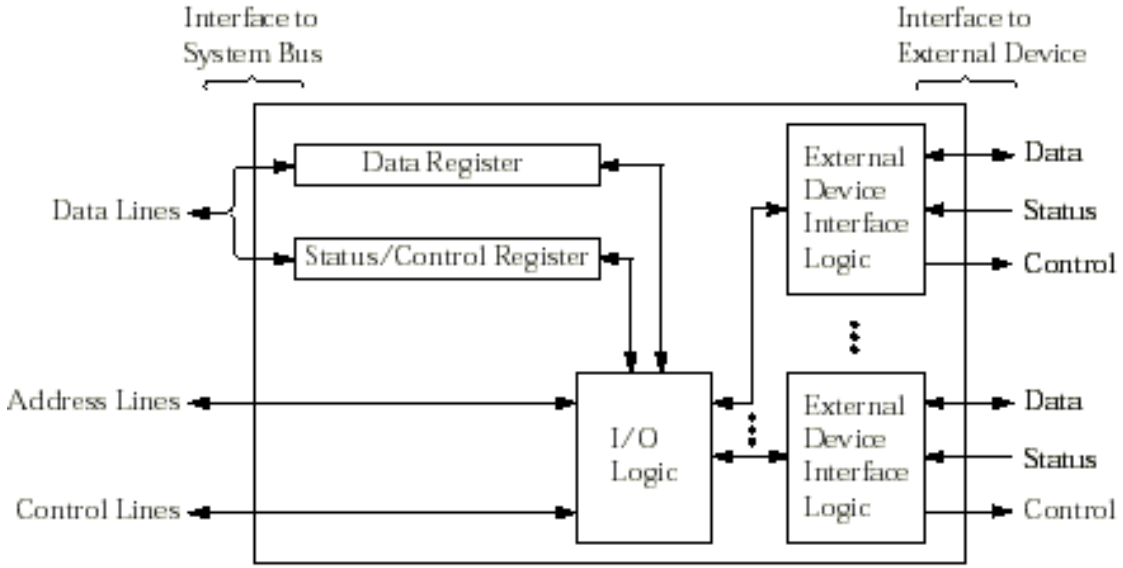
Overview

Computer System/Operating System Overview

Basic Components of a Computer System

- **Processor (CPU)**
- **Main Memory (aka real memory, aka primary memory)**
  - holds data and code
- **I/O modules (I/O controllers, I/O processors...)**
  - hardware (with registers called I/O ports) that moves data between cpu and peripherals like:
    - secondary memory devices (eg: hard disks)
    - keyboard, display...
    - communications equipment
- **System interconnection (ie: Buses)**
  - communication among processors, memory, and I/O modules

I/O Module Structure



- **Data to/from system bus are buffered in data register(s)**
- **Status/Control register(s) holds**
  - current status information of the I/O operation
  - current control information from CPU
- **I/O logic interact with CPU via control bus**
- **Contains logic specific to the interface of each device**

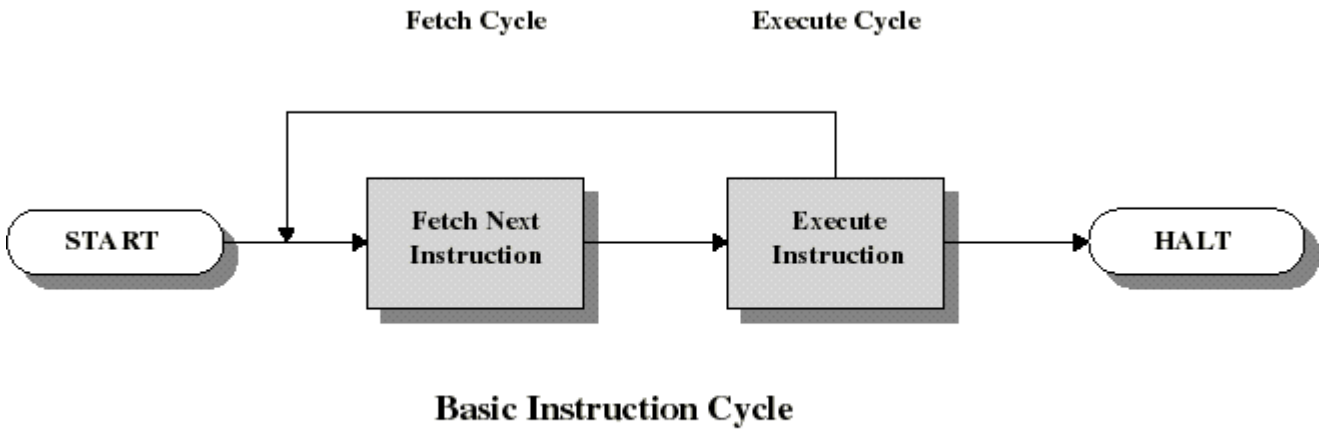
CPU Registers (fast memory on cpu)

