

*University Of Diyala
College Of Engineering
Computer Engineering Department*



COMPUTER SYSTEM ARCHITECTURE

INPUT-OUTPUT ORGANIZATION

Dr. Yasir Amer Abbas
Second stage
2017

INPUT-OUTPUT ORGANIZATION

- **Peripheral Devices**
- **Input-Output Interface**
- **Asynchronous Data Transfer**
- **Modes of Transfer**
- **Priority Interrupt**
- **Direct Memory Access**
- **Input-Output Processor**
- **Serial Communication**



PERIPHERAL DEVICES

Input Devices

- Keyboard
- Optical input devices
 - Card Reader
 - Paper Tape Reader
 - Bar code reader
 - Digitizer
 - Optical Mark Reader
- Magnetic Input Devices
 - Magnetic Stripe Reader
- Screen Input Devices
 - Touch Screen
 - Light Pen
 - Mouse
- Analog Input Devices

Output Devices

- Card Puncher, Paper Tape Puncher
- CRT
- Printer (Impact, Ink Jet, Laser, Dot Matrix)
- Plotter
- Analog
- Voice

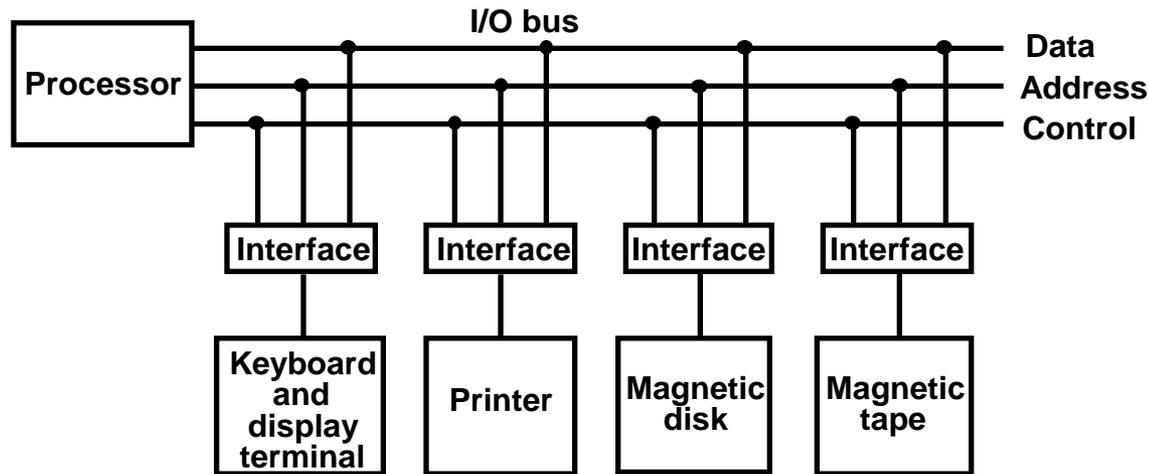


INPUT/OUTPUT INTERFACES

- * **Provides a method for transferring information between internal storage (such as memory and CPU registers) and external I/O devices**
- * **Resolves the *differences* between the computer and peripheral devices**
 - **Peripherals - Electromechanical Devices**
CPU or Memory - Electronic Device
 - **Data Transfer Rate**
 - Peripherals - Usually slower**
 - CPU or Memory - Usually faster than peripherals**
 - Some kinds of Synchronization mechanism may be needed**
 - **Unit of Information**
 - Peripherals - Byte**
 - CPU or Memory - Word**
 - **Operating Modes**
 - Peripherals - Autonomous, Asynchronous**
 - CPU or Memory - Synchronous**



