Vidyavardhaka College of Engineering, Mysuru

Autonomous Institute, Affiliated to VTU

Accredited by NBA | NAAC with ‘A’ Grade

Digital Design and Computer Organization

Module 4

3rd SEM

B-sec

Our Vision: “VVCE shall be a leading Institution in engineering and management education enabling individuals for significant contribution to the society”

Topics to be covered …

- Input/output Organization.
  - Accessing I/O devices
  - Interrupts
- Direct Memory Access.
  - Bus Arbitration
- Memory System
  - Speed, Size and cost
  - Cache Memories

Input / Output Organization

- The basic features of computer is its ability to exchange data with other devices.

- This Communication capability enables a human operator, e.g, a keyboard and a display screen to process text and graphics.

- we make extensive use of computers to communicate with other computers over the Internet and access information around the globe.

Accessing I/O Devices

- A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement.

- The bus enables all the devices connected to it to exchange the information.

- Typically , it consists of three sets of lines used to carry address, data , and control signals.

- Each I/O device is assigned a unique set of addresses.

- When the processor places a particular address on the address lines, the device that recognizes this address responds to the commands issued on the control lines.
- The processor requests either a read or a write operations , and the requested data are transferred over the data lines.

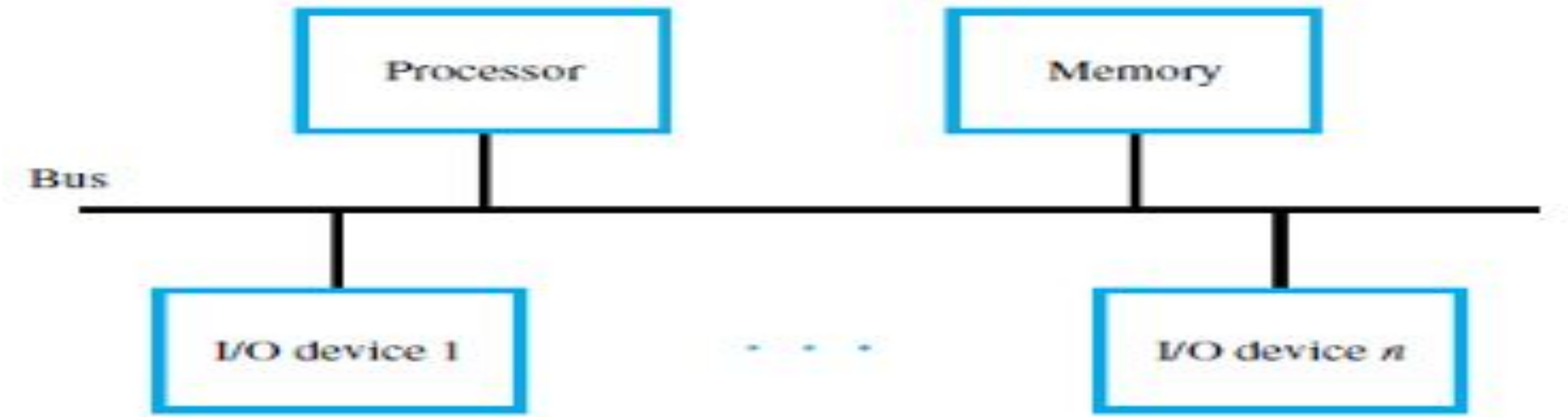


Figure 2

- When I / O devices and the memory share the same address space, the arrangement is called Memory – mapped I/O.
- With memory – mapped I /O , any machine instruction that can access memory can be used to transfer data to or from an I / O device.

E.g., if DATAIN is the address of the input buffer associated with the keyboard, the instruction

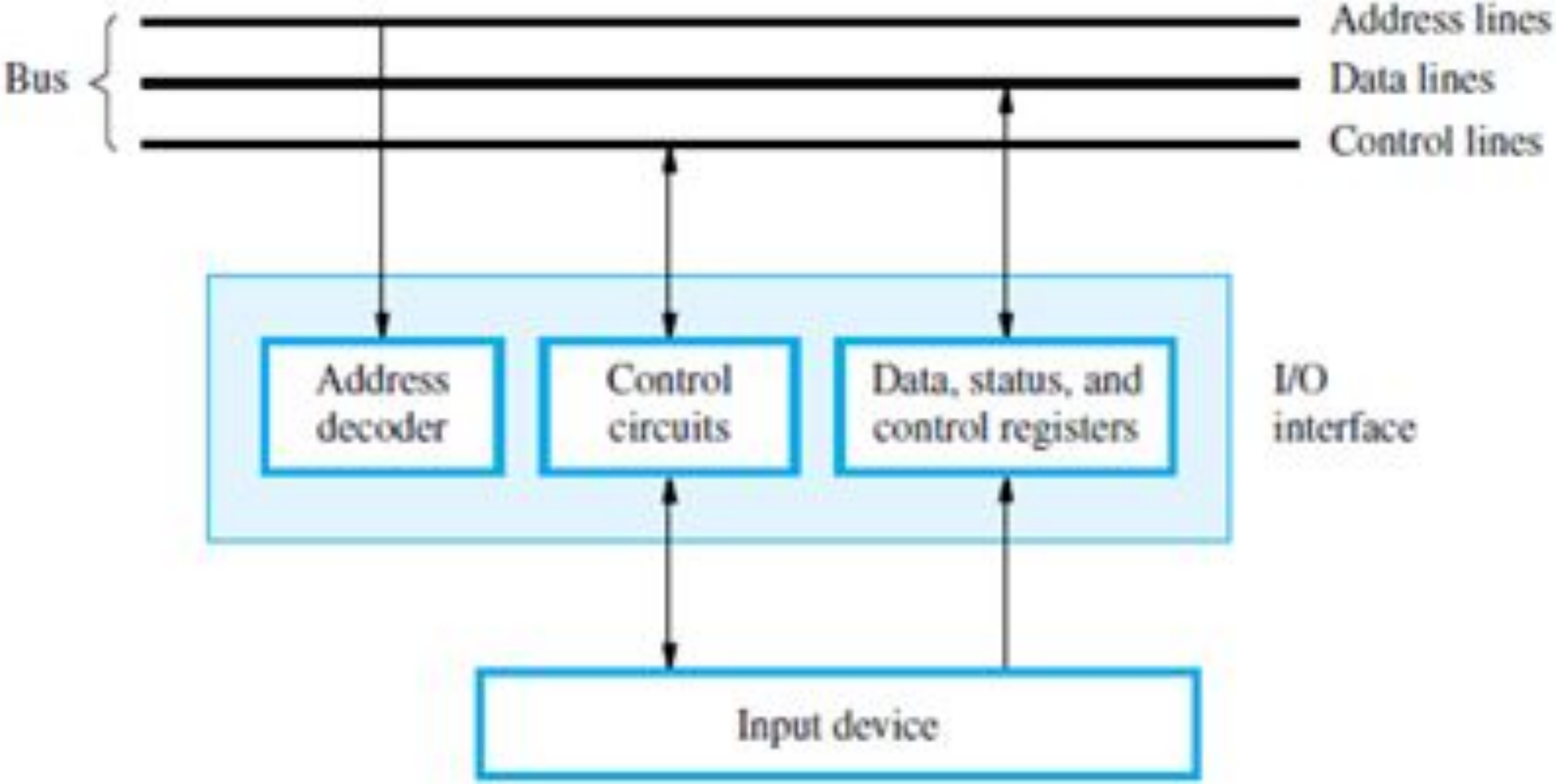
```
Move   DATAIN, RO
```

Simillarly ,

```
Move RO, DATAOUT
```

Sends the contents of register to the location DATAOUT.

Hardware needed to access I/O devices



- Address Decoder: It enables the device to recognize its address when the address appears on the address line.
- Data register : It holds the data being transferred to or from the processor.
- Status register: It contains information relevant to the operation of the I/O device.
  - Both data and status register are connected to data bus and assigned with unique address.

- For an input device such as Keyboard, status flag, SIN is included in the interface circuit as a part of status flag .
- The flag is set to 1 when the character is entered at the keyboard (program reads the input data register).
- The flag is cleared or set 0 once the character is read by the processor.

Consider a simple example of I / O operations involving a keyboard and a display device in a computer system.

The four registers shown below are used in the data transfer operations

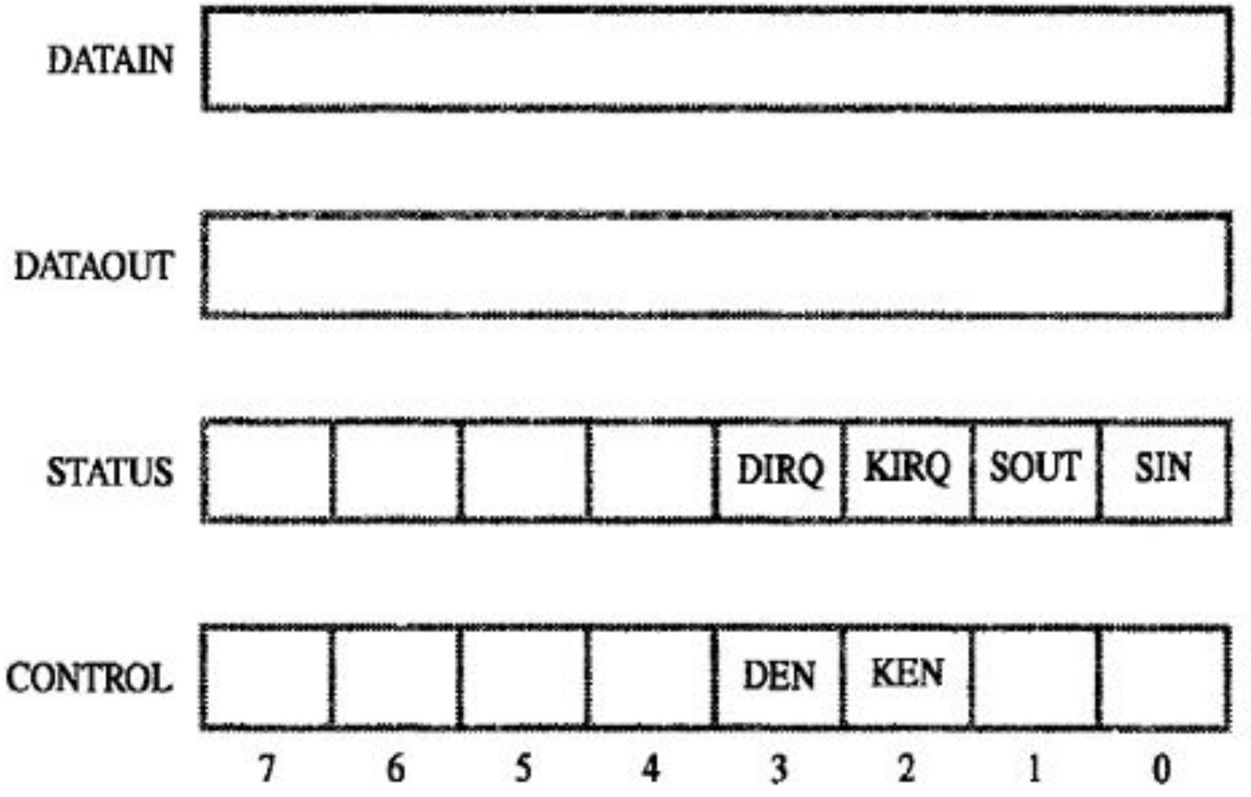
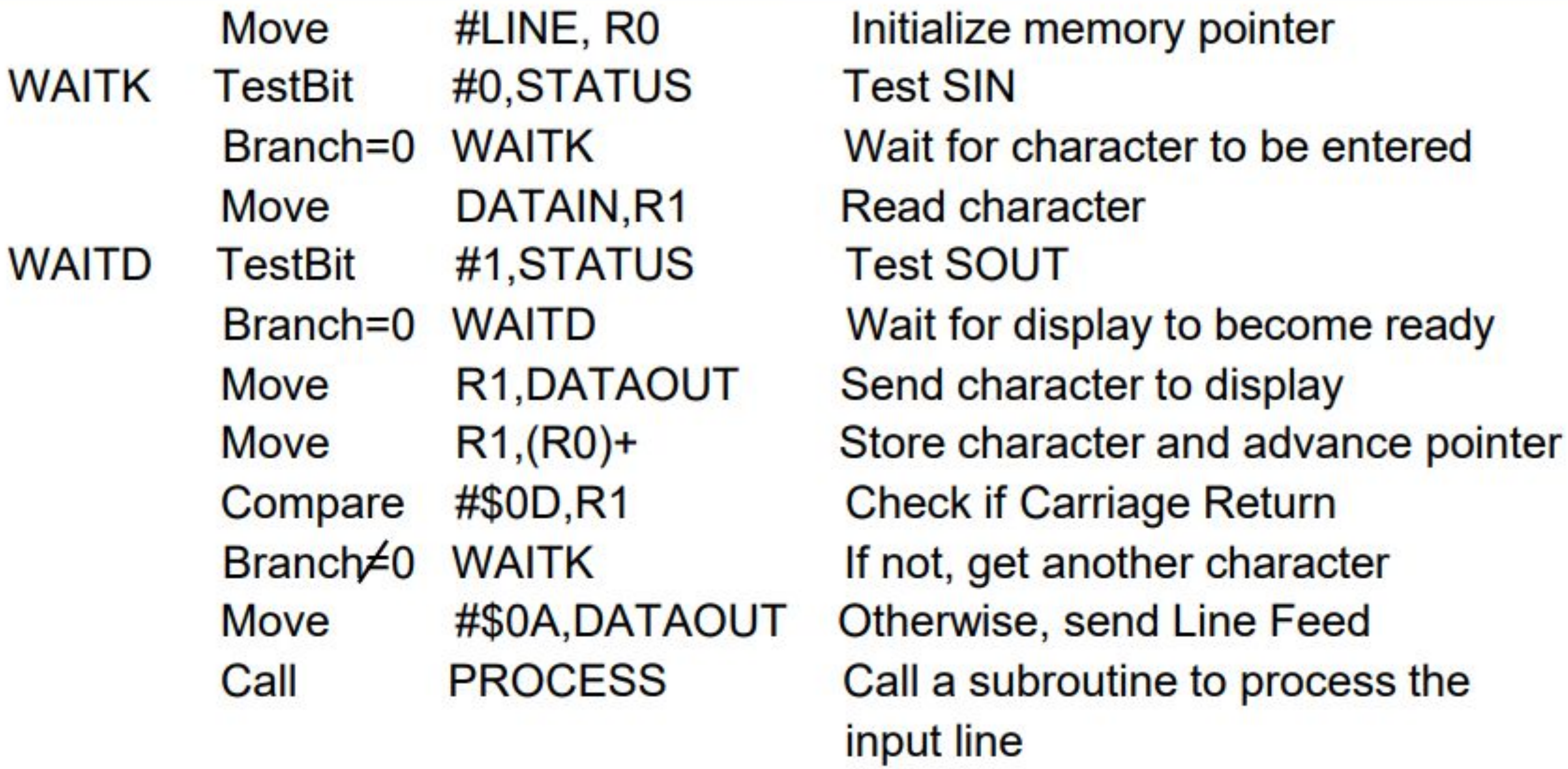


Figure 4.3 Registers in keyboard and display interfaces.



```
         Move       #LINE, R0        Initialize memory pointer
WAITK    TestBit    #0,STATUS        Test SIN
         Branch=0   WAITK            Wait for character to be entered
         Move       DATAIN,R1        Read character
WAITD    TestBit    #1,STATUS        Test SOUT
         Branch=0   WAITD            Wait for display to become ready
         Move       R1,DATAOUT       Send character to display
         Move       R1,(R0)+         Store character and advance pointer
         Compare    #$0D,R1          Check if Carriage Return
         Branch≠0   WAITK            If not, get another character
         Move       #$0A,DATAOUT     Otherwise, send Line Feed
         Call       PROCESS          Call a subroutine to process the
                                     input line
```

Interrupt

- An interrupt is a signal sent to the processor that temporarily halts the current execution of instructions so that the system can respond to an urgent task or event.
- Once the interrupt is handled, the processor typically resumes the original task where it left off. Interrupts are essential for handling events like I/O operations, hardware malfunctions, or timer events efficiently.

