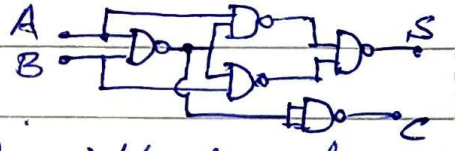
Introduction:

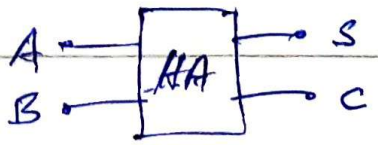
Computers have become an essential part of our daily life. They are everywhere in the form of desktop computers or laptops or tablets, or mobile phones or other intelligent appliances as embedded systems.
Hence it is required to understand how does a computer work.
We need to distinguish first between Computer Architecture and Computer Organization.

Computer Organization: It is the design of the components and functional blocks using which computer systems are built.
Analogy: (i) Civil engineers task during building construction (cement, bricks, iron rods and other building materials)



(ii) Half Adder ckt. with NAND Gates.

Computer Architecture: (i) It deals with how to integrate the components to build a computer system to achieve a desired level of performance.
Analogy: An architect's task during the planning of a building. (overall layout, floor planning etc.)
(ii) Half adder block diagram.



Scanned with Any Scanner

classmate
Date
Page

Computer Architecture:

- Deals with functional behaviour of computer systems.
- Design implementation for the various parts of computer.

Computer Organization:

- Deals with structural relationship
- Operational attributes are linked together and contribute to realize the architectural specification.

Scanned with Any Scanner

Generations of Electronic computers

Generation	Technology and Architecture	Software and Applications	Representative Systems.
1st (1945-54)	Vacuum tubes and relay memories, fixed point arithmetic.	Machine/assembly languages, single user,	ENIAC, Princeton IAS, IBM 701
2nd (1955-64)	Transistors, memories, floating point arithmetics, I/O processors	HLL used with compilers, subroutine libraries	IBM 7090, CDC 1604, Univac LARC.
3rd (1965-74)	Integrated circuits (SSI/MSI), Micro-programming, pipelining, cache.	Multiprogramming and time sharing OS, Multi user applications	IBM 360/370, CDC 6600, TI-ASC PDP-8 Intel-8008.
4th (1975-90)	LSI/VLSI and semiconductor memory, multi-processors,	Multiprocessor OS, languages, compilers,	IBM 3090, Intel 8086, 8088.
5th (1991-present)	Advanced VLSI processors, memory, high density packaging	Super scalar processors, systems on chip.	Intel 486

Scanned with Any Scanner

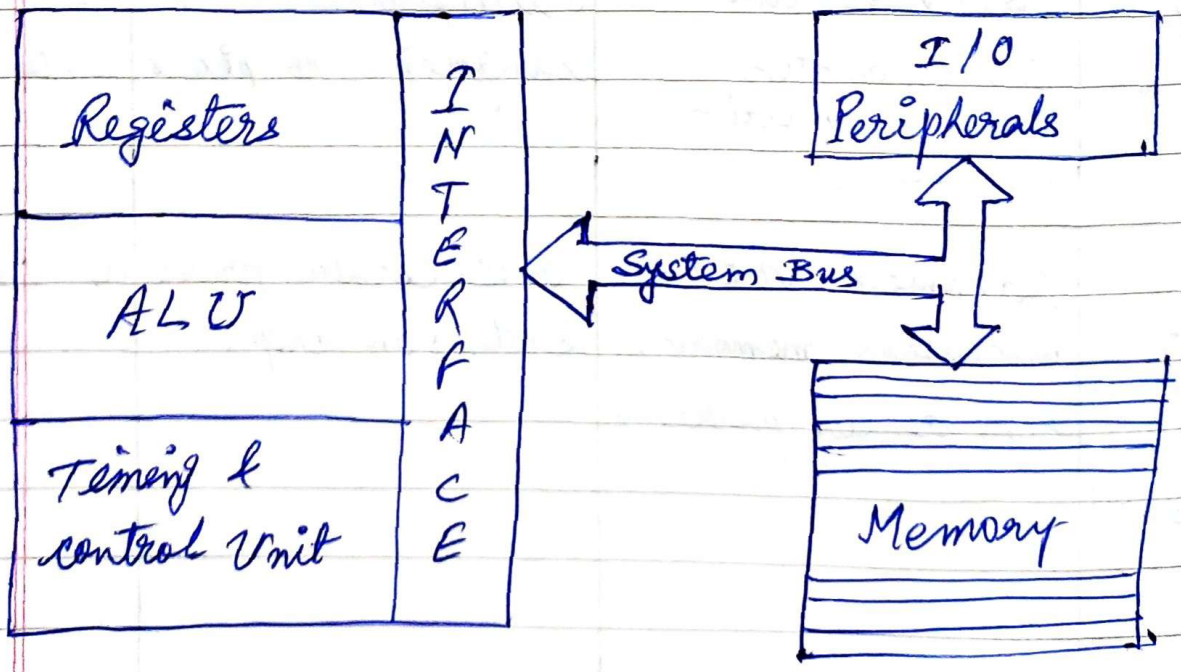
Functional Units of a generic computer

① **Processor:-** most important part. It consist of a collection of Registers, an Arithmetic Logic Unit (ALU) and Timing and control Unit. Along with it, it has another section, called Interface. This is (the processor) called the Brain of the system.

② **Memory:** It stores all the instructions and data inside the memory.

③ **I/O Peripheral:** Inputs and outputs are generated through the I/O peripherals and sent to the processor or stored in the memory.

④ **System Bus:** The interconnection of all the above mentioned functional components ~~are~~ is carried out through the system bus.



Scanned with Any Scanner

Historically there have been two types of computers:

1. **Fixed Program Computers**: Their function is very specific and they can not be reprogrammed, e.g. calculators.

2. **Stored Program Computers**: They can be programmed to carry out many different tasks. Applications are stored on them, hence the name. e.g. modern day P.C.

and Harvard Architecture → uses two diff. phys. add. for storing instruction & data.

The two most popular computer architectures are —

(I) **Von Neumann Architecture**: This concept was first published by John Von Neumann in 1945. His computer architecture design consists of a control unit, Arithmetic and Logic Unit (ALU), Memory Unit, Registers and Inputs/Outputs.

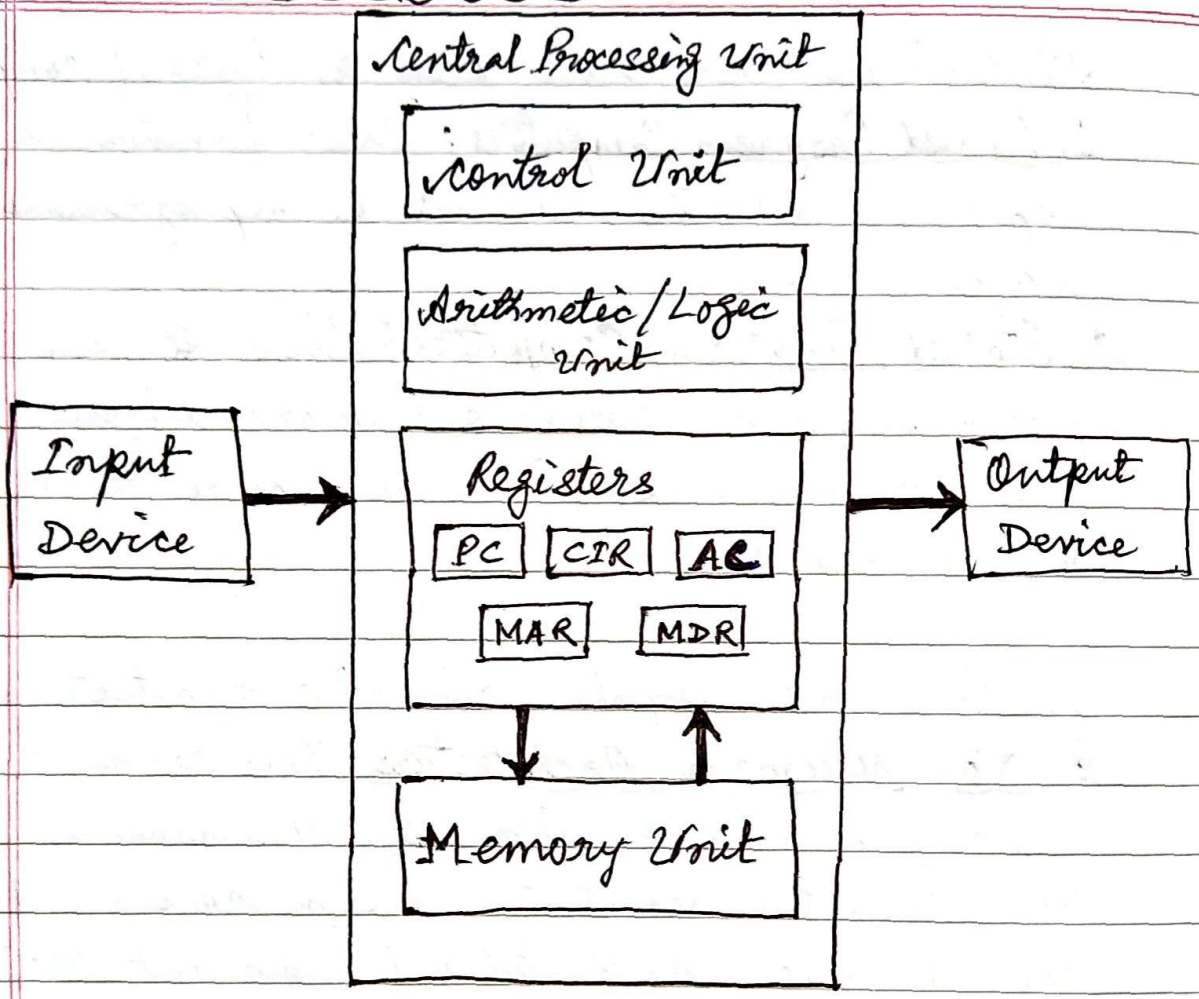
This architecture is based on the Stored-program computer concept, where instruction data and program data are stored in the same memory.

CPU: The Central Processing Unit (CPU) is the electronic circuit responsible for executing the instructions of a computer program. It is sometimes referred to as the microprocessor or processor.

Registers: Registers are the high speed storage devices in the CPU. All data must be stored in the registers before it can be processed.

Scanned with Any Scanner

Von Neumann Architecture



PC : Program Counter ⇒ it contains the address of the next instruction to be executed.

AC : Accumulator ⇒ in this register intermediate arithmetic and logic results are stored.

CIR : Current Instruction Register ⇒ it contains the current instruction during processing.

MAR : Memory Address Register ⇒ it holds the memory location of the data that needs to be accessed.

MDR : Memory Data Register ⇒ it holds the data that is being transferred to or from memory.

Scanned with Any Scanner

ALU: Arithmetic Logic Unit ⇒ The ALU allows arithmetic operations (add, subtract etc.) and logical operations (AND, OR, NOT etc.) to be carried out.

Control Unit (CU) ⇒ The control unit controls the operation of the computer's ALU, memory and input/output devices telling them how to respond to the program instructions it has just read and interpreted from the memory unit.
The control unit also provides the timing and control signals required by other computer components.

Buses: Buses are a collection of wires meant for data transmission from one part of the computer to another, connecting all major internal components to the CPU and memory.

A standard CPU system bus is comprised of the following three types of buses.

(i) Address bus: It carries the addresses of the data (but not the data) between the processor and memory.

(ii) Data bus: it carries data between the processor, the memory unit and the input/output devices.

(iii) Control bus: it carries the control signals/commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer.

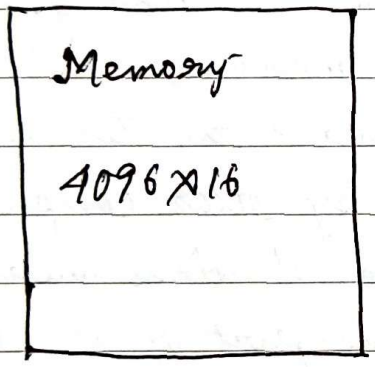
Scanned with Any Scanner

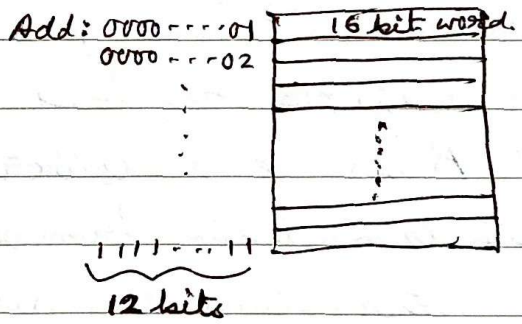
Memory Unit : The memory unit consists of RAM, sometimes referred to as Primary or main memory. Unlike a harddrive (secondary memory), this memory is fast and also directly accessible by the CPU.

RAM is split into partitions. Each partition consists of an address and its contents (bol

Diff. types of Registers

$4096 = 2^{12}$ = no. of words





11 0

| Address Register | → it holds the address of the memory location from where data is to be fetched.

Address register gives the address to the memory which decodes the address and then fetches the data from the address.

Size of address register = 12 bits. (0000 ... 00 ... 1111 ... 11)

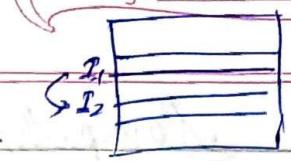
15 0

| Data Register | size of data register = 16 bits

15 0

| Accumulator | → size = 16 bits

classmate
Date
Page



11 0

Program Counter → size = 12 bits.

It stores the address of the next instruction to be executed.

Format of Instruction

1 bit	3 bits	12 bits
I	opCode	operand
0/1	eg. add/sub/etc.	data

0 → direct addressing

1 → Indirect addressing.

opCode operand.
↓ ↓
0 ADD 4500
↑
add. where data is available.

1 ADD 4500
↳ 1002
another address where data is available.

15 16 bit 0

Instruction Register → it holds the current instruction.

15 0

Temporary Register (16 bits) → stores data temporarily
size = 16 bit.

Input Register → size = 8 bits (may be changed)
It takes data from i/p devices and stores them before sending it to accumulator or ALU.

Output Register → size = 8 bits
It takes the data from ALU and sends the data to the o/p devices.

Scanned with Any Scanner

Computer Architecture : (def)

It is the design of computers, including their instruction sets, hardware components and system organization.

