Input / Output organization

- How program-controlled I/O is performed using polling
- How interrupts are used in I/O transfers
- Direct memory access as an I/O mechanism
- Synchronous and asynchronous bus operation
- Interface circuits
- Commercial standards, such as USB, SCSI, and PCI Express

Introduction

- Computer is its ability to exchange data with other devices.
- Communication capability enables a human operator
- Applications like an integral part of home appliances, manufacturing equipment, transportation systems, banking and point-of-sale (PoS) terminals.
- Input from a sensor switch, a digital camera, a microphone, or a fire alarm.
- Output like a sound signal sent to a speaker, or a digitally signal that changes the speed of a motor, opens a valve, or a robot to move in a specified manner.
- Computers should have the ability to exchange digital and analog information with a wide range of devices in many different environments.

Accessing I/O Devices

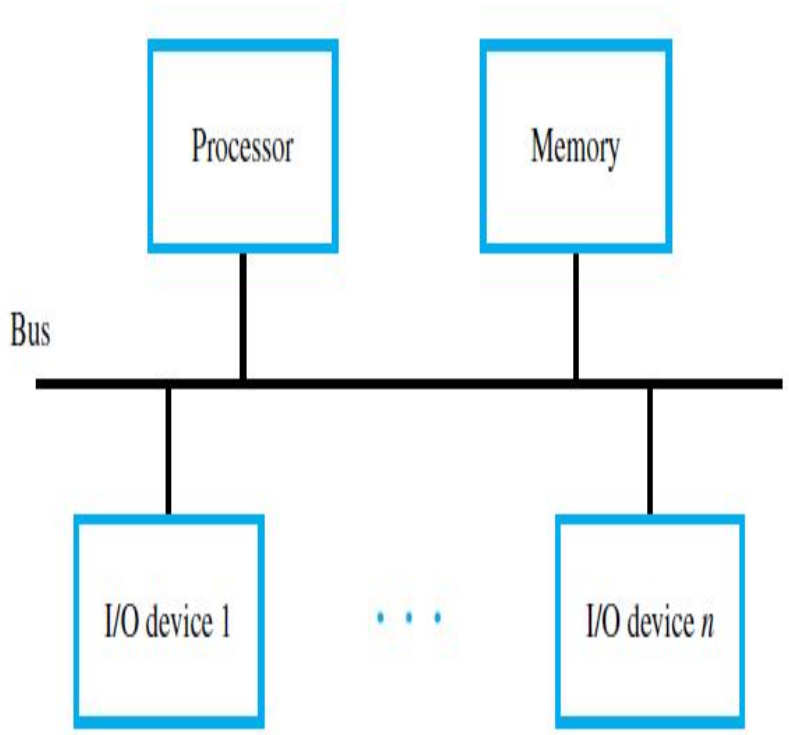
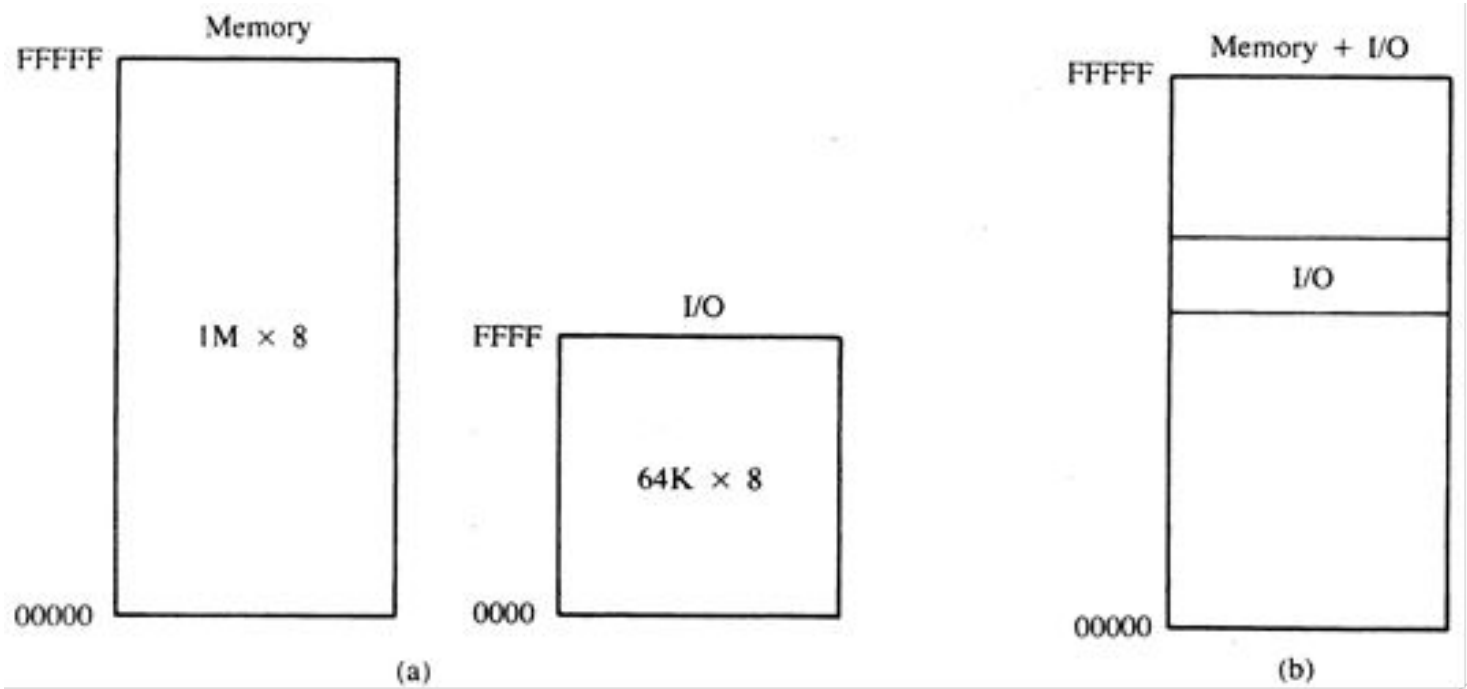


Figure 7.1 A single-bus structure.

- I/O device must appear to the processor as consisting of some addressable locations, just like the memory.
- The I/O devices and the memory share the same address space, this arrangement is called ***memory-mapped I/O.***



I/O Mapped I/O

Memory Mapped I/O

Count..

- Memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device.
- For example, if DATAIN is the address of a register in an input device, the instruction

  Load R2, DATAIN

  reads the data from the DATAIN register and loads them into processor register R2.
- Similarly, the instruction

  Store R2, DATAOUT

  sends the contents of register R2 to location DATAOUT, which is a register in an output device.

In Intel Processor IN and OUT instructions used

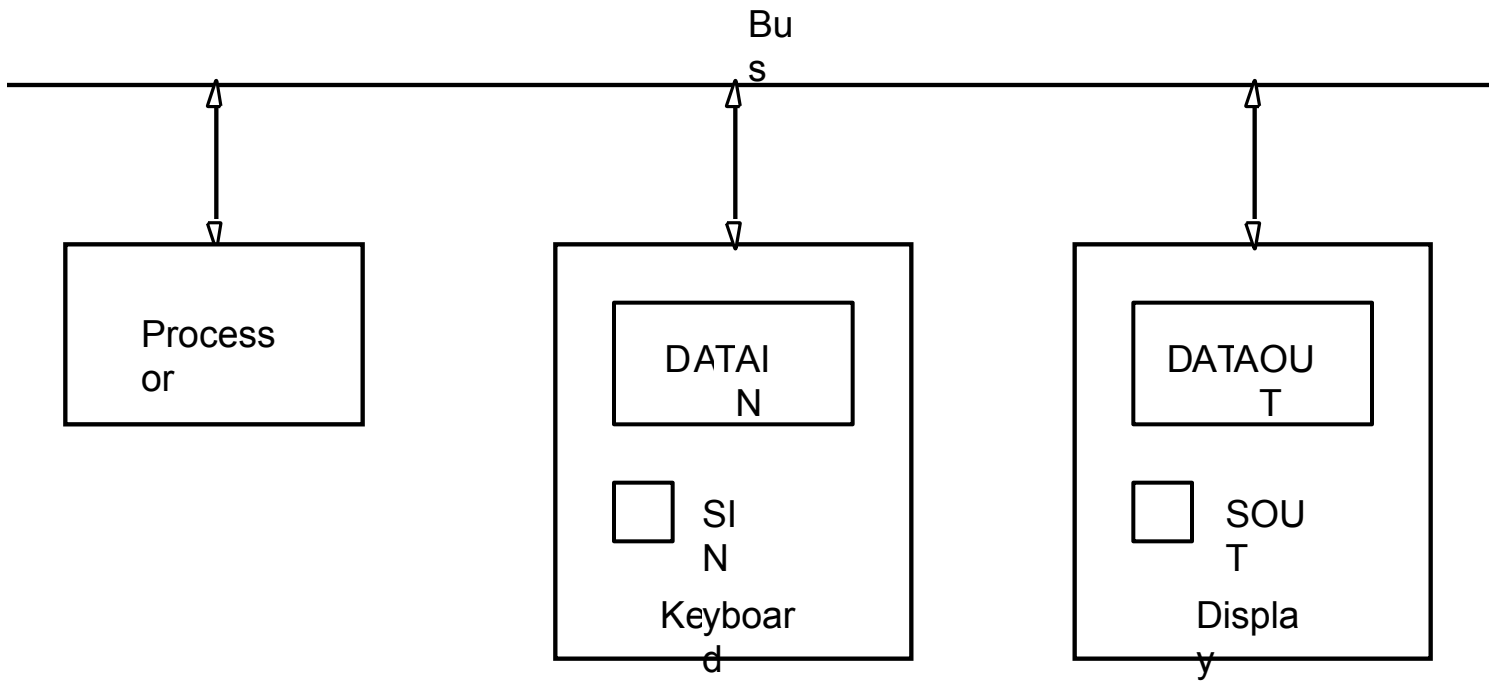


Figure 2.19 Bus connection for processor , keyboard, and display .