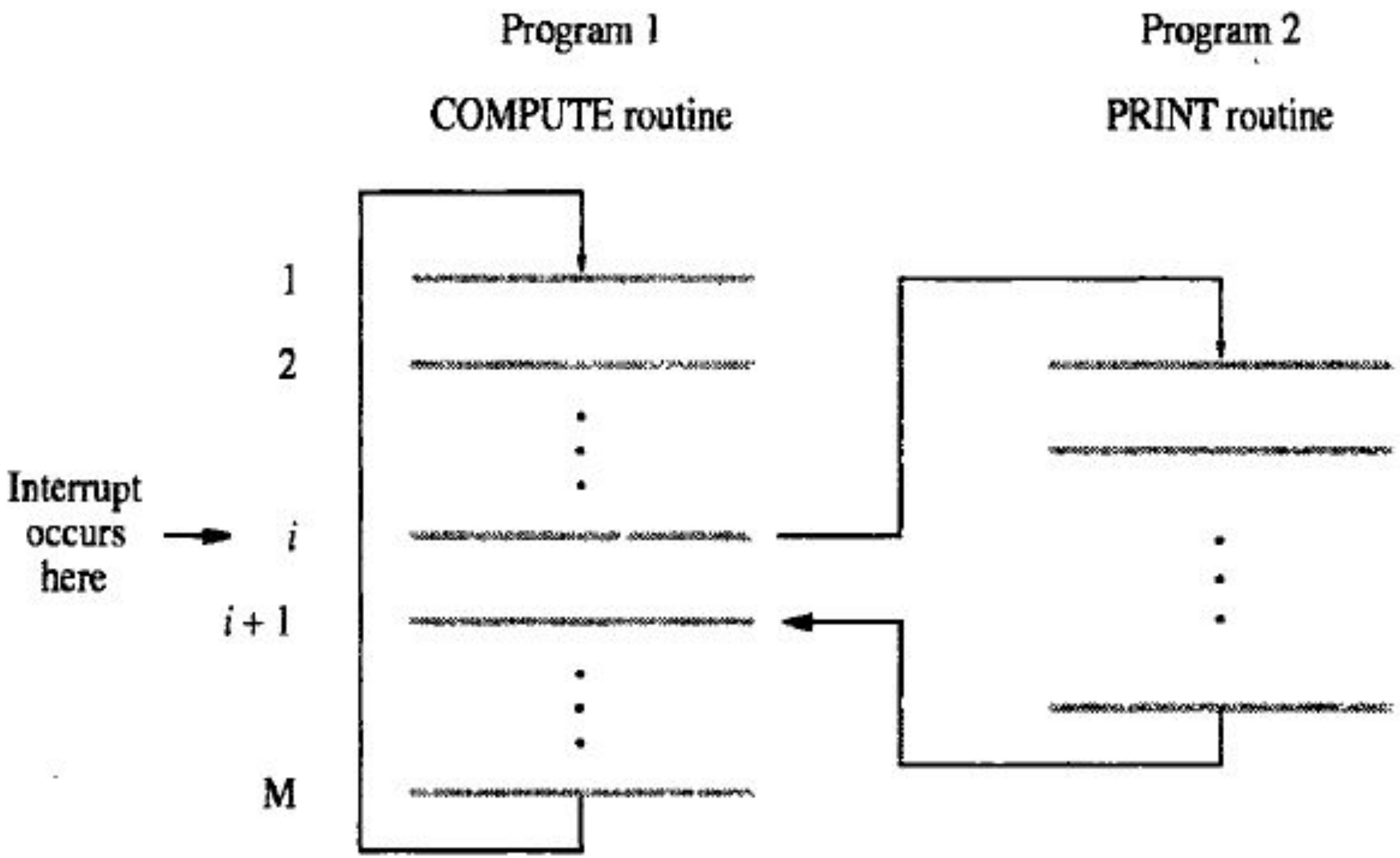


Figure 4.5 Transfer of control through the use of interrupts.

How interrupts are executed

- Routine executed in response to an interrupt request is called the **Interrupt-Service Routines.**
- When an interrupt occurs , control must be transferred to the interrupt service routine.
- But before transferring control, the current contents of the PC, must be saved in a known location.
- This will enable the return-from-interrupt instruction to resume execution.
- Return address, or the contents of the PC are usually stored on the Processor Stack.

Interrupt-Service Routine & Subroutine

- **Interrupt Service Routine** (ISR): An ISR is invoked automatically when an interrupt signal occurs, typically from hardware (e.g., a timer, a peripheral device).
- The processor stops executing the current task to handle the interrupt and executes the corresponding ISR.
- It operates asynchronously, meaning it can interrupt any part of the code without prior knowledge or control from the programmer.

Subroutine

- A subroutine is explicitly called by the programmer within the code using a function call.
- It is executed as part of the normal flow of the program, and the processor continues executing the program sequentially after the subroutine finishes.
- Subroutines are synchronous, meaning they are called and returned in a controlled manner within the program flow.

Interrupt-Service Routine

- As a result, before branching to the interrupt-service routine, not only the PC, but other information such as conditional code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
- This will enable the interrupted program to resume execution upon return from interrupt service routine.

- Saving and restoring information can be done automatically by the processor or explicitly by program instructions.
- Saving and restoring registers involves memory transfers:
  - Increases the total execution time.
  - Increases the delay between the time an interrupt request is received , and the start of execution of the interrupt – service routine. This delay is called **interrupt latency.**
- In order to reduce the interrupt latency, most processors save only the minimal amount of information:
- It includes Program Counter(PC) and Processor Status Registers (FLAG).
- Any additional information that must be saved explicitly by the program instructions at the beginning of the interrupt service routine.

- When a processor receives an interrupt –request (INTR), it must branch to the interrupt service routine.

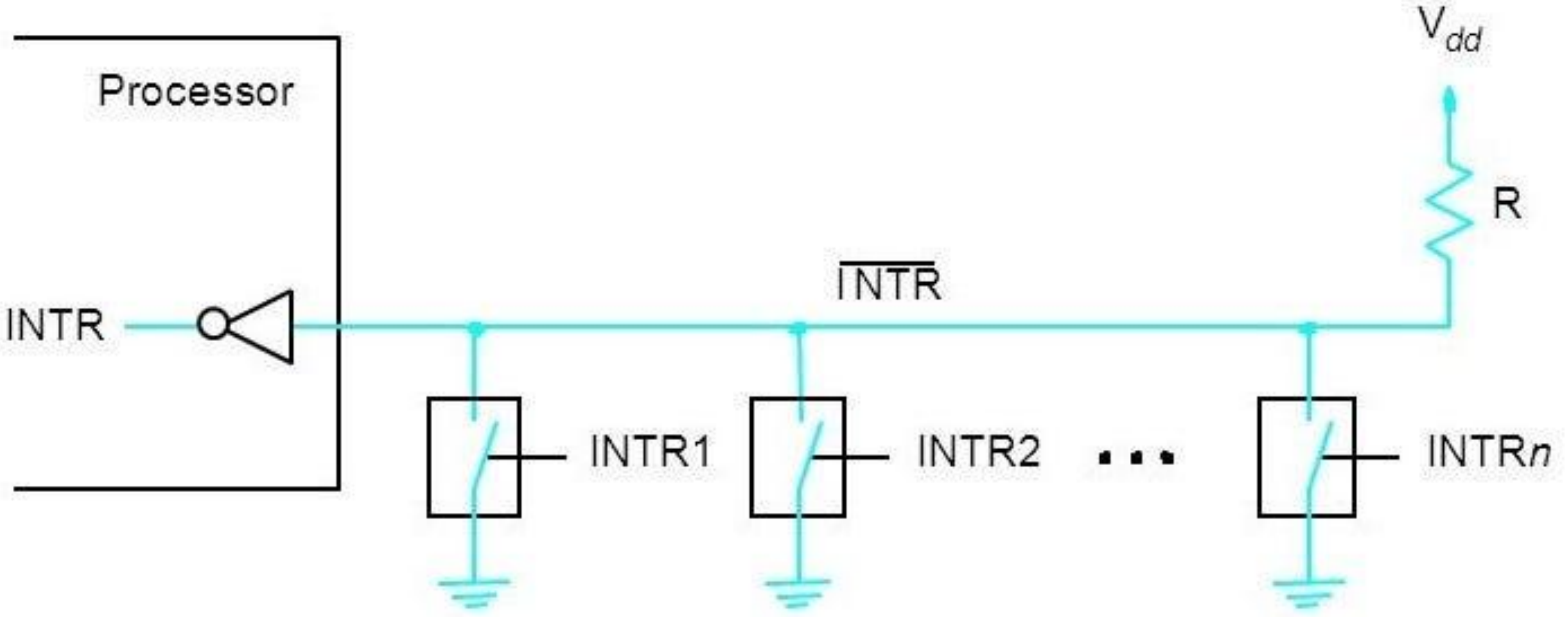
- It must also inform the device that it has recognized the interrupt request.

- This can be accomplished in two ways:

 Some processors have an explicit interrupt- acknowledge control signal (INTA) for this purpose.

 In other cases, the data transfer that takes place between the device and the processor can be used to inform the device.

Interrupt Hardware



Enabling and disabling Interrupts

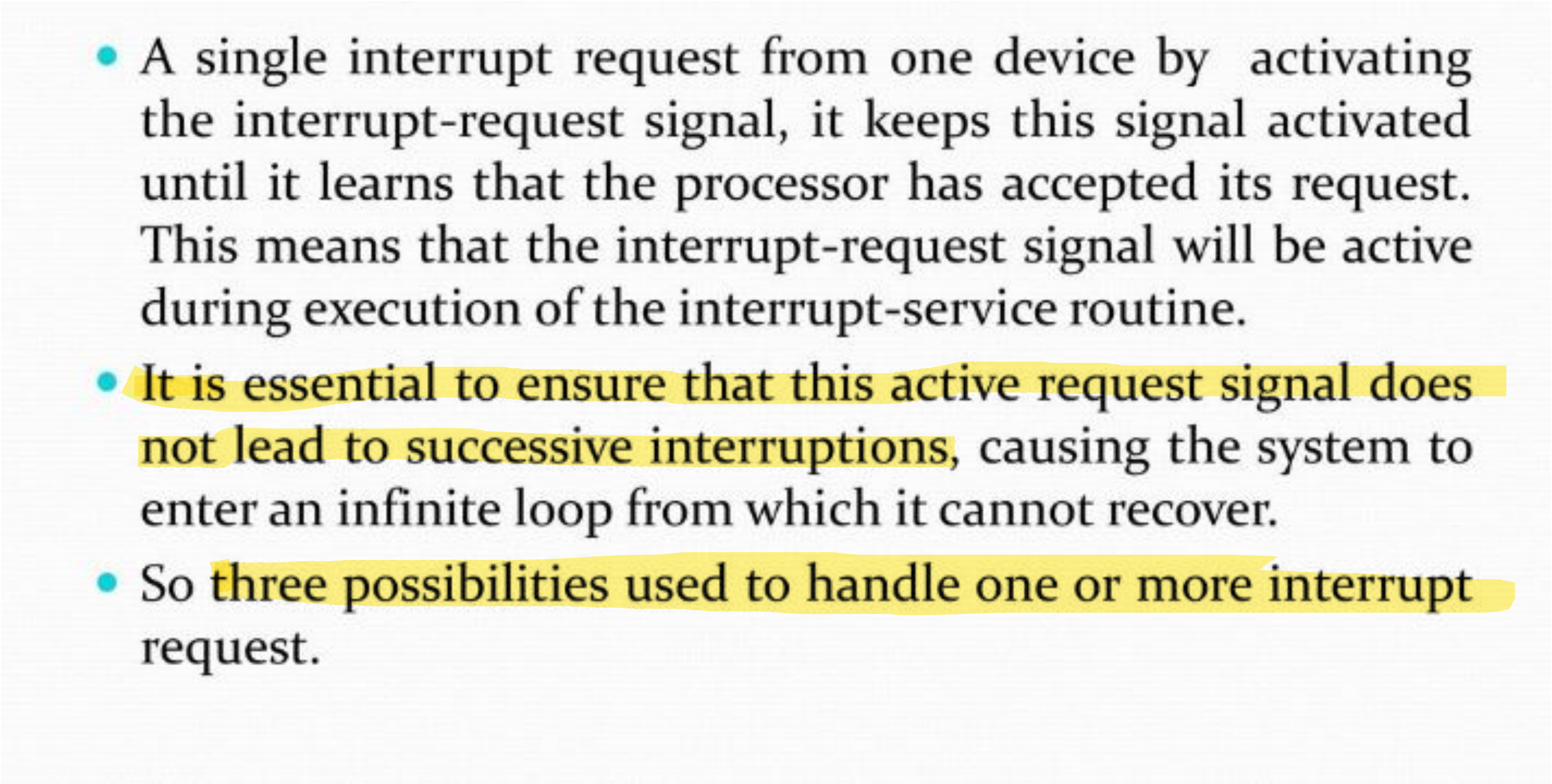
- Interrupt-request (INTR) interrupt the execution of the program, may alter the intended sequence of events :
  - Sometimes such alterations may be undesirable and must not be allowed.
  - For example, the processor may not want to be interrupted by the some device while executing its Interrupt – Service routine.
- Processors generally provide the ability to enable and disable such interruptions as desirable.

Enabling Interrupts

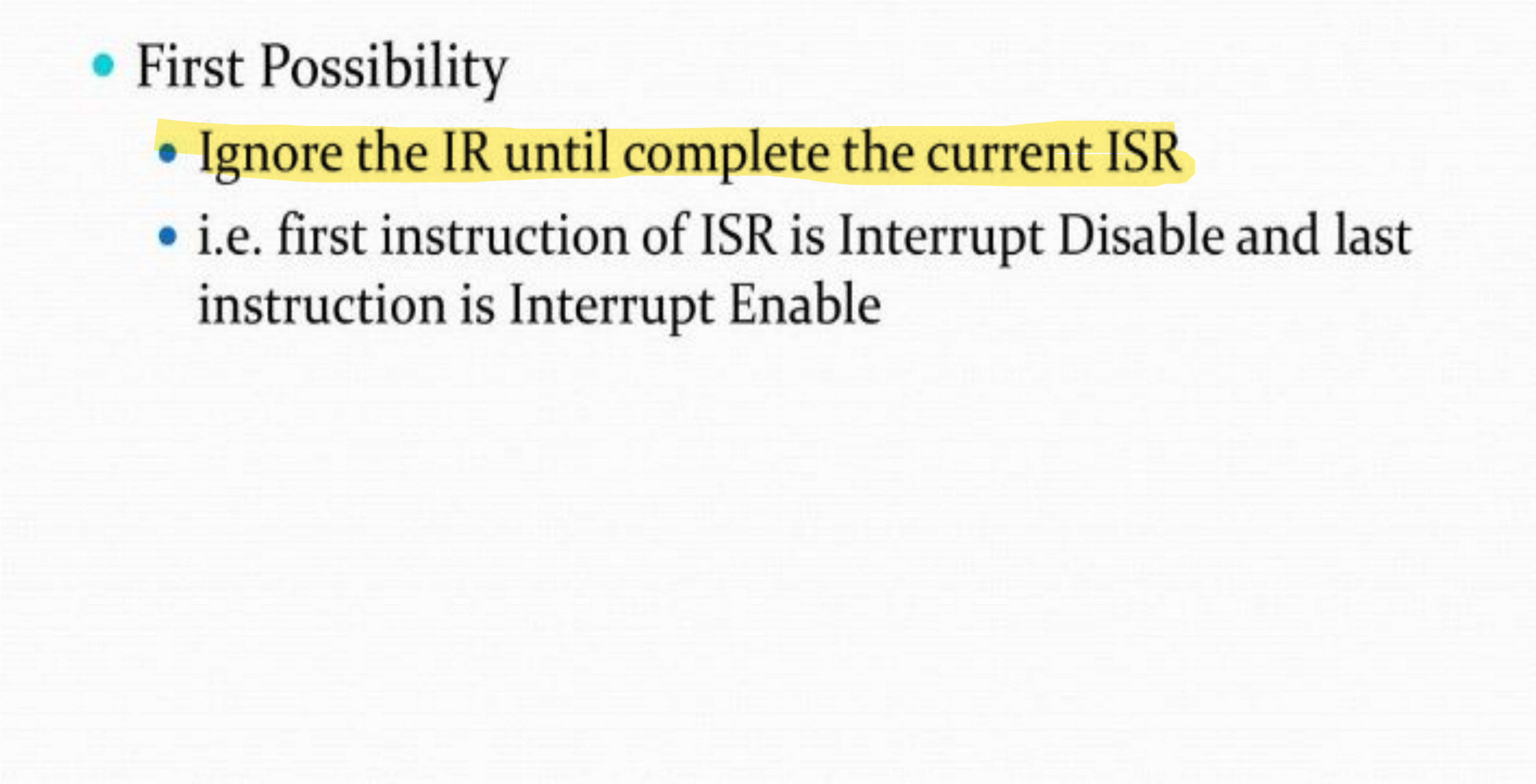
- When interrupts are enabled, the processor is allowed to be interrupted by signals.
- This means if a peripheral device or a process needs immediate attention (such as hardware failures, input/output operations, etc.), it can interrupt the current task the processor is handling.
- This allows real-time and efficient responses to external events.
- For example, when a user presses a key on the keyboard, an interrupt signal is sent to the processor. If interrupts are enabled, the processor temporarily stops what it's doing and handles the key press.

