**

| Size | Instruction Cache | Data Cache | Unified Cache |
|---|---|---|---|
| 1 KB | 3.06% | 24.61% | 13.34% |
| 2 KB | 2.26% | 20.57% | 9.78% |
| 4 KB | 1.78% | 15.94% | 7.24% |
| 8 KB | 1.10% | 10.19% | 4.57% |
| 16 KB | 0.64% | 6.47% | 2.87% |
| 32 KB | 0.39% | 4.82% | 1.99% |
| 64 KB | 0.15% | 3.77% | 1.35% |
| 128 KB | 0.02% | 2.88% | 0.95% |

## Example: Split vs. Unified Cache

- Which has the lower average memory access time?
  - Split cache : 16 KB instructions + 16 KB data
  - Unified cache: 32 KB (instructions + data)
- Assumptions
  - Use miss rates from previous chart
  - Miss penalty is 50 cycles
  - Hit time is 1 cycle
  - 75% of the total memory accesses for instructions and 25% of the total memory accesses for data
  - On the unified cache, a load or store hit takes an extra cycle, since there is only one port for instructions and data

## Example: Split vs. Unified Cache

Average memory-access time = Hit time + Miss rate x Miss penalty

AMAT = %instr x (instr hit time + instr miss rate x instr miss penalty) +
%data x (data hit time + data miss rate x data miss penalty)

For the split cache:

AMAT = 75% x (1 + 0.64% x 50) + 25% (1 + 6.47% x 50) = 2.05

For the unified cache

AMAT = 75% x (1 + 1.99% x 50) + 25% x (2 + 1.99% x 50) = 2.24

The unified cache has a longer AMAT, even though its miss rate is lower, due to conflicts for instruction and data hazards.

# Example

- Which has a lower miss rate?
  - A split cache (16KB instruction cache +16KB Data cache) or a 32 KB unified cache?
- Compute the respective AMAT also.
- 40% Load/Store instructions
- Hit time = 1 cycle
- Miss penalty = 100 cycles
- Simulator showed:
  - 40 misses per thousand instructions for data cache
  - 4 misses per thousand instr for instruction cache
  - 44 misses per thousand instr for unified cache

# Answer

- **misses/instruction=miss rate*(memory Access/insuction)**
- Miss rate = (misses/instructions)/(mem accesses/instruction)
- miss rate=[(misses/1000 instruction)/1000]/(memory access/Instruction)
- Instruction cache miss rate= 4/1000=0.004
- Data cache miss rate = (40/1000)/0.4 =0.1
- Unified cache miss rate = (44/1000)/1.4 =0.03142
- Overall miss rate for split cache = 0.3*0.1+0.7*0.004=0.0303

## Answer

**cont…**

- AMAT (split cache)= 0.7*(1+0.004*100)+0.3(1+0.1*100)=4.28
- AMAT (Unified)= 0.7(1+0.03142*100)+0.3(1+1+0.03142*100)=5.3846

## cache performance

cycle counts=total number of cycles required to execute a program

CPI=CC/IC

CC=IC *CPI

cpuexecution time=clock cycles*clock cycle time

CPUExecutiontime = ICx CPI x Clock cycle time

for cache memory

total no of clock cycle =cpu clock cycle +memory stall cycle

CPU execution time = (CPU clock cycles + Memory stall cycles )xClock cycle time

(Memory stall cycles: Number of cycles during which processor is stalled waiting for a memory access.)

Memory stall cycles= Number of misses*miss penalty

Memory stall cycles=IC*(misses/instruction)*miss penalty

misses/instruction=miss rate*(memory Access/insuction)

Memory stall cycles=IC*(memory Access/insuction)*miss rate*miss penalty

# cache performance

CPI = $(CPI_{execution}$ + Mem Stall CPI )

where $CPI_{execution}$ = CPI with ideal memory

CPU Executiontime = IC x $(CPI_{execution}$ + Mem Stall CPI) x Clock cycle time

mem stall CPI=memory stall cycles /IC

but Memory stall cycles=IC*(misses/instruction)*miss penalty

hence

mem stall CPI=( IC*(misses/instruction)*miss penalty)/IC

mem stall CPI=(misses/instruction)*miss penalty

CPU Executiontime = IC x $(CPI_{execution}$ + (misses/instruction)*miss penalty) x Clock cycle time

mem stall CPI=(memory Access/insuction)*miss rate*miss penalty)

CPU Executiontime = IC x $(CPI_{execution}$ + (memory Access/insuction)*miss rate*miss penalty) x Clock cycle time

## Examples

Assuming the following execution and cache parameters:

Cache miss penalty = 50 cycles

Normal instruction execution CPI ignoring memory stalls = 2.0 cycles

Miss rate = 2%

Average memory references/instruction = 1.33

clock rate =3.2 GHz

Number of Instruction =100

caclulate Cpu execution time?

