MODULE 3: INPUT/OUTPUT ORGANIZATION

3.1 ACCESSING I/O-DEVICES

• A **single bus-structure** can be used for connecting I/O-devices to a computer (Figure 7.1).
• Each I/O device is assigned a unique set of address.
• Bus consists of 3 sets of lines to carry address, data & control signals.
• When processor places an address on address-lines, the intended-device responds to the command.
• The processor requests either a read or write-operation.
• The requested-data are transferred over the data-lines.

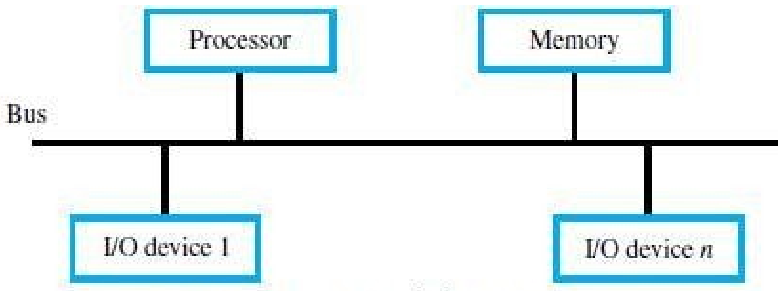


Figure 7.1 A single-bus structure.

• There are 2 ways to deal with I/O-devices: 1) Memory-mapped I/O & 2) I/O-mapped I/O.

1) Memory-Mapped I/O

➢ Memory and I/O-devices share a common address-space.
➢ Any data-transfer instruction (like Move, Load) can be used to exchange information.
➢ For example,
Move DATAIN, R0; This instruction sends the contents of location DATAIN to register R0.
Here, DATAIN → address of the input-buffer of the keyboard.

2) I/O-Mapped I/O

➢ Memory and I/0 address-spaces are different.
➢ A special instructions named **IN** and **OUT** are used for data-transfer.
➢ Advantage of separate I/O space: I/O-devices deal with fewer address-lines.

I/O Interface for an Input Device

1) **Address Decoder:** enables the device to recognize its address when this address appears on the address-lines (Figure 7.2).
2) **Status Register:** contains information relevant to operation of I/O-device.
3) **Data Register:** holds data being transferred to or from processor. There are 2 types:
   i) DATAIN → Input-buffer associated with keyboard.
   ii) DATAOUT → Output data buffer of a display/printer.

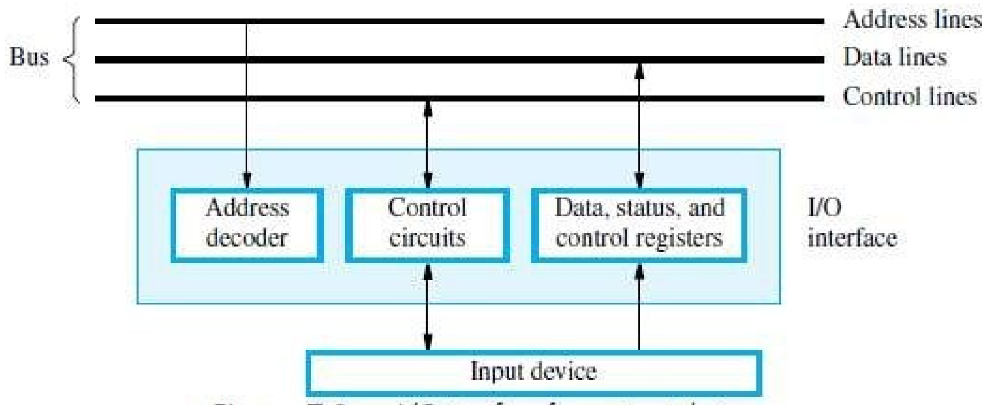


Figure 7.2 I/O interface for an input device.

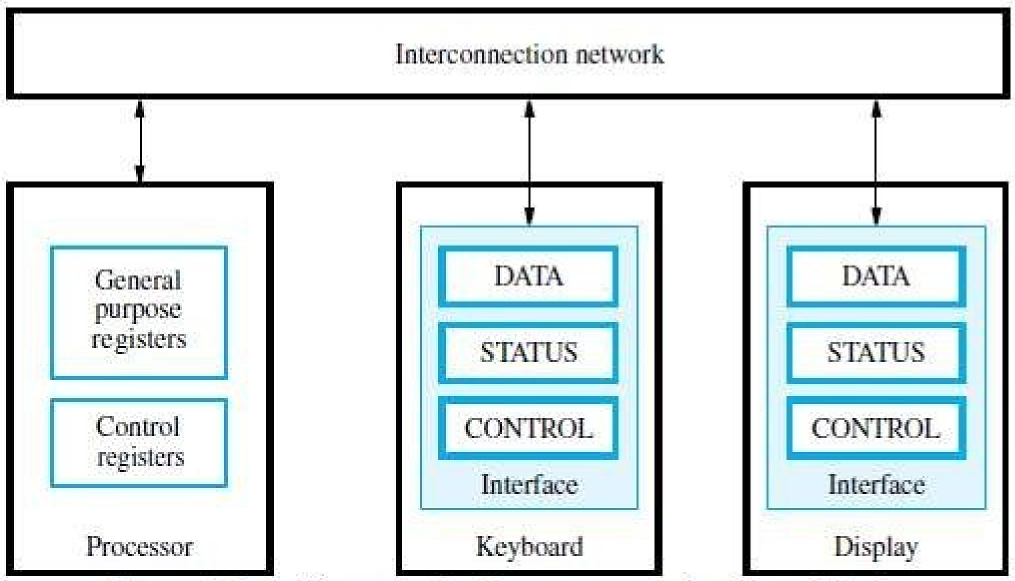


Figure 3.2 The connection for processor, keyboard, and display.

