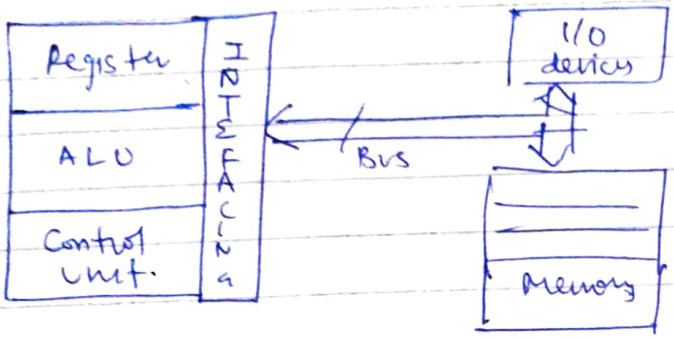
NOTES

COMPUTER SYSTEM ARCHITECTURE

Computer System Architecture

↳ deals with functional behaviour of computer system.
↳ design implementation for the various part of computer.

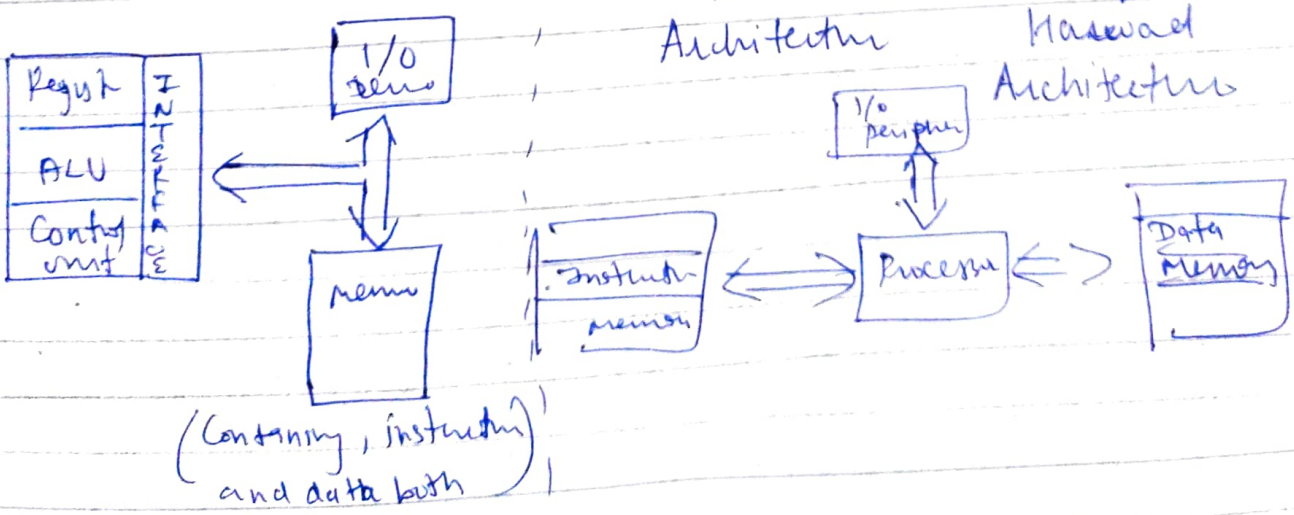


Architecture

- Instruction Set Architecture (ISA)
  - MOV R1, 02H
    MOV R2, 03H
    ADD A, R2
    STORE X, R1.
- Hardware System Architecture (HSA)
  - R1, R2 → Adder

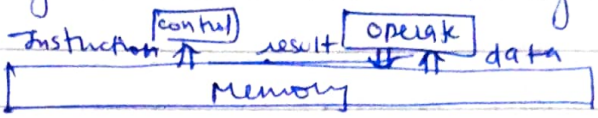
Computer Architecture

- Von neumann Architecture (A.K.A Princeton Architecture)
- Non-von Neumann Architecture
  - Harvard Architecture
  - Modified Harvard Architecture



Computer Architecture

↳ SISD → [Single instruction stream, Single data stream]



↳ SIMD

↳ MISD (Not used yet.)

