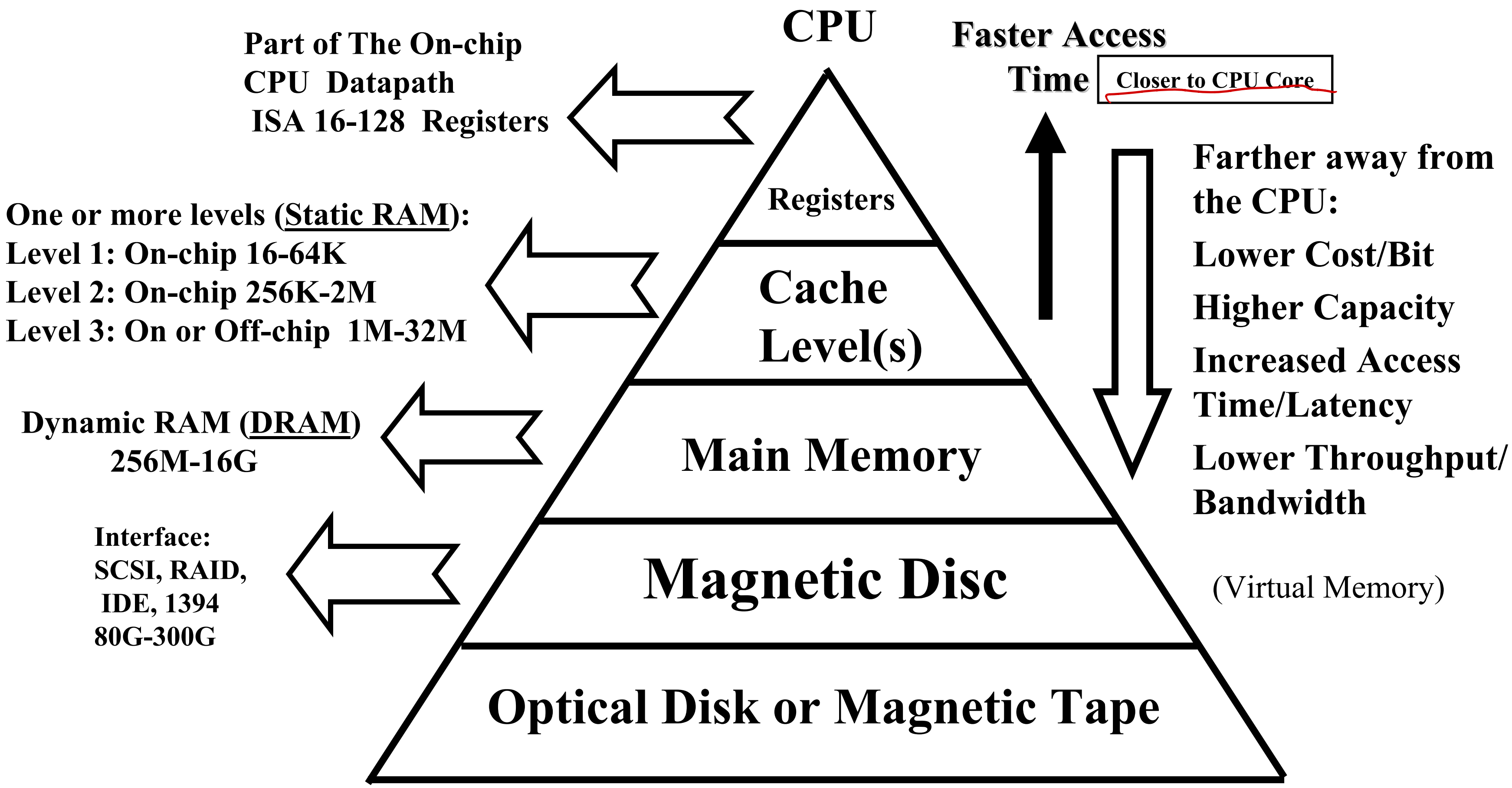
Levels of The Memory Hierarchy

In this course, we concentrate on the design, operation and performance of a single level of cache L1 (either unified or separate) when using non-ideal main memory



CPU

Registers

Cache Level(s)

Main Memory

Magnetic Disc

Optical Disk or Magnetic Tape

Part of The On-chip
CPU Datapath
ISA 16-128 Registers

One or more levels (Static RAM):
Level 1: On-chip 16-64K
Level 2: On-chip 256K-2M
Level 3: On or Off-chip 1M-32M

Dynamic RAM (DRAM)
256M-16G

Interface:
SCSI, RAID,
IDE, 1394
80G-300G

Faster Access Time

Closer to CPU Core

Farther away from the CPU:
Lower Cost/Bit
Higher Capacity
Increased Access Time/Latency
Lower Throughput/ Bandwidth

(Virtual Memory)

4th Edition Chapter 5.1-5.3 - 3rd Edition Chapter 7.1-5.3

Memory Hierarchy Operation

- If an instruction or operand is required by the CPU, the levels of the memory hierarchy are searched for the item starting with the level closest to the CPU (Level 1 cache):
  - If the item is found, it's delivered to the CPU resulting in a cache hit without searching lower levels.

    Hit rate for level one cache $= H_1$

    Hit rate for level one cache $= H_1$

  - If the item is missing from an upper level, resulting in a cache miss, the level just below is searched.

    Cache Miss

    Miss rate for level one cache $= 1 –$ Hit rate $= 1 - H_1$

  - For systems with several levels of cache, the search continues with cache level 2, 3 etc.
  - If all levels of cache report a miss then main memory is accessed for the item.
    - CPU ↔ cache ↔ memory: Managed by hardware.
  - If the item is not found in main memory resulting in a page fault, then disk (virtual memory), is accessed for the item.
    - Memory ↔ disk: Managed by the operating system with hardware support

In this course, we concentrate on the design, operation and performance of a single level of cache L1 (either unified or separate) when using non-ideal main memory

Memory Hierarchy: Terminology

- **A Block:** The smallest unit of information transferred between two levels.
- **Hit:** Item is found in some block in the upper level (example: Block X)
  - **Hit Rate:** The fraction of memory access found in the upper level.
  - **Hit Time:** Time to access the upper level which consists of

    (S)RAM access time + Time to determine hit/miss

e. g. H1

Hit rate for level one cache = H_1

Ideally = 1 Cycle

- **Miss:** Item needs to be retrieved from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level +

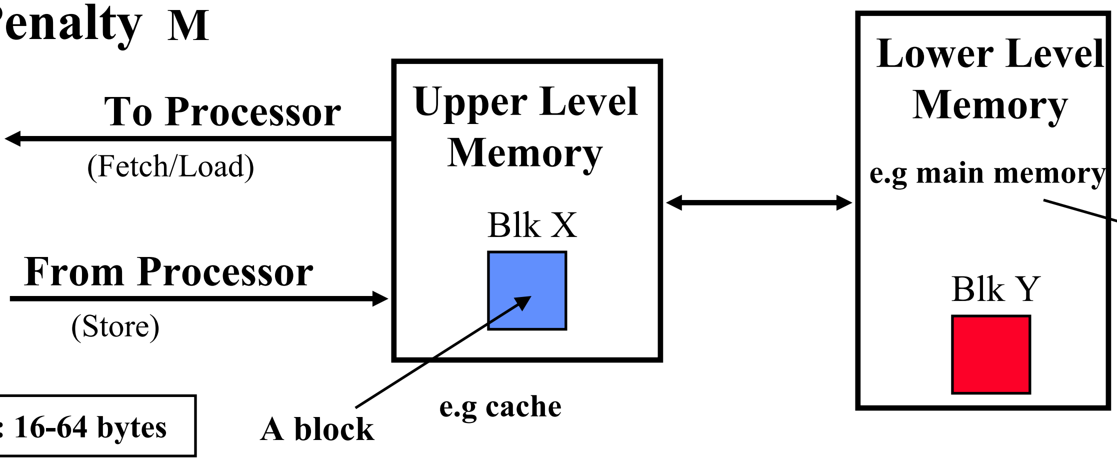
    M Time to deliver the missed block to the processor

e. g. 1- H1

Miss rate for level one cache = 1 – Hit rate = 1 - H_1

- Hit Time << Miss Penalty M

Ideally = 1 Cycle



M Stall cycles on a miss

Typical Cache Block (or line) Size: 16-64 bytes

Hit if block is found in cache

Basic Cache Concepts

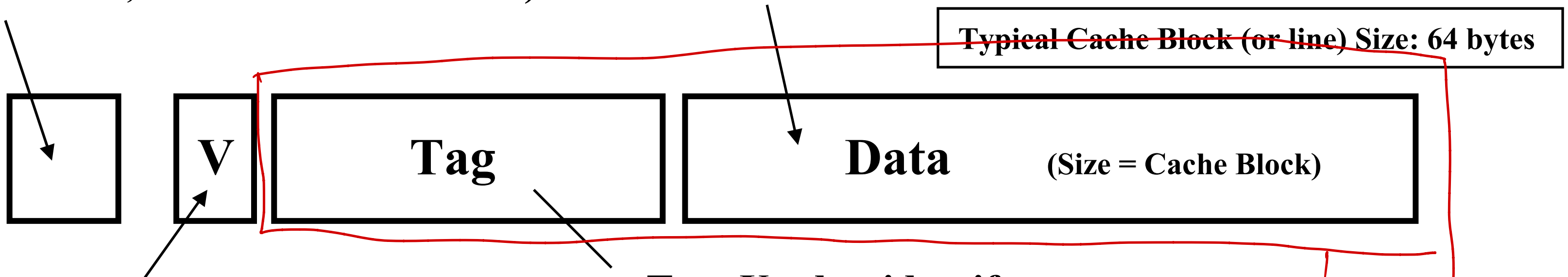
- **Cache is the first level of the memory hierarchy once the address leaves the CPU and is searched first for the requested data.**
- **If the data requested by the CPU is present in the cache, it is retrieved from cache and the data access is a cache hit otherwise a cache miss and data must be read from main memory.**
- **On a cache miss a block of data must be brought in from main memory to cache to possibly *replace* an existing cache block.**
- **The allowed block addresses where blocks can be mapped (placed) into cache from main memory is determined by *cache placement strategy*.**
- **Locating a block of data in cache is handled by cache block identification mechanism (tag checking).**
- **On a cache miss choosing the cache block being removed (replaced) is handled by the *block replacement strategy* in place.**