MECHANISMS USED FOR INTERFACING I/O-DEVICES

1) Program Controlled I/O

- Processor repeatedly checks status-flag to achieve required synchronization b/w processor & I/O device. (We say that the processor polls the device).
- Main drawback:

The processor wastes time in checking status of device before actual data-transfer takes place.

2) Interrupt I/O

- I/O-device initiates the action instead of the processor.
- I/O-device sends an INTR signal over bus whenever it is ready for a data-transfer operation.
- Like this, required synchronization is done between processor & I/O device.

3) Direct Memory Access (DMA)

- Device-interface transfer data directly to/from the memory w/o continuous involvement by the processor.
- DMA is a technique used for high speed I/O-device.

3.2 INTERRUPTS

- There are many situations where other tasks can be performed while waiting for an I/O device to become ready.
- A hardware signal called an Interrupt will alert the processor when an I/O device becomes ready.
- Interrupt-signal is sent on the interrupt-request line.
- The processor can be performing its own task without the need to continuously check the I/O-device.
- The routine executed in response to an interrupt-request is called ISR.
- The processor must inform the device that its request has been recognized by sending INTA signal. (INTR → Interrupt Request, INTA → Interrupt Acknowledge, ISR → Interrupt Service Routine)
- For example, consider COMPUTE and PRINT routines (Figure 3.6).

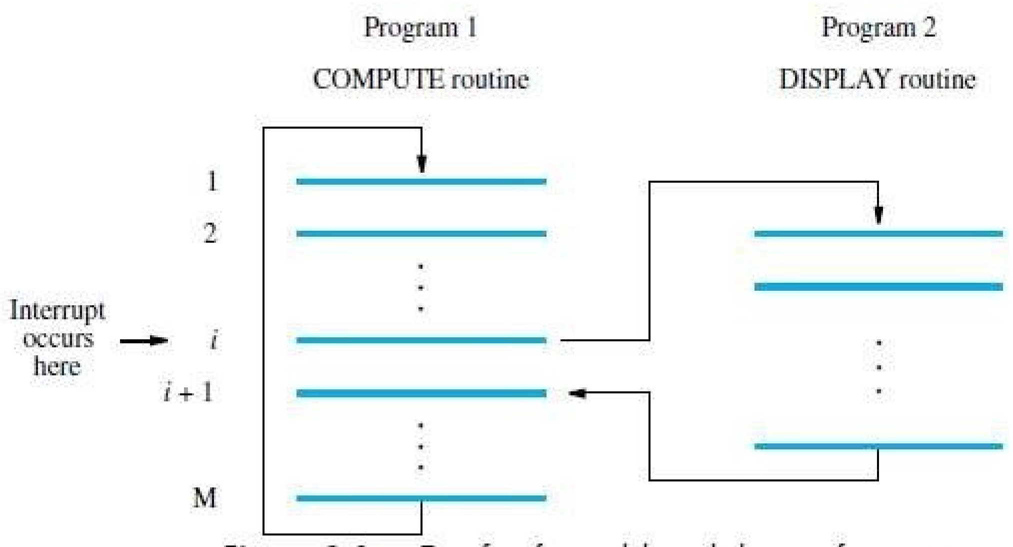


Figure 3.6 Transfer of control through the use of interrupts.

- The processor first completes the execution of instruction i.
- Then, processor loads the PC with the address of the first instruction of the ISR.
- After the execution of ISR, the processor has to come back to instruction i+1.
- Therefore, when an interrupt occurs, the current content of PC is put in temporary storage location.
- A return at the end of ISR reloads the PC from that temporary storage location.
- This causes the execution to resume at instruction i+1.
- When processor is handling interrupts, it must inform device that its request has been recognized.
- This may be accomplished by INTA signal.
- The task of saving and restoring the information can be done automatically by the processor.
- The processor saves only the contents of **PC & Status register.**
- Saving registers also increases the Interrupt Latency.
- **Interrupt Latency** is a delay between
  → time an interrupt-request is received and
  → start of the execution of the ISR.
- Generally, the long interrupt latency in unacceptable.

Difference between Subroutine & ISR

Subroutine	**ISR**
A subroutine performs a function required by the program from which it is called.	ISR may not have anything in common with program being executed at time INTR is received
Subroutine is just a linkage of 2 or more function related to each other.	Interrupt is a mechanism for coordinating I/O transfers.

3.2.1 INTERRUPT HARDWARE

- Most computers have several I/O devices that can request an interrupt.
- A single interrupt-request (IR) line may be used to serve n devices (Figure 4.6).
- All devices are connected to IR line via switches to ground.
- To request an interrupt, a device closes its associated switch.
- Thus, if all IR signals are inactive, the voltage on the IR line will be equal to V_{dd}.
- When a device requests an interrupt, the voltage on the line drops to 0.
- This causes the INTR received by the processor to go to 1.
- The value of INTR is the logical OR of the requests from individual devices.

$$INTR=INTR_1+ INTR_{2+}\ldots\ldots\ldots\ldots+INTR_n$$

- A special gates known as open-collector or open-drain are used to drive the INTR line.

