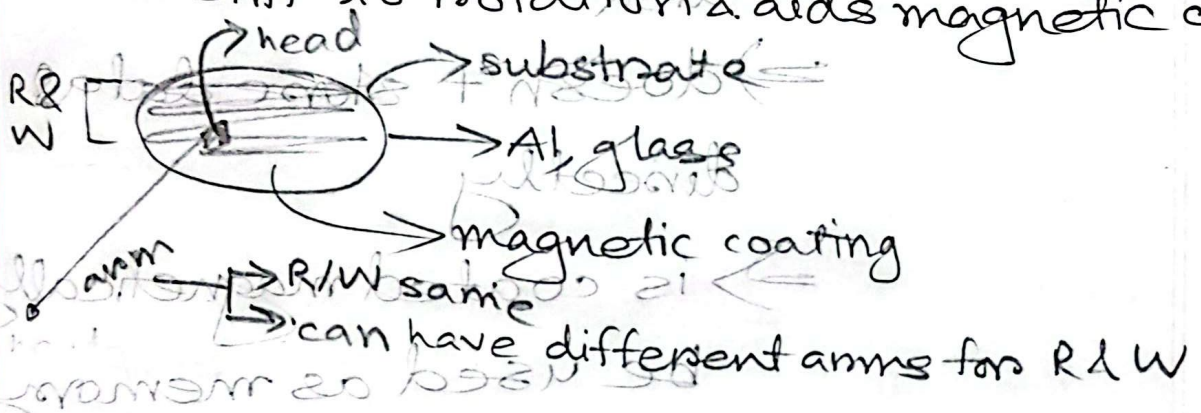
19-10-2024
SATURDAY

COA 4305
Hanjajul Ashmafee sir

Chapter-6

* Magnetic disks are created of Al and glass (more suitable). Its perimeters are called substrate which can't be read or written upon. Often multiple plates are used to make a disk and both the surfaces can be magnetized. Glass is more stiff to rotation & aids magnetic coating.

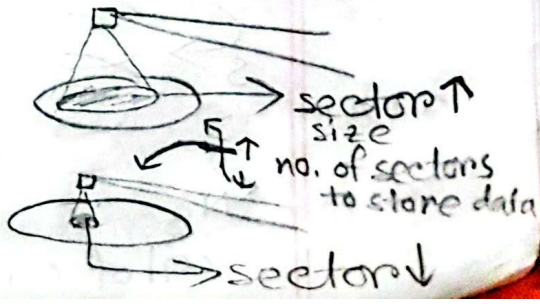


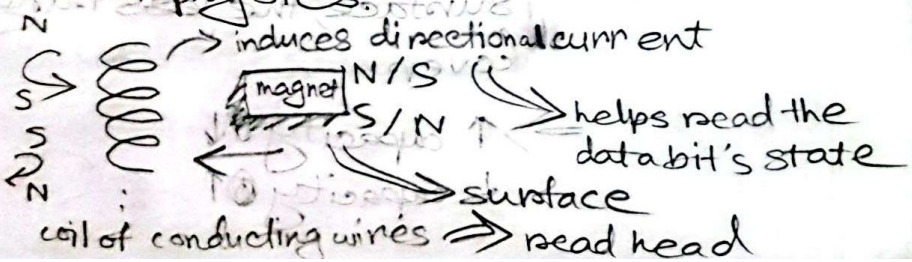
* Due to dynamic inertia, arm may not be aligned with the aimed cell if the substrate isn't stiff enough.

* Fly Height ⇒ the closer the head to the surface, the less range it'll cover

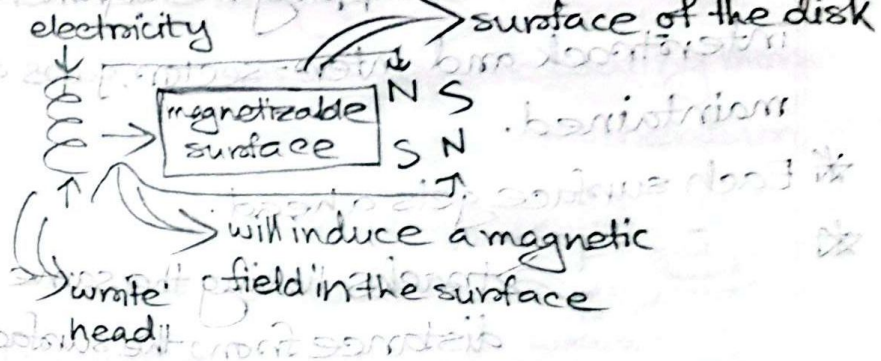
⇒ ↑ capacity ↓

↓ capacity ↑



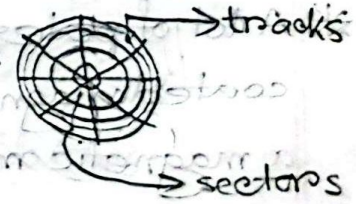
* A R/W operation can be run on a sector which is a packet of data.
* Substrate's ~~suf~~ surfaces are magnetized. They can be made of different materials.
* Substrate ⇒ primary material of the disk
  ⇒ doesn't store data directly
  ⇒ is coated magnetically to be used as memory devices
* Head can be separated into R & W or can be merged. They operate on a single bit at a time. They function based on 2 laws of physics:
* 

So, based on the direction of the current, the bit's state is read.

* 

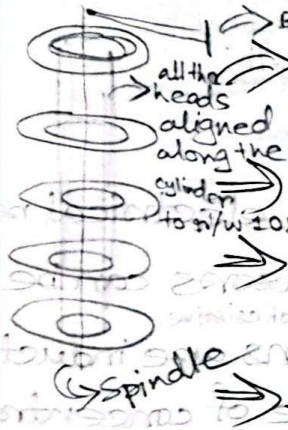
To properly sense the electrical resistance of the signals, MR sensors can be used.
↳ MagnetoResistive

* Magnetic R/W operations are inductive.
* Magnetic disks are made of concentric circles /rings. That's how data is organized on a surface. These circles are called Tracks. Each track is divided into Sectors.



* The closer to the center, the smaller in size the tracks & the sectors get. Each track has the

same no. of sectors.

* To prevent overlapping misalignments, interstrack and inter-sector gaps are maintained.
* Each surface gets a head.
* 

  ⇒ tracks having the same distance from the surface
  ⇒ 5 disks X 2 surfaces = 10 tracks
  ⇒ can work as a cylindrical stack to store a 10-bit data.
  ⇒ this method allows the R/W operations to be carried out simultaneously on multiple platters
* A lot of plates, with both the surfaces ~~ma~~ coated magnetically, are adjusted into a magnetic memory disk.

5-11-2024
TUESDAY

COA04305
~~[illegible]~~
Hamjajul Ashmafee sir

* Magnetic disks are a common example of external memory.
* Glass makes the magnetic coating more uniform and less prone to momentum, shock and damage, allows lower fly height.
* Head does the R&W.
* fly height ↓ ⇒ cell size ↓ (unit area for being R&W operable)
* Heads are made up of coils of wires which utilize electromagnetic field.
* MR sensor increases (or decreases) the resistance in the read head for proper reading. Sheild prevents side-reading.
* Head size = Track width
* To prevent magnetic influence of adjacent cells, a gap is maintained.
* For each head, one coil is set up.

* Magnetic disks are separated into tracks & sectors to locate a cell.
  tracks: concentric rings across the surface

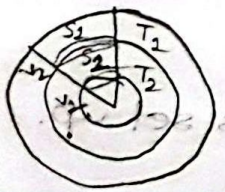
* The closer to the centre, the less circumference will be for a track.

* Gaps between track (Intertrack Gaps) & sectors (Intersector Gaps) are installed to prevent misalignment.

* The size of a sector decreases as the tracks trickle towards the center.

* Cylinder shape ~~operations for~~ allows R-W operations over several surfaces simultaneously.

* The arm is connected to the Boom while the disk spins ~~around~~ around the spindle.

* 

  At time t, $V_1 = \frac{d_1}{t}$ and $V_2 = \frac{d_2}{t}$ (distance from centre to t_1, t_2)

  $V_2 > V_1 \Rightarrow T_1$ must be passed faster

Angular velocity $= \omega$

$\omega_1 = \frac{\theta_1}{t}$, $\omega_2 = \theta_2 / t \Rightarrow \omega_1 = \omega_2$

For the constant angular velocity approach, which is easier to implement but spacially wasteful, all sectors in all tracks will have the same amount of data (limited by the size of the sector in the 1st track), so the head can operate on same size of data on all tracks of a sector.

* Zone division method maintains bit density (bits/sector length). It's more complicated to implement but ensures spacial efficiency. However, the head must be smart enough to calculate the zone & operate accordingly. Each zone's bit density is fixed.

* If a sector is permanently damaged, it's a Bad sector. A non-smart OS will throw

an error in this case. A smart one, however, will allocate some of the OS reserved memory (driver's) to run the operation on the redirected location.

* A sector's identity is held at ID Field. Head no. is necessary if there are multiple heads for each surfaces.
* Sync bytes can be 0000 or 1111 signify the start of a R or W operation.
* no. of sectors × bytes/sectors = track size
* Average time to access a track isn't much helpful as the time requirement is much different for tracks on opposite ends of the disk.
* Fixed head => no. of heads = no. of tracks

  Movable head => 1 head for a surface
* Floppy disk is dynamic, HDD is fixed.

* Head mechanisms:
  1. Fixed gap head (physical disconnection)
  2. Contact head
* Head gets to the lowest fly height (≈ 0) when a R-W operation is ongoing, but during rotation, an air pressure uplifts the head to shield it from damage due to aerodynamic friction.
* Access Time => time needed for a R/W operation

  Seek Time => time needed to spot the desired ~~track~~ location (track)

  => T_s

  Rotational Delay => time needed to get the desired sector of the chosen track

  => T_r

  Wait for Device => EM may be serving a prior request

  Wait for Channel => System bus may be busy

serving a prior request

$$T = T_s + T_r + T_t$$

$$= T_s + \frac{1}{2r} + T_t$$

r → rotational speed

2 → rotation can occur in both directions

$$= T_s + \frac{1}{2r} + \frac{1}{r} \cdot \frac{b}{N}$$

b → portion of the track to be transferred

b → no. of bytes to be transferred

N → no. of bytes in a track

r → unit rotational speed

$\frac{1}{r}$ is the speed needed to transfer one byte from that track

r is calculated for the entire track.

5-11-2024
TUESDAY

DBMS 4307
Prof. Dr. Abu Raihan Mostofa Kamal sir

* 0 precision ⇒ no decimal point
* σ is for conditions (i.e, it captures the "Where" part) whereas π captures the "Select" portion.
* OLTP has multiple users and features the acid property.
* Data security hasn't improved dramatically since migrating to 3-tier architecture.
* Cardinality must consider the global picture on a long-term speculation.

Chapter-5

Advanced SQL

* "new" keyword is used to create an object.
* JDBC ⇒ standard get-away for Java-based applications
  ⇒ Java DataBase Connectivity API
* "main" is the method from where the execution begins
* "static" functions don't belong to any instances. To call these, creating any objects isn't necessary.

6-11-2024 WEDNESDAY

OOP 4301

Faisal Hussein sir

* In Java, even the "main" function is defined inside a class.
* A code is always expected to be refactored. However, adding new codes is preferred to modifying older ones.
* ~~Br~~ Behaviours should be added where it belongs (depending on the requirement).
* A Pointer is a special type of variable that holds addresses.
* A memory space is reserved just for the program's variables.
* * ⇒ Deference/Indirection Operator
* All types of pointers occupy the same size. However, mismatched assignment isn't allowed (except for void pointers).

HW * const int* ≠ int* const

* "new" is an operator which replaces DMA malloc in C++.

example: int* i = new int [size]; → If omitted, a single variable/object is created.

* DMA is a runtime operation.
* "delete" is used to deallocate DMA.

example: delete[] arr;
for arrays

If not used ~~properly~~, the error of Runaway Pointer may occur while space is wasted

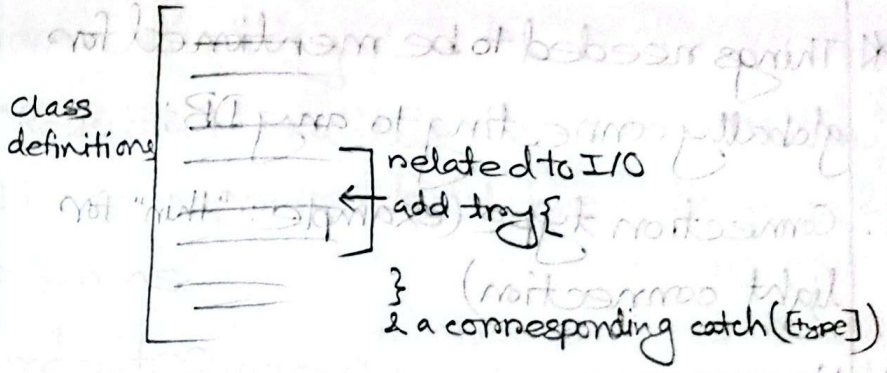
* Creating a pointer doesn't trigger the constructor.
* A base class' pointer can point to its derived classes' objects. Vice-versa isn't allowed. However, the former type of pointer can only call/access the base members (functions). Also, it won't call the functions overriden in the derived classes.

* When a pointer is declared, only its type or class definition is used.
* To make the base class' pointer access the derived class' members, the function to be overriden should be virtual. In that case, the base class' pointer will check its contents (i.e., class type that's assigned) and look for overriden functions first. It'll return to the base class' definition only if no overriden function is found.

-11-2024
THURSDAY

DBMS 4307
Prof. Dr. Abu Raihan sir

* There are 2 exceptions handling;
1. Runtime ⇒ divide by 0 based on user input is an example
2. I/O
* Runtime error handling can be optional, but it's mandatory for I/O exceptions.

* I/O exceptions relate accessing stuff outside JVM (Java Virtual Machine).
* 

* "try" & "catch" are the way to detect I/O errors in Java.
* PL SQL ⇒ Programmable SQL
  ⇒ this is for Oracle, alternatives exist for other platforms.
* CONNECT userName/passcode @ DB_Alias;
  name of the DB
* Socket connects the services w/ its port numbers (can be ephemeral). PN & IP address are needed to universally connect to any (global/local) database.

and "Statement"

* "Connection", "DriverManager", are classes in Java.
* Things needed to be mentioned for globally connecting to any DB:

1. Connection type (example: "thin" for light connection)
2. Username
3. Passcode
4. DB's name (XE for Oracle 11)
5. Port no.
6. IP address/name of the machine (Linux can translate these)

* '+' is used to concatanate strings in Java,
* statement objects can be null.
* statement & connection objects should be closed after usage.
* When Java uses the keyword "update", it refers to SQL commands inside its code (different from SQL's update).
* Statements can execute updates in the form of strings.
* "ResultSet" is a class similar to a 2D array. It can represent a table with rows & columns.
* "executeQuery" treats SQL commands as string
* rset.next() automatically increments the row no.
* Everything inside the "System.out.println" must be strings.
* Records' indexing starts from 1, not 0, for the attributes under query. They can be replaced by their attribute's name as strings.
* SQL Injection deals with queries based on user inputs; used to be a security concern.