Disabling Interrupts

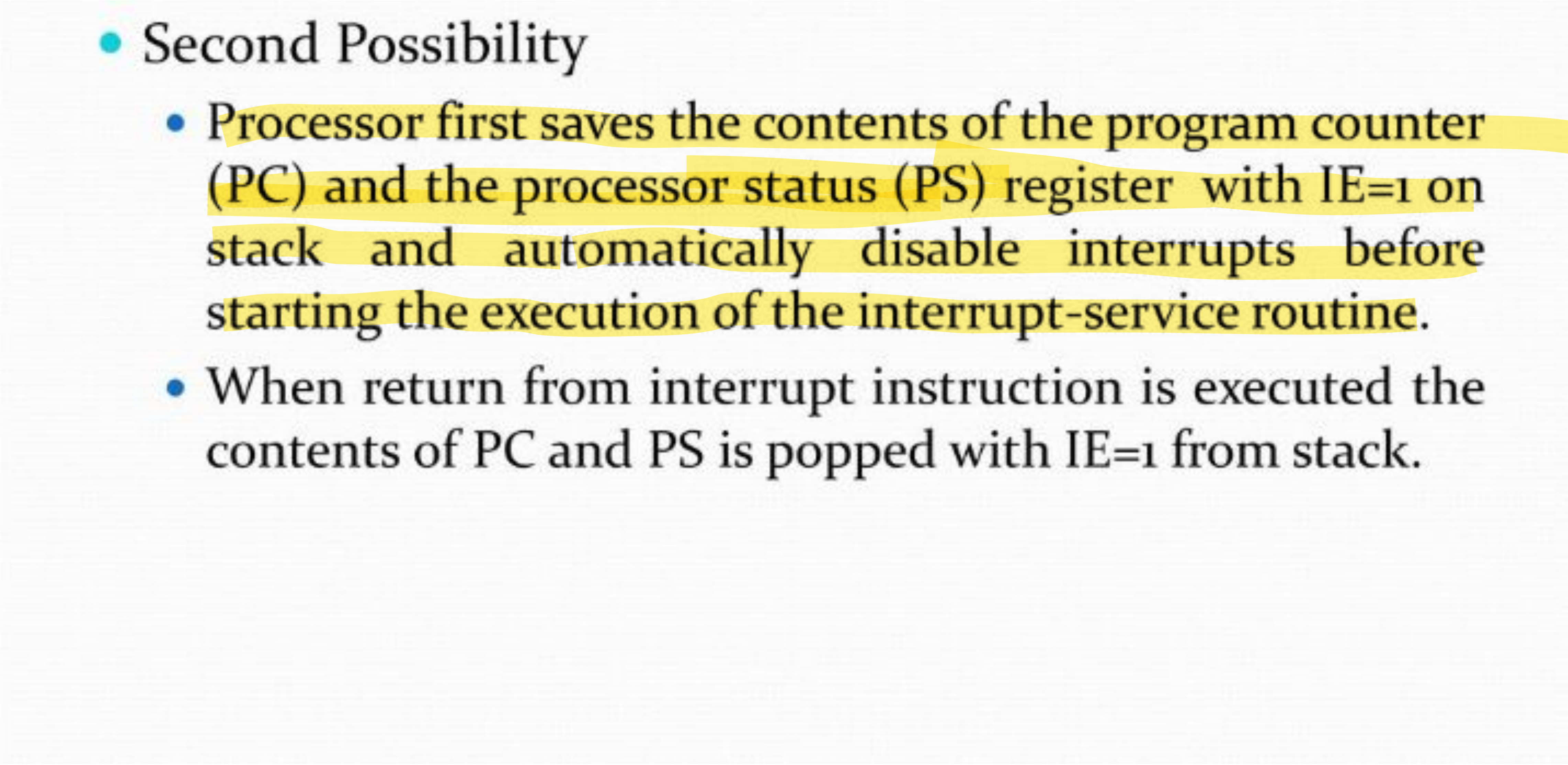
- When interrupts are disabled, the processor ignores interrupt signals.
- This ensures that no other task can interrupt the current execution.
- This can be useful when the system is in a critical section of code where interruptions could cause errors or corruption.
- For example, while writing data to memory, disabling interrupts can ensure the integrity of the data being written. Once the critical operation is complete, interrupts can be re-enabled to allow the processor to respond to other signals.



- A single interrupt request from one device by activating the interrupt-request signal, it keeps this signal activated until it learns that the processor has accepted its request. This means that the interrupt-request signal will be active during execution of the interrupt-service routine.
- It is essential to ensure that this active request signal does not lead to successive interruptions, causing the system to enter an infinite loop from which it cannot recover.
- So three possibilities used to handle one or more interrupt request.

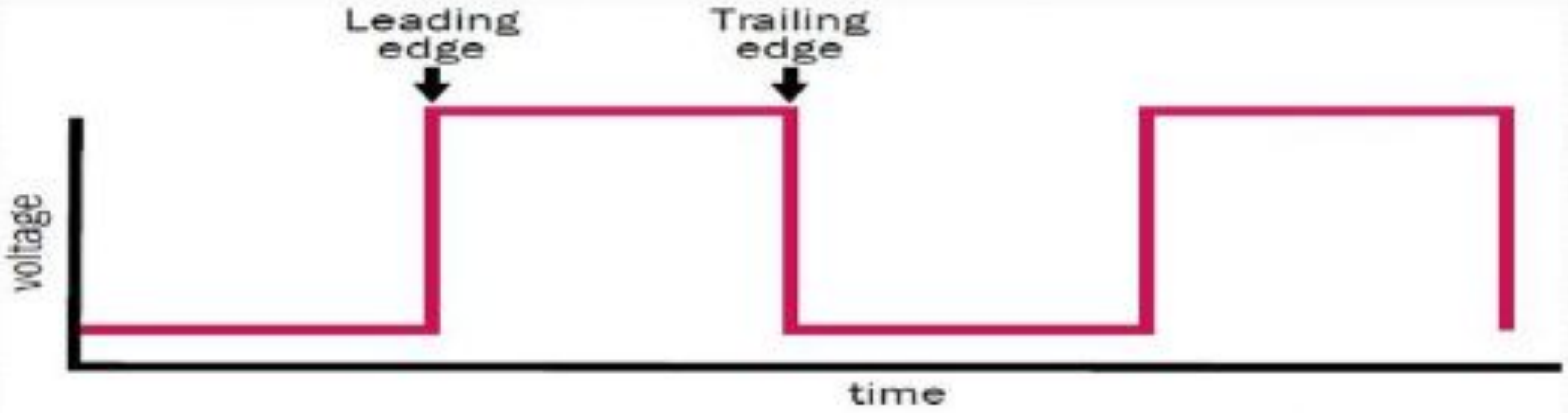


- First Possibility
  - Ignore the IR until complete the current ISR
  - i.e. first instruction of ISR is Interrupt Disable and last instruction is Interrupt Enable



- Second Possibility
  - Processor first saves the contents of the program counter (PC) and the processor status (PS) register with IE=1 on stack and automatically disable interrupts before starting the execution of the interrupt-service routine.
  - When return from interrupt instruction is executed the contents of PC and PS is popped with IE=1 from stack.

- Third Possibility
  - IR line must accept only leading edge of the signal (Edge triggered line)
  - Processor receive only one request regardless of how long the line is activated.
  - So no question of multiple interruption or no need of explicit instruction for enable/disable interrupt.



Sequence of events involved in handling an interrupt request from a Single device

1. The device raises an Interrupt request.
2. Processor interrupts the program currently being executed.
3. Interrupts are disabled by changing the control bits in the Processor Status(PS).
4. Device is informed that its request has been recognized and the device deactivates the interrupt request signal.
5. The action requested by the interrupts is performed by the interrupt service routine.
6. Interrupt are enabled and execution of the interrupted program is resumed.

Handling Multiple Devices

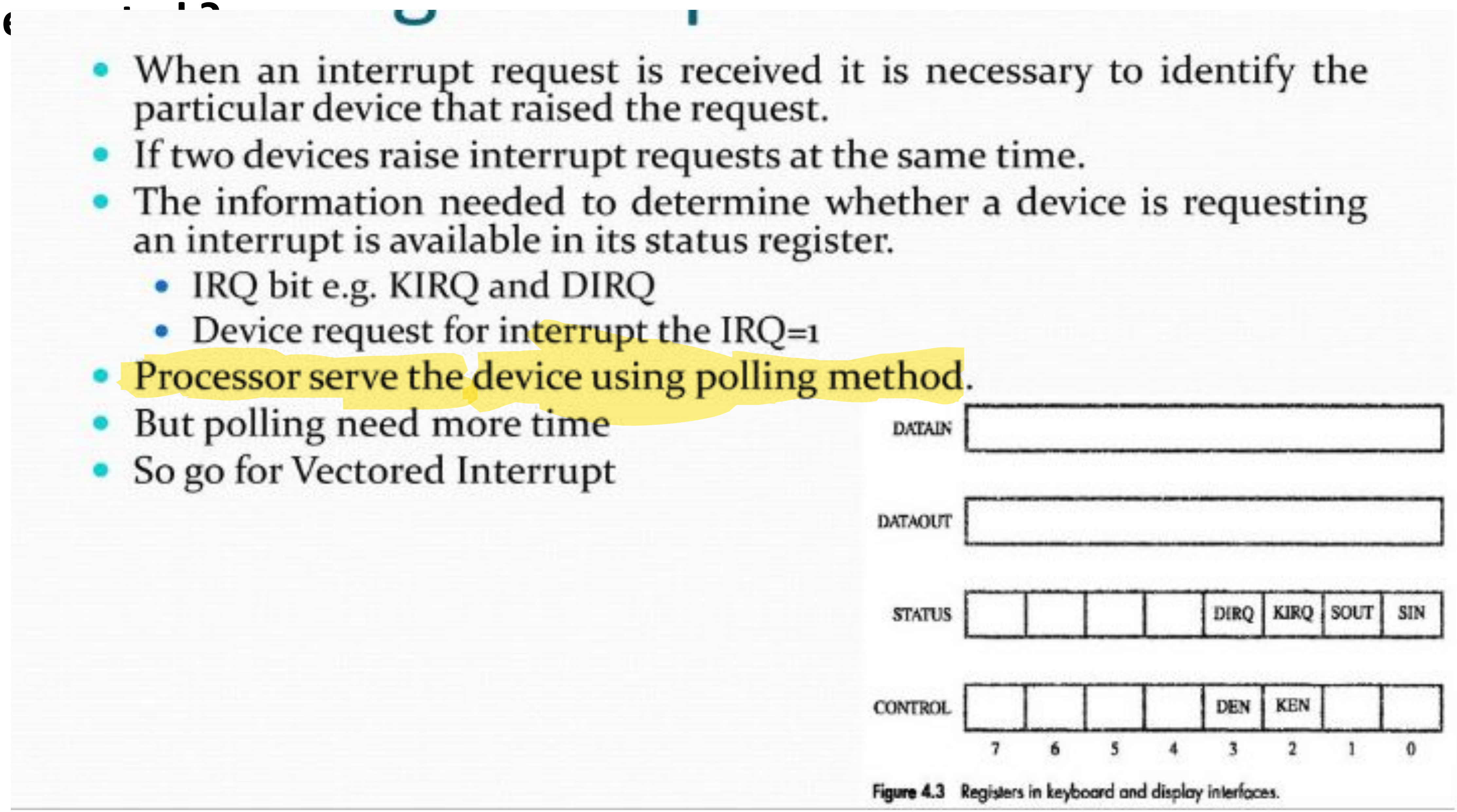
- Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt request.
- Each device operates independently , and hence no definite order can be imposed on how the devices generate interrupt requests ?

1. How does the processor know which device has generated an interrupt ?
2. How does the processor know which interrupt service routine needs to be executed ?
3. When the processor is executing an interrupt service routine for one device ,can another device interrupt the processor ?
4. If two interrupt-requests are received simultaneously , then how to break the tie ?

How does the processor know which device has generated an interrupt ?

- Consider , where all devices send their interrupts – requests over a single control line in the bus.
- When the processor receives an interrupt request over this control line, how does it know which device is requesting an interrupt ?
- This information is available in the status register of the device requesting an interrupt.
- The status register of each device has an IRQ bit which it sets to 1 when it requests an interrupt.

How does the processor know which interrupt service routine needs to be executed?



- When an interrupt request is received it is necessary to identify the particular device that raised the request.
- If two devices raise interrupt requests at the same time.
- The information needed to determine whether a device is requesting an interrupt is available in its status register.
  - IRQ bit e.g. KIRQ and DIRQ
  - Device request for interrupt the IRQ=1
- Processor serve the device using polling method.
- But polling need more time
- So go for Vectored Interrupt

DATAIN								
DATAOUT								
STATUS					DIRQ	KIRQ	SOUT	SIN
CONTROL					DEN	KEN		
	7	6	5	4	3	2	1	0

Figure 4.3 Registers in keyboard and display interfaces.

Vectored Interrupts

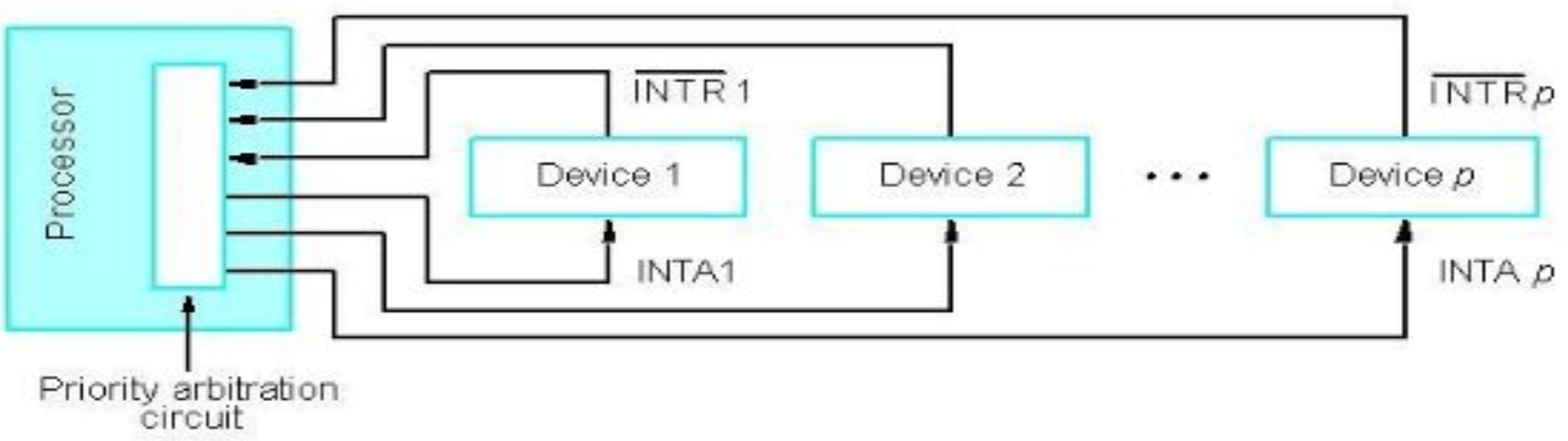
- A **vectored interrupt** is a type of interrupt where the interrupting device provides the address of the interrupt service routine (ISR) directly to the processor.
- This allows the processor to immediately jump to the correct ISR without needing to check for the source of the interrupt, which speeds up the interrupt handling process.