- **Control & Status Registers**
  - Generally not available to user programs
  - some used by CPU to control its operation
  - some used by OS to control program execution
- **User-visible Registers**
  - available to system (OS) and user programs
  - holds data, addresses, and some condition codes

Examples of Control & Status Registers

- **Program Counter (PC)**
  - Contains the address of the next instruction to be fetched
- **Instruction Register (IR)**
  - Contains the instruction most recently fetched
- **Program Status Word (PSW)**
  - A register or group of registers containing:
    - condition codes and status info bits
    - Interrupt enable/disable bit
    - Supervisor(OS)/user mode bit

The Basic Instruction Cycle

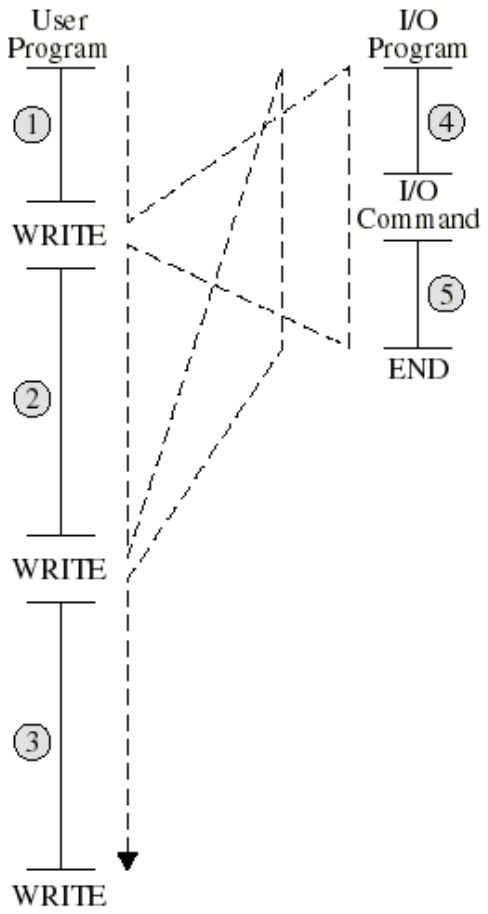


Basic Instruction Cycle

- The CPU fetches the next instruction (with operands) from memory.
- Then the CPU executes the instruction
- Program counter (PC) holds address of the instruction to be fetched next
- Program counter is automatically incremented after each fetch

Then CPU must wait for I/O to complete!