7-11-2024
THURSDAY

OOP4301
Faisal Hussein sir

* C# adds some advantages over Java (such as: embedded SQL), but it must work with a SQL server.
* For assigning an address to a pointer, the compiler checks the compatibility between the types.
* A Virtual Function is a function whose behaviour can be overridden within an inheriting class with a function with the same signature. It allows the pointers late/dynamic binding as it checks the type of the object pointed at by the base class' pointer & call the overridden function if it's present.
* In C++, the keyword "override" isn't mandatory. "@override" is necessary to be mentioned before the method for Java. Such conventions ~~check for~~ prevent errors.
* base_ptr = &drv_object; can't call members not present in the base class (error: "class [base]' has no member named '---'").
* Virtual Functions ⇒ 1 interface, multiple methods
  ⇒ example of polymorphism
* Late/dynamic binding allows runtime polymorphism.
* Virtual functions are hierarchial.
* Base class' pointers support the address of objects from ~~multiple~~ classes which used multi-level inheritance.
* Because of the property of inheritance, derived objects and pointers can access members (which are accessible) defined in the base class.
* Pointers only stores one address. In case of contiguous memory spaces, it's the 1st cell.
* Error message for drv_ptr = &base_object;

"Invalid convertion from * [base] to * [derived]."

* Lack of overridden counterpart will lead the virtual function back to the base class.

* "override" keyword in the derived class won't compensate for the lack of "virtual" keyword in the base class.

* An inherited class must be careful about the base class' constructors. If it's 0 arg., it can be omitted; otherwise the initializer list must be used.

* Even for multi-level inheritance, for a virtual function, the object's overridden function will be called, otherwise the compiler will return to the base class and the intermediate classes will be ignored (however, the properties of inheritance make this principle obsolete).

* base_ptr = ldm_obj; allows single interface for multiple methods.

* Virtual functions are used for run-time executions.

* A blank func. for a base class may allow its virtual nature to ensure elegant & dynamic func. calling for its derived classes.

* A Pure Virtual Function —

— is blank (only the prototype is mentioned, & not the body)

— derived classes must override this func.

— isn't really there, but we assume its virtual existence to ensure derived classes' functionalities

* virtual type funcName() = 0; ⇒ Pure Virtual Function

* If a PVF is not overriden in a derived class, a compilation error will pop up.

* If a class has PVF, it's an Abstract Class. It cannot have objects and cannot be instantiated

~~der~~ directly. If the derived class doesn't override the PVF, it also becomes an abstract class. However they can have pointers.

* An object of an abstract class can't be created as it isn't fully defined. Pointers are possible due to polymorphism.
* instantiation ⇒ creating an object

11-2024
THURSDAY

DS 4303
Sabbir Ahmed sir

Hashing

* Hash-Table allows searching (/accessing) in O(1).
* Hash-Table works in {key, value(s)} method.
* Hash function is used to access hash-table.
* Hash-Tables, made using the hashing technique, don't maintain orders.
* Maps are implemented using Red-Black tree, Unordered Map uses hashing.
* Hash table is indexed in the BG.
* Hash table's memory allocation isn't sequential or contagious.
* Hash functions map the key to the hash table.
* If two keys are allocated the same space or index, a Collision occurs. Hash functions try to minimize it by distributing the keys evenly throughout the cells. The simplest solution is to drive the new key to the next free space.
* Searching in hash function can be approximate on the memory level, hence even distribution is recommended.
* Hash value is chosen by the function. It shouldn't be so simple to intensify collisions.
* For integer keys, (key % tablesize) indexing is the most general strategy. Better if tablesize is prime for proper distribution and less collisions.
* Prime numbers are preferred for calculations in hashing as it decreases collisions.

* If a hash function's mechanism allocates the same index for 2 different keys, such a mechanism is called Associative.
* Separate Chaining method maintains a chain for all indices & the particular chain grows if there's a collision. It saves all the collided indices in a list.
* Load Factor, $\lambda = \frac{\text{no. of items}}{\text{table size}}$

$\lambda \uparrow$ (possibility of) collisions $\uparrow$

$\lambda \approx 0.5$ is the optimal value. If it starts to exceed it, the STL map will do a mechanism called Rehashing (transferring the indices to a larger table).
* Separate chaining isn't memory efficient, its TC isn't much bright either.
* Open Addressing uses a helper function for collision resolution. ⇒ index = (key + f(i)) % ts

— Linear Probing ⇒ $f(i) = i$

⇒ i starts w/ 0 and is incremented

* Technically, the separate chaining isn't restrained by the table size. Open addressing, however, is.

⇒ insertion can run for O(n) as it can build clusters (Primary Clustering)

— Quadratic Probing ⇒ $f(i) = i^2$ (popular choice)

⇒ generally a free slot found within 2-3 attempts

⇒ still can create clusters as a jumping pattern persists (secondary clustering not as obvious)

⇒ can lead to O(n) insertion

⇒ distribution is more even

Linear probing guarantees a free slot (if all the space isn't filled), but quadratic probing can't (as its pattern is fixed) if $\lambda > 0.5$.

10-11-2024
SUNDAY

COA 4305
Md. Hamjajul Ashmafee sir

* RAID: This mechanism is applied on magnetic disks.

⇒ Redundant Arrays of Independent/ (theoretical)
Inexpensive Disks (practical) □ □ □ □... different slots

⇒ counter-mechanism of SLED
one large space Single Large Expensive Disk

* RAID distributes data into small portions while SLED stores them all at a one, large list. Each disk can only process one operation at a time (RAID allows parallelism as multiple portions can process distinct commands).

* Redundancy is used for crucial/critical/vulnerable data, so that if one copy is damaged or pre-occupied, the machine can access another copy instead of freezing. OS manages which data to be made redundant copies. Although the data are physically redundant and distributed, the machine interprets it as a single logical storage. AMS (Array Management Software) takes care of this mapping.

* Logical data should be stored sequentially. Physically, they can be mapped to a strip/stripe (unit block of physical data) of a physical disk.

a set of strip distributed evenly & equally (in terms of position) among physical disks ⇐ similar to the cylinder concept

distribution of physical data

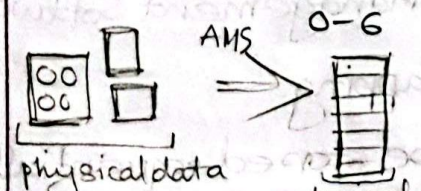
* Mirroring & Stripping are mechanisms used to perform RAID operations.

— strips: A B C D

copied to multiple place: ~~A B C D~~ ⇒ mirroring (similar to redundancy)

A A
B B
C C
D D

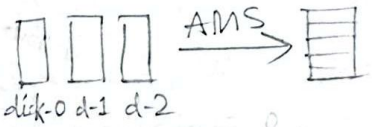
* Redundancy allows speedy recovery.
* AMS handles striping
* RAIDs have 7 levels (not hierarchial).



↳ SW doesn't recognize distribution
— sequential.

* Mirroring happens on strip-level, not disk-level.
* RAID-0 has no to absolutely rudimentary features of RAID. Here, it's not standard, rather partially RAID-compatible.

— no redundancy

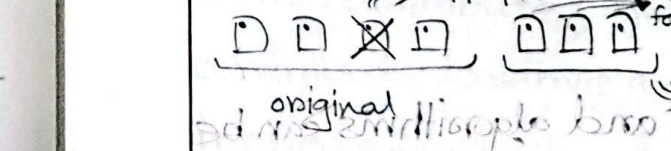


— data is physically distributed and logically sequential
— disks can independently serve I/O requests (benefit of distribution). This parallelism enhances the speed-up.
— data channel must be wide enough to carry multiple requests simultaneously (bandwidth should be sufficiently increased)
— non-redundant, but distributed (# of pd = # of ld)
* RAID-1 introduces redundancy via mirroring
— no. of physical disks = 2x no. of logical disks
— strip-level accessible
* RAID-2 reduces the space requirement (by reducing the redundancy)
— Hamming Code or other ECCs can be used on the original data

* Blocks are larger than strips.

— data regeneration is still possible through ECC
— $m+k \leq 2^k+1$

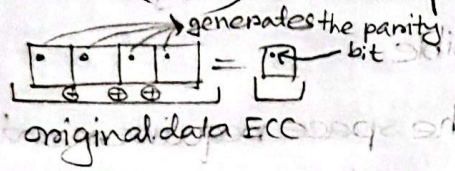
let $m=4, k=3$ ⇒ not fixed, rather dynamic (based on m)



After the disk has been identified, the bit is toggled. This method can handle 2 damaged disks at a moment (for the scope of our knowledge).

* RAID-3 involves complex mathematical calculation & mechanisms.

- redundancy further reduced.
- single disk (fixed) to recover data if any of 4 disks (fixed) is corrupted.



- somehow, it must be known which disk is corrupted (then parity bit will help in the recovery), otherwise we must revert to RAID-2.
- parity bit signals if a suspicious disk's data is intact or not (assumed to enough effective of a mechanism for our level of study).

* Multiple approach and algorithms can be combined as ECC & ~~ECD~~ EDC.

* Write Penalty is an issue in RAID-2 & -3. ~~Determining the strip~~ All the disks must be updated for which the strip is involved (redundant disks & code disks)

WP = # of disks affected for a single write operation (on a strip) (includes the # of disks to be read & written upon for the operation to run smoothly)

WP $\xrightarrow{\text{RAID-1}}$ 2

$\xrightarrow{\text{RAID-2}}$ 7 (for one strip to be updated, all the codes must be modified)

$\xrightarrow{\text{RAID-3}}$ 2 $\boxed{x_1} \oplus \boxed{x_2} \oplus \boxed{x_3} \oplus \boxed{x_4} = \boxed{p}$

$\Rightarrow x_1' \oplus x_2 \oplus x_3 \oplus x_4 = p'$

$\Rightarrow \underbrace{x_1 \oplus x_1}_{0} \oplus x_1' \oplus x_2 \oplus x_3 \oplus x_4 = p'$

$\Rightarrow x_1 \oplus x_1' \oplus \underbrace{x_1 \oplus x_2 \oplus x_3 \oplus x_4}_{p} = p'$

$\Rightarrow \underbrace{x_1 \oplus x_1' \oplus p = p'}_{\text{2 disks}}$

* Upto now, RAIDs worked on small-sized strips. In this distributed arrangement, all disks contribute to serve a single I/O request. This increases the bandwidth, but can be wasteful in terms of energy. Hence, later RAIDs introduce larger strips (blocks) so that multiple requests can

be served simultaneously.

* RAID-4 operates on blocks instead of stripes.
- all disks can serve different requests & channels.
- all data disks are linked to the same parity disk. This bottleneck disallows multiple write operations at a moment.
- parity disk's access is allowed to one disk at a moment.

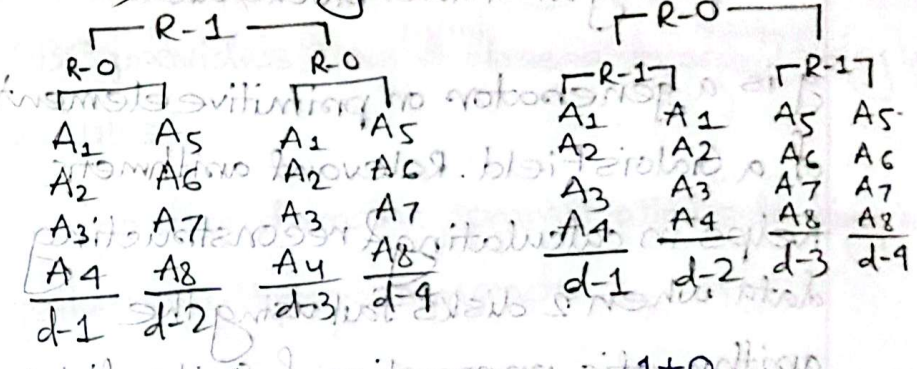
* RAID-5 distributes the parity blocks among different data disks.
- multiple fetching is (often) possible.
- no separate parity disks.
- which disk has the corresponding parity block, it must be calculated.

* RAID-6 introduced dual redundancy.
- 2 corrupted disks can be detected.
- 2 different parity generation codes are stored for each block.

* RAID levels can be nested.

example: 0+1 ⟹ distribution, then mirroring

1+0 ⟹ mirroring then distribution



* RAID-6 utilizes multiple ECC codes at once to ensure integrity.

example- $P = \bigoplus_i D_i = D_0 \oplus D_1 \oplus D_2 \oplus \ldots \oplus D_{n-1}$

$Q = \bigoplus_i g^i D_i = g^0 D_0 \oplus g^1 D_1 \oplus \ldots \oplus g^{n-1} D_{n-1}$

If disks D_i & D_j are affected ⟹

$A = P \oplus \left(\bigoplus_{l: l \neq i, l \neq j} D_l \right) = D_i \oplus D_j$

$B = Q \oplus \left(\bigoplus_{l: l \neq i, l \neq j} g^l D_l \right) = g^i D_i \oplus g^j D_j$

$D_j = (g^{m-i+j} \oplus 1)^{-1} (g^{m-i} B \oplus A)$

$D_i = A \oplus D_j$

m: no. of disks/blocks

A: P parity for intact blocks/disks

B: Q parity for intact blocks/disks.

g is a generator or primitive element of a Galois Field. Relevant arithmetic helps in calculating & reconstructing data when 2 disks fail using the arithmetic properties of finite ~~disks~~ fields. RAID-6 applies these for redundancy and recovery.

12-11-2024
TUESDAY

DBMS 4307
Prof. Dr. Mustafa Kamal sir

* Running static queries from UI is obsolete. Rather programming and using variables make the query dynamic.
* The syntax for strings is same in Java and C/C++.

example: SELECT *
FROM T
WHERE TEST= "+t" → string in Java, variable, string in SQL

* Taking user inputs in Java and passing it to SQL can be exploited using SQL injection. This has been, more or less, solved globally.
* JDBC provides a set of classes to seamlessly use SQL DBs.
* Turning off automatic commit allows us to ~~bing~~ bring together multiple commands.
* ODBC => Open DataBase Connectivity
  => uses C# or Python
* PL/SQL => Programmable SQL
* Block => smallest unit of executables bundled together.