How a Vectored Interrupt works

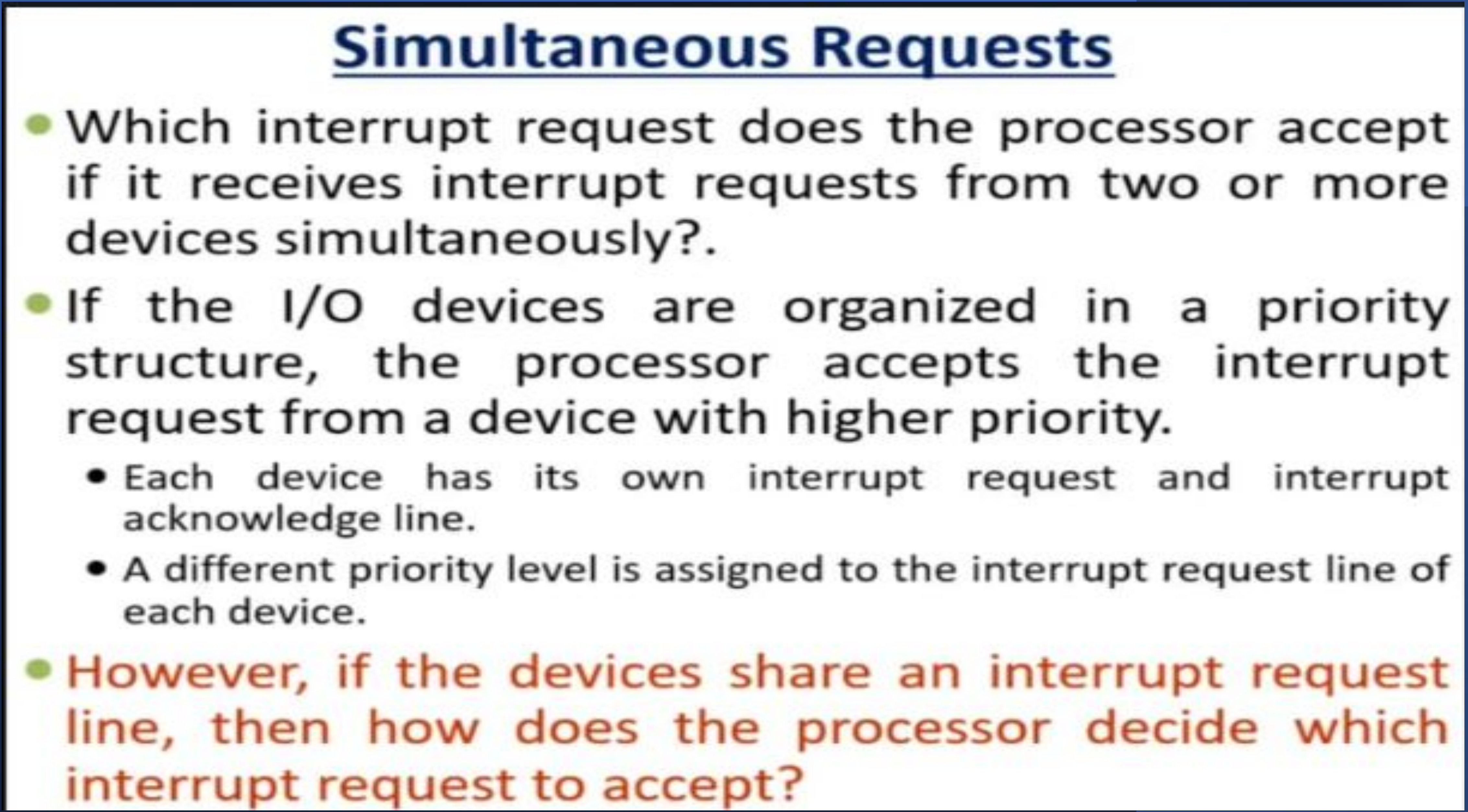
- When a device generates an interrupt, it sends both the interrupt signal and a unique identifier (interrupt vector) to the processor.
- The interrupt vector contains the address of the ISR associated with the device that triggered the interrupt.
- The processor uses this address to directly jump to the appropriate ISR and handle the interrupt

When the processor is executing an interrupt service routine for one device ,can another device interrupt the processor ?

- **Interrupt Nesting**
- The processor started executing the interrupt service routine for a device, it disabled the interrupts from the device.
- Since the interrupt service routines are usually short , the delay that this causes is generally acceptable.
- However, for certain devices this delay may not be acceptable.
- These devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device ?


Processor
$\overline{\text{INTR}}$ 1
$\overline{\text{INTR}}\,p$
Device 1
Device 2
• • •
Device p
INTA1
INTA p
Priority arbitration circuit

- Each device have a separate interrupt-request and interrupt-acknowledge line.
- Each interrupt- request line is assigned a different priority level.
- Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.
- If the interrupt requests has a higher priority level that the priority of the processor , then the request is accepted.



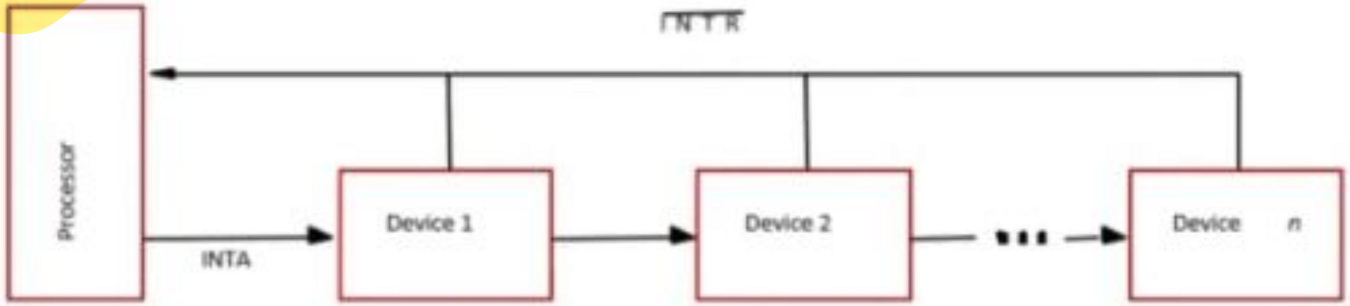
Simultaneous Requests

- Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?.
- If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.
  - Each device has its own interrupt request and interrupt acknowledge line.
  - A different priority level is assigned to the interrupt request line of each device.
- However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

Polling scheme:

- *If the processor uses a polling mechanism to poll the status registers of I/O devices to determine which device is requesting an interrupt.*
- *In this case the priority is determined by the order in which the devices are polled.*
- *The first device with status bit set to 1 is the device whose interrupt request is accepted.*

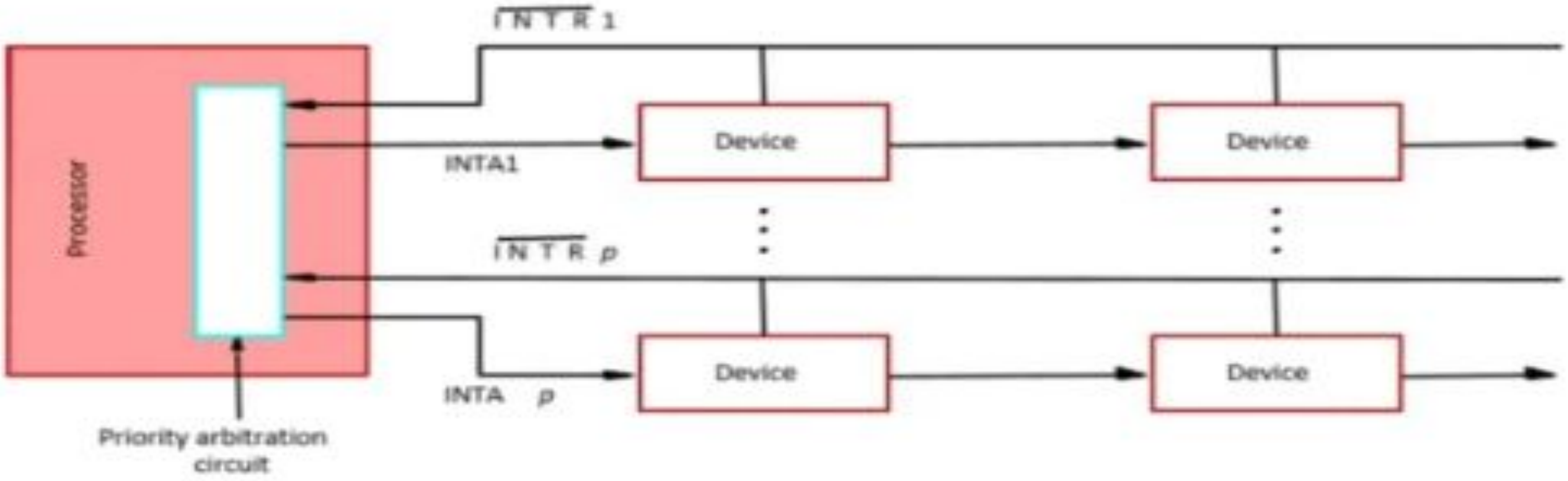
Daisy chain scheme:



- *Devices are connected to form a daisy chain.*
- *Devices share the interrupt-request line, and interrupt-acknowledge line is connected to form a daisy chain.*
- *When devices raise an interrupt request, the interrupt-request line is activated.*
- *The processor in response activates interrupt-acknowledge.*
- *Received by device 1, if device 1 does not need service, it passes the signal to device 2.*
- *Device that is electrically closest to the processor has the highest priority.*

Priority Groups:

- *When I/O devices were organized into a* **priority structure***, each device had its own interrupt-request (***INTR***) and interrupt-acknowledge (***INTA***) line.*
- *When I/O devices were organized in a* **daisy chain** *fashion, the devices shared an interrupt-request line, and the interrupt-acknowledge propagated through the devices.*
- *A* **combination** *of priority structure and daisy chain scheme can also used.*



- *Devices are organized into groups.*
- *Each group is assigned a different priority level.*
- *All the devices within a single group share an interrupt-request line, and are connected to form a daisy chain.*

MECHANISMS USED FOR INTERFACING I/O-DEVICES

1) **Program Controlled I/O**

- Processor repeatedly checks status-flag to achieve required synchronization b/w processor & I/O device.
- Main drawback: The processor wastes time in checking status of device before actual data-transfer takes place.

2) **Interrupt I/O**

- I/O-device initiates the action instead of the processor.
- I/O-device sends an INTR signal over bus whenever it is ready for a data-transfer operation.
- Like this, required synchronization is done between processor & I/O device.

3) **Direct Memory Access (DMA)**

- Device-interface transfer data directly to/from the memory w/o continuous involvement by the processor.
- DMA is a technique used for high speed I/O-device.

Direct Memory Access (DMA)

- A Direct Memory Access (DMA) is an operation in which data is copied from one resource to another resource in a computer system without involvement of the CPU.

- A Special control unit may be provided to transfer a block of data directly between an I/O device and the main memory, without continuous intervention by the processor.

- Control unit which performs these transfers is a part of the I/O devices interface circuit. This control unit is called as a **DMA Controller** (DMAC)

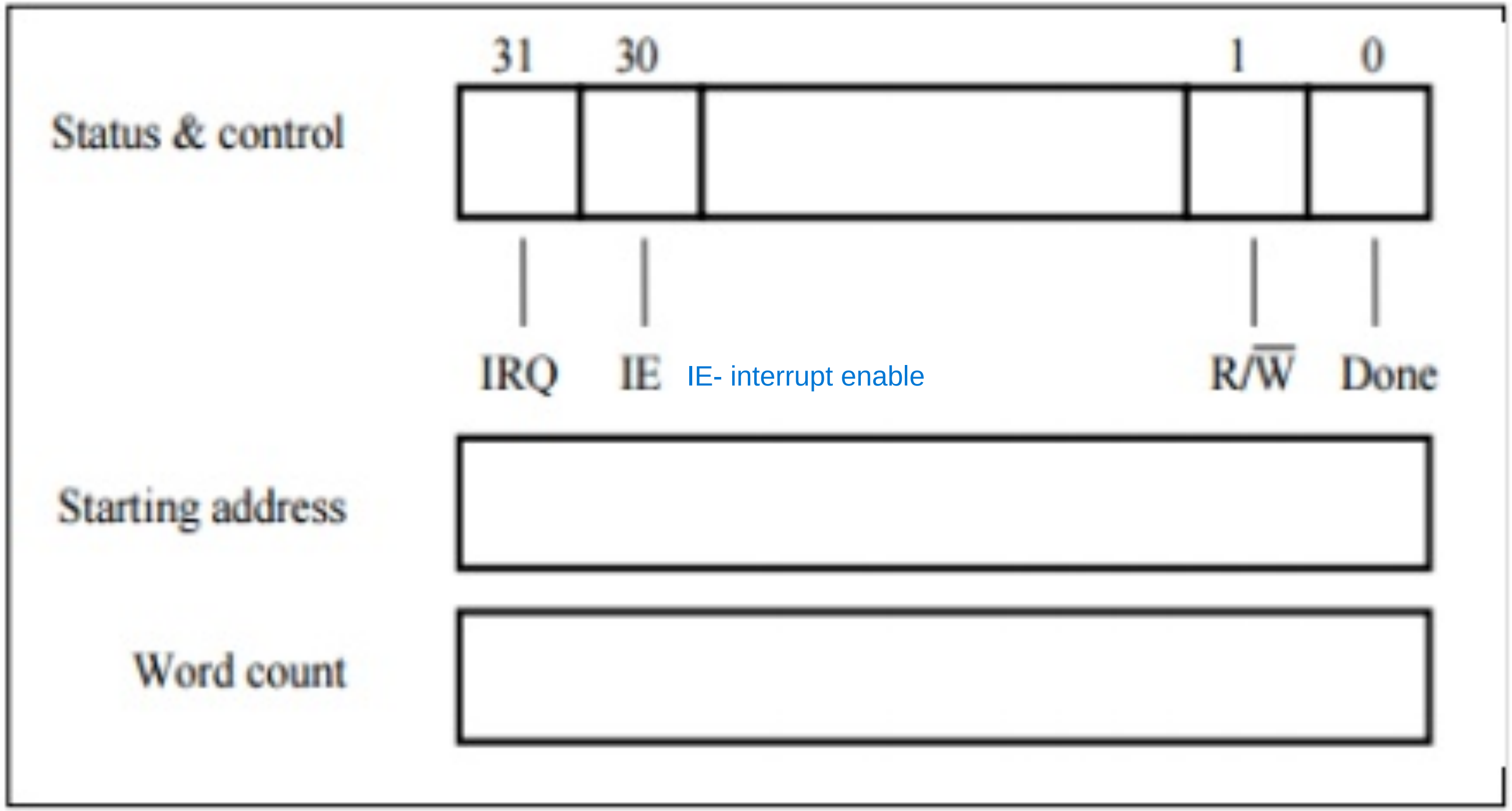
Key Aspects of DMA:

- **Bypasses the CPU**: DMA controllers take over data transfer tasks, freeing the CPU for other processing activities and improving overall efficiency.
- **High-speed data transfer**: It is especially useful for devices like hard drives, graphics cards, and sound cards, where fast data movement is crucial.

- DMA Controller performs functions that would be normally carried out by the processor :
    - For each word, it provides the memory address and all the control signals.
    - To transfer a block of data , it increments the memory addresses and keeps track of the number of transfers.
- To initiate the DMA transfer, the processor informs the DMA controller of:
    - Starting address,
    - Number of words in the block.
    - Direction of transfer (to the memory , or from the memory).

- Once DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.

DMA Controller Registers



Types of DMA Transfer

- DMA has high priority than the processor.
- **Burst Mode**: Transfers a block of data all at once.
- **Cycle Stealing**: DMA temporarily pauses the CPU’s access to memory for each byte transferred.
- **Transparent Mode**: DMA operates only when the CPU isn’t using memory, ensuring minimal impact on CPU performance.

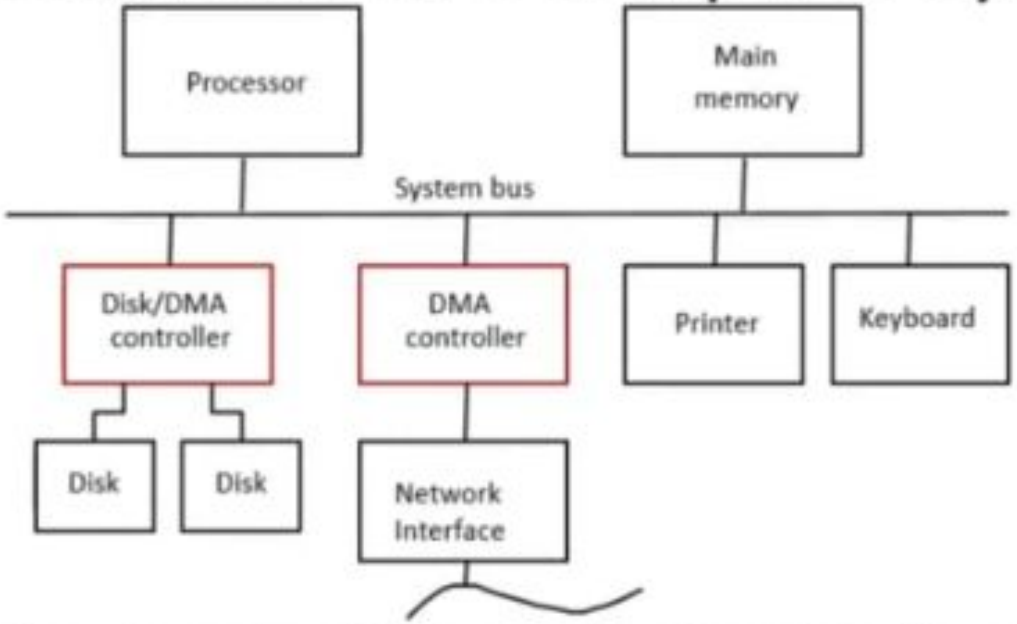
How does DMA work

- Direct Memory Access (DMA) enables peripherals (like hard drives, graphics cards, and network cards) to transfer data directly to and from the main memory, bypassing the CPU. This process enhances system efficiency, especially for tasks involving large amounts of data.
- **Steps of DMA Operation**
- **Request for Data Transfer**:
  - When a peripheral device (such as a hard drive) needs to read from or write to the memory, it sends a request to the **DMA controller**.
- **CPU Grants Control**:
  - The CPU is notified of the request, and if it approves, it relinquishes control over the memory bus, allowing the DMA controller to access the memory directly.
  - The CPU is free to handle other tasks, reducing the overall load.