Cache Block Frame

Cache is comprised of a number of cache block frames

Other status/access bits:
(e,g. modified, read/write access bits)

Data Storage: Number of bytes is the size of a cache block or cache line size (Cached instructions or data go here)



Typical Cache Block (or line) Size: 64 bytes

Valid Bit: Indicates whether the cache block frame contains valid data

Tag: Used to identify if the address supplied matches the address of the data stored

nominal cache capacity

The tag and valid bit are used to determine whether we have a cache hit or miss

Nominal Cache Size

Stated nominal cache capacity or size only accounts for space used to store instructions/data and ignores the storage needed for tags and status bits:

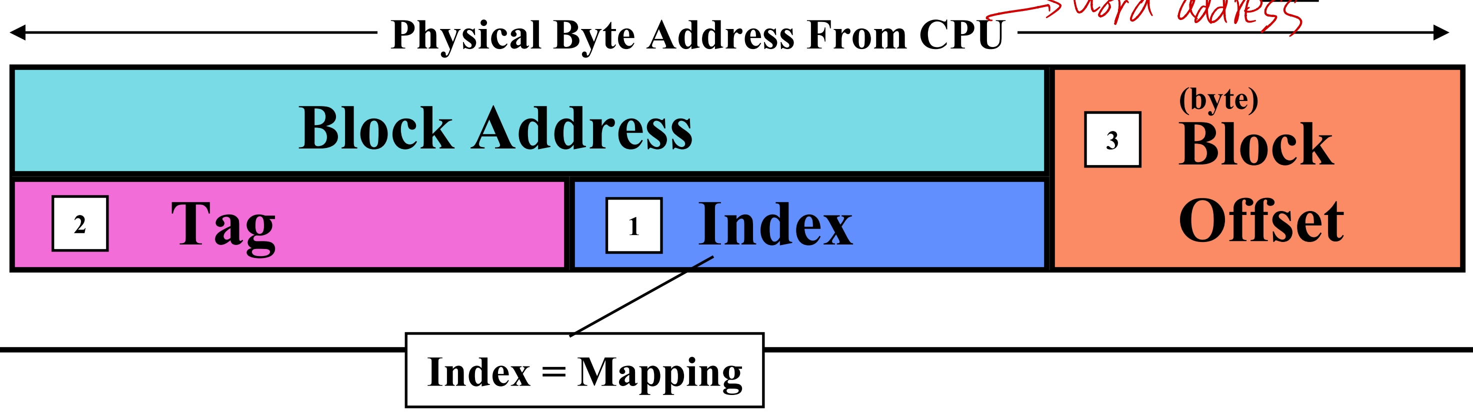
Nominal Cache Capacity = Number of Cache Block Frames x Cache Block Size → for a single cache block

e.g For a cache with block size = 16 bytes and 1024 = 2^{10} = 1k cache block frames
Nominal cache capacity = 16 x 1k = 16 Kbytes

Cache utilizes faster memory (SRAM)

Locating A Data Block in Cache

- Each block frame in cache has an address tag.
- The tags of every cache block that might contain the required data are checked or searched in parallel. Tag Matching
- A valid bit is added to the tag to indicate whether this entry contains a valid address.
- The byte address from the CPU to cache is divided into:
  - A block address, further divided into:
    - 1 • An index field to choose/map a block set in cache. (no index field when fully associative).
    - 2 • A tag field to search and match addresses in the selected set.
  - A byte block offset to select the data from the block. 3



Cache Organization & Placement Strategies

Placement strategies or mapping of a main memory data block onto cache block frame addresses divide cache into three organizations:

1 **Direct mapped cache:** A block can be placed in only one location (cache block frame), given by the mapping function:

Least complex to implement

Mapping Function

index= (Block address) MOD (Number of blocks in cache)

= Frame #

2 **Fully associative cache:** A block can be placed anywhere in cache. (no mapping function).

Most complex cache organization to implement

3 **Set associative cache:** A block can be placed in a restricted set of places, or cache block frames. A set is a group of block frames in the cache. A block is first mapped onto the set and then it can be placed anywhere within the set. The set in this case is chosen by:

Mapping Function

index = (Block address) MOD (Number of sets in cache)

↕ index

= Set #

If there are *n* blocks in a set the cache placement is called *n*-way set-associative.

Most common cache organization

Cache Organization:
Direct Mapped Cache

V | Tag | Data

Cache Block Frame

A block in memory can be placed in one location (cache block frame) only, given by: (Block address) MOD (Number of blocks in cache)

In this case, mapping function: (Block address) MOD (8) = Index

(i.e low three bits of block address)

Assume block size is 8 bytes / block
64 bits

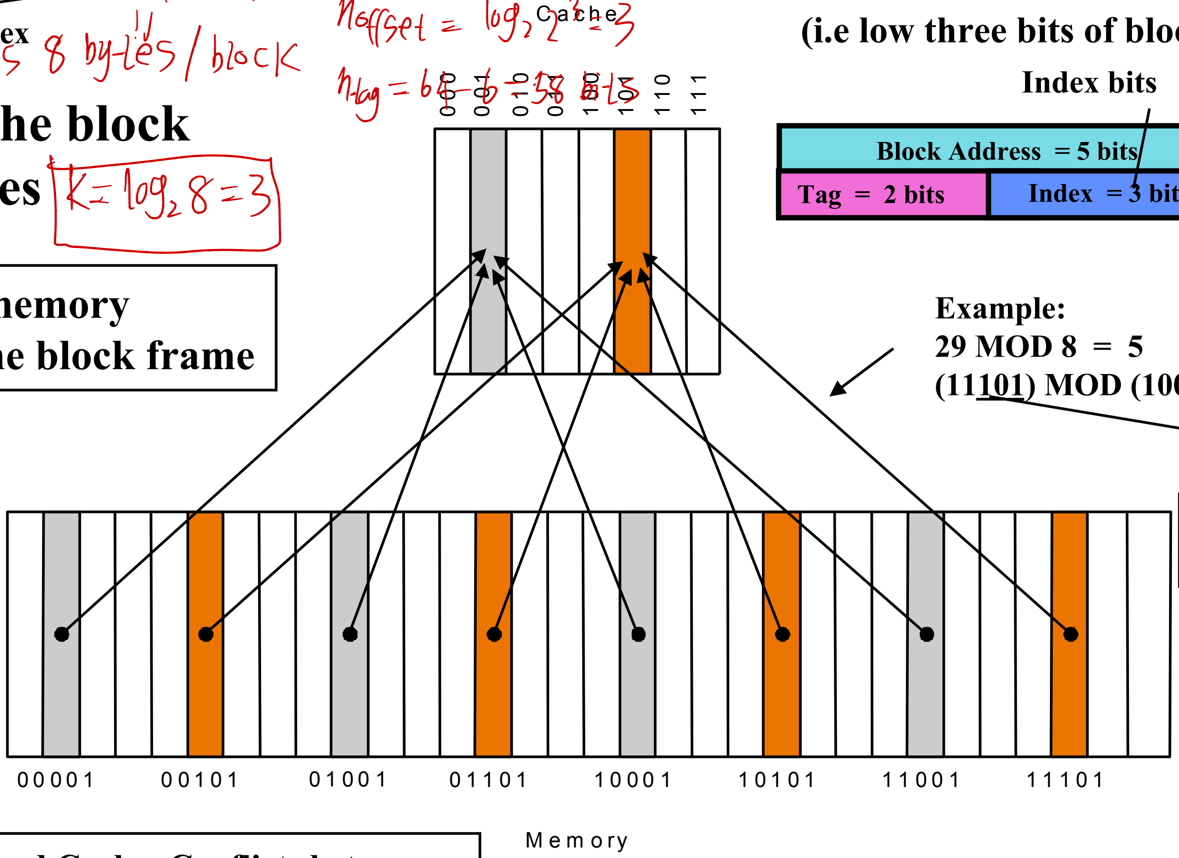
$n_{offset} = \log_2 2^3 = 3$

$n_{tag} = 64 - 6 = 58$ bits

8 cache block frames $\boxed{K = \log_2 8 = 3}$

Here four blocks in memory map to the same cache block frame

32 memory blocks cacheable



Index bits

Block Address = 5 bits		Block offset
Tag = 2 bits	Index = 3 bits	

cannot determine, since the blocksize is unknown

Example:
29 MOD 8 = 5
(11101) MOD (1000) = 101

index

Index size = Log_2 8
= 3 bits

Limitation of Direct Mapped Cache: Conflicts between memory blocks that map to the same cache block frame may result in conflict cache misses