Hardware needed to access I/O devices

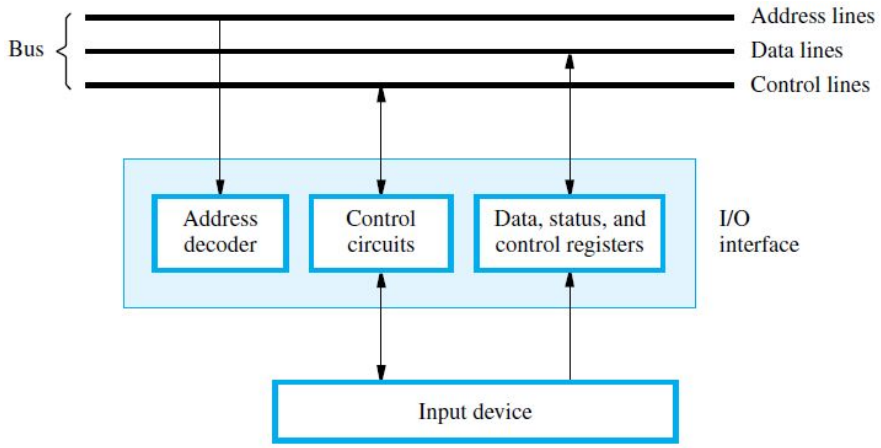


Figure 7.2 I/O interface for an input device.

- Processor places a particular address on the address lines, it is examined by the address decoders of all devices on the bus.
- The device that recognizes this address responds to the commands issued on the control lines.
- The processor uses the control lines to request either a Read or a Write operation, and the requested data are transferred over the data lines.
- Intel processor uses I/O mapped I/O.

Example

- Program to reads one line from the keyboard till return key press, store it in memory buffer and echoes it back to the display.

Count..

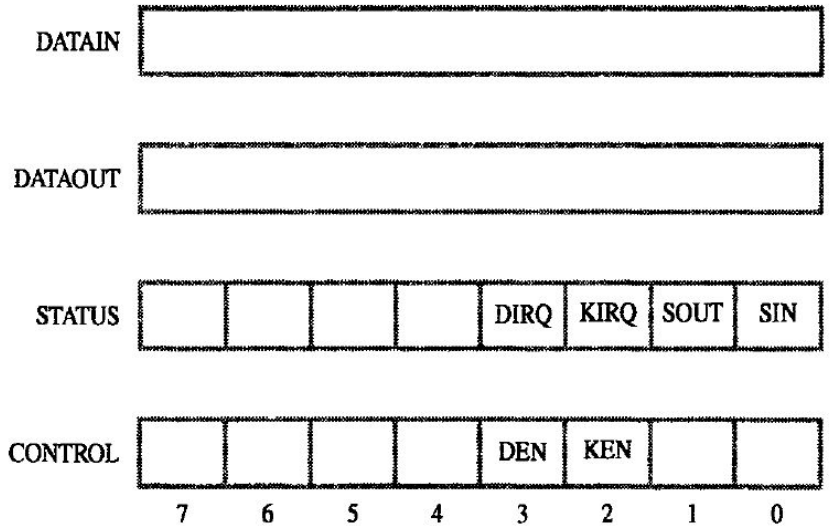


Figure 4.3 Registers in keyboard and display interfaces.

Program Controlled I/O

- Above Example is of Program Controlled I/O
- In this processor repeatedly checks a status flags to achieve the required synchronization between the processor and an input or output device
- Also called **Processor Polls the device**
- Other methods are Interrupt and DMA

Interrupts -1

- Polling method – Processor waits for response from I/O device. During wait period processor not able perform useful computation.
- Come out from these problem using Interrupt
- Can arrange for the I/O device to alert the processor when it becomes ready. It can do so by sending a hardware signal called an ***interrupt request*** to the processor.

Interrupt Example 1

- Consider a task that requires continuous extensive computations to be performed and the results to be printed on a printer.

Interrupt Example 2

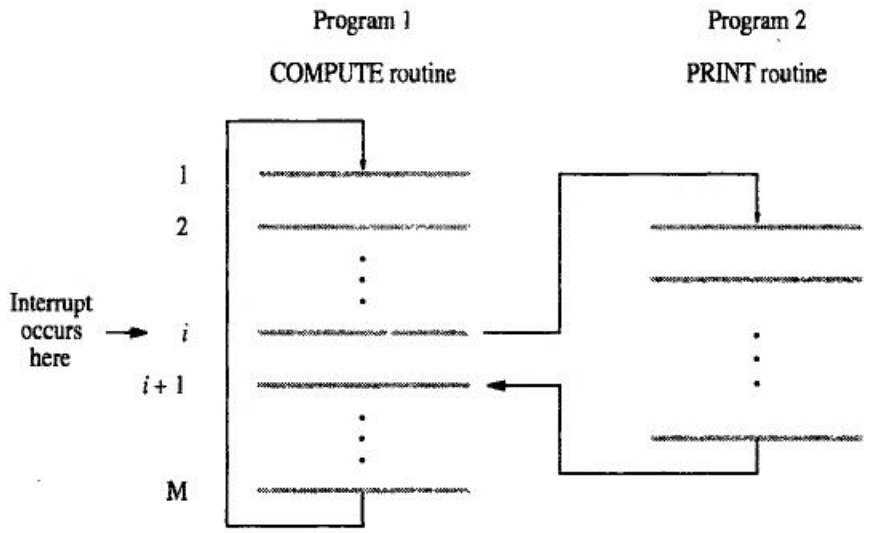


Figure 4.5 Transfer of control through the use of interrupts.

- COMPUTE produces a set of n lines of output.
- PRINT routine print this line
- After printing one line printer sends Interrupt request signal to the processor. (At i th line.)
- So processor interrupt the execution of COMPUTE routine and transfer the control to PRINT routine.

Interrupt -2

- The routine executed in response to an interrupt request is called the ***interrupt-service routine***.

  E.g. PRINT routine.
- ISR/ISP similar to Subroutine
- Processor responds or interrupt request responds to interrupted device by sending the control signal called **Interrupt Acknowledge**.
- When interrupt occurs during execution of program processor register used, flag status information must saved in stack before execution of the interrupted program is resumed.
- In this way, the original program can continue execution without being affected in any way by the interruption, **except for the time delay**.
- The task of saving and restoring information can be done automatically by the processor **or by program instructions.**

Interrupt -3

- The process of saving and restoring registers involves memory transfers that increase the total execution time, and hence represent execution overhead.
- Saving registers also 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***.
- The amount of information saved automatically by the processor when an interrupt request is accepted **should be kept to a minimum**.
- This kind of delay not acceptable in **Real-time processing**.
- The ISR returns to interrupted program using Return-from-Interrupt instruction.

Interrupt Hardware

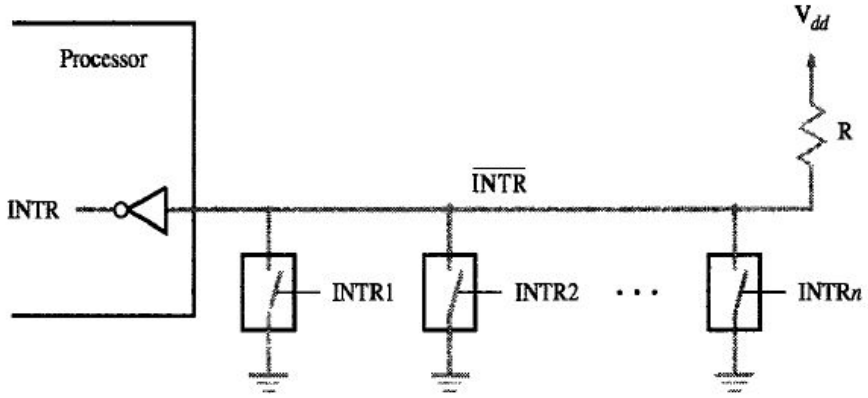


Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

- Interrupt request
- If several I/O can request an interrupt.
- A single interrupt request line may be used as shown in figure using switch.
- When device make interrupt request by closing switch the voltage on the line is ‘0’
- INTR is logic OR
- INTR= INTR1+INTR2+…..+INTRn
- INTR is active low signal

Enabling and Disabling Interrupt-1

- Interrupt can occur any time which alter the execution sequence.
- To provide programmer complete control over program execution event.
- So interruption of program carefully controlled i. e. by Enable and disable interrupt as desire
- Some situation interrupt have to ignore e.g. PRINT interrupt the processor even COMPUTE is not ready with text to print.

Enabling and Disabling Interrupt-2

- 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.

Enabling and Disabling Interrupt-3

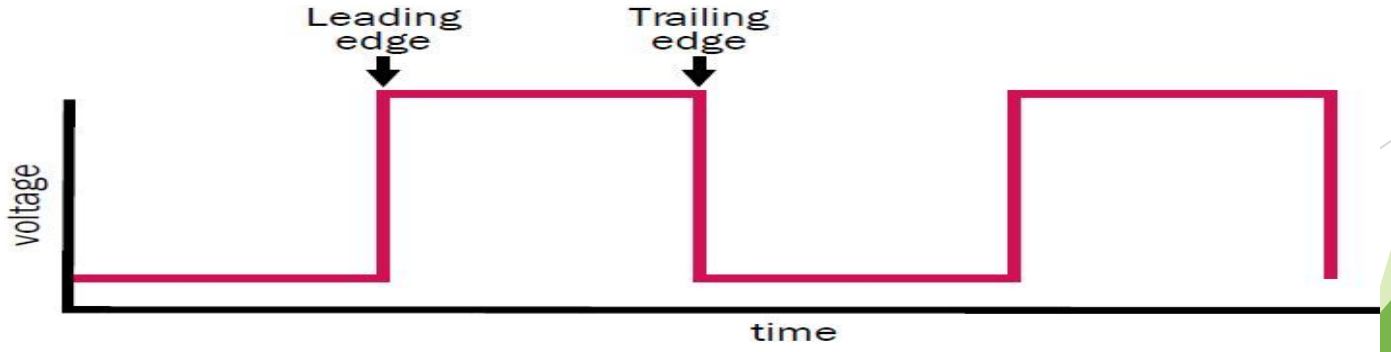
- First Possibility
  - Ignore the IR until complete the current ISR
  - i.e. first instruction of ISR is Interrupt Disable and last instruction is Interrupt Enable

Enabling and Disabling Interrupt-4

- 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.

