8/1/24

Computer Architecture : A Quantitative Approach (5th Edition)
→ Hennessy

UNIT 1

FUNDAMENTALS OF QUANTITATIVE DESIGN & ANALYSIS

Classes of computer

PERSONAL MOBILE DEVICE
- should be cost effective, battery operated → minimal power consumption desired
- portable device, compact → area efficient
- key characteristics expected : responsiveness → minimal response time ⇒ maximum efficiency
  predictability

DESKTOP COMPUTING
- used for performing repeated operations
- run some general-purpose applications

Performance → computer performance, graphics performance → microprocessor choice

SERVERS
- large scale enterprise computing
- availability, scalability, efficient throughput
- efficiency & cost effectiveness
- responsiveness to an individual request
  ⇒ key metric : how many requests can be handled in unit time

CLUSTERS
- group of computers connected through LAN

EMBEDDED COMPUTERS
- application specific
- all the requirements for the application are present within a given area (mounted on Si area)
- key characteristics : power, cost
- primary goal : meet performance requirements at a minimum price

Performance parameters
- area
- power
- speed

* Larger area, more power is consumed in interacting with the components and the speed reduces (takes longer due to larger distance)

PARALLELISM IN APPLICATIONS

Data-level (DLP)
arises because there are many data items that can be operated on at the same time

Task/~~Instruction~~-level (TLP)
arises because tasks of work are created that can operate independently and largely in parallel

Pseudo parallelism ⇒ Pipelining

TASK - Smallest operation carried out by a device in order to meet the requirements of an application

Computer hardware exploits application parallelism
- INSTRUCTION-LEVEL PARALLELISM (ILP)
- VECTOR ARCHITECTURES & GPUs (Unit 3)
- THREAD-LEVEL PARALLELISM (Unit 4)
- REQUEST-LEVEL PARALLELISM
  - Modularity
  - when stringent execution executed/expected

DLP - Different instructions being performed on the same data items

ILP - Same instruction is given to multiple execution units simultaneously after reading the instruction only once
- uses pipelining and speculative execution (Unit 2)

THREAD - Smallest active element that can be identified by the OS

VECTORIZATION - Try to perform similar set of operations for a large amount of data

TIGHTLY COUPLED HARDWARE - Expected response time is exactly matched; no relaxation

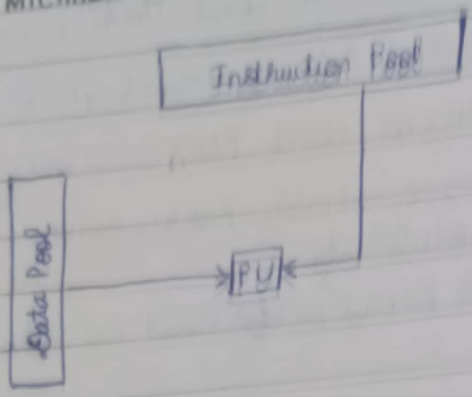
DECOUPLED TASK - Break a task and handle simultaneously based on request

9/1/24

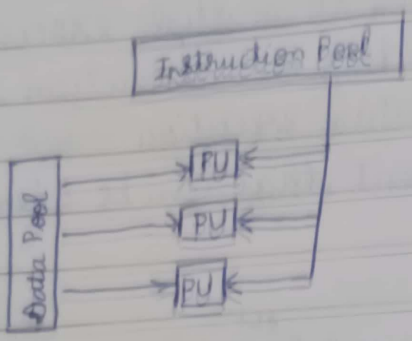
Harvard architecture

Von neumann architecture

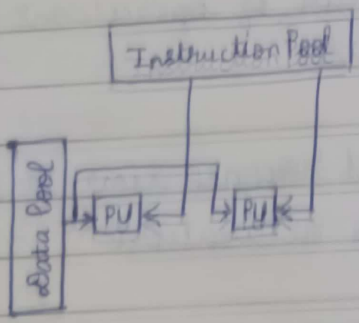
MICHAEL FLYNN Classes of Parallelism & Parallel Architectures



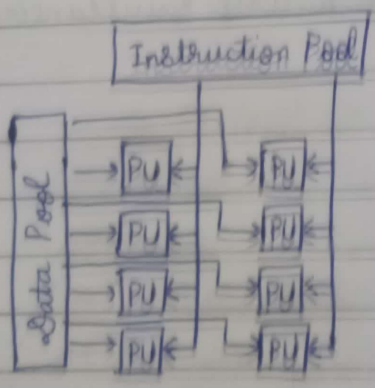
SISD Single Instruction, Sing
Data Stream



SIMD Single Instruction, Multiple
Data stream



MISD Multiple Instruction, Single
Data stream



MIMD Multiple Instruction, Multip
Data stream
Each processor fetches its own
instructions & operates on its own d
& it targets task-level parallelism

* CISC → Efficient pipelining is not possible

CA involves many aspects, including
- instruction set design
- functional organization, logic design, microarchitecture
- implementation

INSTRUCTION SET ARCHITECTURE (ISA)
↳ serves as the boundary between the software & hardware

(1) CLASS OF ISA
- Register-memory ISAs which can access memory as part of many instructions
  Eg: CISC 80x86
- load-store ISAs which can access memory only with load/store instructions
  Eg: RISC

(2) MEMORY ADDRESSING
- Byte addressable
- Word addressable — An access of object of size s bytes at byte address A is aligned if A mod s = 0 (checks alignment)
  RISC

* Pure RISC machine ⇒ Every instruction is orthonormal to each other
  ⇒ Only 32 instructions possible
  (misaligned data is not desired)
* ARM ⇒ 53 instructions possible

(3) ADDRESSING MODE
↳ the way in which an operand is being accessed by an instruction

In RISC V/MIPS
- Register — MOV A, B
- Immediate (for constants) — MOV A, #50h
- Displacement (constant offset added) — Base address ± Offset (BA)

In 80x86
- absolute (no register)
- based indexed with displacement (2 registers) → BA + Di
- based with scaled index and displacement (2 registers ; 1 reg × size of operand in bytes)

(4) TYPES & SIZES OF OPERANDS
(i) 8-bit ASCII character
(ii) 16-bit Unicode character or half word
(iii) 32-bit Integer or word
(iv) 64-bit Double word or long integer
(v) IEEE 754 floating point in 32-bit (single precision)
(vi) IEEE 754 floating point in 64-bit (double precision)
(vii) 80x86 supports 80-bit floating

(5) OPERATIONS
- data transfer
- arithmetic & logical
- control
- floating-point

(6) CONTROL FLOW INSTRUCTIONS (use PC-relative addressing)
- Conditional branches
- Unconditional jumps
- Procedure calls and returns

link register Holds return address
stack (in 80x86) for servicing interrupt
- push PC
- pop return address

10/1/24

(7) ENCODING AN ISA
- FIXED LENGTH All ARM & MIPS instructions are 32 bits long, which simplifies instruction decoding
- VARIABLE LENGTH 80x86 encoding is variable length, ranging from 1 to bytes

Advantageous : less space

ARCHITECTURE covers aspects of computer design → ISA, Organization/Microarchitecture, Hardware

Once ISA is fixed

⇒ ORGANIZATION : includes high-level aspects of a computer's design
- memory system
- memory interconnect
- design of internal processor (CPU)

* For single ISA, multiple microarchitecture possible

⇒ AMD Opteron, Intel Core i7 } implement x86 ; have different instruction set, cache

⇒ HARDWARE

Cache levels: L1 → for processor; L2, L3 → for outside processor

DESIGN OF EFFECTIVE ARCHITECTURE

(1) Take advantage of parallelism
- Bit-level
- Instruction-level
- Data-level
- Thread-level

(2) Principle of Locality

LOCALITY ⇒ predict with reasonable accuracy what instructions & data a program will use in the near future based on its accesses in the recent past

LOCALITY

TEMPORAL

If an item is referenced then it will tend to be again referenced soon within relatively small time durations
⇒ loops in programs

SPATIAL

If a particular storage loca is referenced at a particular then it is likely that nearb memory locations will be ref in the near future
⇒ sequential code execution

(3) Focus on the Common Case
Make common case fast → performance enhanced

AMDAHL'S LAW

Defines the speedup that can be gained by using a particular feature

11/1/24

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

Speedup for some enhancement depends on 2 factors:

1) The fraction of the computation time in the original computer that can be converted to take advantage of the (Fraction$_{enhanced}$), is always less than or equal to 1.