Types in PL/SQL:

1. Anonymous: one-off, not reusable, can't be called from outside.

example:
```
BEGIN
DBMS_OUTPUT.put_line ('HW');
END;
set serveroutput ON; -> precondition
/ => symbol to initiate execution
```

2. Named: Reusable, can be called, stored, similar to functions and procedures.

example:

```
IF condition
THEN
ELSIF condition
THEN
ELSE
END IF;
```

* Functions make codes modular by making them more reusable, managable and readable.

* int func (int par) {
  }
  → Formal Parameter
  → Actual Parameter
  r=func(10);

* IN parameter is similar to those used in C/C++.

* OUT parameters can return multiple values.

* Fixing precision for parameters or return types is not allowed.

* PL/SQL is written at .SQL file, though a .txt file can be executed as well. However, for a .SQL file when writing the command to run the file, the extension is not needed to be mentioned (not the case for other extension).

example: @ absolutePath/t.txt
@ absolutePath/s
→ .sql file

12-11-2024
TUESDAY

OOP 4301
Faisal Hussein sir

* Base class' pointer can't call any functions not present in the base class' definition, otherwise type mismatch will occur.

* Correct syntax: void func() const override {};
this order must be maintained

* "override" keyword is a cautionary addition which matches the syntax for the base and the derived functions. However, the functionalities aren't

altered at all.

* PVF makes overriding mandatory.
* PVF doesn't have any descriptions inside the base class' body.
* PVF makes a class abstract and it can't be instantiated (no objects can be created).
* Each derived class that doesn't define the inherited PVFs also become an abstract class. However, if any of its parent class has the definition, the abstraction is removed.
* Shape *s[3];

  s[0] = new Rectangle(5, 10);

  s[1] = new Triangle(5, 10);

  s[2] = new Circle(0,0,5);
* Some languages allow turning a class abstract by using the keyword "abstract", not available in C++ (needs PVF).

* Destructors can be virtual, too, and often encouraged. For such a case, first the derived destructor will be triggered, and then the base one. It's highly preferable for long-term development scenarios.
* Virtual destructors call both the destructors (base & virtual) so that all DMAs are cleaned off. Hence, an exception is made in the compilation of virtual function's overriding. Otherwise, cleaning up base class' DMA would have been impossible.
* Virtual destructors' features apply for multi-level inheritance.
* Making the base class virtual solves the issue of diamond inheritance. In that case, we must make the inheritance virtual. This makes the derived class' copying once instead of multiple. In the opposite case, the ambiguity needs to be manually ~~remo~~ mentioning the inheritance path. However, virtual inheritance reduces the no.

of inheritance path down to 0.

* Friend functions can operate on objects of multiple different classes.
* Not having access to the class definition code denies the ability of declaring a friend function.
* Making a class friend inside the class body will allow the friend class to access the private and protected elements of the current class. Inheritance will maintain friendship for the derived members.
* Friend class doesn't inherit anything it only allows access ability.

13-11-2024
WEDNESDAY

COA1305
Md. Hamjajul Ashmafee sir

* SSDs are made of Flash NANDs.
  - → internal: (fixed) PCIe
  - → external: (portable) USB → can be shared

SSD compared to HDD:

I/O rate: ↑

Cost: ↑

Size: ↓

Robustness: ↑ ⇒ mecha. error %

Power: ↑

Heat: ↓

* SSD is an external memory and the CPU can't access it directly. Windows will let

CPU → File Explorer → I/O driver / I/O module (Hardware) → SSD (I/O device)

File System SW

* SSD and the host system (OS) are connected through interfaces.
* SSD's controllers manage the interface,

handle its underlying mechanisms and perform BIOS operation.

* The hardwired BIOS commands in SSD are known as Firmware.
* SSD combines multiple components to distribute the data.
* ECC is managed by the controller.
* SSD's data buffer/cache tries to address the speed mismatch between CPU and SSD as it acts as the medium of data transmission.
* Write operation (block-level) can't be performed directly on the components. Rather data from the desired location is copied to the cache (optional), it'll be replaced by the updated data, the previous data on the memory will be completely erased and then the update data will be inserted.

* SSD's problems:

1. Slower than CPU (specially write operation)
2. If no component has enough space left to store a data, a single data may be partitioned to multiple components (read cost ↑)

   Overprovisioning: When a data comes in, enough space is allocated so that future W operations can be performed there.

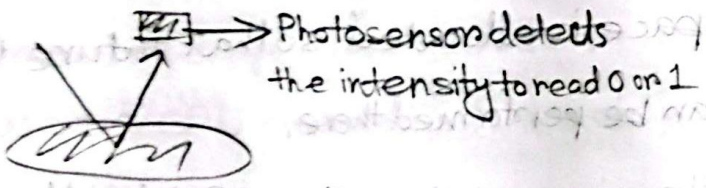
   Streaming: When the CPU is idle, the driver defragments the mem. so that data is pushed from one side & vacant space is formed on the other side.
3. Write count is fixed, crossing it will damage the mem. (tolerance, tamper & ICs). Can be 100000+. This count should be balanced and distributed to all components.

* Flash memory is of the same tech. as SSD.

* CD ⇒ Compact Disk
  - ⇒ works based on optical properties
  - ⇒ lights and rays are used (can be in form of lasers)

* Although all operations have a count due to physical limitations, WC is the main bottleneck of SSD.

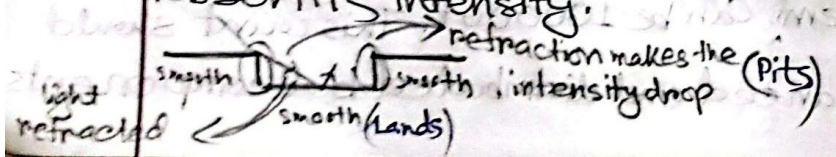
* Laser reads the CD.



CD's surface is scratched and roughened to differentiate 1 to 0 (smoother surface).

* DVD, advanced form of CD, uses UV Ray to make scratches.

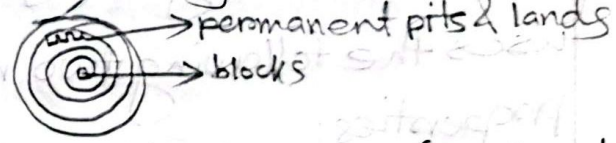
* CD's surface is made rugged to lessen its intensity.



* Reflection high ⇒ 0, less ⇒ 1

* CD's photoreflective surface is coated with Al or Au to make it shiny.

* HDD is a magnetic disk. Similarly, CD is also a disk made up of 1 single spiral track (centre-periphery).



The track is divided into blocks (similar to sectors).

* CD can be read-only, non-volatile.

* CD-R ⇒ CD-Recordable
  - ⇒ can be written once (permanent write)
  - ⇒ empty CD that can be burned
  - ⇒ WORM = Write Once Read Many

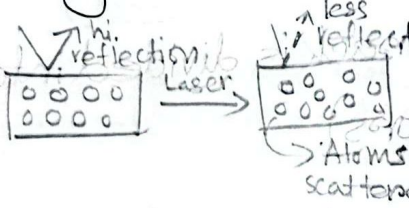
* CD-RW ⇒ CD-ReWritable
  - ⇒ data can be erased

⇒ same physical tech.

⇒ non-permanent W op.

* Special laser generates heat to imprint lands & pits on CD's surface. But CD-RW changes the property of the surface's layer (Dye Layer). It uses the following two molecular properties:

1. Crystalline



2. Amorphous

* CD's SYNC bytes signify the start of a block.

* CD can store data in 3 modes:

0 → remaining space is vacant

1 → remaining space stores ECC

2 → " " holds regular data (enlarging the data capacity)

* CD used to have max. 720 MB size and was gen. used to store audio files, MIN & SEC signify their lengths.

* CD's biggest advantage is portability and mass-production, ~~& its~~ but its access time is increased as it uses the sequential method

* CAV ⇒ Constant Angular Velocity

MZR ⇒ Multiple Zone Recording

HDD maintains these.

* CD follows CLV (Constant Linear Velocity).

DBMS 4307

14-11-2024 THURSDAY

ARMK sir

* PL/SQL is extremely structured.

* Anonymous blocks are generally used for debugging purposes.

* Named blocks are physically stored in the DB, stored in MetaData Repo. (Data Dictionary). Even the codes are stored in a dynamic view.

* IN parameters are read-only.
* OUT parameters allow us to return multiple items.
* ~~VARDESEL~~

```
X NUMBER(1,2);
Y VARCHAR(20);
BEGIN
  SELECT name, cgpa
    ~~INO~~ INTO Y, X
  FROM Student
  WHERE id=s_id;
END;
```

multiple variables can't be declared in one line

Customer: ID, Name | Transaction: ID, CID, Amount, Type(0/1) → withdraw / deposit, Date line

* CREATE OR REPLACE FUNCTION GETSTATUS(CustomerID IN NUMBER)
RETURN VARCHAR2
IS -- variable declarations
ST VARCHAR2(10);
V_D NUMBER(10,2) := 0;
V_W NUMBER(10,2) := 0;
V_B NUMBER(10,2) := 0;
BEGIN
SELECT SUM(Amount) INTO V_D
FROM Transaction
WHERE CID= CustomerID and
TpTran=1;
-- similar code for V_W
V_B=V_D-V_W;
IF V_B > [some value]
THEN ST:= 'VIP';
ELSE ST:= 'NORMAL';
END IF;
RETURN ST;
END;

* It's better for debugging purposes to use NVL on the return variable.
* dual is a virtual view. Anything can be calculate upon it. Hence, it should be used for instant queries.
* DBMS_OUTPUT.PUT_LINE('String' || 'Concatenation')
* Functions can be used in the 'select' operation. Aggragations can be thought of as built-in functions.
* select decode(~~TpTran~~ Type, 1, 'Deposite', 'Withdraw') ≡ (Type, 1, 'Deposite', 0, 'Withdraw', 'Wrong value');
↳ else part ↳ if 1 ↳ else if 0 ↳ else part

* If ~~both~~ 2 rows get the top position, both will be ranked 1, while the next entity holds the rank 3 (so that for n elements, the last element is ranked n).

14-11-2024
THURSDAY

OOP 4301
Faisal Hussein sir

* const mem. functions can modify objects static variables.

* "Constructor() {}" allows the creation of arrays of objects.

* Pointer doesn't create an object automatically

* RTTI ⇒ Run-Time Type Information
  ⇒ object for a class w/ at least a virtual function can have a function to show its data type (class name) at runtime
  ⇒ only available for polymorphic behaviour.
  ⇒ must be enabled in the compiler
  ⇒ not much standard or used

* To turn the base class pointer into derived class pointer, we need to use dynamic_cast.

* dynamic_cast requires the existence of at least one virtual function in the base class.

* Syntax:

lastPtr = dynamic_cast<last*> firstPtr;

14-11-2024
THURSDAY

DS 4303
Sabbir Ahmed sir

* Collision occurs when 2 keys are hashed to the same index.

* Separate Chaining = Open Hashing
  Open Addressing = Closed Hashing
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

  are the methods used for collision resolution.

* Separate Chaining is the simplest to implement and, ~~theoritcu~~ theoretically, will never run out of space.

* In order to initialize a hash table using the separate chaining method, all the pointers in the table to be null.

* Load Factor, λ = avg. length of lists

$$= \frac{\text{total no. of items}}{\text{tableSize}}$$

Max. value of $\lambda = \infty$ when $\to 1 \to \infty$

$0 \leq \lambda \leq \infty$

For chaining, λ can be ≥ 1.

For open addressing, λ's max. value is 1.

* Separate chaining can be useful if the no. of items isn't known beforehand, but it's not that popular in the practical landscape.

* Separate chaining's distribution is criticized. It also needs more memory for storing the links.

* Open addressing generally requires a larger table and λ is kept at ≥ 0.5.

* Open addressing ~~take~~ uses HelperFunctions to implement hashing.

* Linear probing may take much time, but it (due to primary clustering) ~~uses~~ utilizes all the vacant spaces.

* Re-Hashing is the process of re-mapping the elements to a table of ~2x size if the no. of elements close in on the table's size. This requires a new hashing functions.

* Avg. no. of cells examined before insertion in linear probing $= \boxed{\frac{1}{2}\left(1 + \frac{1}{(1-\lambda)^2}\right)}$

* For quadratic probing, when $\lambda \geq 50\%$, elements may come along that'll trigger infinite loop and not find an index despite there being free space.

* It's preferred to set the tableSize to be a prime number; even numbers are discouraged.

* Secondary clustering is very less likely to happen.

* Double hashing uses 2 hash functions. The 2nd hash function should never return 0, as then its

impact is nullified. Example: $\underbrace{\text{size} - (\text{val} \% \text{size})}_{\text{min. value} = 1}$

* Rabin-Karp Matching Algorithm uses hashing to find a certain pattern in a given string.

example: S → abcde

P → bcd

Brute Force approach leads to $O(\text{len}(S) \cdot \text{len}(P))$

R-K M A tries to bring it to $O(\text{len}(S) + \text{len}(P))$

$\cong O(\text{len}(S))$ [$\because \text{len}(S) \gg \text{len}(P)$]

It calculates the hash value of the pattern and then for each possible window. This is an example of Sliding Window Technique. If the hash values don't match, the pattern is absent. Otherwise, we need to check for collisions (for as long as necessary).

Rolling Hash Function is used to bring time complexity of the algorithm down.

The windows: abc
bcd
cde
...

Let the hash value be $\sum S[i] \times 10^i$

$\therefore$ abc $= a \times 10^0 + b \times 10^1 + c \times 10^2$ &

bcd $= \underbrace{b \times 10^0 + c \times 10^1}_{(ii)} + d \times 10^2$ → hash values of the string

$\therefore$ (ii) = (i) / base ⟹ in this case, 10

$$\therefore \boxed{H_i = \{(H_{i-1} - S_{i-1}) / \text{base}\} + S_i \times b^{\text{len}(p)-1}}$$

It doesn't maintain a hash function in the conventional meaning.