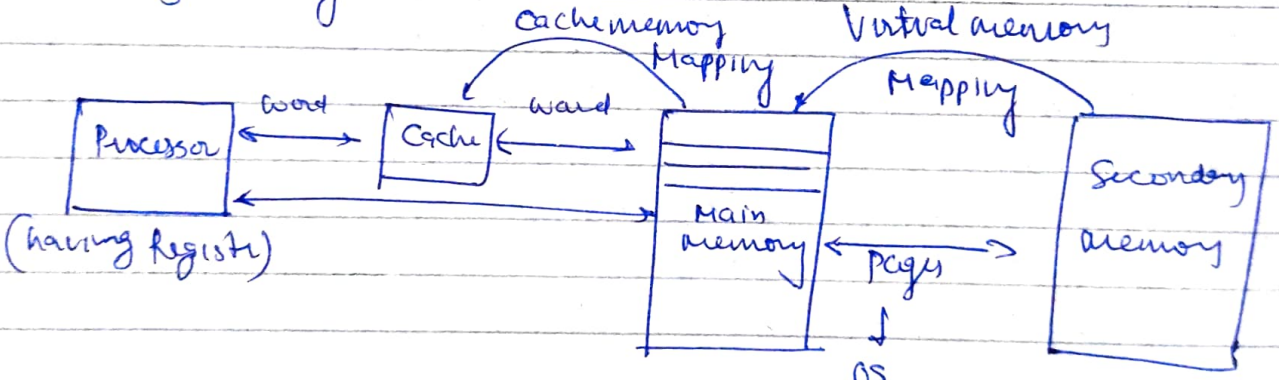
↳ MIMD

Memory

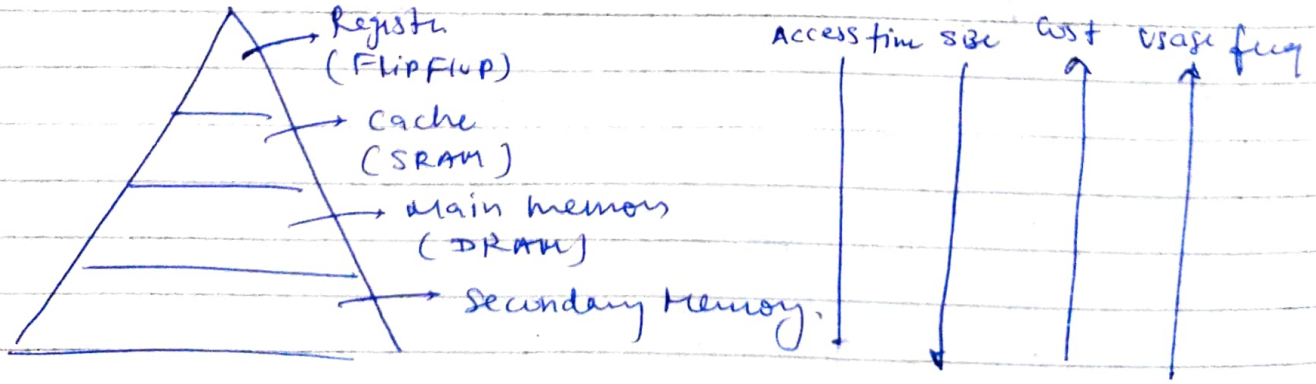
Primary memory → Basically → DRAM. (little slow, cheap, volatile)

Cache → SRAM. (costly, fastest, volatile)

Secondary Memory → Harddisk.



Memory Hierarchy



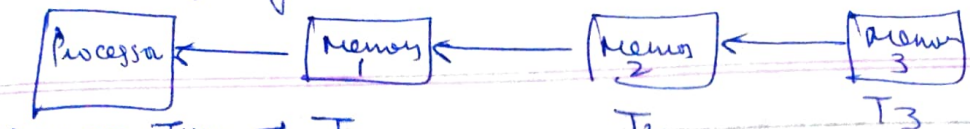
↳ We can't use large amount of fast memory

↳ ① expensive → cost, watt, space.

↳ ② even fast memory chips make slow big memory system.

→ Low level are closer to CPU.

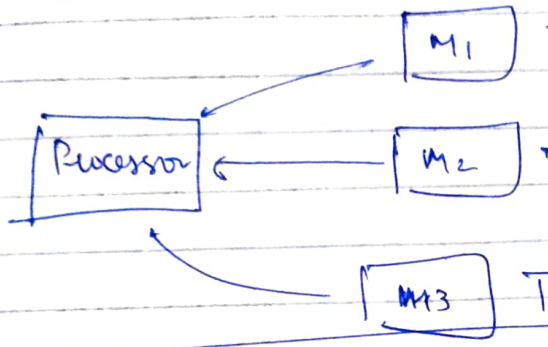
Memory Interfacing



Access Time → T_1 T_2 T_3

Hit ratio → H_1 H_2 H_3

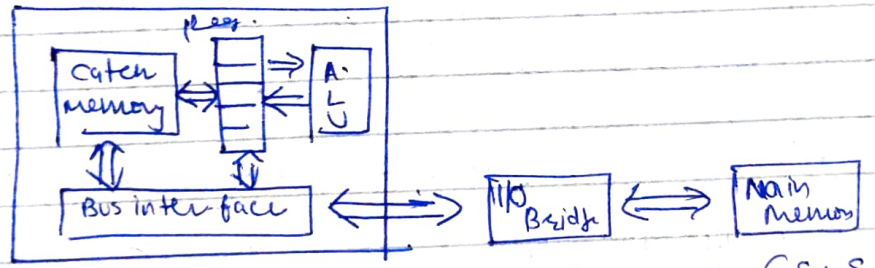
$$H_1T_1 + ((1-H_1)H_2)(T_1+T_2) + ((1-H_1)\times(1-H_2))(T_1+T_2+T_3)$$



$$H_1T_1 + ((1-H_1)H_2)T_2 + ((1-H_1)(1-H_2))T_3$$

Cache memory

→ cpu look first in Catche (L1, L2 and L3) then main memory.



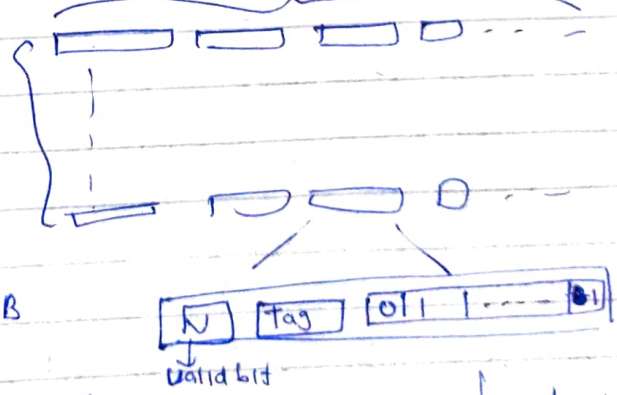
→ General ~~Catche~~ Cache organisation ⇒ (S, Σ, B)

$B = 2^b$

$S = 2^s$ = no. of set in catche

$\Sigma = 2^e$ = no. of lines in set $\Sigma = 2^s$

$B = 2^b$ = no. of bytes in line



⇒ ~~Catch~~ cache size $C = S \times \Sigma \times B$

Catche Performance → • miss rate = fraction of memory references not found in cache = 1 − hit rate

Typical miss rate = 3 − 10% for L1.

< 1% for L2, depend on size of and locality

hit rate :- time to deliver a line in the cache to the processor includes time to determine whether the line is in the ~~cache~~ cache.

Typical time → L1 → 1-2 cycle.
L2 → 5-20 cycle.

Miss penalty → Additional time required for data access because of cache miss.
typically → 50-200 cycles for main memory

Calculating Avg memory access time (AMAT)
L1 → 1 cycle. and miss rate 10%.
L2 → 10 cycle and miss rate 2%.
DRAM → 80 cycle and miss rate 0% (main memory)

AMAT = 1 + 0.10 * 10 + 0.1 X 0.02 * 80 → [always L1 cache + miss L1 * time L2 + miss L1 * miss L2 * time DRAM
= 1 + 1 + 0.16 = 2.16 cycle.