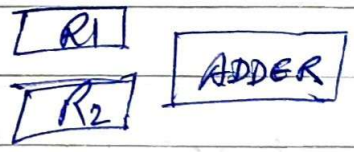
Instruction set Architecture (ISA)

Ex:

```
MOV R1, 02H
MOV R2, 03H
ADD R1, R2
STORE X, R1
```

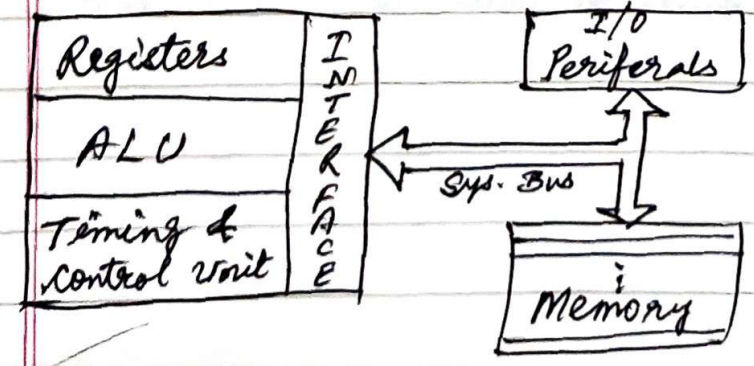
Hardware System Architecture (HSA)

Ex: Two Registers $R1$ & R_2 and an ADDER is required.



Computer Architecture

Von Neumann Architecture

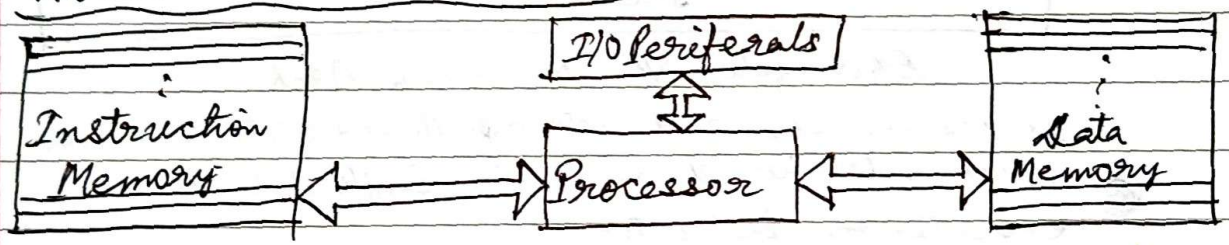


Non - Von Neumann Architecture

- Harvard Architecture
- Modified Harvard Architecture. (It is a combination of Von-Neumann and Harvard Architecture)

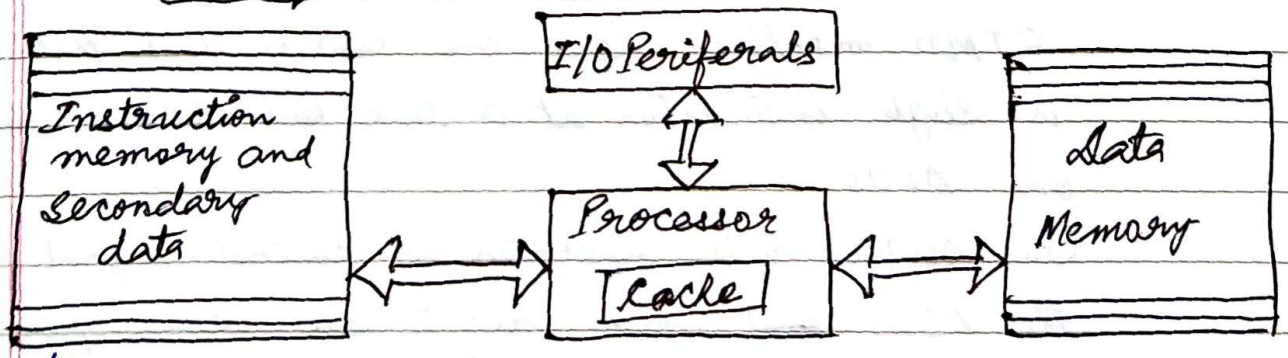
Von Neumann Bottleneck: In Von Neumann architecture both instructions and data are stored in the same memory unit. So, the instruction and data are fetched over the same path, hence ~~both cannot be~~ the processor can not read an instruction and operate on data simultaneously. This phenomenon is called Von Neumann Bottleneck.

Harvard Architecture



Here instruction and data are fetched on different paths. So, the processor is able to read an instruction and operate on data simultaneously.
Hence, Harvard Architecture is faster than Von Neumann Architecture.

Modified Harvard Architecture:



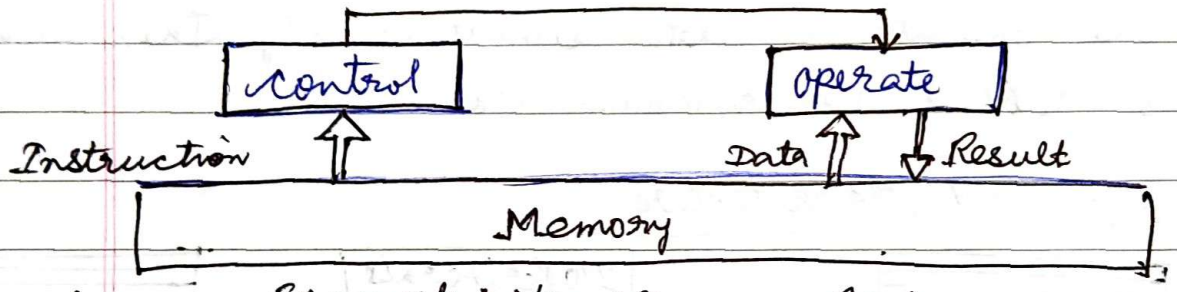
It relaxes the strict division of Instruction memory and data memory. Cache is a fast memory.

Scanned with Any Scanner

Classification: (Flinn's Taxonomy)

depending on number of processors, no. of programs they can execute and the memory structures being used.

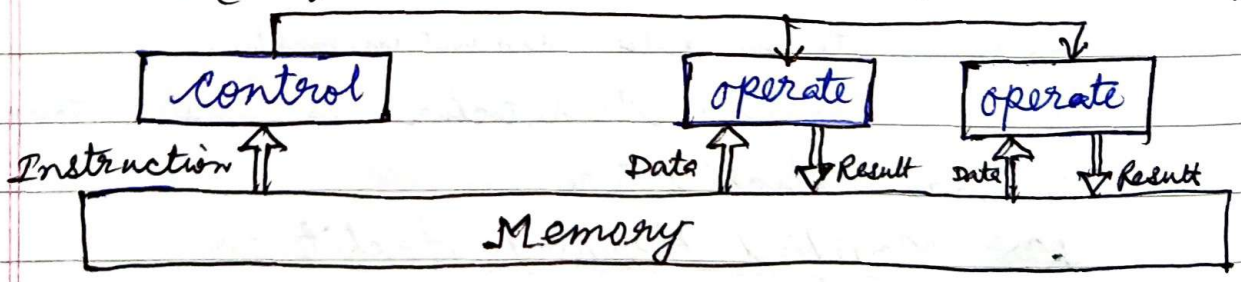
① SISD (Single Instruction stream, Single Data stream).



Example: Von Neumann Arch.

It has one CPU and fetches & stores one data at a time (single intruction stream) (Single data stream) executes one instruction at a time.

② SIMD (Single Instruction Stream, Multiple Data stream)



SIMD machines have one control unit that execute a single instruction at a time but have more than one ALUs.

The control unit generates the control signal for all the ALUs ~~and~~ which execute the same operation on different set of data items, hence it is called Multiple Data streams.

Scanned with Any Scanner

MISD

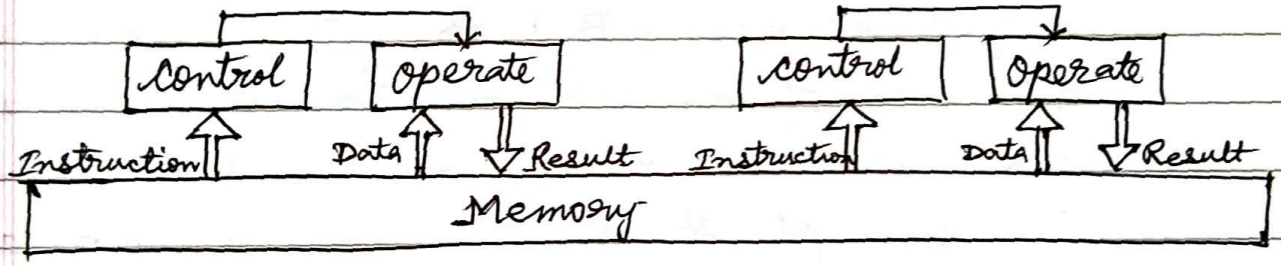
③ Multipe Instruction stream, Single Data stream.

- it does not have any practical implementation.
- machine of this class would execute various different programs on the same data items.

④ MIMD:

(Multiple Instruction stream, Multiple Data stream)

- these are also known as Multiprocessors.



- these are comprised of more than one independent processors and each one of them can execute a different set of instruction, hence Multiple Instruction stream.
- it operates on different set of data, hence Multiple Data stream.

Scanned with Any Scanner

Memory:

Everything in a computer eg. text, audio file, video file, image etc. are stored in the computer memory encoded as bits. That is each memory cell can have a 0 or a 1 and all the data are comprised of millions of bits and are processed by the processor which is the brain of the computer. So store more data, we need larger memories. But with the increase in the size of memory, the time to access them also increases which make the computer slower.
For example if we have 2GH processor then
$Time = \frac{1}{freq} = \frac{1}{2 \times 10^9}$ sec $= 0.5 \times 10^{-9}$ sec $= 0.5$ nsec.

that is the computer takes 0.5 nsec to complete a task.
So, if the memory device is slower than CPU then the CPU will be idle most of time and we wouldn't have an efficient machine.
Also, the size and cost are considered while designing memory along with speed.
That's why we have various memory devices associated to our computer.

DRAM

The memory devices meant to perform immediate

tasks are called ~~intermedi~~ Primary memory. The memory which is used for permanent storage are called Secondary memory.

- To operate on a file the OS first brings the data from the secondary memory into the Primary memory.
- To make the execution of the instructions as fast as possible, the cells in the primary memory can be accessed in any order. That's why these are called Random Access Memory or RAM. These are actually Dynamic RAM (DRAM). because for each memory location there is a transistor connected to a capacitor. The transistor can hold a binary bit as long as the capacitor is charged. Hence a periodic recharging is required for data retention.

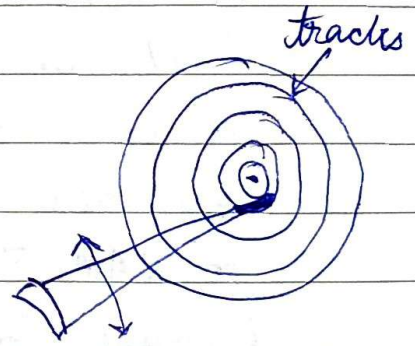
- Another faster memory is the cache memory. It is made up of Static RAM which does not have any capacitors. Cache memory is very costly as compared to the main or primary memory. It is the fastest memory
- The ~~ch~~ cache and primary memory are volatile memory. That means, they can retain the data as long as there is power supply.

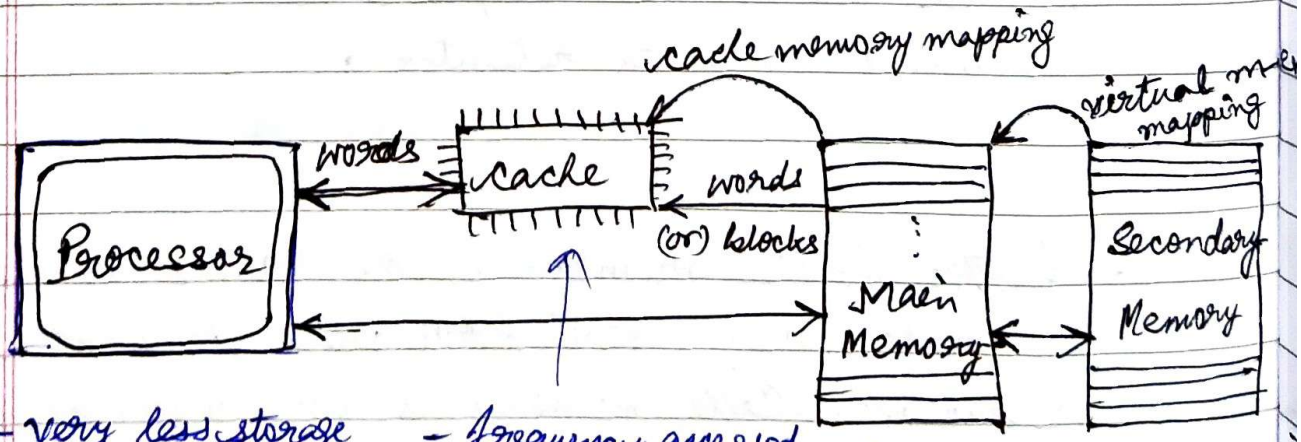
Scanned with Any Scanner

- To store the data permanently secondary memory is used.

Secondary Memory:

- Secondary memory are slower than primary memory.
- They retain data permanently even after power
- bigger in size.
- cheaper.
- semi-random accessibility.
  getting access to any track is random, but getting to the particular block of data is a sequential movement. So, the time to access any data in the hard drive becomes longer.