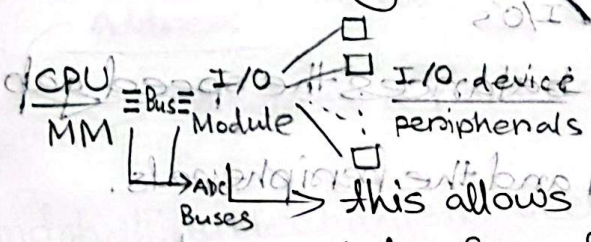
18-11-2024
MONDAY

COA4305

Md. Hamjajul Ashmafee sir

Chapter-7: I/O

* I/O can't connect to the internal computer systems directly. I/O module does it.



this allows a universal interface for all I/O devices protocols

CPU's commands are in the form of:

In/Out + Address/Port

I/O modules' external / internal interface

connects w/ I/O devices / buses

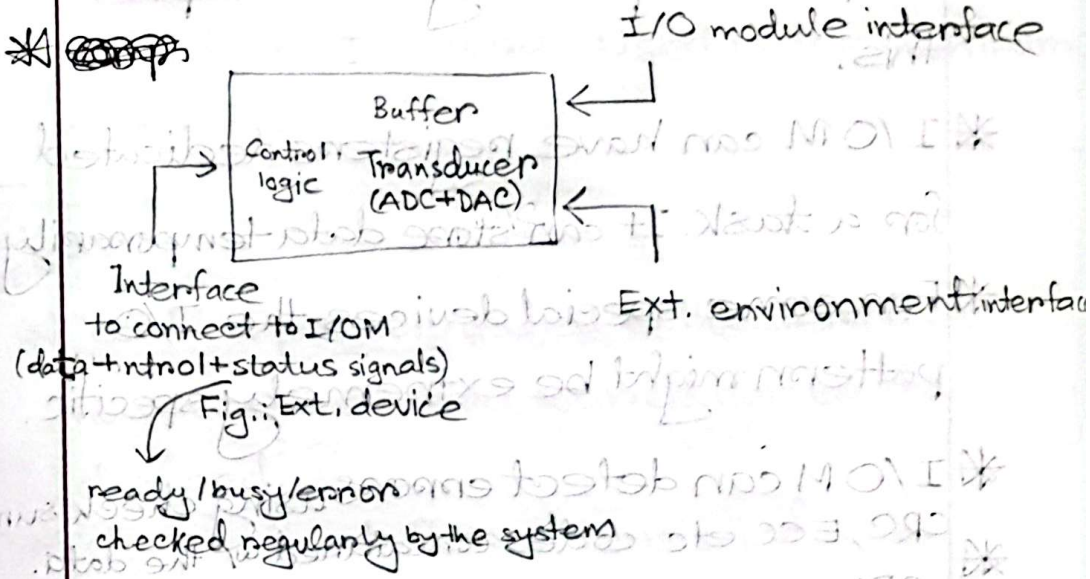
* I/O devices can be user-accessible (keyboard, mouse etc.) while others are machine-readable (storage, HDD etc.). Other devices are used for communication purposes (router, modem etc.).

* By storing CPU's / I/O's data at the module's buffer, it balances the speed gap betw the system and the peripherals.

* Router > CPU/MM ⇒ speed

* Disk (HDD/SSD) is an I/O memory device.

CPU contacts HDD or SSD via a module.

* 

* I/O logic controls the systems & their underlying components.

* CS — Control, Status, Data, Address → buses

I/O Module — [Same buses] — I/O devices

Address → to select a particular port/device

Handshaking Mechanism occurs after the address has been Broadcast to all the peripherals and the desired device will

respond. Req-Ack. signals accomplish this.

* I/O M can have registers dedicated for a task. It can store data temporarily.
* For some special devices, the I/O pattern might be extremely specific.
* I/O M can detect errors using check sum, CRC, ECC etc. codes concatanted w/ the data.
* CPU / Mem. $\rightleftharpoons$ I/O M $\rightleftharpoons$ Peripherals

Errors can be of 2 types: soft & hard

* I/O M can detect problems w/ the devices, also. It'll report back the diagnostics to the CPU.
* I/O Ms are of 2 types:

1. I/O controllers : only exchanges data
2. I/O Channel: have their own processor to operate on I/O data. Used in the mainframe computers.

19-11-2024
TUESDAY

OOP 4301
Faisal Hussein sir

* "cout" is pre-connected to the terminal, however their data source can be manually changed.
* "clog" is the buffered version of "cerr."
* The main base class of the stream hierarchy of streams classes.
* "cin", "cout", "cerr" & "clog" are objects of the "iostream" class.
* "istream" & "ostream" are classes.
* Classes can be of template style.
* Formatting flags is implemented using enums.
* setf $\Rightarrow$ Set Formatting Flag
* setf $\xrightarrow{X}$ unsetf
* Changing any of the independent flag doesn't

disturb the other flags.

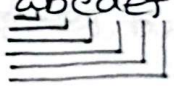
* "ios" flags are generally of the static (const) type.
* Bitwise OR operation allows the combination of flag variables. XOR operation allows us to turn off particular variables.
* "bitset<bitNo.>" operator prints out the result in binary bits.
* cout's flags can be modified as well.
* Manipulators can be visualized as functions which feed formatting instructions directly into a stream.
* Manipulators may or may not have arguments.
* setiosflags(ll) $\overset{X}{\longleftrightarrow}$ resetiosflags(ll)
* eof ⇒ End of File
* No stream buffer is associated for an invalid operation (spotted using badbit), although the program won't abruptly terminate.
* goodbit is turned on if all the error detecting bits are off.

19-11-2024
TUESDAY

DS 4303
Sabbir Ahmed sir

* Trie is efficient for string-based operations. It can search a string in O(L) TC where L is the length of the string.
* Prefix tree is a variation of trie.

Valid prefixes for: abcdef



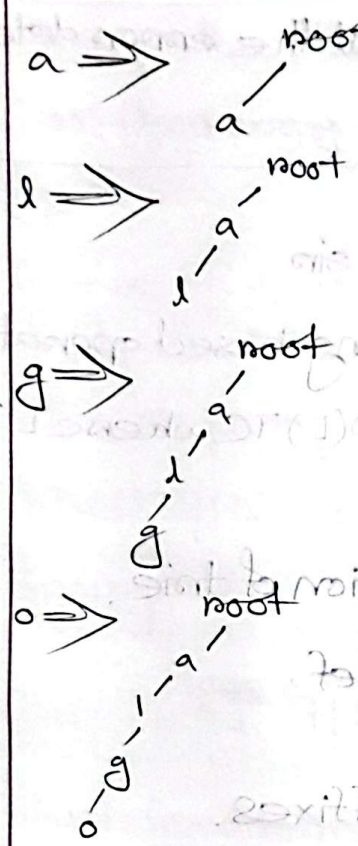
Opposit is true for suffixes.

* Trie is good at retrievals. (info.)
* Trie is similar to the tree DS where each node has pointers equal to the number of unique characters (26 for lower characters only).

node* next[26]; ⇒ all initialized to nullptr signifies absence

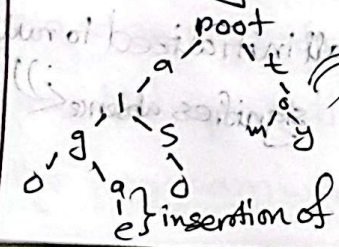
Insertion of 'algo':

root



All other 25 pointers for all the nodes are set to nullptr.

Insertion of 'also': [nothing happens for 'a' & 'l']



* Trie doesn't physically store the letters.
* Checking one pointer among the 26 possible ones is enough to spot the presence or absence of a particular word.
* "endmark", by default false, is a flag signals the completion of insertion of a word.
* "prefixCount" is defaulted to 0 and is incremented as we pass a node (except for the root node),
* Trie is good for prefixes, but can't work w/ substrings.

* <u>Trie pseudocode:</u>

```
struct Node {
    bool endmark
    Node* next[26]
    int prefixCount
    Node() {
        endmark = false
        for i in range 26 ⇒ next[i] = nullptr
        prefixCount = 0.
    }
}
```

For N words, O(NL)
↳ avg./max. len

TC: O(len)
SC: O(N×L×alphabetSize)
↳ no overlaps

```
void insert(str, len) {
    Node *curr = root
    for (i=0 upto len) {
        index = str[i] - 'a'
        if (curr->next[index]) {
            curr->prefixCount++
            curr = curr->next[index]
        } else {
            curr->prefixCount++
            curr->next[index] = newNode()
            curr = curr->next[index]
        }
        curr->endmark = true
    }
}
```

TC: O(L)

```
bool search(str, len) {
    Node *curr = root
    for (i=0 upto len)
        idx = str[i] - 'a'
        if (!curr->next[idx]) return false
        curr = curr->next[idx]
    if (curr->endmark) return true ①
    else return false ⓘ
}
```

① + ⓘ ⇒ return curr->endmark

⇒ advanced recursion ⇒ TC = O(NL×AS)

⇒ does the backtracking

```
void display(Node *curr, str[], level) {
    if (curr->endmark)
        str[level] = '\0'
        print(str)
    for (i=0 upto 26)
        if (curr->next[i])
            ch = 'a' + i
            str[level] = ch
            display(curr->next[i], str, level+1)
}
```

⇒ lexicographically sorted by nature

* Trie can store binary numbers as well.

20-11-2024
WEDNESDAY

COA 4305
Md. Hamjajul Ashmafee sir

* I/O can be:
programmed
interrupt-driven
DMA ⇒ Direct Memory Access

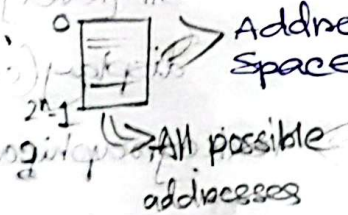
CPU executes this via

* I/O ins^n ≠ I/O command → generated by the CPU → then passed to the peripherals
  - ~~faster~~ first / comes before
  - generated by the programmer
  - stored in MM

CPU sends I/O commands to the peripherals.

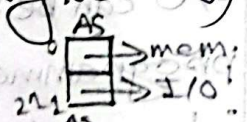
* ① Addressing Modes are used to identify I/O devices via the address bus.

If the add. bus has width of n lines, each address will be n-bytes long and there can be 2^n distinct addresses.

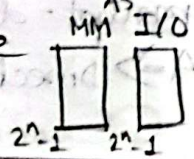


There are 2 options for addressing:

1. The space can be shared (some cells reserved for storing memory locations) ⇒



2. Separate usage of the space



⇒ Isolated I/O / I/O Mapped I/O

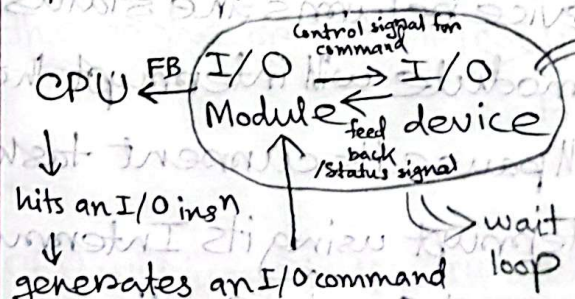
It allows for more options.

* ~~No. of diff. ins^n~~ Read = I/P write = O/P
  = Load = Store

These 2 ins^n are enough for the shared space, and as the addresses are different, CPU can distinguish its desired device.

* R-W for mem.
  I/P-O/P for I/O

CPU needs 4 diff. type of ins^n to interact w/ the separate space.

* 

→ For programmed I/O, CPU is engaged with these operations (waiting)

During the waiting time, the CPU regularly checks the peripheral's status, even before receiving the final signal (when it'll terminate).

* I/O commands are of 4 types: Control (specifying the command), Test (to check the status), Read & Write.

* For interrupt-driven I/Os, CPU doesn't engage with the execution of the command & offloads it to the module, rather it leaves the scenario and continue its regular tasks, which allows parallel multi-tasking on the CPU's end.

When I/O device returns the status signal, the module will interrupt the CPU which will pause its current task to serve the interrupt using its Interrupt Handler/Interrupt Service Routine.

* DMA devices can directly access the MM bypassing the CPU, which leaves after making the handshake. Here, multiple tasks are treated as a single one which the CPU can initiate or terminate. It takes the feedback (returned as an interrupt signal) after the completion of the task.

* I/O module acts on behalf of the devices.

* IH/ISR (a function) responds to an interrupt. CPU may've even shifted to a diff. program altogether. Its response steps are:

1. Suspend the curr. task.
2. Store the curr. context of the registers in the cache or the MM. ⟹ it has a portion dedicated to handling interruptions known as System Stack
3. CPU will jump to the IH address (the 1st address) provided by the OS. ⟹ to differentiate separate interruptions, some generic, some specific ⟹ gets loaded to the PC

This was the hardware side of things.

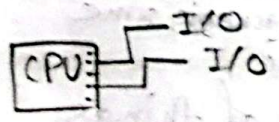
How ISR responds;

1. Save the Stack Address in a register, so that CPU can return there
2. Interrupt-specific tasks are done
   — identifying the module & device
   — does some tests
3. Returns the stack address
4. Updates the PC so that CPU can continue its paused task

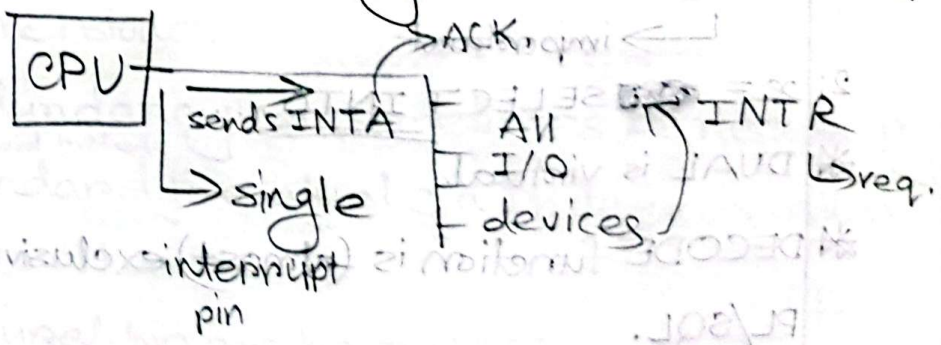
* Identifying the interrupting device is known as the Identification Problem as the returning (module) signal may not carry the port address.

Possible soln:

1. Multiple interrupt lines by connecting diff. devices to diff. devices (limited growth)



2. Software Code is generated before choosing the ~~general~~ specific ISR for the device. The general ISR is executed first to identify the device so that the device-specific ISR can be triggered. It runs a test command one-by-one by going through the address list of I/O devices to look for the ACK signal. Hence, it's slow.
3. Hardware poll = daisy chain = vector interrupt



INTA goes sequentially and if the device is interrupting, it'll respond, otherwise pass it down to following devices.

INTA is generated for each INTR CPU receives.

This process is similar to encoder's mechanisms.