I/O BUS AND INTERFACE MODULES



Each peripheral has an interface module associated with it

Interface

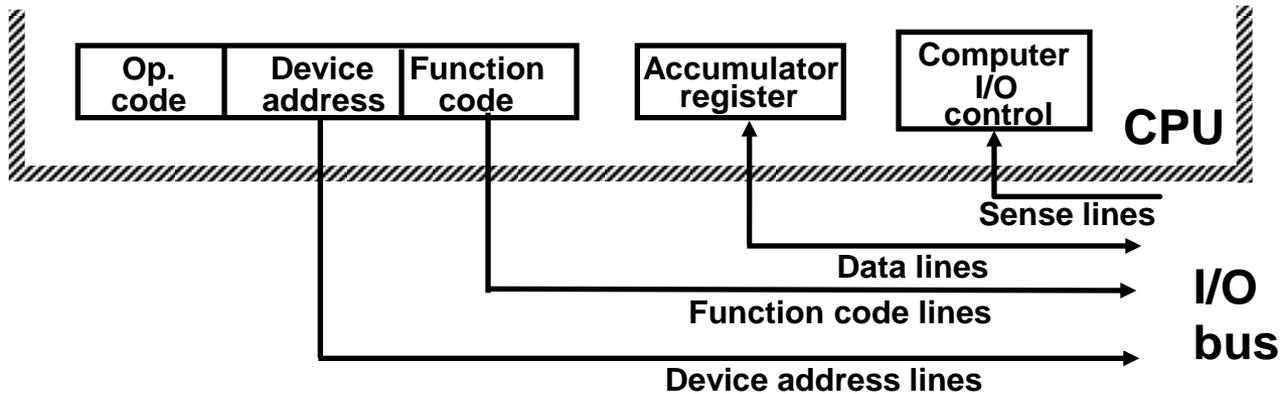
- Decodes the device address (device code)
- Decodes the commands (operation)
- Provides signals for the peripheral controller
- Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory

Typical I/O instruction

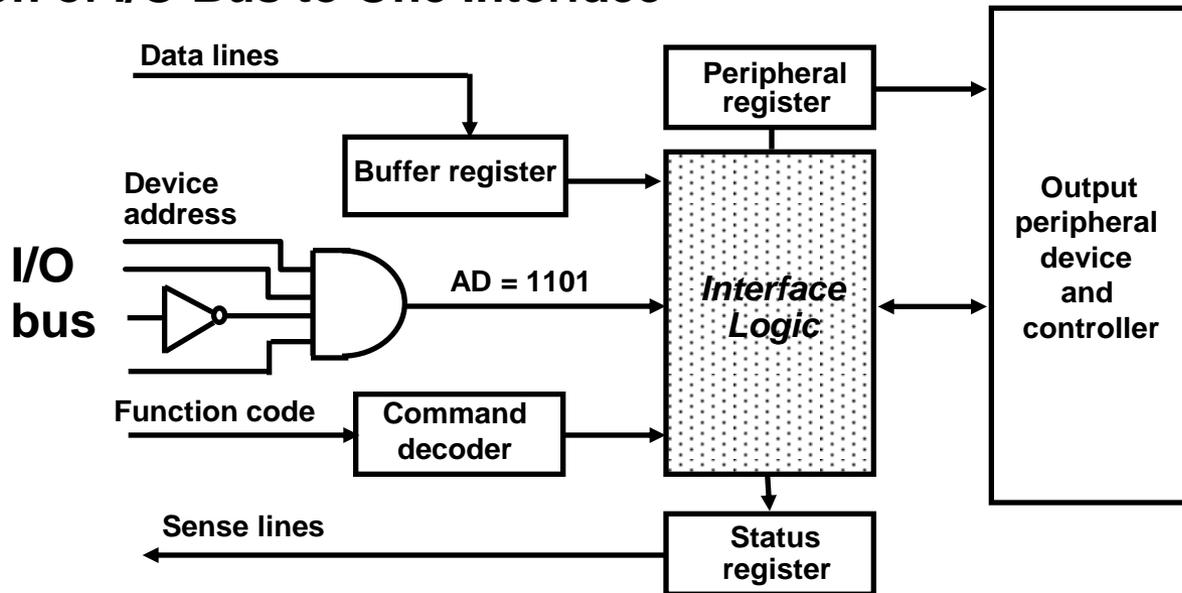


CONNECTION OF I/O BUS

Connection of I/O Bus to CPU



Connection of I/O Bus to One Interface



I/O BUS AND MEMORY BUS

Functions of Buses

- * **MEMORY BUS** is for information transfers between CPU and the MM
- * **I/O BUS** is for information transfers between CPU and I/O devices through their I/O interface

Physical Organizations

- * Many computers use a common single bus system for both memory and I/O interface units
 - Use one common bus but separate control lines for each function
 - Use one common bus with common control lines for both functions
- * Some computer systems use two separate buses, one to communicate with memory and the other with I/O interfaces

I/O Bus

- Communication between CPU and all interface units is via a common I/O Bus
 - An interface connected to a peripheral device may have a number of *data registers*, a *control register*, and a *status register*
 - A command is passed to the peripheral by sending to the appropriate interface register
 - Function code and sense lines are not needed (Transfer of data, control, and status information is always via the common I/O Bus)
- 