- The Output of the open collector control is equal to a switch to the ground that is
  → open when gates input is in "0" state and
  → closed when the gates input is in "1" state.
- Resistor R is called a **Pull-up Resistor** because
  it pulls the line voltage up to the high-voltage state when the switches are open.

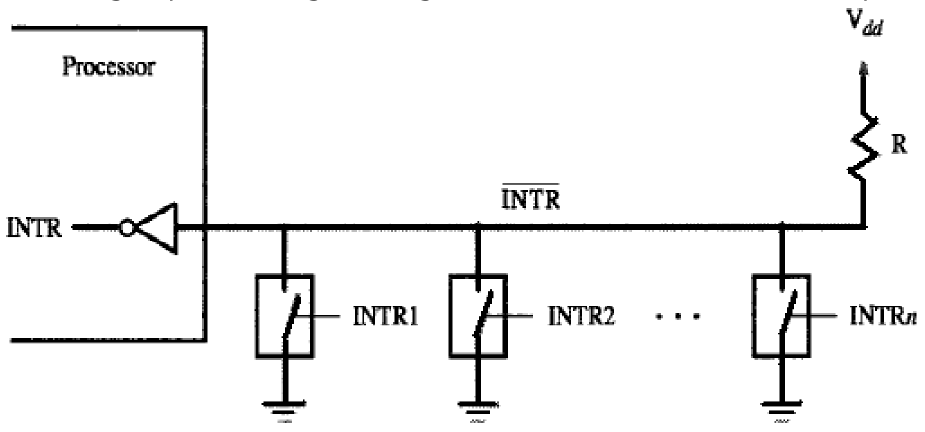


Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

3.2.2 ENABLING & DISABLING INTERRUPTS

- All computers fundamentally should be able to enable and disable interruptions as desired.
- The problem of infinite loop occurs due to successive interruptions of active INTR signals.
- There are 3 mechanisms to solve problem of infinite loop:
  1) Processor should ignore the interrupts until execution of first instruction of the ISR.
  2) Processor should automatically disable interrupts before starting the execution of the ISR.
  3) Processor has a special INTR line for which the interrupt-handling circuit.
     Interrupt-circuit responds only to leading edge of signal. Such line is called edge-triggered.
- Sequence of events involved in handling an interrupt-request:
  1) The device raises an interrupt-request.
  2) The processor interrupts the program currently being executed.
  3) Interrupts are disabled by changing the control bits in the processor status register (PS).
  4) The device is informed that its request has been recognized.
     In response, the device deactivates the interrupt-request signal.
  5) The action requested by the interrupt is performed by the interrupt-service routine.
  6) Interrupts are enabled and execution of the interrupted program is resumed.

3.3 HANDLING MULTIPLE DEVICES

- While handling multiple devices, the issues concerned are:
  1) How can the processor recognize the device requesting an interrupt?
  2) How can the processor obtain the starting address of the appropriate ISR?
  3) Should a device be allowed to interrupt the processor while another interrupt is being serviced?
  4) How should 2 or more simultaneous interrupt-requests be handled?

POLLING

- Information needed to determine whether device is requesting interrupt is available in status-register
- Following condition-codes are used:
  - ➢ DIRQ → Interrupt-request for display.
  - ➢ KIRQ → Interrupt-request for keyboard.
  - ➢ KEN → keyboard enable.

- ➢ DEN → Display Enable.
- ➢ SIN, SOUT → status flags.

• For an input device, SIN status flag in used.
SIN = 1 → when a character is entered at the keyboard.
SIN = 0 → when the character is read by processor.
IRQ=1 → when a device raises an interrupt-requests (Figure 4.3).

• Simplest way to identify interrupting-device is to have ISR poll all devices connected to bus.
• The first device encountered with its IRQ bit set is serviced.
• After servicing first device, next requests may be serviced.
• **Advantage:** Simple & easy to implement.
Disadvantage: More time spent polling IRQ bits of all devices.

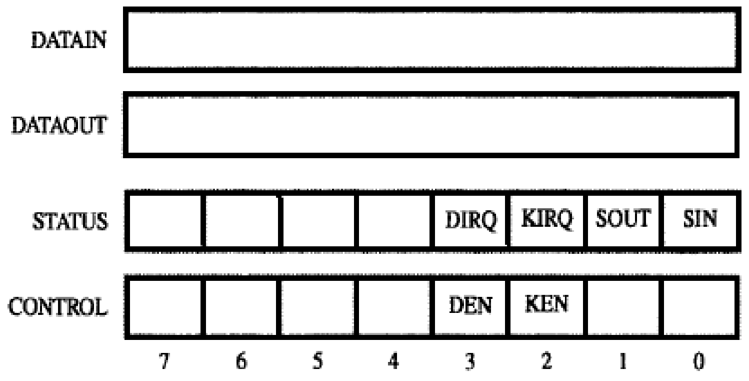


Figure 4.3 Registers in keyboard and display interfaces.

```
        Move        #LINE,R0        Initialize memory pointer.
WAITK   TestBit     #0,STATUS       Test SIN.
        Branch=0    WAITK           Wait for character to be entered.
        Move        DATAIN,R1       Read character.
WAITD   TestBit     #1,STATUS       Test SOUT.
        Branch=0    WAITD           Wait for display to become ready.
        Move        R1,DATAOUT      Send character to display.
        Move        R1,(R0)+        Store charater and advance pointer.
        Compare     #$0D,R1         Check if Carriage Return.
        Branch≠0    WAITK           If not, get another character.
        Move        #$0A,DATAOUT    Otherwise, send Line Feed.
        Call        PROCESS         Call a subroutine to process the
                                      the input line.
```

Figure 4.4 A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display.