Alternative →
0.9 * 1 + 0.1 * 0.98 (10+1) + 0.1 * 0.2 * (1+10+80).
$H_1T_1 + (1-H_1)H_2(T_1+T_2) + (1-H_1)(1-H_2)(T_1+T_2+T_3)$
= 2.16 cycles

~~Is~~ would you believe 99% hit is twice as good as 97%.)
Consider. L1 cache hits → 1 cycle
L1 miss penalty → 100 cycle
97% L1 hits :- 1 + 0.03 X 100 cycle = 4 cycles
1 + 0.01 X 100 cycle = 2 cycle.

CPU Time → Program execution cycle
↳ Include cache hit time.
→ Memory stall cycle.
↳ Mainly from cache memory

Memory stalling cycle = $\frac{\text{Memory accesses}}{\text{Program}}$ × miss rate × Miss penalty

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

Eg - Instruction cache miss rate = 2%

- data " " " = 4%
- Miss penalty = 100 cycles
- Base CPI (with ideal cache performance) = 2
  - Cycles per instruct.
- Load and store are 36% instructions.

Miss cycles per instruction

- Instruction cache ⇒ 0.02 × 100 = 2
- Data cache : 0.36 × 0.04 = 1.44.

Actual CPI = 2 + 2 + 1.44 = 5.44.

→ Ideal CPU is 5.44/2 = 2.72 times faster

→ we spend 2.44/5.44 = 63% of our execution time on memory stall.

Let's Reduce ideal CPI

→ what if we improve the datapath so that avg ideal CPI was reduced → assume base CPI = 1.5

Actual CPI = 1.5 + 2 + 1.44 = 4.94.

→ Ideal CPU is 4.94/1.5 = 3.29 times faster.

→ we spend 3.44/4.94 = 70% of our execution time on memory stall

→ So, ① Decrease base CPI → greater proportion of time spent on the memory stall.

② Increase clock rate → memory stalls accounted for more CPU cycles.

Multilevel Caches Consideration

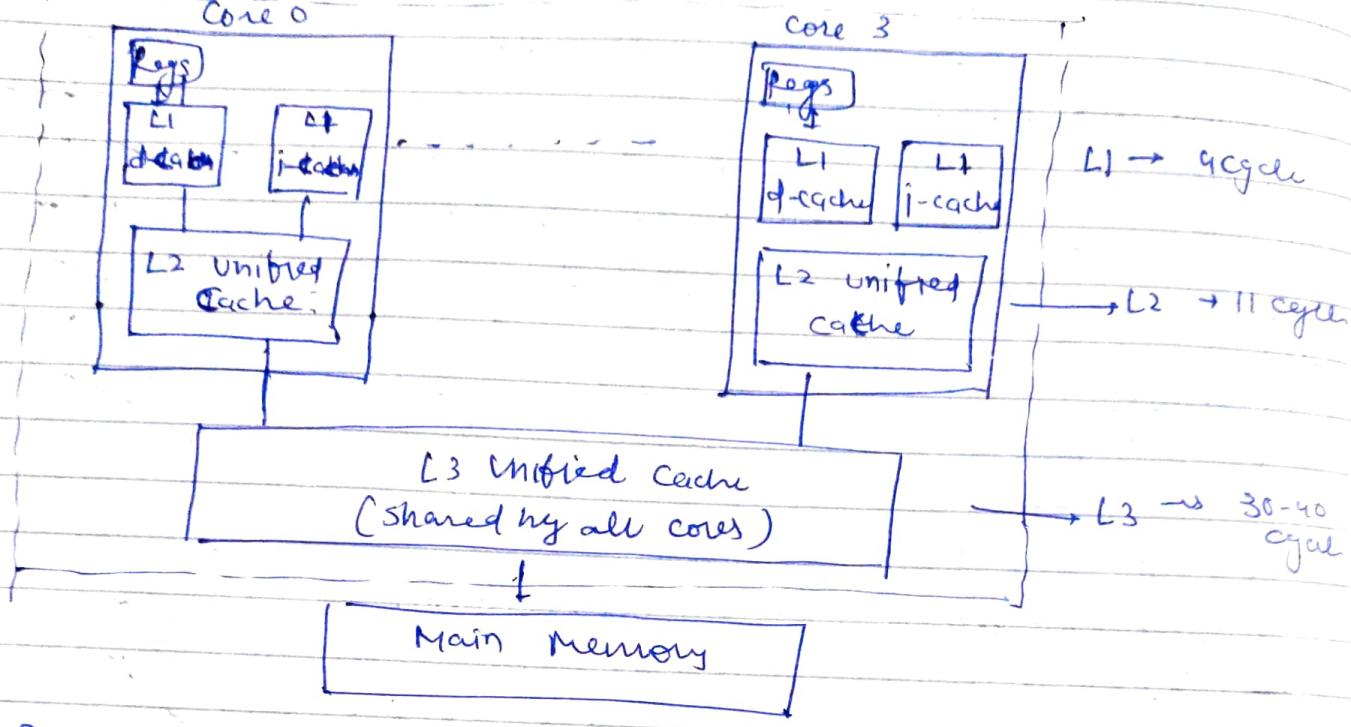
L1 → focus on minimal hit time

L2 → focus on less miss rate

Result → L1 cache → usually smaller than single cache

L1 cache block size smaller than L2 block size.

