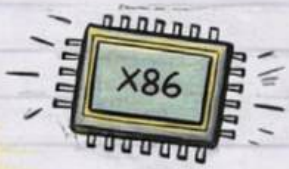
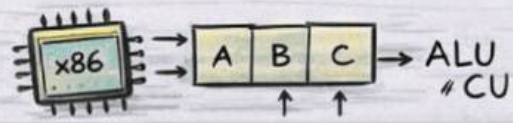


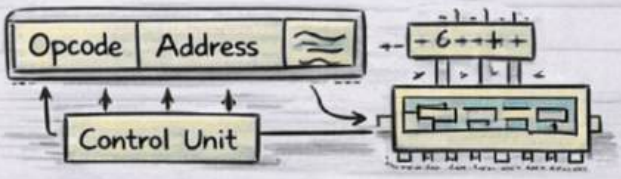
COA
Module-2 Notes

by pyqfort.com

Contents Covered:



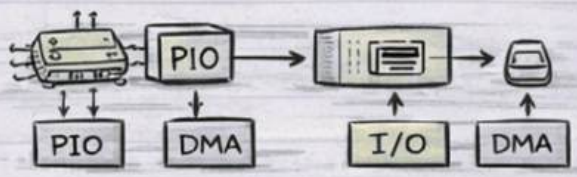
- x86 Architecture



- CPU Control Unit Design
- Microinstruction

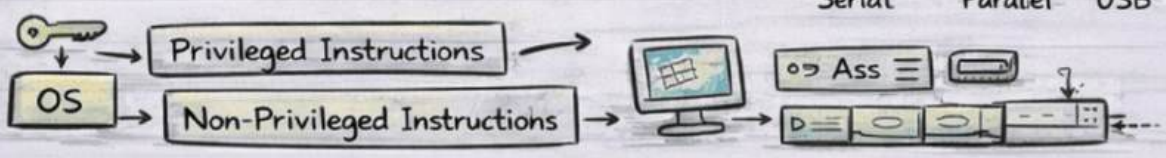
- Bus Transfer & Arbitration
- Main Memory Overview

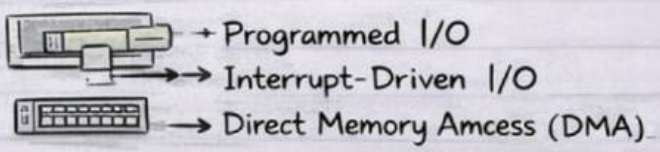
- Peripheral Devices
- Interface Units



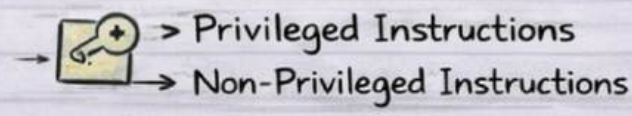
- I/O Transfers
- I/O Device Interfaces

- Privileged & Non-Privileged Instructions





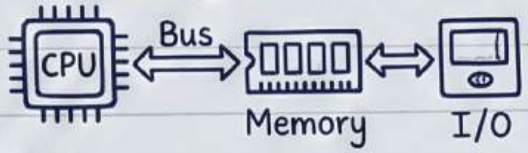
- I/O Transfers
- I/O Device Interfaces



Module 2 : x86 Architecture Intro (8086 BIU)

Bus Interface Unit (BIU)

- 'BIU' handles communication with 'memory' & 'I/O devices' via the 'system bus'.
- Main Purpose :
  1. 'Fetches instructions' from memory.
  2. 'Reads/Writes data' to/from Memory & I/O.
  3. Provides 'address relocation'.

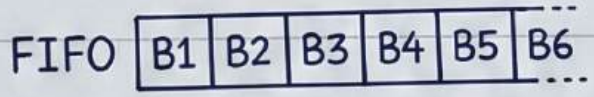


BIU Components

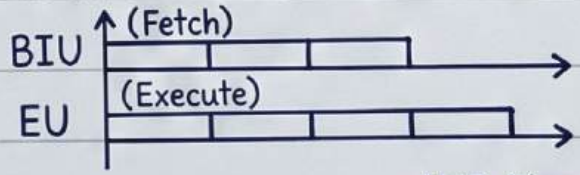
- Contains three main parts : 'Segment Registers', 'Instruction Queue', 'Instruction Pointer'.

Instruction Queue

- 'Prefetches six instruction bytes' in advance.
- Stored in 'high speed registers' (the queue) in 'FIFO order'.

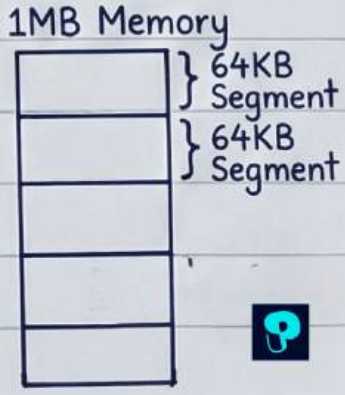


- BIU & 'Execution Unit (EU)' work in 'parallel'.
- Fetching next instruction while EU executes current is called 'pipelining'.



Segment Registers

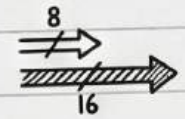
- 8086 can address '1MB (2^{20}) memory'.
- Divided into '16 local segments' of '64KB' each.