Enabling and Disabling Interrupt-4

- 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.



Summarize the sequence of events involved in handling an interrupt request from a single device

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed and saves the contents of the PC and PS registers.
3. Interrupts are disabled by clearing the IE bit in the PS to 0.
4. The action requested by the interrupt is performed by the interrupt-service routine, during which time the device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5. Upon completion of the interrupt-service routine, the saved contents of the PC and PS registers are restored (enabling interrupts by setting the IE bit to 1), and execution of the interrupted program is resumed.

Handling Multiple Devices-1

- The situation where a number of devices capable of initiating interrupts are connected to the processor
- E.g. X and Y devices make IR

Handling Multiple Devices-2

- Several devices may request interrupts at exactly the same time. This gives rise to a number of questions:

1. How can the processor determine which device is requesting an interrupt?
2. Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?
3. Should a device be allowed to interrupt the processor while another interrupt is being serviced?
4. How should two or more simultaneous interrupt requests be handled?

Handling Multiple Devices-3

- 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

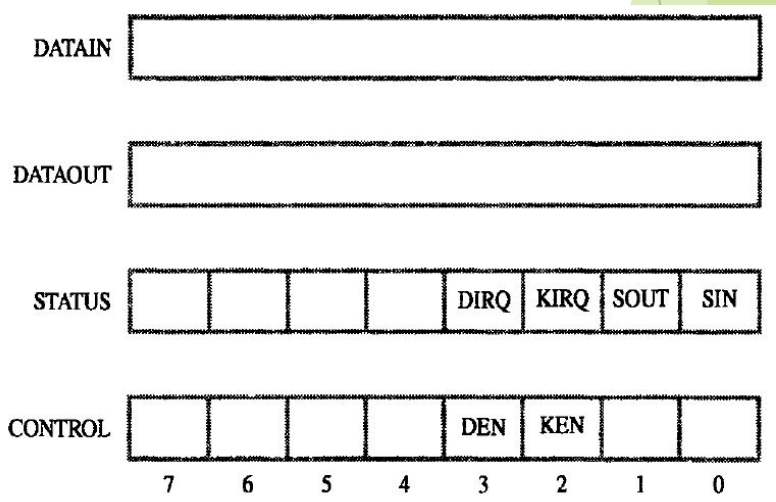


Figure 4.3 Registers in keyboard and display interfaces.

Handling Multiple Devices-4

- **Vectored Interrupt**
  - To reduce the time involved in the polling process, a device requesting an interrupt may identify itself directly to the processor. Then, the processor can immediately start executing the corresponding ISR.
  - A device requesting an interrupt can identify itself if it has its own interrupt-request signal, or if it can send a special code (like memory address of ISR) to the processor through the interconnection network.
  - A commonly used scheme is to allocate permanently an area in the memory to hold the addresses of interrupt-service routines. These addresses are usually referred to as **interrupt vectors**, and they are said to constitute the **interrupt-vector table**.
  - When an interrupt request arrives, the information provided by the requesting device is used as a pointer into the interrupt-vector table, and the address in the corresponding interrupt vector is automatically loaded into the program counter.

Handling Multiple Devices-5

- **Interrupt Nesting**
  - If request from more than one device.
  - Sometimes some device need immediate response from processor e.g. System Clock, Real time system.
  - This can be resolved by using **Priority Based Interrupt**.
  - Multiple-level priority
  - Interrupt requests will be accepted from some devices but not from others, depending upon the device's priority. To implement this scheme, we can assign a priority level to the processor that can be changed under program control.
  - Interrupt requests from higher-priority devices will accepted.
  - The processor's priority can be encoded in a few bits of the processor status register.

Handling Multiple Devices-6

- **Privileged Instruction**
  - Works in supervisor mode i.e. when executing operating system routine.
  - But in Used mode user program cannot accidently or intentionally change the priority which disrupt the system operation which can be avoided using privilege exception.

Handling Multiple Devices-7

- Multiple priority scheme implemented using individual INTR and INTA lines.

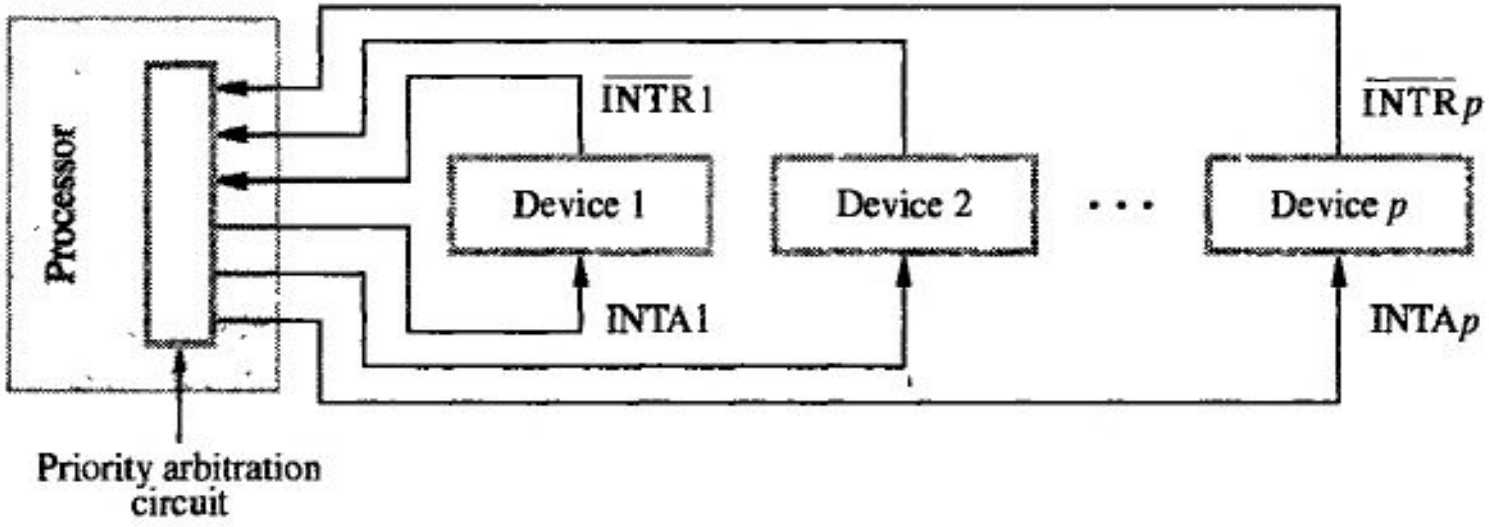


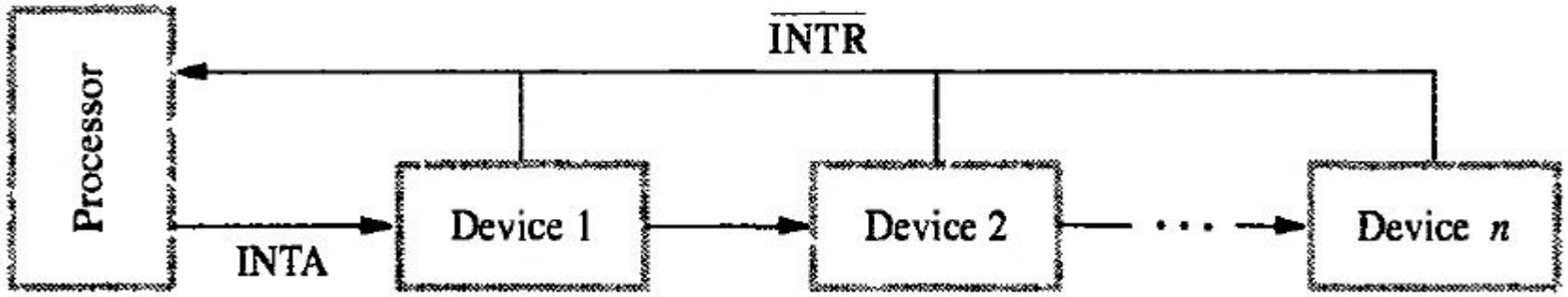
Figure 4.7 Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

- Priority arbitration circuit: A logic circuit which combines all interrupts but allows only the highest-priority request.

Handling Multiple Devices-8

- **Simultaneous Requests**
  - Polling the status registers of the I/O devices is the simplest such mechanism. In this case, priority is determined by the order in which the devices are polled.
  - When vectored interrupts are used, we must ensure that only one device is selected to send its interrupt vector code.

Handling Multiple Devices-9



(a) Daisy chain

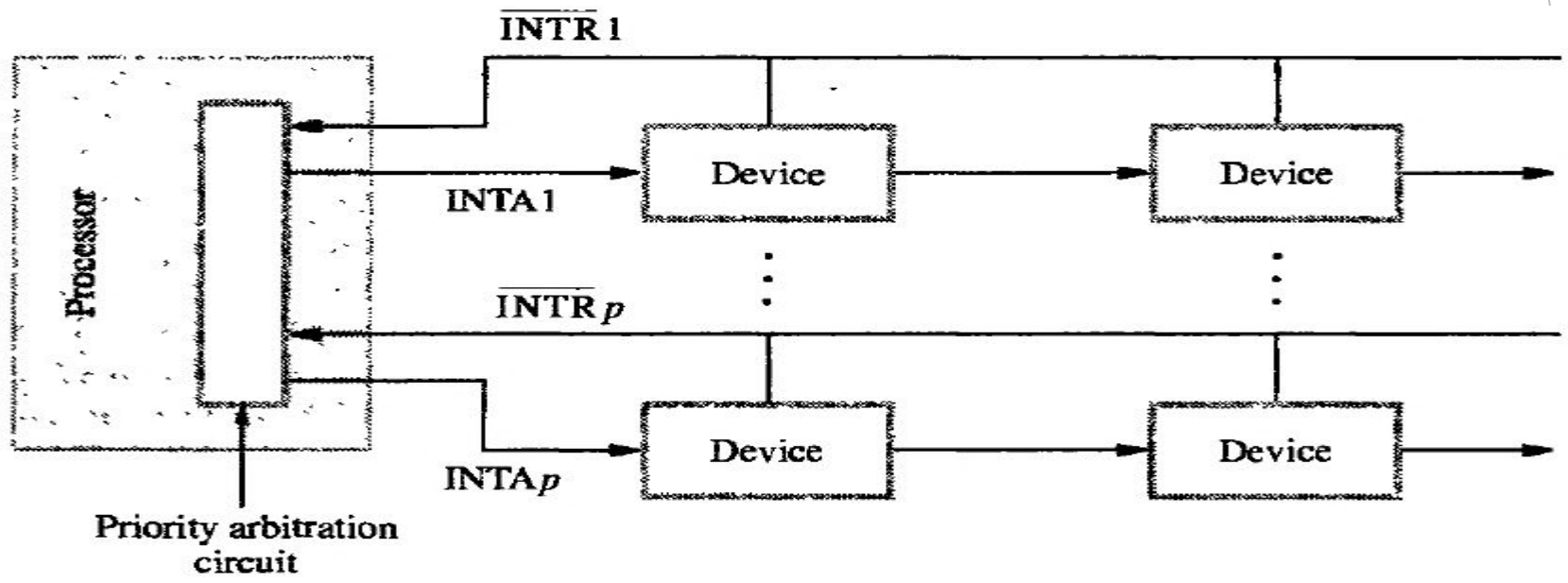
- INTR is common to all device
- INTA connected in daisy chain fashion where INTA signal propagate serially through the devices.