Cache capacity

4KB Direct Mapped Cache Example

(1024 blocks)

4 Kbytes = Nominal Cache Capacity

1K = 2^{10} = 1024 Blocks

Each block = one word (4 bytes)

$n_{offset} = \log_2 2^2 = 2$

||

32 bits

$k = \log_2 2^{10} = 10$

Can cache up to 2^{32} bytes = 4 GB of memory

Mapping function:

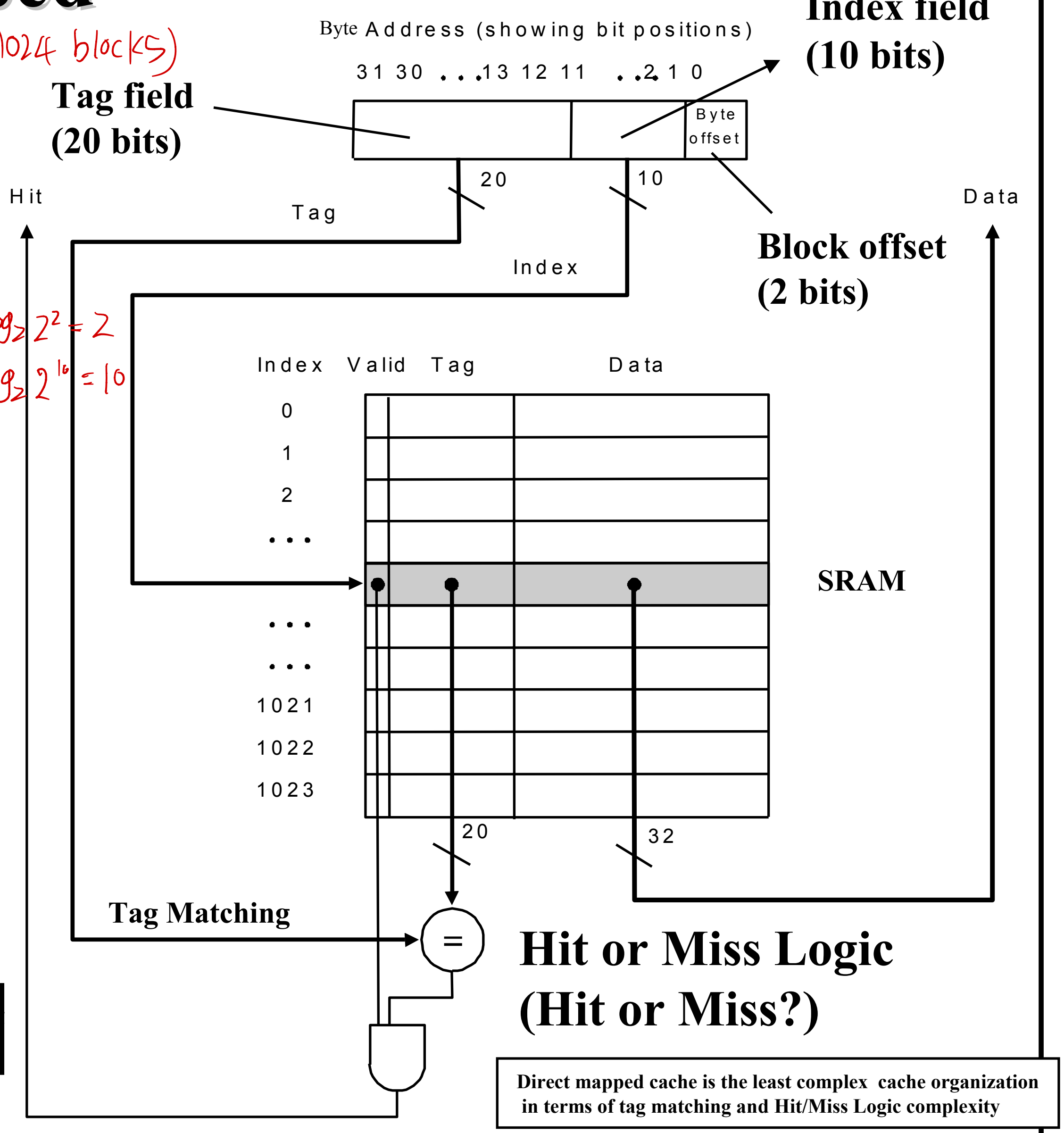
Cache Block frame number = (Block address) MOD (1024)

i.e . Index field or 10 low bits of block address

Block Address = 30 bits		Block offset = 2 bits
Tag = 20 bits	Index = 10 bits	
Tag	Index	Offset

Mapping

Address from CPU



Index field (10 bits)

Tag field (20 bits)

Block offset (2 bits)

SRAM

Hit or Miss Logic (Hit or Miss?)

Direct mapped cache is the least complex cache organization in terms of tag matching and Hit/Miss Logic complexity

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

Direct Mapped Cache Operation Example

- Given a series of 16 memory address references given as word addresses:

  1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17.

  4 bytes = 32 bits

- Assume a direct mapped cache with 16 one-word blocks that is initially empty, label each reference as a hit or miss and show the final content of cache

  $n_{block} = 16$ $K = \log_2 16 = 4$ [illegible]

  $n_{tag} = 32 - 4 - 2 = 26$ bits

  bits

  $n_{offset} = \log_2 4 = 2$ bits

- Here: Block Address = Word Address Mapping Function = (Block Address) MOD 16 = Index

Here:
Block Address = Word Address

Cache Block Frame# index		1	4	8	5	20	17	19	56	9	11	4	43	5	6	9	17	
		Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	$\boxed{\text{Hit}}$	Miss	$\boxed{\text{Hit}}$	$\boxed{\text{Hit}}$	Hit/Miss
0																		
1		$\boxed{1}$	1	1	1	1	$\boxed{17}$	17	17	17	17	17	17	17	17	17	$\boxed{17}$	
2																		
3								$\boxed{19}$	19	19	19	19	19	19	19	19	19	
4			$\boxed{4}$	4	4	$\boxed{20}$	20	20	20	20	20	$\boxed{4}$	4	4	4	4	4	
5					$\boxed{5}$	5	5	5	5	5	5	5	5	$\boxed{5}$	5	5	5	
6															$\boxed{6}$	6	6	
7																		
8				$\boxed{8}$	8	8	8	8	$\boxed{56}$	56	56	56	56	56	56	56	56	
9										$\boxed{9}$	9	9	9	9	9	$\boxed{9}$	9	
10																		
11											$\boxed{11}$	11	$\boxed{43}$	43	43	43	43	
12																		
13																		
14																		
15																		

Initial Cache Content (empty)

Cache Content After Each Reference

Final Cache Content

Hit Rate = # of hits / # memory references = 3/16 = 18.75%

Mapping Function = Index = (Block Address) MOD 16
i.e 4 low bits of block address

64KB Direct Mapped Cache Example

Nominal Capacity

Tag field (16 bits)