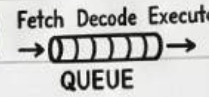
x86 Architecture (Continued)

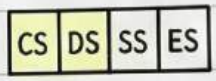
8085 vs. 8086 COMPARISON:

Feature	8085	8086
Data Bus	8-bit	16-bit (processes more data)
Address Bus	16-bit (64KB)	20-bit (1MB)
Pipelining	No	Yes (instruction prefetch queue)
Segmentation	No	Yes (access larger memory)
Speed/Complexity	Slower, simpler	Faster, more complex



64KB 1MB





FLAG REGISTER (PSW):

- "Indicator lights" for CPU status/result.
- Crucial for program decisions (e.g., jumps).

Specific Flags:

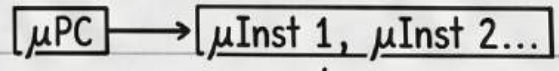
- Zero Flag (ZF): Result is zero. (Light ON if yes). 0 = ON
- Carry Flag (CF): Carry-out from MSB (unsigned overflow). e.g., 250+10. 1 [00...]
- Sign Flag (SF): Result is negative (MSB is 1). (-) = ON
- Overflow Flag (OF): Signed arithmetic result too large/small. (+)·(+)=(-)? → OF!
- Parity Flag (PF): Result has even number of 1s. 1011 → PF=0, 1100 → PF=1
- Auxiliary Carry Flag (AF): For BCD arithmetic.

BUS ARBITRATION:

- Deciding which device gets the system bus.
- Like a "traffic controller" to avoid collisions.

MICROPROGRAM CONTROL:

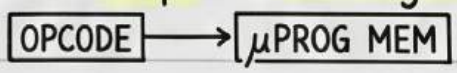
- Microprogram Counter (μPC) usually increments to fetch next microinstruction.



- Not incremented if:
  - Branch: μPC loaded with new address (jump).
  - Subroutine Call: μPC loaded with start of micro-subroutine.

SUBROUTINE

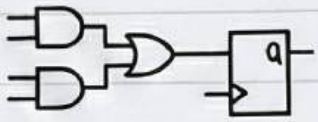
  - Mapping Operation: Opcode maps to starting address in microprogram memory.



CPU Control Unit Design: Hardwired + Micro-Programmed Approaches

TWO MAJOR TYPES OF CONTROL ORGANIZATION:
- Hardwired Control
- Microprogrammed Control

1. HARDWIRED CONTROL:

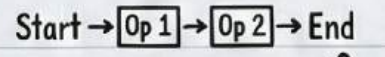


- Implemented with gates, flip-flops, decoders, and other digital circuits
- Advantage: Optimized for fast mode of operation

- Disadvantage: Requires wiring changes if design is modified

2. MICROPROGRAMMED CONTROL:

- Control information is stored in a control memory
- Initiates required sequence of microoperations



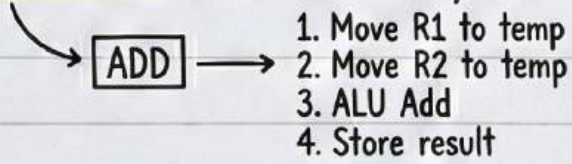
- Modifications done by updating microprogram in control memory

- Concept: Like having a "tiny, very simple computer" inside the main computer

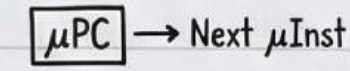
- It has its own microprogram made of microinstructions

HOW MICROPROGRAMMING WORKS (Example):

- Main computer gets machine instruction (e.g., ADD R1, R2)
- Control unit looks up sequence of tiny microinstructions to carry out operation step-by-step



MICROPROGRAM COUNTER (μPC):



- Usually increments to fetch next microinstruction
- Not incremented for:
  - Branch: Loaded with new address
  - Subroutine Call: Loaded with start of micro-subroutine
  - Mapping Operation: Opcode maps to starting address

Microinstruction Structure & Address Sequencing

1. MICROINSTRUCTION STRUCTURE:

- A microinstruction is a single, very basic control command within the microprogram.

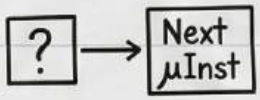
- Specifies elementary operations for one clock cycle.

- Bits directly control CPU parts:
  - Which registers to use.

  - What operation ALU should perform (add, subtract, etc.).

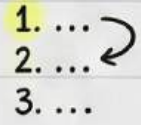
  - Where data moves (e.g., register to ALU, memory to register).

  - How to determine the next microinstruction (address sequencing).



2. ADDRESS SEQUENCING:

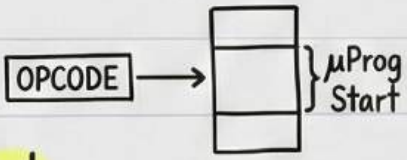
- Deciding which microinstruction to execute next.
- Common ways:
  - Incrementing: Go to the next in sequence (like a recipe).



  - Branching (Conditional/Unconditional): Jump to a specific address.

    ↳ Conditional uses CPU status flags (e.g., Zero flag).

  - Mapping from Opcode: Opcode maps to starting address of micro-routine.

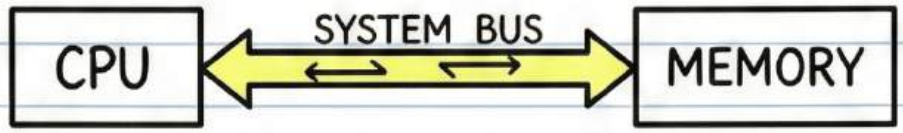


  - Subroutines: Common micro-routines called and returned from.