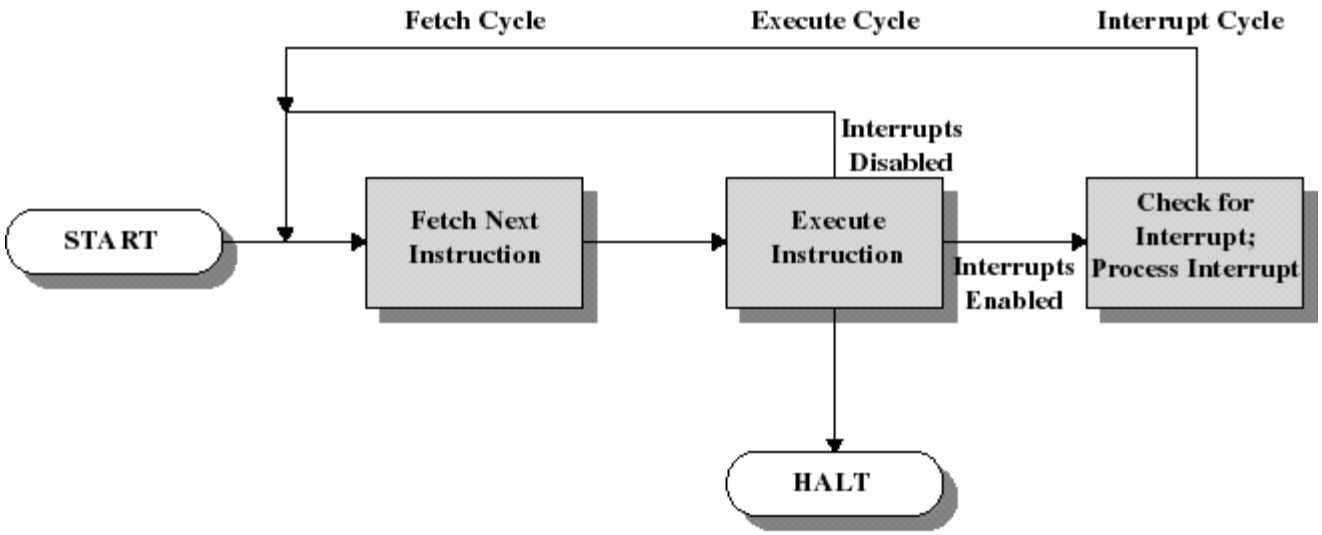
- WRITE transfer control to the printer driver (I/O pgm)
- I/O pgm prepare I/O module for printing (4)
- CPU has to WAIT for I/O command to complete
- Long wait for a printer
- I/O pgm finishes in (5) and report status of operation



Interrupts

- Computers now permit I/O modules to INTERRUPT the CPU.
- For this the I/O module just asserts an interrupt request line on the control bus
- Then CPU transfers control to an Interrupt Handler Routine (normally part of the OS)

Instruction Cycle with Interrupts!



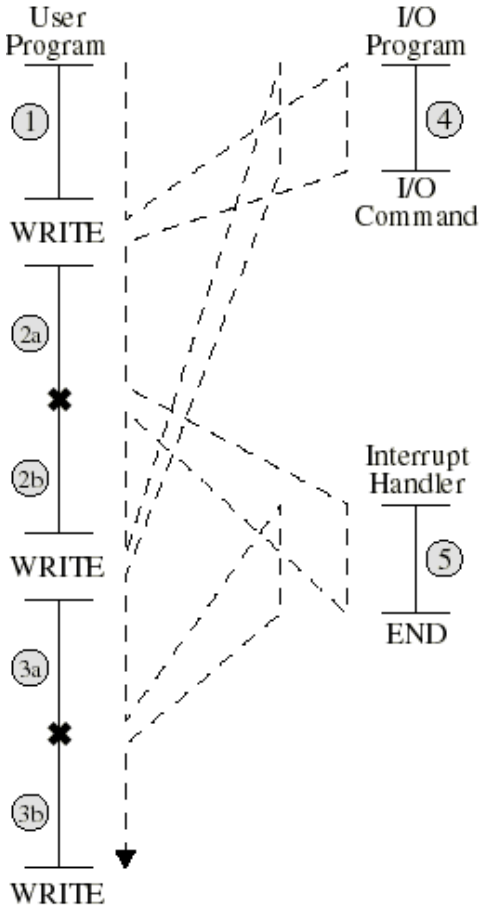
- CPU checks for interrupts after each instruction
- If no interrupts, then fetch the next instruction from the current program
- If an interrupt is pending, then suspend execution of the current program, and execute the interrupt handler

Interrupt Handler

- Is a program that determines nature of the interrupt and performs whatever actions are needed
- Control is transferred to this program
- Control must be transferred back to the interrupted program so that it can be resumed from the point of interruption
- This point of interruption can occur anywhere in the program