- DMA Controller Manages Transfer:
  - The DMA controller handles the data transfer between the peripheral and memory.
  - It uses an address register (to keep track of where to read/write) and a counter register (to manage the number of bytes transferred).
- Data Transfer Modes:
  - The transfer can occur in several modes:
    - Burst Mode
    - Cycle Stealing
    - Transparent Mode

- **Completion and CPU Notification**:
  - Once the transfer is complete, the DMA controller sends an interrupt signal to the CPU, notifying it that the transfer is done.
  - The CPU regains control of the memory bus and can now access the transferred data for further processing.

DMA Controller in a computer system



- *DMA controller connects a high-speed network to the computer bus.*
- *Disk controller, which controls two disks also has DMA capability. It provides two DMA channels.*
- *It can perform two independent DMA operations, as if each disk has its own DMA controller. The registers to store the memory address, word count and status and control information are duplicated.*

Bus arbitration

- **Bus arbitration** is the process by which a system determines which component (CPU, DMA controller, or I/O device) gets control over the shared communication bus at any given time. Since multiple devices may need to use the bus simultaneously, an efficient arbitration mechanism is essential to prevent conflicts and ensure orderly data transfer.
- The device that is allowed to initiate transfers on the bus at any given time is called the bus master.
- There can be only one bus-master at any given time.

Bus Arbitration is the process by which

→ next device to become the bus-master is selected &

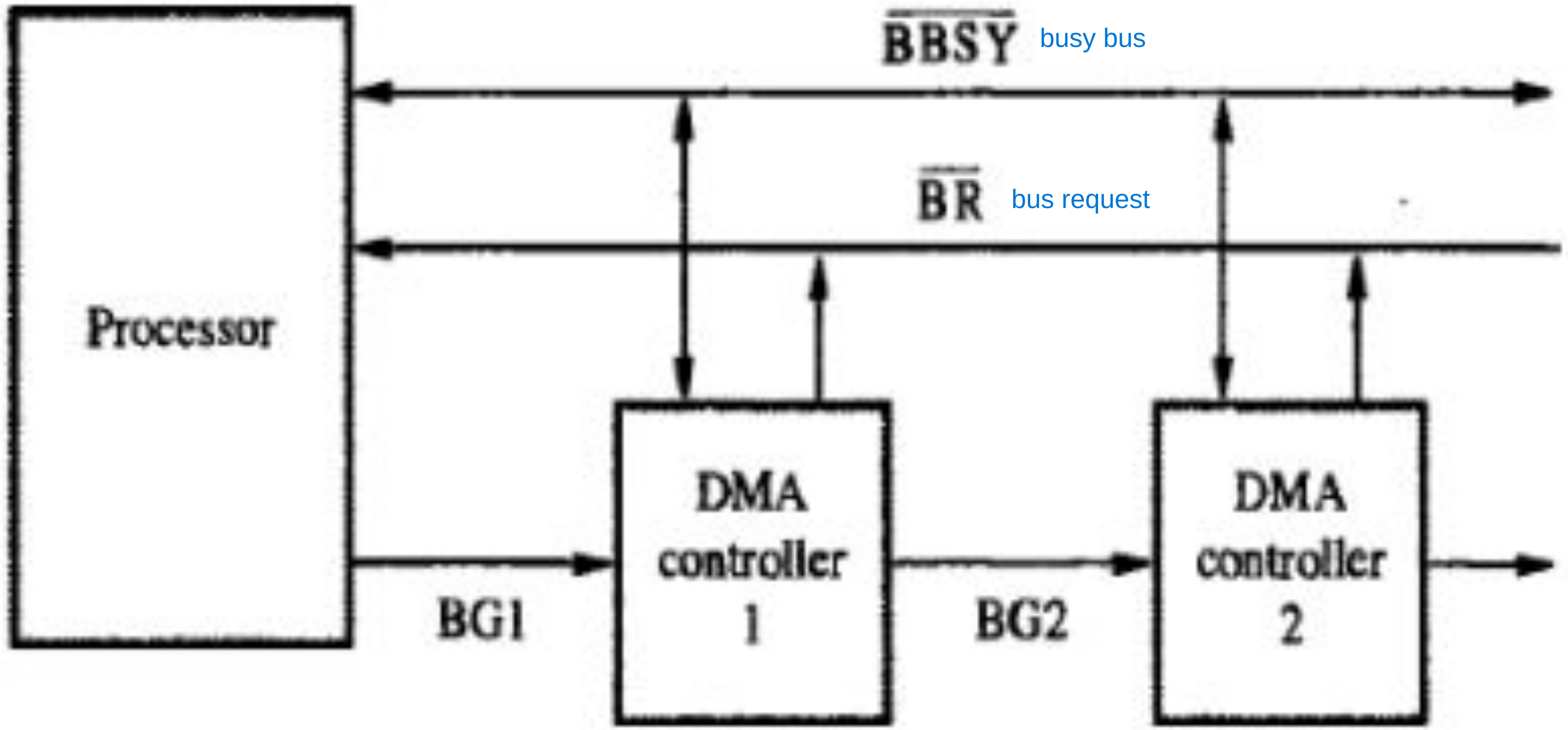
→ bus-mastership is transferred to that device.

The two approaches are:

1) **Centralized Arbitration:** A single bus-arbiter performs the required arbitration.

2) **Distributed Arbitration:** All devices participate in selection of next bus-master.

Centralized Bus Arbitration



busy bus

bus request

Figure 4.20 A simple arrangement for bus arbitration using a daisy chain.

- A single bus-arbiter performs the required arbitration (Figure: 4.20).
- Normally, processor is the bus-master.
- Processor may grant bus-master ship to one of the DMA controllers.
- A DMA controller indicates that it needs to become bus-master by activating BR line.
- The signal on the BR line is the logical OR of bus-requests from all devices connected to it.

bus grant 1

- Then, processor activates BG1 signal indicating to DMA controllers to use bus when it becomes free.
- BG1 signal is connected to all DMA controllers using a daisy-chain arrangement.
- If DMA controller-1 is requesting the bus, Then, DMA controller-1 blocks propagation of grant-signal to other devices. Otherwise, DMA controller-1 passes the grant downstream by asserting BG2.
- Current bus-master indicates to all devices that it is using bus by activating BBSY line.
- The bus-arbiter is used to coordinate the activities of all devices requesting memory transfers.
- Arbiter ensures that only 1 request is granted at any given time according to a priority scheme.

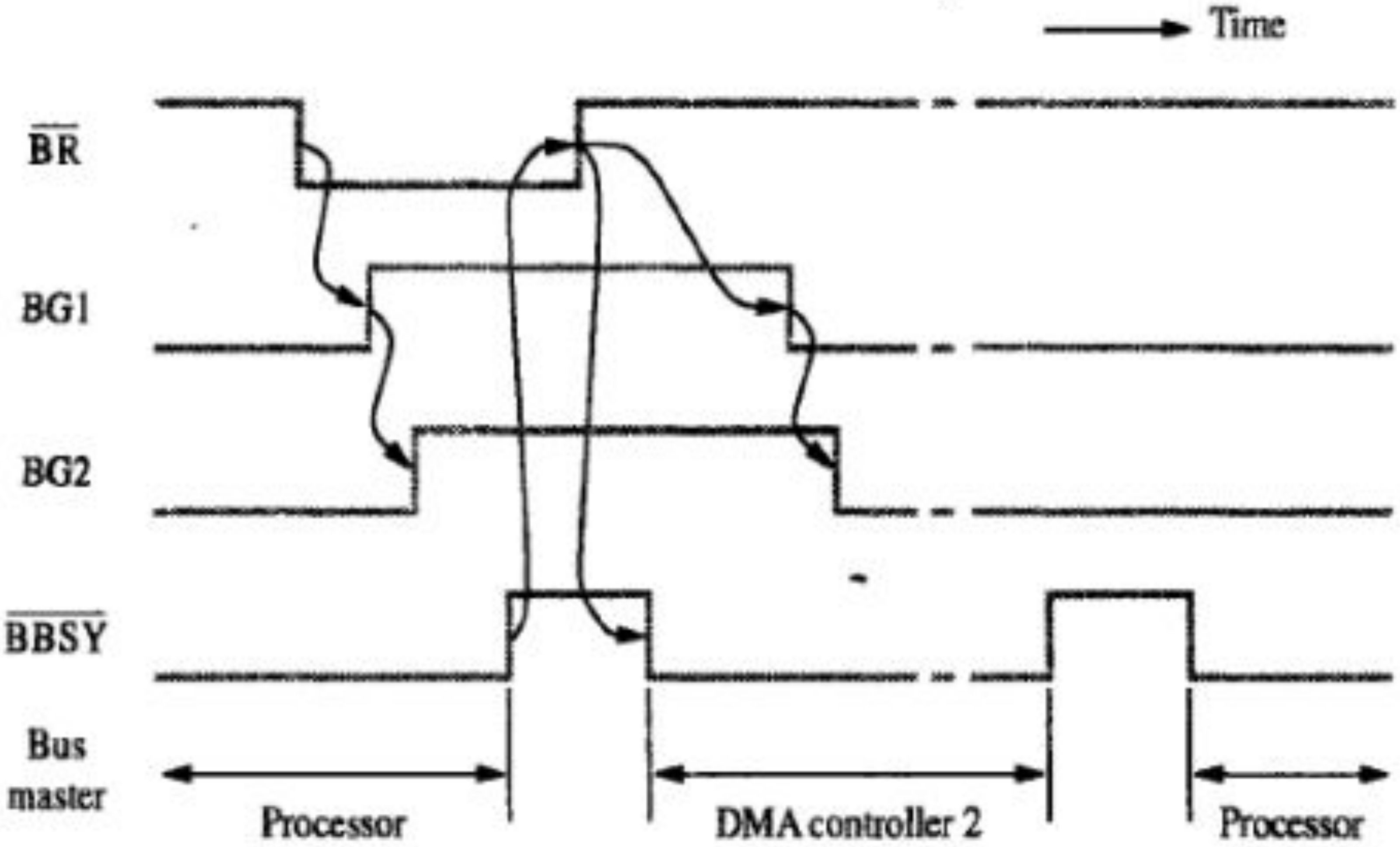


Figure 4.21 Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

DISTRIBUTED ARBITRATION

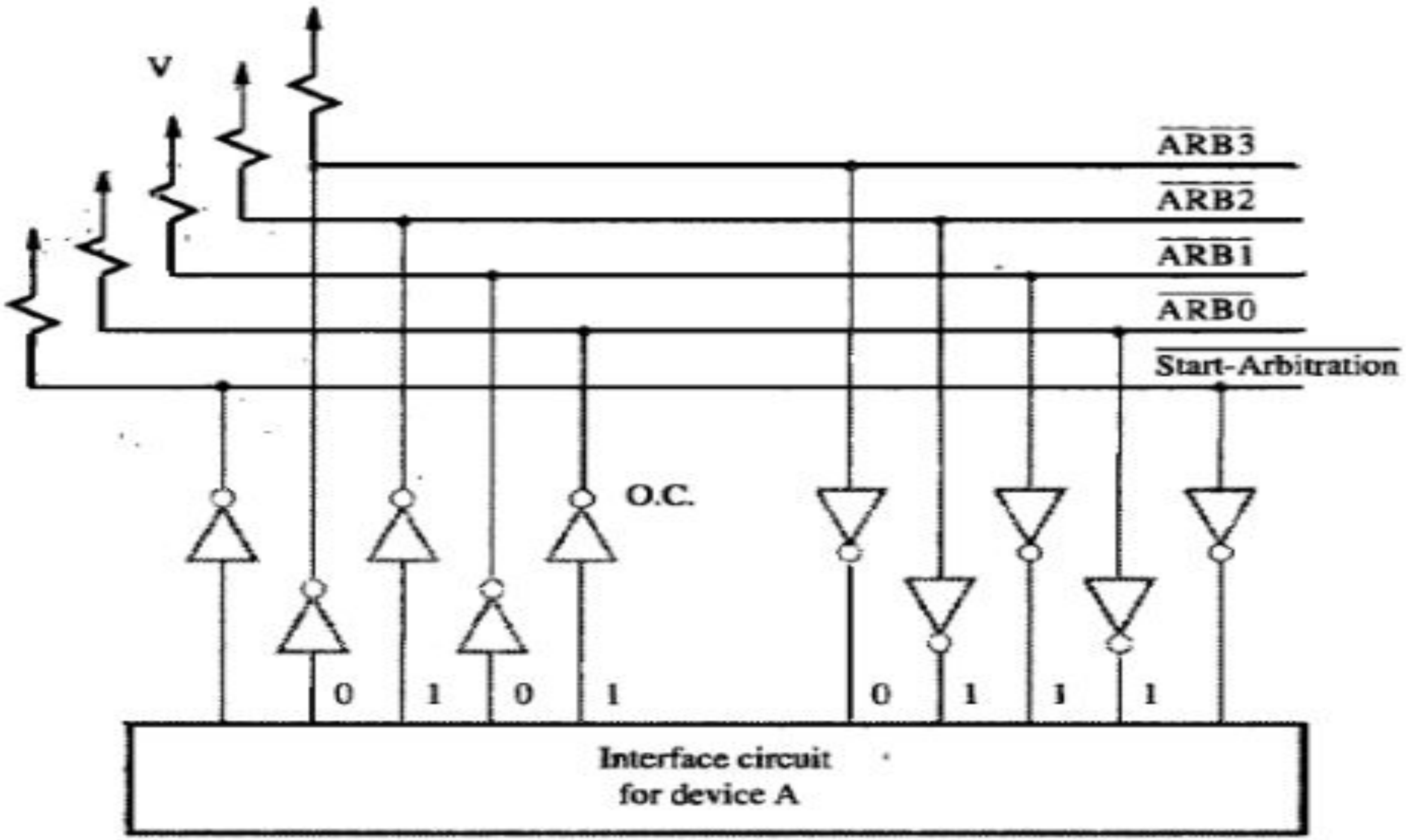


Figure 4.22 A distributed arbitration scheme.

Distributed BUS Arbitration

- All devices waiting to use the bus share the responsibility of carrying out the arbitration process.
    - Arbitration process does not depend on a central arbiter and hence distributed arbitration has **higher reliability.**
- Each device is assigned a 4-bit ID number.
- All the devices are connected using 5 lines, 4 arbitration lines to transmit the ID, and one line for the Start-Arbitration signal.

- To request the bus, a device:
  - Asserts the **Start-Arbitration signal**.
  - Places its **4-bit ID number** on the arbitration lines.
- The pattern that appears on the arbitration lines is the **logical-OR** of all the 4-bit device IDs placed on the arbitration lines.

- Each device **compares** the pattern that appears on the arbitration lines to its own ID, starting with MSB.
- If it detects a difference, it transmits 0s on the arbitration lines for **that and all lower** bit positions.

Speed , Size and Cost

- The entire computer memory can be viewed as the hierarchy.

