**CIS-350**
**Infrastructure Technologies**

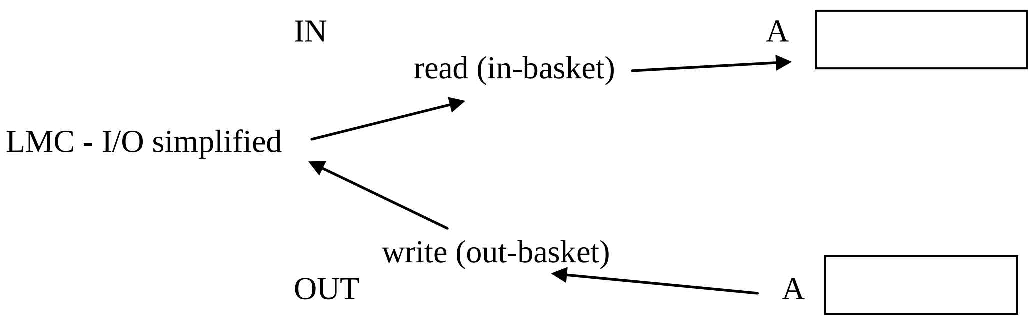
# Chapter 9 - Input/Output (I/O)

Read chapter 9.

The power and usefulness of a computer system depends not only on the CPU and memory but also on I/O capabilities. This chapter is about I/O operations/issues, not about specific I/O (peripheral) devices such as: keyboard, screen, printer, mouse, plotter, disk, and tape. The latter are covered in chapter 10.

The clock synchronizes the operation of the CPU and memory. Peripheral devices (PD) or I/O devices operate in an asynchronous manner, and are not governed by the clock.

In the LMC computer, PDs and I/O operations were simplified.




Modern computers are much more spohisticated.

## Issues/Problems to Solve

A. I/O devices operate at different speeds from each other (for example, printers are much slower than hard disks) and are much slower than the CPU and memory

B. I/O devices have different control requirements and require data to be sent to them in the form that they (I/O devices) can understand

C. Data Transfer (Speed and Amount) – some I/O devices transfer one character at a time (keyboard), others transfer large blocks of data (disk drives)

D. How does the CPU distinguish between many peripheral devices?

E. How does an I/O device communicate with the CPU?

F. How does the CPU respond to an interrupt?

## Solutions

A. I/O devices operate at different speeds from each other (for example, printers are much slower than hard disks) and are much slower than the CPU and memory.

Speed of the

CPU – 1 GHz (the clock pulse occurs every 1ns),
– 2 GHz (the clock pulse occurs every 0.5 ns)

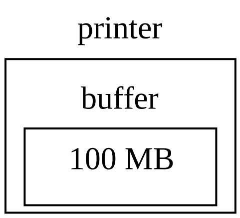
Registers have almost instant access
SRAM - 10ns (access time) (1ns=$10^{-9}$s)
DRAM - 30ns
hard disk - 10ms (1ms=$10^{-3}$s)

CPU and memory are about $10^6$ faster than hard disk.

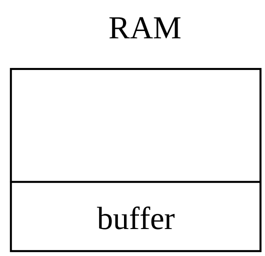
How is this speed disparity solved?

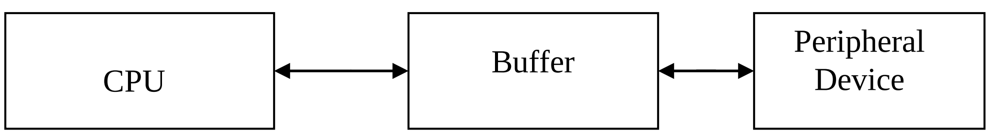
Solution: Buffers are used to alleviate speed disparity