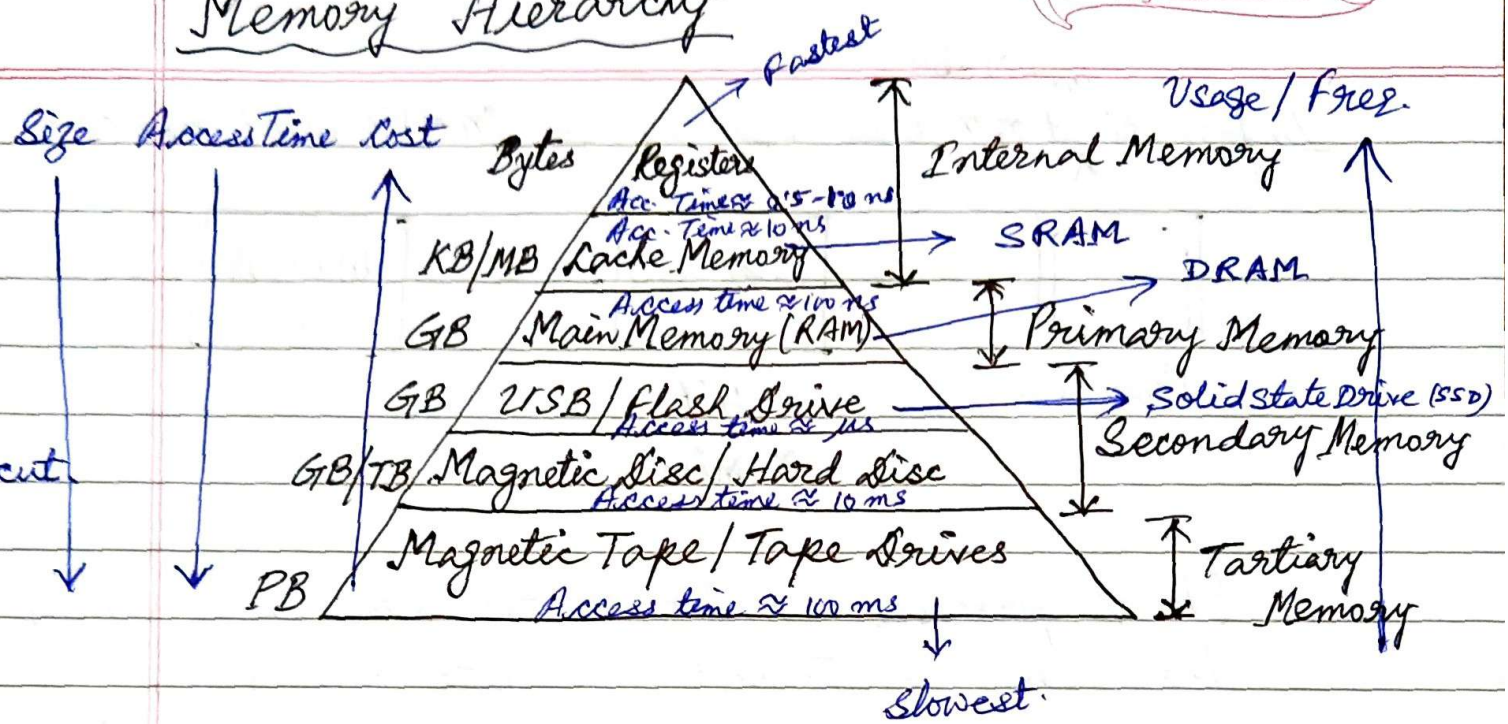


- very less storage capacity
- frequency accessed data/instructions are stored in the small but fastest memory.
- slow for the processor

- the way in which the cache and main memory communicate is known as cache memory mapping.

Scanned with Any Scanner

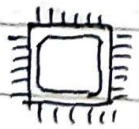
Memory Hierarchy



The purpose of memory hierarchy is to bridge ~~between the fast req~~ the speed mismatch between the fast processor and the slow memory at a reasonable cost.

Memory Interfacing:

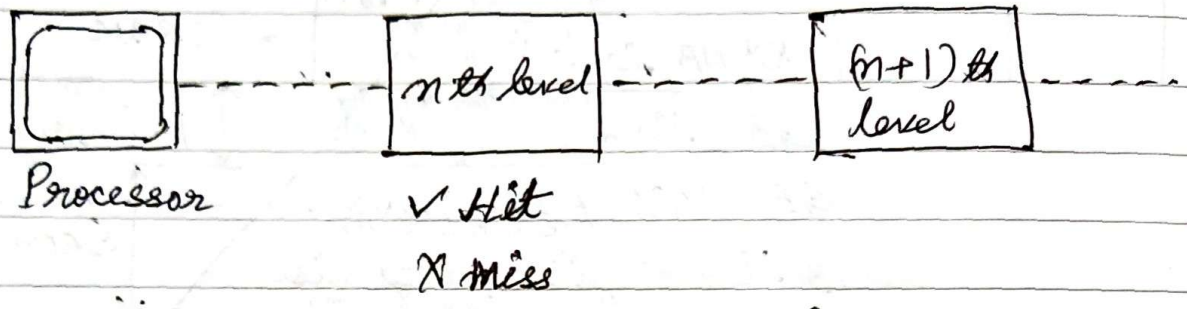
- It is a part of Computer Organization
- It deals with the way of connecting various levels of Memory units to the Processor or I/O peripherals.



Unit of measuring speed: MIPS
(Million Instruction Per Second)

Scanned with Any Scanner

Information available at nth level memory $\subseteq$ (n+1)th level memory.

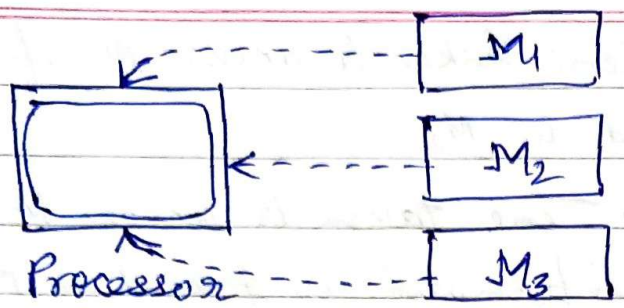


If the processor refers to the nth level memory for some instruction or data and it is found there, it is termed as Hit and if it is not found in the nth level memory then it is termed as a Miss.

In case of a Miss, the processor goes to the next level of [(n+1)th level] memory to look for the same and the time taken should be considered for both the nth level & (n+1)th level. However it depends on the way the interfacing is done.

<u>case-I</u> All the three level memories are connected simultaneously to the processor. Whenever the processor looks for some information it looks into all the three levels of memory side by side.

classmate
Date
Page



	Access time	Hit Ratio
M_1	T_1	H_1
M_2	T_2	H_2
M_3	T_3	

Hit Ratio :-

Let, Total no. of instructions = 100 stored in secondary memory

No. of instructions copied to main memory = 80.

$\therefore$ Main Memory's Hit Ratio $= \frac{80}{100} = 0.8$ or, 80%

That is, there is 80% chance of the processor of getting the required information inside the Main Memory.

As the last level of storage, the Hit ratio of Secondary Memory is 1.0 or, 100%, because all the informations are stored in the secondary memory.

Effective/Averge Memory Access Time (T_{avg}):

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

$H_1 T_1 \Rightarrow$ Time taken to access M_1 if its a Hit in M_1.

$(1-H_1) \Rightarrow$ chance of processor may not find the info in M_1 which is known as Mis Ratio of M_1.

Scanned with Any Scanner

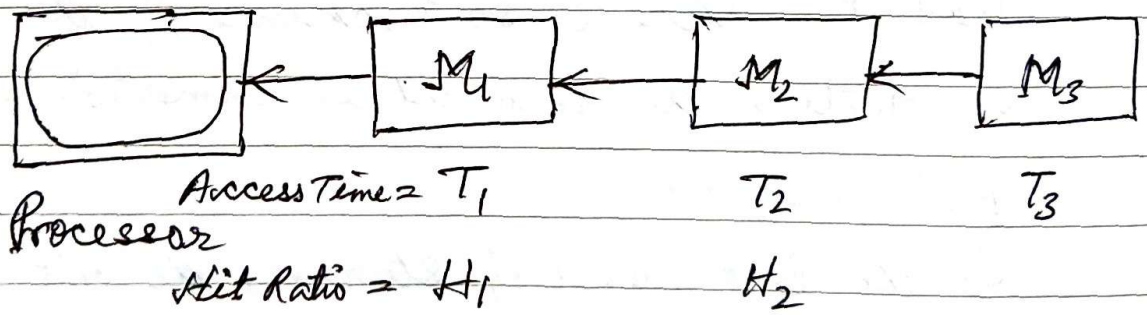
$(1-H_1) \times H_2 \times T_2 \Rightarrow$ Time taken to access M_2 if its found in M_2.

$((1-H_1)(1-H_2)) \times T_3 \Rightarrow$ Time taken to access M_3 if the information is found in M_3.

Here all the memory units are connected simultaneously. So, the ~~total~~ Average Memory Access Time (T_{avg})

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

<u>Case-II</u>



Averge Memory Access Time (T_{avg}):

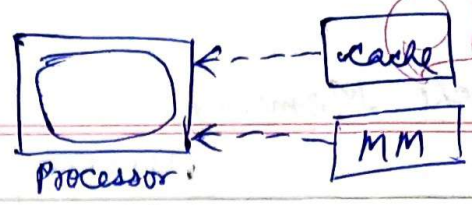
$$= H_1 T_1 + ((1-H_1) \times H_2)(T_1 + T_2) + ((1-H_1)(1-H_2))(T_1 + T_2 + T_3)$$

Q. # A cache memory needs an access time of 30 ns and main memory 150 ns. What is the average access time of the CPU? (Assume 80% hit ratio)

Scanned with Any Scanner

classmate
Date
Page

Case-I



Soln Access time

$T_{cache} = 30\ ns$ Hit ratio

$T_{MM} = 150\ ns$ $H_{cache} = 80\% = 0.8$

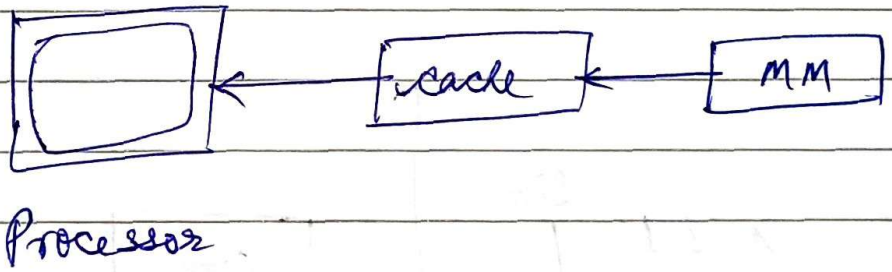
$$T_{avg} = H_{cache} T_{cache} + (1 - H_{cache}) \times T_{MM}$$

$$= 0.8 \times 30 + (1 - 0.8) \times 150\ ns$$

$$= 24 + 0.2 \times 150\ ns$$

$$= 54\ ns$$

Case-II



$$T_{avg} = H_{cache} \times T_{cache} + (1 - H_{cache})(T_{cache} + T_{MM})$$

$$= 0.8 \times 30 + (1 - 0.8)(30 + 150)\ ns$$

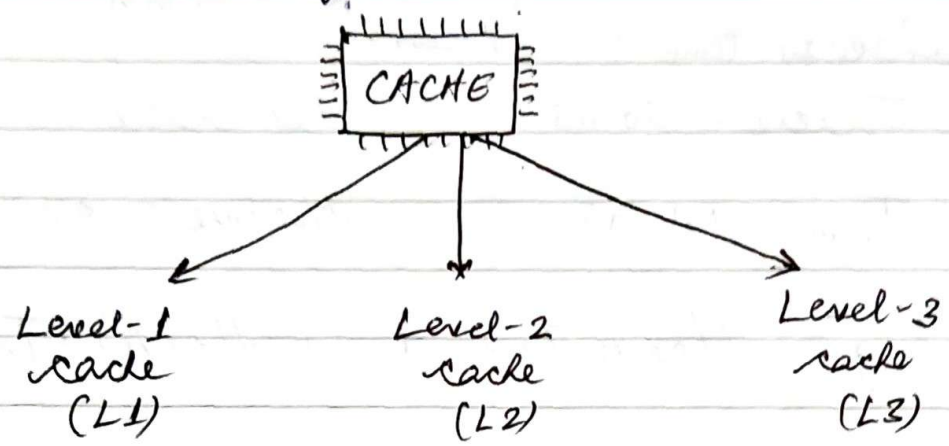
$$= 24 + 0.2 \times 180\ ns$$

$$= 24 + 36\ ns$$

$$= 60\ ns.$$

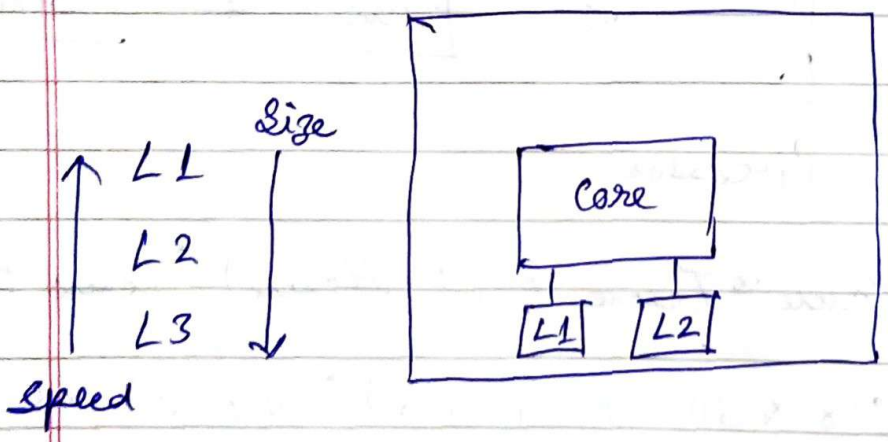
Scanned with Any Scanner

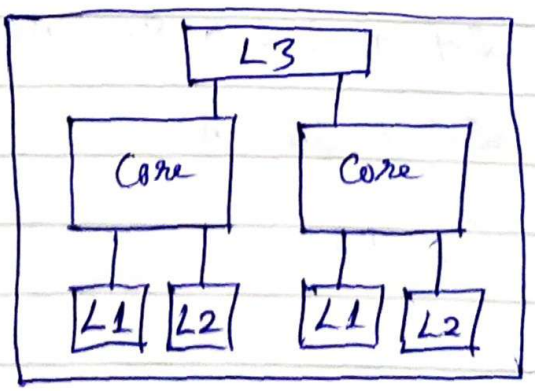
Cache Memory



Dual core
Quad core
Octa core

MIMD
(Flynn's Taxonomy)





Scanned with Any Scanner

Cache Related Terms

- 'Cache Hit' (if data found in cache memory)
- Time required by this process is called 'Hit Latency'.
- Tag Directory (Data structure) — using DS the processor finds data in the cache memory.

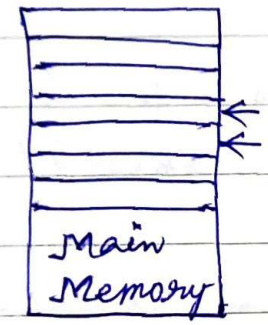
- if the information is not found in cache, it is called 'cache Miss.'
- then the processor looks into the Main memory for the information and copies it into the cache from the Main Memory.
- the entire period of time is called 'Miss Latency'.

- if the information is also absent in the Main Memory, the situation is called 'Page Fault'. and if information is found, it is called 'Page Hit'.
- During 'Page Fault' the processor looks for the information in the last level of the hierarchy, i.e. in the secondary memory, and brings it to the Main Memory.
- this entire process is called 'Page Fault Service' and the time taken is called "Page Fault Service Time".
- the information with highest frequency are kept in the cache. This is done through "Locality of Reference".
-

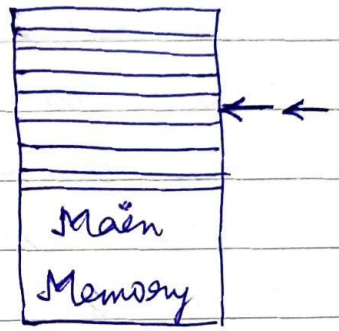
Scanned with Any Scanner

Locality of Reference

Spatial Locality



Temporal Locality.