ISOLATED vs MEMORY MAPPED I/O

Isolated I/O

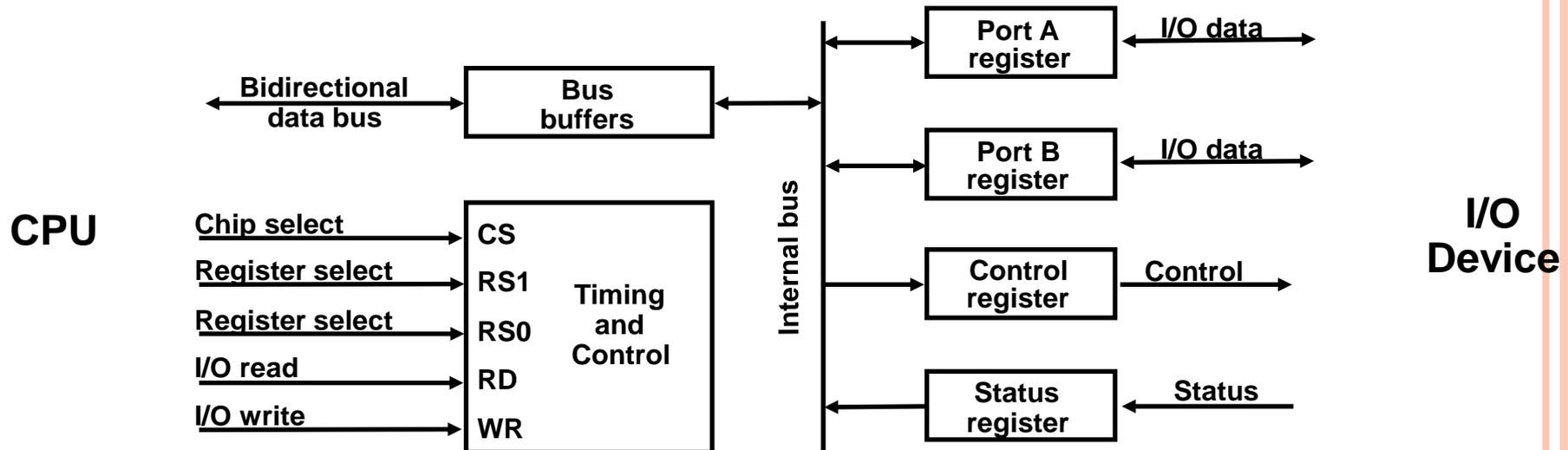
- **Separate I/O read/write control lines in addition to memory read/write control lines**
- **Separate (isolated) memory and I/O address spaces**
- **Distinct input and output instructions**

Memory-mapped I/O

- **A single set of read/write control lines
(no distinction between memory and I/O transfer)**
- **Memory and I/O addresses share the common address space**
 - > **reduces memory address range available**
- **No specific input or output instruction**
 - > **The same memory reference instructions can be used for I/O transfers**
- **Considerable flexibility in handling I/O operations**



I/O INTERFACE



CS	RS1	RS0	Register selected
0	x	x	None - data bus in high-impedance
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Programmable Interface

- Information in each port can be assigned a meaning depending on the mode of operation of the I/O device
 -> Port A = Data; Port B = Command; Port C = Status
- CPU initializes(loads) each port by transferring a byte to the Control Register
 -> Allows CPU can define the mode of operation of each port
 -> **Programmable Port**: By changing the bits in the control register, it is possible to change the interface characteristics

ASYNCHRONOUS DATA TRANSFER

Synchronous and Asynchronous Operations

Synchronous - All devices derive the timing information from common clock line

Asynchronous - No common clock

Asynchronous Data Transfer

Asynchronous data transfer between two independent units requires that *control signals* be transmitted between the communicating units to *indicate the time at which data is being transmitted*

Two Asynchronous Data Transfer Methods

Strobe pulse

- A strobe pulse is supplied by one unit to indicate the other unit when the transfer has to occur

Handshaking

- A control signal is accompanied with each data being transmitted to indicate the presence of data
- The receiving unit responds with another control signal to acknowledge receipt of the data