4K= 2^{12} = 4096 blocks

Each block = four words = 16 bytes

one of them must be given in problems

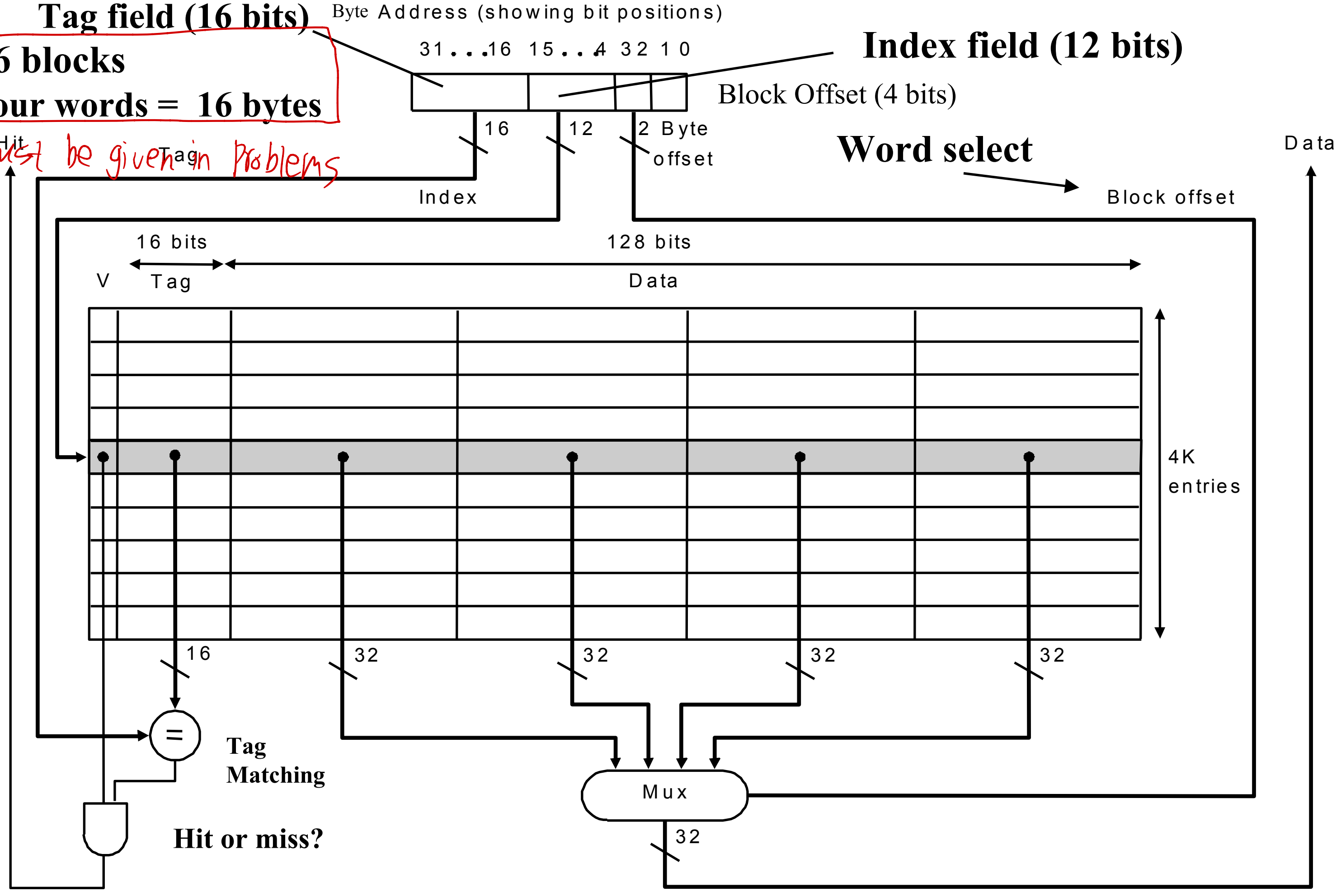
Can cache up to 2^{32} bytes = 4 GB of memory

Index field (12 bits)

Block Offset (4 bits)

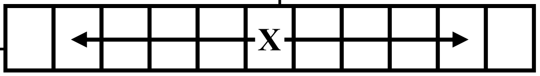
Word select

SRAM



Typical cache Block or line size: 64 bytes

Larger cache blocks take better advantage of spatial locality and thus may result in a lower miss rate



Block Address = 28 bits		Block offset = 4 bits
Tag = 16 bits	Index = 12 bits	

Mapping Function: Cache Block frame number = (Block address) MOD (4096)

i.e. index field or 12 low bit of block address

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

Direct Mapped Cache Operation Example

With Larger Cache Block Frames

- Given the same series of 16 memory address references given as word addresses:

  1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17.

  4 words = 16 bytes = 128 bits $k = \log_2 4 = 2$ $n_{index} = 2$ bits (0~3)

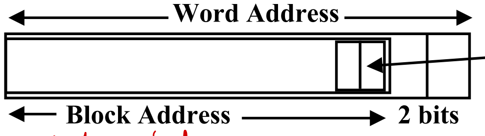
- Assume a direct mapped cache with four word blocks and a total of 16 words that is initially empty, label each reference as a hit or miss and show the final content of cache

  2 4 blocks $n_{offset} = \log_2 16 = 4$ bits

- Cache has 16/4 = 4 cache block frames (each has four words)
- Here: Block Address = Integer (Word Address/4)

i.e We need to find block addresses for mapping

Or



(tag + index)

Mapping Function = (Block Address) MOD 4

(index)

i.e 2 low bits of block address

00 01 10 11

Block addresses		0	1	2	1	5	4	4	14	2	2	1	10	1	1	2	4	
Cache Block Frame#		1	4	8	5	20	17	19	56	9	11	4	43	5	6	9	17	Word addresses
		Miss	Miss	Miss	$\boxed{\text{Hit}}$	Miss	Miss	$\boxed{\text{Hit}}$	Miss	Miss	$\boxed{\text{Hit}}$	Miss	Miss	$\boxed{\text{Hit}}$	$\boxed{\text{Hit}}$	Miss	$\boxed{\text{Hit}}$	Hit/Miss
0		$\boxed{0}$	0	0	0	0	$\boxed{16}$	16	16	16	16	16	16	16	16	16	$\boxed{16}$	
1			$\boxed{4}$	4	$\boxed{4}$	$\boxed{20}$	20	20	20	20	20	$\boxed{4}$	4	$\boxed{4}$	$\boxed{4}$	4	4	
2				$\boxed{8}$	8	8	8	8	$\boxed{56}$	$\boxed{8}$	$\boxed{8}$	8	$\boxed{40}$	40	40	$\boxed{8}$	8	
3																		

Initial Cache Content (empty)

Starting word address of Cache Frames Content After Each Reference

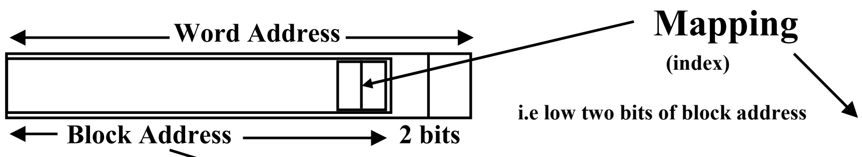
Final Cache Content

Hit Rate = # of hits / # memory references = 6/16 = 37.5%

Here: Block Address ≠ Word Address

Word Addresses vs. Block Addresses and Frame Content for Previous Example

Block size = 4 words



Given Word address	Block address	Cache Block Frame # (Block address)mod 4	word address range in frame (4 words)
1	0	0	0-3
4	1	1	4-7
8	2	2	8-11
5	1	1	4-7
20	5	1	20-23
17	4	0	16-19
19	4	0	16-19
56	14	2	56-59
9	2	2	8-11
11	2	2	8-11
4	1	1	4-7
43	10	2	40-43
5	1	1	4-7
6	1	1	4-7
9	2	2	8-11
17	4	0	16-19

Block Address = Integer (Word Address/4)

Cache Organization: Set Associative Cache

V	Tag	Data

Cache Block Frame

Why set associative?

Set associative cache reduces cache misses by <u>reducing conflicts</u> between blocks that would have been mapped to the same cache block frame in the case of direct mapped cache

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

1-way set associative: (direct mapped) 1 block frame per set

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

2-way set associative: 2 blocks frames per set

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

4-way set associative: 4 blocks frames per set

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

8-way set associative: 8 blocks frames per set In this case it becomes fully associative since total number of block frames = 8