If a device is interrupting, it returns an Address Vector specific to the ~~def~~ device.

Port Address

21-11-2024 THURSDAY

DBMS 4307

Prof. Dr. Abu Rayhan sir

* 2 common mistakes in PL/SQL:

1: x := 10;
→ important

2: x = [redacted] SELECT INTO...

* DUAL is virtual.

* DECODE function is (almost) exclusive to PL/SQL.

* Triggers need not be called explicitly. It's fired automatically. They occur when some pre-defined event occurs.

* There's no way to call a trigger.

* Trigger = condition + Action

* Triggers are of 3+ types:

1. System-level trigger → will be fired once for a statement, regardless of the no. of records affected

2. DML - trigger: Row-level & Statement-level / Transaction (no "FOR EACH ROW" command)

→ trigger will be fired for each row ("FOR EACH ROW")

* Designing a DB is algorithmic. For most of the algorithms, functional dependency is the key/cornerstone.

* Redundancy can sometimes be necessary.

* Standard (mathematical method) to evaluate a DB design is called Normal Forms.

* Natural Join can be expensive.

* Expression of functional dependency:

deptName → budget

deptName → independent var.

budget → dependent var.

→ determines

* Func. dependency does a mapping similar to what a func. might do.

* Normal Forms are the mathematically formal methodology to determine func. dependency.
* Lossy Decomposition incurs incorrect data set.
* Lossless Decomposition: $R = R_1 \cup R_2$

21-11-2024
THURSDAY

OOP4301
Faisal Hussein sir

* "<fstream>" class works w/ disk file I/Os. The object of the class must be created separately.
* File objects aren't needed to be closed explicitly in C++.
* Space is a delimiter for strings.
* "good" func. doesn't return 1 for "eof" character.
* File I/O is formatted.
* Binary data is memory efficient.
* RAM works in binary.
* Files are made up of character texts.
* Binary data are stored as a whole, instead of character-by-character.
* string's memory access is dynamic.

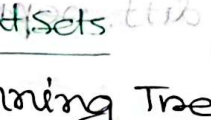
* "fclose", first writes the data stored in the buffer to the file and then closes it. It's safer for navigating file I/O crashes or errors.
  ofstream.close

21-11-2024
THURSDAY

DS 4303
Sabbir Ahmed sir

Disjoint Sets

* Minimum Spanning Tree is an application of graph.
* 2 nodes are connected if they're reachable through a path.



A, C ⇒ connected
B, D ⇒ disjoint

Disjoint Sets can help us figure out —
— connected components
— whether the graph contains any cycle
— minimum spanning tree

Rules:

1. Those who're connected/friends w/ each other will belong to the same set.
2. no. of dis. sets = no. of subgraphs/clusters

3. Every dis. set will have a "representative" among them for the set to be recognizable.

4. If 2 items have the same representative, they may belong to the same set.

* Items having diff. representatives belong to diff. sets.

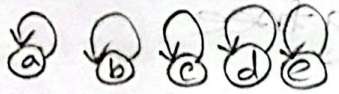
* 3 operations of dis. sets:

1. Make Set ⇒ create separate sets w/ each items (initializes the DS)
2. Find | Union-Find ⇒ returns the representative of an item
3. Union ⇒ connects the representatives of 2 sets to form a new set

* Simulation of dis. set DS:

a b c d e

Make Sets:



→ Is a & b connected (i.e, are their representatives the same?)?

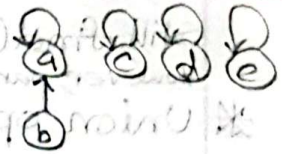
Find(a) = a
Find(b) = b | as they're not equal, they're disjointed

can be dictated by preferred policies

→ Union(a, b)

Previously,
Find(a) = a Find(b) = b

Now,
Find(a) = Find(b) = a

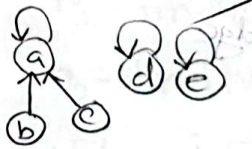


→ Union(c, a)

Variants of Union operation — [Union-by-rank (most common) / height (of the rep.) ; Union-by-size]

rank(a) = 1 > rank(c) = 0 ⇒ c will be connected to a

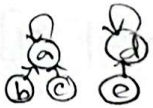
size(a) = 2 > size(c) = 1 ⇒ c will be connected to a



to keep the set's size smaller

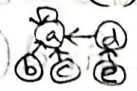
If a tie occurs, the 2nd element will be connected to the 1st one.

→ Union(d, e):



→ Union(c, e): Find(c) = a, Find(e) = d

rank(a) = rank(d) = 1



size(a) = 3 > size(d) = 2

Still, find(e) = a. Now, b & e are connected.
However, parent(e) = d.

* Union op. takes O(1) TC
* If 2 same ranked sets are united, the resultant set's rank increases.
* "Find" is a recursive function.
* For union(a,b), if their representatives don't match, the entire set's representative moves to new one. Had they matched, no changes in the result then. This (a redundant edge) confirms the existence of cycle in the graph.

```
makeSet (vertices[]) {
    for (i=0 upto N)
        parent[i]=i
}
find(u) {
    if (parent[u]==u) return u;
    else find(parent[u])
}
union(u,v) {
    parent[find(v)] = find(u)
}
```

* "find" operation grows costly as the size increases
* Path Compression adds an extra layer of efficiency in the "find" & "union" func.s after introducing -by-rank policy. It updates the parent array into the representative array.

```
unionByRank (u,v) {   default = 0
    r1 = find(u)
    r2 = find(v)
    if (rank[r1] > rank[r2])
        rep[r2] = r1
    else if (rank[r1] < rank[r2])
        rep[r1] = r2
    else
        rep[r2] = r1
        rank[r1]++
}
```

* Path compression is an algorithm

```
find(u) {
    if (parent[u] == u) return u
    else
        parent[u] = find(parent[u])
        return parent[u]
}
```

17-2-2025
MONDAY

COA 4305
Tanjajul Ashmalee sir
Ch.-8: OS

* OS is a piece of program or software which provides the platform for other applications to run. It's an interface that controls other applications. It sits between h/w and other s/ws. It connects the user to the computer.
* OS has 3 basic functionalities:
1. It's an interface
2. Resource Management
3. Program Scheduling
* OS is stored in a portion of the MM. It's a special type of controller (as it's not isolated from the system which it supervises).
* OS is executed by the processor. When another program is run, the OS gives in its control (it leaves the processor for the desired program to take control of the processor). After the program reaches its end command, the OS takes back the control of the processor.
* OS makes the sequence in which the programs will be executed and allocate resources to them
* OS makes a system efficient & user-friendly
* Other functionalities of OS:
1. Program execution using instruction cycle
   ↳ lines of code / lists of instructions
   Makes ins. cycle efficient by improving the conditions of execution
2. Program creation (developing apps)
3. Access control (managing how resources should

be shared)
→ files/data
→ I/O devices' scheduling
→ processor
→ shared buses
etc.

If the full system is shared/public (multiple users), OS will partition resources to diff. users through authentication. It also checks the instructions to determine which ones can be run.

4. Accounting ⇒ keeps the users' log
5. Error detection & response
   ↳ can be of H/W or S/W → can be compile-time or runtime
   can be logical or unauthorazided access

The program won't crash as OS will stop it.

Every component of a system is accessed through an address.

Potential responses: stopping the app, reinitiating it, reporting it to CPU, etc.

* Users directly interact with app. programs to access hardware components. API carries the messages bet^n. OS & apps.
  ↳ comes before libraries & utilities/drivers
  ↳ machine-oriented code

Execution H/W (processor) first receives the instructions.

* I/O devices are accessed through buses.
* OS $\xrightarrow{\text{instructions}}$ ISA $\xrightarrow{\text{ins.}}$ CPU
  ISA ⇓ Instruction Set Architecture
* Only the kernel/nucleus part of the OS is stored permanently in the MM. The rest remains in the secondary storage (example: C:/) and only called upon when necessary.
* <u>Types of OS:</u>

# of programs executed at a time
- → uni-programming (sequential)
- → multi-programming
  - ↳ for a single core, interrupt can be used for such OSes

interaction
- → batch (everything placed together & passed to the processor with no user interaction during executio
- → interactive

* A computer can have no OS. In that case, a programmer must toggle w/ the console.
  ⇓
  If an error occurs, the entire system will crash.

~~[illegible]~~ Certain time limits for programs are robustly maintained

Each program must load all the necessary environment.

Simple Batch OS was the 1st solution that was applied to these problems.

Multi-program Batch OS introduced interruptions to let CPU switch betw programs (multitasking).

Time Share fixed the amount of time for resource sharing.

$$\xrightarrow{P_1,\ P_2,\ P_3,\ P_1,\ P_2,\ P_3,\ P_1}$$
$$0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6$$

Interactive OS lets users access the system while an execution is ongoing.

* Multiprogram utilizes system resources much better.

* For simple batch OSes. kernels are also called Monitors.

18-2-2025
TUESDAY

CSE4301(OOP)

Dr. Rafsanjany Kushol sir

Ch.-15: STL

* Containers can be of 2 types: sequence and associative
  - sequence → deque, array, vector, list etc.
  - associative ↳ map, set etc.

* end() moves the ~~it~~ iterator past-the-end point of a container

* Pointers can be used for iterators.
* ~~source~~ source() returns an iterator if a certain patter or sequence is found. The iterator sits at the first index of the matched pattern in the container.
* merge(src1, src1+x, src2, src2+y, dest);

  It works for sorted sequences and stores the concatanation ~~is~~ in a ~~sord~~ sorted order, too.
* "greater<double>()" is a comparator function which is an example of operator ~~functio~~ overloading. It sorts a container of data type "double" in descending order.
* iterator-begin() (arr pointer = index

* Adding "-if" to algorithms allows us to use comparator functions in algorithms other than sort(). "for_each()" allows the same for containers (the original values are not stored or retained).
* Vectors can be filled in using iterators or pointers (can be initialized, too). It has insert() & erase() functions as well.
* "list<dataType>" creates a doubly linked list.
* Deques are sequential containers & can thus be accessed index-wise.

18-2-2025
TUESDAY

CSE4303 (DS)
Sabbir Ahmed sir

* Trie is best for searches, specially for strings.
  - ↳ Prefix Tree
    - ↳ as it works on the prefixes of strings, specially
  - ↳ O(len(str))
    - ↳ not dependent on the size of the database.
  - ↳ Re trie val

* Insertion operation takes O(len(str))
* Disjoint Sets (Union-Find DS) are used for dynamic group elements and checking connectivity among them. Items can be connected directly (an edge between them) or indirectly (if both the elements are in the same set).
* If an edge connects vertices that already belong to the same set, it ensures the existence of a cycle in the graph.
* find(element); ⇒ returns its representative
union(x, y); ⇒ connects x's rep. to y's
makeset(x); ⇒ (x)
* setUnion(i, j) {
    ri = find(i), rj = find(j)
    if(ri != rj) {
        parent[rj] = ri;
    }
}

* Union-by-Rank method improves the efficiency of the find() function as it restricts the ~~height of th~~ growth of the height of the tree. In this style, rank only increases if the nodes/reps being united had the same rank.
* Default rank can start from 0.
* Path Compression optimization directly connects each of the nodes to its rep. instead of going through the parents' lines to make future find() operations more efficient.

```
find(u) {
    if(parent[u] == u) {
        return u;
    }

    return parent[u] = find(parent[u]);
}
```

* Disjoint sets can generate mazes.

18-2-2025
TUESDAY

CSE4307 (DBMS)
Prof. Dr. Abu Rayhan sir
Ch.-7: Normalized Design (part-1)

* Normal Forms is the standard method to evaluate a design.
* Some redundancies can be necessary.
* Larger schemas are gaining traction to avoid expensive natural join operations.
* Functional Dependency: $\alpha \rightarrow \beta$ (α determines β)
Everytime α is found, β must be associated (but vice versa is not true), otherwise the schema is invalid. This is used to detect redundancies.
* Decomposition $\Rightarrow$ splitting a large schema into smaller ones
  - Lossy (bad design)
  - Lossless (good design)
* $\boxtimes \Rightarrow$ Natural Join

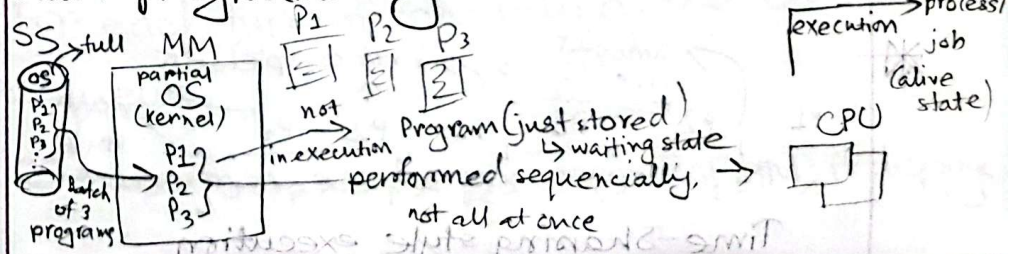
$r \subset r_1 \boxtimes r_2$
proper subset
proper superset

* $r \subset r_1 \boxtimes r_2 \Rightarrow$ Lossy Decomposition
$r_1 \boxtimes r_2 = r \Rightarrow$ Lossless Decomposition
* If all the instances are Legal Instances (valid for not violating any constraints), it's a Legal Database.
* $F \Rightarrow$ set of functional dependencies
* $\alpha, \beta, \delta, \ldots \Rightarrow$ attributes
* Closure refers to the implied constraints. It turns F into F^+.
* $\alpha \rightarrow \beta,\ \beta \rightarrow \delta \Rightarrow \alpha \rightarrow \delta$ [law of transitivity]
* $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$ | Lossless Decomposition

19-2-2025
WEDNESDAY

COA 4305
Md. Hamjajul Ashmafee sir

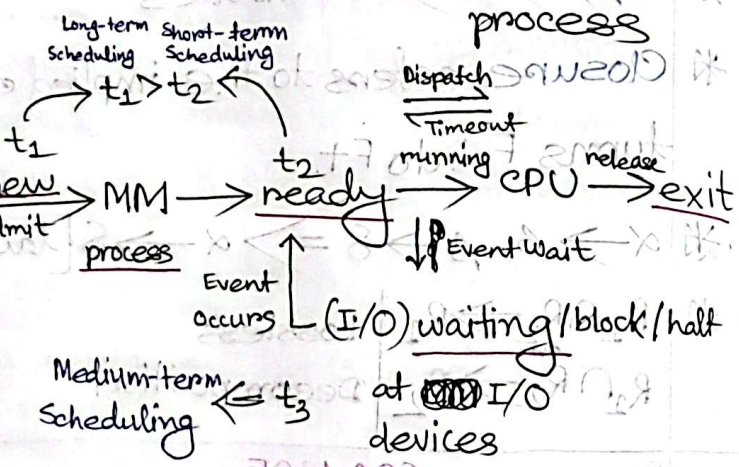
* Multi-programming:



I/O devices cause waiting time

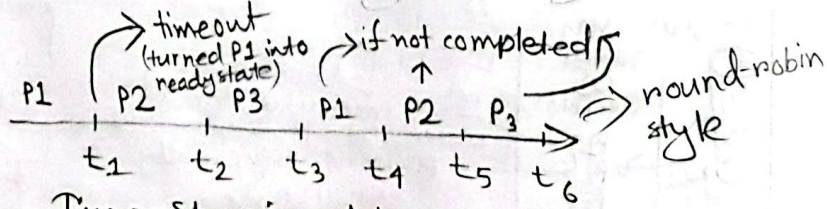
* CPU switches betw processes (in multi-programming OSes) to reduce waiting time. So, the execution isn't strictly sequential.