Bus Transfer & Bus Arbitration

1. BUS TRANSFER:

- Process of **sending data** from one component (e.g., memory or CPU) to another via the **system bus.**

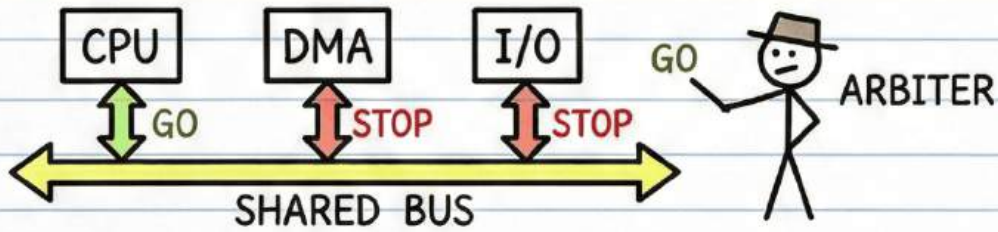


- Involves three types of buses:
  - **Data bus:** Carries data.
  - **Address bus:** Carries location.
  - **Control bus:** Manages signals for the transfer.

- Enables **communication** between hardware units, allowing instructions & data to be **moved efficiently.**

2. BUS ARBITRATION:

- Process of **determining which device** among multiple contenders **gets control** of a **shared communication bus.**



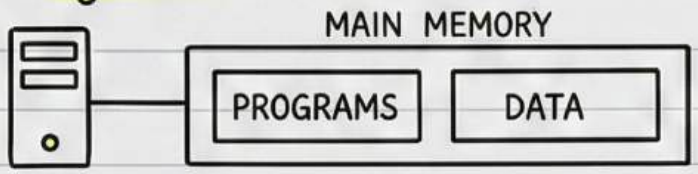
- Needed when **multiple components** (CPU, DMA, I/O) **request access simultaneously.**

- Ensures **only one device** uses the bus at a time to **prevent data collision** and ensure **orderly communication.**

Main Memory Overview & SRAM (Static RAM)

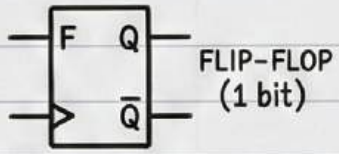
1. MAIN MEMORY OVERVIEW:

- The central storage unit in a computer system.
- It is a relatively large and fast memory used to store programs and data during operation.
- Based on semiconductor integrated circuits.
- Integrated circuit RAM chips are available in two modes: static and dynamic.

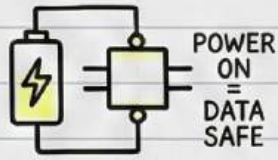


2. SRAM (Static RAM):

- Uses flip-flops (latches) to store each bit.



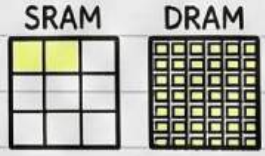
- A flip-flop is a bistable circuit with two stable states (0 or 1) and remains in that state as long as power is supplied.
- Refreshing: Does not need refreshing. Data stays as long as power is maintained. That's why it's "static".



- Speed: Faster than DRAM. Access times are lower (no refresh cycle).

- Cost: More expensive per bit (more transistors per cell, complex).

- Density: Less dense (fewer bits per chip). Each cell takes more space.



- Typical Uses: Used for cache memory (L1, L2, L3 caches), register files, and high-speed applications.

DRAM (Dynamic RAM)

1. Storage Mechanism & Basics:

- Uses a **transistor** and a **capacitor** to store each bit.

1 bit = Charge on Capacitor

- It is **slower** and **cheaper** than SRAM.
- **More dense** (stores more bits per chip).

2. The Need for Refreshing:

- Needs to be **refreshed periodically** because the **capacitor leaks charge**.

- The charge on the capacitors **leaks away over time** (even with power applied).
- To **prevent data loss**, the memory controller must **read the data and rewrite it** (**recharge the capacitor**) every few milliseconds.

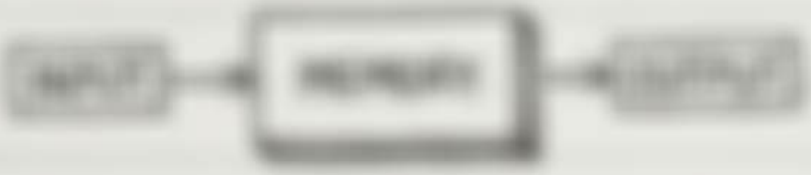
3. Characteristics & Use:

- Speed: Generally **slower than SRAM** due to refreshing and access method.
- Cost: **Less expensive** to produce per bit (simpler cell).
- Density: **Higher memory density** than SRAM.
- Typical Uses: Used for **main system memory** (the **RAM sticks**) in computers.

Peripheral Devices

- Devices under direct control of the computer are connected on-line.

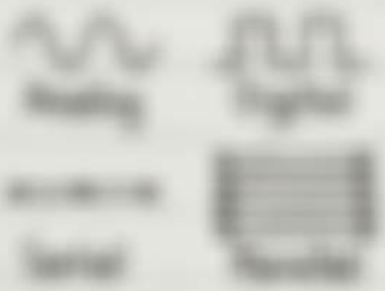
- They are designed to read information into or out of the memory unit upon command from the CPU.



- Input or output devices attached to the computer are called peripherals.

- There are three types: input, output, and input-output peripherals.

- They may be analog or digital and serial or parallel.



- Common examples: keyboards, display units, and printers.

Interface Units

1. What are they?

- Special hardware components located between the CPU and peripherals.