Interrupts improve CPU usage

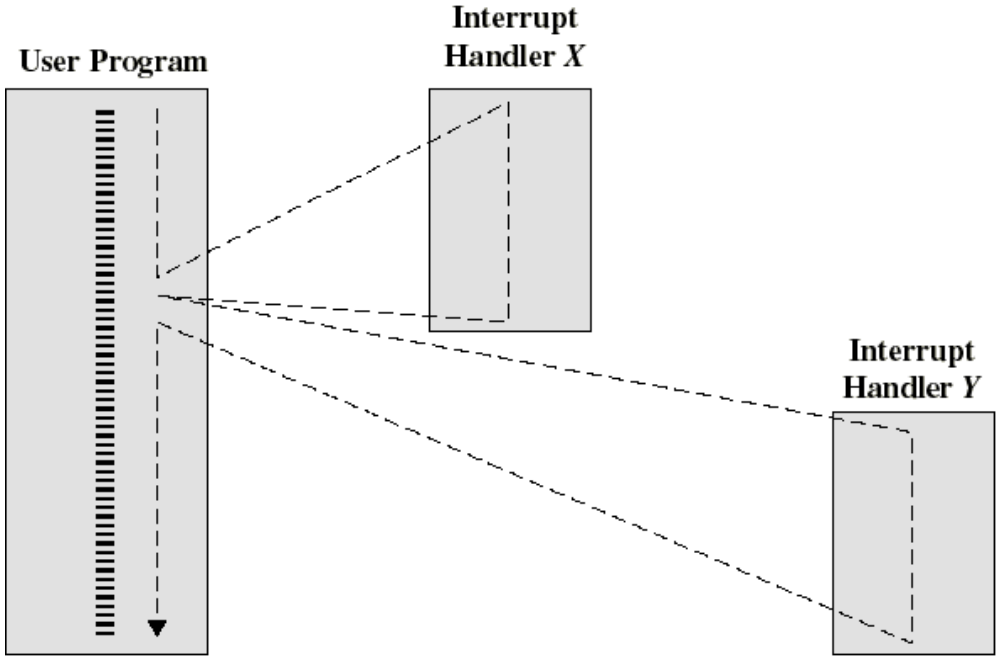
- I/O pgm prepares the I/O module and issues the I/O command (eg: to printer)
- I/O pgm branches to user pgm
- User code gets executed during I/O operation (eg: printing): no waiting
- User pgm gets interrupted (x) when I/O operation is done and branches to interrupt handler to examine status of I/O module
- Execution of user code resumes



Classes of Interrupts

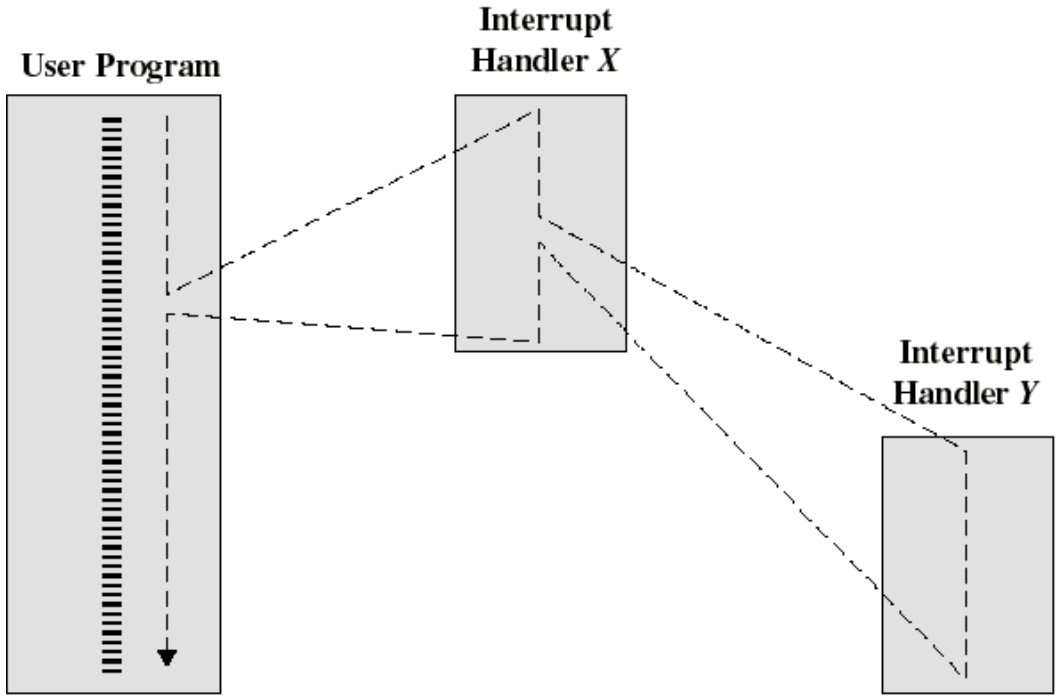
- **I/O**
  - signals normal completion of operation or error
- **Program Exception**
  - Divide overflows
  - try to execute illegal instruction
  - reference outside user's memory space
- **Timer**
  - preempts a pgm from performing another task
- **Hardware failure (eg: memory parity error)**