VECTORED INTERRUPTS

- A device requesting an interrupt identifies itself by sending a special-code to processor over bus.
- Then, the processor starts executing the ISR.
- The special-code indicates starting-address of ISR.
- The special-code length ranges from 4 to 8 bits.
- The location pointed to by the interrupting-device is used to store the staring address to ISR.
- The staring address to ISR is called the **interrupt vector**.
- Processor
  → loads interrupt-vector into PC &
  → executes appropriate ISR.
- When processor is ready to receive interrupt-vector code, it activates INTA line.

- Then, I/O-device responds by sending its interrupt-vector code & turning off the INTR signal.
- The interrupt vector also includes a new value for the Processor Status Register.

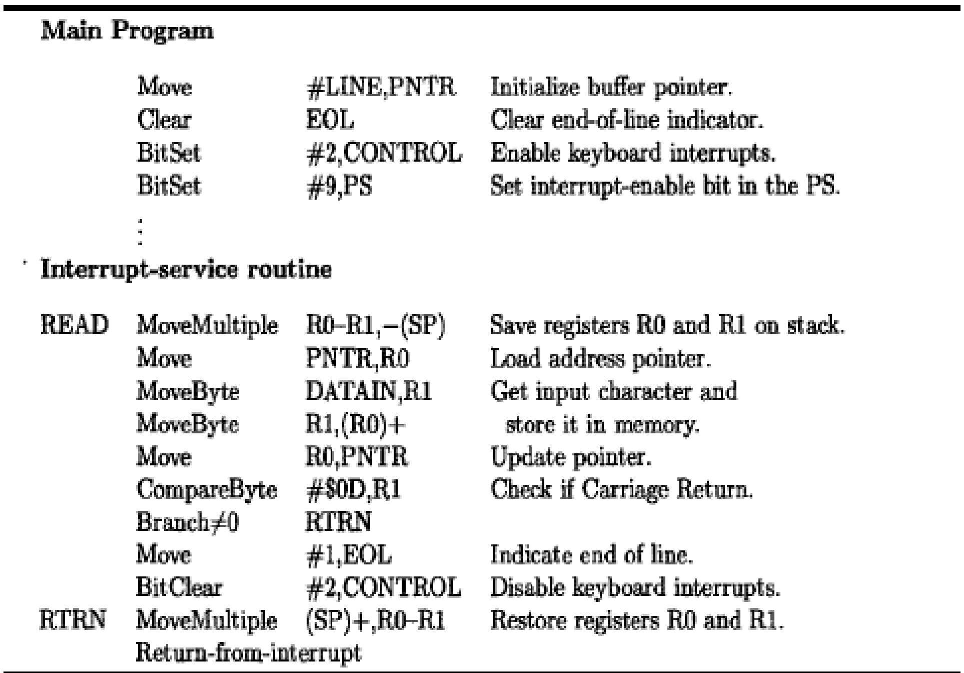
3.4 CONTROLLING DEVICE REQUESTS

- Following condition-codes are used:
  - KEN → Keyboard Interrupt Enable.
  - DEN → Display Interrupt Enable.
  - KIRQ/DIRQ → Keyboard/Display unit requesting an interrupt.
- There are 2 independent methods for controlling interrupt-requests. (IE → interrupt-enable).

1) At Device-end

IE bit in a control-register determines whether device is allowed to generate an interrupt-request.