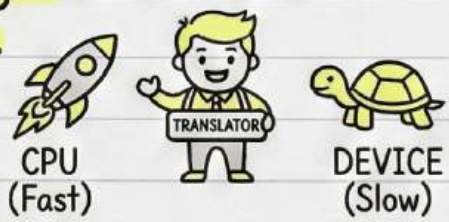
- Their purpose is to supervise and synchronize all input and output transfers.

2. The "Interface" Concept:

- General term for the point of contact between two parts of a system.
- In computers, it's a set of signal connection points for data exchange.

3. Why Use an I/O Interface? (Translator & Manager)

- Think of it as a "translator and manager" between the fast CPU and slower, diverse peripherals.



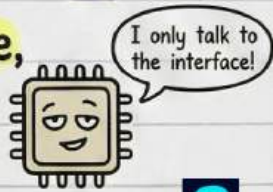
4. Resolving Differences:

- Helps manage differences to simplify the CPU's job.
  - Speed Differences: Manages the speed mismatch between fast CPUs and slow devices.

  - Different Formats: Converts device-specific data formats & control signals into a standard format the CPU understands (and vice-versa).

  - Electrical Differences: Handles different voltage levels used by various devices.

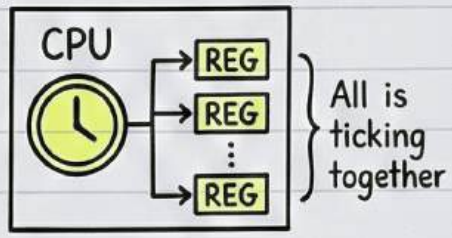
  - Simplifies CPU's Job: CPU just talks to the interface, it doesn't need to know specific device details.



Asynchronous Communication

1. The Context: Synchronous Operations (Within a Single Unit)

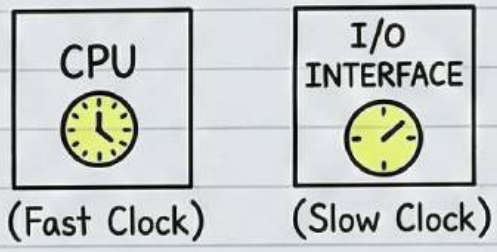
- In a digital system, internal operations within a single unit (like a CPU) are often synchronized by a common clock.



- All registers change state simultaneously with clock pulses. This is a synchronous operation.

2. The Reality: Multiple Independent Units

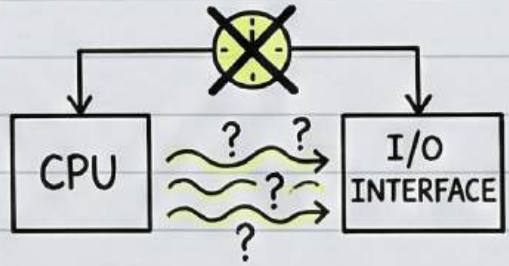
- Computer systems are typically built from multiple units designed independently, such as a CPU and an I/O interface.



- Each unit has its own private clock for its internal registers.

3. What is Asynchronous Communication?

- When these two independent units need to communicate and transfer data, they are said to be asynchronous to each other.



- Their operations are not tied to a shared clock.

- This approach is widely used in most computer systems.

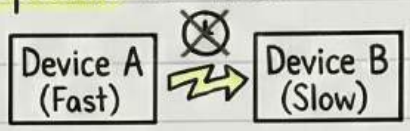
4. Why it's needed (Role of the Interface)

- The I/O interface acts as a translator and manager to manage differences between the fast CPU and slower devices.

- It handles speed differences, format differences, and electrical differences, simplifying the CPU's job.

Asynchronous Data Transfer

Asynchronous data transfer is a method of sending data between two devices (e.g., CPU & peripheral) that don't share a common clock signal to time the transfer. It's useful when communicating devices operate at different speeds or their timing isn't predictable.



Two Primary Methods:

- Strobe Control: Uses a single control line (the strobe) to time each data transfer. A strobe pulse indicates when the transfer has to occur.

- Handshaking: Uses two or more control signals for a more robust, acknowledged data transfer. It is more reliable than strobe control.

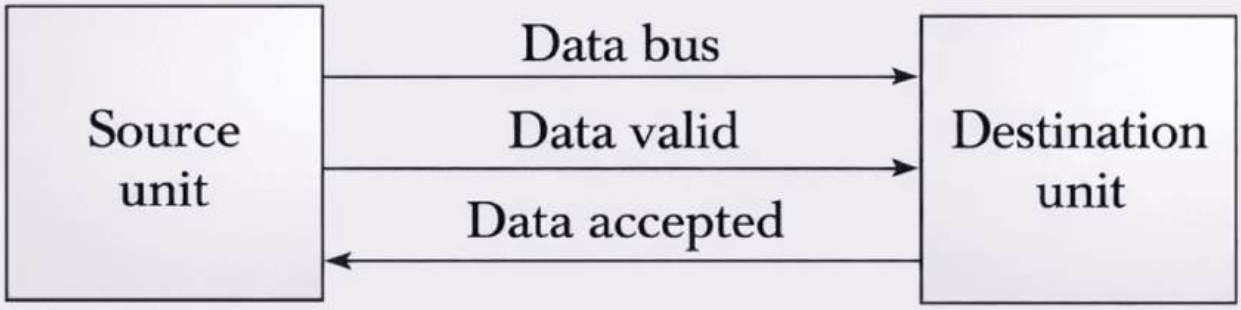
Handshaking Sequences:

1. Source-initiated (Fig 11-5):

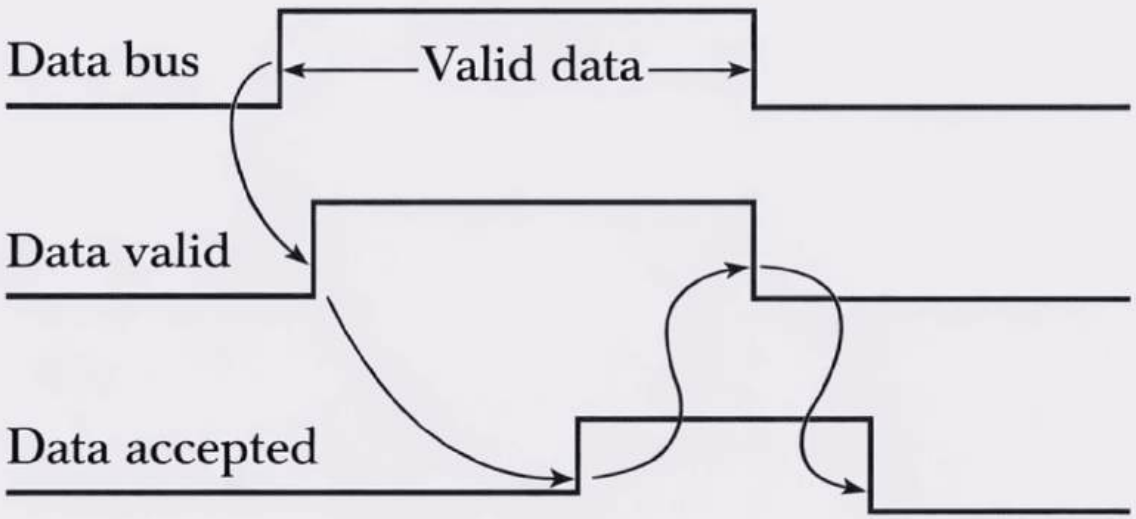
① Source places data, asserts Data valid → ② Destination accepts data, asserts Data accepted → ③ Source negates Data valid → ④ Destination negates Data accepted (Ready for next)

2. Destination-initiated (Fig 11-6):

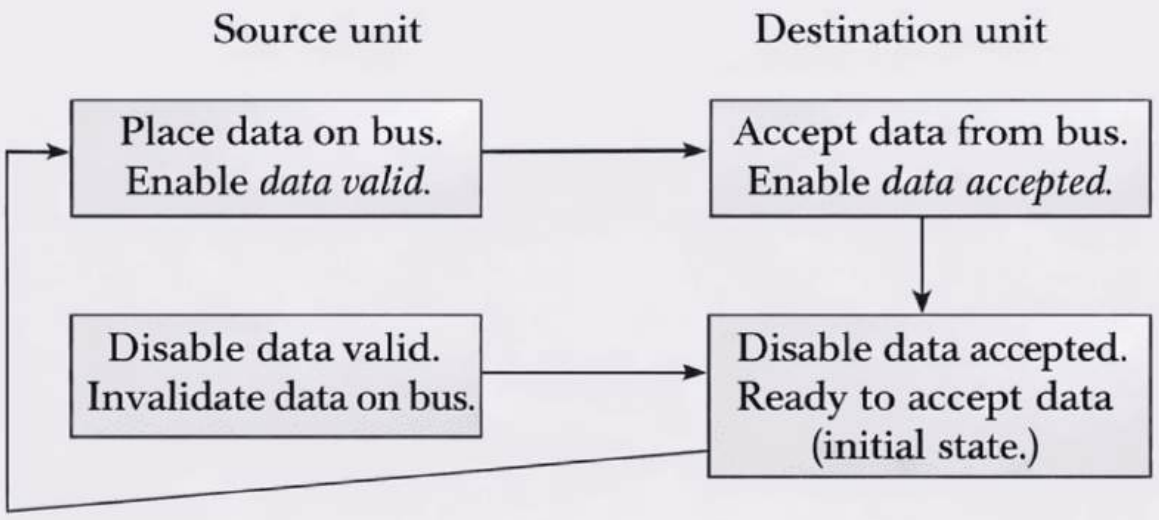
① Destination ready, asserts Ready for data → ② Source places data, asserts Data valid → ③ Destination accepts, negates Ready for data → ④ Source negates Data valid