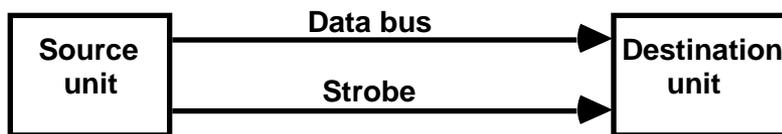


STROBE CONTROL

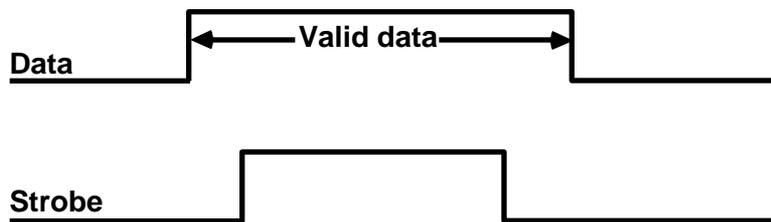
- * Employs a single control line to time each transfer
- * The strobe may be activated by either the source or the destination unit

Source-Initiated Strobe for Data Transfer

Block Diagram

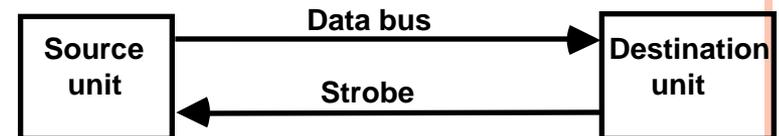


Timing Diagram

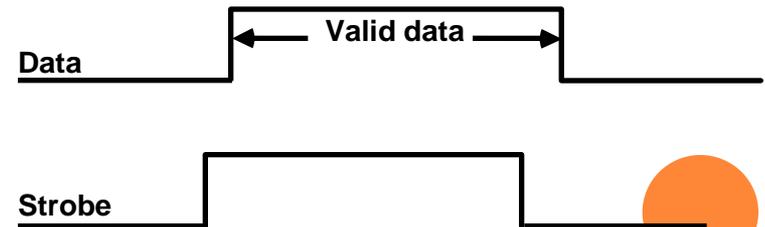


Destination-Initiated Strobe for Data Transfer

Block Diagram



Timing Diagram



HANDSHAKING

Strobe Methods

Source-Initiated

The source unit that initiates the transfer has no way of knowing whether the destination unit has actually received data

Destination-Initiated

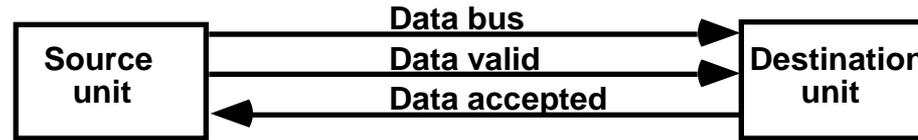
The destination unit that initiates the transfer has no way of knowing whether the source has actually placed the data on the bus

To solve this problem, the *HANDSHAKE* method introduces a second control signal to provide a *Reply* to the unit that initiates the transfer