A peripheral device may have its own memory/buffer




A portion of RAM is allocated as a buffer



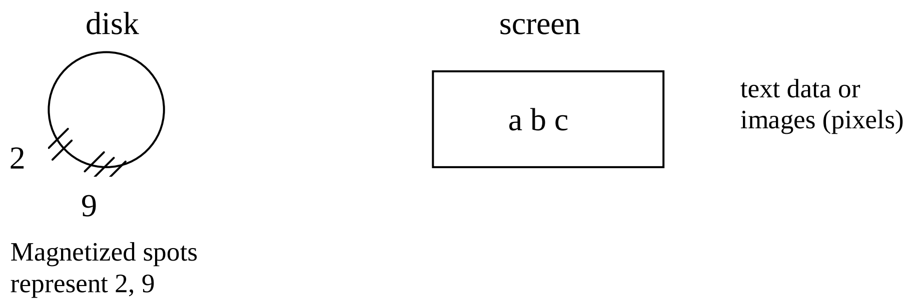

CPU writes/reads data to/from a buffer at the buffer's speed

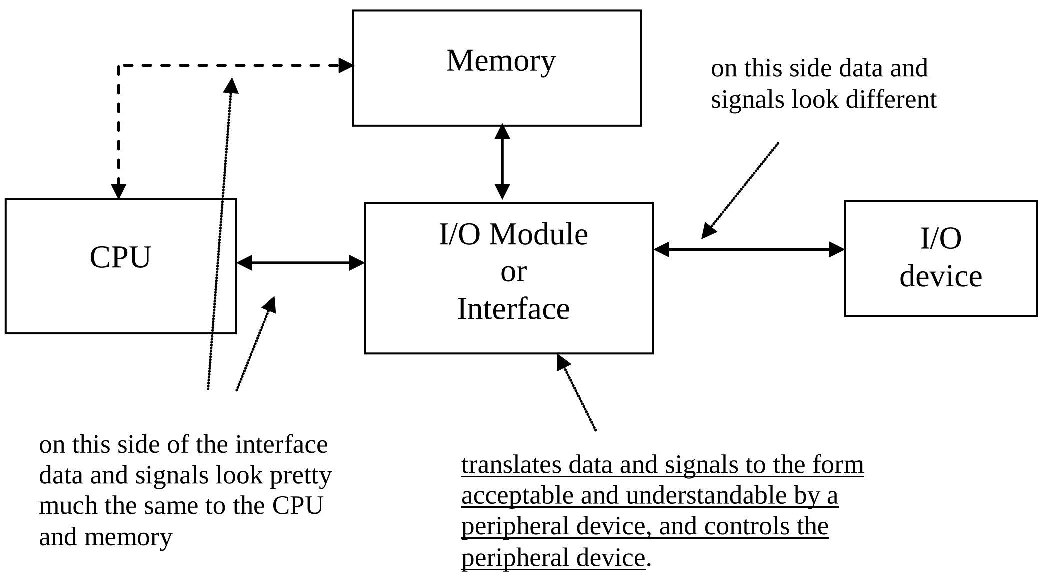
When a buffer is empty, it will be replenished with data

PD writes/reads to/from the buffer at the PD's speed

B. <u>I/O devices have different control requirements and require data to be sent to them in the form that they (I/O devices) can understand</u>.

Electronically, peripheral devices are very different.




Solution: Each I/O device has its own <u>unique</u> interface (PC). <u>The interface translates data and controls the peripheral device</u>.

<u>Synonyms</u>: interface - board, card, controller, I/O module (the term used in the textbook)

To be able to <u>initiate</u> the data transfer between memory and disk, the CPU needs to:

- have the address of data in memory and on disk
- know the size of the block to transfer, and
- issue the r/w signal concerning the direction of the transfer (write: memory $\rightarrow$ I/O device; or read: memory $\leftarrow$ I/O device)

This information has to be passed to the I/O module/interface.

The interface executes the primitive I/O program to accomplish and monitor data transfer. For example, the I/O disk interface controls the R/W head and its movement to position it on an appropriate track:

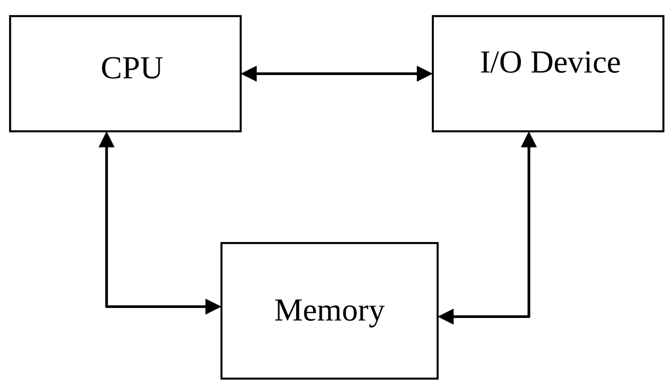
In other words, the operations:

SEEK track 20
WRITE to track 20
READ from track 20

are performed by the interface, not by the CPU. The CPU only initiates the communication with the peripheral device. When the I/O is complete, the I/O module will notify the CPU by sending an electronic signal called an interrupt.

Slightly More Complex I/O Module Arrangement – Fig. 9.3, p.283

C. Data Transfer (Speed and Amount) – some I/O devices transfer one character at a time (keyboard), others transfer large blocks of data (disk drives)




- one word at a time (one character at a time) - programmed I/O
- one block at a time – Direct Memory Access (DMA) transfer

CPU always initiates the data transfer and then does something else.

I/O devices operate under CPU program control, but the CPU does not monitor or supervise peripheral devices all the time.

**Characteristics of I/O devices** with respect to speed and quantity of data transfer

Figure 9.1, p. 281, presents data rates for typical I/O devices (discuss).

a. keyboard - input device, character-based device, you enter one character at a time, 100bps.

b. printers - output device, receive blocks of data, have their own memory (buffers), operate over a wide range of data rates, laser printer – 3.2 Mbps.

c. graphics display (computer screen) - output device, receive blocks of data, bitmap images - pixels consume much memory space, also updates and display must be fast – 800-8000 Mbps.

d. disk - I/O device, store the entire programs and data files, permanent storage, data is transferred in blocks at rate 240-3000Mbps; if disk serves as a server, it operates even faster.

The concept of a physical and logical record (block) (obsolete)

For efficiency reasons, a physical record (block) is transferred by hardware.

| 50000 bytes | | | | | |
|---|---|---|---|---|---|
| 500 bytes | | | | | ------- |

logical record is accessed by application software

In this example, we have 100 logical records in one physical record.

During the read operation, the OS module called IOCS extracts logical records from a physical record and passes them on to the application program. During the write operation IOCS collects records generated by an application program and transfers them as one block (physical record) to a peripheral device.