Handling Multiple Devices-10

- When IR is active on INTR line then INTA sent to Device 1 and passes to device 2 if it does not require any service.
- In daisy chain, the device that is electrically closest to the processor had the highest priority.

Handling Multiple Devices-11

- Combining the Multiple-priority and daisy chain.
- Where device organized in group and each have different priority level.



(b) Arrangement of priority groups

Controlling Device Request-1

- It is important to ensure that interrupt requests are generated only by those I/O devices that the processor is currently willing to recognize.
- Need a mechanism in the interface circuits of individual devices to control whether a device is allowed to interrupt the processor.
- The control needed is usually provided in the form of an *interrupt-enable* bit in the device's interface circuit.
- To reduce unnecessary interrupt from other I/O devices which are not used by current executing program.

Controlling Device Request-2

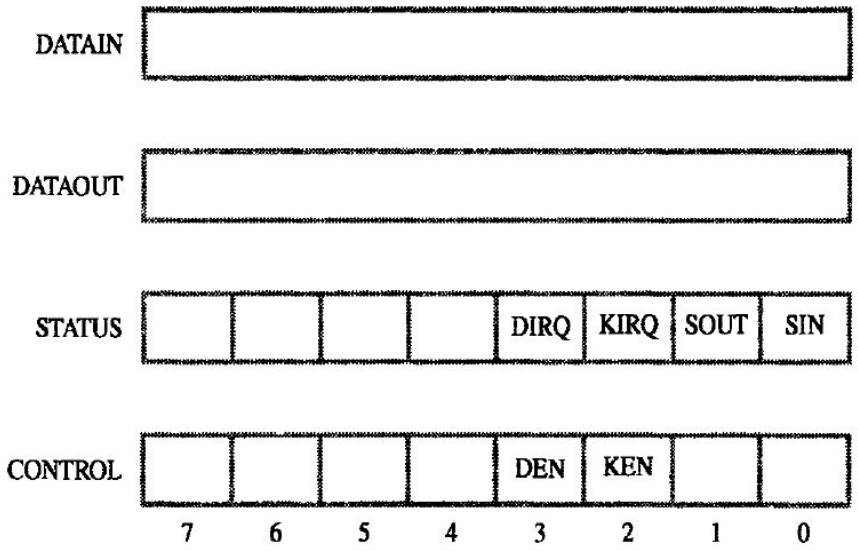


Figure 4.3 Registers in keyboard and display interfaces.

- Keyboard interrupt enable bit KEN
- Display interrupt enable bit DEN
- KEN and/or DEN = 1 **then only** interface circuit generate an IR.
- Same time KIRQ and/or DIRQ = 1 **indicate that both devices is requesting an interrupt.**

Controlling Device Request-3

- Example
- When a program wish to read an input line from the keyboard and store the character in successive byte location

Controlling Device Request-4

Main Program

```
        Move        #LINE,PNTR      Initialize buffer p
        Clear       EOL             Clear end-of-line
        BitSet      #2,CONTROL      Enable keyboard
        BitSet      #9,PS           Set interrupt-ena
        :
```

Interrupt-service routine

```
READ    MoveMultiple  R0–R1,–(SP)   Save registers R0
        Move          PNTR,R0       Load address poi
        MoveByte      DATAIN,R1     Get input charac
        MoveByte      R1,(R0)+        store it in mem
        Move          R0,PNTR       Update pointer.
        CompareByte   #$0D,R1       Check if Carriag
        Branch≠0      RTRN
        Move          #1,EOL        Indicate end of line.
        BitClear      #2,CONTROL    Disable keyboard interrupts.
RTRN    MoveMultiple  (SP)+,R0–R1   Restore registers R0 and R1.
        Return-from-interrupt
```

Figure 4.9 Using interrupts to read a line of characters from a keyboard via the registers in Figure 4.3.

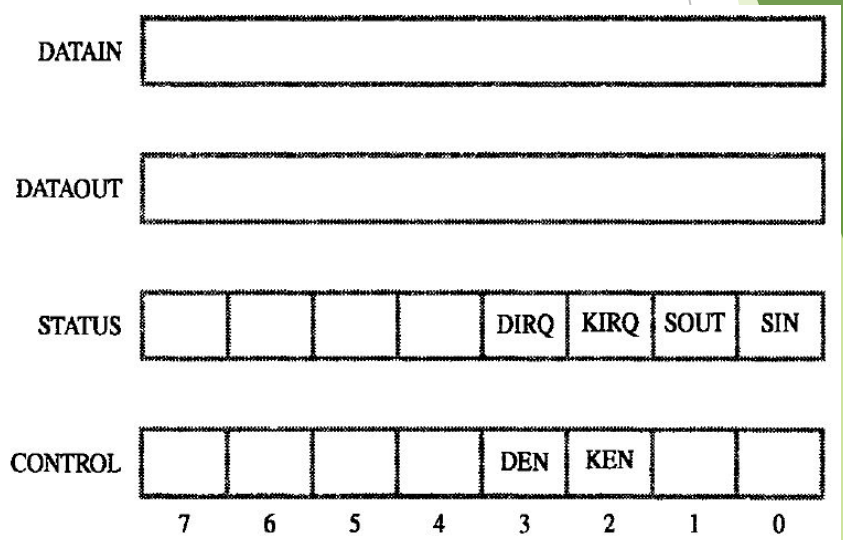


Figure 4.3 Registers in keyboard and display interfaces.

Exceptions-1

- Any event that causes an interruption.
- I/O interrupts are one example of an exception
- but other types of exceptions

Exceptions-2

- **Recovery from Errors**
  - Many computers include an error-checking code in the main memory, which allows detection of errors in the stored data. If an error occurs, the control hardware detects it and informs the processor by raising an interrupt.
  - The processor proceeds in exactly the same manner as in the case of an I/O interrupt request.
  - It suspends the program being executed and starts an exception-service routine, which takes appropriate action to recover from the error, if possible or to inform the user about it.

Exceptions-3

- **Debugging**
  - System software usually includes a program called a debugger, which helps the programmer find errors in a program.
  - The debugger uses exceptions to provide two important facilities: trace mode and breakpoints.

Exceptions-4
Debugging

- **Trace Mode**
  - In this mode, an interrupt occurs after the execution of every instruction. An interrupt-service routine in the debugger program is invoked each time this interrupt occurs.
  - It allows the debugger to assume execution control, enabling the user to enter commands for examining the contents of registers and memory locations.
  - When the user enters a command to resume execution of the object program, a Return from- interrupt instruction is executed.
  - The next instruction in the program being debugged is executed, then the debugger is activated again with another interrupt.

Exceptions-5
Debugging

- **Breakpoints**
  - Breakpoints provide a similar interrupt-based debugging facility, except that the object program being debugged is interrupted only at specific points indicated by the programmer.
  - For example, the programmer set a breakpoint to determine whether a particular subroutine.
  - A special instruction called **Trap or Software-interrupt** is usually used to implement breakpoints.

Exceptions-6

- **Privilege Exception**
  - To protect the OS from being corrupted by user programs certain instructions (Privilege Instructions) can executed only when processor is in superior mode.
  - In user mode these instruction not executed, so that user program does not change the priority level of the processor or program access area.

Direct Memory Access (DMA)-1

- To transfer large blocks of data at high speed, an alternative approach is used.
- A special control unit provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called DMA.

Direct Memory Access (DMA)-2

- DMA transfers are performed by a control circuit that is part of the I/O device interface called **DMA controller**.
- **For each word transferred, processor provides the memory address and all the bus signals that control data transfer.**
- Since processor has to transfer blocks of data, the DMA controller must increment the memory address for successive words and keep track of the number of transfers.

Direct Memory Access (DMA)-3

- DMA controller can transfer data without intervention by the processor, but its operation under the control of a program executed by the processor.
- **To initiate the transfer of a block of words, the processor sends the starting address, the number of words in the block, and the direction of the transfer.**
- On receiving this information, the DMA controller proceeds to perform the requested operation.
- **When the entire block has been transferred, the controller informs the processor by raising an interrupt signal.**
- While a DMA transfer is taking place, the program that requested the transfer cannot continue, and the processor can be used to execute another program.
- After the DMA transfer is completed, the processor can return to the program that requested the transfer.

Direct Memory Access (DMA)-4

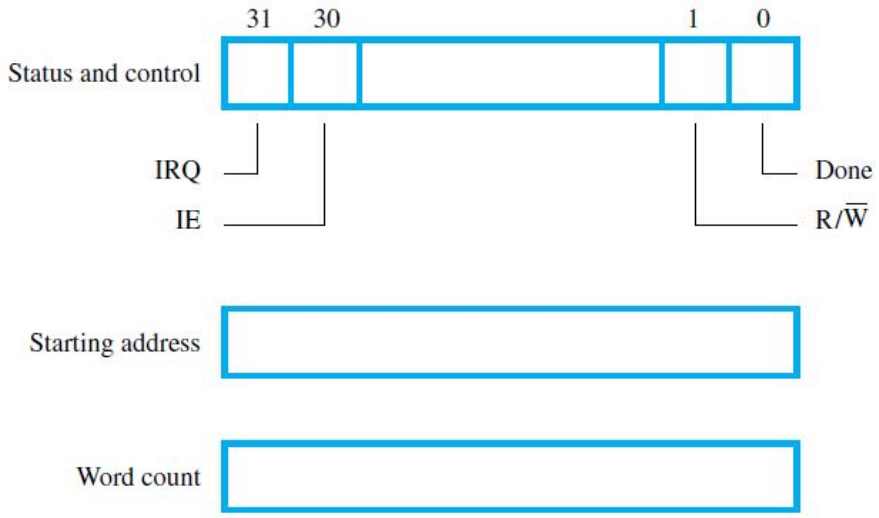


Figure 8.12 Typical registers in a DMA controller.