Scanned with Any Scanner

Primary Memory

ROM	RAM
- stores Firmware	(analogy ⇒ use of plate while eating, then washing it)
- non-volatile	- volatile

Firmwares are the set of instructions / software that are hardwared or permanently stored in the ROM.

Primary Memory - ROM

B - Basic
I - Input
O - Output
S - System

The BIOS Firmware is loaded in a ROM chip, called the BIOS chip, and it resides in the Motherboard. The BIOS chip contains the Bootstrap Program.

- When the computer is turned on the 'POST' process is executed. [Power On Self Test] and 'POST' screen (black screen) is displayed.
- At this time the following tasks are done —

1. BIOS chip is checked.
2. CMOS chip is checked. CMOS is a volatile chip, that's why it is connected with a CMOS Battery. In CMOS chip the BIOS setting is maintained.

- Removing the CMOS battery resets the BIOS set up.

Scanned with Any Scanner

- When the BIOS and CMOS chip are found to be ok, there after the processor is initialized.
- Next, all the associated hardware components are tested (like keyboard, mouse, RAM module etc) and if they are found ok, then after a beep sound further process execution is moved.
- Otherwise some beep codes are generated depending on the fault found during checking.
- After the completion of POST program execution, the BIOS instructions stored in the Bootstrap program looks for the valid Boot Disk where the Operating system is permanently stored.
- Next, the OS is loaded into the RAM and the Desktop is loaded on the screen.
- All these are programmed in the BIOS chip, which is a ROM chip.

Classification of ROM:

① PROM - Programmable ROM
- it is programmed using IC burners.
- PROM can be programmed only once

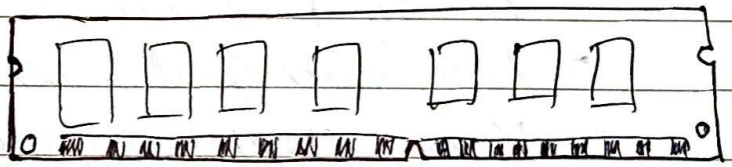
② EPROM - Erasable Programmable ROM.
- these can be re written many times.

Scanned with Any Scanner

- these are erased using UV rays.

③ EEPROM :- Electrically Erasable Programmable ROM.
- these are erased electrically.
- these can be erased and re-written thousands of times.
- flash memory is an example of EEPROM which can be re-written millions of times.

RAM : it is a volatile temporary storage.



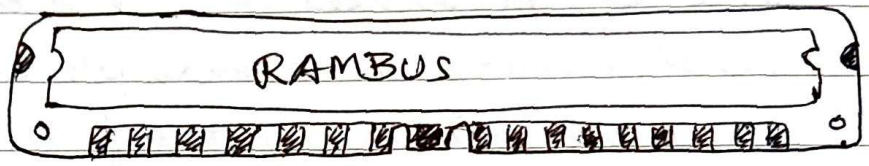
Two rows of pins

- these are called Dual Inline Memory Module (DIMM)
- no. of pins in RAM can be 168, 184, 240 or 288.
- modern motherboards have 2 to 4 RAM slots.
- due to periodic recharging these are called DRAM.
- DRAMs are asynchronous
- SDRAMs (Synchronous Dynamic RAM) are synchronous and run in synchronism with the system clock. As a result these are faster than DRAMs.
- The storage capacity of DIMM SDRAMs range from 128 MB to 32 GB and they can transfer 64 bits in one clock cycle or 8 bytes per clock cycle.
- DIMMs have 8 Byte wide Data path.

Scanned with Any Scanner

RDRAM : Invented in 1999.

- RIMMs or Rambus Inline Memory Module - RDRAM
- RDRAM → Rambus Dynamic RAM.
- these were invented by Rambus Incorporation.



↑ ↑
Two notches

- no. of pins in RIMM RDRAM is 184.
- Very much faster than DIMMs.
- Maxm speed of DIMMs is 133 MHz whereas the maxm speed of RIMMs is 800 MHz.
- RIMMs have only 2 Bytes wide Datapaths.
- Bandwidth of RIMMs : 800 MHz × 2 Bytes
  = 1600 MB/s
- Bandwidth of DIMMs = 133 MHz × 8 Bytes
  = 1064 MB/s.

DDR-RAM:

Double Data Rate RAM

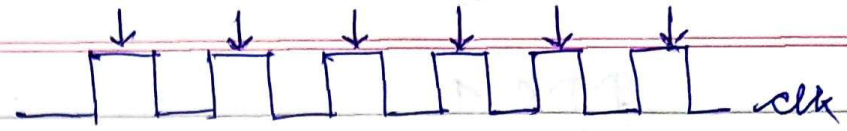
Due to the small Data path the RIMM-RDRAMs are replaced by DDR-RAM

- these can transfer double amount of data in 1 clock cycle.

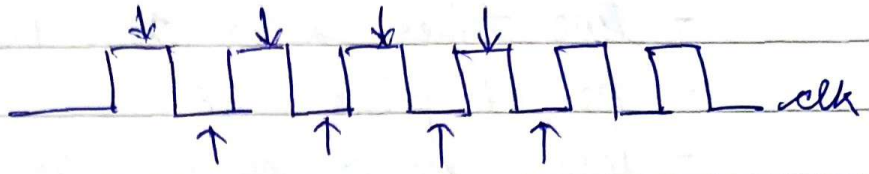
Scanned with Any Scanner

classmate
Date
Page

NON-DDR RAM :



DDR RAM :



- DDR-RAM uses both the positive and negative level of the clock signal for data transfer, thus making it double.
- DDR-RAM includes both Clock Speed and Bandwidth in the label
- Total no. of pins in DDR-RAM is 184.

Bandwidth = 333 MHz × 8 Bytes. (DIMM-SDRAM)
clock speed. → DDR

= 2700 MB/s.

DDR-2 :

- faster than DDR
- total no. of pins in DDR-2 RAM is 240.

Bandwidth = 800 MHz × 8 Bytes

= 6400 MB/s.

DDR-3 :

- two times faster than DDR-2
- no. of pins is 240
- lesser power consumption than DDR-2

Bandwidth = 1600 MHz × 8 Bytes

= 12800 MB/s

Scanned with Any Scanner

DDR-4 :

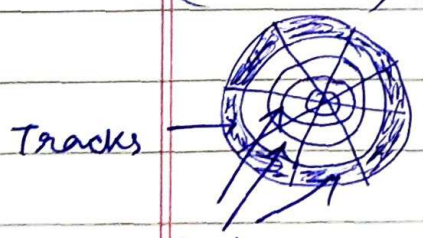
- two times faster than DDR-3
- no. of pins 288.
- lesser power consumption than DDR-3.

Bandwidth = 4266 MHz × 8 Bytes (clock speed.)
= 34128 MB/s.

Secondary Memory

Removable Auxiliary Storage

- Magnetic tapes
- Floppy Disk (3.5 inch 1.2 to 1.44 MB data)
- optical Disk (12 cm dia) — CD (700 MB) — DVD (4.7 GB) — BRD. (Single layer 25 – 33.4 GB, Quad ~~Double~~ layer upto 128 GB)



- Flash Memory. (EEPROM)
  - SD Card (Secure Digital)
  - Mini SD Card
  - Micro SD Card
- Flash Drives (with USB interface)
  - 4 GB to 256 GB. life = 10 – 100 years

Fixed (HDD) Auxiliary Storage

- stack of magnetic disks.
- divided into tracks and sectors.
- each sector size = 512 Bytes
- capacity of one disk = no. of tracks × no. of sectors per track × no. of Bytes per sector.

** Recording density of the smaller tracks is higher than the larger tracks, thus the track capacity is same for all the tracks.

Scanned with Any Scanner

HDD types : 5400 rpm (used in laptops) 7200 rpm (used in desktops) 10,000 rpm (used in high end computers)

HDD is connected to the motherboard through SATA cable

HDD: Hard Disk Drive (SATA: Serial Advanced Technology Attachment)

Average Access Time of the HDD,

$$T_{avg} = (T_S + T_R + T_{data\ transfer}).$$

where, T_S = Seek Time = time taken to move the read/write head from one track to another

T_R = Rotational Delay = time taken to rotate the disk and align the sector under the read/write head.

In one rotational delay, T_R the read/write head can read the entire track i.e. all sectors within that track.

Format Overhead ⇒ it is the memory / sector used (per sector) to store diff. information about tracks & sectors.

Consider a HDD with the following specifications:

1. 4 Surfaces
2. 512 tracks per Surface.
3. 1024 sectors per track
4. Size of each sector = 512 Bytes.

(a) Find out the capacity of the HDD.

(b) If Formal overhead is 8 Bytes per sector then find out the formatted capacity of the HDD.

(c) Find the number of bits required to address each sector.

Scanned with Any Scanner

Soln. (a) Capacity = no. of surfaces × no. of tracks/surface × no. of sectors/Track × Sector size

$= 4 \times 512 \times 1024 \times 512$ Bytes

$= 2^2 \times 2^9 \times 2^{10} \times 2^9$ Bytes

$= 2^{30}$ Bytes

$= 1$ GB.

(b) Total capacity = 1 GB = 1024 MB.

Total no. of sectors = no. of surfaces × no. of tracks per sector × no. of sectors per track

$= 4 \times 512 \times 1024$

$= 2^2 \times 2^9 \times 2^{10}$

$= 2^{21}$ sectors.

Formatting Overhead $= 2^{21} \times 8$ Bytes

$= 2^{21} \times 2^3$ Bytes

$= 2^{24}$ Bytes

$= 2^4 \times 2^{20}$

$= 16$ MB.

∴ Formatted capacity $= (1024 - 16)$ MB.

$= 1008$ MB.

Scanned with Any Scanner

(c) Total no. of sectors = no. of surfaces × no. of tracks/sector × no. of sectors/track.

$$= 2^2 \times 2^9 \times 2^{10} = 2^{21}.$$

∴ No. of bits required to address each sector $= \log_2 2^{21} = 21.$

So, with 21 bits we can address all the sectors in each of the surfaces.

#1) Find the value of x, y and z in $(11)_2 + (22)_3 + (33)_4 + (44)_5 = (xyz)_6$.

#2) $(56)_8 = (?)_5$

#3) $(215)_8 = (?)_{16}$

"Find frequently used areas and non-frequently used areas and add enhancement to the frequently used area." — Amdahl's law.

Focus on the common case:

- When adding two numbers in the processor, we can expect overflow to be a rare case and can therefore improve performance by optimizing the more common case of no overflow.
- This emphasis may slow down the case when overflow occurs, but if that is rare, then overall performance will be improved by optimizing for the normal case.
- We have to decide what the frequent case is and how much performance can be improved by making the case faster.
- Amdahl's law is used to quantify this principle.

Scanned with Any Scanner

Quantitative Techniques in Computer Design

Topics : 1. CPU Performance Equation.
2. Clock Cycle Time : Hardware & Organization.
3. CPI : Organization and Instruction set architecture.
4. Instruction Count : Instruction set architecture and Compiler Technology.

The most common and pervasive principle of computer design is to make the common case fast. In applying this principle we have to decide what the frequent case is and how much performance can be improved by making the case faster.
A fundamental law, called Amdahl's law can be used to quantify this principle.

Amdahl's law

The performance gain that can be obtained by improving some portion of a computer can be calculated using Amdahl's law.

"Amdahl's law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used."

Scanned with Any Scanner

"Focus more on the common case" is the principle.

Amdahl's law defines the Speedup that can be gained by using a particular feature.

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

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

$$\text{Speedup} = \frac{\text{Serial Execution time}}{\text{Parallel Execution time}}$$

$$= \frac{\text{Time (original)}}{\text{Time (enhanced)}}$$

$$= \frac{\text{Performance (enhanced)}}{\text{Performance (Original)}}$$

Ex. # If the serial application execution in 6720 sec and a corresponding parallel application runs in 126.7 sec (using 64 threads and cores), Find Speedup of parallel application.

Soln. $$\text{Speedup} = \frac{\text{Serial execution time}}{\text{Parallel execution time}}$$

$$= \frac{6720}{126.7} = 53.038 = 53\times$$

Scanned with Any Scanner

Date
Page

Speedup tells us how much faster a task will run using the machine with the enhancement as opposed to the original machine.

Amdahl's law gives us a quick way to find the speedup from some enhancement, which depend on two factors:

(i) The fraction of the computation time in the original machine that can be converted to take advantage of the enhancement.

e.g. If 20 seconds of the execution time of a program that takes 60 seconds in total can use an enhancement, the fraction is 20/60. This value is called Fraction(enhancement) and is always less than or equal to 1.

(ii) The improvement gained by the enhanced execution mode, that is how much faster the task would run if the enhanced mode were used for the entire program - this value is the time of the original mode over the time of the enhanced mode.

Scanned with Any Scanner

Ex. # If the enhanced mode takes 2 seconds for some portion of the program that can completely use the mode, while the original mode took 5 seconds for the same portion, the improvement is 5/2.
This is called Speedup(enhanced) and is always greater than 1.

CPU or Processor Performance Equation:

- Essentially all the computers are constructed using a clock, running at a constant rate. These discrete time events are called ticks or clock periods, or clocks or cycles or clock cycles.
- Computer ~~programm~~ designers refer to the time of a clock period by its duration, ie. 1 ns or by its rate/frequency eg. 1 GHz.

CPU time for a program can be expressed as:

$$\text{CPU time} = \text{CPU clock cycle per program} \times \text{clock cycle time}$$

$$= \frac{\text{CPU clock cycle for a program}}{\text{clock rate}}$$

In addition to the number of clock cycles needed to execute a program, we can also count the

CPU time = 100 × 1 ns = 100 ns.

Scanned with Any Scanner

One prog. → | Instr-1 | Instr-2 | Instr-3 |

clock cycles → n_1 n_2 n_3

Instruction count = 3

CPU clock cycles per prog = $(n_1 + n_2 + n_3)$.

number of instructions executed, the instruction path length or the instruction count (IC). If we know the number of clock cycles and the instruction count, we can calculate the average number of clock cycles per instructions (CPI).

$$CPI = \frac{\text{CPU clock cycles per Program}}{\text{Instruction Count}}$$

or, CPU clock cycles for a program = CPI × Instruction Count.

∴ CPU time = Instruction count × ~~clock cycle~~ ~~time~~ Cycles per Instructions (CPI) × clock cycle time.