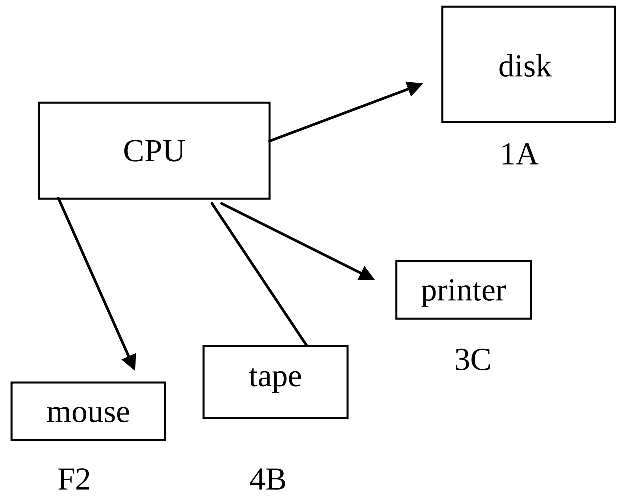
The concept of physical and logical records is obsolete. It comes from batch processing in COBOL that uses the traditional data structure: file, record, field, and byte.

How the data is transferred depends on the OS. For example, some modern OS such as UNIX transfer data as a stream of bytes. They do not impose any structure on data.

D. How does the CPU distinguish between many peripheral devices?

Solution: Each peripheral device has its own unique address.

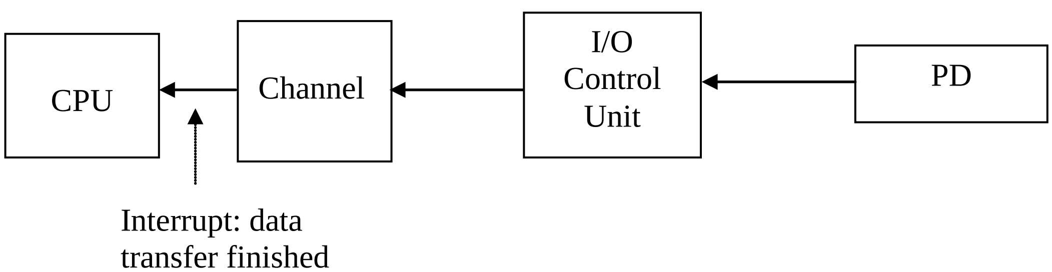
This is how the CPU distinguishes between various I/O devices. The address of the currently accessed PD is stored in the CPU's I/O address register.



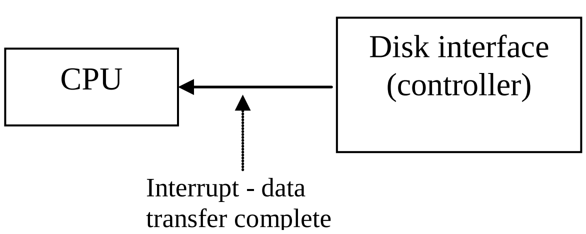

E. How does an I/O device communicate with the CPU?

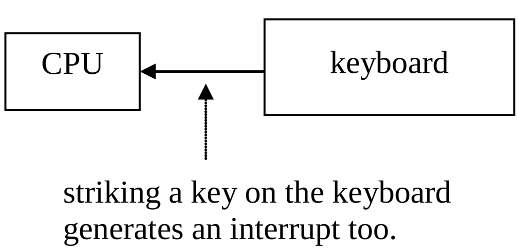
Solution: A peripheral device communicates with the CPU by sending an electronic signal called an interrupt to the CPU. An interrupt indicates that a special event happened.

IBM



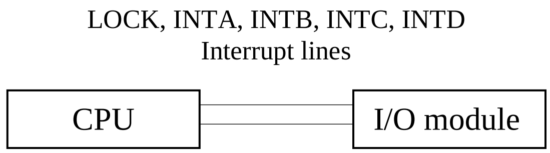

PC



Interrupts are sent through interrupt lines usually denoted on diagrams by INT (interrupt) or IRQ (interrupt request).

See Figure 11.8, p.353 - PCI bus connections




F. How does the CPU respond to an interrupt?

Solution: The CPU executes an interrupt handling routine to service an interrupt.

We will revisit interrupts in a moment as they are they important and occur very frequently. Simply, computers are interrupt-driven machines.

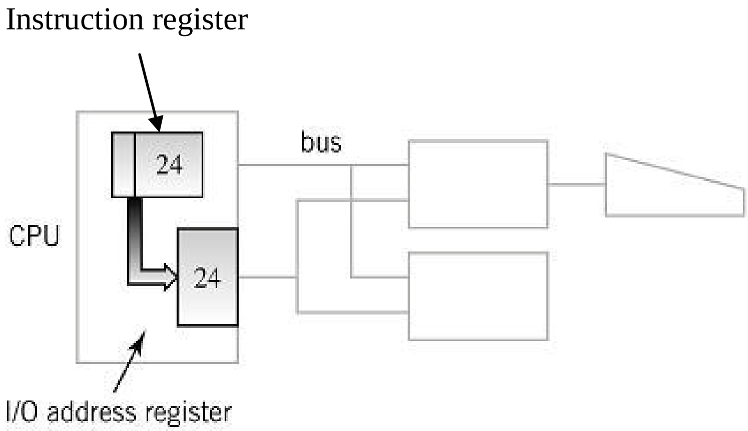
## METHODS/ARCHITECTURES FOR DATA TRANSFER