- DMA controller registers that are accessed by the processor to initiate transfer operations.
- Two registers are used for storing the Starting address and the word count.
- The third register contains status and control flags.
- The R/W bit determines the direction of the transfer.
  - When this bit is set to 1 by a program instruction, the controller performs a read operation, that is, it transfers data from the memory to the I/O device.
- When the controller has completed transferring a block of data and is ready to receive another command, it sets the Done flag to 1.
- Bit 30 is the Interrupt-enable flag, IE. When this flag is set to 1, it causes the controller to raise an interrupt after it has completed transferring a block of data.

Direct Memory Access (DMA)-5

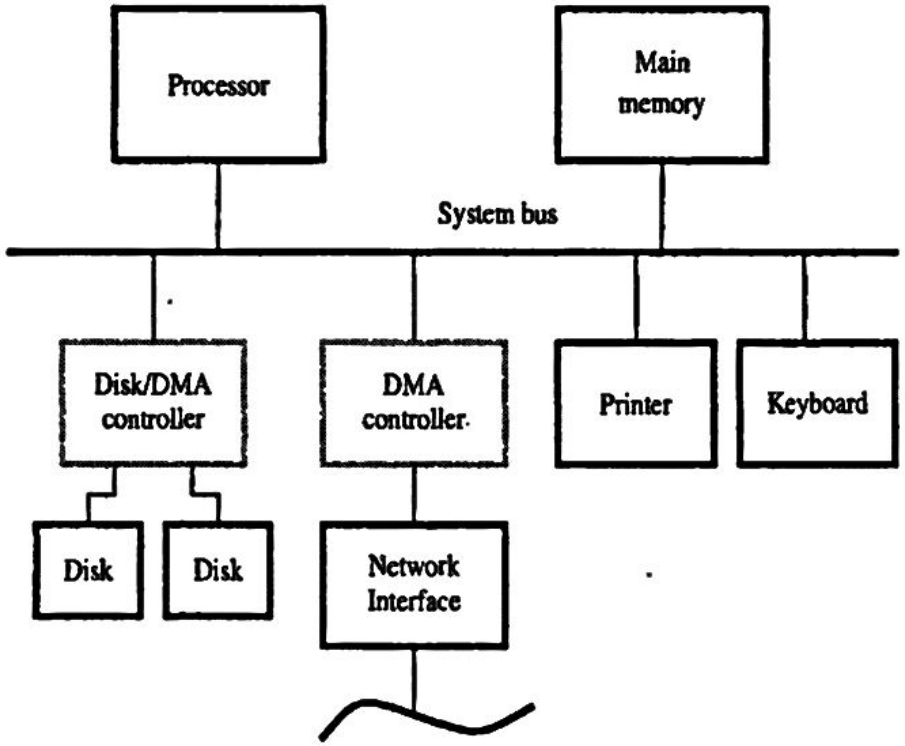


Figure 4.19 Use of DMA controllers in a computer system.

Direct Memory Access (DMA)-6

- **The disk controller, which controls two disks, also has DMA capability and provides two DMA channels. It can perform two independent DMA operations, as if each disk had its own DMA controller.**
- The registers needed to store the memory address, the word count, and so on are duplicated, so that one set can be used with each device.

Direct Memory Access (DMA)-8

- Requests by DMA devices for using the bus are always given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface, or a graphics display device.
- Since the processor originates most memory access cycles, the DMA controller can be said to **"steal" memory cycles from the processor**.
- **Interweaving technique is usually called cycle stealing**.
- Alternatively, the DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as **block or burst mode**.
- A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve these conflicts, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers.

Direct Memory Access (DMA)-9

- **Bus Arbitration**
  - The device that is allowed to initiate data transfers on the bus at any given time is called **the bus master**.
  - When the current master relinquishes control of the bus, another device can acquire this status.
  - **Bus arbitration** is the process by which the next device to become the bus master is selected and bus mastership is transferred to it.
  - The selection of the bus master must take into account the needs of various devices by establishing a priority system for gaining access to the bus.
  - There are two approaches to bus arbitration: **centralized and distributed.**
  - In centralized arbitration, a single bus arbiter performs the required arbitration. In distributed arbitration, all devices participate in the selection of the next bus master.

Direct Memory Access (DMA)-10

- **Bus Arbitration**
  - **Centralized Arbitration**
  - The bus arbiter may be the processor or a separate unit connected to the bus. The processor is normally the bus master unless it grants bus mastership to one of the DMA controllers.
  - A DMA controller indicates that it needs to become the bus master by activating the **Bus-Request line BR (Active Low)**.
  - When **Bus-Request is activated,** the processor activates the **Bus-Grant signal, BG1**, indicating to the DMA controllers that they may use the bus when it becomes free.
  - This signal is connected to all **DMA controllers using a daisy-chain arrangement**.
  - The current bus master indicates to all device that it is using the bus by activating line called **Bus-Busy (Active Low)**
  - Bus-Busy to prevent other devices from using the bus at the same time

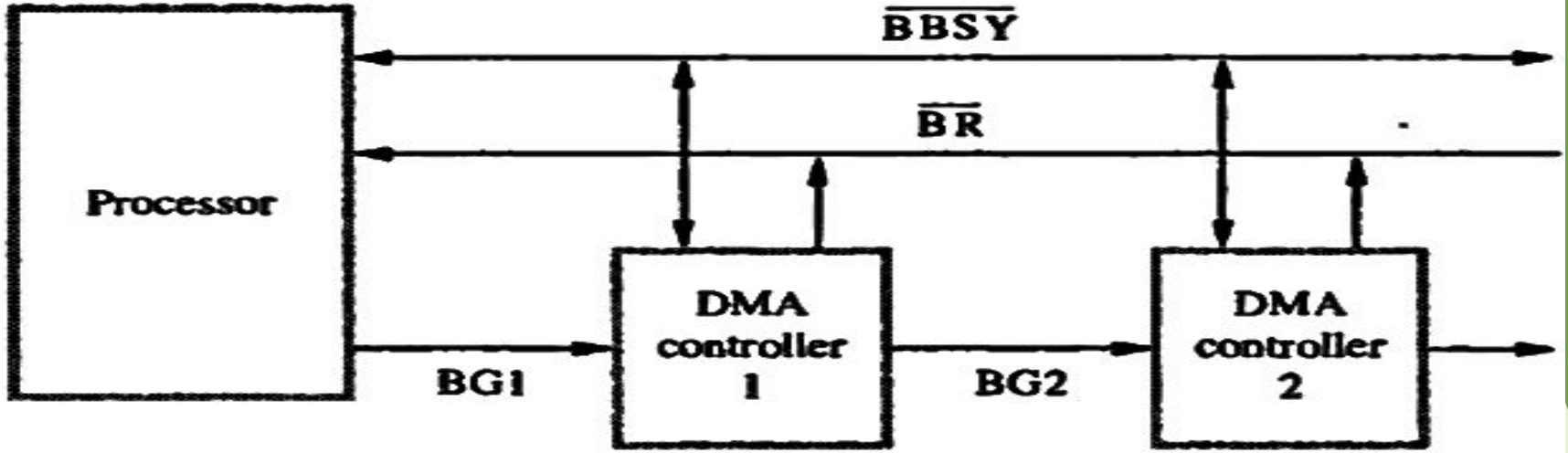


Figure 4.20 A simple arrangement for bus arbitration using a daisy chain.

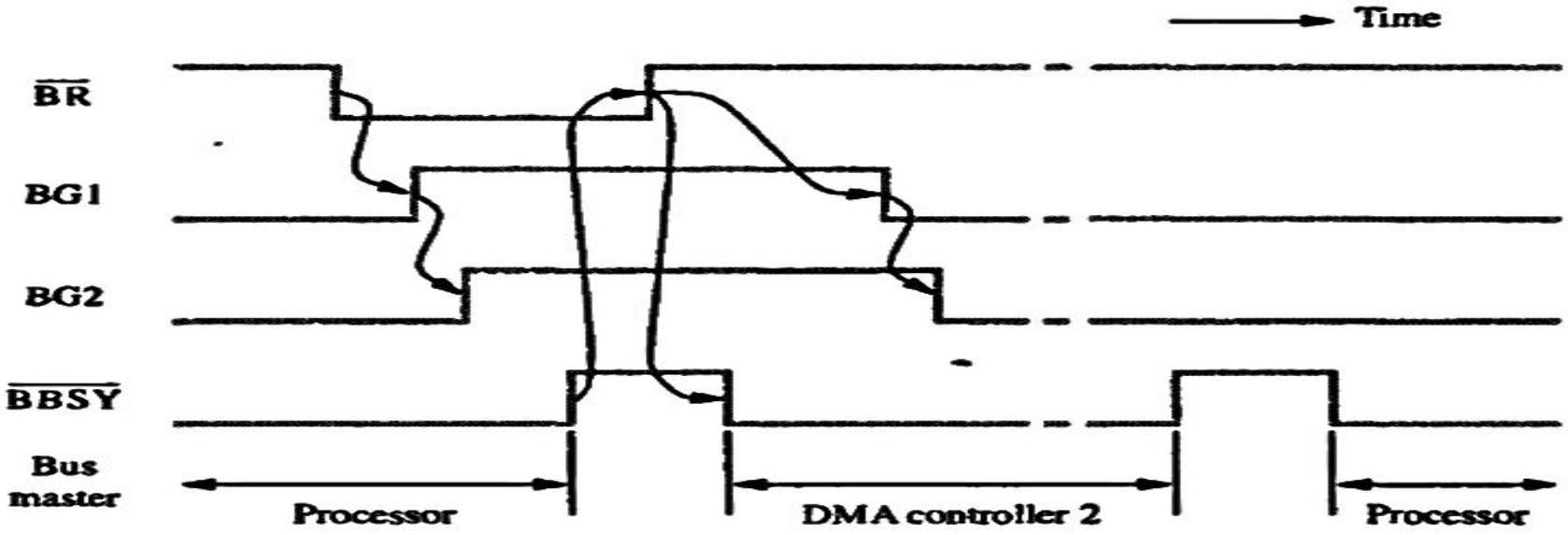


Figure 4.21 Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

Direct Memory Access (DMA)-12

- **Bus Arbitration**
  - **Distributed Arbitration**
  - Distributed arbitration means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter.
  - Each device on the bus assigned a 4-bit identification number. When one or more devices request the bus, they assert the **Start Arbitration (Active Low)** signal and place their 4-bit ID numbers on four line, ARB0 through ARB3 .
  - A winner is selected as a result of the interaction among the signals transmitted over those liens by all contenders.
  - The net outcome is that the code on the four lines represents the request that has the highest ID number.