A cache with a total of 8 cache block frames shown

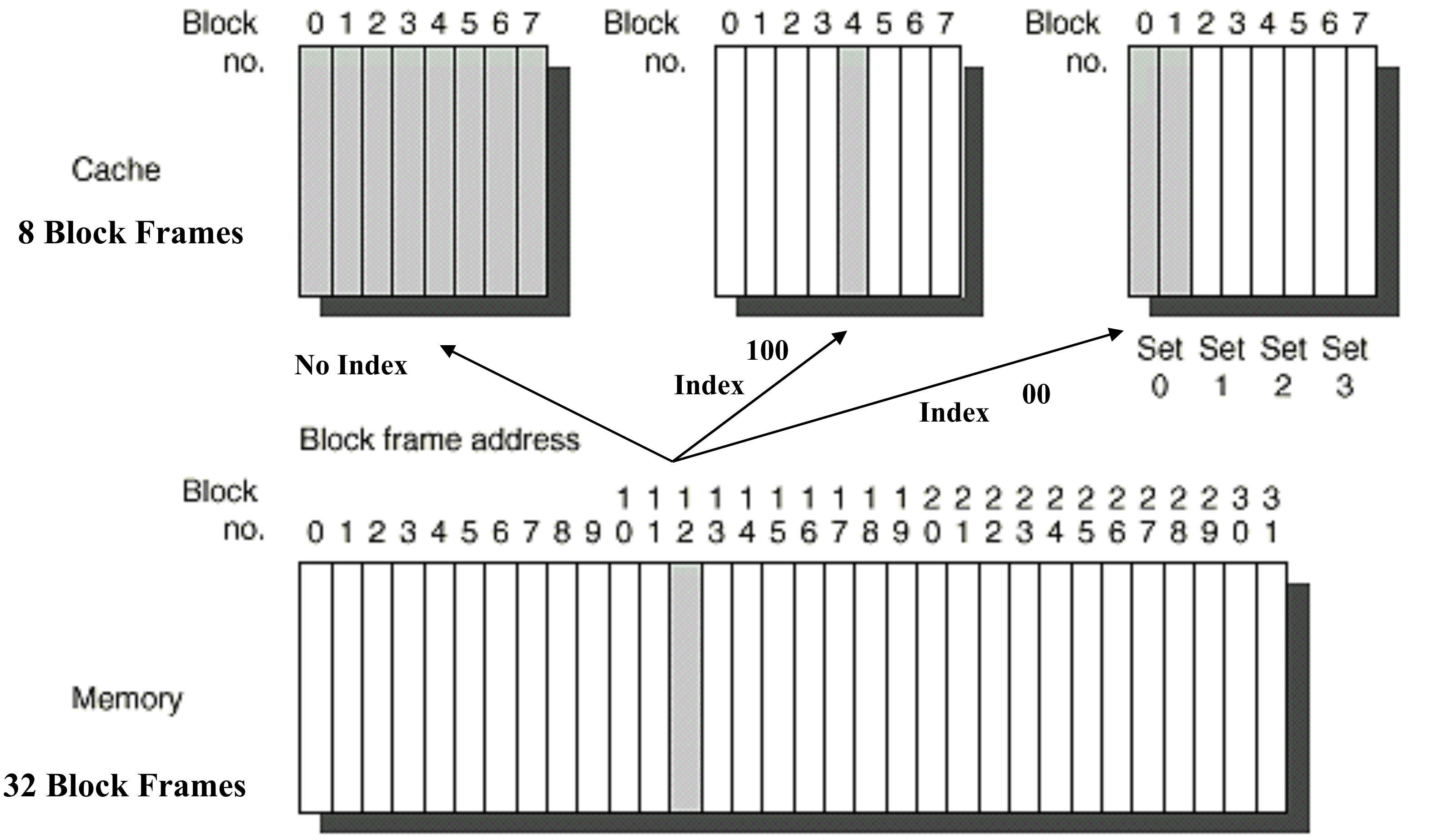
Cache Organization/Mapping Example

Fully associative:
block 12 can go
anywhere

(No mapping function)

Direct mapped:
block 12 can go
only into block 4
(12 mod 8) = index = 100

2-way
Set associative:
block 12 can go
anywhere in set 0
(12 mod 4) = index = 00



This example cache has eight block frames and memory has 32 blocks.

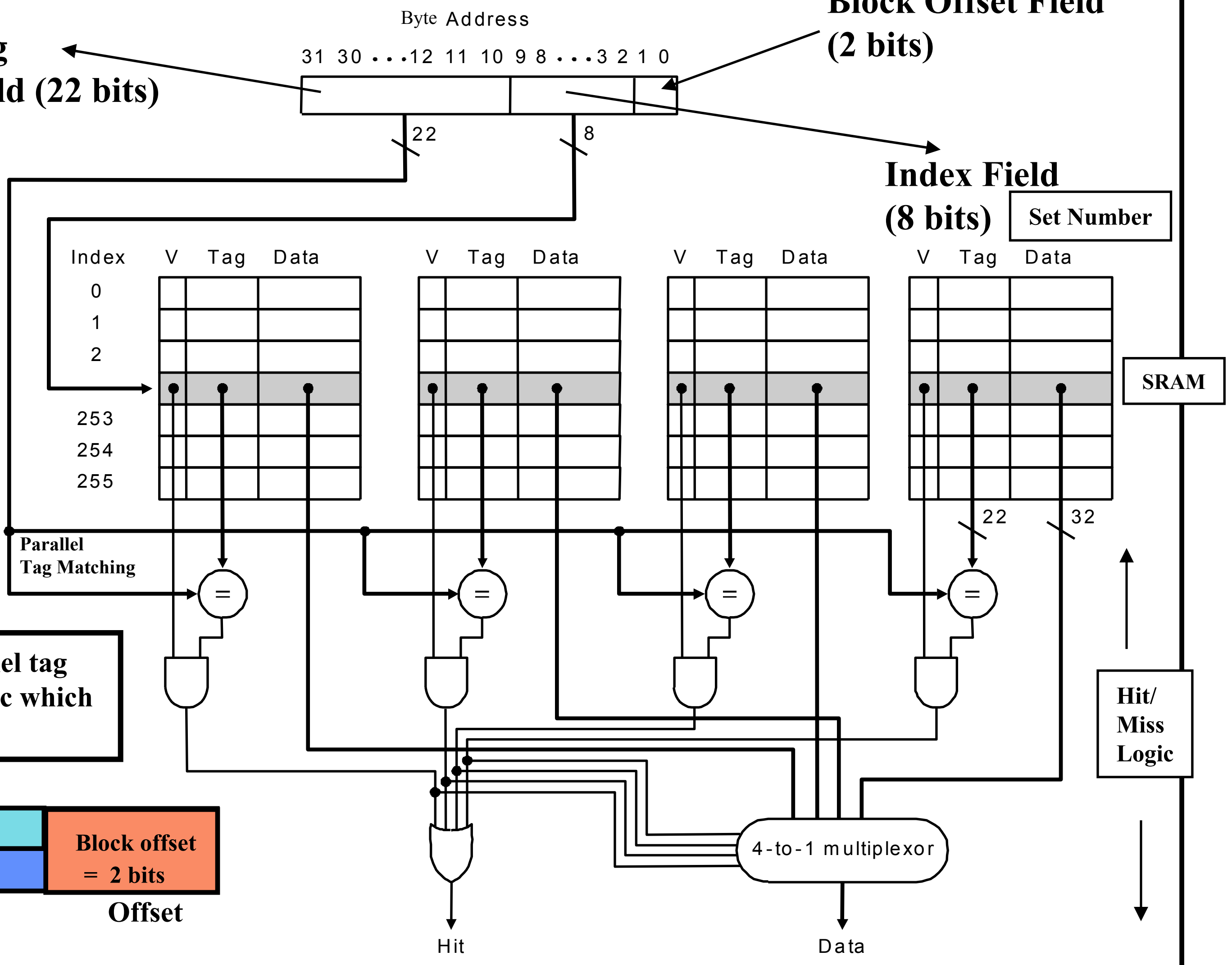
12 = 1100

4K Four-Way Set Associative Cache: MIPS Implementation Example

Nominal Capacity

1024 block frames
Each block = one word
4-way set associative
1024 / 4= 2^8= 256 sets

Can cache up to
2^{32} bytes = 4 GB
of memory



Tag Field (22 bits)

Block Offset Field (2 bits)

Index Field (8 bits)

Set Number

SRAM

Parallel Tag Matching

Hit/ Miss Logic

Set associative cache requires parallel tag matching and more complex hit logic which may increase hit time

Block Address = 30 bits		Block offset = 2 bits
Tag = 22 bits	Index = 8 bits	
Tag	Index	Offset

<u>Mapping Function:</u> Cache Set Number = index= (Block address) MOD (256)

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

Cache Replacement Policy

Which block to replace on a cache miss?

- When a cache miss occurs the cache controller may have to select a block of cache data to be removed from a cache block frame and replaced with the requested data, such a block is selected by one of three methods:

  (No cache replacement policy in direct mapped cache)

  No choice on which block to replace

1 – **Random:**

  - Any block is randomly selected for replacement providing uniform allocation.
  - Simple to build in hardware. Most widely used cache replacement strategy.

2 – **Least-recently used (LRU):**

  - Accesses to blocks are recorded and and the block replaced is the one that was not used for the longest period of time.
  - Full LRU is *expensive* to implement, as the number of blocks to be tracked increases, and is usually approximated by block usage bits that are cleared at regular time intervals.

3 – **First In, First Out (FIFO:**

  - Because LRU can be complicated to implement, this approximates LRU by determining the oldest block rather than LRU

Miss Rates for Caches with Different Size, Associativity & Replacement Algorithm

Sample Data

Nominal

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Program steady state cache miss rates are given
Initially cache is empty and miss rates ~ 100%

FIFO replacement miss rates (not shown here) is <u>better than random</u> but <u>worse than LRU</u>

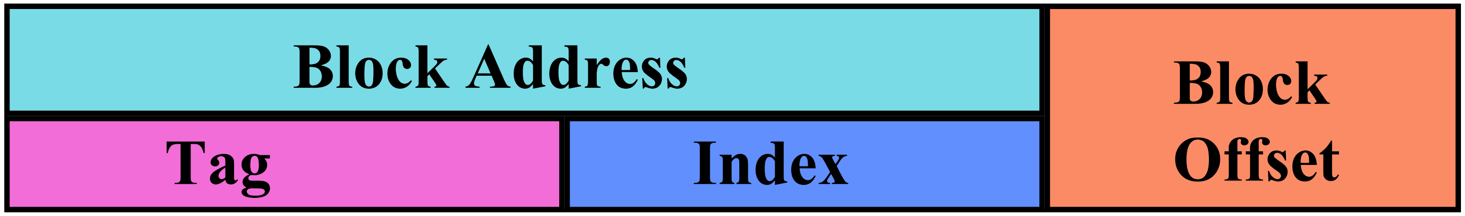
For SPEC92

Miss Rate = 1 – Hit Rate = 1 – H1

Address Field Sizes/Mapping

← Physical Address Generated by CPU →

(The size of this address depends on amount of cacheable physical main memory)