2) At Processor-end, interrupt-request is determined by
→ IE bit in the PS register or
→ Priority structure



```
Main Program

        Move           #LINE,PNTR     Initialize buffer pointer.
        Clear          EOL            Clear end-of-line indicator.
        BitSet         #2,CONTROL     Enable keyboard interrupts.
        BitSet         #9,PS          Set interrupt-enable bit in the PS.
          :
Interrupt-service routine

READ    MoveMultiple   R0–R1,–(SP)    Save registers R0 and R1 on stack.
        Move           PNTR,R0        Load address pointer.
        MoveByte       DATAIN,R1      Get input character and
        MoveByte       R1,(R0)+         store it in memory.
        Move           R0,PNTR        Update pointer.
        CompareByte    #$0D,R1        Check if Carriage Return.
        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.

INTERRUPT NESTING

- A multiple-priority scheme is implemented by using separate INTR & INTA lines for each
  device
- Each INTR line is assigned a different priority-level (Figure 4.7).
- Priority-level of processor is the priority of program that is currently being executed.
- Processor accepts interrupts only from devices that have higher-priority than its own.
- At the time of execution of ISR for some device, priority of processor is raised to that
  of the device.
- Thus, interrupts from devices at the same level of priority or lower are disabled.

Privileged Instruction

- Processor's priority is encoded in a few bits of PS word. (PS → Processor-Status).
- Encoded-bits can be changed by **Privileged Instructions** that write into PS.
- Privileged-instructions can be executed only while processor is running in **Supervisor Mode**.
- Processor is in supervisor-mode only when executing operating-system routines.

Privileged Exception

- User program cannot
  → accidently or intentionally change the priority of the processor &
  → disrupt the system-operation.
- An attempt to execute a privileged-instruction while in user-mode leads to a **Privileged Exception**.

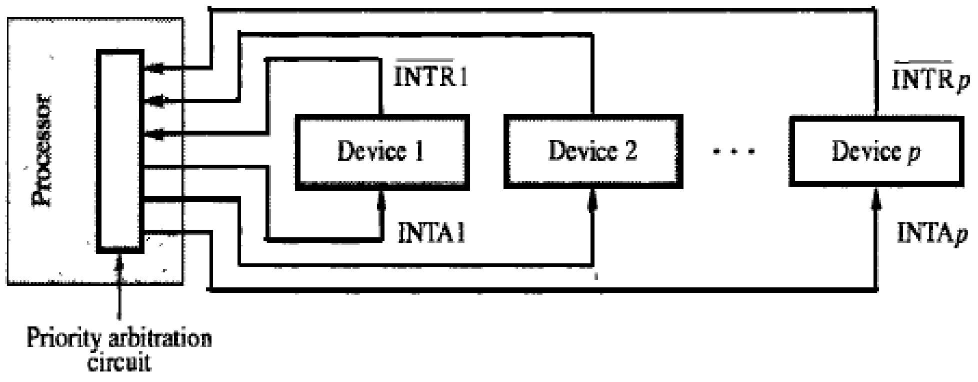

Processor
$\overline{\text{INTR}}1$
$\overline{\text{INTR}}p$
Device 1
Device 2
Device p
INTA1
INTAp
Priority arbitration circuit

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

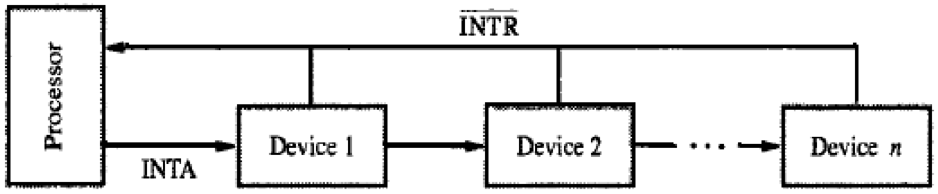
SIMULTANEOUS REQUESTS

- The processor must have some mechanisms to decide which request to service when simultaneous requests arrive.
- INTR line is common to all devices (Figure 4.8a).
- INTA line is connected in a daisy-chain fashion.
- INTA signal propagates serially through devices.
- When several devices raise an interrupt-request, INTR line is activated.
- Processor responds by setting INTA line to 1. This signal is received by device 1.
- Device-1 passes signal on to device 2 only if it does not require any service.

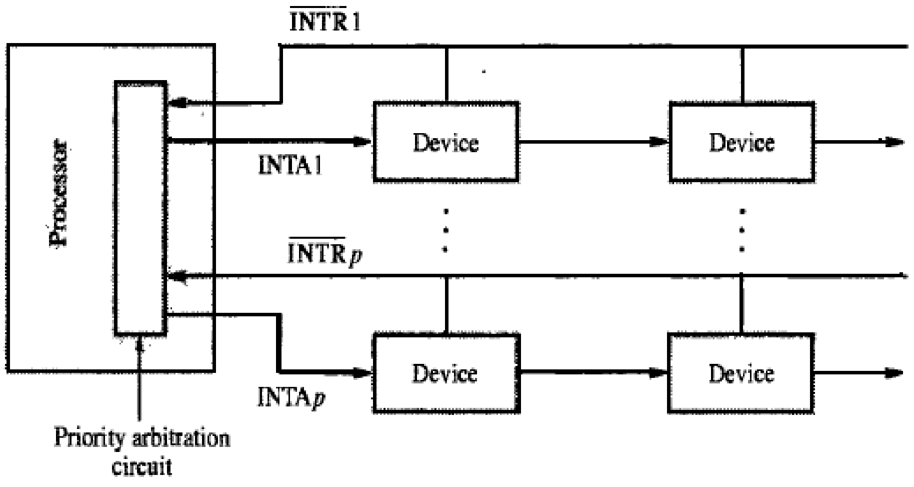
- If device-1 has a pending-request for interrupt, the device-1
  → blocks INTA signal &
  → proceeds to put its identifying-code on data-lines.
- Device that is electrically closest to processor has highest priority.
- **Advantage:** It requires fewer wires than the individual connections.

Arrangement of Priority Groups

- Here, the devices are organized in groups & each group is connected at a different priority level.
- Within a group, devices are connected in a daisy chain. (Figure 4.8b).



(a) Daisy chain



(b) Arrangement of priority groups

Figure 4.8 Interrupt priority schemes.

EXCEPTIONS

- An **interrupt** is an event that causes
  → execution of one program to be suspended &