Multiple interrupts: sequential order



- Disable interrupts during an interrupt
- Interrupts remain pending until the processor enables interrupts
- After interrupt handler routine completes, the processor checks for additional interrupts

Multiple Interrupts: priorities



- Higher priority interrupts cause lower-priority interrupts to wait
- Causes a lower-priority interrupt handler to be interrupted
- Example: when input arrives from communication line, it needs to be absorbed quickly to make room for more input

Multiprogramming

- When a program reads a value on a I/O device it will need to wait for the I/O operation to complete
- Interrupts are mostly effective when a single CPU is shared among several concurrently active processes.
- The CPU can then switch to execute another program when a program waits for the result of the read operation. (more later)

I/O communication techniques

- **3 techniques are possible for I/O operation**
  - Programmed I/O
    - Does not use interrupts: CPU has to wait for completion of each I/O operation
  - Interrupt-driven I/O
    - CPU can execute code during I/O operation: it gets interrupted when I/O operation is done.
  - Direct Memory Access
    - A block of data is transferred directly from/to memory without going through CPU

Programmed I/O

- I/O module performs the action, on behalf of the processor
- But the I/O module does not interrupt the CPU when I/O is done
- Processor is kept busy checking status of I/O module

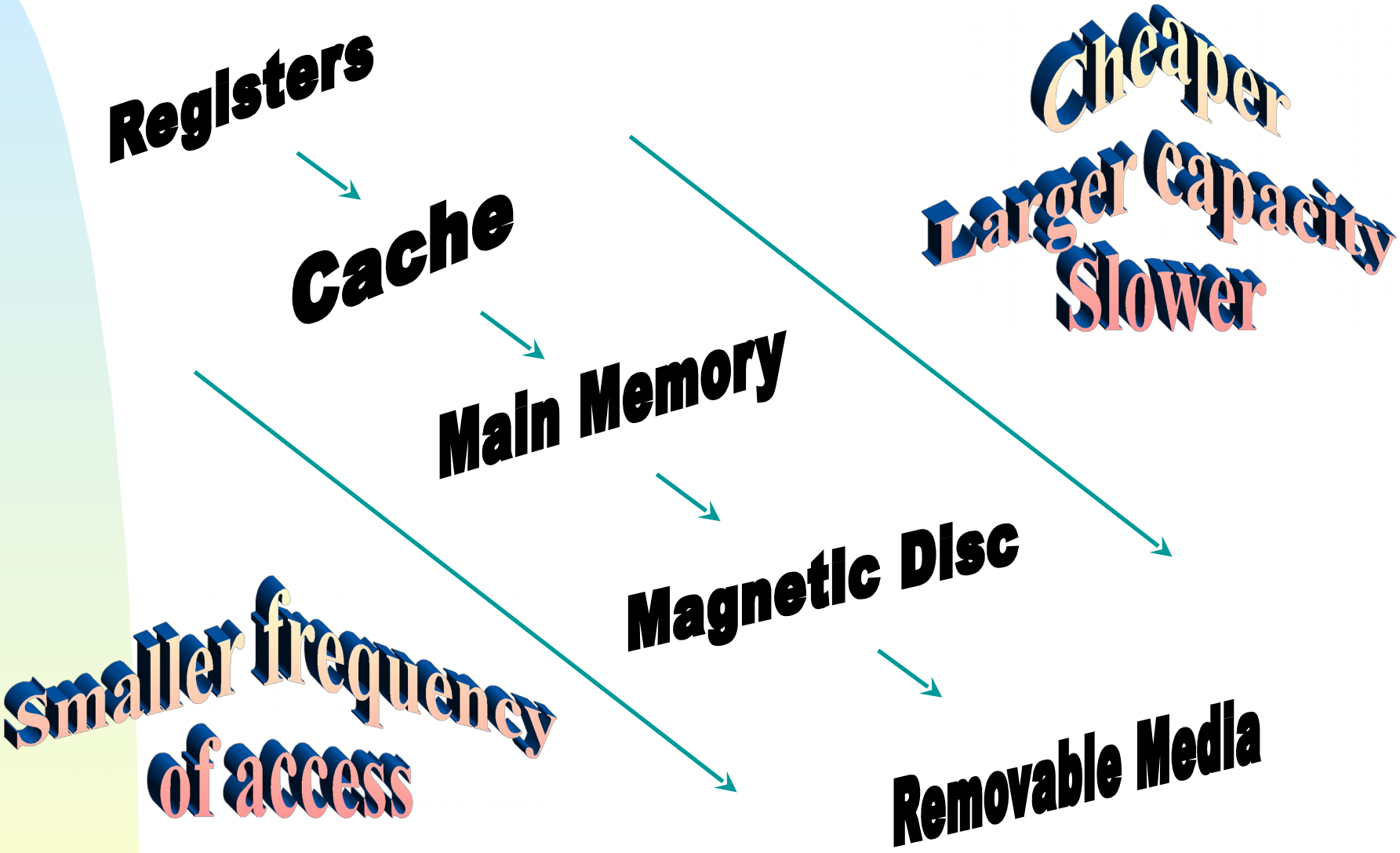
Interrupt-Driven I/O

- Processor is interrupted when I/O module ready to exchange data
- Processor is free to do other work
- No needless waiting
- Consumes a lot of processor time because every word read or written passes through the processor and requires an interrupt

Direct Memory Access

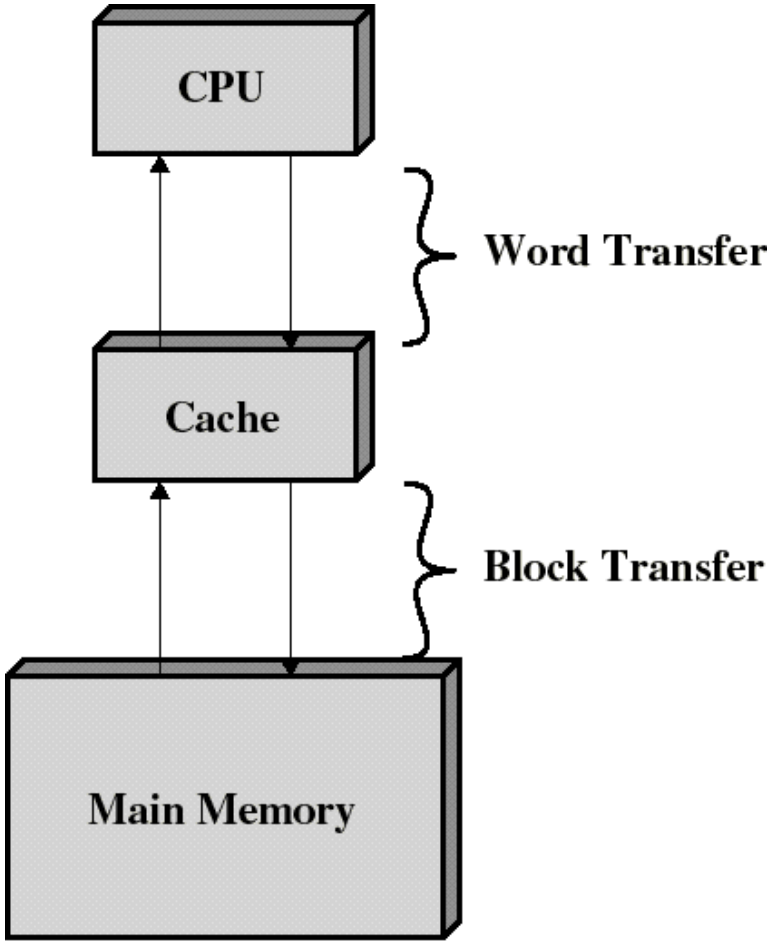
- CPU issues request to a DMA module (separate module or incorporated into I/O module)
- DMA module transfers a block of data directly to or from memory (without going through CPU)
- An interrupt is sent when the task is complete
- The CPU is only involved at the beginning and end of the transfer
- The CPU is free to perform other tasks during data transfer