Block offset size = $\log_2$(block size)

Index size = $\log_2$(Total number of blocks/associativity)

Number of Sets in cache

Tag size = address size - index size - offset size

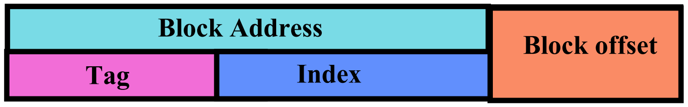
<u>Mapping function:</u> (From memory block to cache)

Cache set or block frame number = Index =

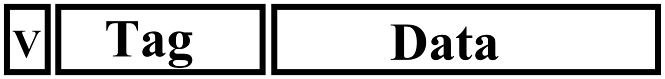
= (Block Address) MOD (Number of Sets)

Fully associative cache has no index field or mapping function
e.g. no index field

Calculating Number of Cache Bits Needed



Address Fields



Cache Block Frame (or just cache block)

- How many total bits are needed for a direct- mapped cache with 64 KBytes = 16K words of data and one word blocks, assuming a 32-bit address? 16K = 2^{14}

  i.e nominal cache Capacity = 64 KB

  - 64 Kbytes = 16 K words = 2^{14} words = 2^{14} blocks
  - Block size = 4 bytes => offset size = $\log_2(4)$ = 2 bits, $n_{offset} = \log_2 4 = 2$ bits
  - #sets = #blocks = 2^{14} => index size = 14 bits $n_{index} = \log_2 2^{14} = 14$ bits
  - Tag size = address size - index size - offset size = 32 - 14 - 2 = 16 bits $n_{tag} = 16$ bits
  - Bits/block = data bits + tag bits + valid bit = 32 + 16 + 1 = 49
  - Bits in cache = #blocks x bits/block = 2^{14} x 49 = 98 Kbytes

  Number of cache block frames

  Actual number of bits in a cache block frame

- How many total bits would be needed for a 4-way set associative cache to store the same amount of data?
  - Block size and #blocks does not change. 2^{14}
  - #sets = #blocks/4 = $(2^{14})/4$ = 2^{12} => index size = 12 bits $n_{set} = 2^{14}/4 = 2^{12}$ => $n_{index} = n_{set} = 12$ bits
  - Tag size = address size - index size - offset = 32 - 12 - 2 = 18 bits $n_{tag} = 18$ bits
  - Bits/block = data bits + tag bits + valid bit = 32 + 18 + 1 = 51
  - Bits in cache = #blocks x bits/block = 2^{14} x 51 = 102 Kbytes

- Increase associativity => increase bits in cache

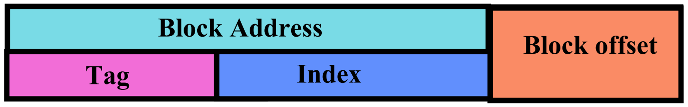
More bits in tag

Word = 4 bytes

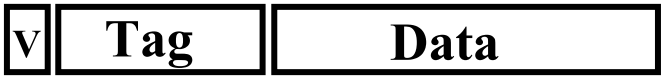
1 k = 1024 = 2^{10}

??

Calculating Cache Bits Needed



Address Fields



Cache Block Frame (or just cache block)

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and 8 word (32 byte) blocks, assuming a 32-bit address (it can cache 2^{32} bytes in memory)?

  $n_{block} = 2K = 2^{11}$

  - 64 Kbytes = 2^{14} words = $(2^{14})/8 = 2^{11}$ blocks

    Number of cache block frames

    bits

  - block size = 32 bytes

    $n_{offset} = \log_2 32 = 5$ $n_{index} = \log_2 2^{11} = 11$ bits $n_{tag} = 16$ bits

    => offset size = block offset + byte offset = $\log_2(32)$ = 5 bits,

  - #sets = #blocks = 2^{11} => index size = 11 bits

    bits / block = $n_{tag} + n_{valid} + n_{data}$

    $= 8 \times 32 + 16 + 1 = 273$ bits

  - tag size = address size - index size - offset size = 32 - 11 - 5 = 16 bits

    ??

  - bits/block = data bits + tag bits + valid bit = 8 x 32 + 16 + 1 = 273 bits

    Actual number of bits in a cache block frame

  - bits in cache = #blocks x bits/block = 2^{11} x 273 = 68.25 Kbytes

- Increase block size => decrease bits in cache.

Fewer cache block frames thus fewer tags/valid bits

Word = 4 bytes 1 k = 1024 = 2^{10}

Unified vs. Separate Level 1 Cache

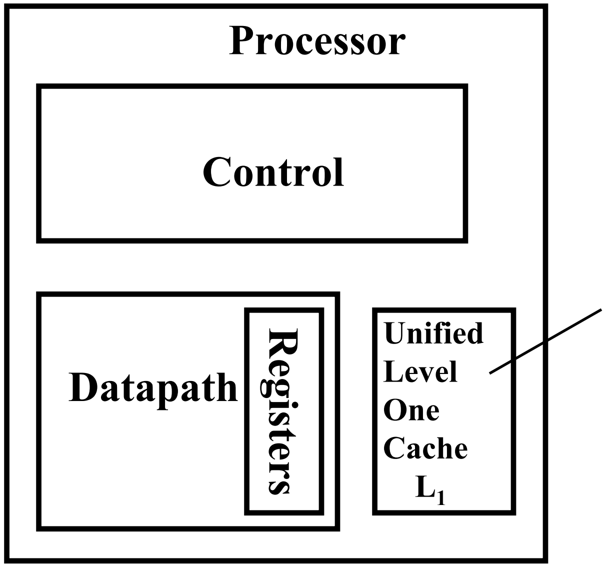
- **Unified Level 1 Cache (Princeton Memory Architecture).** AKA Shared Cache

  A single level 1 (L_1) cache is used for both instructions and data.

Or Split

- **Separate instruction/data Level 1 caches (Harvard Memory Architecture):**

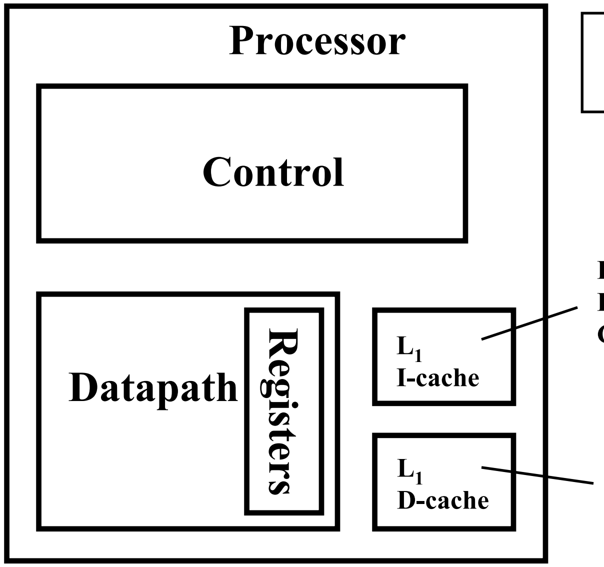
  The level 1 (L_1) cache is split into two caches, one for instructions (instruction cache, L_1 I-cache) and the other for data (data cache, L_1 D-cache).



Accessed for both instructions And data

AKA shared

Unified Level 1 Cache
(Princeton Memory Architecture)



Most Common

Instruction Level 1 Cache

Data Level 1 Cache

Separate (Split) Level 1 Caches
(Harvard Memory Architecture)

Split Level 1 Cache is more preferred in pipelined CPUs to avoid instruction fetch/Data access structural hazards

Memory Hierarchy/Cache Performance:

Average Memory Access Time (AMAT), Memory Stall cycles

- <u>The Average Memory Access Time (AMAT):</u> The number of cycles required to complete an average memory access request by the CPU.
- <u>Memory stall cycles per memory access:</u> The number of stall cycles added to CPU execution cycles for one memory access.
- Memory stall cycles per average memory access = (AMAT -1)
- For ideal memory: AMAT = 1 cycle, this results in zero memory stall cycles.