Clock Cycle Time:

- Clock cycle is the speed of a computer processor or CPU. It is the time between two pulses of an oscillator.
- higher the number of clock cycles per second faster the computer processor.

Clock period or cycle time (T_c)

- It is the time between the rising edges of a repititive clock signal.

Scanned with Any Scanner

$$\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{clock cycles}}{\text{Instruction}} \times \frac{\text{seconds}}{\text{clock cycle}} = \frac{\text{seconds}}{\text{Program}}$$

Clock Frequency (f_c)

- It is the reciprocal of clock period.

$$f_c = \frac{1}{T_c}$$

~~Average Clock Cycles per Instruction,~~ Number of total processor clock cycles :

$$CPI_{(avg)} = \frac{\sum_{i=1}^{n} (IC_i)(CPI_i)}{\text{clock cycles}}$$

where, IC_i = no. of times instructions i is executed ~~for a given instruction type i.~~ in a prog.

CPI_i = ~~clock cycles~~ average number of clocks per instruction for instruction i.

Instruction Count :

- The total number of instructions that get executed for a particular task, algorithm, workload, or program is referred to as the instruction count.
- The instruction count forms the basis for various performance aspects of the processor, such as Instructions per cycle (IPC) or Cycles per Instruction (CPI).

Scanned with Any Scanner

Amdahl's law

- The execution time using the original computer with the enhanced mode will be the time spent using the unenhanced portion of the computer plus the time spent using the enhancement:

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

- The overall Speedup is the ratio of the execution times:

$$\text{Speedup}_{overall} = \frac{\text{Execution time}_{old}}{\text{Execution time}_{new}}$$

$$= \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

Amdahl's law defines the term Speedup:

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

Scanned with Any Scanner

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

Speedup (enhanced)

Fraction (enhanced)

$$Fraction_{enhanced} = 40\% = 0.4$$

$$Speedup_{enhanced} = 10$$

$$Speedup_{overall} = \frac{1}{(1-0.4) + \frac{0.4}{10}}$$

$$= \frac{1}{0.6 + 0.04} = \frac{1}{0.64} \approx 1.56.$$

So,

- Amdahl's law is particularly useful for comparing the overall system performance of two alternatives.
- It can also be applied to compare two processor design alternatives.

Scanned with Any Scanner

Ex. # A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics. Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQR hardware and speedup this operation by a factor of 10. The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6. FP instructions are responsible for half of the execution time for the application. The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root. Compare these two design alternatives.

We can compare these two alternatives by comparing the speedups:

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

where, $\text{Fraction}_{enhanced} = 0.2$; $\text{Speedup}_{enhanced} = 10$.

$$Speedup_{FP} = \frac{1}{(1-0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8125} \approx 1.23.$$

where $Fraction_{enhanced} = 0.5$, $Speedup_{enhanced} = 1.6$.
Hence, we find that improving the performance of FP operations overall is slightly better because of the higher frequency.

The Processor Performance Equation

- Processor performance and CPU time depends on clock cycle, CPI and IC.
- A 10% improvement in anyone of these leads to a 10% improvement in CPU time.
- Clock cycle depends on hardware technology and organization.
- CPI depends on organization and instruction set architecture.
- IC depends on instruction set architecture and compiler technology.

The number of total processor clock cycles can be calculated as:

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

IC_i = no. of times instruction i is executed in a program.
CPI_i = averge no. of clocks per instruction for instruction i.

Scanned with Any Scanner

CPU time is given as:

$$\Rightarrow \text{CPU time} = \left(\sum_{i=1}^{n} IC_i \times CPI_i\right) \times \text{clock cycle time}$$

CPU time for all the instructions

$\Rightarrow$ Overall CPI is given as:

$$CPI = \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{\text{Instruction Count}} = \sum_{i=1}^{n} \frac{IC_i}{\text{Instruction count}} \times CPI_i$$

Ex #1. What is the overall Speedup if you make 10% of a program 90 times faster?

#2. What is the overall Speedup if you make 90% of a program 10 times faster?

Soln. Amdahl's law:

$$\text{Overall Speedup} = \frac{1}{(1-f) + \frac{f}{s}}$$

1. $\text{Overall Speedup}_{\text{Case-I}} = \frac{1}{(1-0.1) + \frac{0.1}{90}} = \frac{1}{0.9011} \approx 1.11$

2. $\text{Overall Speedup}_{\text{Case-II}} = \frac{1}{(1-0.9) + \frac{0.9}{10}} = \frac{1}{0.19} \approx 5.26$

Scanned with Any Scanner

#3. Here an enhancement to the processor of a Web server is considered. The new CPU is 20 times faster on search queries than the old processor. The old processor is busy with search queries 70% of the time. What is the Speedup gained by integrating the enhanced CPU?

Soln:

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

Here, $\text{Fraction}_{enhanced} = 70\% = 0.7$.

$\text{Speedup}_{enhanced} = 20$.

$$\therefore \text{Speedup} = \frac{1}{(1-0.7) + \frac{0.7}{20}} = \frac{1}{0.335} \approx 2.985$$

That is, the enhanced CPU will be 2.985 faster than the old CPU.

Here an enhancement to the processor of a server is considered. The new CPU is 10x faster. The I/O bound server has 60% writing time for I/O. Compare the old and new CPU.

Scanned with Any Scanner

Soln: Speedup = 10, 40% time is processing time.

$$\text{Fraction}_{\text{enhanced}} = (1-0.6) = 0.4.$$

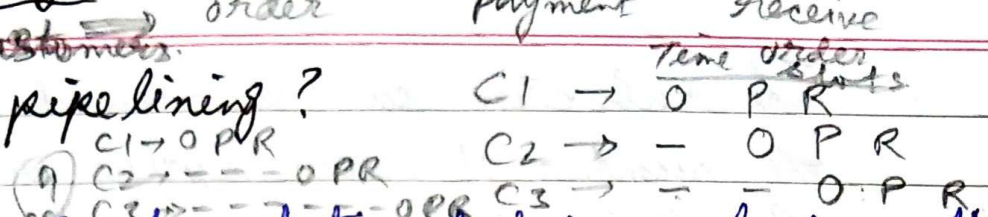
$$\therefore \text{Speedup}_{\text{enhanced}} = \frac{1}{(1-0.4) + \frac{0.4}{10}}$$

$$= \frac{1}{0.64} \approx 1.56$$

Scanned with Any Scanner

Pipelining

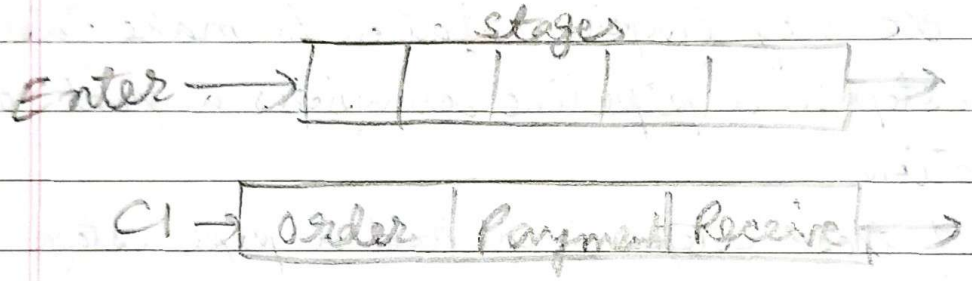
What is pipelining?



- It is an implementation technique whereby multiple instructions are overlapped in execution. 5 clock cycles.
- It takes advantage of parallelism.
- It is the key implementation to make fast CPUs.
- Each step in a pipeline completes a part of an instruction
- Each of these steps is called a pipe stage or a pipe segment.
- The stages are connected one to the next to form a pipe.
- Instructions enter at one end, progress through stages and exit at the other end.
- The throughput of an instruction pipeline is determined by how often an instruction exits the pipeline.
- Because all stages proceed at the same time, the length of a processor cycle is determined by the time required for the slowest pipe stage.
- If the stages are perfectly balanced, then the time per instruction on the pipelined processor is equal to: $\frac{\text{Time per instruction on unpipelined machine}}{\text{Number of pipe stages}}$
- Pipeline yields a reduction in the average execution time per instruction.

Scanned with Any Scanner

- The reduction can be viewed as decreasing the number of clock cycles per instruction.
- Pipelining is an implementation technique that exploits parallelism among the instructions in a sequential instruction stream.



		complete task in
non-pipelined system	C1 → O P R	3 min
	C2 → - - - O P R	6 min
	C3 → - - - - - - O P R	9 min
	Total 9 mins	avg time $\frac{9}{3} = 3$ min

		complete task in
pipelined system	C1 → O P R	3
	C2 → - O P R	4
	C3 → - - O P R	5
	Total 5 mins	avg time $= \frac{5}{3} = 1.66$ min

Scanned with Any Scanner

Computer architecture can be classified into two types based on the instruction sets.

① CISC → Complex Instruction Set computer.

② RISC → Reduced Instruction Set Computer.

	CISC	RISC
1.	Complex and large number of instructions.	Simple and reduced number of instructions.
2.	Variable length instruction format	Fixed length instruction format.
3.	Large no. of addressing modes	Few no. of add. modes.
4.	Cost is high	Cost is low
5.	More powerful	Less powerful
6.	Multiple clock cycle instructions	Single clock cycle instructions.
7.	Manipulation is done directly in memory	Manipulation is done in registers.
8.	Microprogrammed control unit	Harwired control unit.
9.	Uses more RAM and less Registers	Uses more Registers and less RAM.
10.	Found in PCs & Servers	Preferred in mobile devices, IOT and consumer applications.
11.	Examples: Motorola 68K, X86, System/360, VAX. etc.	Examples: Sun's SPARC, Power PC, MIPS, AVR, ARM.

Scanned with Any Scanner

The basics of RISC Instruction Set

Here MIPS is used as the default RISC architecture.

Characteristics of RISC architecture:

- All operations on data apply to data in registers and typically change the entire register (content).
- The only operations that affect memory are LOAD and STORE operations that move data from memory to a register or to memory from a register, respectively.
- The instruction formats are a few in number, with all instructions typically being of same size.
- MIPS instruction set provides 32 registers.
- There are three classes of instructions:

(i) ALU instructions: instructions that take either two registers or a register and an immediate value and store the result into a third register.

(ii) LOAD and STORE instructions: These instructions take a register source, called the base register and an immediate field, called the operand. The sum is stored in a memory address.

(iii) Branches and jumps:

Scanned with Any Scanner

A Simple Implementation of a RISC Instruction Set

- Considering the implementation without pipelining.
- every instruction takes at most 5 clock cycles.
- the 5 clock cycles are —
  - (i) Instruction Fetch cycle (IF)
  - (ii) Instruction Decode/register fetch cycle (ID)
  - (iii) Execution / Effective address cycle (EX)
  - (iv) Memory Access (MEM)
  - (v) Write Back (WB)

5 stage RISC Pipeline of MIPS Processor.

- We can pipeline the execution by simply starting a new instruction on each clock cycle.

	Clock Number								
Instruction No.	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB.

$k = 5$, $n = 5$

No. of stages $= k$.

No. of instructions $= n$

To complete the 1st instruction clock cycle required $= 1 \times k$

" " the rest of the " " " " $= (n-1)$

$\therefore$ Total clock cycles required $= k + (n-1) = 5 + (5-1)$
$= 09$.

Scanned with Any Scanner

Total clock cycles required without pipelining
$= 5 \times 5 = 25$

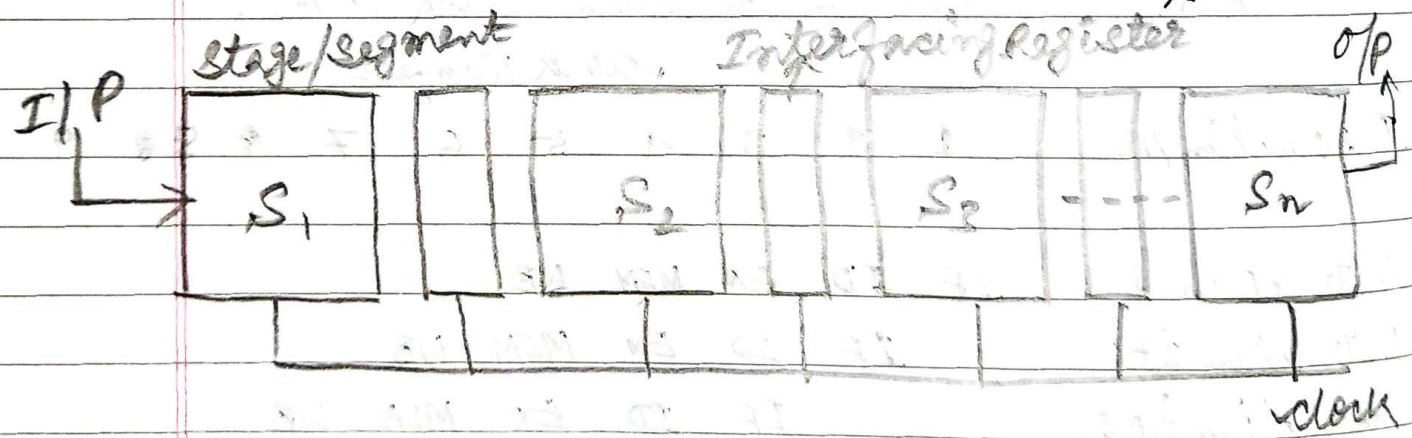
Total clock cycles required using pipelining
$= 9$

$\therefore$ Speedup $= \frac{25}{9} \approx 2.78$

<u>Efficiency / Utilization</u> $= \frac{\text{No. of boxes used}}{\text{No. of boxes}}$

$= \frac{25}{45} = \frac{5}{9}$

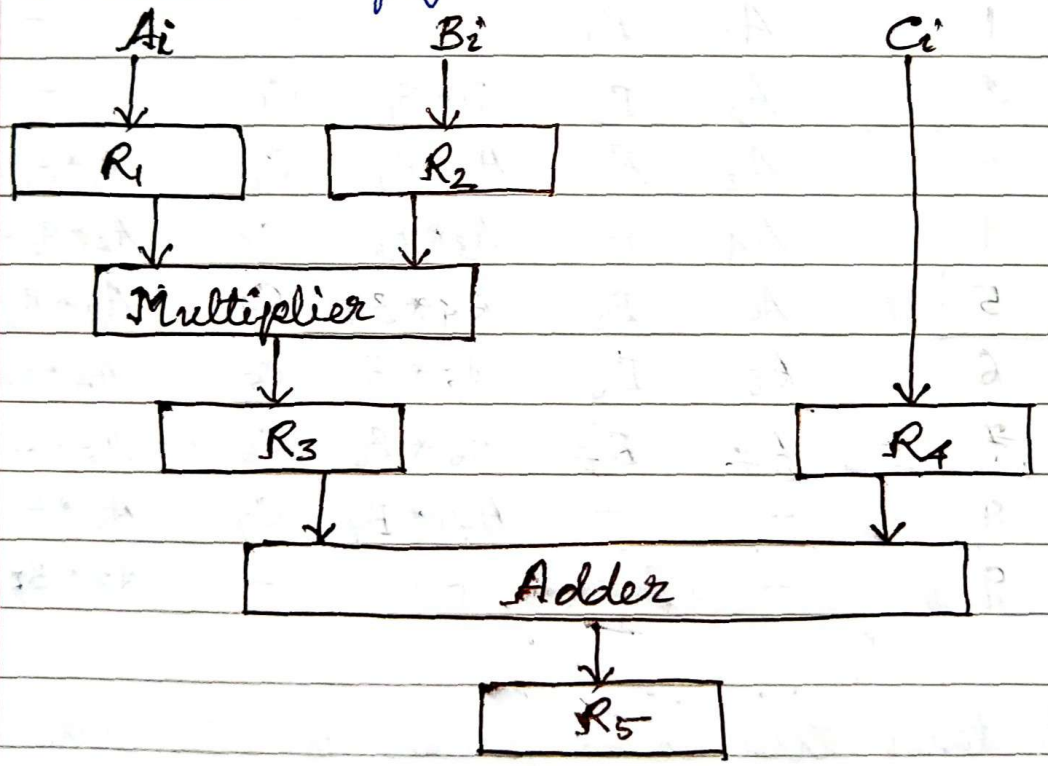
$= 0.55$ or 55%.