Memory Hierarchy



Cache Memory

- Small cache of expensive but very fast memory interacting with slower but much larger memory
- Invisible to OS and user programs but interact with other memory management hardware
- Processor first checks if word referenced to is in cache
- If not found in cache, a small block of memory containing the word is moved to the cache

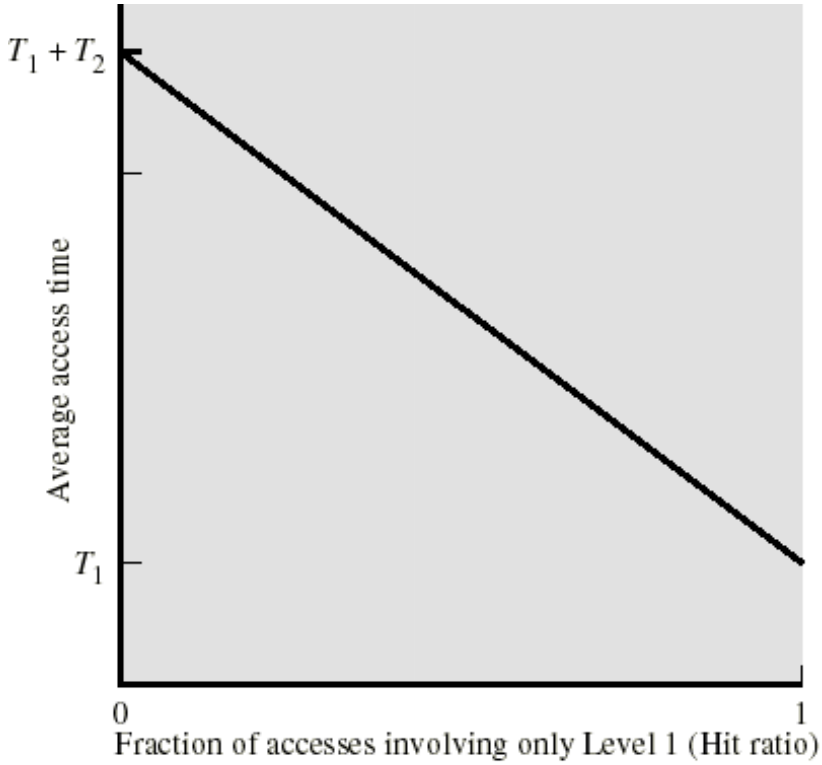


Access time

- **The access time is the time required to bring the referenced word into the processor**
  - T1 = access time for the cache
  - T2 = access time for main memory
  - T = total access time to bring the referenced word into the processor
- **If the referenced word is in the cache:**
  - T = T1
- **If the referenced word is NOT in the cache:**
  - First, it cost T1 to examine the cache (and notice that the referenced word is not there)
  - Then it cost T2 to bring the referenced word from main memory into the processor (simultaneously, that word is placed into the cache)
  - The total access time is thus T = T1 + T2

The Hit Ratio

- Hit ratio = fraction of access where data is in cache
- T2 >> T1
- When hit ratio is close to 1 the average access time is close to T1



Locality of reference

- Memory reference for both instruction and data tends to cluster over a long period of time.
- Example: once a loop is entered, there is frequent access to a small set of instructions.

- Hence: once a word gets referenced, it is likely that nearby words will get referenced often in the near future.
- Thus, the hit ratio will be close to 1 even for a small cache.

Operating System

- **Is a program that controls the execution of application programs**
  - OS must relinquish control to user programs and regain it safely and efficiently
  - Tells the CPU when to execute other pgms
- **Is an interface between the user and hardware**
  - Masks the details of the hardware to application programs
  - Hence OS must deal with hardware details

Kernel?