(a) Block diagram



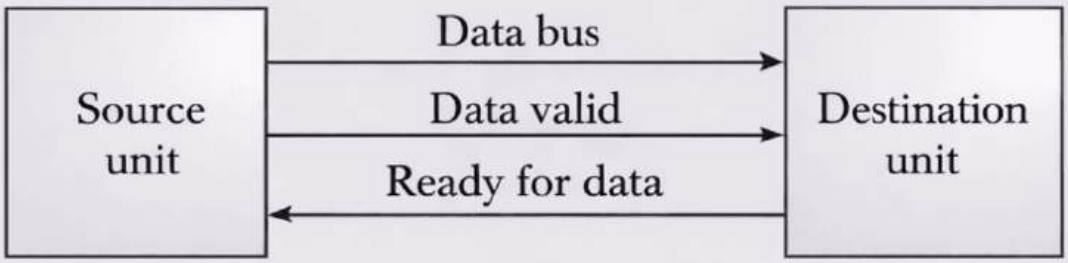
(b) Timing diagram



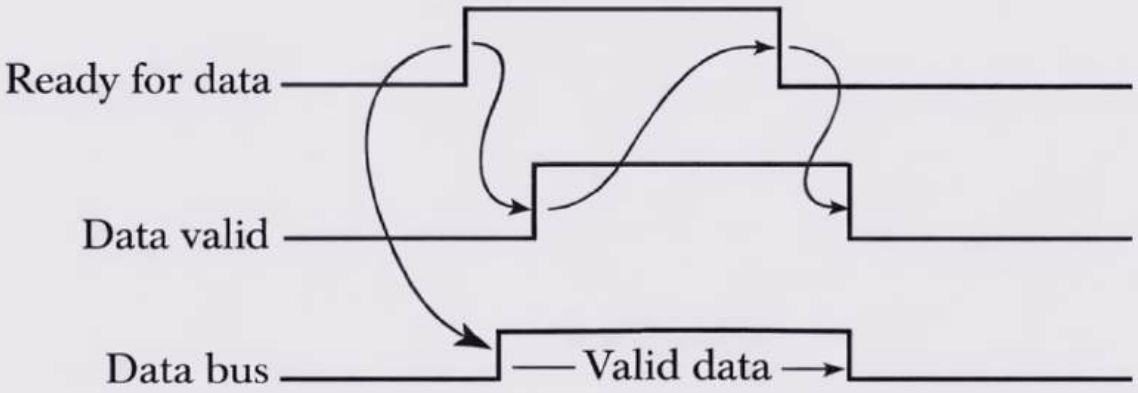
(c) Sequence of events

Figure 11-5 Source-initiated transfer using handshaking.

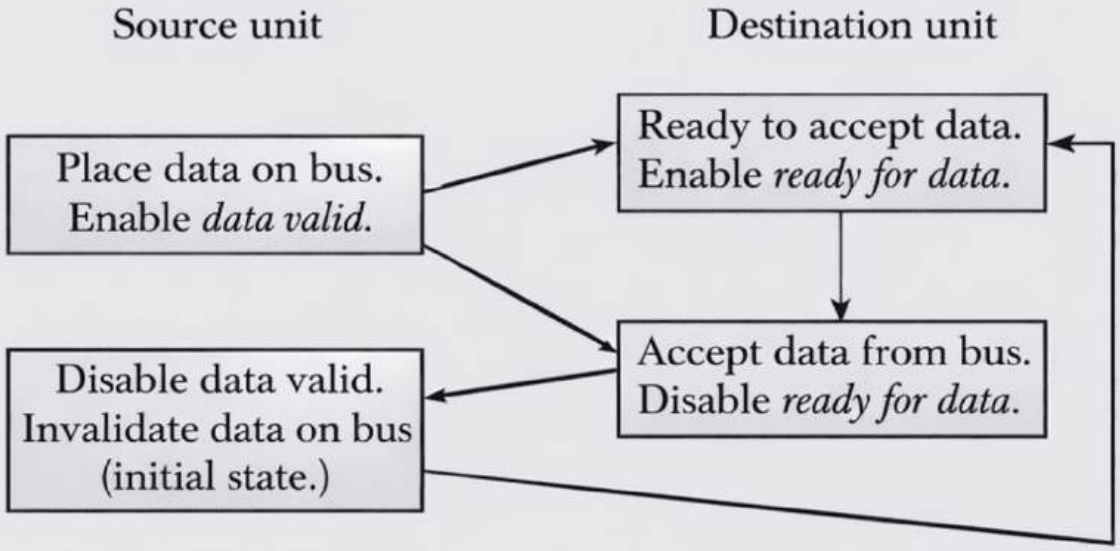
Figure 11-6 Destination-initiated transfer using handshaking.



(a) Block diagram



(b) Timing diagram

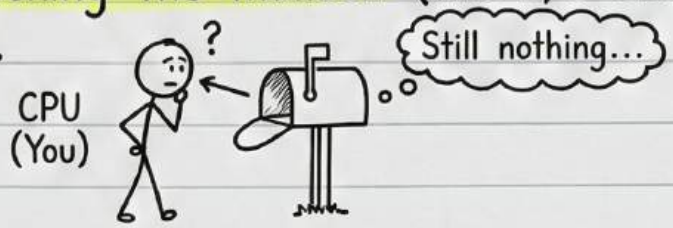


(c) Sequence of events

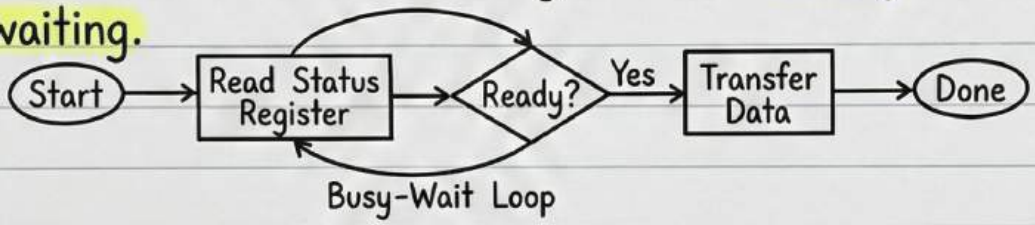
Figure 11-6 Destination-initiated transfer using handshaking.

I/O Transfers: Program Controlled I/O

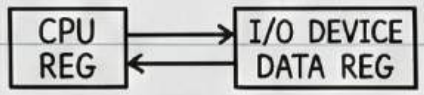
1. The Concept (Analogy):
   - Imagine you're waiting for a letter. You (the CPU) keep constantly checking the mailbox (the I/O device) to see if it has arrived.