Direct Memory Access (DMA)-13

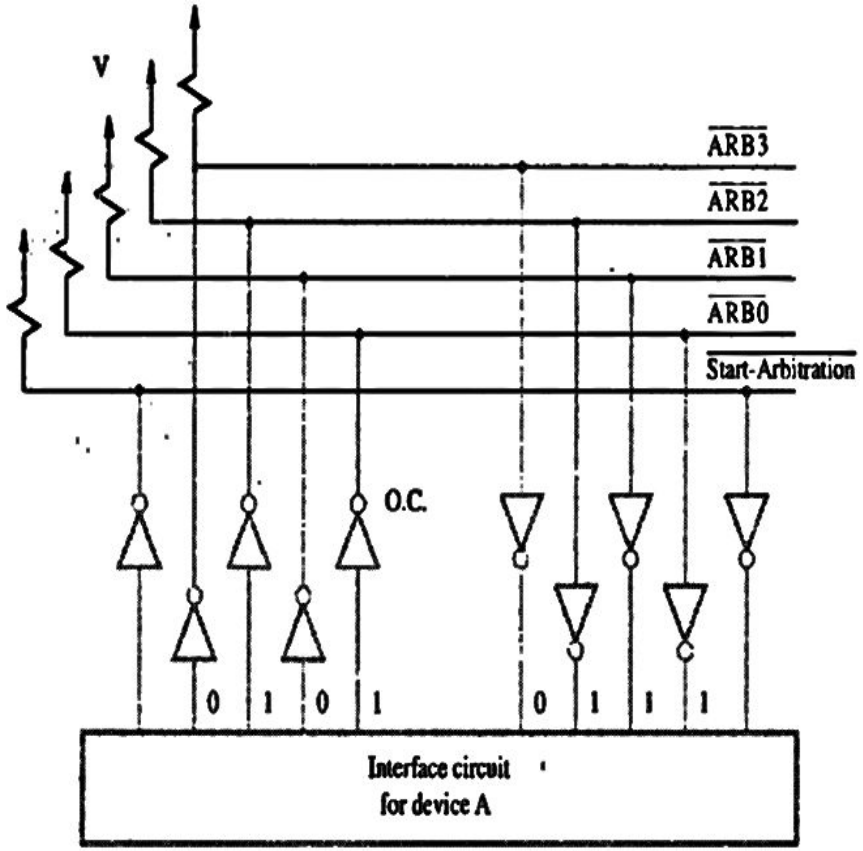


Figure 4.22 A distributed arbitration scheme.

- Device A and B having ID 5 (0101) and 6 (0110).
- Both sent the ID
- Code seen by both device is 0111.
- Each device compares the pattern on the arbitration lines to it own ID, starting from MSB. If it detects a difference at any bit position, it disables its drivers at that bit position and for all lower order bits.

Direct Memory Access (DMA)-13

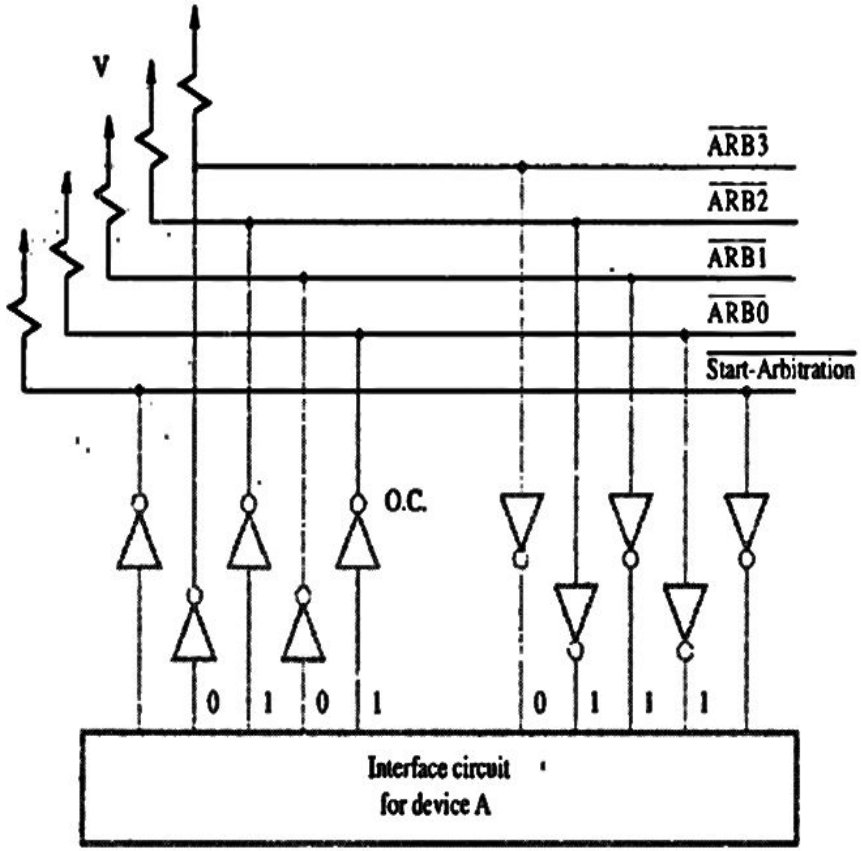


Figure 4.22 A distributed arbitration scheme.

- It does so by placing 0 at the input of these drivers.
- Device-A detects a difference on line ARB1 so it disable line ARB1 and ARB0.
- So Device-B wins because pattern change to 0110.

Buses-1

- The bus protocol determines when a device may place information on the bus, when it may load the data on the bus into one of its registers, and so on.
- These rules are implemented by control signals that indicate what and when actions are to be taken.
- **Master or initiator- Device which initiate data transfers**
- **Slave or target- Device addressed by master**

Buses-2 Synchronous Bus

- All devices derive timing information from a control line called the *bus clock*
- The timing diagram shows an **idealized representation** of the actions that take place on the bus lines.

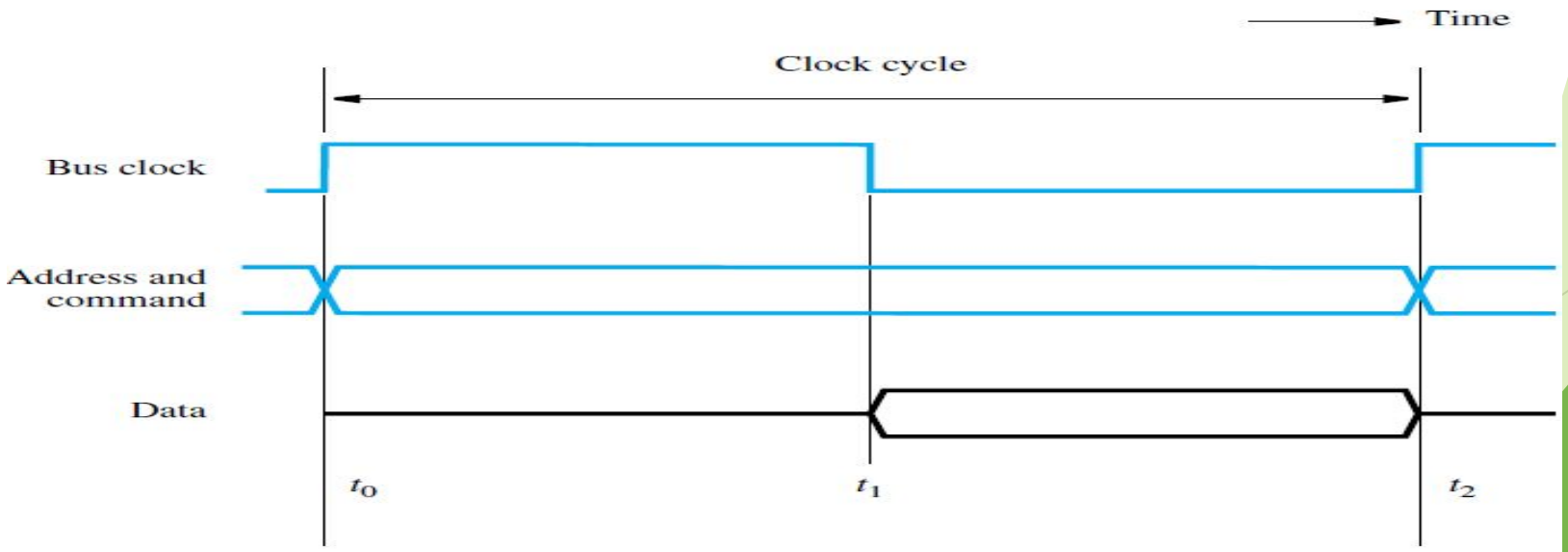


Figure 7.3 Timing of an input transfer on a synchronous bus.

Buses-3 Synchronous Bus

- The sequence of signal events during an input (Read) operation.
- At time ***t*0**, the master places the device address on the address lines and sends a command on the control lines indicating a Read operation.
- Information travels over the bus.
- The clock pulse width, ***t*1 – *t*0**, must be longer than the maximum propagation delay over the bus. Also, it must be long enough to allow all devices to decode the address and control signals, so that the addressed device (the slave) can respond at time ***t*1** by placing the requested input data on the data lines.
- At the end of the clock cycle, at time ***t*2**, the master loads the data on the data lines into one of its registers.
- To be loaded correctly into a register, data must be available for a period greater than the setup time of the register.
- Hence, the period ***t*2 – *t*1** must be greater than the maximum propagation time on the bus plus the setup time of the master's register.