* A process can have 5 states: New, Ready, Running, Waiting, Exit => life cycle of a process



A process waits after its I/O operation is done at MM.

*



Time-Sharing style execution

* OS maintains a profile for each process (Process Control Block - PCB) to remember where to continue execution from for an incomplete process. Program Counter (PC) holds the data for the current instruction to be executed. It's the distance of that line from the process' beginning (Memory Pointer).

Context Data holds the register values.

* OS increases the priority for a program that takes less CPU cycles.

* PCB is stored in MM.

* During the timeout phase, the OS is in the CPU.

* For I/O devices sequences, I/O scheduling is used.

* For each type of scheduling, the OS maintains a queue.

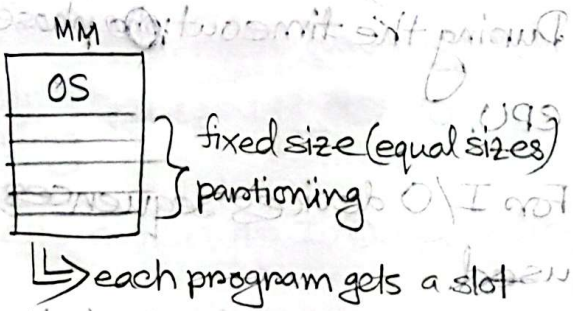
* OS ~~mang~~ manages the processes. At MM, it deploys

Swapping, Partitioning, Paging, VM, segmentation etc. methods to increase efficiency.

* Degree of Multi-Programming, n ⇒ no. of processes which can be run simultaneously.

* OS maintains Intermediate Queue in the SS to keep track of incomplete executions. It ~~[illegible]~~ needs an interrupt from I/O devices to stop. The processes remain in waiting state in the SS. This system is called Swapping.

* Partitioning:



Unequal Size Partioning is a better method. /Variable

Dynamic Partioning is further advanced as it solves the issue of residue. No. of partitions remain fixed to decrease the size of address. This doesn't apply to dynamic partitioning, however.

Dynamic Partitioning wastes space due to fragmentation. This is solved using De-Fragmentation. However, this changes the physical address (not the logical one though). MMU maintains the sync.

* Paging is similar to caching.

MM stores the pages of the same size in a frame.

Page size = Frame size

Pages can be stored at any place/frame of MM depending on vacancy.

* PC = Offset

* Paging eases the task for PCB.

* For paging, ~~offset = page no. + PC~~ physical address = frame no. + offset ~~/address~~ ↳ how far the page has been executed

* Pages occupy frames in random order. Page table (according to the size of the process) stores the value of ~~frame address~~ physical address, page-wise. It's stored in the MM.

* Paging allows for larger processes to be executed using swapping bet^{w} MM & SS. This system is known as Virtual Memory (VM).

* Segmentation specially place the repeated sequences at their appropriate access points.

20-2-2025
THURSDAY

DS 4303
Sabbir Ahmed sir

Graphs

* $G=(V,E)$
  - $E \rightarrow$ edges
  - $V \rightarrow$ vertices

$V=\{v_1, v_2, \ldots v_n\} \Rightarrow |V|=n$

$e \Rightarrow (v,w);\ v,w \in V$

* Graphs
  - $\rightarrow$ Directed (order matters for edges)
  - $\rightarrow$ Undirected (order doesn't matter)

* Max. no of edges in an undirected graph $= \frac{V(V-1)}{2} = O(V^2)$

$E \leq \binom{|V|}{2}$

* Degree of a vertex = no. of $\underbrace{\text{adjacent vertices}}_{\text{Neighbours}}$

* Subset of edges & vertices of a graph creates its Sub-Graph.

* Simple Paths don't contain repetitions (except for maybe the 1st or the last one).

* Length of a path = no. of vertices − 1

* Graphs
  - $\rightarrow$ Connected
  - $\rightarrow$ Unconnected

* Edges can have weights for Weighted Graphs, as opposed to Unweighted Graphs.

length of a path = Σ edges in the path

* Edges have directions for Directed Graphs.

* Max. no. of edges in directed graphs =

$|V| \times (|V|-1) = O(V^2)$

$E \leq \underbrace{2\binom{|V|}{2}}_{|V| \times (|V|-1)}$

* Vertices in directed graphs have inDegree (sources if 0) and outDegree (sinks if 0).

* Vertices ≡ Nodes
* Directed graphs can be strongly or weakly connected.
* Directed graphs can be weighted or unweighted. (v_i, v_j) and (v_j, v_i) may not have the same weight.
* DAG ⇒ Directed Acyclic Graph
  ⇒ contains no cycles
  ⇒ used in topological sorting
* Binary-Relation List is the most inefficient method to store a graph.
  ⇒ $O(|E|)$
* Adjacency Matrix uses more memory. It uses a 2D array.
* K_n ⇒ Complete Graph
  $K_{n,m}$ ⇒ Biparptite Graph
* Topological Sort works based on dependencies.

* Topological ordering may not be unique, It can start from any source node.
* Topological sorting can be used to detect cycles through circular dependency.

24-2-2025
MONDAY

COA 4305
Hanyajul Ashmafee sir

Ch. 12: Instruction Sets

* Programmer for S/W ≡ Designer for H/W
* ISA is the collection of raw instructions
* HLL ⇒ ISA ⇒ Signal
  ↳ High-Level Language
  ISA → stored in MM, executed by the processor
  Signal → stored temporarily in IR (Instruction Register)
  Signal → raw commands to H/W
* MP ⇒ Micro-Processor
* MP can't know beforehand what the operation entails before it executes the command.
* IS ⇒ Instruction Set
  ⇒ range of commands MP understands
  ⇒

op. code	address of data (operand)

Ins. format

* We may require 4 addresses for an op.:

2 for binary / 1 for unary, 1 for result & 1 for next ins^n.

→ can come from MM, register or peripherals → port address to be.

* Address → implicit → no need ~~for~~ mentioning directly in the ins^n (used in sequential executions)
→ explicit

* Data could be fetched directly as a part of an ins^n (example: const values like pi).

* HLL → Assembly (a symbolic lang.) $\xrightarrow[\text{(through compiler)}]{\text{binary}}$ ISA (object code) → MP

* In Assembly, 'A[num.]' is written to denote address

* X=X+Y ⇒ HLL

Assembly:

Load X, ACC	ACC ← X
Add ACC, Y	ACC ← ACC + Y
Store ACC, X	X ← ACC

* $Y = \dfrac{A \overline{\oplus} B}{C + (D \times E)}$

→ 3 address: Op. Code, src, dest

they can be switched based on the architecture

Op. Code, Dest., S_1, S_2

Mult.	Y	D	E
Add.	Y	C	Y
Sub.	T	A	B
Div.	Y	T	Y

→ temp. var. (if no inputs can be changed)

→ will work as the 2nd src. before op.

→ 2 address: Op. Code, Dest., Src.

Load	Y A
Sub	Y B
Load	T E
Mult.	T D
Add.	T C
Div	Y T

→ temp. var.

→ 1 address: Op. code, Dest. (ACC implied)

↳ multi-purpose var. (can also work as src.)

Load E	ACC ← E
Mult. D	ACC ← ACC × D
Add C	ACC ← ACC + C
Store Y	Y ← ACC
Load A	ACC ← A
Sub B	ACC ← A − B
Div Y	ACC ← ACC / Y
Store Y	Y ← ACC

→ 0 address: A stack will be used to maintain & access the data. All loading & storing will occur at the top of index of the stack.

OOP 4301

Dr. Rafsanjany Kushol sir

25-2-2025 TUESDAY

* less<dataType> ⇒ ~~descending~~ ascending order

greater<dataType> ⇒ ~~ascending~~ descending order

* Syntax of sets:

set<dataType, comparatorFunc.>

↳ optional, default: ascending order

* syntax of maps:

map<key, value, comp.Func.> :: iterator → to declare an iterator

↳ optional, default: ascending order of the keys

* Associative containers are more efficient in searching.

* List-like DTs are better for frequent insertions & deletions.

* Syntax for accumulate func.:

accumulate (begin, end, startingValue, func.);

↳ default: plus<dT>()

25-2-2025 TUESDAY

DS4303
SabbirAhmed sir

* BFS & DFS are elementary graph algorithms.
* BFS tree breeds the shortest possible path (smallest no. of edges).
* BFS results into a tree.
* For BFS, $O(|V| + |E|) = O(V+E)$ if we use adjacency list

$O(V^2)$ if we use adjacency matrix

* F => Forward Edge (gray → black) → descendant (started after me, finished before me) → existence of cycle
B => Back Edge (gray → gray)
C => Cross Edge (gray → black) → no overlap
T => Tree Edge (gray → white)

25-2-2025 TUESDAY

DBMS4307
ARMK sir

* Loseless decomposition works based on keys.
α → intersection attribute
* If $R_1 \cap R_2 = PK(R_2)$, it can be rewritten as:
$R_1 \cap R_2 \rightarrow R_2$ (FK(R_1)) → applicable for lossless decomposition.
↳ determines

* independent var. $\xrightarrow{\text{determines}}$ dependent var.
↳ functional dependency
* Rules of a schema are enforced by the means of FD over all data during the entirety.
* $\alpha \rightarrow \beta$ doesn't necessarily imply $\beta \rightarrow \alpha$.
↳ (sort of) a key
* FD can be semi-trivial too.
* NF => Natural Forms
=> has degrees (1, 2, 2.5, 3, 4, ...)
* $\alpha \rightarrow \alpha$ => Trivial FD (always true & obvious, no need to check)
$\alpha \rightarrow \beta$
$\beta \subseteq \alpha$
↳ $(\alpha, \beta) \rightarrow \beta$
* $\alpha \rightarrow \beta$, $\alpha \cap \beta = \emptyset$ => Non-Trivial FD (needs to be checked)
* $\alpha \rightarrow \beta$, $\beta \not\subseteq \alpha$, $\alpha \cap \beta \neq \emptyset$ => Semi-Trivial FD (needs to be checked)

example: ID, Name → Grade, Name

* $\alpha \rightarrow \beta$, $\beta \rightarrow \gamma$ => $\alpha \rightarrow \gamma$ => Transitive FD (proved from the rules of inferrence)

* F → set of FDs in a schema

F+ → includes the implied FDs alongside the given ones

→ closure of F

$F \subseteq F+$

↳ algorithms determine NF based on F+

* Secondary rules of Armstrong's axioms /Additional are there for the purpose easing the process but are not necessary.

* Reflexivity: Dept., Hall → Dept.

Augmentation: Dept. → Budget

⇒ Dept., Est. → Budget, Est.

* Armstrong's axioms are not necessarily symmetrical or reversible (i.e., if $\alpha \rightarrow \beta$, both $\beta \rightarrow \alpha$ ~~and~~ or $\beta \not\rightarrow \alpha$ can happen depending on the context and data)

* $A \rightarrow BC$, $B \rightarrow E$, $CD \rightarrow EF$

Does $AD \rightarrow F$ holds?

~~$A \rightarrow BC$~~ ~~$B \rightarrow E$~~ ~~$B \rightarrow E$~~ ~~$CD \rightarrow EF$~~

~~$AB \rightarrow BCE$~~ ~~$BCD \rightarrow EF$~~

~~⇒ $A \rightarrow CE$~~

~~$CD \rightarrow EF$~~

~~$ACD \rightarrow CEF$~~

~~⇒ $AD \rightarrow EF$~~

$A \rightarrow BC$ → $A \rightarrow B$ ⇒ $AD \rightarrow BD$

→ $A \rightarrow C$ ⇒ $AD \rightarrow CD$

$CD \rightarrow EF$ ⇒ $AD \rightarrow EF$

∴ $AD \rightarrow F$ holds

Ch.-7: Normalized DB Design (pt. 2)

* X+ ⇒ Attribute closure

⇒ ind. var. → dependent var. (s)

⇒ if α can become $\alpha+$, it's a super key

27-2-2025 THURSDAY

OOP4301

Kushol Rafsanjani sir

* STLs use Function Templates as it allows for flexibility in terms of data types.
* Instantiation occurs only once for a particular data type and is reused.
* Function templates can work w/ objects as well.
* For FTs, arguments' data-types must match.
* A FT can deploy multiple types of template argument. It can cause the compiler to create more codes though.
* Templates are similar to macros.
* Macros don't perform/validate any type checking, thus ~~using~~ datatypes are a weak point for macros compared to templates.
* Classes can have templates, too.

27-2-2025 THURSDAY

COA4305

~~Sabbir Ahmed sir~~

Hamjajul Ashmafee sir

* Types of data/operand:

1. Arithmetic/Numbers (signed, unsigned, float, packed etc.) → binary representations
2. Characters (stored in binary ASCII codes)
3. Boolean/Logic (stored in bits)
4. Address ⇒ treated as unsigned int.

↳ processor should know 2 things: type/mode (register or memory) and actual/effective (where operands are stored) conveyed in logical/virtual address

* Problems faced by computers when dealing w/ numbers: range limitation & precision (for floating-point numbers).
* Bits can represent any types of operands.
* Types of operations:

Arithmetic

Logical

Data transfer ⇒ processor mostly just calculates the addresses (src., dest. implicitly no. of bits)

I/O

Conversion

System control

Transfer control

* Modes of data transfer → Mem. / Register
  → calculate EA (Effective Address)
  → convert to LA (Logical Address)
  → check for free cache

* 2 types of arithmetic operations: ~~unary~~ unary and binary. May involve data transfer.