2. Mechanism (Polling & Busy-Waiting):
   - The CPU is fully in charge of the data transfer.
   - It executes a program that repeatedly checks the device status (reads a status register) to see if it's ready.
   - This process of constant checking is called Polling or busy-waiting.



3. Data Transfer & Path:
   - Once ready, the CPU reads/writes data from/to the device's data register.
   - Data transfer is between a CPU register and the peripheral.
   - Separate instructions are needed to move data between the CPU and main memory.



4. Characteristics (Pros & Cons):
   - CPU Involvement: Requires constant monitoring, keeping the CPU continuously busy.
   - Simplicity: Relatively simple to implement (hardware & software).
   - Efficiency: Very inefficient if the device is slow. The CPU spends a lot of time in a "busy-wait" loop, unable to perform other tasks.

I/O Transfers: Interrupt Driven I/O

1. The Concept (Analogy):

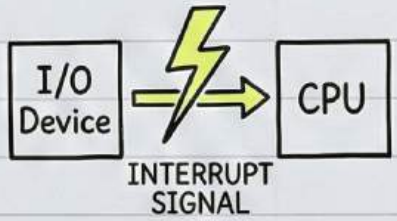
- Imagine the postman (I/O device) rings your doorbell (sends an interrupt (sends an interrupt) only when the letter arrives.

2. Purpose & Efficiency:

- Avoids the time-consuming busy-waiting of programmed I/O.
- The CPU (you) can do other tasks and only pays attention when the doorbell rings. Much more efficient.

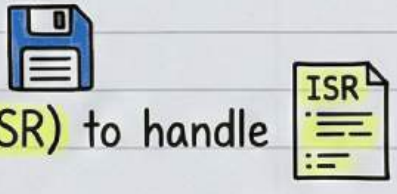
3. Mechanism:

- CPU issues special commands to the interface to start an operation.
- Instead of polling, the I/O device notifies the CPU when ready by sending an interrupt signal.

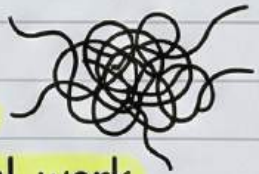


4. CPU Action (How it handles it):

- While waiting, CPU proceeds to execute other programs.
- When interrupted:
  - Suspends current task.
  - Saves its current state (PC, registers).
  - Executes an Interrupt Service Routine (ISR) to handle the transfer.
  - Restores saved state and resumes suspended task.



5. Pros & Cons:

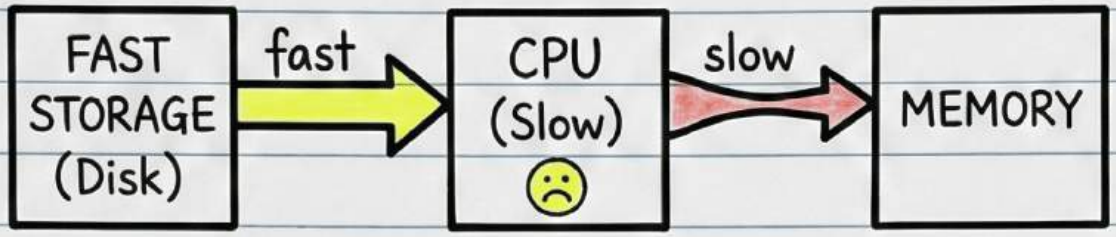


- CPU Involvement: Only when device needs attention.
- Efficiency: Much more efficient, CPU performs useful work while waiting.
- Complexity: More complex to implement (hardware & software).

I/O Transfers: DMA (Direct Memory Access)

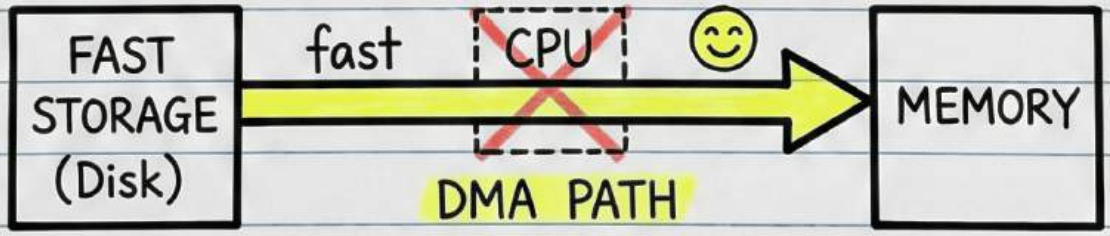
1. The Problem (CPU Bottleneck):

- Data transfer between fast storage devices (e.g., magnetic disk) and memory is often limited by the speed of the CPU.



2. The Solution (Bypass CPU):

- To improve speed, we remove the CPU from the path.
- We let the peripheral (I/O) device manage the memory buses directly.

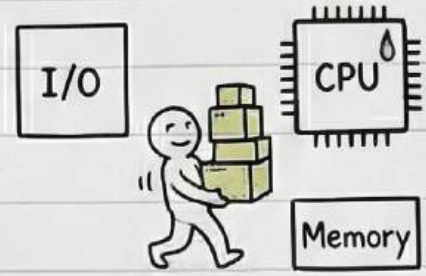


3. Definition & Benefit:

- This transfer technique is called direct memory access (DMA).
- It significantly improves the speed of transfer for large data blocks.