  → execution of another program to begin.
- **Exception** refers to any event that causes an interruption. For ex: I/O interrupts.

1. Recovery from Errors

- These are techniques to ensure that all hardware components are operating properly.
- For ex: Many computers include an ECC in memory which allows detection of errors in stored-data. (ECC → Error Checking Code, ESR → Exception Service Routine).

- If an error occurs, control-hardware
  → detects the errors &
  → informs processor by raising an interrupt.
- When exception processing is initiated (as a result of errors), processor.
  → suspends program being executed &
  → starts an ESR. This routine takes appropriate action to recover from the error.

2. Debugging

- Debugger
  → is used to find errors in a program and
  → uses exceptions to provide 2 important facilities: i) Trace & ii) Breakpoints

i) Trace

- When a processor is operating in trace-mode, an exception occurs after execution of every instruction (using debugging-program as ESR).
- Debugging-program enables user to examine contents of registers, memory-locations and so on.
- On return from debugging-program, next instruction in program being debugged is executed, then debugging-program is activated again.
- The trace exception is disabled during the execution of the debugging-program.

ii) Breakpoints

- Here, the program being debugged is interrupted only at specific points selected by user.
- An instruction called Trap (or Software interrupt) is usually provided for this purpose.
- When program is executed & reaches breakpoint, the user can examine memory & register contents.

3. Privilege Exception

- To protect OS from being corrupted by user-programs, **Privileged Instructions** are executed only while processor is in supervisor-mode.
- For e.g.

When processor runs in user-mode, it will not execute instruction that change priority of processor.

- An attempt to execute privileged-instruction will produce a **Privilege Exception**.
- As a result, processor switches to supervisor-mode & begins to execute an appropriate routine in OS.

3.5 DIRECT MEMORY ACCESS (DMA)

- The transfer of a block of data directly b/w an external device & main-memory w/o continuous involvement by processor is called DMA.
- DMA controller
  → is a control circuit that performs DMA transfers (Figure 8.13).
  → is a part of the I/O device interface.
  → performs the functions that would normally be carried out by processor.
- While a DMA transfer is taking place, the processor can be used to execute another

program.

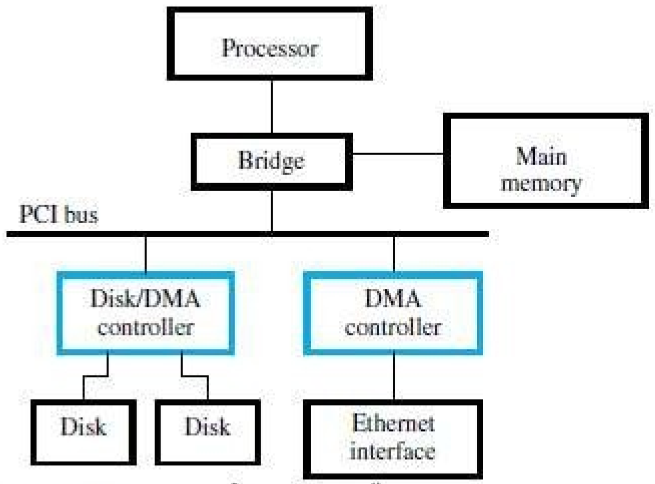


Figure 8.13 Use of DMA controllers in a computer system.

• DMA interface has three registers (Figure 8.12):
  1) First register is used for storing starting-address.
  2) Second register is used for storing word-count.
  3) Third register contains status- & control-flags.

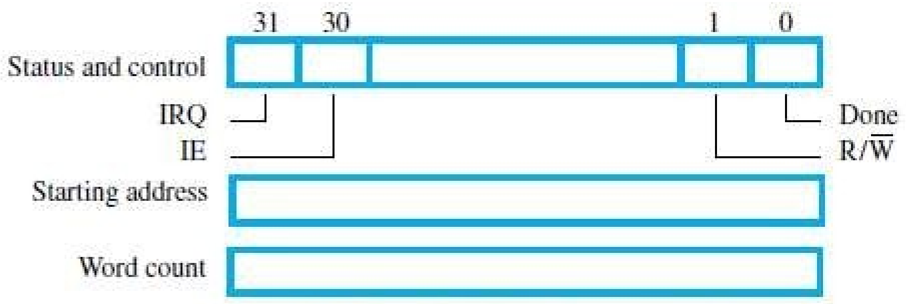


Figure 8.12 Typical registers in a DMA controller.

- The R/W bit determines direction of transfer.
  If R/W=1, controller performs a read-operation (i.e. it transfers data from memory to I/O), Otherwise, controller performs a write-operation (i.e. it transfers data from I/O to memory).
- If Done=1, the controller
  → has completed transferring a block of data and
  → is ready to receive another command. (IE → Interrupt Enable).
- If IE=1, controller raises an interrupt after it has completed transferring a block of data.
- If IRQ=1, controller requests an interrupt.
- Requests by DMA devices for using the bus are always given higher priority than processor requests.
- There are 2 ways in which the DMA operation can be carried out:
  1) Processor originates most memory-access cycles.
  ➢ DMA controller is said to "steal" memory cycles from processor.
  ➢ Hence, this technique is usually called **Cycle Stealing**.
  2) DMA controller is given exclusive access to main-memory to transfer a block of data without any interruption. This is known as **Block Mode** (or burst mode).

BUS ARBITRATION

- The device that is allowed to initiate data-transfers on bus at any given time is called **bus-master**.
- There can be only one bus-master at any given time.
- **Bus Arbitration** is the process by which
  → next device to become the bus-master is selected &
  → bus-mastership is transferred to that device.
- The two approaches are:
  1) **Centralized Arbitration:** A single bus-arbiter performs the required arbitration.
  2) **Distributed Arbitration:** All devices participate in selection of next bus-master.
- 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 this, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers.
- The bus arbiter may be the processor or a separate unit connected to the bus.

CENTRALIZED ARBITRATION

- A single bus-arbiter performs the required arbitration (Figure: 4.20).
- Normally, processor is the bus-master.
- Processor may grant bus-mastership to one of the DMA controllers.
- A DMA controller indicates that it needs to become bus-master by activating BR line.
- The signal on the BR line is the logical OR of bus-requests from all devices connected to it.
- Then, processor activates BG1 signal indicating to DMA controllers to use bus when it becomes free.
- BG1 signal is connected to all DMA controllers using a daisy-chain arrangement.
- If DMA controller-1 is requesting the bus,
  Then, DMA controller-1 blocks propagation of grant-signal to other devices. Otherwise, DMA controller-1 passes the grant downstream by asserting BG2.
- Current bus-master indicates to all devices that it is using bus by activating BBSY line.
- The bus-arbiter is used to coordinate the activities of all devices requesting memory transfers.
- Arbiter ensures that only 1 request is granted at any given time according to a priority scheme. (BR → Bus-Request, BG → Bus-Grant, BBSY → Bus Busy).

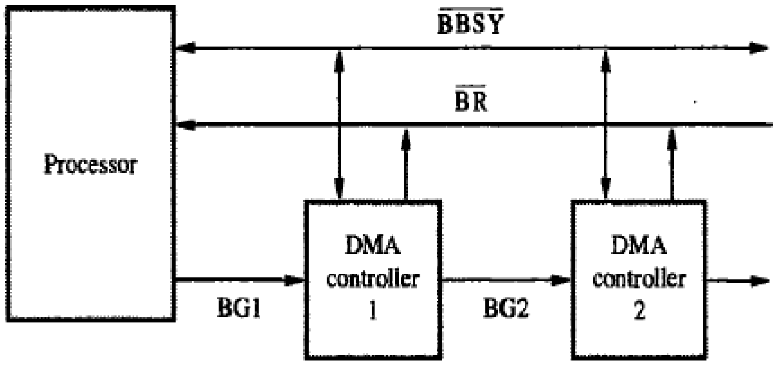


Figure 4.20 A simple arrangement for bus arbitration using a daisy chain.

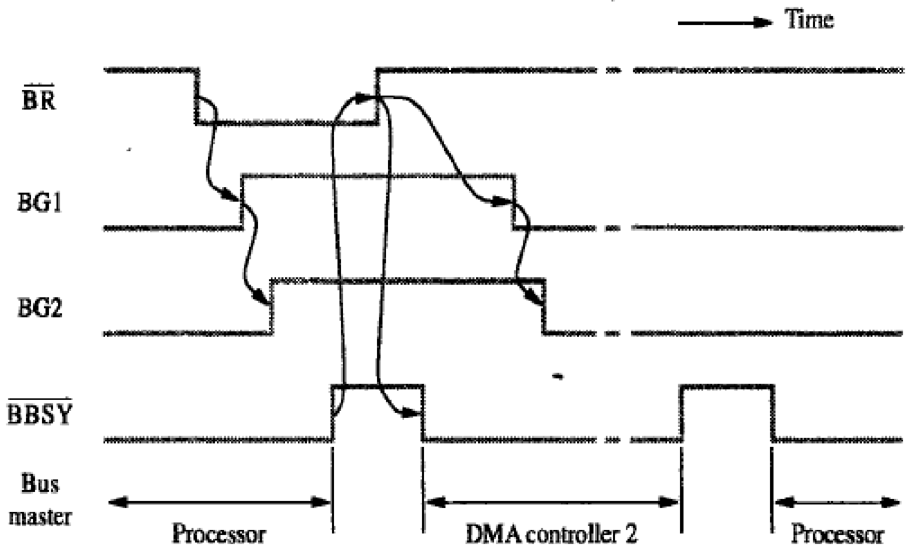


Figure 4.21 Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

- The timing diagram shows the sequence of events for the devices connected to the processor.
- DMA controller-2
  → requests and acquires bus-mastership and
  → later releases the bus. (Figure: 4.21).
- After DMA controller-2 releases the bus, the processor resources bus-mastership.

DISTRIBUTED ARBITRATION

- All device participate in the selection of next bus-master (Figure 4.22).
- Each device on bus is assigned a 4-bit identification number (ID).
- When 1 or more devices request bus, they
  → assert Start-Arbitration signal &
  → place their 4-bit ID numbers on four open-collector lines $\overline{ARB0}$ through $\overline{ARB3}$.

- A winner is selected as a result of interaction among signals transmitted over these lines.
- Net-outcome is that the code on 4 lines represents request that has the highest ID number.
- **Advantage:**
  This approach offers higher reliability since operation of bus is not dependent on any single device.

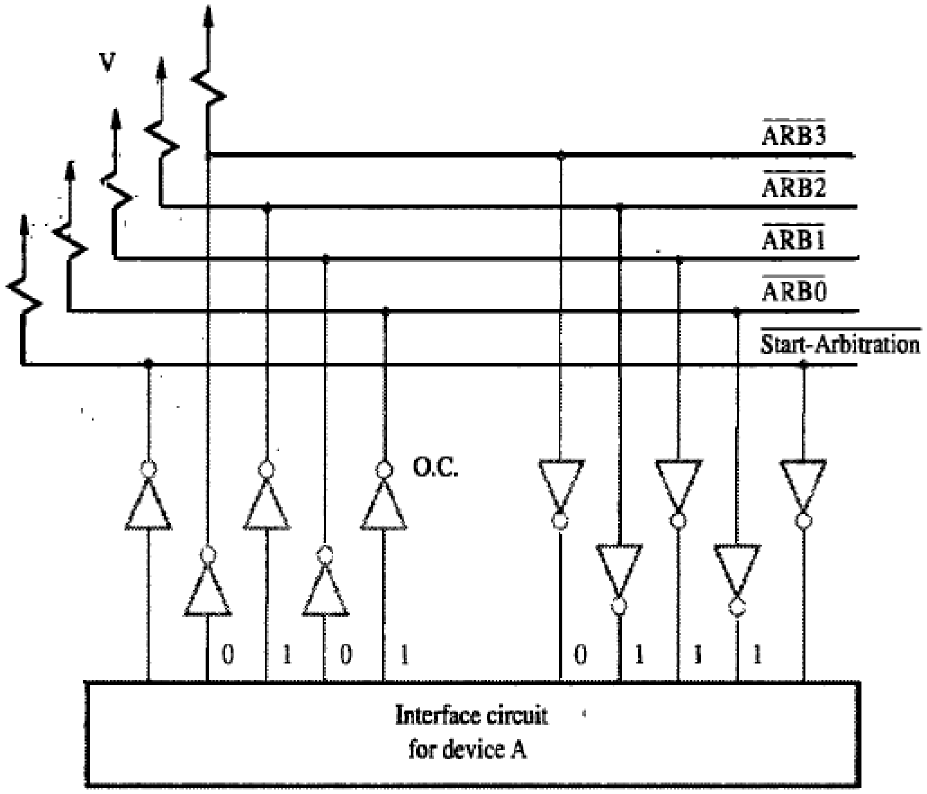


Figure 4.22 A distributed arbitration scheme.

For example:

- Assume 2 devices A & B have their ID 5 (0101), 6 (0110) and their code is 0111.