Intel Core i7 cache Hierarchy



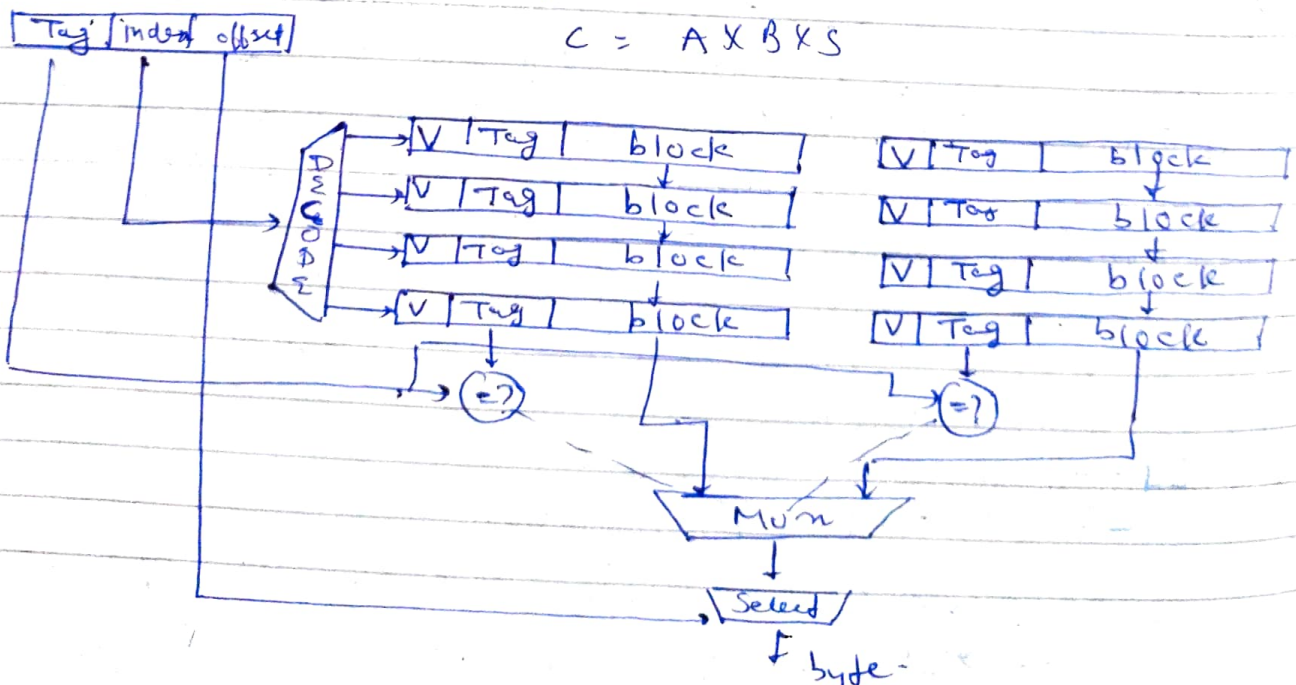
- Some other Points on Cache design.

Ideally → Each word is stored separately
→ Any word can stored anywhere
→ Requires too many comparator.

Reality → Group of words into blocks.

Classic Cache → eg- A for $E = 2$, $B = 32B$, $C = 256$ bytes.
$S = 4$.

$C = A \times B \times S$



no of lines (Blocks) = (C/B)

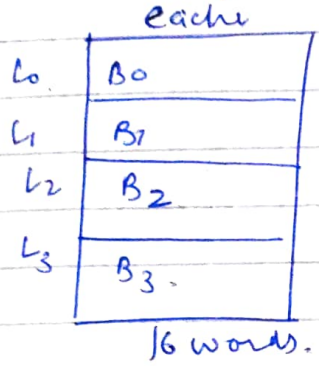
MIPS → million instruction per second

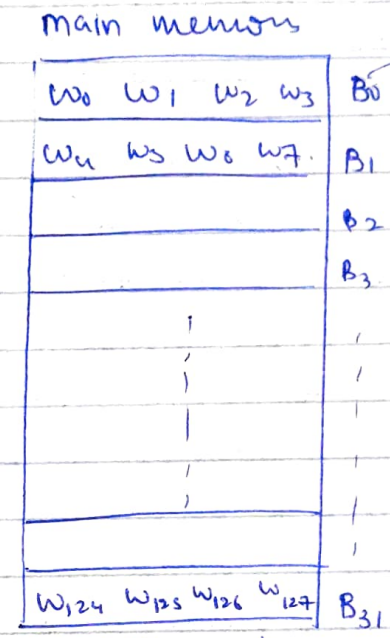
CPI → cycle per instruction

cache mapping

- Direct Mapping
- Fully associative
- k-way set associative

Direct Mapping → k mod n





line size = Block size

$n = \frac{16}{4} = 4$

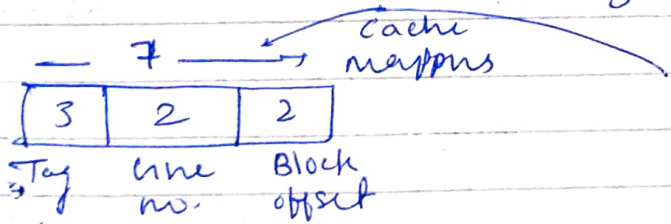
4 → Block size (4 words)

$\frac{128}{4} = 32$

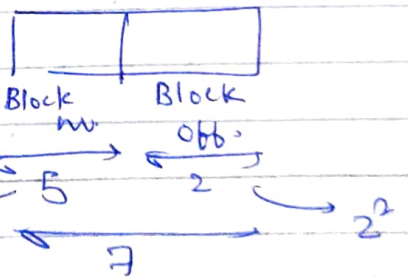
k Mod 4 → 0 Mod 4 = 0 → B0

01 Mod 4 = 1 → B1

0 – 127 → 2^7 → 7 bit



Physical Add.

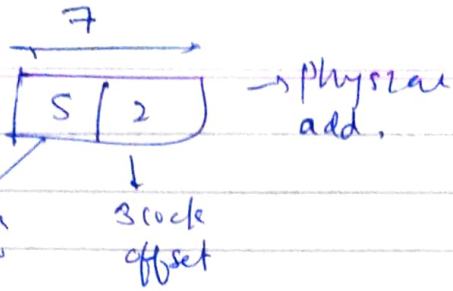


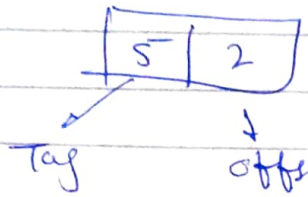
it will tell which block of K Mod 4. is present → eg → $B_0, B_4, B_8, B_{12} \dots B_{124}$ (128) which one ✓

→ disadvantage → Block assigned Placed, if other are vacant, it can't assigned to them.