2) The improvement gained by enhanced execution mode, that is, how much faster the task would run if the enhanced mode were used for the entire program ⇒ Speedup$_{enhanced}$ ≥ 1

$$\text{Execution time}_{new} = \text{Execution time}_{old} \times \left\{ (1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right\}$$

Overall Speedup → Ratio of execution times

$$\text{Speedup}_{overall} = \frac{\text{Execution time}_{old}}{\text{Execution time}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

Example 1

Suppose, we want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in Web serving application than the original processor. Assuming the original processor is busy with computation 40% of the time & is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

Speedup$_{enhanced}$ = 10 (→ only computation performance can be enhanced)

Fraction$_{enhanced}$ = $\frac{40}{100}$ = 0.4

$$\text{Speedup}_{overall} = \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.6 + 0.04} = \frac{1}{0.64}$$

$$\boxed{\text{Speedup}_{overall} = 1.5625}$$

Example 2 Floating-point square root (FPSQR) ⇒ 20% of execution ti
2 design alternatives proposed
(i) Enhance FPSQR hardware and speedup the operation
factor of 10
(ii) Try to make all FP instructions in the graphics
processor run faster by a factor of 1.6; FP instructio
are responsible half of the execution time

(i) $\text{Speedup}_{FPSQR} = \frac{1}{(1-0.2)+\frac{0.2}{10}} = \frac{1}{0.82} = 1.22$

(ii) $\text{Speedup}_{FP} = \frac{1}{(1-0.55)+\frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$

Ans Improving the performance of FP operations overall is sli
better because of the higher frequency

11 Example 3 Assume a disk subsystem with the following component
MTTF:
- 10 disks, each rated at 1,000,000-hour MTTF
- 1 ATA controller, 500,000-hour MTTF
- 1 power supply, 200,000-hour MTTF
- 1 fan, 200,000-hour MTTF
- 1 ATA cable, 1,000,000-hour MTTF

Using the simplifying assumptions that the lifetimes are exp
distributed & that failures are independent, compute the
MTTF of the system as a whole

$$\text{Failure rate}_{system} = 10 \times \frac{1}{1000000} + \frac{1}{500000} + \frac{1}{200000} + \frac{1}{200000} + \frac{1}{1000000} = \frac{23}{1000000}$$

Taking a reference of 1 billion unit time

Failure rate system = $\frac{23000}{1000000000 \text{ hours}}$ = 23,000 FIT (Fail in time)

MTTF = $\frac{1}{\text{Failure rate}}$ = 43500 hours ≃ 5 years

16/1/24

Mean Time To Failure (MTTF)
Mean Time Between Failure (MTBF)
Mean Time to Repair (MTTR)

Example 4 Disk subsystems often have redundant power supplies to improve dependability. Using the components & MTTFs from Example 3, calculate the reliability of redundant power supplies. Assume one power supply is sufficient to run the disk subsystem & that one redundant power supply is being added.

MTTF for redundant power supplies is the mean time until one power supply fails divided by the chance that the other will fail before the first one is replaced.
Mean time until one disk fails is $\frac{MTTF_{\text{power supply}}}{2}$

$MTTF_{\text{power supply pair}} = \frac{\frac{MTTF_{\text{power supply}}}{2}}{\frac{MTTR_{\text{power supply}}}{MTTF_{\text{power supply}}}}$

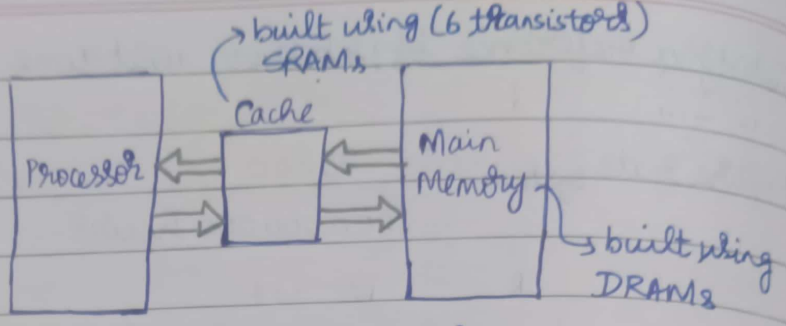
Assuming it takes the operator 24 hours to detect & repair the power supply after failure

$= \frac{200000^2}{2 \times 24} = 833333333.3$

$\frac{833333333}{200000} = 4166$

No refreshing in SRAMs
↳ faster access

MEMORY



As we move away from the Processor :
- access time increases
- storage capacity increases
- cost of storage per byte reduces

Operations
- Read (load) · contents unaltered
- Write (store) · contents altered

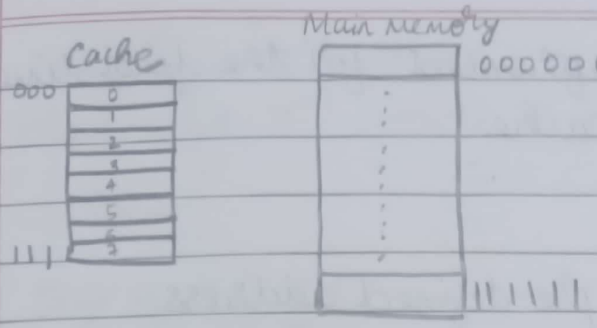
Processor places the address of the location on which the operation is to be performed on the address bus
If the operation is successful ⇒ HIT
failure ⇒ MISS → more time consuming
↳ not found in cache
↓
go to Main Memory
⇓
assumed to have the content pertaining to the specific app

Memory Management Unit (MMU)

Access time
+ Read operation takes lesser amount of time than write operation

Mapping
↳ Direct Mapping (mod operation)

Computer Organisation by Hennesy & Patterson
by Carl



⇒ Accessing 6
6 %. 8 = 6 stored in 110

⇒ Accessing 62
62 %. 8 = 6 6 replaced with 62 in cache

6 bits | 3 bits | 3 bits
Tag ⇓ Index ⇓

↳ have to be matched in order to get a HIT

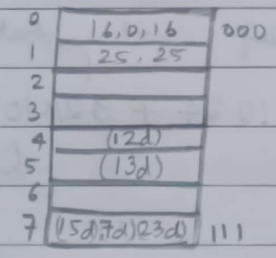
Valid bit → 0 by default

Protection bits
↳ ensures that there is no compromise in the data

Assume a cache with 8 blocks and the following references are made to the cache : 12d, 13d, 16d, 0d, 16d, 25d, 25d, 15d, 07d, 23d.