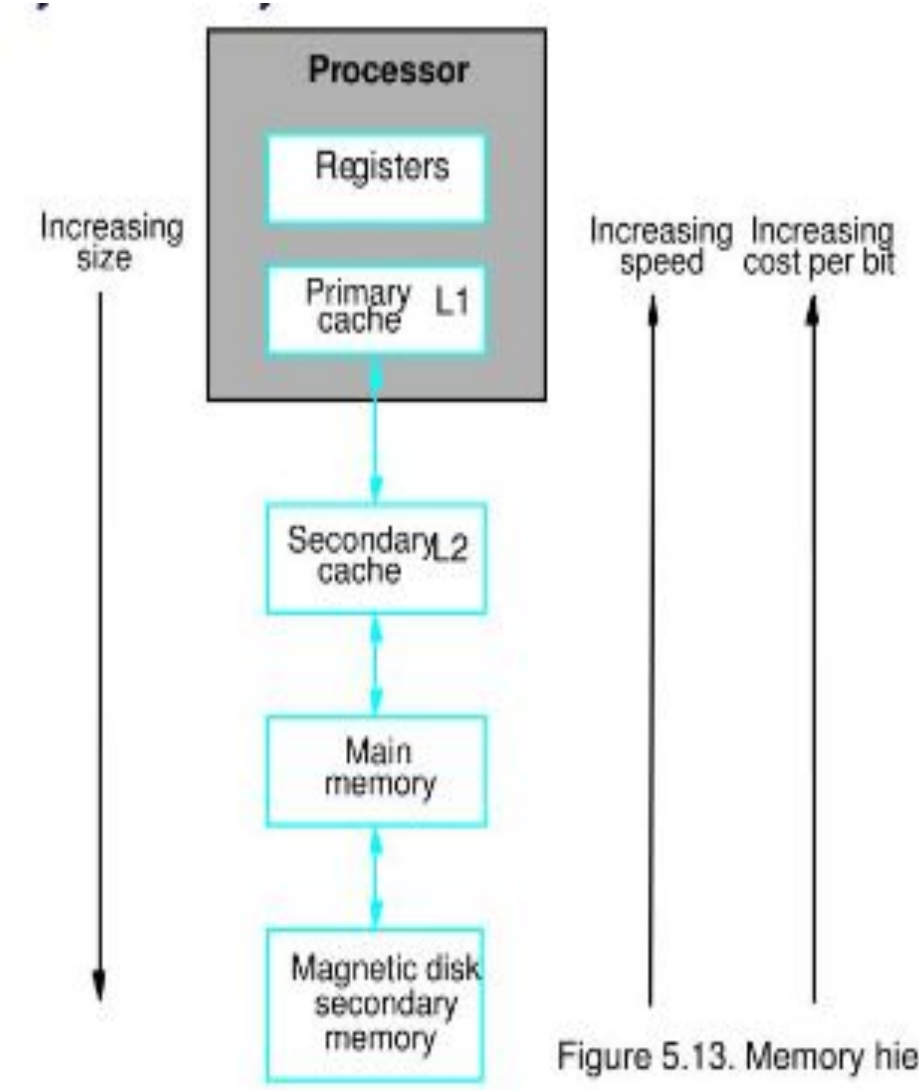
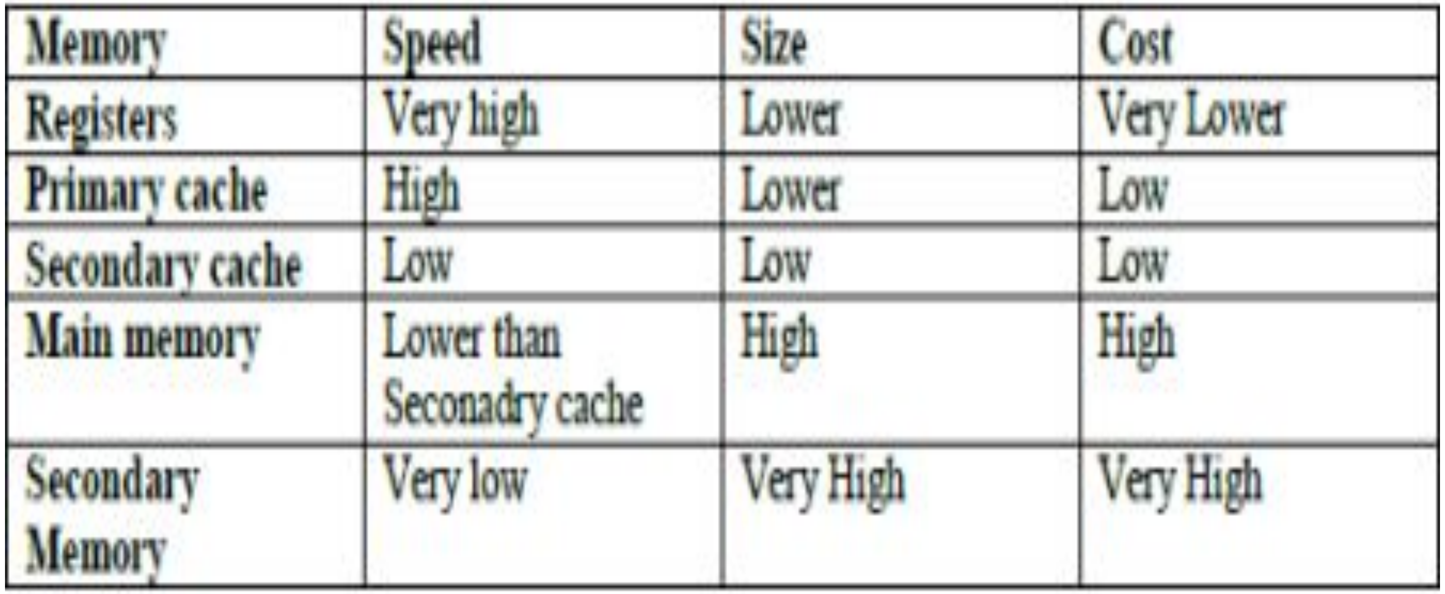


Figure 5.13. Memory hierarchy.



Memory	Speed	Size	Cost
Registers	Very high	Lower	Very Lower
Primary cache	High	Lower	Low
Secondary cache	Low	Low	Low
Main memory	Lower than Seconadry cache	High	High
Secondary Memory	Very low	Very High	Very High

❖ The fastest access is to data held in processor registers. The processor registers are at the top in terms of the speed of access.

❖ At the next level of the hierarchy is a relatively small amount of memory that can be implemented directly on the processor chips.

❖ The Cache-memory is of 2 types:

- **Primary/Processor Cache (Level1 or L1 cache)**

  It is always located on the processor-chip.

- **Secondary Cache (Level2 or L2 cache)**

  It is placed between the primary-cache and the rest of the memory

❖ The next level in the hierarchy is called main memory.

- Disk devices provides a huge amount of inexpensive storage.

Cache Memories

- The speed of main memory is very low in comparison with the speed of modern processor.
- For a good performance , the processor cannot spend much of its time waiting to access instructions and data in main memory.
- Hence , it is important to devise a scheme that reduces the time needed to access the necessary information.

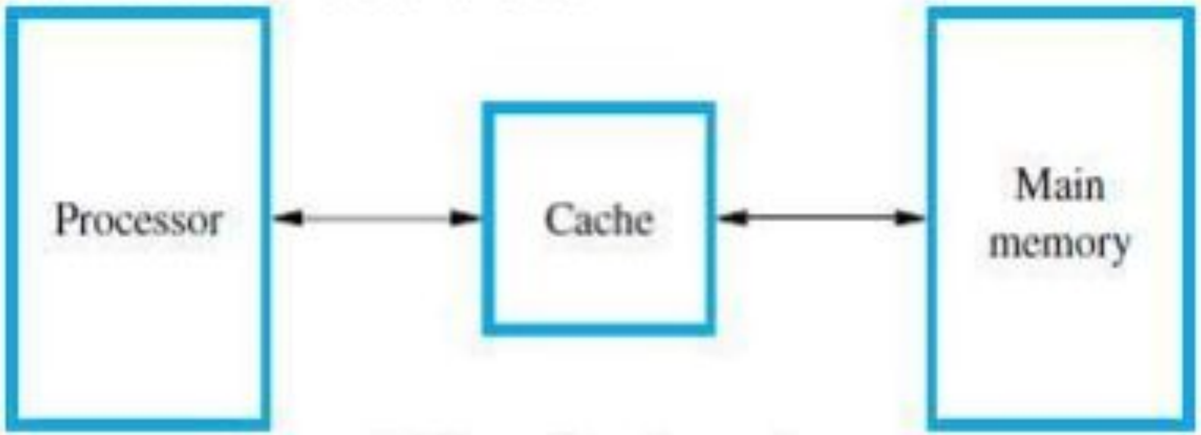


Figure 8.15 Use of a cache memory.

Cache Memories

- Cache memory is based on the property of computer programs known as "**locality of references".**

- At any given time , only some blocks in the main memory are held in the cache. Which blocks in the main memory are in the cache is determined by a **"mapping function".**

- When the cache is full, and a block of words needs to be transferred from main memory , some block of words in the cache must be replaced. This is determined by a **"replacement algorithms"**

Locality of References

- Analysis of program indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while others are accessed relatively less frequently.
- E.g., looping instructions, nested loop or few procedures calling each other repeatedly.
- Types of Locality of References:

❖ **Temporal locality of references :**

Recently executed instructions is likely to be executed again very soon.

❖ **Spatial locality of references :**

Instructions in close proximity to a recently executed instructions are also likely to be executed again very soon.

Cache hit

- The processor requests for a data if it is available in the cache then it is called cache hit.
- Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner.
- If data is in the cache it is called a **Read hit or Write hit**.
- **Read Hit :**

  The data is obtained from the cache.

E.g., a = a + b

- **Write hit :**

  e.g., a = a + b

❖ Cache has a replica of the contents of the main memory.

❖ **Write through :**

Contents of the cache and main memory may be updated simultaneously. This is called write through protocol.

❖ **Dirty bit or modified bit :**

Update the contents of the cache , and mark it as updated by setting a bit known as the dirty bit or modified bit.

❖ **Write back or copy back :**

The contents of the main memory are updated when this block is replaced. This is called write – back or copy – back protocol.

Cache Miss

- If the data is not available in the cache then it is called cache miss. It has to brought from main memory.
- It can be **Read miss or Write miss**.
- **Read Miss :**

❖ Block of words containing this requested word is transferred from memory, after the block is transferred , the desired words is forwarded to the processor.

❖ The desired word may also be forwarded to the processor as soon as it is transferred without waiting for entire block to be transferred. This is called **load-through or early – restart**

- **Write – miss :**
- **Write – through protocol** is used , then the contents of the main memory are updated directly.
- **Write – back protocol** is used, the block containing the addressed word is first brought into the cache . The desired word is overwritten with new information.

Mapping Functions

The process of keeping information of moved data blocks from main memory to cache memory is known as mapping.

- There are 3 types of Mapping functions

1) Direct Mapping

2) Associative Mapping

3) Set-Associative Mapping.

Direct Mapping

- Direct mapping is a procedure used to assign each memory block in the main memory to a particular line in the cache.
- If a line is already filled with a memory block and a new block needs to be loaded, then the old block is discarded from the cache.
- i = j modulo m,

Where,

i= cache address line

J= main memory block number

m = no of lines in a cache.

Tag bits are associated with its location in the cache.

The 5 bits of address are compared with tag bits associated with that cache location. If they match, then the desired word is in that block of the cache. If there is no match , desired word need to be loaded in the cache

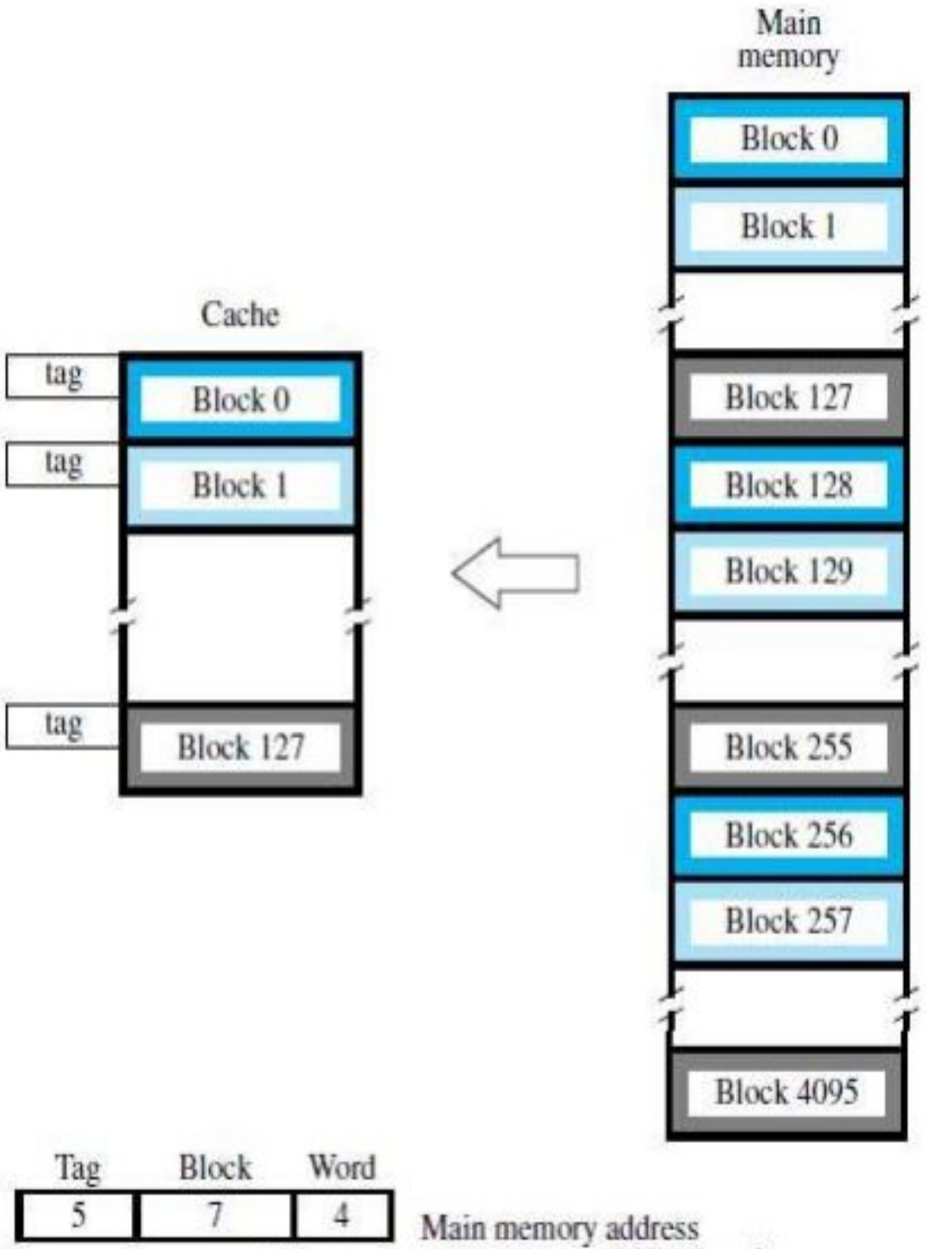


Figure 8.16 Direct-mapped cache.

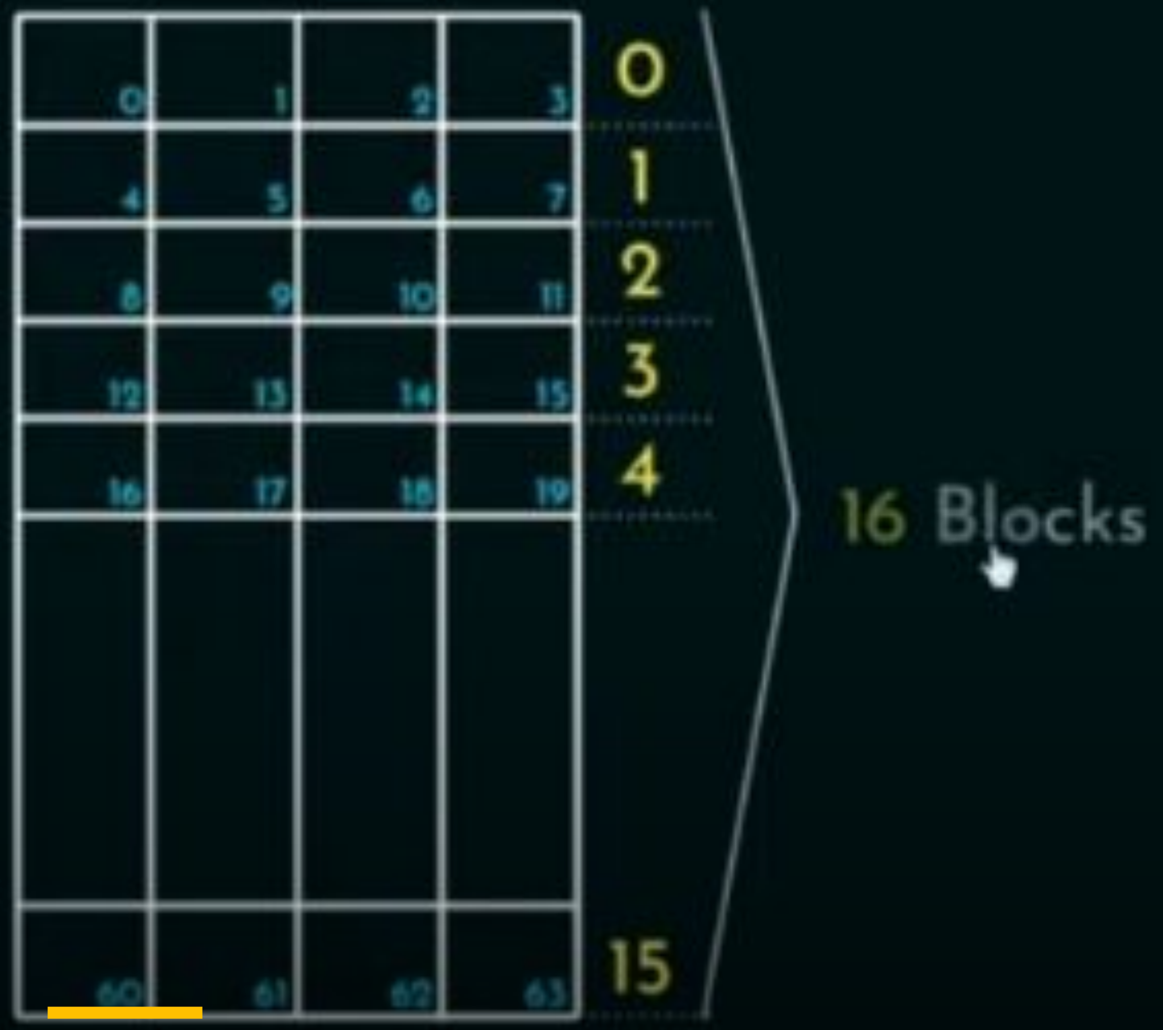
Main Memory Size : 64 words i.e. (0, 1, . . . ,63)