Fully associative Mapping

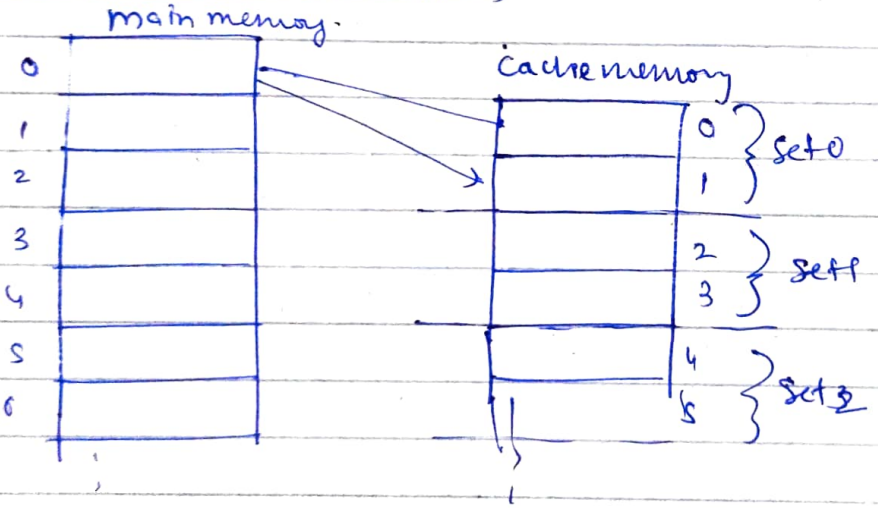
Any Block can assign any line of cache.





here we are not fixing line, so it is not present

Set ~~a~~Associative Mapping



1 set contains k no. of lines → k-way set associative Cache

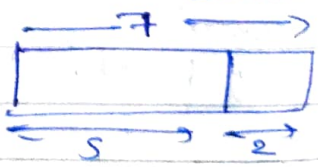
eg. Byte addressable MM size = 128 byte(B) → 2^7 byte = 7 bit (address line)

Cache Size = 32B

Block Size = 4B → 2^2 = 2 bit

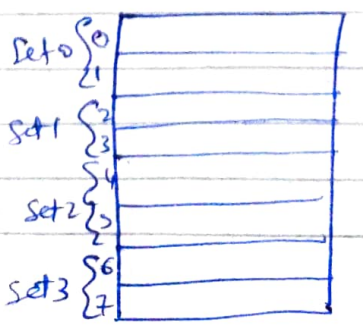
2-way associative

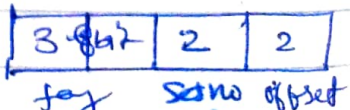
no. of memory block = $\frac{2^7}{2^2} = 2^5$



no. of lines in cache. → $\frac{2^5}{2^2} = 2^3 = 8$

no. of sets = $\frac{2^3}{2} = 2^2 = 4$





Categorization based on the instruction

CISC	RISC
→ Complex instruction set Computer.	→ Reduced instruction set Computer.
→ Large no. of instructions	→ less no. of instructions
→ variable length instruction format	→ Fixed length instruction format
↓ Program counter will face some difficulties.	
→ Cost is High.	→ Cost is low.
→ Large no. of addressing mode	→ few no. of addressing modes
→ More powerfull	
→ Several cycle instruction	
→ Manipulation directly in memory	
→ Microprogrammed control unit.	Hard wired Control unit.
→ Eg :- Motorola 6800, Intel 8080	Eg → MISP, ARM, AVR, SPARC

MULT 2:2, 3:3

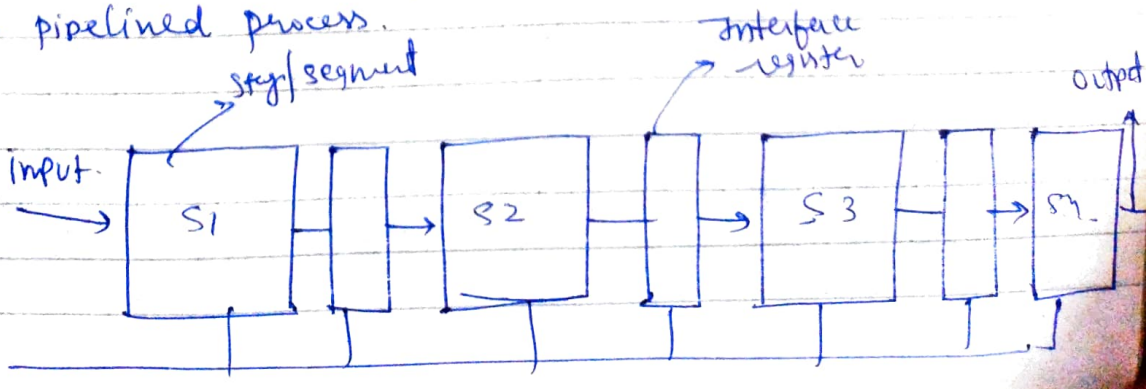
MOV RA, 3:3
MOV RB, 2:2
MUL RA, RB
Store RA,

} → Load
} Store

Pipelining

is the process of arrangment of hardware elements of CPU such that its overall performance is increased.

→ Simultaneous execution of more than one instruction take place in pipelined process.