Identify the number of hits and misses ~~using~~ for a direct mapped cache

Initially empty cache (garbage values)



12 mod 8 = 4 placed at 100 ⇒ miss
13 mod 8 = 5 placed at 101 ⇒ miss
16 mod 8 = 0 placed at 000 ⇒ miss
0 mod 8 = 0 replaces 16 present at 000 ⇒ miss
16 mod 8 = 0 replaces 0's content ⇒ miss

000 | 000
010 | 000
Tag bits

16d content taken away & stored in main memory

∴ 1 hit
9 misses

25 mod 8 = 1 25d content placed at 001 ⇒ miss
25 mod 8 = 1 already present at 001 ⇒ hit
15 mod 8 = 7 15d content placed at 111 ⇒ miss
7 mod 8 = 7 replaces 15d content at 111 ⇒ miss
23 mod 8 = 7 replaces 7d content at 111 ⇒ miss

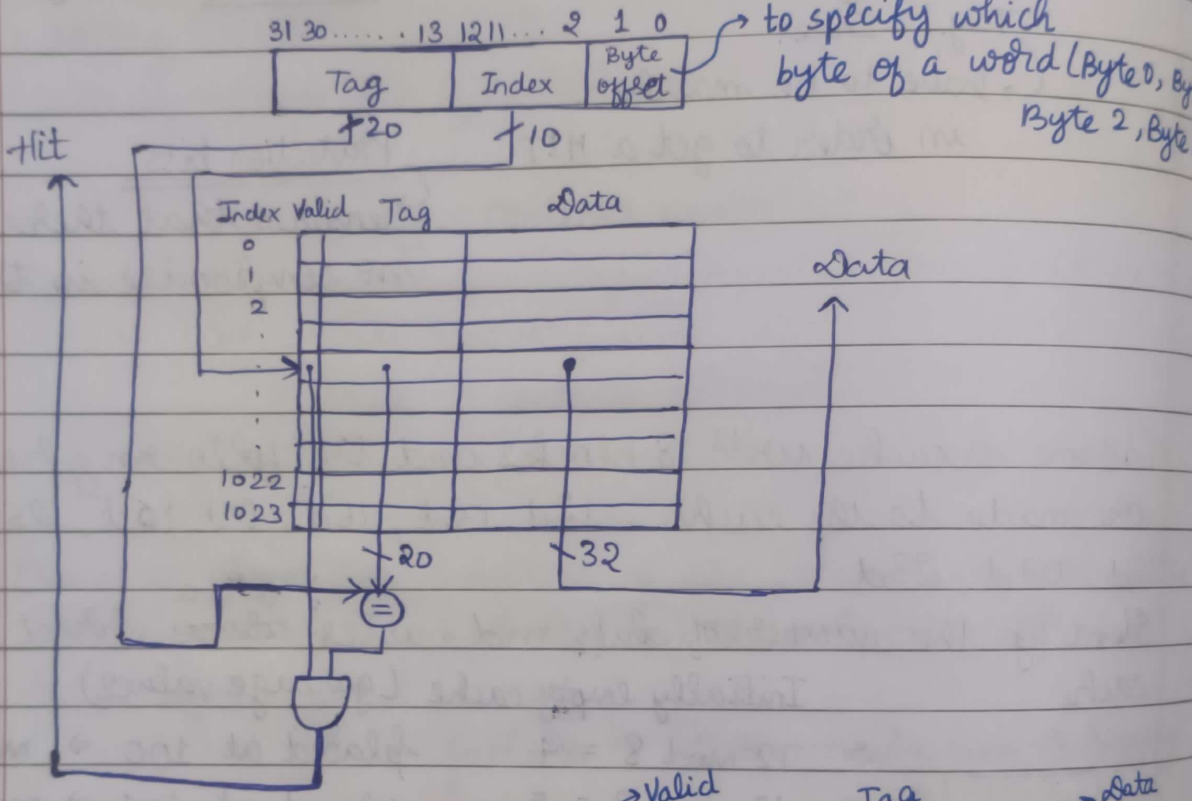
* Even though there were other available free spaces, several misses encountered

COMPULSORY MISS → Data referenced for the first time an initially empty cache

ADVANTAGES
- simple mapping
- always gets mapped to a pre-defined address

DISADVANTAGE
- available cache space is not efficiently utilized

17/1/24



Size of Cache = ~~10 X 1024~~ → 1 X 1024 + 20 X 1024 + 32 X 1024 (Valid bit, Tag, Data)
= 54272 bits
= 6784 bytes
= 6.625 KB

Actual Data Size = 32 X 1024 bits
= 4 KB

Cache size 2^n blocks $\Rightarrow$ n bits used for index

Block size 2^m words $= 2^{m+2}$ bytes $\Rightarrow$ m bits for the word within the block

$2^2 = 4$ bytes per word

Size of tag field $32 - (n + m + 2)$ bits

(2 $\rightarrow$ byte offset)

Total number of bits in direct-mapped cache $= 2^n \times$ (block size + tag size + valid field size)

Q1. How many total bits are required for a direct-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

~~Words = 4 = 2^2 $\Rightarrow$ m = 2~~ Block size = 4 words = 2^2 $\Rightarrow$ m = 2

$\frac{16}{4}$ = 4 kilo words = 4096 words $\Rightarrow 2^{12}$ words

$= 4 \times 1024$

4 words per block

Total number of blocks = $\frac{2^{12}}{2^2} = 2^{10} = 1024$ blocks

$\Rightarrow$ 10 bits used for index

Size of tag field = $32 - (10 + 2 + 2) = 32 - 14$

$= 18$ bits

Total number of bits in direct-mapped cache $= 2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1)$

$= 2^{10} \times 147$

$= 147$ Kibibits $= 147/8$

$= \underline{18.375 \text{ KiB}}$ for a 16 KiB cache

$\frac{18.375}{16} = 1.15$ times

Q2. Consider a cache with 64 blocks and a block size of 16 bytes. To what block address does the byte address 1200 map?

64 blocks $= 2^6$ blocks $\Rightarrow n = 6$

Block size 16 bytes $= 4$ words $= 2^2$ words $\Rightarrow m = 2$

Size of tag field $= 32 - (6+2+2) = 22$ bits

Byte address 1200

Block address $\frac{1200}{4} = 300$

$\frac{1200}{4 \times 4} = 75$ (4 words per block; 4 bytes per word)

Byte address in block $\frac{300}{4} = 75$ ↳ in main memory

64 blocks present in the cache

$\Rightarrow 75 \bmod 64 = 11$ ↳ in cache

∴ The byte address 1200 maps to block address 11.

* (Block address) mod (Number of blocks in cache)

* Block address $= \frac{\text{Byte address}}{\text{Bytes per block}}$

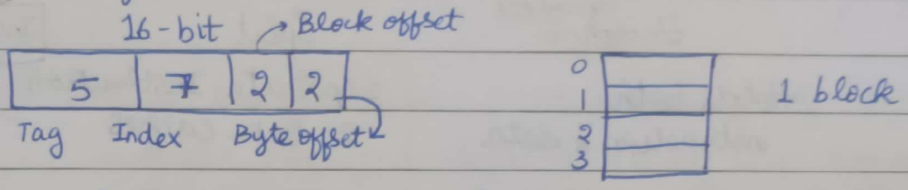
* Block address is the block containing all addresses between $\left[\frac{\text{Byte address}}{\text{Bytes per block}}\right] \times$ Bytes per block

18/1/24

Q3. Consider a cache with 128 blocks each with 16 bytes. Find the number of bits required to represent tag field, index field, block offset and byte field a) For a byte address 8200, find the mapped index value in cache. Assume 16-bit addressing. (Using direct-mapping)

128 blocks = 2^7 blocks ⇒ n = 7 Index field bits

Block size: 16 bytes = 4 words = 2^2 words ⇒ m = 2



Byte offset → 2 bits to specify which Byte within a word

Block offset → 2 bits to specify which of the 4 words in the block is being referred to

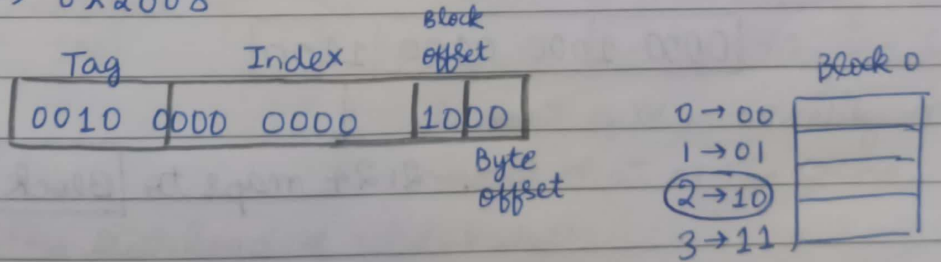
Index → 7 bits ∵ 128 blocks = 2^7 blocks