Suppose, we want to perform the combined multiply and add operations with a stream of numbers.

$$A_i * B_i + C_i \quad \text{for } i = 1, 2, 3, \ldots 7.$$

Each suboperation is to be implemented in a segment within a pipeline. Each segment has one or two registers and a combinational circuit as shown in fig.



R_1 through R_5 are registers that receive new data with every clock pulse. The multiplier and adder are combinational circuits. The suboperations performed in each segment of the pipeline are as follows:

Scanned with Any Scanner

$R_1 \leftarrow A_i$, $R_2 \leftarrow B_i$ Input A_i and B_i

$R_3 \leftarrow R_1 * R_2$, $R_4 \leftarrow C_i$ Multiply and input C_i

$R_5 \leftarrow R_3 + R_4$ Add C_i to the product.

Content of Registers in Pipeline Example.

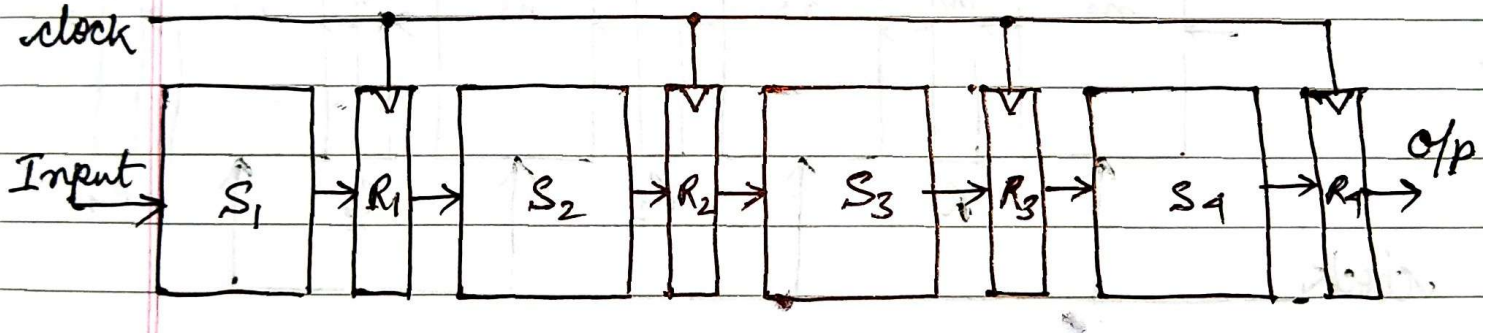
clock pulse Number	Segment-1		Segment-2		Segment-3
	R_1	R_2	R_3	R_4	R_5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

It takes three clock pulses to fill up the pipe and retrieve the first output from R_5. From there on, each clock produces a new output and moves the data one step down the pipeline. This happens as long as new input data flow into the system. When no more input data are available, the clock must continue until the last output emerges out of the pipeline.

Scanned with Any Scanner

Any operation that can be decomposed into a sequence of suboperations of about the same complexity can be implemented by a pipeline processor. The technique is efficient for those applications that need to repeat the same task many times with different set of data.

The general structure of a four segment pipeline is illustrated in the fig below.



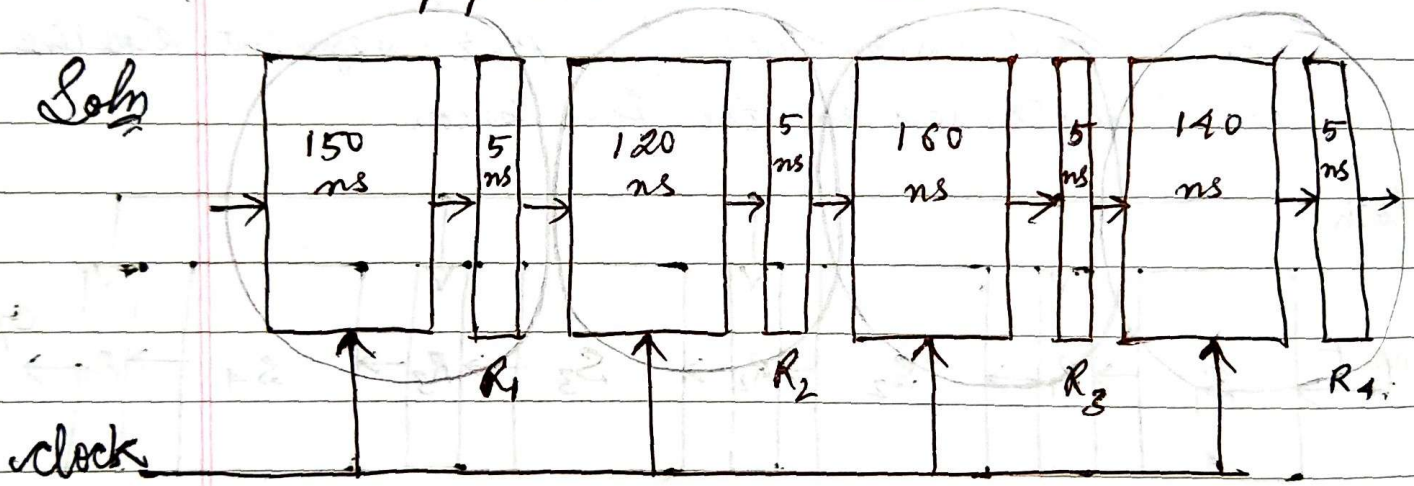
Four Segment Pipeline

The operands pass through all four segments in a fixed sequence. Each segment consists of a combinational circuit S_i that performs a suboperation over the data stream flowing through the pipe.

Scanned with Any Scanner

A four stage pipeline has stage delays as 150 ns, 120 ns, 160 ns and 140 ns. Registers are used between stages and have a delay of 5 ns each. Assuming constant clock rate, the total time taken to process 1000 ~~items~~ data items on this pipeline will be _____.

Soln



$T = (160+5)\,ns = 165\,ns$ [maxm. value]

$\therefore f = \frac{1}{T} = \frac{1}{165\,ns} = \frac{1}{165 \times 10^{-9}} =$

Total time taken $= 1 \times 4 \times 165\,ns + 999 \times 1 \times 165\,ns.$

$= 165495\,ns$

$= 165495 \times 10^{-9}\,s$

$= \frac{165495}{1000} \times 10^{-6}\,s$

$= 165.495\,\mu s.$

Scanned with Any Scanner

Consider a non-pipelined processor with a clock rate of 2.5 GHz and average cycles/instruction of four. The same processor is upgraded to a pipelined processor with five stages, but due to internal pipeline delay, the clock speed is reduced to 2 GHz. Assume that there is no stall in the pipeline. Calculate the Speedup achieved in the pipeline processor.

Soln: $\text{Speedup} = \frac{T_{NP}}{T_P}$; T_{NP} = Time period without pipeline

T_P = Time with pipeline

Non-pipelined processor:

$f = 2.5\,GHz \Rightarrow T = \frac{1}{f} = \frac{1}{2.5 \times 10^9}$ sec.

$\therefore T_{NP} = 4 \times \frac{1}{2.5 \times 10^9}$ sec.

Pipelined processor:

$f = 2.0\,GHz \Rightarrow f = \frac{1}{T} = \frac{1}{2.0 \times 10^9}$ sec

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

$$\therefore \text{Speedup} = \frac{T_{NP}}{T_P} = \frac{4 \times \frac{1}{2.5 \times 10^9}\ \text{sec}}{\frac{1}{2.0 \times 10^9}\ \text{sec}} = \frac{8}{2.5} = 3.2$$

Scanned with Any Scanner

Hazards in Pipelining:

Pipeline Hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycles.

** Hazards reduce the performance from the ideal speedup gained by pipelining

Any condition that causes a stall in the pipeline operations can be called a Hazard.

Pipeline Hazards
- Data Hazard
- Structural Hazard
- Control/ Instruction Hazard.

Data Hazards:

Data Hazards occur when data is used before it is ready.

OR

When the read and write of the shared variables by different instructions in a pipeline may lead to different results if these instructions are executed out of order.

Type of Data Hazards
- RAW (Read After Write) Hazards (TD → True Dependency)
- WAW (Write after Write) Hazards (OD → O/P Dependency)
- WAR (Write after read) Hazards. (AD → Anti Dep.)

Scanned with Any Scanner

- Ideal CPI for pipelining is 1.

Example of Data Hazard :- (RAW)

I_1 : $R_2 \leftarrow R_2 + R_3$

I_2 : $R_5 \leftarrow R_2 + R_4$

	CC-1	CC-2	CC-3	CC-4	CC-5	CC-6
I_1 :	IF/D	OF	EX	Mem	WB	
I_2 :		IF/D	OF	EX	Mem	WB

Final value of R_2 is available.

Fetching value of R_2 (Garbage Value)

Example of Structural Hazard :-

	C-1	C-2	C-3	C-4	C-5	C-6
I_1 :	IF/D	OF	EX	Mem	WB	
I_2 :		IF/D	OF	EX	Mem	WB

Same memory is accessed simultaneously which causes structural Hazard.

Example of Control Hazard :- It occurs due to branch conditions.

PC Add.	Instr.
500	I_1
501	I_2 (branch Instr.)
502	
700	

PC points to next instr.

CC-1	CC-2	CC-3	CC-4	CC-5
IF	ID	OF	EX	
	IF	ID	OF	EX
		IF	ID	OF	EX
			IF	ID	OF	EX
				700		

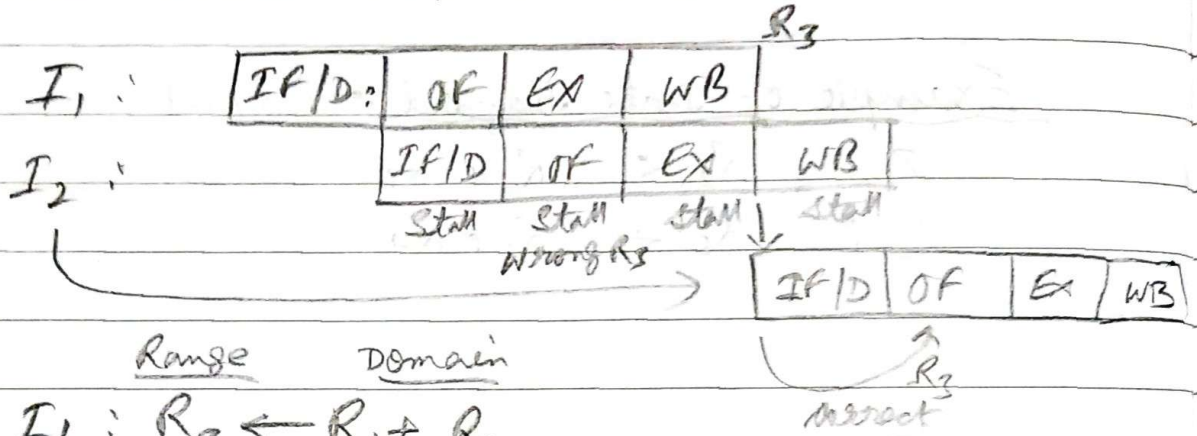
mem. Addr. 700 to be executed.

Scanned with Any Scanner

Data Hazard

most common Hazard.

(i) RAW (Read After Write) Hazard. (True Data Dependency)



	Range	Domain
I_1 :	$R_3 \leftarrow R_1 + R_2$	
I_2 :	$R_5 \leftarrow R_3 + R_4$	

	Domain	Range
I_1 :	R_1, R_2	R_3
I_2 :	R_3, R_4	R_5

if $I_1(\text{Range}) \cap I_2(\text{Domain}) \neq \phi \Rightarrow$ RAW Hazard.

null

$R_3 \qquad R_3 \neq \phi \Rightarrow$ RAW

Therefore, if the actual execution order of I_1 and I_2 instructions violate the program order, incorrect results are read or written, thereby producing Hazard.

Therefore, Hazard should be prevented before these instructions enter the pipeline, such as by holding instruction I_2 until the dependence on instruction I_1 is resolved.

Scanned with Any Scanner

(Anti Data Dependency)

(ii) <u>WAR (Write After Read) Hazard</u>: WAR Hazard occurs when I_2 attempts to modify some data object read by I_1.

I_1 : $R_1 \leftarrow R_2 * R_3$

I_2 : $R_2 \leftarrow R_4 + R_5$

WAR Hazard occurs when instruction I_2 tries to write data before instruction I_1 reads it.
Here, I_2 is trying to write R_2 before I_1 have read it.

	Domain	Range
I_1:	R_2, R_3	R_1
I_2:	R_4, R_5	R_2

if $\underset{R_2, R_3}{\text{Domain}(I_1)} \cap \underset{R_2}{\text{Range}(I_2)} \neq \phi \Rightarrow$ WAR Hazard.

(iii) <u>WAW (Write After Write) Hazard</u>:
(Output Data Dependency)
WAW Hazard may occur if both instructions I_1 and I_2 attempt to modify the same data object.