* 2 types of logical operations: ~~unary~~ unary and binary. (shift, compliment, rotate, arithmetic shift etc.)

* Both arithmetic & logical operations use ALU.

* 1101 $\xrightarrow{\text{Left Shift}}$ 1010 $\xrightarrow{\text{LS}}$ 0100

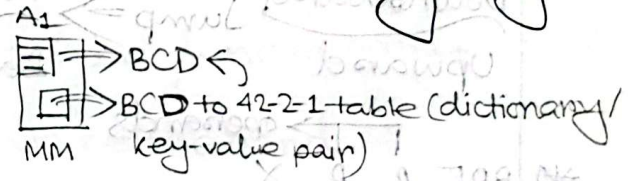
  $\xrightarrow{\text{Right Shift}}$ 0110 $\xrightarrow{\text{RS}}$ 0011

  $\xrightarrow{\text{Right Rotate}}$ 1110

  $\xrightarrow{\text{Arithmetic Shift}}$ doesn't change the sign (normal shifts may change the sign of signed numbers)

* Logical or arithmetic operations change the Flags (Status Registers) in CPU to see the changes brought in by the operations.

* Conversion/Translation is a special type of operation which maintains mapping tables to change the format of a data/number w/o doing any calculations.



Thus this operation takes in 2 addresses: which address to translate and where the table is stored. Implicitly, the no. of bits to be translated should be mentioned, too.

* System Control is a priviledged operation which only the OS can offer the CPU.

* Transfer Control allows multi-tasking as CPU transfers control if it comes across some instructions (like: Branch/Jump, Skip, Call (func.) + Return, etc.)
  → breaks the sequential exec. of a program by changing the PC's value

* Ins^n style: Branch/Jump X → Target Address
  ↳ PC is updated to X

Jump can be conditional or unconditional

examples: conditions, loops, func. calls, etc.
(applied on the last operation) (modular programming)

Downward / Upward Jump ⇒ Forward / Backward Jump

* BRE R_1, R_2, X → operands
  ↳ equal?
  ↳ Target address
  ⇒ if ($R_1 == R_2$) jump X;

* Skip ⇒ PC += 2
  ⇒ ignores exactly one ~~inere~~ ins^n
  ⇒ should be conditional
  ⇒ ISZ → skip-if-zero

* "Return" address ⇒ address of func. call + 1
  ⇒ PC's value

* Stacks, instead of registers or memories, are used to keep track of func. calls.
  ⇒ pushed PC for a call
  ⇒ popped to PC for a return

* Similarly 2 stacks are used to store parameters (one for pointer). Mem. isn't an option as the parameters can be dynamically sized & cannot be known beforehand.

* WORD ⇒ no. of bytes
  ⇒ unit of mem.

example:

Index:	0	1	2	3
WA: 184	12 (0001 0010)	34	56	78
BA:	184	185	186	187

⇒ array

MSB → LSB
↳ byte
↳ one order (Big Endian)

LSB → MSB can be another order to store fragmented data. ⇒ Little Endian

Indexing will dictate the order (if given) as the order can't change the indexing. Same technique

can be applied for bits.

OOP4301
Ratsanjanij Kushal sir

4-3-2025
TUESDAY

* Exceptions are errors which occur during run-time caused by classes.
* In a class definitation mistake, error can occur w/o a func. being explicitly called.
* Exception class throws the exception. It's created as a member func. of the problematic class.
* Catch Block / Exception Handler ⇒ shows an error message
* Exception classes can inherit other such classes, otherwise it's empty inside as they're abstract classes, although they can have member func.s & variables.
* Exception classes can accept arguments if designed in that way.

* "bad_alloc" is a built-in exception class in C++ to detect failed memory allocation using the "new" keyword.

DBMS 4307
ARMK sir

4-3-2025
TUESDAY

* Attribute closure allows us to find super keys.
* DA is static, PA is dynamic. $\alpha = DA \cap PA$
  ⇒ $\emptyset$ means no other CK
  ⇒ else go through the replacement process (RP)

example: $R(A,B,C,D,E)$ FD: $A \rightarrow B$ (I)
$D \rightarrow E$ (II)

— $DA = \{B, E\}$

$AB\cancel{C}DE^{+} = \{A, B, C, D, E\}$ (I)

⇒ $ACD\cancel{E}^{+} = \{A, B, C, D, E\}$ (II)

⇒ $ACD^{+} = \{A, B, C, D, E\}$

RP ensures that the right side always remains the same when the left side changes.

ACD^+ is a RS & a SK.

↳ has 2^3 subsets

↳ has 2^3-2 proper subsets (PSS)

↳ $\emptyset$ & ACD^+

ACD^+ is a CK as none of its PSS is a SK.

$\underbrace{\{A,C,D\} \cap \{B,E\}}_{\alpha} = \emptyset \Rightarrow$ no need for further RPs as no more CKs exist here

PA DA

* Having circular dependency normally ensures the existence of more CKs.

* $R(A,B,C,D)$ FD: $A \to B$, $B \to C$, $C \to A$

- $ABCD^+ = \{all\}$ PSS $A^+ = \{A,B,C\}$

$\Rightarrow ACD^+ = \{all\}$ $D^+ = \{D\}$

$\Rightarrow AD^+ = \{all\}$ $\therefore AD^+$ is a CK

↳ SK

$DA = \{A,B,C\}$ $PA = \{A,D\}$

$\alpha = DA \cap PA = \{A\} \neq \emptyset \Rightarrow$ more CKs exist

Let's try to find $X \to A \Rightarrow$ RP

$AD^+ \xrightarrow{C \to A} CD^+$

↳ SK & CK

$C^+ = \{A,B,C\}$ | No PSSes are SKs

$D^+ = \{D\}$

$CK_1 = AD^+$, $CK_2 = CD^+$

$\alpha = \{A,B,C\} \cap \{A,C,D\} = \{A,C\}$ already tested

Let's find $X \to C$ which is $B \to C$

$CD^+ \xrightarrow{B \to C} BD^+ \to$ SK & CK

$\alpha = \{A,B,C\} \cap \{A,B,C,D\} = \{A,B,C\}$

Next iteration, we'll get AD^+ again which will close the loop. The CKs we got are AD^+, BD^+ and CD^+.

* $R(A,B,C,D)$, FD: $AB \to CD$, $C \to A$, $D \to B$ ($AB \to C$, $AB \to D$)

$ABCD^+ = AB^+$ $\alpha = \{A,B\}$ $X_i \to A$, $X_j \to B$ $\Rightarrow BC^+$, AD^+ (BC: $AB = AB$; AD: $AB = AB$)

later down the road, we'll get CD^+ as well

5-3-2025
WEDNESDAY

COA 4305
Hamjajul Ashmafee Sir

Ch.13: IS-Addressing Modes & Formats

* OpCode ⇒ name/code of the ins^n
* Ins^n: [ ] 20 . Address Space: 16 GB MM
  Possible addresses, $2^1 \times 2^{30}$ =

  ↓ ↓ ↓
  OpCode (4-bit) | Src... (5/6-bit) | Dest (5/6-bit) ⇐ Hashing ⇒ 2^{31} ⇒ 31 bits

Addressing modes/types allow us to accomodate the entire address space into an instruction.

→ **Immediate AM**: fast, but limited space.

A → Mem. address
R → Register "
EA ⇒ Effective " ⇒ exact location of where the operand is stored
(X) → Content of X
↳ can be A or R

Op. Code	Value(s)

↳ no need for fetching from MM

→ **Direct AM**:

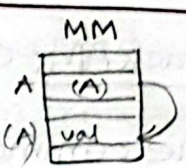
Op. Code	A

A → MM → val ⇒ fetch from MM & its content will be updated

→ **Indirect AM**:

Op. Code	(A)

↓
EA



It covers the full range of address space, but it takes 2 fetches. OS handles the out-of-bound error.

→ **Register AM**:

Op. Code	R

⇒ fast as reg.s are closer to CPU

Ideally, the no. of registers in a system is 8-32 (requires 3-5 bits); much efficient.

→ **Register Indirect AM**:

Op. Code	(R)

⇒ EA = (R)

→ **Displacement AM**: Direct + Reg. Indirect

EA = A + (R)

Op. Code	A + (R)

Its application has diff. variants:

↳ Relative AM: 1 address is implicit (PC) = (R) (add an offset to that) ↳ A should 2's compliment BN (as both + & - values are possible)

↳ Base-Reg. AM: (B) + A
↳ can be assigned a reg. to B

↳ Indexing AM: A + (IR)
↳ Index Register
A → mem. Array address

→ Stack AM: 0 bits required for addressing. The command would retrieve the top, anyway.

* 3 mod bits are added before every address to indicate the AM. Another way is to embed this inside the op. code.

* If EA = (X), size of X no. of addresses can be accessed ⇒ 2^X. If it's always stored in the cache, it significantly reduces the mem. access time. This block is called Page-0 (the page that comes before the procedure). This can only cause 1 page fault/cache miss (which is page-0 is absent).

↳ mem. size

size (page-0) ≥ 2^A

* EA can have multiple levels of indirection.

example: EA = ((...A...))

Multi-pointers use this type of Multi-Level/Cascading Indirection. A single bit is used to denote whether that address holds a value or another address.

* Indexing Displacement AM can be: (A)+(IR)
Pre-Indexing

or A+(IR)
Post-Indexing

or EA = A+(IR)
(IR) = (IR) ± 1
Auto-Indexing

* N = I + J + K

Load I	I → ACC	Converted to Hexadecimal/ Symbolic/Binary code or Assembly Code
Add J	ACC = ACC + J	
Add K	ACC = ACC + K	
Store N	ACC → N	

6-3-2025 THURSDAY

DS4303

Sabbir Ahmed sir

* Prefix Sum doesn't work w/ non-invertible operations. It works based on pre-computation, hence it doesn't allow updating the values.

Sparse Table

* Range min./max./GCD queries in O(1).

* Good for non-invertible operation; doesn't allow changes. Segment Trees are suitable for those;
* Table ⇒ 2D Array
* no. of columns in a sparse table for an n-sized array: $\lfloor \log_2(n) \rfloor$

12-3-2025
WEDNESDAY

COA 4305

Md. Hamjajul Ashmafee sir

Chapter-14

* Let there be k stages for an inst where each inst takes τ_i time.

$\therefore$ avg. time for each stage, $\tau = \cancel{\frac{\sum_{i=1}^{k} \tau_i}{k}}$

$= \max_i(\tau_i) + d$

τ_{max}

Latch Time (time to transfer register data)

* $T_{n,k} = \{k+(n-1)\}\tau$

total time needed

no. of pipelines

no. of inst

$$\text{Speedup} = \frac{T_{bp}}{T_{ap}} = \frac{n \times k \times \cancel{\tau}}{\{k+(n-1)\}\cancel{\tau}}$$

before pipeline

after pipeline

$$\therefore \boxed{\text{Speedup} = \frac{nk}{k+(n-1)}}$$

$k\uparrow \Rightarrow$ Speedup $\uparrow$

$n\uparrow \Rightarrow$ Speedup $\uparrow$

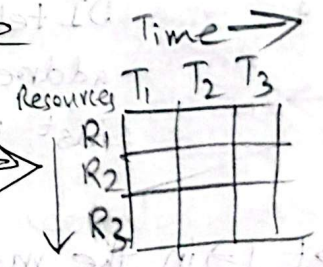
* Hazards ⇒ conditions that stop inst.s from progressing in a pipeline

⇒ 3 types: resource, data & control / branch

↳ solns: no. of resources ↑, make delay (sequential inst) or reservation table)

* F ⇒ Fetch
I ⇒ Instruction
O ⇒ Operand
D ⇒ Decode
W ⇒ Write
E ⇒ Execute

* Reservation Table ⇒



For an incoming inst, this determines whether the needed resources are available.

* Data hazard:
i → x = x+1 (W)
i+1 → y = y+x (R)
→ pipeline will try to E these parallely, but x must be updated

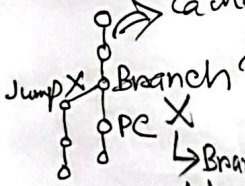
first.

i →	R	R	W	W
i+1 →	R	W	R	W
	NH	DH	DH	DH

↳ No Hazard

→ must involve updating values

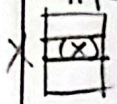
* Branch inst.s can cause control hazard (if-else is an example). This determines whether PC should be followed or not.



→ EI will determine the branching path.

↳ Branch Address DI tells CPU the branching address & the fact that this inst. is a branching inst.

MM(X) → Branch Target



DI → EI → in the meantime, CPU executes a certain branch. If it's not selected, the results are flushed.

* 5 ways to solve CH:

1. Multiple Branch: If CPU is powerful enough, it can precompute all the branches, but it can overwhelm the no. of pipelines.

2. Prefetch Branch Target: Temporarily fetch (X) and stored in PR (Prefetch Register) waiting to be executed.

3. Loop Buffer: Same as Direct Memory Access. (Direct Mapping) → follow the tag & if matched, return the cache value

PC ⇒ not taken
X ⇒ taken
} Cache determines

4. Branch Prediction: 5 ways of doing this —

(a) Follow history of the branch inst from ISA from previous encounters.

i. Always taken
ii. Always not taken
iii. Op. Code (s probability of hit)
} Static

Dynamic

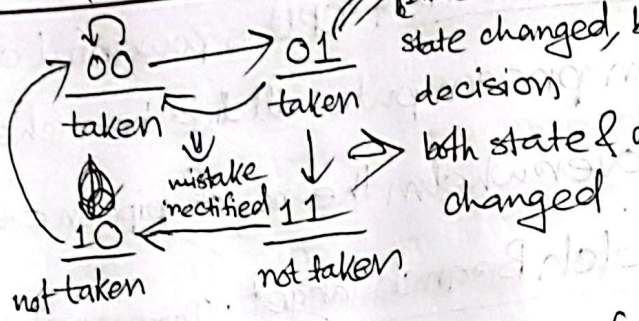
iv. Switch Bit (compare branch inst. v/s its history in 1 bit) → not executed immediately
↳ changed if the decision is confirmed to be wrong

History stored in 2 bits, allowing 4 options.

continue the previous story (decision = confirmation)

example:

history bits →



if mistake, decision ≠ confirmation

state changed, but not decision

both state & decision changed

(Dynamic) iv. Branch Prediction Table: Extra info. added to the history table. Offers 3 options: PC history table, and X.

* PC ⇒ sequential execution

Dynamic v. Delayed Branch: Applied if the branches merge at the end. So, this method delays the branched inst. & executes the safe/common ones first.