- Programmed I/O
- Direct Memory Access Concept
- Channel Architecture

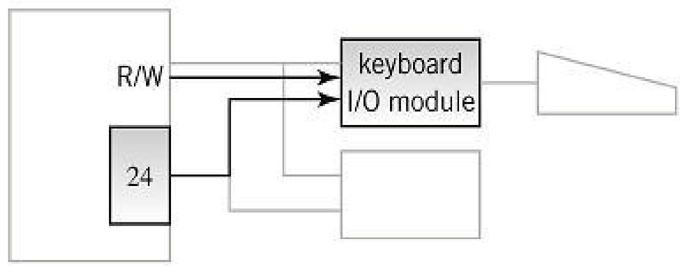
### Programmed I/O

The LMC uses programmed I/O only - I/O operations are performed directly under program control. In programmed I/O, one character is typically transferred at a time. Each instruction produces a single input or output. Programmed I/O is very slow and used primarily with keyboards (int a; read a; print a; ). It is also used for transmission between the CPU and the I/O modules to set up parameters for the Direct Memory Access (DMA) transfer and to initiate I/O operations. The CPU is heavily involved in every step of programmed I/O operation.

See Figures 9.3 and 9.4 in textbook to understand programmed I/O. Programmed I/O is not suitable for transfering large amounts of data. This requires high-speed data transfers. For high-speed data transfers, the DMA concept (PC) or channel architecure (mainframe) is used. In both latter methods, the CPU is not heavily involved in I/O operations and can be released to do other, more important, tasks.

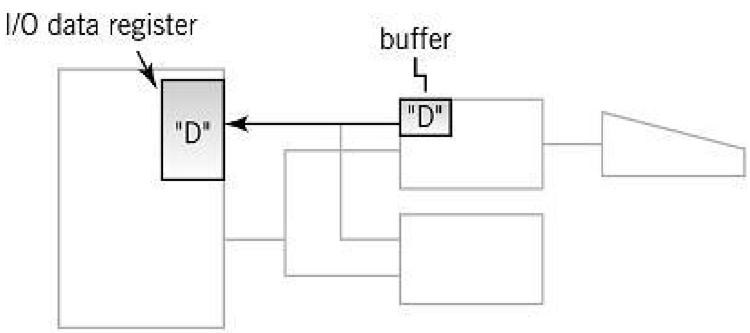



1. CPU executes INPUT 24 instruction. Address 24 is copied to the I/O address register.

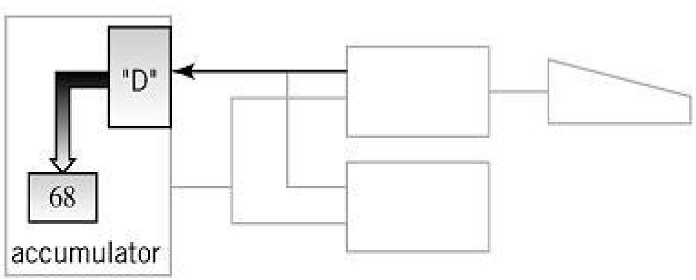



2. Address 24 is recognized by the keyboard I/O module. A read/write control line indicates that the instruction is an INPUT.

(Figure continues on next slide)



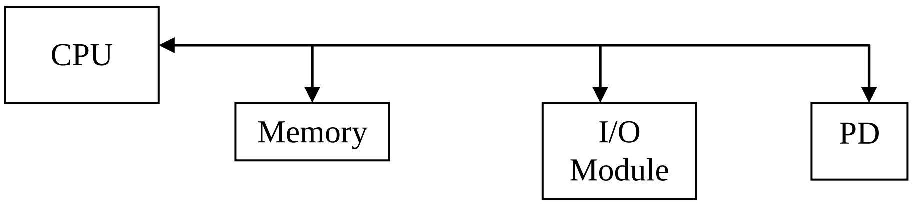

3. A buffer in the I/O module holds a keystroke, in this case ASCII 68, the letter "D". The data is transferred to the I/O data register.



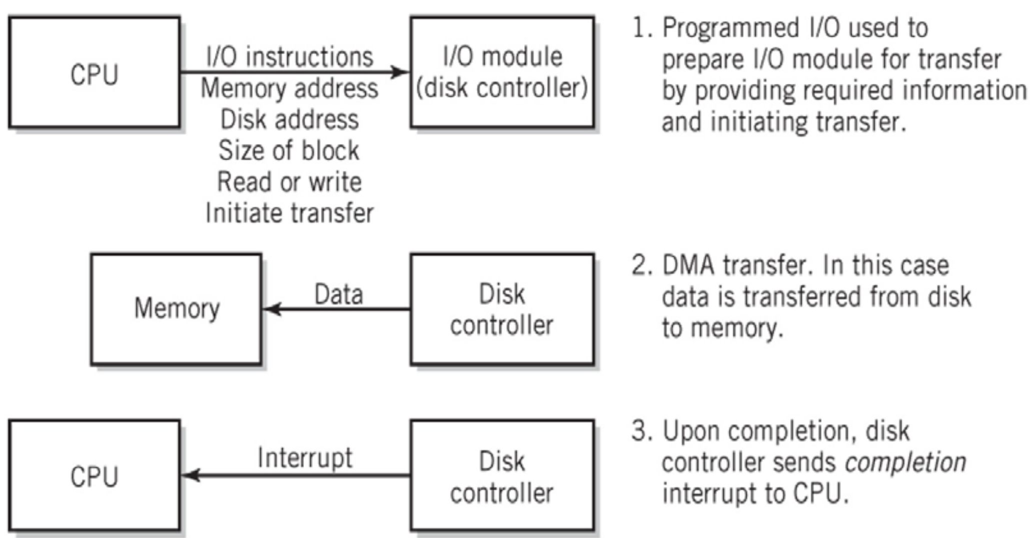

4. From there it is copied to the appropriate accumulator or general-purpose register, completing the operation.

Direct Memory Access (DMA) Concept

The I/O module has a direct access to memory. DMA is particularly suited for high-speed disk transfers. Data transfers are accomplished without involvement of the CPU. The CPU can do other tasks when I/O transfers take place.