I_1 : $\overset{200}{R_3} \leftarrow \overset{10}{R_1} * \overset{20}{R_2}$ Domain$(I_1) = R_1, R_2$

I_2 : $\underset{100}{R_3} \leftarrow \underset{60}{R_4} + \underset{10}{R_5}$ Domain$(I_2) = R_4, R_5$

Range$(I_1) = R_3$

Range$(I_2) = R_3$

Write operations : ~~200~~ 100 ✓

~~100~~ 200 ✗

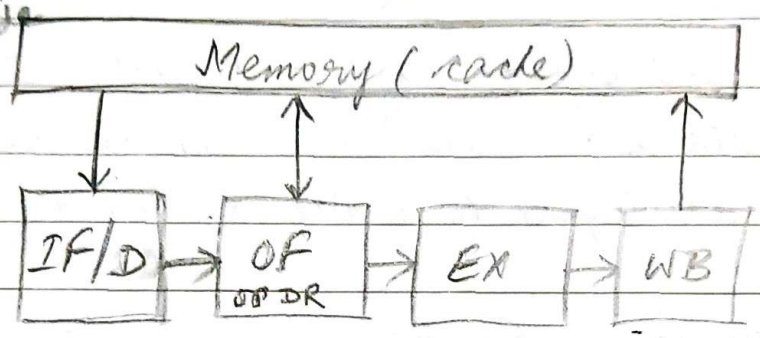
** if $\underset{R_3}{\text{Range}(I_1)} \cap \underset{R_3}{\text{Range}(I_2)} \neq \phi \Rightarrow$ WAW Hazard.

Scanned with Any Scanner

Structural Hazards

Structural Hazard arises when two instructions require a given hardware resource at the same time and hence, for one of the instructions the pipeline needs to be stalled.

IF/D = Instr. Fetch & Decode
OF = Operand Fetch
DR = Data read



Instr. \ cc	1	2	3	4	5	6	7	8	9	10
I_1	IF/D	OF	EX	WB						
I_2		IF/D	OF	EX	WB					
I_3			IF/D	OF	EX	WB				
I_4				IF/D	OF	EX	WB			

Instr. \ cc	1	2	3	4	5	6	7	8	9	10
I_1	IF/D	OF	EX	WB						
I_2		X	IF/D	X	OF	EX	WB			
I_3		stall		X	X	IF/D	X	OF	EX	WB
I_4										

Structural Hazard is hardware dependent. Hence, Structural Hazard can not be removed using efficient program. Therefore, one solution could be resource duplication, but the cost of implementation will be very high.

Scanned with Any Scanner

Control Hazards

This type of Hazard occurs during the transfer of control instructions such as BRANCH, CALL, JUMP etc. On many instruction architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline. Due to this unwanted instructions are fed to the pipeline.

- All instructions that change the program counter leads to control Hazard.

Branch Instructions

- Conditional (If, for, switch)
  - Condition True (Jump)
  - Condition False (NO Jump)
- Unconditional (Goto) → (Jump)

PC		1	2	3	4	5	6	7	8	9	10	11
1000	I_1:	IF	ID	EX	Mem	WB						
1004	I_2:		IF	ID	EX	Mem	WB → Jump to Address 2000					
1008	I_3:			~~IF~~	~~ID~~	~~EX~~	Mem					
1012	I_4:				~~IF~~	~~ID~~	~~EX~~					
1016	I_5:			Flush		~~IF~~	~~ID~~					
1020	I_6:						~~IF~~					
⋮												
2000	I_{10}:						2000	IF	ID	EX	Mem	WB

BNEZ R_1, 2000

⇒ $R_1 = 0$ → NO Prob.

$R_1 \neq 0$ → Branch to Add

Scanned with Any Scanner

classmate
Date
Page

Solⁿ of Control Hazard: Early detection of jump instruction.

Early detection of jump instruction.

PC							
1004	I_1:	IF	ID	Ex	Mem	WB	
1008	I_2:		IF	~~Stall~~			
2000	I_{10}:			2000	IF	ID	Ex

Consider a system where 20% instructions are branch instruction and because of them the system suffers the following stall.

Branch Instruction = (a) 1 stall.
(b) 2 stall

Find the CPI because of these control dependencies.

Solⁿ (a) Branch instructions = 20% ⇒ CPI = 0.2 × [1+1]
Non-Branch instructions = 80% ⇒ CPI = 0.8 × 1

∴ Avg. CPI = 0.2 × 2 + 0.8 × 1
= 0.4 + 0.8
= 1.2 ⇒ System 20% slower.

(b) Branch Instructions = 20% ⇒ CPI = 0.2 × [1+2]
Non-branch " = 80% ⇒ CPI = 0.8 × 1

∴ Avg. CPI = 0.2 × 3 + 0.8 × 1
= 0.6 + 0.8 = 1.4 ⇒ System 40% slower.

Scanned with Any Scanner

Consider a 6 stage instruction pipeline, where all stages are perfectly balanced. Assume that there is no cycle-time overhead of pipelining. When an application is executing on this 6 stage pipeline, calculate the speedup achieved with respect to non-pipelined execution, if 25% of the instructions incur 2 pipeline stall cycles.

Soln: Time without pipelining, $T_{wp} = 6$
(1 clock cycle per stage)

Time with pipelining, $T_p = 0.75 \times 1 + 0.25 \times [1+2]$
$= 0.75 + 0.75$
$= 1.5$

$\therefore$ Speedup $= \frac{T_{wp}}{T_p} = \frac{6}{1.5} = 4$.

Solution of Control Hazard:-

- Flushing / stalling or NOP (NO operation): It is the worst case scenario, either we remove the computed data or we don't work at all. Which means, the compiler doesn't execute any instruction here.
- Code rearrangement or delayed load: Here with smart compilers we can first execute some instructions which are independent from the current logic.

Scanned with Any Scanner

- Estimation method / Predictive method:
Here we use either 2 stage prediction, then policy change or 1 stage prediction, then policy change.

Scanned with Any Scanner

Register Renaming

WAR Hazard

2 100/50

I_1: $R_3 \leftarrow R_1 / R_2$

300 100 100

I_2: $R_1 \leftarrow R_4 + R_5$

If I_2 is executed prior to I_1:

then 4 200/50

$R_3 \leftarrow R_1 / R_2$

the final value of R_3 will be 4 instead of 2 which is wrong.

Temp. Reg.

T_1	100
T_2	200
T_3	
T_4	2
T_5	30
T_6	

WAW Hazard

2 10/5

I_1: $R_3 \leftarrow R_1 / R_2$

30 10 + 20

I_2: $R_3 \leftarrow R_4 + R_5$

2

If I_2 is executed prior to I_1 then R_3's final value will be 2 instead of 30, which is wrong.

Scanned with Any Scanner

Exception Handling in Pipelining.

Exception Situations : where the instruction execution order is changed in unexpected ways.

- Exception ~~handling~~ situations are harder to handle in a pipelined CPU because the overlapping of instructions makes it more difficult to know whether an instruction can safely change the state of the CPU.
- In a pipelined CPU, an instruction is executed piece by piece and is not completed for several clock cycles.
- Unfortunately, other instructions in the pipeline can raise exceptions that may force the CPU to abort the instructions in the pipeline before they complete.

Types of Exceptions and Requirements

- I/O device request
- Invoking an OS service from a user program.
- Tracing instruction execution.
- Break point.
- Integer arithmatic overflow.
- FP arithmatic anomaly.
- Page fault

Scanned with Any Scanner

- Misaligned memory access.
- Memory protection violation.
- Using an undefined or unimplemented instruction
- Hardware ~~&~~ malfunction
- Power failure.

Exceptions in MIPS (Microprocessor without Interlocked Pipelined Stages)

Pipeline Stage	Problem exceptions occuring
IF	Page fault on instruction fetch; misaligned memory access; memory protection violation.
ID	Undefined or illegal opcode.
EX	Arithmetic exception
MEM	Page fault on data fetch; misaligned memory access; memory protection violation.
WB	None

Scanned with Any Scanner

Instruction Level Parallelism (ILP)

- The overlap among instructions is called Instruction level Parallelism.
- To improve performance
- Two largely separable approaches to exploit parallelism:
  - An approach that relies on hardware to help, discover and exploit the parallelism dynamically.
  - An approach that relies on software technology to find parallelism statically at compile time.
- The simplest and most common way to increase the ILP is to exploit parallelism among iterations of a loop.
- This is often called loop level parallelism.
- All processors since 1985 use pipelining to overlap the execution of Instructions and improve performance.

Basic Compiler Techniques for Exposing ILP.

Aim:

- To use simple compiler technology to enhance a processor's ability to exploit ILP.
- These techniques are crucial for processors that static issue or static scheduling.

Scanned with Any Scanner

Basic Pipeline Scheduling and Loop Unrolling

- To keep a pipeline full, parallelism among instructions must be exploited by finding sequences of unrelated instructions that can be overlapped in the pipeline.
- To avoid a pipeline stall, the execution of a dependent instruction must be separated from the source instruction by a distance in clock cycles equal to the pipeline latency of that source instruction.
- We assume that the functional units are fully pipelined or replicated, so that an operation of any type can be issued on every clock cycle and there are no structural hazards.

There are two separate techniques to exploit ILP

- Dynamic - it depends on the hardware to locate parallelism.
- Static - it relies much more on software.

- The dynamic hardware intensive approaches dominate processors like P-III, P-IV, iF (multicore arch)

pipeline CPI = Ideal pipeline CPI +
Structural Stalls +
Data Hazard Stalls +
control stalls.

Scanned with Any Scanner

Basic Block (BB) ILP is quite small:

- BB: a straight line code sequence with no branches in except to the entry and no branches out except at the exit.
- average dynamic branch frequency 15% to 25%

⇒ 4 to 7 instructions execute between a pair of branches.

- Plus instructions in BB likely to depend on each other.

⇒ To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks

- Simplest approach: loop-level parallelism to exploit parallelism among iterations of a loop.
  - Vector is one way (vector processors / vector architecture)
  - If not using vector processors, then either dynamic via branch prediction or static via loop unrolling by compiler.
- Hardware/Software must preserve program order:
  - that is, the order instructions would execute if executed sequentially, one at a time, as determined by the original source program.
- Instructions involved in a name dependance can execute simultaneously if name used in instructions is changed so instructions do not conflict.

Scanned with Any Scanner

- Register renaming resolves name dependance for registers
- Either by compiler or by hardware.

Preserving exception behaviour ⇒ any changes in instruction execution order must not change how exceptions are raised in programs (⇒ no new exceptions)

Data flow: actual flow of data values among instructions that produce results and those that consume them.
- branches make flow dynamic, determine which instruction is the supplier of data.

Dynamic Scheduling:

Dynamic scheduling is a technique in which the hardware rearranges the instruction execution to reduce the stalls while maintaining data flow and exception behaviour.
- handles cases when dependences unknown at compile time.
  - (eg because they may involve a memory reference)
- it simplifies the compiler.
- allows code compiled for one pipeline to run efficiently on a different pipeline.
- Hardware speculation, a technique with significant performance advantages, that builds on dynamic scheduling.

Scanned with Any Scanner

So, in a dynamically scheduled pipeline, all instructions pass through the issue stage in order; however they can be stalled or bypass each other in the second stage and thus enter execution out of order.

ILP is a measure of how many instructions can be executed simultaneously. There are two approaches to Instruction Level Parallelism.

(a) Hardware

(b) Software

Hardware level works upon dynamic parallelism, whereas software level works upon static parallelism. In dynamic parallelism the processor decides which instructions to execute in parallel, whereas in static parallelism compiler decides which instruction to execute in parallel.

Advantage: It will increase the speed up of execution.

Difference between ILP & Parallel Architecture

ILP	Parallel Arch.
(1) Overlap individual machine operations (e.g. +, −, *, /) so that they can execute parallely.	(1) Having separate processor with the separate code of the program.

Scanned with Any Scanner

② Transparent to the programmer.	② Non-transparent to the programmer.
③ It will speed up the execution.	③ It will speed up as well as quality up.

Techniqe for increasing ILP:

ILP is an architectural techniqe that allow overlap of the overall individual machine operation. Here multiple operations are executed parallely. In other words, it is a measurement of how many instructions can be executed simultaneously. There are two techniques to increase ILP.

① Static Scheduling – It is also known as compiler techniqe for scheduling instructions to separate dependent instruction and minimize the number of actual hazard.

② Dynamic Scheduling – here the hardware re-arranges the instructions

Advange – ① Dependency among instruction can be handled which were unknown to the compiler.

② It simplifies the compiler.

Superscaler: Superscaler processor is a CPU that implement a form of parallelism called ILP within a single processor. It can execute more

than one instruction during a clock cycle by simultaneously dispatching multiple instructions to different execution units on the processor. In other word, multiple instructions are dynamically scheduled. Scheduling of instruction is determined by two factors.

(a) True data dependency.

(b) Resource dependency.

In superscaler processor multiple pipelines are there and multiple independent instructions are executed parallely.

- It is more commonly used in RISC computers.
- In superscaler processor there are multiple functional units each of which is implemented as a pipeline.
- Superscaler processors are much faster than scaler processors because it increases throughput as the CPU can execute multiple instructions per clock cycle.
- A scaler processor works on one data item, while in Superscaler processor each instruction processes one data item, but as there are multiple execution units within each CPU, thus multiple instructions can be processing separate data items concurrently.

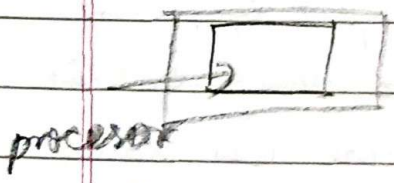
Scanned with Any Scanner

- A superscaler processor typically fetches multiple instructions at a time and then attempts to find nearby instructions that are independent of one another and can therefore be executed in parallel. But if there is any dependency between input and output of instructions then that instruction can not be executed parallely.

	1	2	3	4	5	6	7	8
I_1	IF	ID	EX	MEM	WB			
I_2		IF	ID	EX	MEM	WB		
I_3			IF	ID	EX	Mem	WB	
I_4				IF	ID	EX	Mem	WB

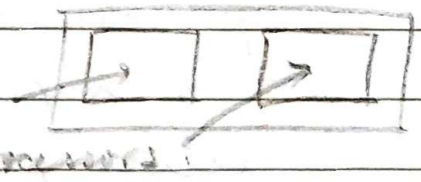
	1	2	3	4	5	6
I_1	IF	ID	EX	Mem	WB	
I_2	IF	ID	EX	Mem	WB	
I_3	IF	ID	EX	Mem	WB	
I_4	IF	ID	EX	Mem	WB	
	IF	ID	EX	M		
	IF	ID	EX	M		

- One pipe line



- It issues one instruction per clock cycle and can perform one pipeline stage per clock cycle.