13-3-2025 THURSDAY

DBMS 4307

ARMK sir

* Prime × Non-Prime = Non-Prime

↳ attributes found in the discovered CKs

* Negating condition of 2NF: PSS(CK_i) → non-PA. Failing of it causes misunderstanding of object boundaries, attribute relationships and indicates a bad schema design.

* Negating condition of 3NF: non-PA → non-PA

* NFs try to respect object boundaries to clean up duplications/repititions & unnecessary redundancies.

* ~~Assenting~~ 3.5NF has an asserting condition: for a non-trivial FD $\alpha \rightarrow \beta$, α must be a SK. 3.5NF is rarely needed in the real world, ensuring upto 3NF should suffice.

13-3-2025 THURSDAY

OOP 4301

Dr. Rakshanjany Kushol sir

Ch.-13: Multifile Programs

* Multi-file programs are used to better organize the codes, specially if the project is large. This allows us to capitalize IP (intellectual property) by sharing the object/binary files while keeping the codes' logics secrets.

* Linker connects all the necessary files. It adds on the object files.
* .cpp files are separately compiled.
* int var; ⇒ both a definition & a declaration
* A global variable can only be declared in only one place in a program/file. It can't be assigned in other files.
* "extern" is a keyword in C/C++ which allows a global var. or func. or something to be declared at one file and defined elsewhere.

example:

File-A:	File-B:
int globalVar;	extern int globalVar; globarVar = 3;

can't be initialized as compiler then ignores the "extern" characteristic

* "static" variables/functions are only visible in their respective files. Same goes for "const".
* Multiple declarations in a file is OK.
* To use a func. defined in another file, we must declare it first.
* A class must be defined inside a file for it to make use of the class, because the compiler must the data types of all the operands it's going to work with. Header files come in to help here.
* Header files should just declare and not define. This is true for selling coding IP.
* If a ~~defin~~ declared block is defined at multiple places, it'll cause multiple definition error.
* Including the same header file more than once causes the multiple include hazard. IFNDEF can prevent potential errors of this kind.
* "namespace"'s codes can be divided into multiple parts & scattered across the code.
* A code can use multiple namespaces. It includes all the definitions inside the namespace, however they can only be used after the "using namespace ␣;" line has been written.

* "typedef" can create aliases for ~~var~~ data types, class names and struct names.

13-3-2025
THURSDAY

DS 4303

Sabbir Ahmed sir

Fenwick Tree

* Helps w/ ranged queries.
* Introduces updating arrays in the prefix sum's concept.
* ≈ Binary Indexed Tree
* Fenwick tree is good for point updates while segment tree is good for ranged updates.
* The size of the array for Fenwick tree cannot be changed.
* prefixSum(n) ⇒ O(log n) in Fenwick tree
* Steps for prefixSum(11):

$11 \rightarrow 10 \rightarrow 8$

$\hookrightarrow (101\cancel{1})_2 = (10\cancel{1}0)_2 = (\cancel{1}000)_2 = (0000)_2$

→ keep ~~cutti~~ resetting the LSB

↓ stop cascading downward

* LSB of a number = (num) & (-num)

example: $11 = (1010)_2$

2's complement → 0110

& 1010

0010

* Range queries ⇒ $O(2\log_2 n)$

$\approx O(\log_2 n)$

* For point updates, we need to cascade upward (now we add, instead of subtract, the LSB value).
* Fenwick tree is constructed using & for calculating prefix sums.
* Fenwick tree is applied through an array but can be visualized as a tree.

18-3-2025
TUESDAY

DBMS 4301

ARMK ~~Rabsangkaushal~~ sir

Ch. 14 - Indexing

* Indexing is used for faster searching.
* DBMS maintains a tree underneath to handle indexing & storing data.

* Performance is generally more important or valuable than space/memory.

* Clustering Index ≡ Primary Index
  - ⇒ similar to PK
  - ⇒ one table can have only one
  - ⇒ based on PK, created automatically

Secondary Index ⇒ separate from the table
- ⇒ more flexible, can be larger
- ⇒ needs to be created manually

* Multilevel Indexing ⇒ Outer Indexing + Inner Indexing

* Bitmap Indexing is efficient for categorical data (i.e., the no. of distinct options for a data type).

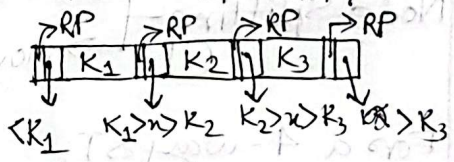
Length of vector for bitmap indexing = no. of records

* Underlying DS of DBMS: B+ Tree

In DBMS' BST, each node contains the key. No. of keys and no. of children are not fixed. In BST they'd have been 1 & 2 respectively. For a tree, however, the values remain constant.

For a m-way Search Tree, each node will have (m−1) keys & m children.

example: a 4-way ST:



Each of the ~~middle~~ pointers will have a Record Pointer (RP) associated w/ them pointing at the tuples themselves.

* These trees should be balanced following some rules: m-way ST + rules = B Tree

1. $\lceil m/2 \rceil$ children (other than the root)

2. Root should have at least 2 children
3. Bottom-up
4. Balanced (will automatically occur if the previous 3 are satisfied)

Nodes splitting is the most common operation in this regard.

* For a m-way ST, a node can store upto or below (m-1) values/keys.

* Node Splitting → move it upward (can be left or right biased)
  → move it sideways

example: For a 4-way ST:-

[10, 20, 30] ← 40

⇒ [10, 20, 30, 40]

MU (right-biased)

⇒ 10, 20 — 30 — 40

9-3-2025 WEDNESDAY

COA4305

Hamjajul Ashmafee sir

* CU. v/s 1st gen. IAS ⇒ control signals

* Bus → MBR ⇒ read (C_5) seen in subcycles!
  ↳ fetch, indirect

* ICC
  Fetch — 00 ⇒ $p'q'$
  Indirect — 01 ⇒ $p'q$
  Execute — 10 ⇒ pq'
  Interrupt — 11 ⇒ pq

$\therefore C_5 = p'q't_1 + p'q \cdot t_2$ ⇒ H/W implementation

[...] ⇒ Control Word in Control Mem. (Microprogramming Implementation)
⇓
executed by a general circuit

* Mem. access + I/O access ⇒ Bus Access (Machine Cycle)

* ⊔⊓ ⇒ Clock Pulse / T-State

* ALE = Address Latch Enable
  ⇒ bus gets to know it's an address if ALE is activated

20-3-2025
THURSDAY

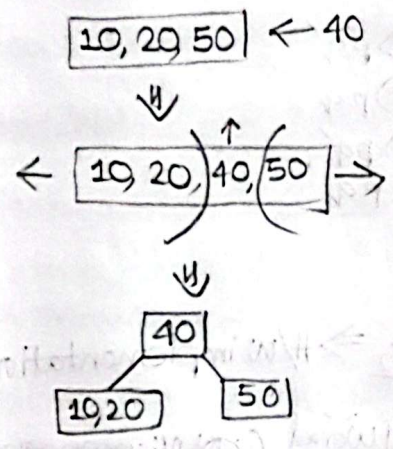
DBMS 4307
ARMK sir

* Every node's keys must be sorted.
* ~~Subtrees~~ Node-splitting occurs after the node's been sorted.

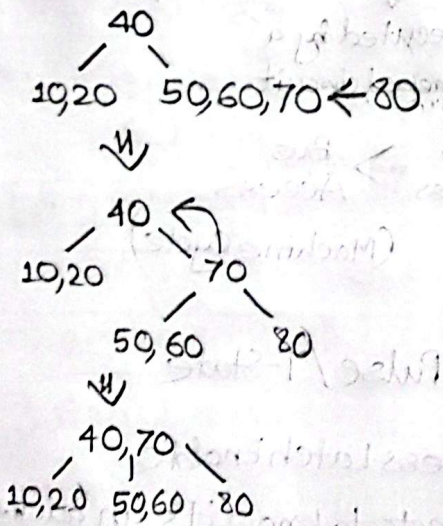
* example

K=3

t_1:

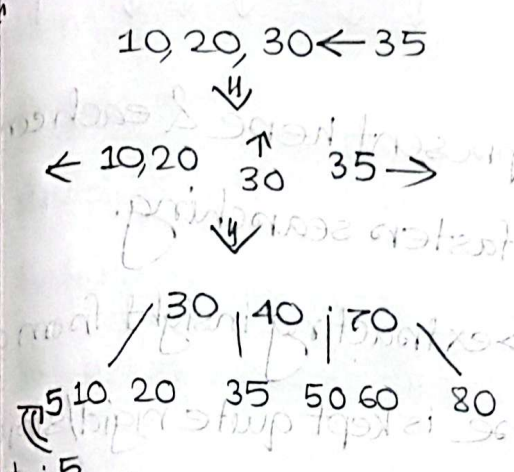


t_2: 60, 70, 80



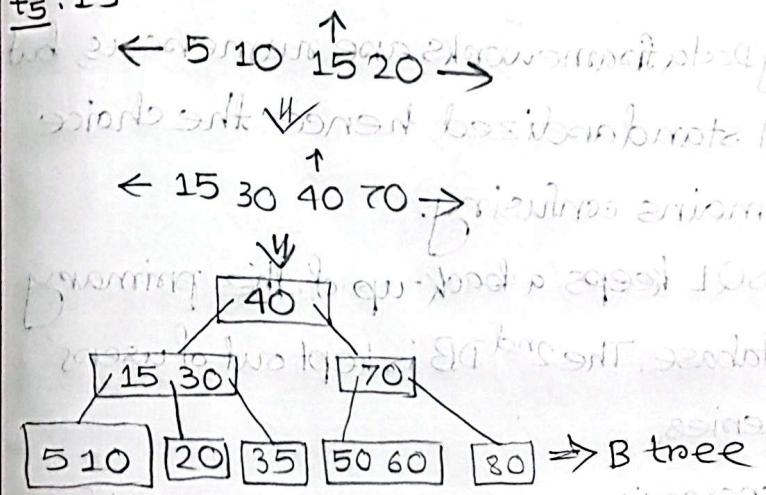
That's how the tree gets recursively balanced.

t_3: 30, 35



t_4: 5

t_5: 15



Convert this to a B+ tree which works on the leaf-level. As it's not a complete tree, we need to add the record pointers and redraw the figure:

B+ tree:

5, 10 → 15, 20 → 30, 35 → 40, 50, 60 → 70, 80

(RP)

All nodes are present here & each contain a RP to enable faster searching.

* Data Analytics ⇒ extracting insight from data
* RDBMS' structure is kept quite rigid/strict. Big Data model is more flexible/fluid.
* Big Data frameworks are numerous, but not yet standardized, hence the choice remains confusing.
* NoSQL keeps a back-up of the primary database. The 2nd DB is kept out of users' queries.
* Microservices are aggregated in JSON format as it provides flexibility in designing.

17-3-2025 MONDAY

COA 4305

Hamjajul Ashmafee sir

Ch.-20: Control Unit Instructions

* Mem. returns instructions via the data bus.
* A program is a set of inst. Each inst can be divided into 4 stages:

Fetch – Indirect – Execution – Interruption

These steps can be further divided into Micro-Operations (μOp)/Programming. If 2 or more successive μOp.s are independent of each other, they can be grouped together to save time.

* The atomic tasks for "fetch" inst (executed in one cycle):

PC → MAR (stored in the address bus)

Mem. (R) → MBR

MBR → IR
PC+1 → PC
] can be grouped together

takes 4 unit times (not considering grouping).

* Inst: | Op Code | A | ... → address pointing to another address

* Atomic tasks for "indirect" stage:

IR(address) → MAR

MM → MBR

MBR → IR(address)

* Atomic tasks for "interrupt" stage:

Current values and states of the registers are pushed to a stack. PC will be updated to activate ISR.

t_1(R): PC → MBR(stack)

↳ control Unit(CU) knows the address

t_2(W): MBR/stack → MAR

t_3: MBR → MM

t_4: ISR address → PC

can be grouped together

* Execute subcycle can be complex or diverse and depends on the specific/exact type of operation it's supposed to accomplish.

example:

ADD R1 X

IR(address) → X

IR → ADD

R1 → (reg.)

X → (MM)

IR(address)/X → MAR

MM → MBR

MBR+R1 → MBR

increment ⇒ (X)++

skip zero

example: ISZ X → mem. location

⇒ if $(X)_{new}$ == 0, skip next inst. (PC++ again).

PC++ before to reach here, PC++ after it to skip the inst

Branch (Jump)

example: BSA X → Jump to X

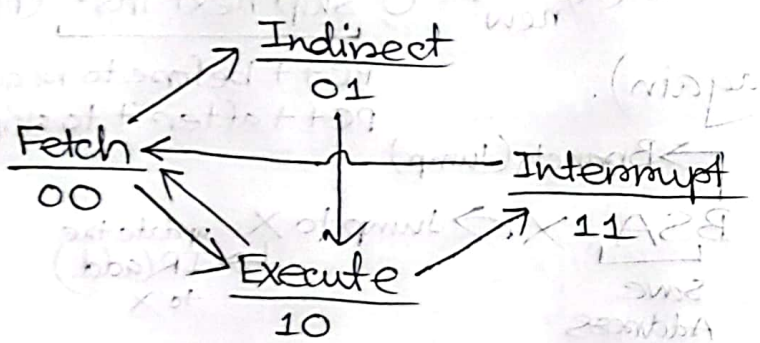
BSA: save Address

X → update the IR(add.) to X

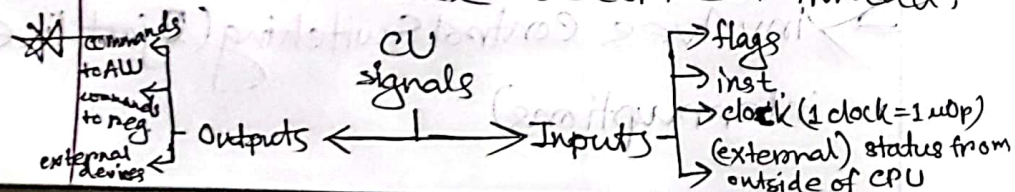
⇒ can indicate func. call (func. starts from the location X)

⇒ involves control switching (just like interruptions)

* At the beginning of the fetch cycle, among all the registers, only PC's value is known.
* Control Unit makes the sequence of μOp.s. Executing them is the job of the CPU and the ALU.
* CU should know the present state to order the sequences. Hence, the op.s are codified using the Instruction Cycle Code (ICC).



* "Execute" subcycle's op. code must first be decoded to choose the correct thread.



* CU emits control signals.
* Control signals are transmitted along the data paths.
* ALU performs, among other things, increment-decrement operations.