Block Size : 4 words

No. of Blocks in MM : 64 / 4 = 16

□

0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19
5	20	21	22	23
6	24	25	26	27
7	28	29	30	31
8	32	33	34	35
9	36	37	38	39
10	40	41	42	43
11	44	45	46	47
12	48	49	50	51
13	52	53	54	55
14	56	57	58	59
15	60	61	62	63

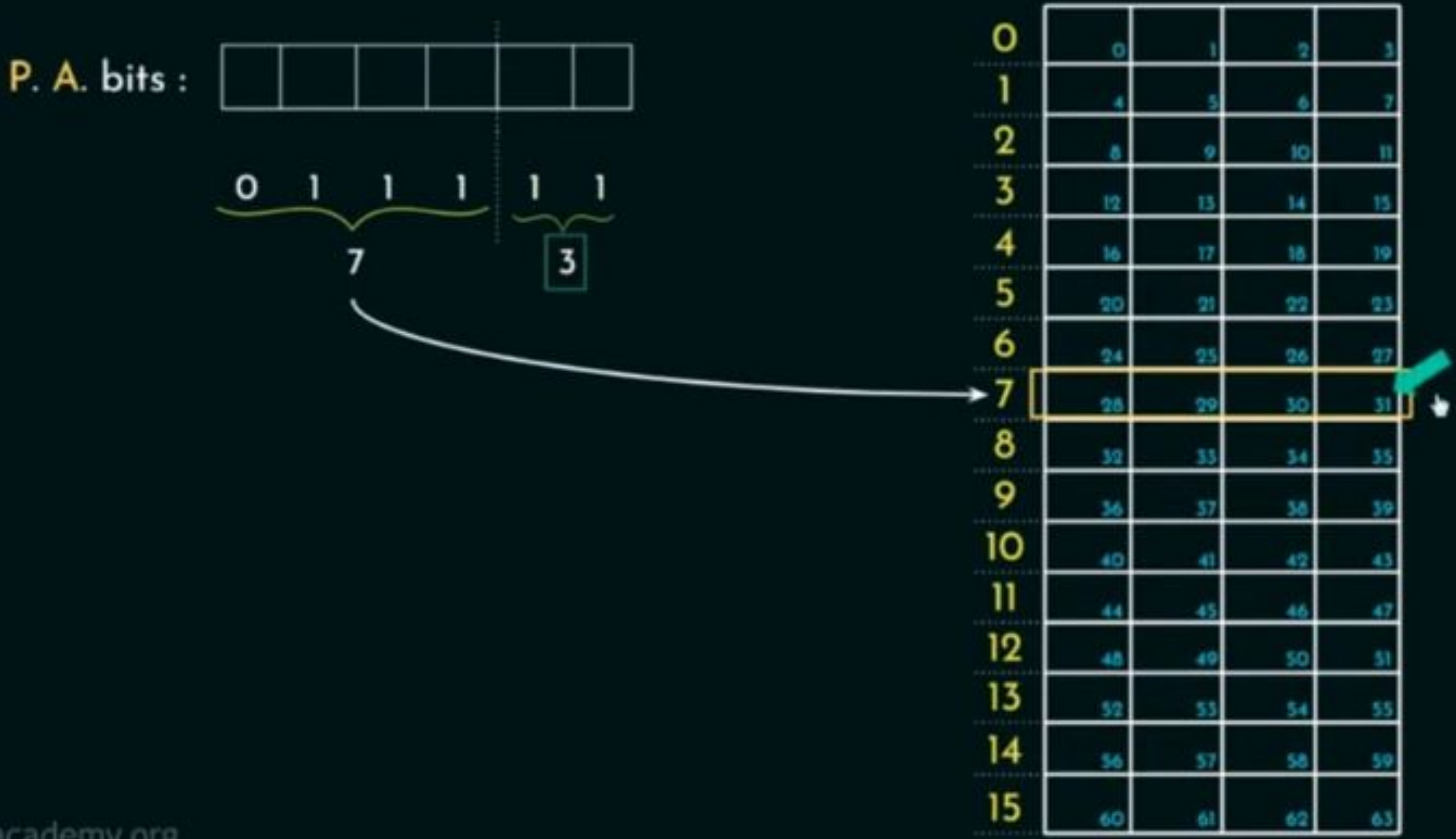


Physical Address Space

$$\log_2 64 = \log_2 2^6 = 6 \text{ bits}$$

P. A. bits : □□□□□

(Physical Address bits)



P. A. bits :

0 1 1 1 1 1

7

$\boxed{3}$

0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19
5	20	21	22	23
6	24	25	26	27
7	28	29	30	31
8	32	33	34	35
9	36	37	38	39
10	40	41	42	43
11	44	45	46	47
12	48	49	50	51
13	52	53	54	55
14	56	57	58	59
15	60	61	62	63

Cache Size : 16 words

Block Size : 4 words

Line Size : 4 words

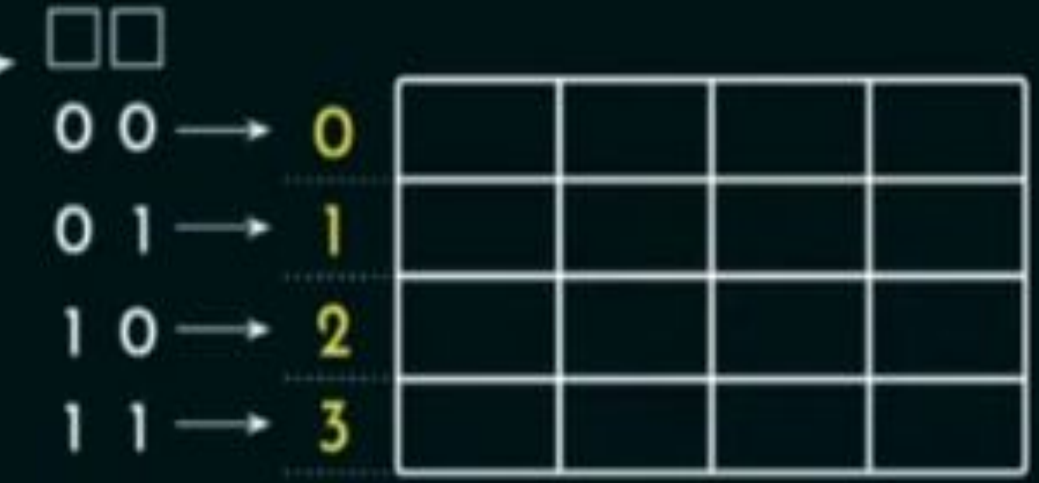
Block Size = Line Size

No. of Lines in Cache : 16 / 4 = 4 i.e. 0, 1, 2, 3

4 Lines

$\log_2 4 = \log_2 2^2$

$= 2$ bits



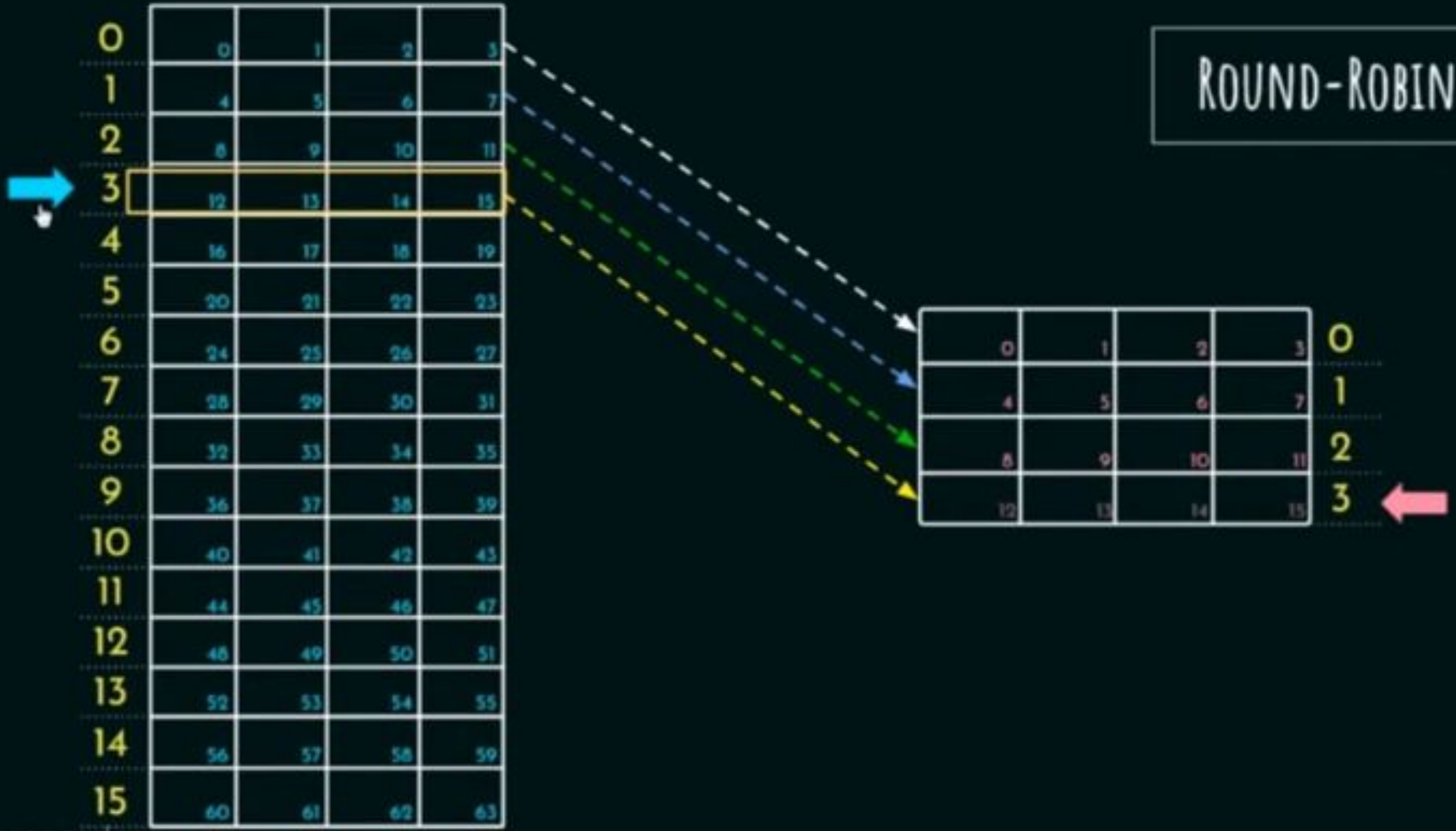
0 0 → 0

0 1 → 1

1 0 → 2

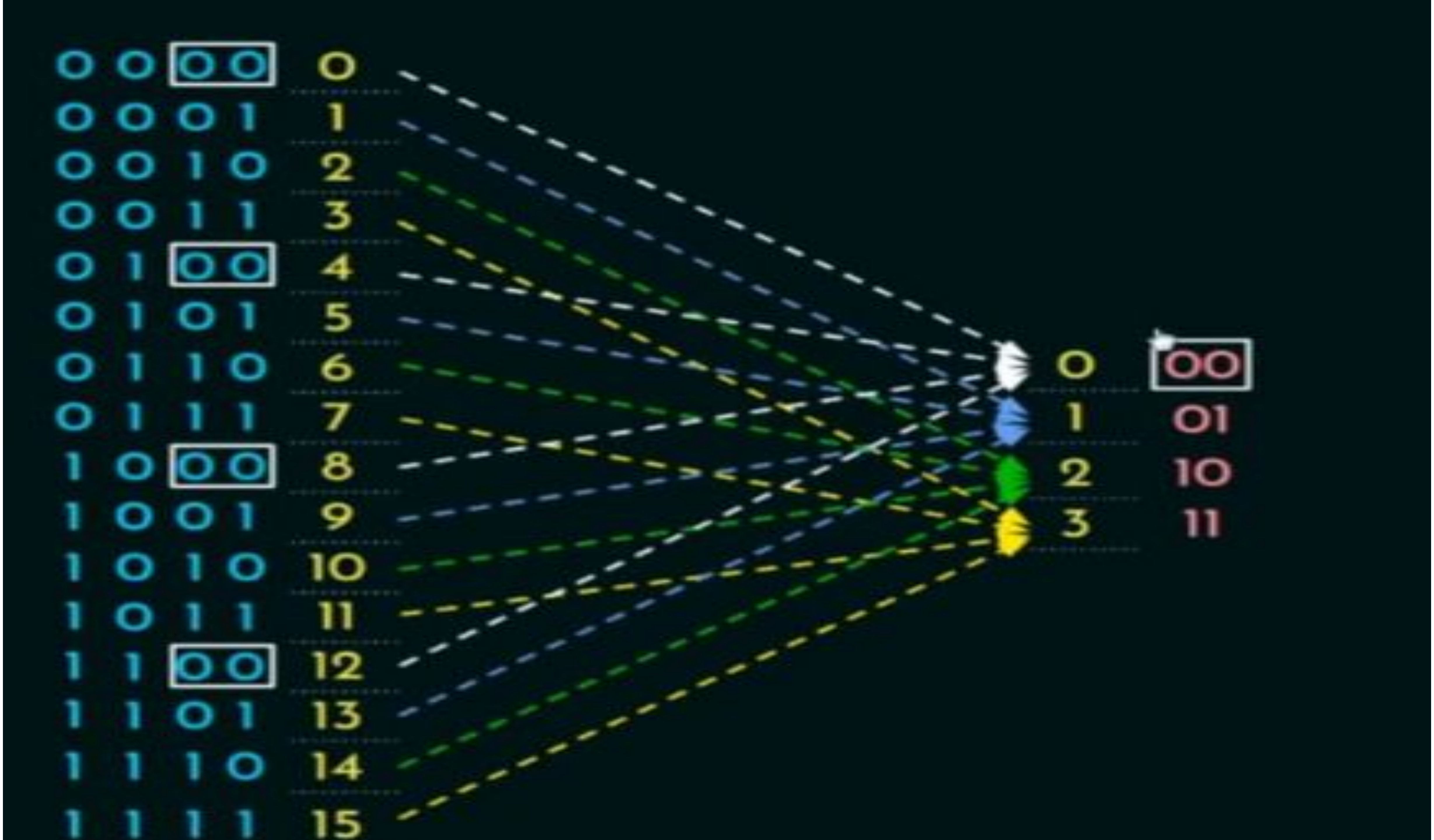
1 1 → 3

ROUND-ROBIN



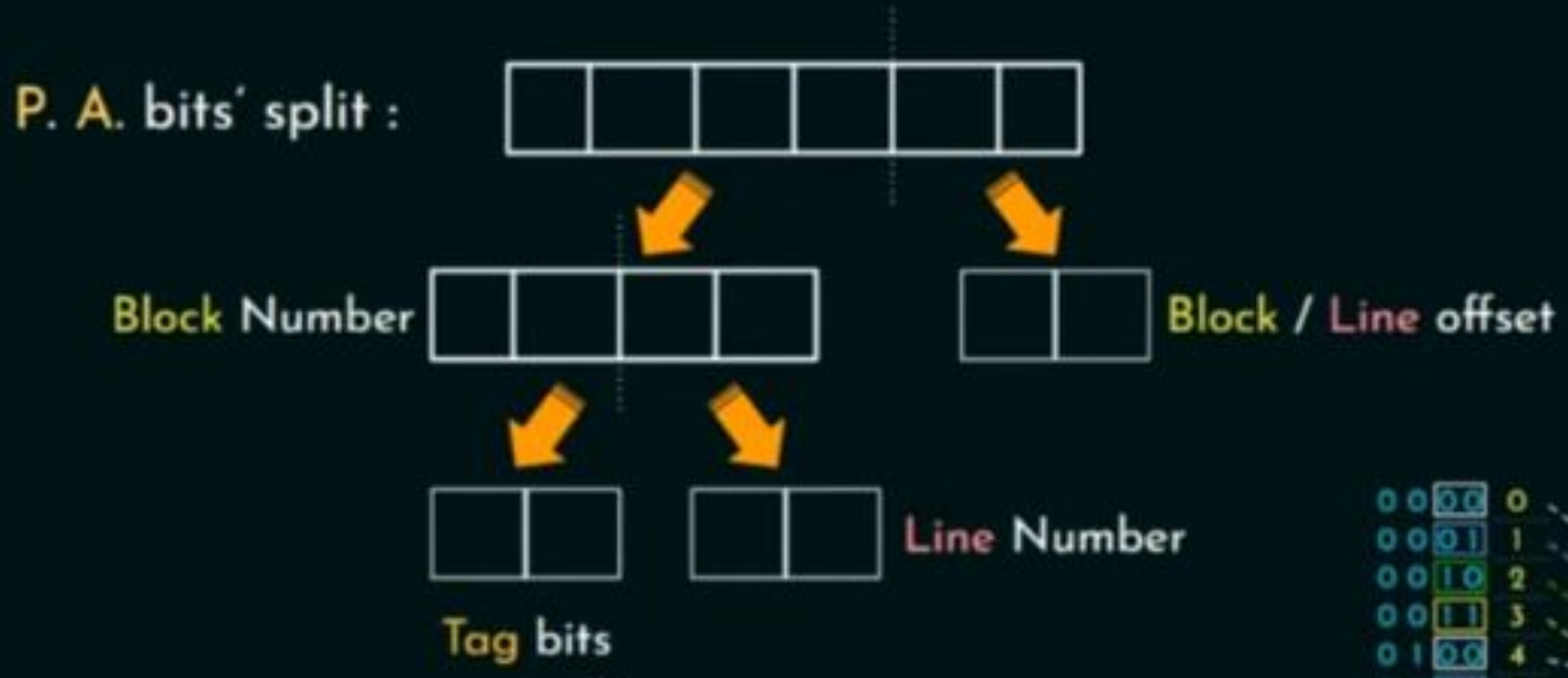
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19
5	20	21	22	23
6	24	25	26	27
7	28	29	30	31
8	32	33	34	35
9	36	37	38	39
10	40	41	42	43
11	44	45	46	47
12	48	49	50	51
13	52	53	54	55
14	56	57	58	59
15	60	61	62	63