## Assuming the following execution and cache parameters:

**Cache miss penalty = 50 cycles**

**Normal instruction execution CPI ignoring memory stalls = 2.0 cycles**

**Miss rate = 2%**

**Average memory references/instruction = 1.33**

**clock rate =3.2 GHz**

**Number of Instruction =100**

**caclulate Cpu time**

**CPU time =**

**IC x [CPI $_{execution}$ + Memory accesses/instruction x Miss rate xMiss penalty ] x Clock cycle time**

CPUtime $_{\text{with cache}}$ = IC x (2.0 + (1.33 x 2% x 50)) x clock cycle time

= IC x 3.33 x Clock cycle time

= 100*3.33*0.3125

=104.06ns

→

## Example

**Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.**

**$CPI_{execution}$ = 1.1**

**Instruction mix: 50% arith/logic, 30% load/store, 20% control**

**Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles**.Calculate CPI?

**Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.**
**$CPI_{execution}$ = 1.1**
**Instruction mix: 50% arith/logic, 30% load/store, 20% control**
**Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles**.Calculate CPI?

Solution:

CPI = $CPI_{execution}$ + mem stalls per instruction
Mem Stalls per instruction = Mem accesses per instruction x Miss rate x Miss penalty
Mem accesses per instruction = 1 + .3 = 1.3

Mem Stalls per instruction = 1.3 x .015 x 50 = 0.975
CPI = 1.1 + .975 = 2.075

The ideal memory CPU with no misses is 2.075/1.1 = 1.88 times faster

Memory Stall CPI

= Miss per inst $\times$ miss penalty

= % Memory Access/Instr $\times$ Miss rate $\times$ Miss Penalty

Example: Assume 20% memory acc/instruction, 2% miss rate, 400-cycle miss penalty. How much is memory stall CPI?

Memory Stall CPI= 0.2*0.02*400=1.6 cycles

## 5th optimization:

## Reducing Miss penalty with Giving priority to read misses over writes

## Reducing Miss Penalty : Read Priority over Write on Miss

- In a write-back scheme:
  - Normally a dirty block is stored in a write buffer temporarily.
  - Usual:
    - Write all blocks from the write buffer to memory, and then do the read.
  - Instead:
    - Check write buffer first, if not found, then initiate read.
    - CPU stall cycles would be less.

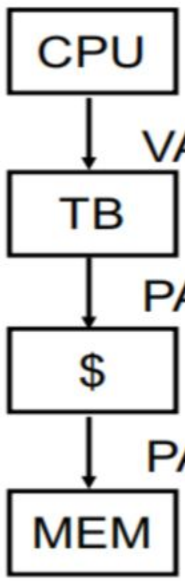
# Reducing Miss Penalty : Read Priority over Write on Miss

- A write buffer with a write through:
  - Allows cache writes to occur at the speed of the cache.
- Write buffer however complicates memory access:
  - They may hold the updated value of a location needed on a read miss.

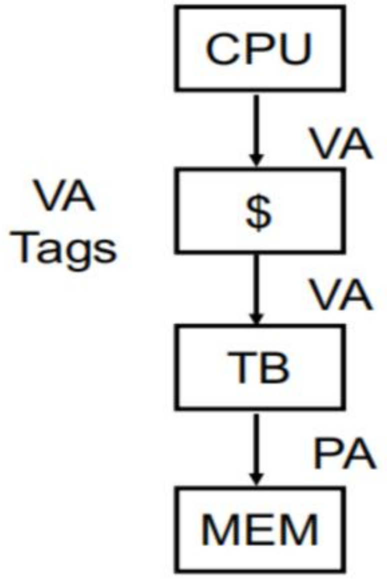
## Reducing Miss Penalty : Read Priority over Write on Miss

- Write-through with write buffers:
  - Read priority over write: Check write buffer contents before read; if no conflicts, let the memory access continue.
  - Write priority over read: Waiting for write buffer to first empty, can increase read miss penalty.

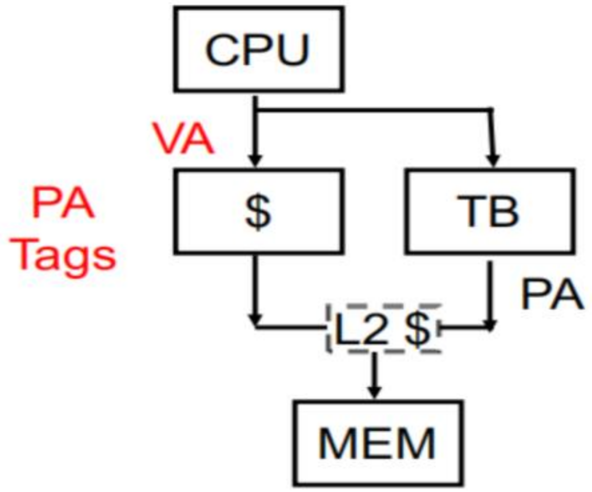
- 5th optimization:Reducing cache hit time
  - Avoiding address translation during indexing of the cache to reduce hit time



Conventional Organization



Virtually Addressed Cache
Translate only on miss



Overlap cache access with VA translation: requires $ index to remain invariant across translation