I/O modules use special software programs called device drivers to translate data and control peripheral devices. Device drivers are often built into the operating systems or have to be installed separately. For example, there are hundreds of printers on the market today. There is one printer interface installed in the computer for all these printers. However, each printer is equipped with the unique device driver that is added and integrated at the time of OS installation. Plug-and-play feature in Windows - the OS can detect the type of peripheral devices connected to the system and installs their corresponding device drivers automatically, on the fly, without powering the system off. Most common device drivers are already included in the OS kernel. For example, about 50% of Unix kernel contains device drivers. They can also be downloaded from the Internet.

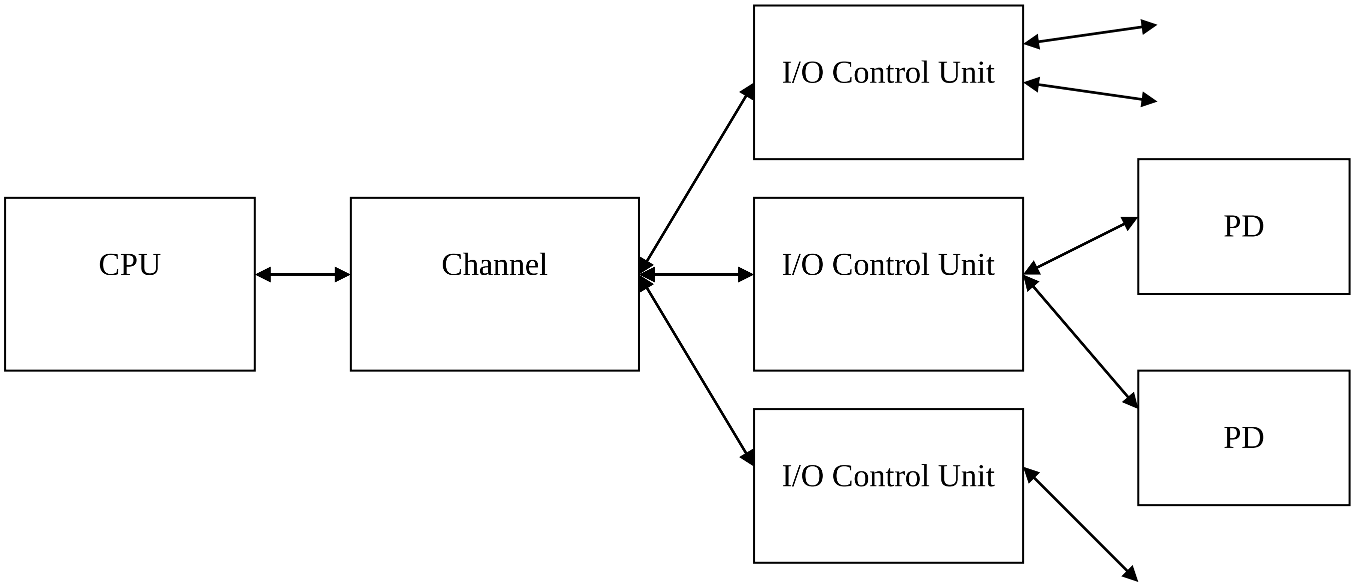



Channel Architecture - Mainframes (IBM/z System)

Need more intelligent interfaces for the following reasons:

- expensive
- used by many users simultaneously
- execute many programs concurrently
- CPU has to be utilized well.

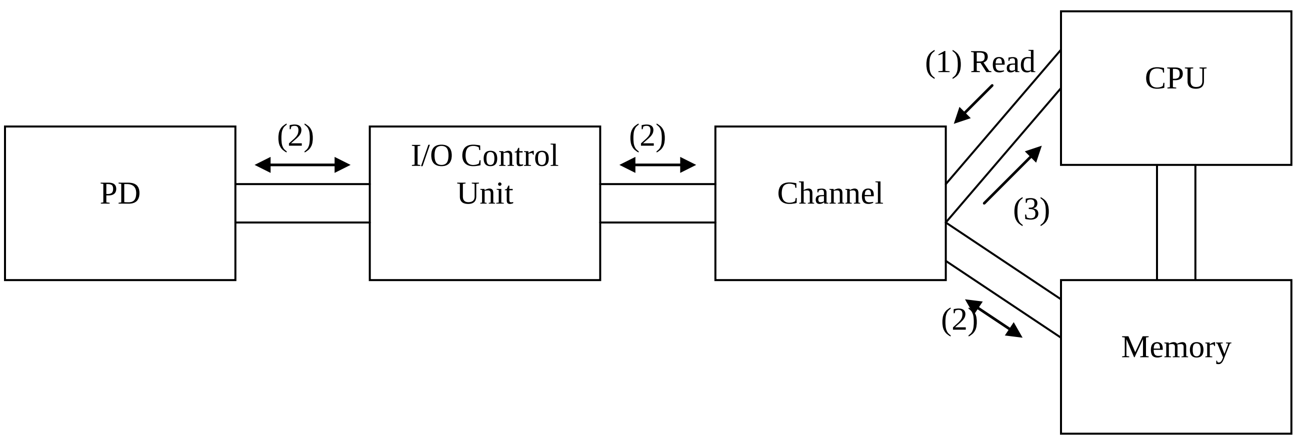
Different architecture is needed.