→ just hit

AMAT = hit time + penalty time · miss rate

- Memory stall cycles per average instruction =

Number of memory accesses per instruction

x Memory stall cycles per average memory access

Instruction Fetch

= (1 + fraction of loads/stores) x (AMAT -1)

fetch, load, store

(1)

Base CPI = $\text{CPI}_{\text{execution}}$ = CPI with ideal memory

CPI = $\text{CPI}_{\text{execution}}$ + Mem Stall cycles per instruction

cycles = CPU cycles

Cache Performance:
Single Level L1 Princeton (Unified) Memory Architecture

CPUtime = Instruction count x CPI x Clock cycle time

$\text{CPI}_{\text{execution}}$ = CPI with ideal memory

$AMAT = H1 \cdot 1 + (1-H1)(1+M) = 1 + M(1-H1)$

CPI = $\text{CPI}_{\text{execution}}$ + Mem Stall cycles per instruction

Mem Stall cycles per instruction =

Memory accesses per instruction x Memory stall cycles per access

i.e No hit penalty

Assuming no stall cycles on a cache hit (cache access time = 1 cycle, stall = 0)

Cache Hit Rate = H1 Miss Rate = 1- H1 Miss Penalty = M

Memory stall cycles per memory access = Miss rate x Miss penalty = (1- H1) x M

$M(1-H1) = AMAT - 1 \Rightarrow AMAT = 1 + M(1-H1)$

AMAT = 1 + Miss rate x Miss penalty

Memory accesses per instruction = (1 + fraction of loads/stores)

Miss Penalty = M = the number of stall cycles resulting from missing in cache

= Main memory access time - 1

Thus for a unified L1 cache with no stalls on a cache hit:

→ CPI = $\text{CPI}_{\text{execution}}$ + (1 + fraction of loads/stores) x (1 - H1) x M

AMAT = 1 + (1 - H1) x M

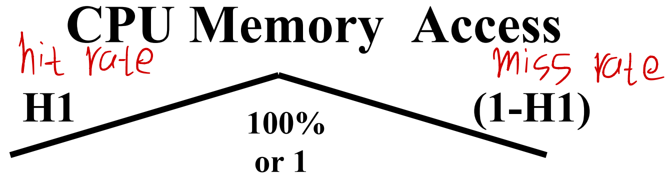
CPI = $\text{CPI}_{\text{execution}}$ + (1 + fraction of loads and stores) x stall cycles per access

= $\text{CPI}_{\text{execution}}$ + (1 + fraction of loads and stores) x (AMAT – 1)

Memory Access Tree:
For Unified Level 1 Cache

miss penalty=M

Probability to be here


CPU Memory Access

hit rate

H1

100%
or 1

miss rate

(1-H1)

Unified
L_1

L1 Hit:
% = Hit Rate = H1
Hit Access Time = 1
Stall cycles per access = 0
Stall= H1 x 0 = 0
(No Stall)

L1 Miss:
% = (1- Hit rate) = (1-H1)
Access time = M + 1
Stall cycles per access = M
Stall = M x (1-H1)

Assuming:
~~Ideal access~~ on a hit

$$AMAT = \overset{\text{Hit Rate}}{H1} \times \overset{\text{Hit Time}}{1} + \overset{\text{Miss Rate}}{(1 - H1)} \times \overset{\text{Miss Time}}{(M+1)} = 1 + M \times (1 - H1)$$

Stall Cycles Per Access = AMAT - 1 = M x (1 -H1)

$CPI = CPI_{execution}$ + (1 + fraction of loads/stores) x M x (1 -H1)

M = Miss Penalty = stall cycles per access resulting from missing in cache
M + 1 = Miss Time = Main memory access time
H1 = Level 1 Hit Rate 1- H1 = Level 1 Miss Rate

AMAT = 1 + Stalls per average memory access

Cache Performance Example

- Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache. $CPI = CPI_{ideal} + CPI_{stall} = CPI_{ideal} + \text{memory access / cycle} \times \text{cycle / access}$
- $CPI_{execution} = 1.1$ (i.e base CPI with ideal memory) $= CPI_{ideal} + (\text{fetch} + \text{load/store})(\text{miss rate} \cdot \text{miss penalty})$
- Instruction mix: 50% arith/logic, 30% load/store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of M= 50 cycles.

0.015 $= 1.1 + (1+0.3) \times (0.015 \times 50)$

$CPI = CPI_{execution} + \text{mem stalls per instruction}$

Mem Stalls per instruction =

(1- H1) = 2.075 M

Mem accesses per instruction x Miss rate x Miss penalty

Mem accesses per instruction = 1 + .3 = 1.3

Instruction fetch Load/store

Mem Stalls per memory access = (1- H1) x M = .015 x 50 = .75 cycles

AMAT = 1 +.75 = 1.75 cycles

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

M = Miss Penalty = stall cycles per access resulting from missing in cache

Cache Performance Example

- Suppose for the previous example we double the clock rate to 400 MHz, how much faster is this machine, assuming similar miss rate, instruction mix?
- Since memory speed is not changed, the miss penalty takes more CPU cycles:

$$\text{Miss penalty} = M = 50 \times 2 = 100 \text{ cycles.}$$

$$\text{CPI} = 1.1 + 1.3 \times .015 \times 100 = 1.1 + 1.95 = 3.05$$

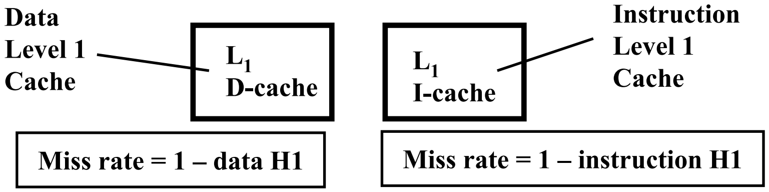
$$\text{Speedup} = (\text{CPI}_{old} \times C_{old}) / (\text{CPI}_{new} \times C_{new})$$

$$= 2.075 \times 2 / 3.05 = 1.36$$

The new machine is only 1.36 times faster rather than 2 times faster due to the increased effect of cache misses.

→ ***CPUs with higher clock rate, have more cycles per cache miss and more memory impact on CPI.***

Cache Performance:



Usually: Data Miss Rate >> Instruction Miss Rate

Single Level L1 Harvard (Split) Memory Architecture

For a CPU with separate or split level one (L1) caches for instructions and data (Harvard memory architecture) and no stalls for cache hits:

CPUtime = Instruction count x CPI x Clock cycle time

$$\boxed{\text{CPI} = \overset{\text{ideal CPI}}{\underline{\text{CPI}_{\text{execution}}}} + \text{Mem Stall cycles per instruction}}$$

This is one method to find stalls per instruction another method is shown in next slide →

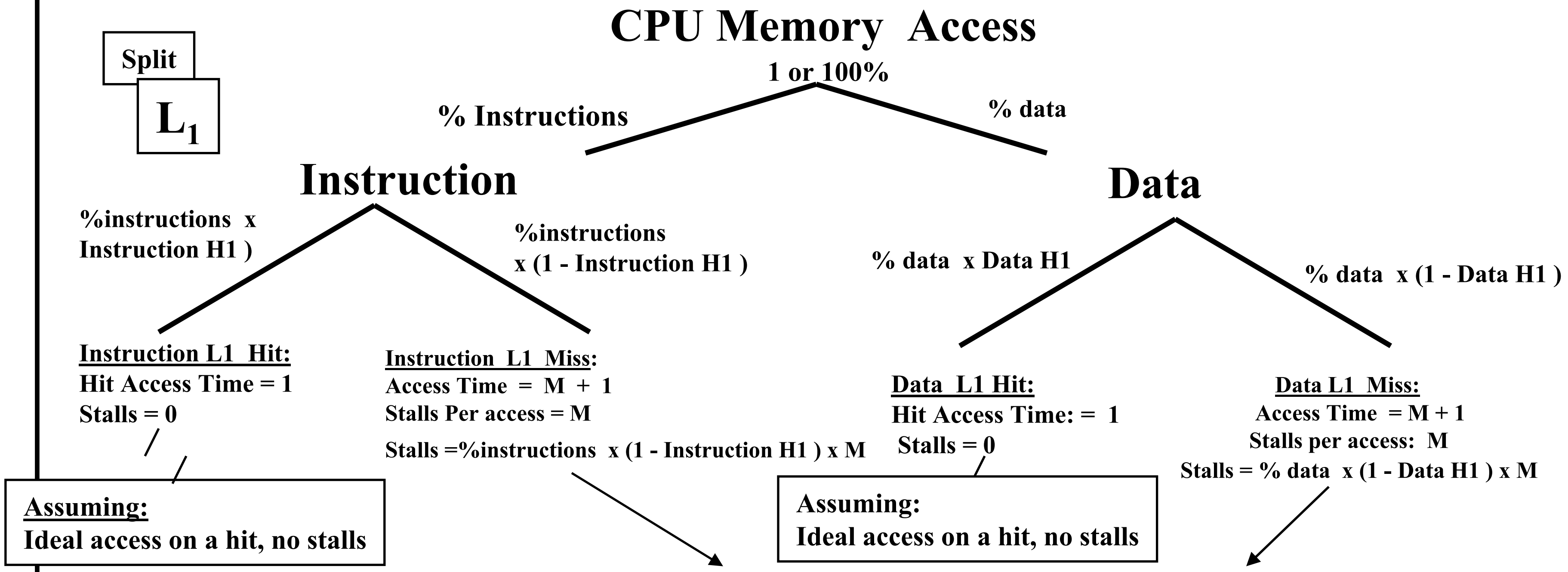
Mem Stall cycles per instruction =

$$\underbrace{\text{Instruction Fetch Miss rate}}_{\text{1- Instruction H1}} \text{ x } \overset{\text{miss penalty}}{\underline{\text{M}}} + \underbrace{\text{Data Memory Accesses Per Instruction}}_{\text{Fraction of Loads and Stores}} \text{ x } \underbrace{\text{Data Miss Rate}}_{\text{1- Data H1}} \text{ x M}$$