Tag field → 16 − (7 + 2 + 2) = 5 bits

$\frac{8200 \text{ byte address}}{16 \text{ bytes}} = 512.5$ $\qquad 0.5_{10} \rightarrow .10_2$

512 mod 128 = 0 ⇒ Block 0

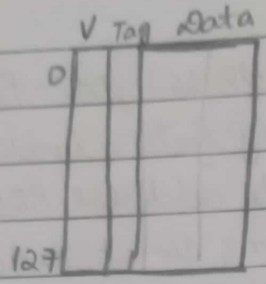
8200 → 0x2008



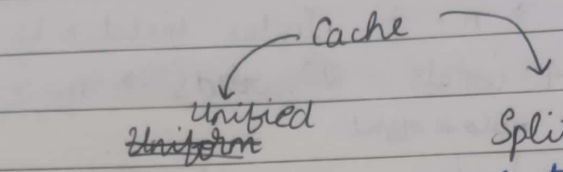
Block offset value = 10_2 ⇒ 2

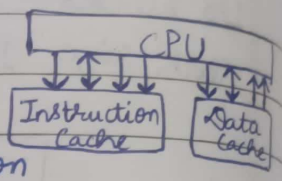
∴ 8200 byte address maps to **Block 0, Word 2**

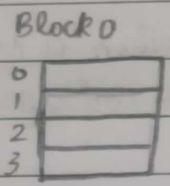
128 Validity bits (1 Valid bit per block)

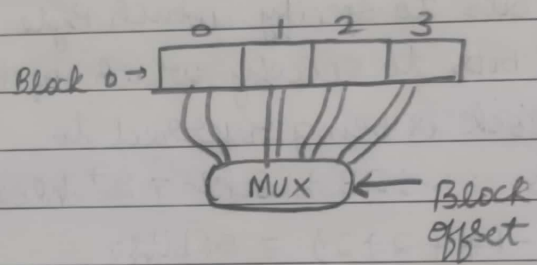


Cache size = $2^7(1 + 5 + 4 \times 32)$ ← Data size, # word per block
$= 2^7(134)$
$= 17152$ bits $= 2144$ bytes
$= 2.09375$ KiB









Intrinsity FASTMATH Processor

Q3. b) Byte address 2124 maps to ?

$\frac{2124}{16} = 132.75$ $\quad 0.75_{10} \rightarrow 0.11_2$

132 mod 128 = 4 Block 4

0000	1000	0100	11	00

∴ 2124 maps to **Block 4, Word 3**

22/1/24

Handling Writes

Store instruction
↳ Data written only into data cache → Data in main memory unchanged

INCONSISTENCY : Memory & cache have different data stored

Write Through Policy Change data in at one memory location, data changed ~~at~~ across ~~all~~ the memory hierarchy ; every change is accounted

WRITE BUFFER • Placed between processor and main memory
• Contains data to be written into cache
• Operations on cache need not be stalled
• Main memory data updated using data present in write buffer

Write Back Policy When a write occurs, the new value is written only to the block in the cache.

Dirty/Modified bit (D) (M) → 0 : Content of main memory matches with that of the cache
→ 1 : Content of Main memory mismatched with that of the cache

• no extra overhead of write buffer
• an extra bit in the cache is introduced

↳ extra time due to miss

$$\text{Write-stall Clock cycles} = \left(\frac{\text{Memory access}}{\text{Program}}\right) \times \text{Miss rate} \times \text{Miss penalty}$$

Separate instruction & data caches → Harvard architecture

Example 1 Assume the miss rate of an instruction cache is 2%. and the miss rate of data cache is 4%. If a processor has a CPI of 2, without any memory stalls & the miss penalty is 100 cycles for all misses, and the frequency of all loads & stores is 36%.

1) determine how much faster a processor would run with a perfect cache that never missed
2) what happens if we speed up the machine by reducing CPI to 1 without changing the clock
3) what happens if we speed up the machine by doubling (a) ~~reducing its~~ the clock rate but absolute time for miss penalty remains the same.

$$\text{CPU time} = \left(\begin{array}{c}\text{CPU execution} \\ \text{clock cycles}\end{array} + \begin{array}{c}\text{Memory-stall} \\ \text{clock cycles}\end{array}\right) \times \begin{array}{c}\text{Clock cycle} \\ \text{time}\end{array}$$

Memory-stall clock cycles = Read-stall cycles + Write-stall cycles

MOV R_0, R_1 → 2 cycles Read contents of R_1 & write into R_0

store MOV [10], R_1 → 2 cycles

load MOV R_1, [10] → 2 cycles

1) Assuming instruction count = I

2 cycles per instruction

Instruction miss cycles = I × 2% × 100 = 2I (2% = miss rate, 100 = miss penalty)

Data miss cycles = (I × 36%) × 4% × 100 = 1.44I (only a percentage of instructions access memory)

Total number of memory-stall cycles = (2 + 1.44)I = 3.44I

$\text{CPU time}_{\text{perfect}} = (2 \times I + 0)$ clock cycles (0 → no stall) $= 2I$ clock cycles

$\text{CPU time}_{\text{stall}} = (2 \times I + 3.44 \times I)$ clock cycles $= 5.44I$ clock cycles

$$\frac{\text{CPU time}_{\text{stall}}}{\text{CPU time}_{\text{perfect}}} = \frac{I \times \text{CPI}_{\text{stall}} \times \text{Clock cycle}}{I \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}}$$

$$= \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2}$$

$$= 2.72$$

∴ Perfect cache's performance is better by 2.72.

2) Processor made faster (CPI changed to 1)

CPU execution time = $(1 \times I + 3.44 \times I)$ clock cycles

$= 4.44I$ clock cycles

System with perfect cache would be $\frac{4.44}{I} = 4.44$ times as fast

3) Amount of execution time spent on memory stalls would have risen from $\frac{3.44}{5.44} = 63\%$ to $\frac{3.44}{4.44} = 77\%$.

Example 2 Assume we have a computer where CPI is 1 when all memory accesses hit in the cache. The only data accesses are loads and stores and these total 50% of the instructions. If the miss penalty is 25 clock cycles, and miss rate is 2%, how much faster would the computer be if all instructions were cache hits?

CPI = 1 Miss rate = 2% Miss penalty = 25 clock cycles

Total number of memory-stall cycles $= (I \times (1 + 50\%)) \times 2\% \times 25 = 0.75I$

(1 → for instruction; 50% → data)

CPU time$_{stall}$ = (1 × I + 0.75 × I) clock cycles = 1.75I × clock cycles

CPU time$_{perfect}$ (all hits) = (1 × I + 0) = 1I × clock cycles

* In memory stall cycles (1 + 0.5) × IC
  (1 → instruction, 0.5 → data)

$$\frac{\text{CPU time}_{stall}}{\text{CPU time}_{perfect}} = \frac{1.75 \times I \times \text{Clock cycle}}{1 \times I \times \text{Clock cycle}} = 1.75$$

∴ Computer with no cache misses is 1.75 times faster

Problem 1

23/1/24

FULLY ASSOCIATIVE (FA) MAPPING

- Data can be placed anywhere in the cache
- The number of hits goes up, reducing the access time
- Searching for the desired data is harder due to random placement of the data ⇒ overhead
  - * search time increases
  - * hardware complexity increases
    - ↳ for example, more comparators are included to perform parallel checks
- makes use of cache memory efficiently
- hard to implement

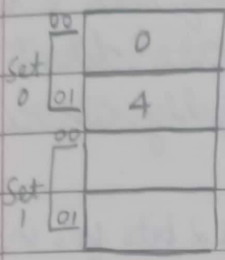
0, 4, 8, 12, 0, 4, 8, 12
M M M M H H H H

0
4
8
12

* As associativity increases, time taken for search operation increases; hardware required also increases

SET ASSOCIATIVITY MAPPING (2-way Set Associative)

↗ 2 blocks per set



- Divide the blocks into sets
- Perform Data % #sets to obtain Set index
- Place in one of the blocks of the set
- reduced hardware complexity

* # Blocks in a set = power of 2

0, 4, 8, 4

M M M H H

0%2 = set 0 | 4%2 = set 0

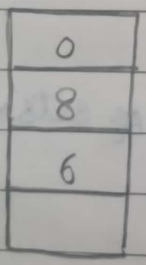
Place in 00 | Place in 01

* FA is a special case of SA, where all of the blocks are considered to be a part of the same set

Cache Mapping

Problem 1 Assume there are 3 small caches, each consisting of 4 one-word blocks. One cache is fully associative, a second is 2-way set associative, and the third is direct-mapped. Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, 8.