A channel (front-end processor or back-end processor)

- is equipped with its own processor
- releases the CPU to do other tasks
- is responsible for I/O operations
- is CPU dependent
- is device independent

The channel and the I/O Control Unit translate data and control I/O devices. IOCU is device dependent - must be unique to a peripheral device. The channel executes the channel program to accomplish the data transfer.




1. Channel receives the I/O signal (Read) from the processor which is executing program A.
2. The channel assumes the responsibility for this operation and releases the processor to do other tasks (program B). The communication between the processor and the channel is broken temporarily. The channel establishes the link (communication) with the peripheral device and supervises the data flow between the peripheral device and memory. While the channel handles data for program A, the CPU executes program B.
3. After the Read operation is complete, the channel issues an interrupt signal to the CPU.

The channel is an asynchronous device that works independently from the processor.

INTERRUPTS

An interrupt is an electronic signal detected by the computer hardware. It indicates that a special event happened.

The mechanism through which an interrupt is accomplished is as follows.

a. CPU executes program A
b. An interrupt occurred
c. CPU suspends the execution of program A and saves all information about the program in the program control block

 (PCB) contains

-- the program counter (PC) content - the address of the next instruction.
-- data stored in all registers
-- process #
-- all information about files used
-- any outstanding I/O requests

d. CPU calls an interrupt handling routine to service the interrupt.
e. When the interrupt handling routine is over, the CPU retrieves all information about program A from the PCB and resumes execution of program A.

The PCB is just a table storing all important information about the suspended program. The PCB is usually stored in main memory in the stack area.

PCB is often called the process control block. The concept of the program and the process are not the same. The difference between them is very profound, in particular, to the OS. The program is a static entity which resides on disk and does not consume resources such as memory and CPU. The process is a dynamic entity with all the resources (CPU and memory) allocated to it by the OS.

All interrupts originate in hardware or software.

When an interrupt occurs, the CPU:
- will finish the currently executing instruction; or
- will not.

It depends on the type of the interrupt.

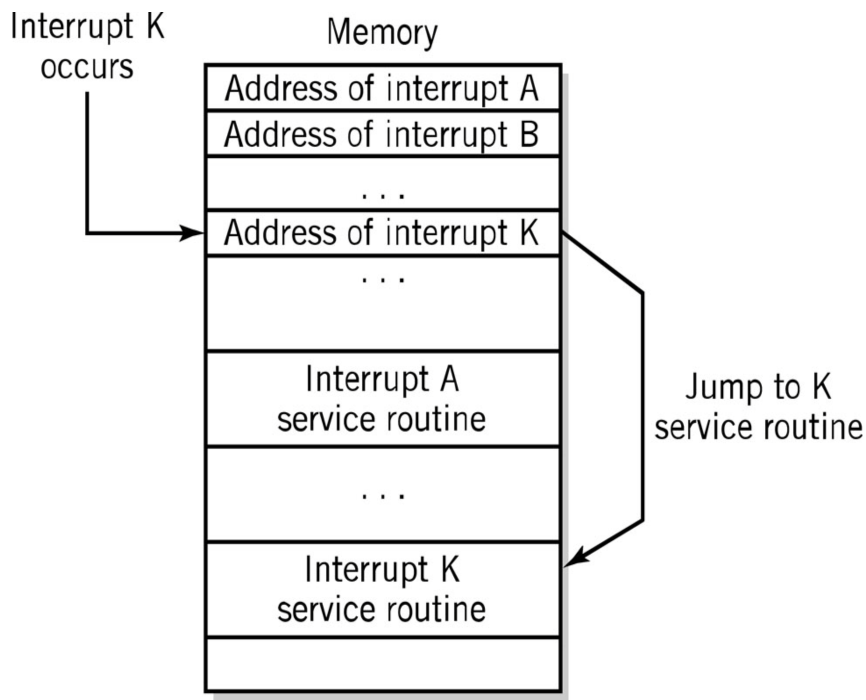
When an interrupt occurs, there are two questions that need to be answered.

(1) How does the computer identify the interrupting device?