M = Miss Penalty = stall cycles per access to main memory resulting from missing in cache

$\text{CPI}_{\text{execution}}$ = base CPI with ideal memory

Memory Access Tree
For Separate Level 1 Caches



Split

L_1

CPU Memory Access

1 or 100%

% Instructions

% data

Instruction

Data

%instructions x Instruction H1)

%instructions x (1 - Instruction H1)

% data x Data H1

% data x (1 - Data H1)

Instruction L1 Hit:
Hit Access Time = 1
Stalls = 0

Instruction L1 Miss:
Access Time = M + 1
Stalls Per access = M
Stalls =%instructions x (1 - Instruction H1) x M

Data L1 Hit:
Hit Access Time: = 1
Stalls = 0

Data L1 Miss:
Access Time = M + 1
Stalls per access: M
Stalls = % data x (1 - Data H1) x M

Assuming:
Ideal access on a hit, no stalls

Assuming:
Ideal access on a hit, no stalls

Stall Cycles Per Access = % Instructions x (1 - Instruction H1) x M + % data x (1 - Data H1) x M

AMAT = 1 + Stall Cycles per access

Stall cycles per instruction = (1 + fraction of loads/stores) x Stall Cycles per access

CPI = $\text{CPI}_{\text{execution}}$ + Stall cycles per instruction
= $\text{CPI}_{\text{execution}}$ + (1 + fraction of loads/stores) x Stall Cycles per access

M = Miss Penalty = stall cycles per access resulting from missing in cache	
M + 1 = Miss Time = Main memory access time	
Data H1 = Level 1 Data Hit Rate	1- Data H1 = Level 1 Data Miss Rate
Instruction H1 = Level 1 Instruction Hit Rate	1- Instruction H1 = Level 1 Instruction Miss Rate
% Instructions = Percentage or fraction of instruction fetches out of all memory accesses	
% Data = Percentage or fraction of data accesses out of all memory accesses	

Split L1 Cache Performance Example

- Suppose a CPU uses separate level one (L1) caches for instructions and data (Harvard memory architecture) with different miss rates for instruction and data access:
  - A cache hit incurs no stall cycles while a cache miss incurs 200 stall cycles for both memory reads and writes.
  - $CPI_{execution} = 1.1$

    (i.e base CPI with ideal memory)
  - Instruction mix: 50% arith/logic, 30% load/store, 20% control
  - Assume a cache miss rate of 0.5% for instruction fetch and a cache data miss rate of 6%.
  - A cache hit incurs no stall cycles while a cache miss incurs 200 stall cycles for both memory reads and writes.

    M
- **Find the resulting stalls per access, AMAT and CPI using this cache?**

$CPI = CPI_{execution}$ + mem stalls per instruction

Memory Stall cycles per instruction = 1 · Instruction Fetch Miss rate x Miss Penalty +
Data Memory Accesses Per Instruction x Data Miss Rate x Miss Penalty

Memory Stall cycles per instruction = 0.5/100 x 200 + 0.3 x 6/100 x 200 = 1 + 3.6 = 4.6 cycles

Stall cycles per average memory access = 4.6/1.3 = 3.54 cycles

AMAT = 1 + Stall cycles per average memory access = 1 + 3.54 = 4.54 cycles

$CPI = CPI_{execution}$ + mem stalls per instruction = 1.1 + 4.6 = 5.7 cycles

- **What is the miss rate of a single level unified cache that has the same performance?**

4.6 = 1.3 x Miss rate x 200 which gives a miss rate of 1.8 % for an equivalent unified cache

- **How much faster is the CPU with ideal memory?**

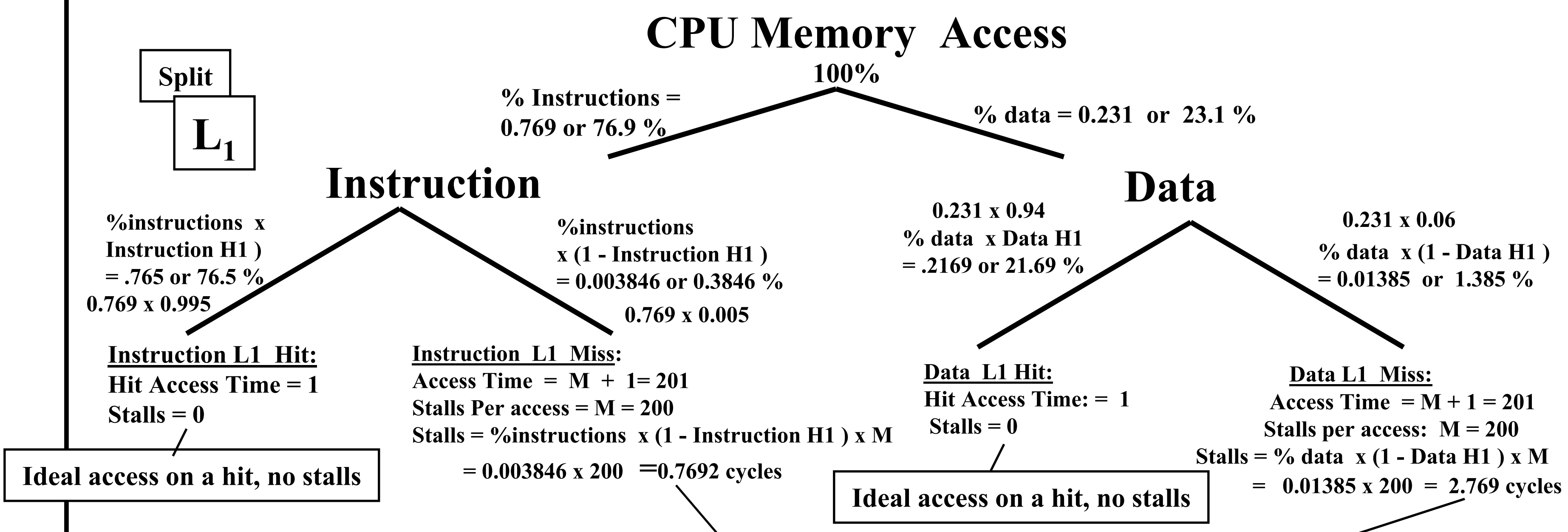
The CPU with ideal cache (no misses) is 5.7/1.1 = 5.18 times faster

With no cache at all the CPI would have been = 1.1 + 1.3 X 200 = 261.1 cycles !!

Memory Access Tree For Separate Level 1 Caches Example

For Last Example

30% of all instructions executed are loads/stores, thus:
Fraction of instruction fetches out of all memory accesses = 1/ (1+0.3) = 1/1.3 = 0.769 or 76.9 %
Fraction of data accesses out of all memory accesses = 0.3/ (1+0.3) = 0.3/1.3 = 0.231 or 23.1 %



Split

L_1

CPU Memory Access

100%

% Instructions =
0.769 or 76.9 %

% data = 0.231 or 23.1 %

Instruction

Data

%instructions x
Instruction H1)
= .765 or 76.5 %
0.769 x 0.995

%instructions
x (1 - Instruction H1)
= 0.003846 or 0.3846 %
0.769 x 0.005

0.231 x 0.94
% data x Data H1
= .2169 or 21.69 %

0.231 x 0.06
% data x (1 - Data H1)
= 0.01385 or 1.385 %

Instruction L1 Hit:
Hit Access Time = 1
Stalls = 0

Ideal access on a hit, no stalls

Instruction L1 Miss:
Access Time = M + 1= 201
Stalls Per access = M = 200
Stalls = %instructions x (1 - Instruction H1) x M
= 0.003846 x 200 =0.7692 cycles

Data L1 Hit:
Hit Access Time: = 1
Stalls = 0

Ideal access on a hit, no stalls

Data L1 Miss:
Access Time = M + 1 = 201
Stalls per access: M = 200
Stalls = % data x (1 - Data H1) x M
= 0.01385 x 200 = 2.769 cycles

Stall Cycles Per Access = % Instructions x (1 - Instruction H1) x M + % data x (1 - Data H1) x M
= 0.7692 + 2.769 = 3.54 cycles

AMAT = 1 + Stall Cycles per access = 1 + 3.5 = 4.54 cycles

Stall cycles per instruction = (1 + fraction of loads/stores) x Stall Cycles per access = 1.3 x 3.54 = 4.6 cycles

CPI = $\text{CPI}_{\text{execution}}$ + Stall cycles per instruction = 1.1 + 4.6 = 5.7

Given as 1.1

M = Miss Penalty = stall cycles per access resulting from missing in cache = 200 cycles
M + 1 = Miss Time = Main memory access time = 200+1 =201 cycles L1 access Time = 1 cycle
Data H1 = 0.94 or 94% 1- Data H1 = 0.06 or 6%
Instruction H1 = 0.995 or 99.5% 1- Instruction H1 = 0.005 or 0.5 %
% Instructions = Percentage or fraction of instruction fetches out of all memory accesses = 76.9 %
% Data = Percentage or fraction of data accesses out of all memory accesses = 23.1 %