- two pipelines



- It issues two instructions per clock cycle and is capable of executing two instances of each stage in parallel.

Scanned with Any Scanner

Limitations of Superscaler processors

1. Superscaler processor depends on the ability to execute multiple instructions in parallel.
2. A combination of compiler based optimization and hardware techniques can be used to maximize instruction level parallelism.
3. If instructions are data dependent and resource dependent and if these dependencies are not handled effectively, it is difficult to achieve an execution rate of more than one instruction per clock cycle.

Superpipelined Processor

- An alternative approach to achieve greater performance is reffered to as Super pipelining.
- In Super pipelining processor; many pipeline stages perform tasks that require less than half a clock cycle.
- A superpipelined architecture is one that makes use of more and more small stages of a pipeline in attempt to shorten the clock period. With more stages, more instructions can be in the pipeline at the same time, increasing parallelism (or throughput).

Scanned with Any Scanner

	1	2	3	4	5	6	7	8
I_1	I	D	E	M	WB			
I_2		I	D	E	M	WB		
I_3			I	D	E	M	WB	
I_4				I	D	E	M	WB

pipelined processor

8 cc for 4 Instr.

	1	2	3	4	5
I_1	I	D	E	M	WB
I_2	I	D	E	M	WB
I_3	I	D	E	M	WB
I_4	I	D	E	M	WB
I_5	I	D	E	M	WB

Superpipelined processor

4 cc for 4 Instr.

- The number of instructions being processed at a given time depends on the number of pipeline stages, commonly termed as pipeline depth.
- Some designer use maximum 8 stages of pipelining

Limitations:

① The speed of execution is limited to the slowest stage of the pipeline.

Benefit ⇒ Increases level of parallelism.

Difference between Superscaler and Superpipelined processor:

- In superscaler processor, the entire execution unit has been doubled and each unit has 5 pipeline stages. (2-3) Therefore in superscaler processor there is more than one execution unit and each one has many stages.

Scanned with Any Scanner

- While in Superpipelined processor, there is only one execution unit with 8 pipeline stages.

VLIW Architecture / VLIW Processor

VLIW ⇒ Very Long Instruction Word.

VLIW refers to a processor architecture designed to take advantage of Instruction Level Parallelism.

- It is less complex approach to higher performance (i.e. multiple operations performed simultaneously or level of parallelism increases)

* In VLIW processor:

- ① Instruction consists of multiple independent operations grouped together.

Single Instr → Ex. ADD R_1, R_2; SUB R_5, R_6; LOAD R_7, data; STORE R_8, data.

- ② There are multiple independent functional units (to execute all the instruction operations)
- ③ Each operation in the instruction is assigned to different functional units.
- ④ All functional units share the use of a common large register file.

- In VLIW processor, compiler is responsible for static scheduling of instructions, i.e. the compiler finds out which operations can be executed in parallel in the program.

Scanned with Any Scanner

- the compiler groups together these ~~instructions~~ operations in a single instruction which is the Very Large Instruction Word (VLIW).
- the instructions within a VLIW instruction are issued and executed in parallel.
- Since in a VLIW processor, one VLIW instruction word encodes multiple operations which allows them to be initiated in a single clock cycle.
- VLIW instructions contain operands and operations to be performed by the various functional units.
- One VLIW instruction encodes at least one operation for each execution unit of the device. So, length of the instruction increases with the number of execution units.
- VLIW instructions are usually atleast 64 bit wide and 1024 bit wide to some architecture.

Comparison of Instruction level parallelism on Sequential processor and on VLIW processor:
Assume that a VLIW processor has 2 Load/store units, 1 multiply unit and 1 add unit (total 4 execution units)

Scanned with Any Scanner

Consider the computation of

$$Z = \underbrace{Ax_1}_{y_1} + \underbrace{Bx_2}_{y_2} + \underbrace{Cx_3}_{y_1}$$

On Sequential processor

clk cycle No.	
1	LOAD A
2	LOAD x_1
3	Multiply y_1, A, x_1
4	LOAD B
5	LOAD x_2
6	Multiply y_2, B, x_2
7	ADD y_3, y_1, y_2
8	LOAD C
9	LOAD x_3
10	Multiply y_1, C, x_3
11	ADD Z, y_1, y_3

$\boxed{\text{Total 11 clock cycles}}$

On VLIW processor

clk cycle No.	
1	LOAD A LOAD x_1
2	LOAD B LOAD x_2 Multiply y_1, A, x_1
3	LOAD C LOAD x_3 Multiply y_2, B, x_2
4	ADD y_3, y_1, y_2 Multiply y_1, C, x_3
5	ADD Z, y_1, y_3

$\boxed{\text{Total 5 clock cycles}}$

Advantages of VLIW:

1. Dependencies are determined by the compiler and used to schedule according to functional unit.
2. Reduces hardware complexity:-
   Tasks such as decoding, data dependency, detection, instruction issue etc. becoming simple and ensures low power consumption.

Scanned with Any Scanner

Disadvantages of VLIW:

1. Higher complexity of the compiler.
2. VLIW processors can not react on dynamic events. It can react on static events.
3. The number of instructions in a VLIW instruction word is usually fixed. So, padding VLIW instructions with no-opns. is needed in case the full issue bandwidth is not be met. This increases the code size.

Applications:

1. It is suitable for digital signal processing (DSP) applications.
2. Processing of media data like compression / decompression of image and speech data.

Examples of VLIW Processors:

- VLIW mini Supercomputers - Multiflow trace 7/300, 14/300, 28/300.
- Single chip VLIW Processors - Philips LIFE CHIPS.
- DSP Processors.

Scanned with Any Scanner

Vector ~~Array~~ Processing

- There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require a vast number of computations that will take a conventional computer days or even weeks to complete.
- Computers with vector processing are able to handle such instructions and they have applications in the following fields —
  - Long range weather forecasting.
  - Petroleum exploration.
  - Medical diagnosis.
  - Aerodynamics and space simulations.
  - Artificial intelligence and expert systems.
  - Image processing.

Complex instructions which operate on multiple data at the same time, requires a better way of instruction execution, which ~~is~~ achieved by vector processors. (SIMD)

Vector processors not only use instruction pipeline but it also pipelines the data, working on multiple data at the same time.

Defn: In computing, a vector processor is a CPU that implements an instruction set containing instructions that operate on one-

dimensional array of data, called vectors, compared to scalar processors, whose instructions operate on single data items.

- Vector processors have high level operations that work on linear arrays of numbers or vectors.

Scalar Quantity : | Data |

Vector Quantity : | Data | Data | Data | Data | Data |

Vector Operations :

$V = [V_1, V_2, V_3, \ldots V_n] \rightarrow$ vector of length n.

The element V_i of vector V, $i = 1, 2, 3, \ldots n$, is written as $V[I]$, where I is the index that refers to memory address or register where the number is stored.

Addition of Values of 2 arrays :

```
int a[10], b[10];
int c[10]
for (i=0, i<n; i++)
c[i] = a[i] + b[i]
```

C-language

Initialize $i = 0$
Read $a[I]$, Read $b[I]$
Store $c[I] = a[I] + b[I]$
Increment $I = I + 1$
if $I < 10$, continue

Assembly language.

Vector processors have the ability to remove overhead of instruction fetch and execution in loop.

Instruction can be written as—

Single vector instruction $\boxed{C(1:10) = A(1:10) + B(1:10)}$

Scanned with Any Scanner

Any Vector Instruction has

Operation Code	Base Address Source 1	Base Address Source 2	Base Address Destination	Vector Length.
Ex: ADD	A	B	C	10

Vector Processing Definitions.

Vector: A vector is an ordered set of scalar data items, all of the same type, stored in memory.

- Vector elements are ordered to have a fixed addressing increment between successive elements called the Stride.

Vector Processor: A vector processor consists of hardware resources like vector registers, functional pipelines, processing elements and register counters for performing vector operations.

Vector Processing: Vector processing occurs when arithmatic or logical operations are applied to vectors. It is distinguished from scalar processing, which operates on one or ~~more~~ one pair of data.

Scanned with Any Scanner

Vectorization: The conversion from scalar code to vector code is called as vectorization.

Array Processing

It performs computations on large arrays of data.

Array Processors
- Attached array processor
- SIMD array processor (Single Instr. Multiple Data stream)

① **Attached array processor:** To improve the performance of the host computer in specific numerical computation tasks, auxiliary processor is attached to it.

② **SIMD array processor:** This is a computer with multiple processing units operating in parallel. Both types of array processors, manipulate vectors but their internal organization is different.

Scanned with Any Scanner

Attached array processor

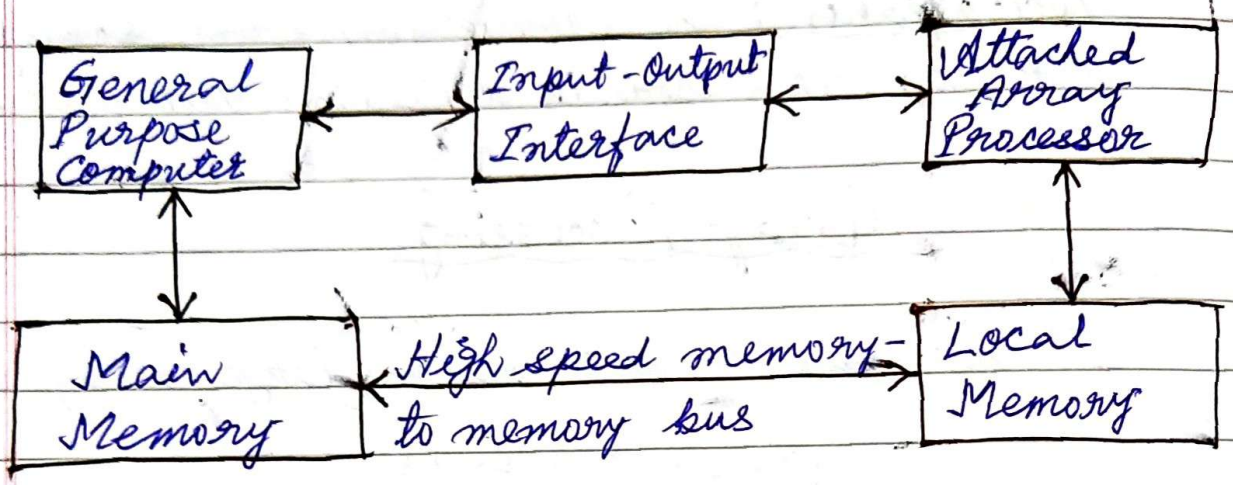


Fig. Interconnection of an attached array processor to a host computer.

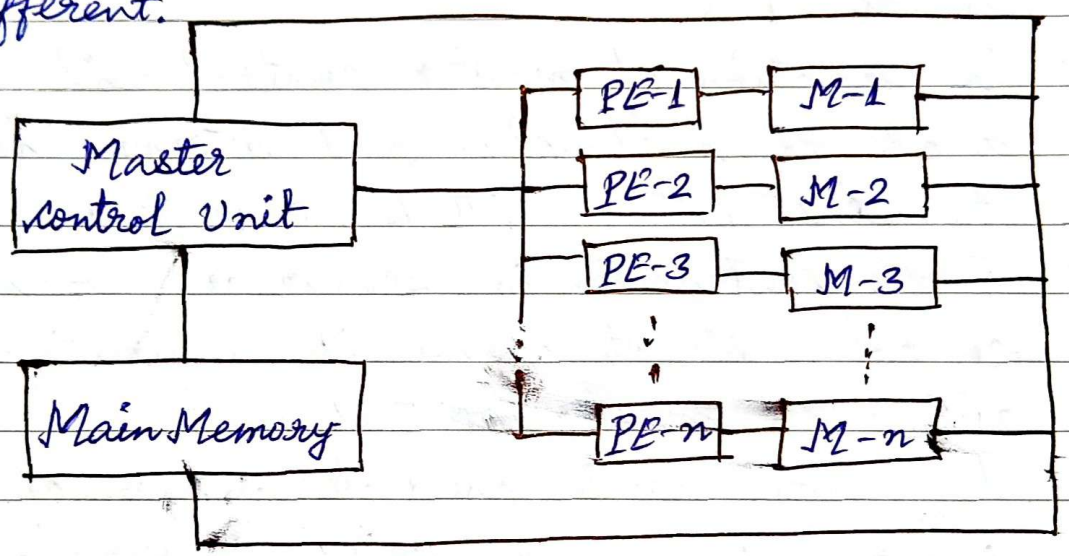
Attached array processor has two interfaces:
① Input-Output interface to a common processor.
② Interface with a local memory.
Here local memory interconnects main memory.
Host computer is General Purpose computer.
Attached processor is Backend machine driven by the host computer.
The array processor is connected through an I/O controller to the computer and the computer treats it as an external interface.

Scanned with Any Scanner

(Single Instruction Stream Multiple Data Stream)

SIMD Array Processor

This is a computer with multiple processing units operating in parallel.
Both types of array processors, manipulate vectors but their internal organization is different.



PE - Processing Element ; M - Memory Element.

- SIMD is a computer with multiple processing units operating in parallel.
- The processing units are synchronized to perform the same operation under the control of a common control unit, thus providing a single instruction stream, multiple data stream (SIMD) organization.
- As shown in fig., SIMD contains a set of identical processing elements (PEs), each having a local memory M.

Scanned with Any Scanner

- Each PE includes
  - ALU
  - Floating point arithmetic unit.
  - Working registers.
- Master control unit controls the operations in the PEs
- The function of master control unit is to decode the instruction and determine how the instructions will be executed.
- If the instruction is scalar or program control instruction, then it is directly executed within the master control unit.
- Main memory is used for storage of the program while each PE uses operands stored in its local memory.

Ex: Let vector addition $C = A + B$.

- The master control unit first stores the ith components a_i & b_i of A and B in local memory M_i for $i = 1, 2, 3, \ldots n$.

  It then sends floating point add instruction $c_i = a_i + b_i$ to all PEs and then the addition takes place simultaneously.
- The components of c_i are stored in fixed locations in each local memory, this produces the desired vector sum in one add cycle

	a_i	b_i	c_i
PE-1 →	1	2	3
PE-2 →	2	4	6
PE-3 →	3	2	5
⋮	⋮	⋮	⋮

- Masking are used to control the status of each PE during the execution of vector instructions.
- Each PE has a flag that is set when the PE ~~instruction and~~ is active and reset when the PE is inactive. flag [1] → Active [0] → Inactive
- only Active PEs ~~only~~ executes the instructions for computation.

Ex: 64 PEs : Vector length = 38

∴ 38 PEs will be active.

Scanned with Any Scanner