FULLY ASSOCIATIVE



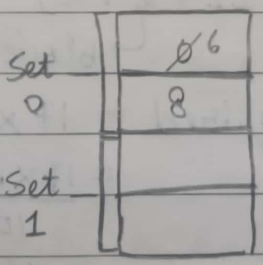
0, 8, 0, 6, 8

M M H M H

Misses = 3

Hits = 2

2-WAY SET ASSOCIATIVE



0, 8, 0, 6, 8

M M H M H

0 mod 2 = 0 set

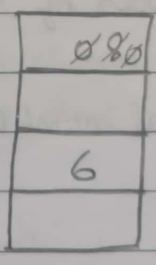
8 mod 2 = 0 set

6 mod 2 = 0 set

Misses = 3

Hits = 2

DIRECT MAPPING



0, 8, 0, 6, 8

M M M M M

0 mod 4 = 0

8 mod 4 = 0

6 mod 4 = 2

Misses = 5

Hits = 0

Problem 2 Assuming a cache of 4096 blocks, a 4-word block size and a 32-bit address, find the total number of sets & the total number of tag bits for caches that are direct mapped, 2-way & 4-way set associative & fully associative.

Number of blocks = 4096 = $4 \times 2^{10} = 2^{12}$ ⇒ m = 12 bits for index

Number of words per block = 4 = 2^2

DIRECT MAPPING

Number of tag bits = 32 − (12 + 2 + 2) = 16 bits
(2 bits for block offset; 2 bits for byte offset)

Total number of tag bits required = 16 × Number of blocks

$= \frac{16 \times 4096}{1024}$

= 64 Kbits

2-way SA

Number of sets = $\frac{4096}{2}$ = 2048 sets ⇒ 11 bits

Number of tag bits = 32 − (11 + 2 + 2) = 17 bits
(2 bits for byte offset; 2 bits for block offset)

Total number of tag bits required = 17 × Number of sets × 2 (for blocks within a set)

$= \frac{17 \times 2048 \times 2}{1024}$

= 68 Kbits

4-way SA

Number of sets = $\frac{4096}{4}$ = 1024 sets ⇒ 10 bits

Number of tag bits = 32 − (10 + 2 + 2) = 18 bits

Total number of tag bits required = $\frac{18 \times 1024 \times 4}{1024}$ = 72 Kbits

351 Cache Simulator

classmate Date Page 25

FA

Number of sets = $\frac{4096}{4096} = 1 \Rightarrow 0$ bits

Number of tag bits = 32-(2+2) = 28 bits

Total number of tag bits = $\frac{28 \times 4096 \times 1}{1024} = 112$ Kbits

LEAST RECENTLY USED (LRU) Computer Organisation by Carl Hamacher

0, 8, 0, 6, 8

Ref.	loaded pages (2)		Referenced String
0	0	-	0,
8	0	8	0, 8
0	0	~~8~~	0, 8, 0
6	~~0~~	6	0, ~~8~~, 0, 6
8	8	6	0, 8, ~~0~~, 6, 8

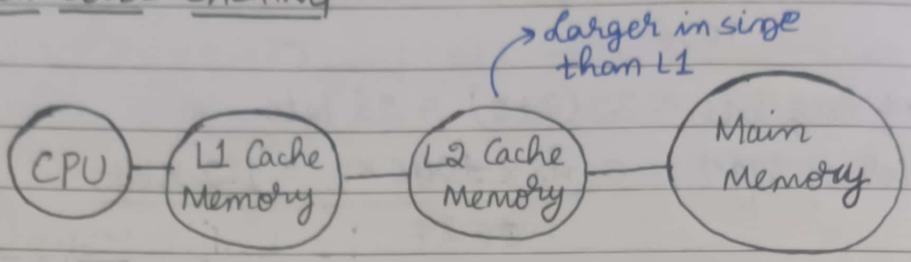
Example Consider the following memory references and find the number of page faults. Assume, the number of loaded pages is 4.

1, 1, 6, 9, 6, 8, 4, 1, 6, 9, 15, 17, 9, 1, 8

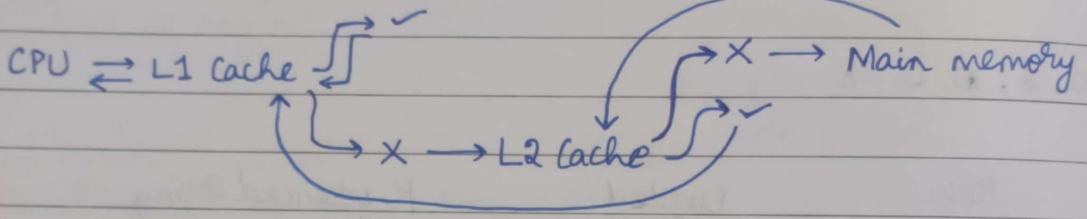
Page faults		loaded pages (4)				
(1)	1	1				1,
	1	1				1, 1
(2)	6	1	6			1, 1, 6
(3)	9	1	6	9		1, 1, 6, 9
~~(4)~~	6	1	6	9		1, 1, 6, 9, 6
(4)	8	~~1~~	6	9	8	1, 1, 6, 9, 6, 8
(5)	4	4	~~6~~	9	8	1, 1, 6, 9, 6, 8, 4
(6)	1	4	1	~~9~~	8	1, 1, 6, 9, 6, 8, 4, 1
(7)	6	4	1	6	8	1, 1, 6, 9, 6, 8, 4, 1, 6
(8)	9	~~4~~	1	6	9	1, 1, 6, 9, 6, 8, 4, 1, 6, 9
(9)	15	15	~~1~~	6	9	1, 1, 6, 9, 6, 8, ~~4~~, 1, 6, 9, 15
(10)	17	15	17	6	9	1, 1, 6, 9, 6, 8, 4, 1, 6, 9, 15, 17
	9	15	17	~~6~~	9	1, 1, 6, 9, 6, 8, 4, 1, 6, 9, 15, 17, 9
(11)	1	~~15~~	17	1	9	1, 1, 6, 9, 6, 8, 4, 1, 6, 9, 15, 17, 9, 1
(12)	8	8	17	1	9	

24/1/24

MULTI-LEVEL CACHING



MEMORY CONTROLLER UNIT manages the maintenance and accessing of the different levels of memory



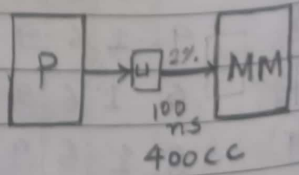
Problem Suppose we have a processor with a base CPI of 1, assuming all references hit in the primary cache, and a clock rate of 4GHz. Assume a main memory access time of 100 ns, including all the miss handling. Suppose, the miss rate per instruction at the primary cache is 2%. How much faster will the processor be if we add a secondary cache that has a 5ns access time for either a hit or miss & is large enough to reduce the miss rate to main memory to 0.5%?