Buses-4 Synchronous Bus

- The **exact times at** which signals change state are somewhat different from **idealized representation**, because of propagation delays on bus wires and in the circuits of the devices.

Buses-5 Synchronous Bus

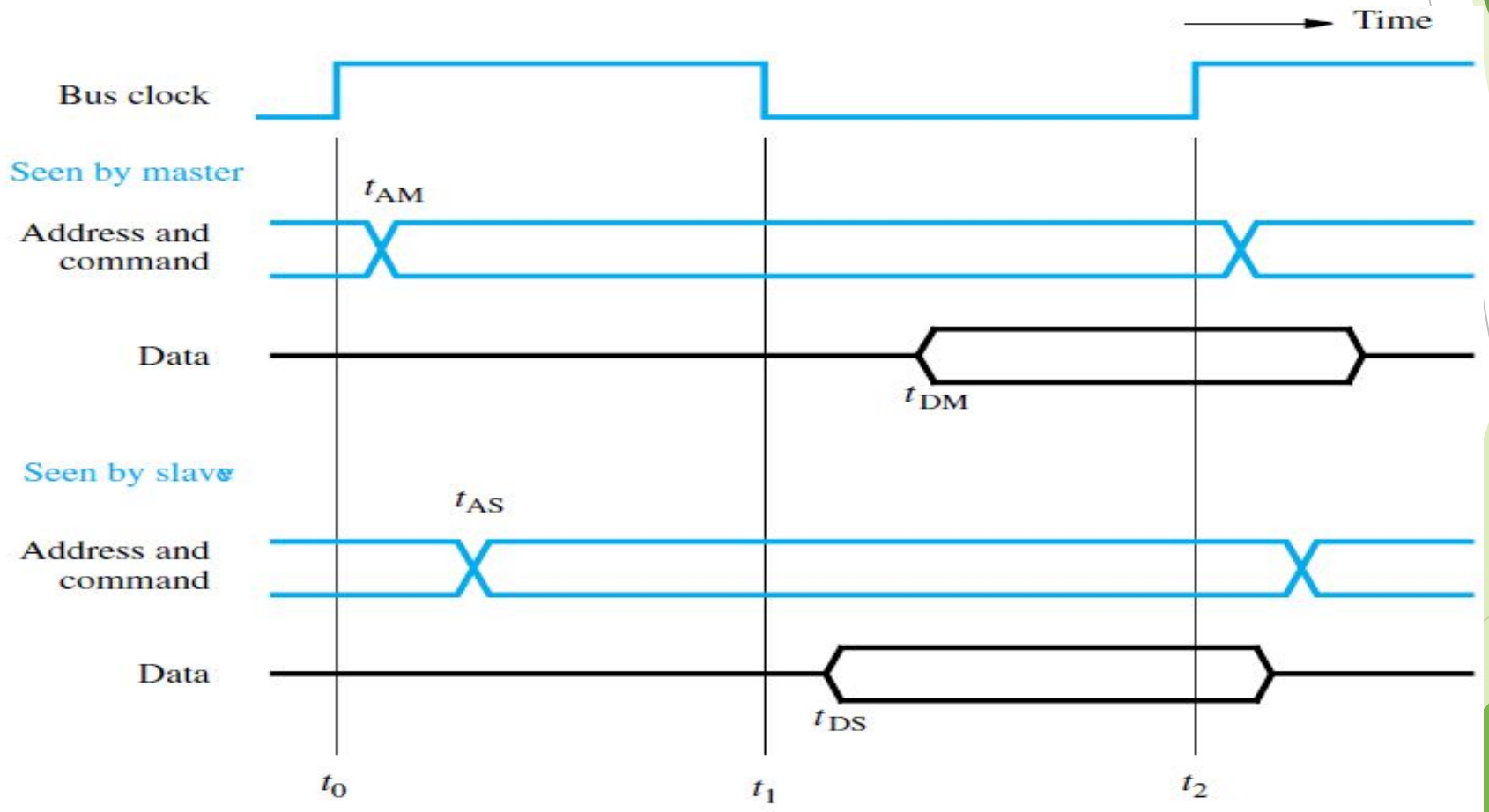


Figure 7.4 A detailed timing diagram for the input transfer of Figure 7.3.

Buses-6 Synchronous Bus

- The master sends the address and command signals on the rising edge of the clock at the beginning of the clock cycle (**at *t*0**). However, these signals do not actually appear on the bus until t_{AM}, largely due to the delay in the electronic circuit output from the master to the bus lines.
- A short while later, at t_{AS}, the signals reach the slave. The slave decodes the address, and at ***t*1** sends the requested data.
- Here again, the data signals do not appear on the bus until t_{DS}.
- They travel toward the master and arrive at t_{DM}. At ***t*2**, the master loads the data into its register.
- Hence the period $t2 - t_{DM}$ must be greater than the setup time of that register. The data must continue to be valid after ***t*2** for a period equal to the hold time requirement of the register.

Buses-7 Synchronous Bus Multiple-Cycle Data Transfer

- However, it has some limitations.
- Because a transfer has to be completed within one clock cycle, the clock period, $t2 - t0$, must be chosen to accommodate the longest delays on the bus and the slowest device interface.
- This forces all devices to operate at the speed of the slowest device.
- To overcome these limitations, most buses incorporate control signals that represent a **response from the device**. These signals inform the master that the slave has recognized its address and that it is ready to participate in a data transfer operation. They also make it possible to adjust the duration of the data transfer period to match the response speeds of different devices.
- This is often accomplished by allowing a complete data transfer operation to span several clock cycles. Then, the number of clock cycles involved can vary from one device to another.

Buses-8 Synchronous Bus Multiple-Cycle Data Transfer

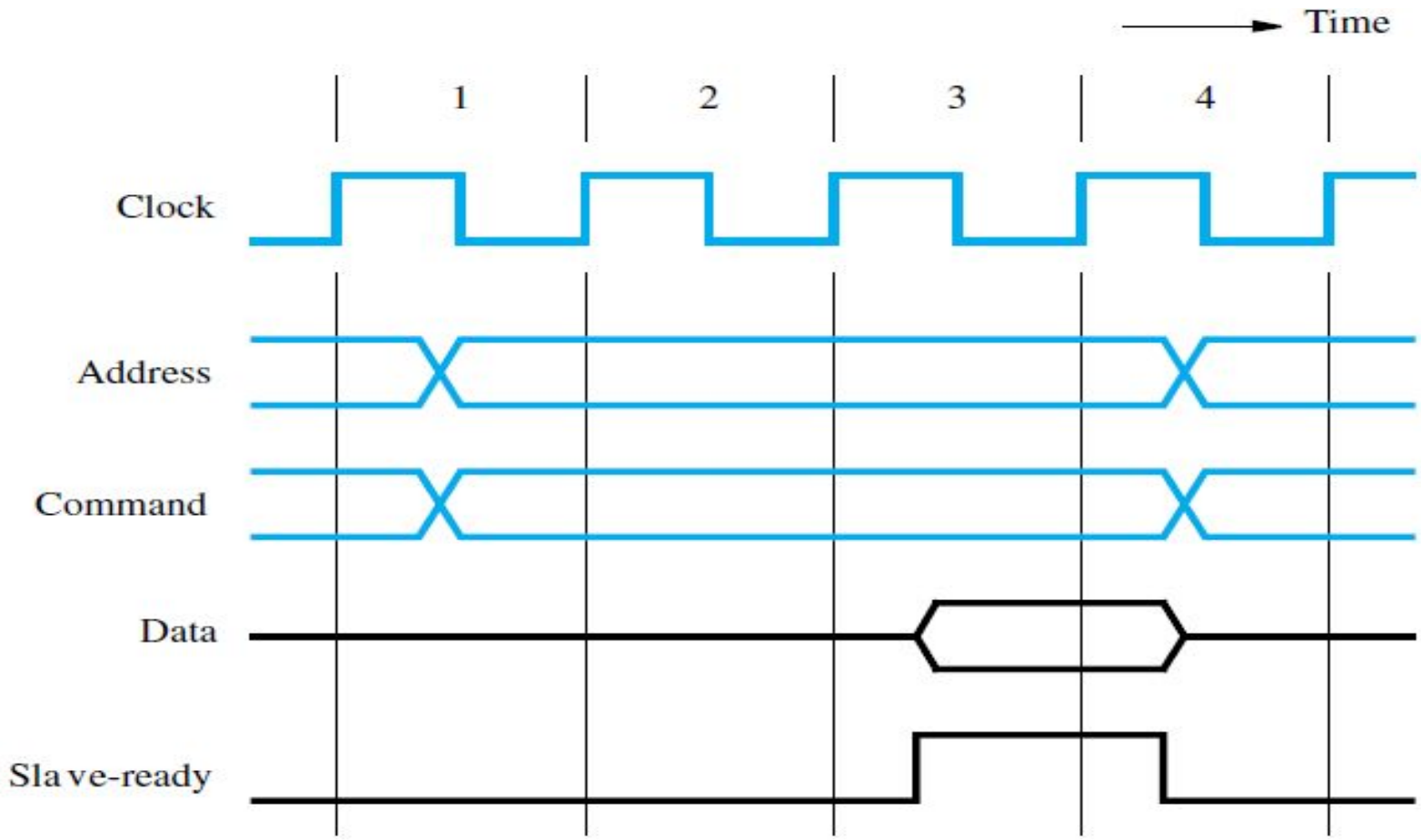


Figure 7.5 An input transfer using multiple clock cycles.