- Each device compares the pattern on the arbitration line to its own ID starting
  from MSB.
- If the device detects a difference at any bit position, it disables the drivers at
  that bit position.
- Driver is disabled by placing ”0” at the input of the driver.
- In e.g. “A” detects a difference in line ARB1, hence it disables the drivers on
  lines ARB1 & ARB0.

This causes pattern on arbitration-line to change to 0110. This means that “B” has won contention.

Problem 1:

The input status bit in an interface-circuit is cleared as soon as the input data register is read. Why is this important?

Solution:

After reading the input data, it is necessary to clear the input status flag before the program begins a new read-operation. Otherwise, the same input data would be read a second time.

Problem 2:

What is the difference between a subroutine and an interrupt-service routine?

Solution:

A subroutine is called by a program instruction to perform a function needed by the calling program.

An interrupt-service routine is initiated by an event such as an input operation or a hardware error. The function it performs may not be at all related to the program being executed at the time of interruption. Hence, it must not affect any of the data or status information relating to that program.

Problem 3:

Three devices A, B, & C are connected to the bus of a computer. I/O transfers for all 3 devices use interrupt control. Interrupt nesting for devices A & B is not allowed, but interrupt-requests from C may be accepted while either A or B is being serviced. Suggest different ways in which this can be accomplished in each of the following cases:

(a) The computer has one interrupt-request line.

(b) Two interrupt-request lines INTR1 & INTR2 are available, with INTR1 having higher priority. Specify when and how interrupts are enabled and disabled in each case.

Solution:

(a) Interrupts should be enabled, except when C is being serviced. The nesting rules can be enforced by manipulating the interrupt-enable flags in the interfaces of A and B.

(b) A and B should be connected to INTR , and C to INTR. When an interrupt-request is received from either A or B, interrupts from the other device will be automatically disabled until the request has been serviced. However, interrupt-requests from C will always be accepted.

Problem 4:

Consider a computer in which several devices are connected to a common interrupt-request line. Explain how you would arrange for interrupts from device j to be accepted before the execution of the interrupt service routine for device i is completed. Comment in particular on the times at which interrupts must be enabled and disabled at various points in the system.

Solution:

Interrupts are disabled before the interrupt-service routine is entered. Once device i turns off its interrupt-request, interrupts may be safely enabled in the processor. If the interface-circuit of device i turns off its interrupt-request when it receives the interrupt acknowledge signal, interrupts may be enabled at the beginning of the interrupt-service routine of device i. Otherwise, interrupts may be enabled only after the instruction that causes device i to turn off its interrupt-request has been executed.

Problem 5:

Consider the daisy chain arrangement. Assume that after a device generates an interrupt-request, it turns off that request as soon as it receives the interrupt acknowledge signal. Is it still necessary to disable interrupts in the processor before entering the interrupt service routine? Why?

Solution:

Yes, because other devices may keep the interrupt-request line asserted.