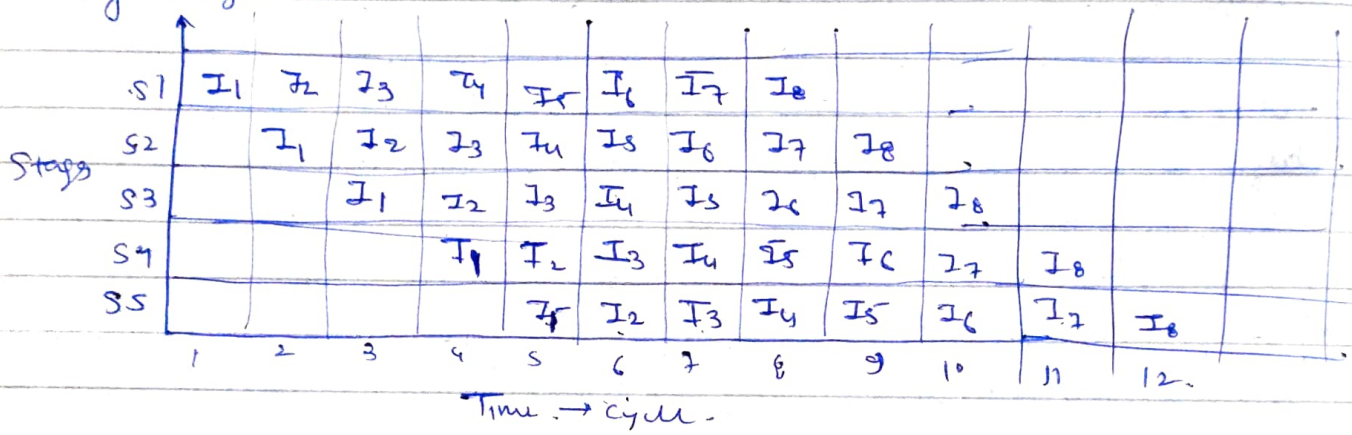


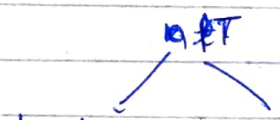
Pipeline → 5 stage → IF
ID
EX
MEM
WB

Generally RISC architecture use this pipelining.

Total Instruction → 8 → $I_1 - I_8$

Stage diagram





$T = k + (n-1)$

k → no. of stages

n → no. of instructions

$5 + (8-1) = 5+7 = 12$

Ref

From Pipelining we want $CPI \cong 1$

eg $n = 1000$, (instructions),

$T = S + (1000-1) = 1004$

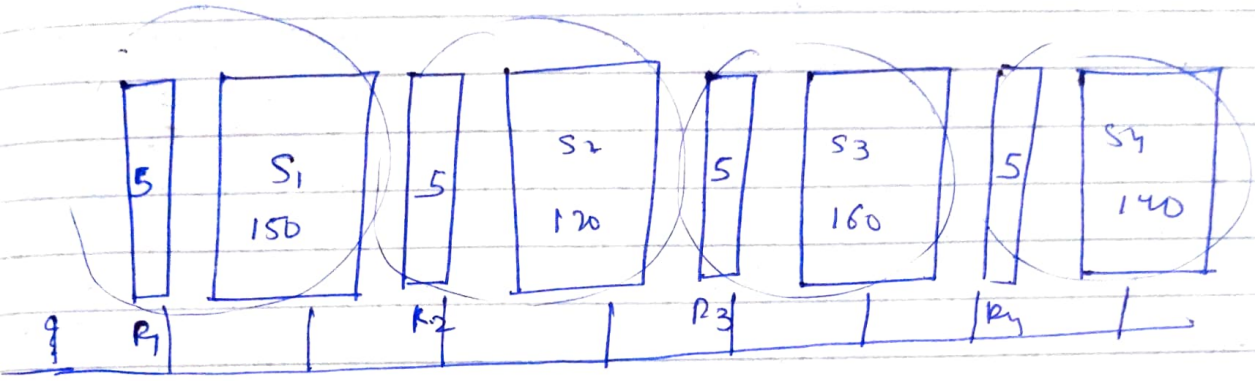
$CPI = \frac{1004}{1000} \cong 1$

Speedup = $\frac{NP\ (Time)}{P\ (Time)}$ ⇒

For 5 instructions → $\frac{40}{12} \cong 3$ (Times)

efficiency = $\frac{\text{Use box.}}{\text{Total Box}} = \frac{8 \times 5}{12 \times 5} = \frac{2}{3}$ efficiency

Q. A 4-stage pipeline has stage delay as 150, 120, 160 and 140 ns. Registers are used b/w stages and have delay of 5 ns each. Assuming constant clock rate, the total time taken to process 1000 items on this pipeline will be ____ ?



First we have to find the clock freq. = $\frac{1}{\text{Clock period}}$

$= \frac{1}{165 \text{ ns}}$

Time = $1 \times 4 \times 165 + 999 \times 1 \times 165$

$= 165 + 495 \text{ ns} \approx 165.5 \text{ µs}$

Q. Considered a non-pipelined processor with clock rate of 2.5 GHz and avg cycles of instructions of 4. The same processor is upgraded to pipelined processor with 5 stages, but due to internal pipeline delay, the clock speed is reduced to 2 GHz. Assume that there is no stall in pipeline. The speedup achieved in pipelined processor is ____ ?

$$\text{Speedup} = \frac{T_{NP}}{T_P}$$

$$T_{NP} = 4 \times \frac{1}{2.5} \times 10^{-9}$$

$$T_P = 1 \times \frac{1}{2} \times 10^{-9}$$

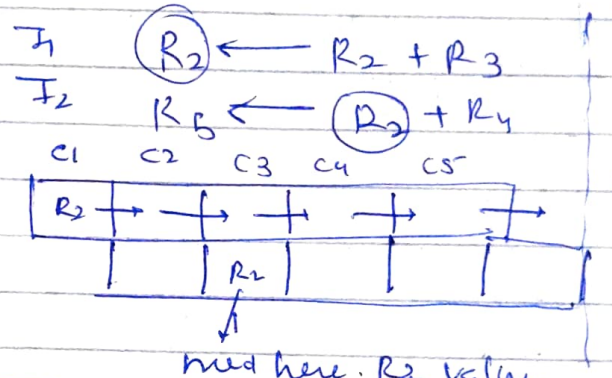
CPI ≈ 1

$$\text{Speedup} = \frac{4 \times \frac{1}{2.5}}{\frac{1}{2}} = 3.2$$

Hazards in pipelining

It is very dificult to make CPI = 1, and those difficulties are called Hazards in pipelining

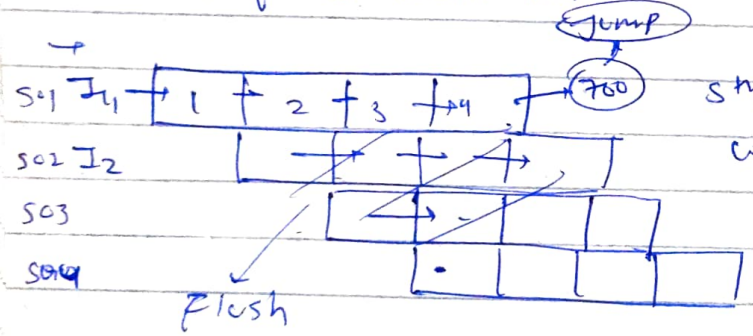
- Data Hazard
  - RAW (TD) True dependancy
  - WAR (AD) → Anti dependancy
  - NAW (OD) → output dependancy
- Structual Hazard → multiple resources like memory, location, perticular resister → access same time
- Control Hazard.



as accessing memory for read and write at same time

But it will present of the same clock, so it is a Data Hazard with Read after write (RAW) problem

when we flush the other processing instructions due to Branching coming at first instruction



Structual Hazard

↳ when the multiple ~~resource~~ instructions need same resource

Solutions → Resource duplication

stall.