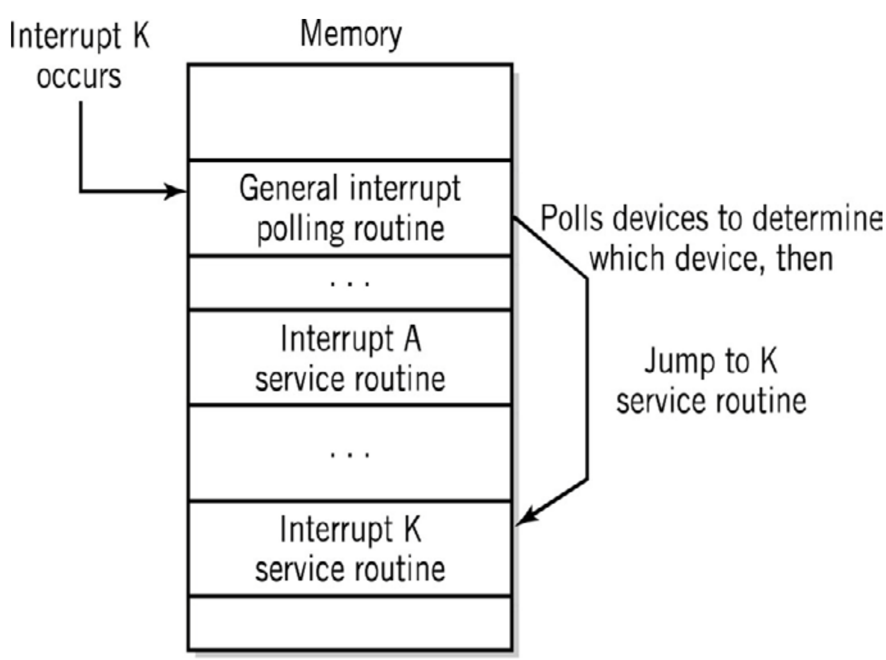
(2) Are there other interrupts awaiting service, and, if so, how does the computer determine the order in which the interrupts get serviced?

(1) Two different processing methods are used for determining which device initiated an interrupt: vectored interrupt and polling. In vectored interrupt method, the address of the interrupting device is included as part of the interrupt. Polling method identifies the interrupting device by polling each device. See Figures 9.10 & 9.11.

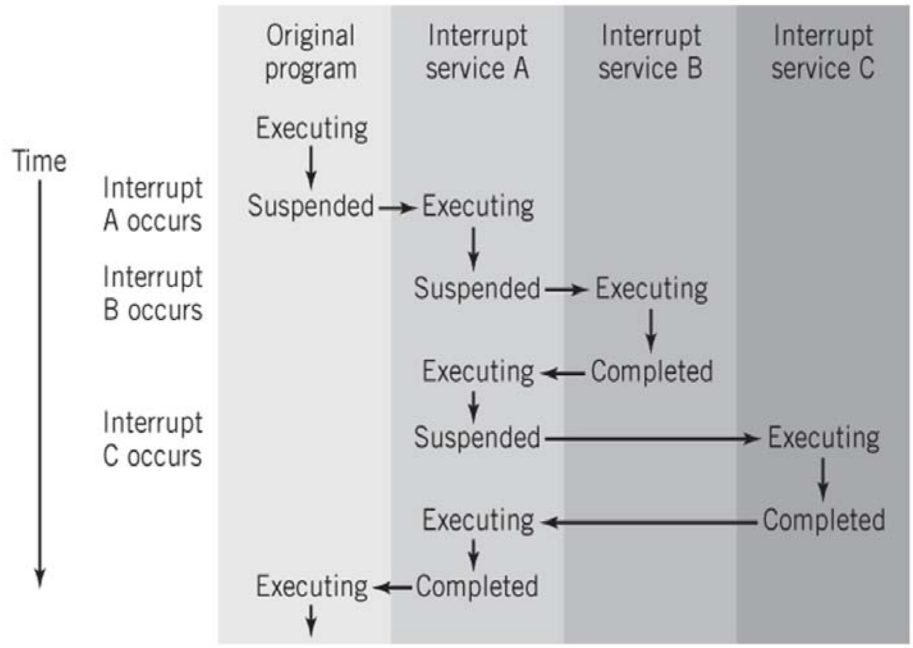
Vectored interrupts




Polled interrupts



(2) Multiple interrupts (Fig. 9.12) can be handled by assigning priorities to each interrupt. Top priority interrupts are handled first. A higher-priority interrupt is allowed to interrupt an interrupt of lower priority, but a lower-priority interrupt has to wait until a higher-priority interrupt is completed.




We can infer form the above figure that Interrupt B has the higher priority than Interrupt A, and Interrupt C has the higher priority than Interrupt A.

On the IBM zSeries Family, interrupts are divided into 6 classes, with the priorities shown below. (Figure 9.9, p. 293)

1. machine check interrupt (nonrecoverable or recoverable hardware errors)
2. Supervisor call (software interrupt requested by program)
3. Program check (software errors: division over 0)
4. External (interval timer expiration)
5. I/O completion signal
6. Restart

All the different types of interrupts within each class are handled by the interrupt handling routine (IHR) for that class.

The PSW is a very important 16-byte Program Status Word (register) on the IBM mainframe. It contains fields from which one can read out the significant information about the status of the current application program and status of the system itself. For example, among other things, it stores the Program Counter.