- The Kernel is the fundamental part of an OS. It is a piece of software responsible for providing secure access of the machine's hardware to various computer programs.
- Since there are many programs, and access to the hardware is limited, the kernel is also responsible for deciding when and how long a program should be able to make use of a piece of hardware.
- Accessing the hardware directly could also be very complex, so kernels usually implement a set of hardware abstractions. These abstractions are a way of hiding the complexity, and providing a clean and uniform interface to the underlying hardware, which makes it easier on application programmers.

Services Provided by the OS

- **Facilities for Program creation (not an integral part of OS)**
  - editors, compilers, linkers, and debuggers
- **Program execution**
  - loading in memory of code and data
- **Access to I/O and files**
  - deals with the specifics of I/O and file formats
- **System access**
  - Protection in access to resources and data
  - Resolves conflicts for resource contention

Services Provided by the OS

- **Error Detection**
  - internal and external hardware errors
    - memory error
    - device failure
  - software errors
    - arithmetic overflow
    - access forbidden memory locations
  - Inability of OS to grant request of application
- **Error Response**
  - simply report error to the application
  - Retry the operation
  - Abort the application

Services Provided by the OS

- **Accounting**
  - collect statistics on resource usage
  - monitor performance (eg: response time)
  - used for system parameter tuning to improve performance
  - useful for anticipating future enhancements
  - used for billing users (on multi-user systems)

Difficulties with OS Design

- **Improper synchronization**
  - Ensures that a program waiting for an I/O device receives the signal
- **Failed mutual exclusion**
  - must permit only one program at a time to perform a transaction on a portion of data
- **Deadlock**
  - It might happen that 2 or more pgms wait endlessly after each other to perform an operation.

An example of deadlock

- Program A wants to copy from disk1 to disk2 and takes control of disk1
- Program B wants to copy from disk2 to disk1 and takes control of disk2

- Program A must wait that program B releases disk2 and program B must wait that program A releases disk1
- Program A and B will wait forever

Major Achievements of OS

- **To meet the difficult requirements of multiprogramming and time sharing, there have been 5 major achievements by OS:**
  - Processes
  - Memory management
  - Information protection and security
  - Scheduling and resource management
  - System structure

Processes

- **Introduced to obtain a systematic way of monitoring and controlling pgm execution**
- **A process is an executable program with:**
  - associated data (variables, buffers…)
  - execution context: ie. all the information that
    - the CPU needs to execute the process
      - content of the processor registers
    - the OS needs to manage the process:
      - priority of the process
      - the event (if any) after which the process is waiting
      - other data (that we will introduce later)

Memory Management

- The key contribution is virtual memory
- It allows programs to address memory from a logical point of view without regard to the amount that is physically available
- While a program is running only a portion of the program and data is kept in (real) memory
- Other portions are kept in blocks on disk
  - the user has access to a memory space that is larger than real memory

Virtual Memory

- **All memory references made by a program are to virtual memory which can be either**
  - a linear address space
  - a collection of segments (variable-length blocks)
- **The hardware (mapper) must map virtual memory address to real memory address**
- **If a reference is made to a virtual address not in memory, then**
  - (1) a portion of the content of real memory is swapped out to disk
  - (2) the desired block of data is swapped in

File System

- **Implements long-term store (often on disk)**
- **Information stored in named objects called files**
  - a convenient unit of access and protection for OS
- **Files (and portions) may be copied into virtual memory for manipulation by programs**

Security and Protection

- **Access control to resources**
  - forbid intruders (unauthorized users) to enter the system
  - forbid user processes to access resources which they are not authorized to

Scheduling and Resource Management

- **Differential responsiveness**
  - discriminate between different classes of jobs
- **Fairness**
  - give equal and fair access to all processes of the same class
- **Efficiency**
  - maximize throughput, minimize response time, and accommodate as many users as possible

Key Elements for Scheduling

- **OS maintains queues of processes waiting for some resource**
  - Short term queue of processes in memory ready to execute
    - The dispatcher (short term scheduler) decides who goes next
  - Long term queue of new jobs waiting to use the system
    - OS must not admit too many processes
  - A queue for each I/O device consisting of processes that want to use that I/O device