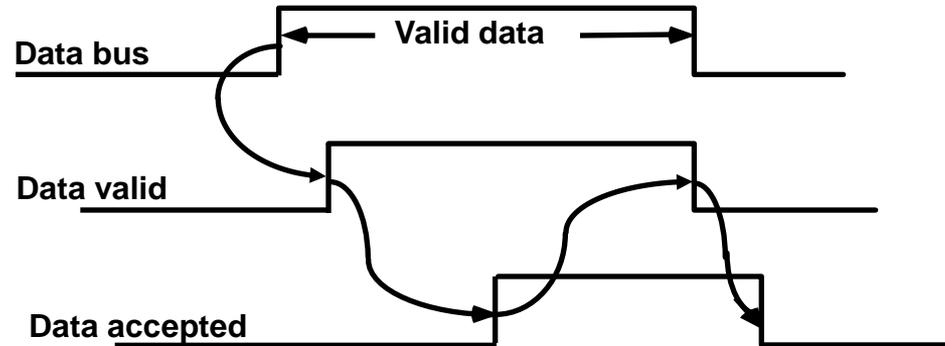


SOURCE-INITIATED TRANSFER USING HANDSHAKE

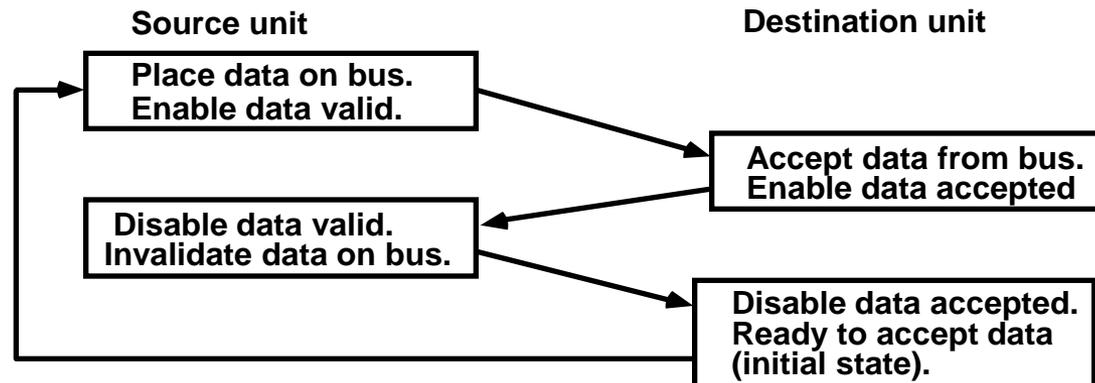
Block Diagram



Timing Diagram



Sequence of Events

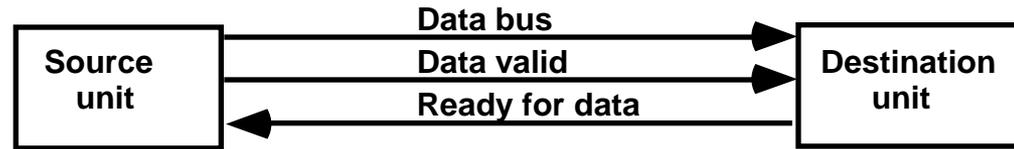


- * Allows arbitrary delays from one state to the next
- * Permits each unit to respond at its own data transfer rate
- * The rate of transfer is determined by the slower unit

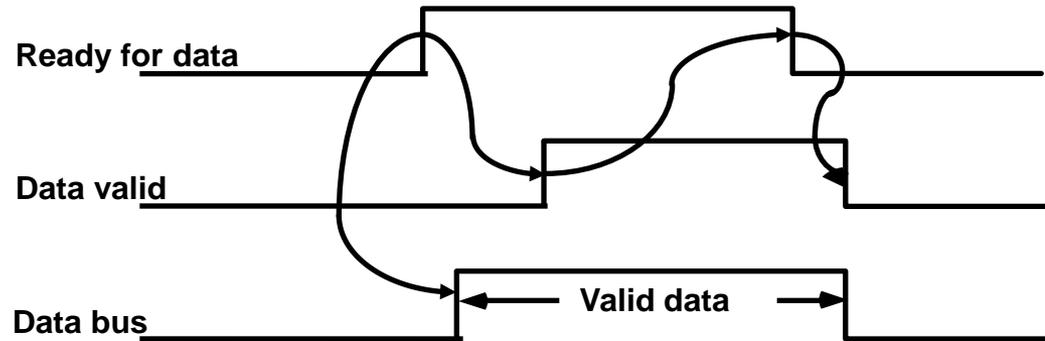


DESTINATION-INITIATED TRANSFER USING HANDSHAKE

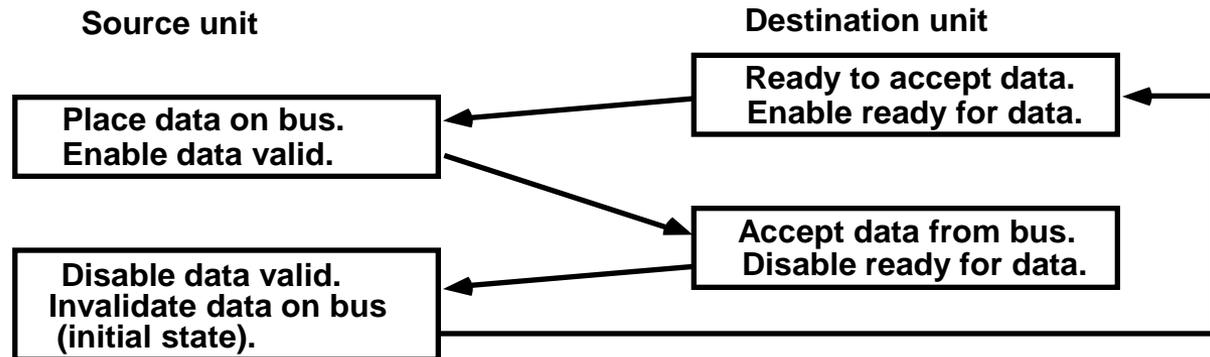
Block Diagram



Timing Diagram



Sequence of Events



- * Handshaking provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units
- * If one unit is faulty, data transfer will not be completed
 - > Can be detected by means of a *timeout* mechanism