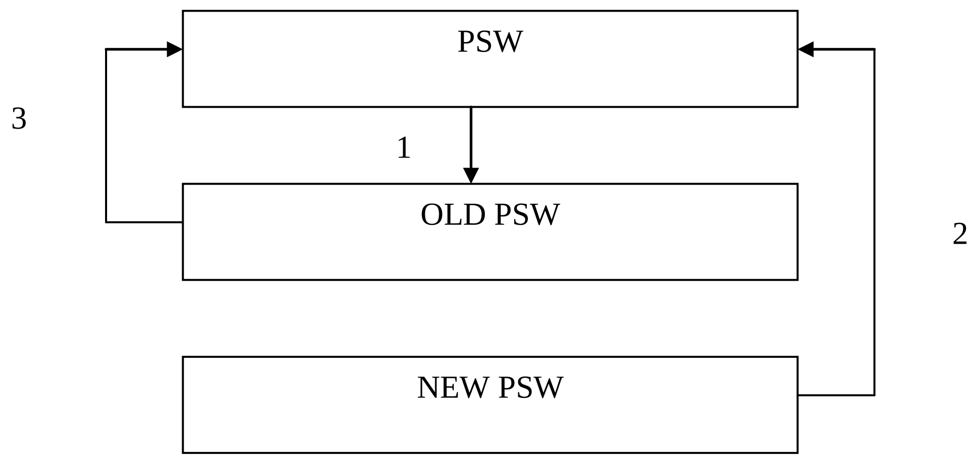
The NEW PSW stores the address of the interrupt handling routine.

The OLD PSW will store the address of the application program after an interrupt occurs.

Because there are 6 different types of interrupts, there are 6 NEW PSWs and 6 OLD PSWs. Eight-byte low memory locations are allocated for each.

On the IBM z Series, interrupts are processed by switching Program Status Words (PSW, OLD PSW, and NEW PSW).

1. When an interrupt occurs, the contents of the PSW which contains the address of the current instruction in the application program is stored in the OLD PSW.
2. The content of the NEW PSW that contains the address of the IHR is loaded into PSW. As a result, the IHR executes.
3. When the IHR is completed, the content of the OLD PSW is loaded back to the PSW resuming execution of the application program.




Also, see
Figure 9.13, p. 296

## Examples (Origin of Interrupts)

### Hardware

1. From the channel or I/O, module (I/O completed)

2. From the timer, (CPU time exceeded)

   (time slice of, say, 10 ms allocated to the program has been used, and the program did not surrender control to the OS)

3. From the hardware, (hardware error)

   Occur unexpectedly, unknowingly

4. From the system operator
   hot job entered the system; all other programs need to be suspended

### Software

1. I/O Request in the program
   open ('R', "c:/myfile.dat")
   read from the file

2. Illegal program instruction
   a=5;
   b=0
   c=a/b; ← division over 0

3. SVC call (Supervisor Call)

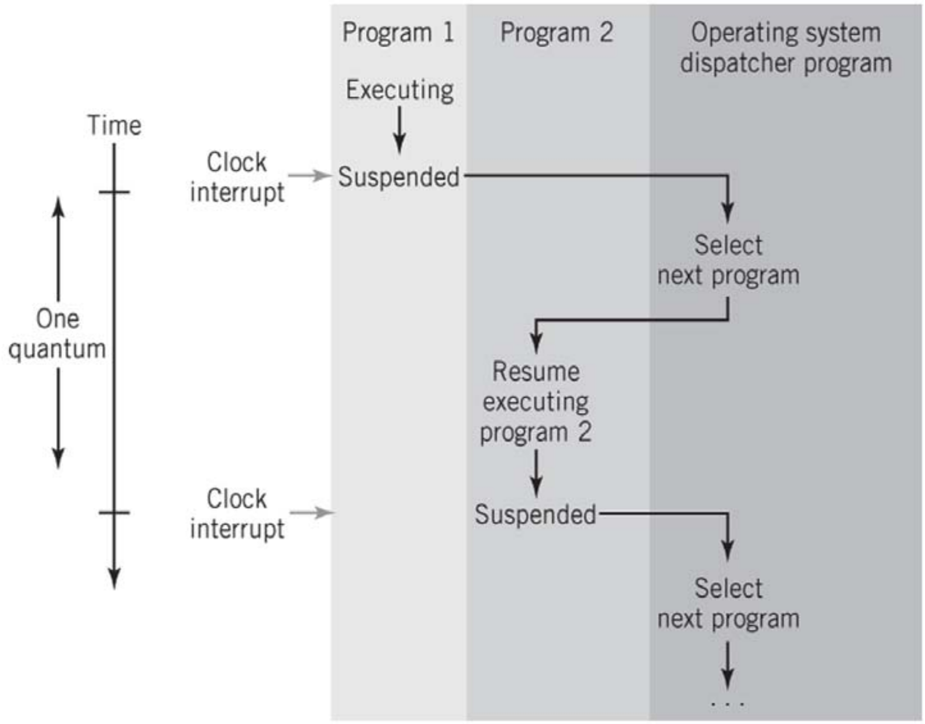
   ```
   LOAD 90
   ADD   92
   CALL SUB1
   SVC 17
   ```
   (control is transferred to the IHR)

   increase storage dynamically or halt the program

   Does not occur unexpectedly, it's issued on purpose, the programmer requests a specific service from the OS. Privileged instruction.

   You can issue the SVC interrupt when you know a great deal about computers and assembly language

Using an interrupt for time-sharing.




The figure above also shows the concept of context switching – very important. When Program 1 is suspended its PCB is saved on stack, and before Program 2 starts/resumes executing its PCB has to be retrieved from stack. Context switching obviously represents an overhead, but it is performed very fast because it is implemented in hardware. In multiprogramming, where several programs are executed concurrently, context switching is very common.

Summary:

The CPU and main memory are synchronized by the clock.
I/O devices function independently and are asynchronous.
In DMA transfer, the CPU is released to do other tasks.
Interrupts originate in software and hardware.
Interrupts are detected by hardware and handled by the OS software.
Interrupts can occur simultaneously and may have different priority.