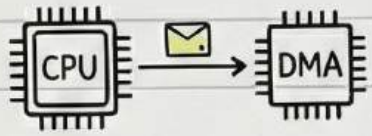
Working of DMA (Direct Memory Access)

DMA is like hiring a dedicated helper to move large amounts of data between memory and an I/O device without bothering makest bothering the busy CPU for every single piece. It allows hardware to access RAM directly, bypassing the CPU.

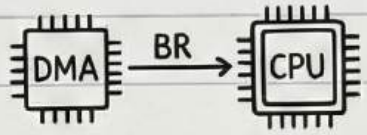


The DMA Transfer Process:

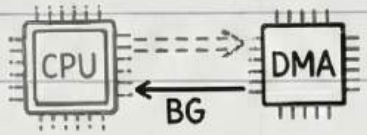
1. **CPU Initiates:** The CPU initializes the DMA by sending the starting address, word count, and transfer mode (read/write).



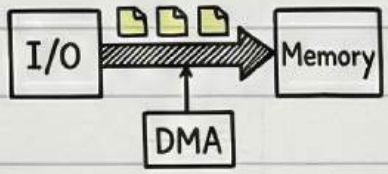
2. **DMA Requests:** The DMA controller sends a Bus Request (BR) to the CPU to ask for control of the system buses.



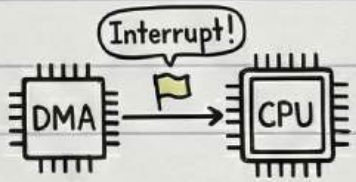
3. **CPU Grants:** The CPU places its buses in a high-impedance state (disconnects) and sends a Bus Grant (BG) signal.



4. **DMA Transfers:** The DMA controller takes over the buses and directly transfers data between memory and I/O In a burst transfer, a block is moved continuously. cite 29



5. **Completion:** Once done, the DMA sends an interrupt signal to the CPU to notify it. The CPU then retakes control of the buses.



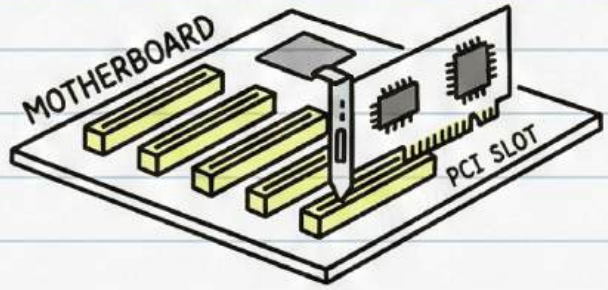
Benefits:

- Increased CPU Efficiency: CPU is free for other tasks.
- Faster Data Transfer: Bypasses the CPU.

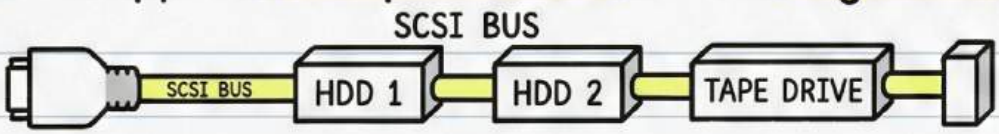
- Improved System Performance: Overall throughput is better.

I/O Device Interfaces: PCI, SCSI, & USB

1. **PCI** (Peripheral Component Interconnect):
   - An **internal bus** on a computer's **motherboard** used to connect hardware devices like **sound cards, network cards, and graphics cards** directly to the system.
   - It's **relatively fast** and allows these components to **communicate with the CPU and memory.**



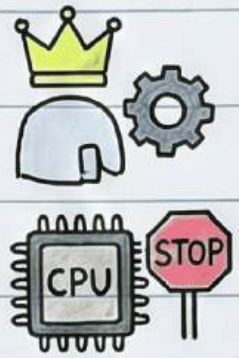
2. **SCSI** (Small Computer System Interface):
   - A **high-performance interface** primarily used for **connecting storage devices** like **hard drives and tape drives**, especially in **servers** and **high-end workstations.**
   - It can support **multiple devices** on a **single bus.**



3. **USB** (Universal Serial Bus):
   - A **very common external interface** used to connect a **wide variety of peripheral devices** (keyboards, mice, printers, flash drives, cameras, phones) to a computer.
   - It's known for its **"plug-and-play" capability** and can also **provide power** to connected devices.
   - It uses a **serial method (data sent bit by bit)** to communicate.

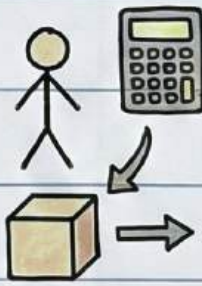
Privileged & Non-Privileged Instructions

Privileged Instructions

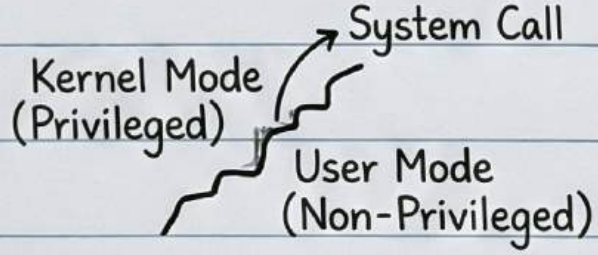


These are powerful instructions that can only be run by the operating system (OS) (in (kernel/supervisor mode). They manage critical system resources (e.g., I/O, memory management, halting the CPU).

Non-Privileged Instructions



These are regular instructions that user programs can run (in user mode). They perform tasks like arithmetic or data movement within the program's own space.



Full content is
available at
'topmate.io/saumy1905'
or you can navigate
to the specific section
of the subject on
'pyqfort.com'
to access it