0	1	2	3	0
4	5	6	7	1
8	9	10	11	2
12	13	14	15	3



0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	10
1 0 1 1	11
1 1 0 0	12
1 1 0 1	13
1 1 1 0	14
1 1 1 1	15

0	00
1	01
2	10
3	11



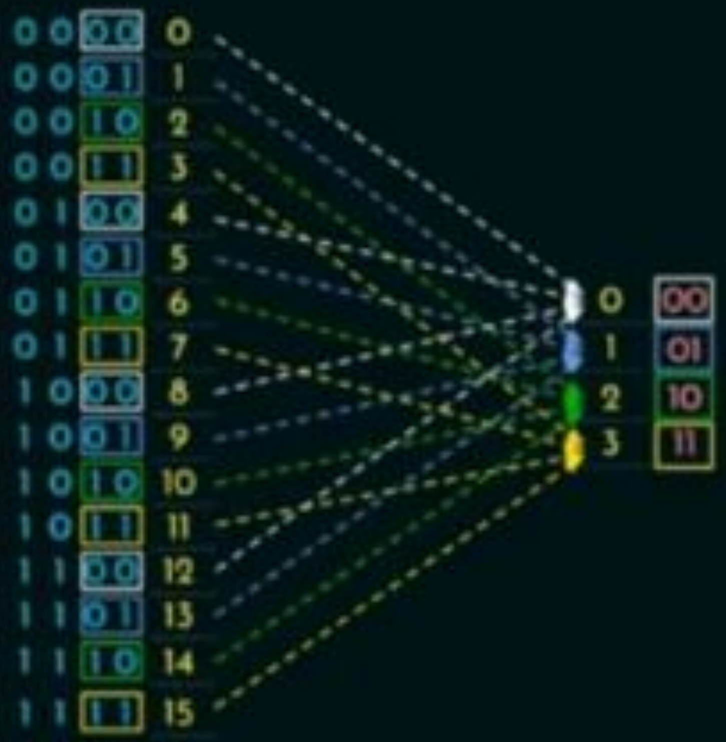
P. A. bits' split :

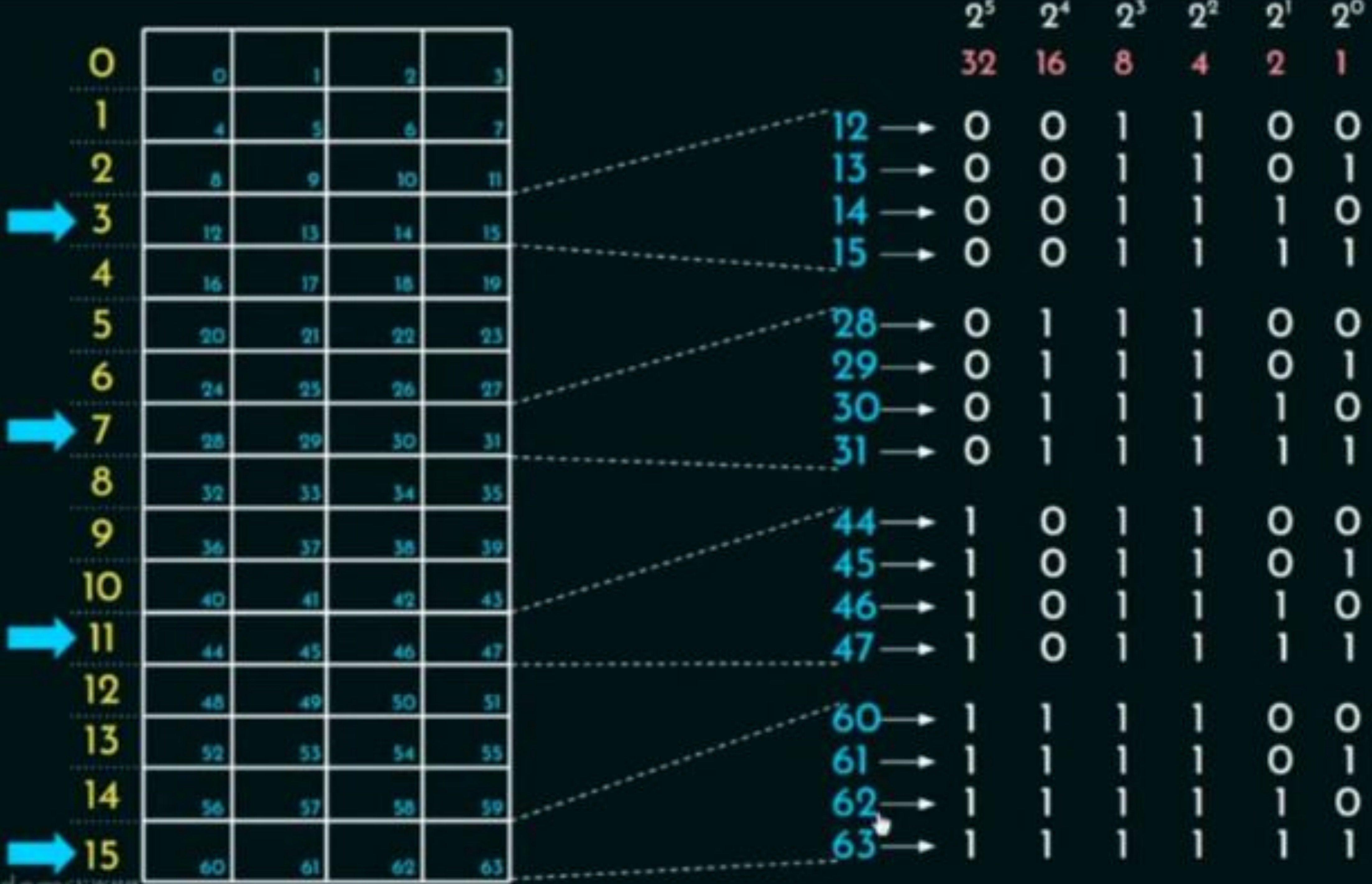
Block Number

Block / Line offset

Line Number

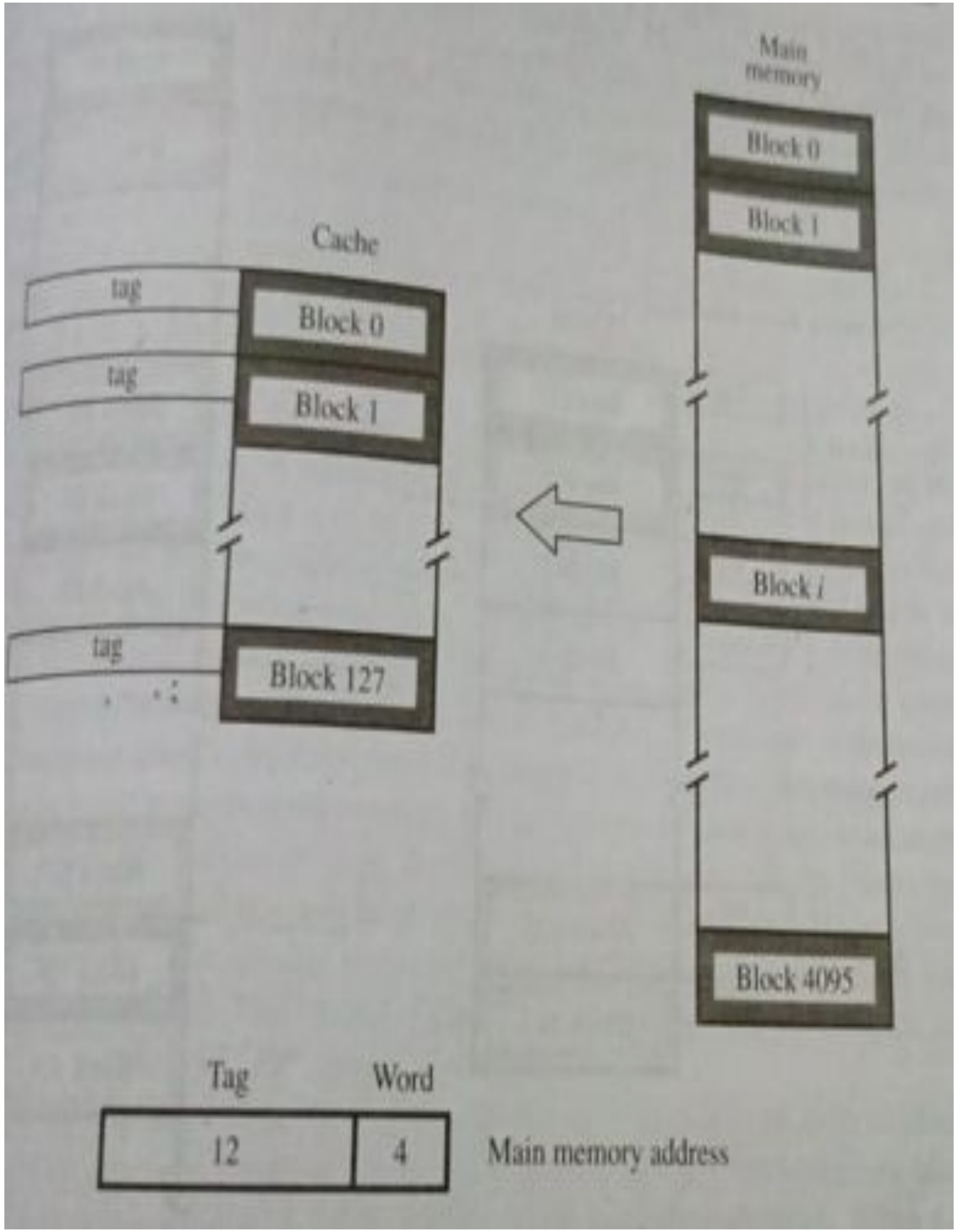
Tag bits





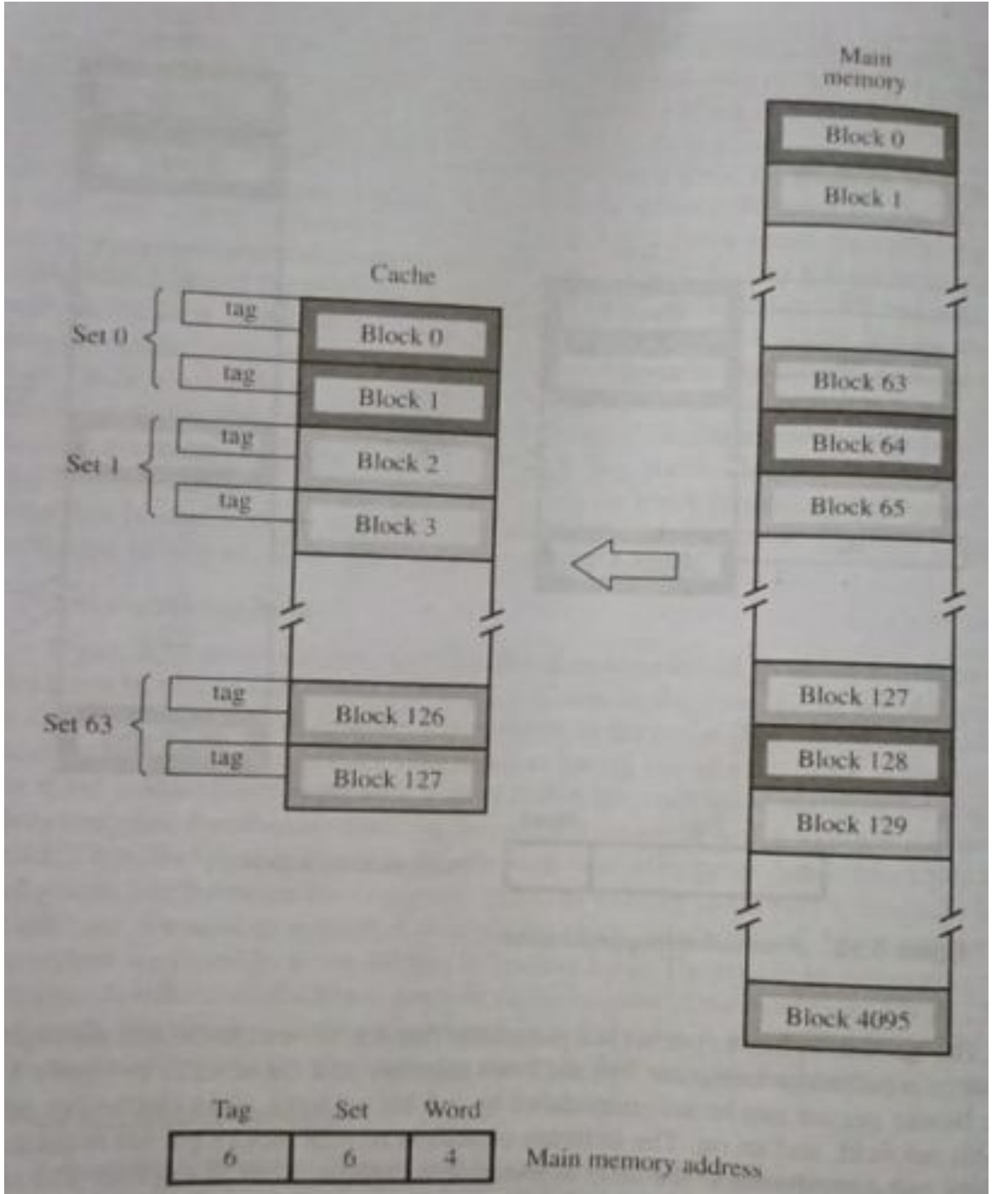
Associate mapping

- Main Memory block can be placed in any cache lines which are free.
- 12 tag bits are required to identify a memory block when it is resident in the cache.
- The tag bits of address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is present.
- A new block that has to be brought into cache has to replace an existing block only if the cache is full.
- We need many replacement algorithms to replace the blocks in cache if cache is full.



Set-Associate Mapping

- A combination of the direct mapping and associate mapping techniques can be used.
- Blocks of cache are grouped into sets, and mapping allows a block of the main memory to reside in any block of a specific sets. Hence contention problem of the direct method is erased by having few choices for block placement.
- the hardware cost is also reduced by decreasing the size of the associate search


Main memory
Block 0
Block 1
Block 63
Block 64
Block 65
Block 127
Block 128
Block 129
Block 4095
Cache
Set 0
tag
Block 0
tag
Block 1
Set 1
tag
Block 2
tag
Block 3
Set 63
tag
Block 126
tag
Block 127
Tag
Set
Word
6
6
4
Main memory address