Clock cycle time = $\frac{1}{\text{Clock rate}} = \frac{1}{4 \times 10^9} = 0.25\ \text{ns/clock cycles}$

1 level of caching

Miss penalty to main memory = $\frac{100\ \text{ns}}{0.25\ \frac{\text{ns}}{\text{clock cycles}}} = 400$ clock cycles (access time to MM)

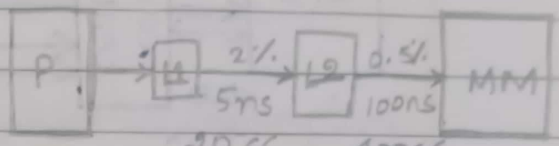
Total CPI = 1 + 2% × 400 = 9 (Base CPI; 2% × 400 = Memory-stall cycles per instruction)



2 levels of caching

Miss penalty for access to 2nd level cache

$= \frac{5\text{ns}}{0.25\ \frac{\text{ns}}{\text{clock cycle}}} = 20$ clock cycles



Total CPI = 1 + 2% × 20 + 0.5% × 400 = 1 + 0.4 + 2 = 3.4

(1° stalls per instr. for 2% × 20; 2° stalls per instr. for 0.5% × 400)

Processor with secondary cache is faster by $\frac{9}{3.4} = \underline{2.64}$

(OR)

$(2\% - 0.5\%)\,20 = 0.3$ — Stall cycles of those references that hit in the secondary cache

\+ $0.5\%\,(20 + 400) = 2.1$ — References that go to main memory (inclusive of 2° cache + main memory access time)

\+ 1 — Base CPI

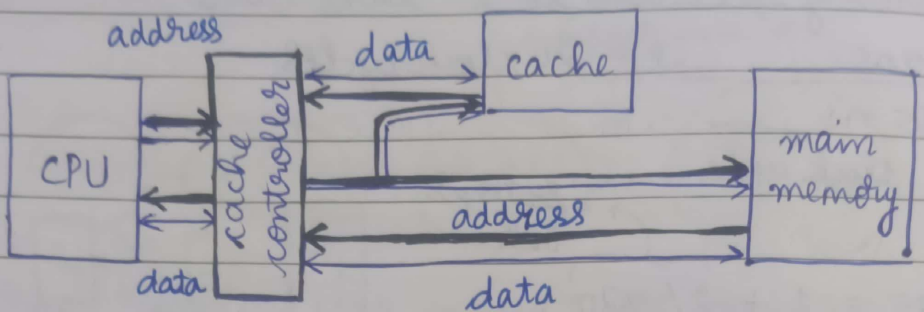
$= \underline{3.4}$

Advantages of Multilevel Cache Organisation
- Reduced access time
- Improved system performance
- Lower cost
- Energy efficiency

Disadvantages of Multilevel Cache Organisation
- Complexity
- Higher latency for cache misses
- Higher cost
- Cache coherence issues

29/1/24

CACHE HIT



CACHE MISS (3 Cs)

→ Compulsory : first access to a block
(cold start or process migration, first reference)

Conflict
(collision)
- multiple memory locations mapped to the same cache location

SOLUTIONS
⇒ increase cache size
⇒ increase associativity

→ Capacity
- cache cannot contain all blocks access by the program

SOLUTION ⇒ increase cache size

Average Memory Access Time = Hit Time + Miss Rate × Miss Penalty

GOALS	BASIC APPROACHES
* Reducing Miss Rate	Larger block size, larger cache size & higher associativity
* Reducing Miss Penalty	Multilevel caches & higher read priority over writes
* Reducing Hit Time	Avoid address translation when indexing the cache → taken care of by the cache controller

Coherence problem ⇒ mismatch in data present across the hierarchies

Basic Cache Optimizations

(i) LARGER BLOCK SIZE
↓ compulsory misses ; ↑ capacity , ↑ conflict misses, ↑ miss penalty

(ii) LARGER TOTAL CACHE CAPACITY TO REDUCE MISS RATE
↑ hit time , ↑ power consumption

(iii) HIGHER ASSOCIATIVITY
↓ conflict misses , ↑ hit time, ↑ power consumption

(iv) HIGHER NUMBER OF CACHE LEVELS
↓ overall memory access time

(v) GIVING PRIORITY TO READ MISSES OVER WRITES
↓ miss penalty

(vi) AVOIDING ADDRESS TRANSLATION IN CACHE INDEXING
↓ hit time

WAR Write After Read
RAW Read After Write
WAW Write After Write
RAR Read After Read

MISS RATE IN MULTILEVEL CACHES

Local Miss Rate

⇒ Number of accesses in a cache / Total number of memory accesses to this cache

Global Miss Rate

⇒ Number of misses in cache / Total number of memory accesses generated by processor

⇒ for L1 : MissrateL1

⇒ for L2 : Miss rate L1 x Miss rate L2

* Appendix B of T1

29/

Q1. Using the data in Figure B.8 (Appendix B) determine whether a 32KB 1-way set associative L1 cache has a faster memory access time than a 32KB 2-way set associative L1 cache. Assume the miss penalty to L2 is 15 times the access time for the faster L1 cache. Ignore misses beyond L2. Which has the faster average memory access time.

	2-way	4-way
Miss rate	0.038	0.037

Assume the access time of 4-way cache is 1.4 times longer

Average Memory Access Time = Hit rate + Miss rate x Miss penalty

Not mentioned in the question ⇒ Assume Hit rate is 1

2-way: $1 + 0.038 \times 15 = 1.57$ time units

4-way: $1.4 + 0.037 \times 15 = 1.807$ time units

$1.4 \approx 1$

$\frac{1.807}{1.57} = 1.15$

Q2. A system has 4GB of main memory, cache of 4KB where the cache has 64 Bytes per block. Write address fields for the following mapping techniques:

a) Direct Mapping b) Fully Associative c) 4-way Set Associative d) 2-way Set Associative

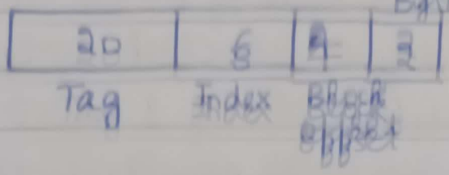
(Assume 32-bit addressing)

Block size = 64 Bytes = 2^6 Bytes ⇒ Index bits = 6

Number of blocks = $\frac{4 \times 1024}{64}$ = 64 blocks = 2^6 Blocks ⇒ 6 index bits

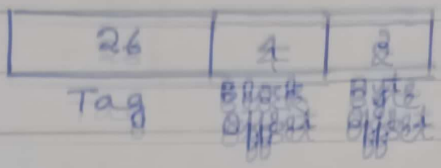
Number of words per block = $\frac{64}{4}$ = 16 words = 2^4 words ⇒ 4 b

a) Number of tag bits = 32 - (8 + 4 + 2) = 20 bits



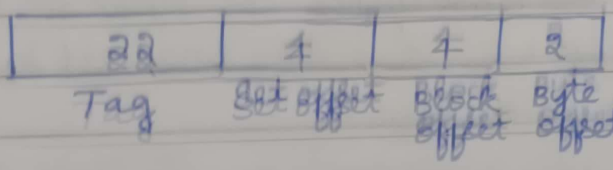
b) Number of sets = $\frac{64}{64} = 1 \Rightarrow 2^0$ 0 Index bits