Control Hazard.

Solution → stall after decoded a jump instruction

flushing → reduce the performance.

Data hazard

RAW → Solution. → stall.

I1 $R_3 \leftarrow R_1 + R_2$

I2 $R_5 \leftarrow R_3 + R_4$

Detection	Domain	Range
I1	R_1 R_2	R_3
I2	R_3 R_4	R_5

I_1 (Range) $\wedge$ I_2(Domain) $\neq \phi$

↳ RAW Data Hazard.

② WAR :

I_1: $R_1 \leftarrow R_2 * R_3$

I_2: $R_2 \leftarrow R_4 + R_5$

If this modify first, and then Instruction I_1 will read R_2, it will occur inconsistency in the data

↳ this is very special case, which can be occure in ~~Auto increment~~ Auto increment.

Detection → Domain $(I_1) \wedge$ Range $(I_2) \neq \phi$

③ WAW

I_1: $R_3 \leftarrow R_1 * R_2$

I_2: $R_3 \leftarrow R_4 + R_5$

Is it can happen when instructions are executing parallely

Range $(I_1) \wedge$ Range $(I_2) \neq \phi$

CPU performance evaluation

CPU clock rate depend on the specific CPU organization and hardware implementation.

Micro operation → elementary hardware operation performe in CPU clock cycle

CPI → (Cycles per instruction) → IPC = $\frac{1}{CPI}$

instruction per cycle.

- Avg (or effective) CPI of a program → Avg CPI of all instruction in the program of given CPU & design.

CPU instruction

↳ Instruction Fetch → Instruction decode → Operand fetch → Execute → Result store

Next instruction

Performance → total execution time should shorter.

↓

{Performance = $\frac{1}{\text{Execution time}}$}

Speed up = $\frac{\text{Performance}_A}{\text{Performance}_B} = n = \frac{\text{Execution time}_B}{\text{Execution time}_A}$.

↓ how to compare performance

↘ n → A is n times faster than B.

CPU Execution time

Clock cycle time = C = $\frac{1}{\text{Clock rate}}$.

CPI → Clock/Instruction.

CPU Time = $\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instruction}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$

$$T = I \times CPI \times C$$

- I → Instruction count
- CPI → Avg or effective CPI for program.
- C → CPU clock cycle

CPI → Total Program execution cycle / Instruction count executed.

{Total cycle = CPI × I}

Q. I = 10,000,000

CPI = 2.5

Clock rate = 200 MHz → C = $\frac{1}{200} \times 10^{-6}$

$T = I \times CPI \times C = 10000000 \times 2.5 \times \frac{1}{200} \times 10^{-6}$

$= \frac{5}{4} = 0.125 = 125$ ms

CPU performance

Affecting factor → ISA (Instruction Set Architecture) / Compiler, program, CPU design, technology (VLSI)

Q. eg. Program A → $I_A = 10000000$

$CPI_A = 2.5$ cycles/Instruction

CPU clock rate = 200

Program B → $I_B = 195000000$

$CPI_B = 3.0$

$Clock\ Rate_B = 300\ MHz$.

Speedup = $\frac{Performance_B}{Performance_A} = \frac{A\ execution\ time}{B\ execution\ time}$

$= \frac{0.125}{0.095} = 1.32$ or 32% faster.

Instruction Types → n types or classes of instructions executed on CPU → eg ALU, Branch.

C_i = Count of instruction of type i executed.

CPI_i = Cycles per instruction of type i

$$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

[and Executed Instruction count $I = \sum C_i$]

eg → Instruction class → A, B, C

	CPI	Instruction count
A	1	2
B	2	1
C	3	2

Clock cycles → $2 \times 1 + 1 \times 2 + 2 \times 3 = 10$

Avg CPI → Clock cycles / I = $10/5 = 2$.

Instruction frequency & CPI.

C_i → Count of instruction of type i

CPI_i →

F_i → Frequency or fraction of instruction of type i

$= \frac{C_i}{\text{total executed Instruction count}} = C_i/I = \frac{C_i}{\sum C_i}$

$$CPI = \sum (CPI_i \times F_i)$$

↙ avg or effective CPI.

fraction of ~~time~~ total time execution for i^{th} instruction = $\frac{CPI_i \times F_i}{CPI}$

eg Instruction Type freq. & CPI

OP	Freq F_i	CPI_i	$\rightarrow$ $CPI_i \times F_i$	% Time
ALU	50%	1	0.5	23% = 0.5/2.2
Load	20%	5	1	45% = 1/2.2
Store	10%	3	0.3	14% = 0.3/2.2
Branch	20%	2	0.4	18% = 0.4/2.2
			2.2	

$CPI =$

Choosing Programs to evaluate Performance

↳ Benchmarks / Level of program → to evaluate performance

① Actual target workload → Full applications that run on the target machine.

② Real Full program - based Benchmark → Programs that are typically targeted application

③ small "kernel" benchmark

④ Microbenchmark → integer, floating point, local memory

MIPS (Million Instructions per second) Rating

↳ measure of how many millions of instructions are executed per second.

$$\text{MIPS Rating} = \text{Instruction count} / (\text{Execution time} \times 10^6)$$

$$= \text{Instruction count} / (\text{CPU clock} \times \text{Cycle Time} \times 10^6)$$

$$= \frac{\text{Instruction count} \times \text{clock rate}}{\text{Instr. time} \times CPI \times 10^6}$$

$$= \text{clock rate} / (CPI \times 10^6)$$

Problem with MIPS rating ⇝ As shown above MIPS rating does account the count of instructions executed.