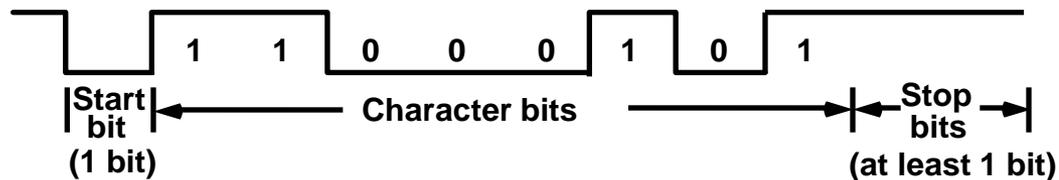
ASYNCHRONOUS SERIAL TRANSFER

Four Different Types of Transfer

Asynchronous serial transfer Synchronous serial transfer Asynchronous parallel transfer Synchronous parallel transfer
--

Asynchronous Serial Transfer

- Employs special bits which are inserted at both ends of the character code
- Each character consists of three parts; **Start bit**; **Data bits**; **Stop bits**.



A character can be detected by the receiver from the knowledge of 4 rules;

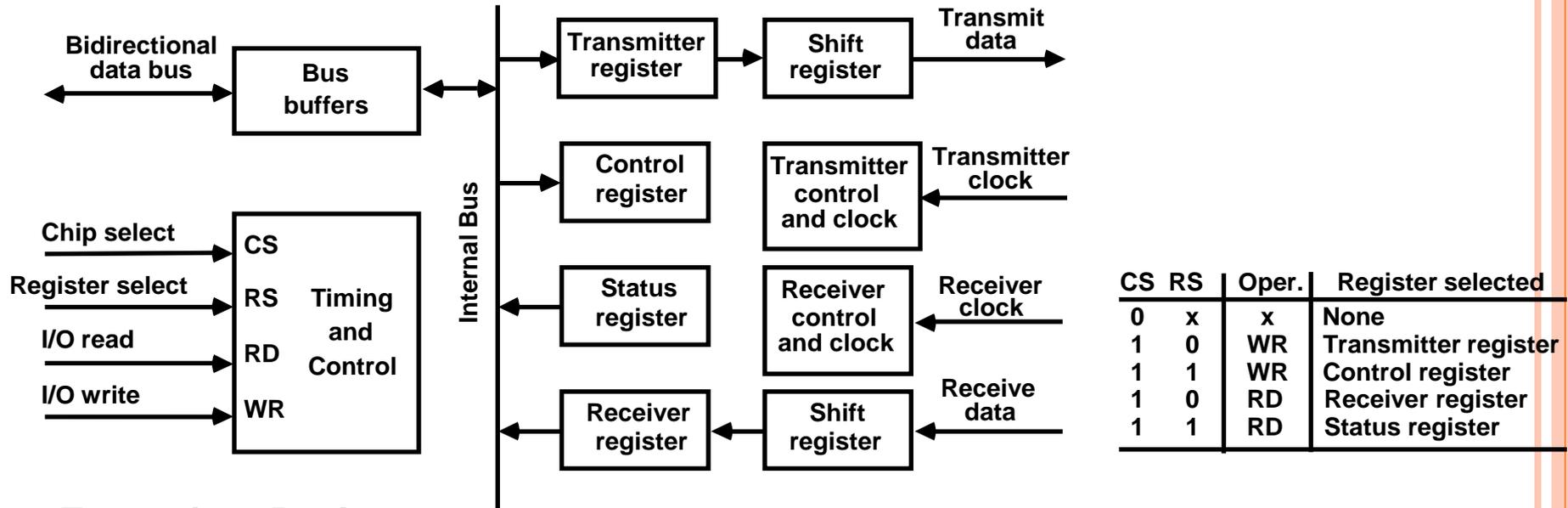
- When data are not being sent, the line is kept in the 1-state (idle state)
- The initiation of a character transmission is detected by a **Start Bit**, which is always a 0
- The character bits always follow the **Start Bit**
- After the last character, a **Stop Bit** is detected when the line returns to the 1-state for at least 1 bit time

The receiver knows in advance the transfer rate of the bits and the number of information bits to expect



UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER - UART -

A typical asynchronous communication interface available as an IC



Transmitter Register

- Accepts a data byte (from CPU) through the data bus
- Transferred to a shift register for serial transmission

Receiver

- Receives serial information into another shift register
- Complete data byte is sent to the receiver register

Status Register Bits

- Used for I/O flags and for recording errors

Control Register Bits

- Define baud rate, no. of bits in each character, whether to generate and check parity, and no. of stop bits