Number of tag bits = 32 - (4 + 2) = 26 bits



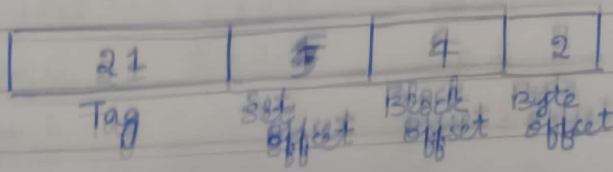
c) Number of sets = $\frac{64}{4} = 16$ sets $\Rightarrow 2^4$ 4 bits

Number of tag bits = 32 - (4 + 4 + 2) = 22 bits



d) Number of sets = $\frac{64}{2} = 32$ sets $\Rightarrow 2^5$ 5 bits

Number of tag bits = 32 - (4 + 5 + 2) = 21 bits



2- 30/1/24

Q1. Find the average memory access time for a processor wi
2ns clock cycle time, a miss rate of 0.04 misses per
instruction, a missed penalty of 25 clock cycles, and a
access time (including hit detection) of 1 clock cycle. A
assume that the read & write miss penalties are the sa
ignore other write stalls

Average memory access time = $1 + 0.04 \times 25 = 2$ clock cycles

1 clock cycle = 2ns

∴ 2 clock cycles = 4ns

Q2. Assume the memory system takes 80 clock cycles of overh
and then delivers 16 bytes every 2 clock cycles. Thus,
supply 16 bytes in 82 clock cycles, 32 bytes in 84 clock c
& so on. Which block size has the smallest average me
access time for each cache size?

	CACHE SIZE			
BLOCK SIZE	4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.7%
64	7%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

Block Size	16	32	64	128	256
Miss penalty	$\frac{16}{16} \times 2 + 80$	$\frac{32}{16} \times 2 + 80$	$\frac{64}{16} \times 2 + 80$	$\frac{128}{16} \times 2 + 80$	$\frac{256}{16} \times 2$
	$= 82$	$= 84$	$= 88$	$= 96$	$= 112$

1-way ⇒ direct mapping
(1 block → 1 set)

Average memory access time = Hit time + Miss rate × Miss penalty
= 1 + (8.57% × 82) = 8.0274 clock cycles
for a ~~256~~ block size of 16 bytes & cache size of 4KB

For 256 B block size & 256 KB cache size
Average memory access time = 1 + (0.49% × 112)
= 1.5488 clock cycles

$\frac{256}{16} \times 2 + 80$ ⇓ overhead
∵ 16 bytes every 2 clock cycles

For 128 B Block size & 16KB cache size
AMAT = ~~1 + {2.77% × (128/16 × 2 + 80)}~~

= $1 + \left\{ 2.77\% \times \left(\frac{128}{16} \times 2 + 80 \right) \right\}$ (2 clock cycles every 16 bytes) = 1 + (2.77% × 96)

= 1 + 2.6592 = 3.6592 clock cycles

For 32B Block size & 64 KB cache size
AMAT = 1 + {1.35% × 84} = 2.134 clock cycles

For 64B Block size & 4KB cache size
AMAT = 1 + (7% × 88) = 7.16 clock cycles

Q3. Assume that higher associativity would increase clock cycle time as:

Clock-cycle time
2-way = 1.36 × clock cycle time 1-way
4-way = 1.44 × clock cycle time 1-way
8-way = 1.52 × clock cycle time 1-way

The clock cycle time for direct mapped cache on hit is 1 CC & for 2-way it increases to 1.36 × direct mapped ...

Assume that hit time = 1 clock cycle
Miss penalty for direct mapped case = 25 clock cycles to a L2 cache that never misses
Miss ~~rate~~ penalty need not be rounded to an integral number. Which statement is true?

(only hit time is getting affected, not miss-penalty)

$AMAT_{8\text{-way}} < AMAT_{4\text{-way}}$
$AMAT_{4\text{-way}} < AMAT_{2\text{-way}}$
$AMAT_{2\text{-way}} < AMAT_{1\text{-way}}$

4KB cache size

1-way $1 + (0.098 \times 25) = 3.45$

2-way $1 + (0.076 \times 25) \times 1.36 = 3.584$

4-way $1 + (0.071 \times 25) \times 1.44 = 3.556$

8-way $1 + (0.071 \times 25) \times 1.52 =$

4KB cache size

1-way $1 + (0.098 \times 25) = 3.45$

2-way $1.36 + (0.076 \times 25) = 3.26$

4-way $1.44 + (0.071 \times 25) = 3.215$

8-way $1.52 + (0.071 \times 25) = 3.295$

For 4KB cache size, $AMAT_{8\text{-way}} < AMAT_{4\text{-way}}$ ✗

$AMAT_{4\text{-way}} < AMAT_{2\text{-way}}$ ✓

$AMAT_{2\text{-way}} < AMAT_{1\text{-way}}$ ✓

Q4. Suppose that in 1000 memory references, there are 40 misses in L1 cache & 20 misses in L2 cache. What are the various miss rate assume miss penalty from L2 cache to memory is 200 clock cycles, hit time of L2 cache is 10 clock cycles, hit time of L1 is 1 clock cycle & there are 1.5 memory references per instruction. What is average memory access time & average stall cycles p instruction? Ignore impact of writes.

	L1	L2
Miss rate	$\frac{40}{1000} = 4\%$	$\frac{20}{1000} = 2\%$ (global)
		$\frac{20}{40} = 50\%$ (local)

Average memory access time = hit time$_{L_1}$ + Miss rate$_{L_1}$ × (Hit rate$_{L_2}$ + Miss rate × Miss penalty$_{L_2}$)

$= 1 + \frac{4}{100}\left(10 + \frac{50}{100} \times 200\right) = 5.4$ clock cycles

$\frac{1000 \text{ memory ref}}{1.5 \text{ mem ref per instr.}} \simeq 667$ instructions

$40 \times 1.5 = 60$ L1 misses $\quad 20 \times 1.5 = 30$ L2 misses per 1000 instructions.

Average memory stalls per instruction = Misses per instr.$_{L1}$ × Hit time$_{L2}$ + Misses per instr.$_{L2}$ × Miss penalty$_{L2}$

$$= \frac{60}{1000} \times 10 + \frac{30}{1000} \times 200$$

$= 6.6$ clock cycles

→ AMAT

$(5.4 - 1) \times 1.5 = 4.4 \times 1.5 = 6.6$ clock cycles

(L1 hit time; avg. number of mem ref per instr.)

31/1/24

Memory-stall clock cycles = (Read-stall cycles + Write-stall cycles)

$$\text{Read-stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

$$\text{Write-stall cycles} = \left(\frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty}\right) + \text{Write buffer stalls}$$

$$\text{Memory-stall clock cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

$$\text{Memory stall clock cycles} = 1.5 \times \frac{40}{1000} \times 10 + 1.5 \times \frac{60}{1000} \times 200 = 6.6 \text{ clock cycles}$$

Associativity → increases hit time
→ decreases miss rate

Q5. Given the data below, what is the impact of L2 cache associativity on its miss penalty?