→ A higher MIPS rating in many cases may not mean higher performance or better execution time.

⇝ Thus it cannot be used to compare computers/CPUs with different instruction set.

Under what conditions can the MIPS rating be used to compare performance of diff. CPUs ?

Conditions: ① The same program is used
② Same ISA is used
③ The same compiler is used.

eg → Q. For a machine (CPU) with instruction set.

Instruction class	CPI
A	1
B	2
C	3

Compiler produces the following executed instruction count

	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

Clock rate = 100 MHz.

$$MIPS = \frac{\text{Clock rate}}{CPI \times 10^6} = \frac{100 \times 10^6}{CPI \times 10^6} = \frac{100}{CPI}$$

For Compiler 1

↳ $CPI_1 = (5\times1 + 1\times2 + 1\times3)/(5+1+1) = 10/7 = 1.43.$

MIPS rating = $\frac{100}{1.428} = 70.0$ MIPS.

CPU time 1 = $(5+1+1)\times1.43/(100\times10^6) = 0.1\ \mu s$

~~CPU time 2~~ :

For compiler 2 :

↳ $CPI_2 = (10\times1 + 1\times2 + 1\times3)/(10+1+1) = 15/12 = 1.25$

$MIPS_2 = \frac{100}{1.25} = 80.0$ MIPS.

CPU time$_2$ = $(10+1+1)\times1.25/(100\times10^6) = 0.15\ \mu s.$

Here we can see that, Compiler 2 is better while in reality code produced by compiler 1 is faster.

MFLOPS → (Million Floating point operation Per Second)

↳ floating point operations → addition, subtraction, multiplication, division operation applied to floating point representation

→ measure the millions of floating point operation per second.

$$MFLOPS = \frac{\text{No. of floating point operation}}{\text{Execution time} \times 10^6}$$

→ It is better to comparision measure b/w different machines. (applied even if ISAs are different) than MISP rating.

→ It is depend on the type of floating point operation present in the program.

Peak MFLOPS → simplest floating point instructions (with the lowest CPI).

Quantative Principles of Computer Design

Amdahl's Law → The performance gain from improving some portion of a computer is calculated by.

$$\text{Speedup} = \frac{\text{Performance of entire task using the enhancement}}{\text{Performance of entire task without using the enhancement.}}$$

ie, using some enhancement

$$\text{or} = \frac{\text{execution time without enhancement}}{\text{execution time for entire task using enhancement.}}$$

→ Performance improvement or speed up due to enhancement E.

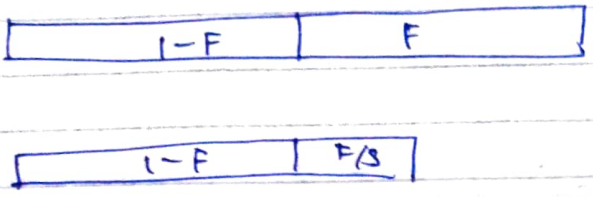
$$\text{speedup}(E) = \frac{\text{Performance with } E}{\text{performance without } E.}$$

↳ Amdahl's law

Enhancement accelerates a fraction F of the execution time by fraction 's'. and remainder of time unaffected.

Execution time with Σ $= ((1-F) + F/S) \times$ Execution time without Σ.

$$\text{Speedup } (\Sigma) = \frac{\text{Execution time without } \Sigma.}{((1-F) + F/S) \times \text{Execution time without } \Sigma}$$



$$\Sigma = \frac{1}{(1-F) + F/S}$$

OP	Freq.	Cycle	CPI(i)	% Time
ALU	50%	1	0.5	23%
Load	20%	5	1.0	45%
Store	10%	3	0.3	14%
Branch	20%	2	0.4	18%

CPI = 2.2 → previous one.

→ Here Load instruction enhanced → 5 to 2 cycles.

Sol: $F = 0.45$ or 45%

$1-F = 1 - 0.45 = 0.55$ or (100 − 45)% ⇒ 55%.

$S = 5/2 = 2.5$

Using Amdahl's law → $\text{Speedup} = \frac{1}{(1-F) + F/S} = \frac{1}{0.55 + 0.45/2.5}$

$= 1.37.$

$CPI_{New} = \frac{2.2}{1.37} = 1.6.$

Alternative Method

Old CPI = 2.2

New CPI = $0.5 \times 1 + 0.2 \times 2 + 0.1 \times 3 + 0.2 \times 2 = 1.6$

$$\text{Speedup} = \frac{\text{original execution time}}{\text{New exe. time}} = \frac{I \times CPI_{old} \times C}{I \times CPI_{New} \times C}$$

$= \frac{2.2}{1.6} = 1.37$

Extending Amdahl's law to multiple Enhancement

$$\text{Speedup} = \frac{1}{((1 - \Sigma F_i) + \Sigma \frac{F_i}{S_i})}$$

→ Note :- All F_i refer to original execution time before the enhancement are applied.

Eg $S_1 = 10$ $F_1 = 20\%$

$S_2 = 15$ $F_2 = 15\%$

$S_3 = 30$ $F_3 = 10\%$

$$\text{Speedup} = \frac{1}{(1-\sum F_i) + \sum \frac{F_i}{S_i}} = \frac{1}{(1-0.2-0.15-0.1) + \left(\frac{0.2}{10} + \frac{0.15}{15} + \frac{0.1}{30}\right)}$$

$$= \left(\frac{1}{0.55 + 0.0333}\right) = 1.71.$$

"Reverse" Multiple Enhancement Amdahl's Law

Amdahl's law assumes that the fraction given refer to original execution time.

→ If for enhancement S_i the fraction F_i it affects is given as fraction of resulting execution time after the enhancement were applied

$$\text{Speedup} = \frac{((1-\sum F_i) + \sum F_i S_i) \times \text{Resulting execution time}}{\text{Resulting execution time}}.$$

$$= ((1-\sum F_i) + \sum F_i S_i)$$

Eg. → Previous data → is used.

then

$$\text{Speedup} = (1-0.2-0.15-0.1) + 0.2\times 10 + 0.15\times 15 + 0.1\times 30$$

$$= 0.55 + 2 + 2.25 + 3$$

$$= 7.8.$$ ✓