Buses-9 Synchronous Bus Multiple-Cycle Data Transfer

- During clock cycle 1, the master sends address and command information on the bus, requesting a Read operation.
- The slave receives this information and decodes it. It begins to access the requested data on the active edge of the clock at the beginning of clock cycle 2.
- Due to the delay involved in getting the data, the slave cannot respond immediately. The data become ready and are placed on the bus during clock cycle 3.
- The slave asserts a control signal called Slave-ready at the same time.
- The master, which has been waiting for this signal, loads the data into its register at the end of the clock cycle.
- The slave removes its data signals from the bus and returns its Slave-ready signal to the low level at the end of cycle 3.
- The bus transfer operation is now complete, and the master may send new address and command signals to start a new transfer in clock cycle 4.
- If the addressed device does not respond at all, the master waits for some predefined maximum number of clock cycles, then aborts the operation.

Buses-10 Asynchronous Bus

- An alternative scheme for controlling data transfers on a bus is based on the use of a *handshake* protocol between the master and the slave.
- A handshake is an exchange of command and response signals between the master and the slave.

Buses-10 Asynchronous Bus

- The master places the address and command information on the bus. Then it indicates to all devices that it has done so by activating the Master-ready line.
- This causes all devices to decode the address. The selected slave performs the required operation and informs the processor that it has done so by activating the Slave-ready line.
- The master waits for Slave-ready to become asserted before it removes its signals from the bus.
- In the case of a Read operation, it also loads the data into one of its registers.

Buses-10 Asynchronous Bus

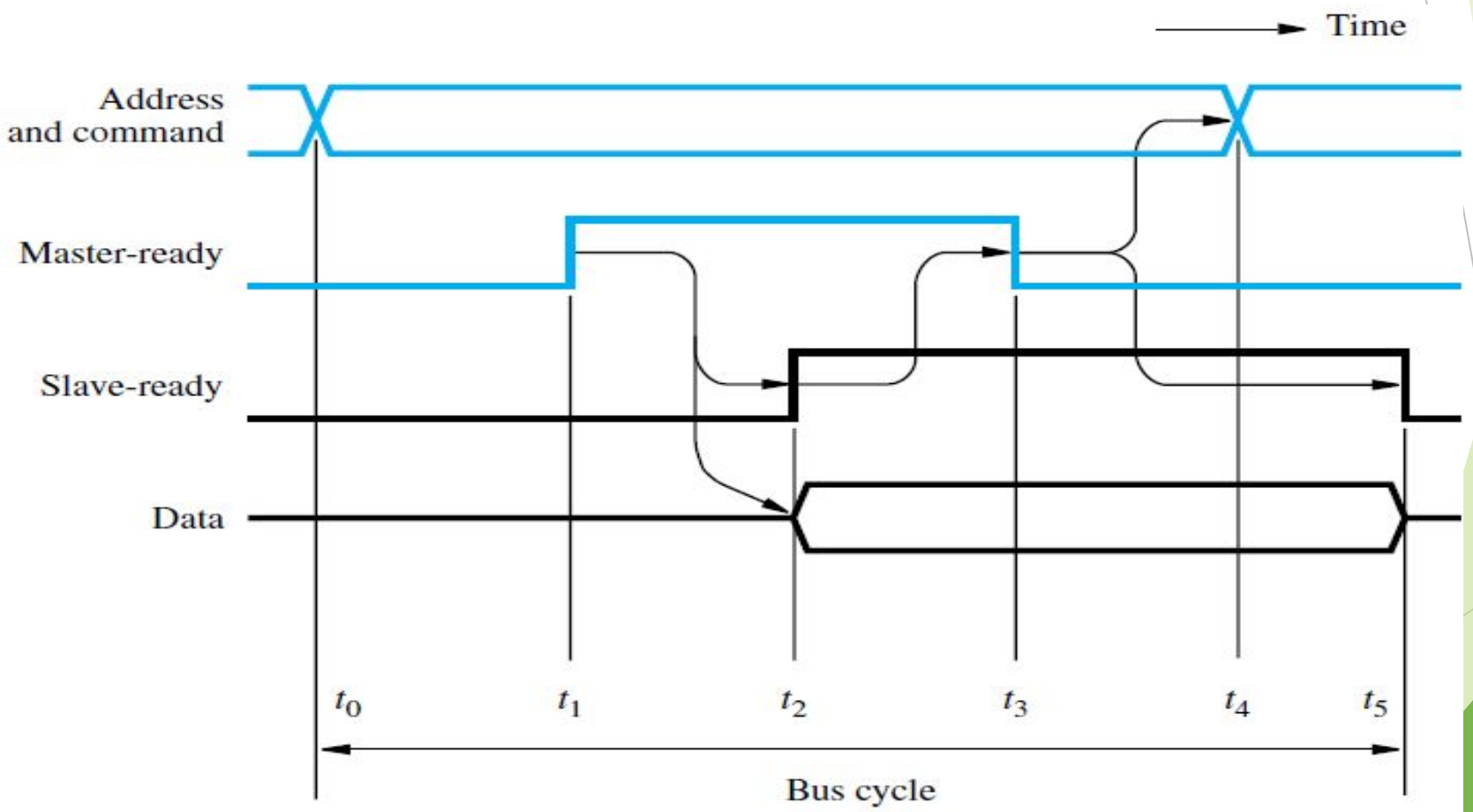


Figure 7.6 Handshake control of data transfer during an input operation.

Buses-10 Asynchronous Bus

- *t*0—The master places the address and command information on the bus, and all devices on the bus decode this information.
- *t*1—The master sets the Master-ready line to 1 to inform the devices that the address and command information is ready. Sufficient time should be allowed for the device interface circuitry to decode the address. The delay needed can be included in the period *t*1 – *t*0.
- *t*2—The selected slave, having decoded the address and command information, performs the required input operation by placing its data on the data lines. At the same time, it sets the Slave-ready signal to 1. If extra delays are introduced by the interface circuitry before it places the data on the bus, the slave must delay the Slave-ready signal accordingly. The period *t*2 – *t*1 depends on the distance between the master and the slave and on the delays introduced by the slave's circuitry.

Buses-11 Asynchronous Bus

- *t*3—The Slave-ready signal arrives at the master, indicating that the input data are available on the bus. After a delay to the master loads the data into its register. Then, it drops the Master-ready signal, indicating that it has received the data.
- *t*4—The master removes the address and command information from the bus. The delay between *t*3 and *t*4 is again intended to allow for bus skew. Erroneous addressing may take place if the address, as seen by some device on the bus, starts to change while the Master-ready signal is still equal to 1.
- *t*5—When the device interface receives the 1-to-0 transition of the Master-ready signal, it removes the data and the Slave-ready signal from the bus. This completes the input transfer.

Buses-12 Asynchronous Bus

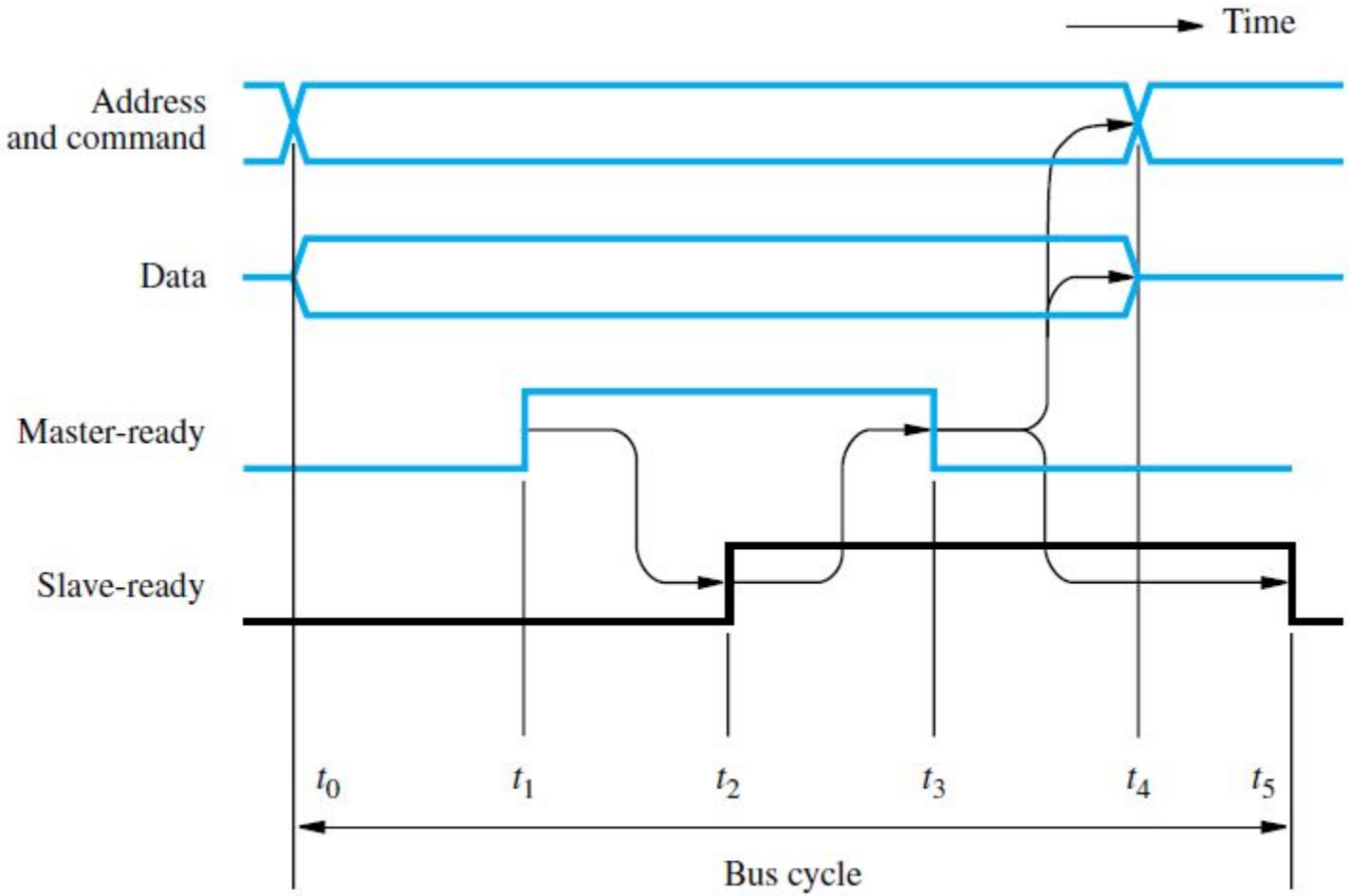


Figure 7.7 Handshake control of data transfer during an output operation.