- hit time_{L2} for direct mapped = 10 clock cycles
- 2-way set associativity increases hit time by 0.1 clock cycle; 10.1 clock cycles
- $\text{Local miss rate}_{L2}$ for direct mapped = 25%.
- $\text{Local miss rate}_{L2}$ for 2-way set associative = 20%.
- Miss penalty_{L2} = 200 clock cycles

For a direct mapped L2 cache

L1 cache miss penalty $\quad \text{Miss penalty}_{1\text{-way L2}} = 10 + 25\% \times 200$

$= 60$ clock cycles

$\text{Miss penalty}_{2\text{-way L2}} = 10.1 + 20\% \times 200$

$= 50.1$ clock cycles

L2 hit time has to be an integral number of clock cycles

↳ 10.1 can be approximated to $10 \Rightarrow 10 + 20\% \times 200 = 50$ CC

$11 \Rightarrow 11 + 20\% \times 200 = 51$ CC

either is an improvement on direct-mapped cache

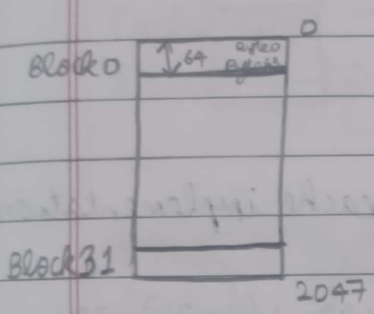
1/2/24

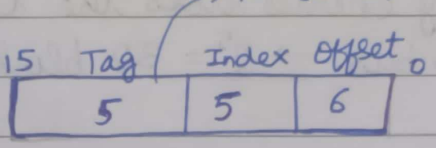
Problems on Cache (Question Bank)

2) 16-bit memory addresses
2KB cache ; 64 bytes per cache block
Size of each memory word = 1 byte

a) Number of blocks = $\frac{2 \times 1024}{64} = 32 = 2^5$ ⇒ 5 bits for index

16 − (5+6) = 16 − 11 = 5

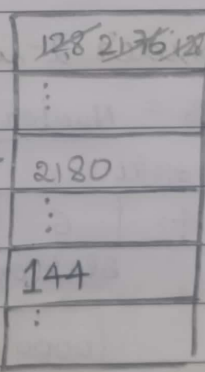


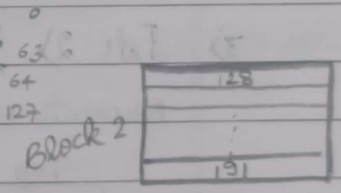


b) Sequential access of 128, 144, 2176, 2180, 128, 2176
Initially empty cache

Block

128 mod 32 = 0 M
144 mod 32 = 16 M
2176 mod 32 = 0 M
2180 mod 32 = 4 M
128 mod 32 = 0 M
2176 mod 32 = 0 M





★ 128 ⇒ 0x80 — 0000 0000 1000 0000 — (M)
Block 2

★ 144 ⇒ 0x90 — 0000 0000 1001 0000 — (H)
Block 2
Block 2 already present (128-191)

★ 2176 ⇒ 0x880 — 0000 1000 1000 0000 — (M)
Block 2
Same offset as 128 ⇒ Block 2 removed & 2176 stored (2176-2239)

★ 2180 ⇒ 0x884 — 0000 1000 1000 0100 — (H)
Block 2
Block 2 already present ⇒ offset + empty (2176-2239)

★ 128 ⇒ 0x80 Block 2 (M) ∵ currently has 2176-2239 memory addresses
★ 2176 ⇒ 0x880 Block 2 (M) ∵ currently has 128-191 memory addresses

4)

Tag	Index	offset
31 - 10	9-5	4-0
22	5	5

Direct-mapped cache
32-bit address

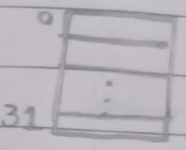
a) Cache block size (in words)

Offset = 5 bits → 2^5 = 32 bytes
= 8 words

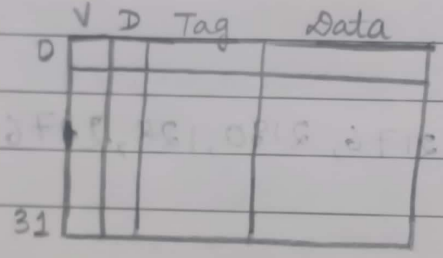
~~Index = 5 bits~~

b) How many entries does the cache have

Index = 5 bits → 2^5 = 32 blocks



c) What is the r total bits required for such a cache implementa



Validity bits (one per block)
words

32 + 32 + 22×32 + (8×32)×32

Dirty bit
Tag bits
per block

= 8960 bits = 8.75 Kbits

3) For 2), consider 2-way set associativity

Number of sets = Number of blocks / 2 = 32/2 = 16 ⇒ 2^4

Tag	Set offset	Block offset
6	4	6

2

Set 2

128
191
2176
2239

(M) 128 ⇒ 0x80
(H) 144
(M) 2176
(H) 2180
(H) 128
(H) 2176

5/2/24

~~For a direct mapped cache, 32-bit addressing~~

5) 64-bit processor has a 8 MB, 4-way set associative cache with 32-byte blocks. How is the address arranged in terms of set, block and offset bits.

* Processor capacity is determined by data bus width and not address bus width

Cache size → 8 MB

1 block → 32 bytes → 8 words

↳ 2^5 ⇒ 5 bits offset

$$\frac{8\text{ MB}}{32\text{ B}} = \frac{8\times 2^{20}}{2^5} = 2^3\times 2^{15} = 2^3\times 2^5\times 2^{10} = 256\times 2^{10}$$

$$= 256\text{ K Blocks}$$

$$= \frac{256\times 2^{10}}{4}$$

$$= 64\text{ K sets}$$

$$= 2^6\times 2^{10} \Rightarrow \text{16 bits for Index}$$

Tag	Index	offset	
11	16	3	2

Tag = 32 − (16 + 3 + 2) = 11 bits

7) Consider a direct mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find the number of tag bits in tag field.

Cache size = 16 KB

Number of blocks = $\frac{16\text{ KB}}{256\text{ B}} = \frac{2^4\times 2^{10}}{2^8} = 2^6$ blocks ⇒ 6 bits for Index

Block size = 256 bytes = 64 words

2^8 ⇒ offset = 8 bits

offset	
6	2

Main memory size = 128 KB = $2^7 \cdot 2^{10}$ ⇒ 17-bit addressing

Tag bits = 17 − (6 + 8) = 3 bits

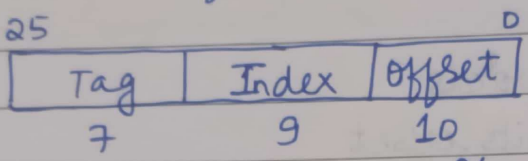
16	3	6	8	0
Tag		Index	offset	

8) Consider a direct mapped cache of size 512 KB with block size 1KB. There are 7 bits in the tag. Find size of main memory.

Block size = 1KB = 1024 bytes = 2^{10} ⇒ 10 bits for offset

Number of blocks = $\frac{512KB}{1KB} = 2^9$ ⇒ 9 bits for index

Total number of bits = 7 + 9 + 10 = 26 bits

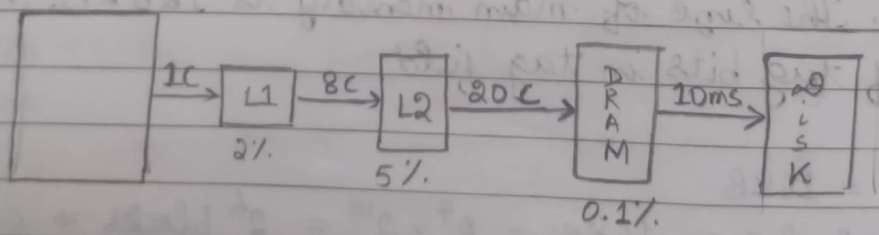


Size of main memory = $2^{26} = 2^6 \cdot 2^{20} =$ 64 MB

9) Given the following data and assuming a clock rate of 1000 MHz

Memory	Hit Time	Miss Rate
L1	1 cycle	2%
L2	8 cycles	5%
DRAM	20 cycles	0.1%
Disk	10 ms	

Calculate the avg memory access time



Clock rate = 1000 MHz

Clock cycles = $\frac{1}{1000 \times 10^6}$ = 1 ns ⇒ 1 clock cycle

1 ns → 1 CC

10 ms → ? ⇒ $\frac{10ms}{1ns} = \frac{10 \times 10^{-3}}{10^{-9}} = 10^7 =$ 10M clock cycles

Average memory access time = $1 + 2\% \cdot (8 + 5\% \cdot (20 + 0.1\% \cdot (10 \times 10^6)))$

= 11.18 cycles.