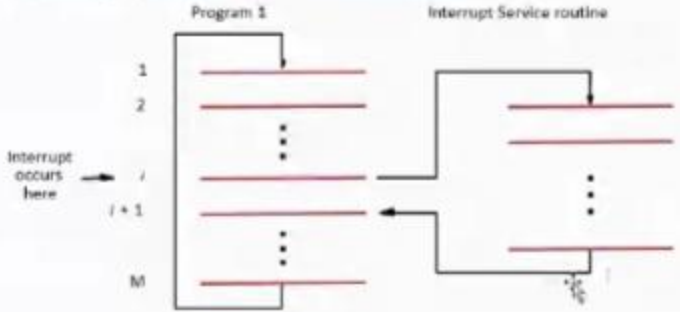
Interrupts (contd..)



- *Processor is executing the instruction located at address i when an interrupt occurs.*
- *Routine executed in response to an interrupt request is called the interrupt-service routine.*
- *When an interrupt occurs, control must be transferred to the interrupt service routine.*
- *But before transferring control, the current contents of the PC (i+1), must be saved in a known location.*
- *This will enable the return-from-interrupt instruction to resume execution at i+1.*
- *Return address, or the contents of the PC are usually stored on the processor stack.*

$INTR = INTR_1 + INTR_2 + \ldots + INTR_n$

- 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.

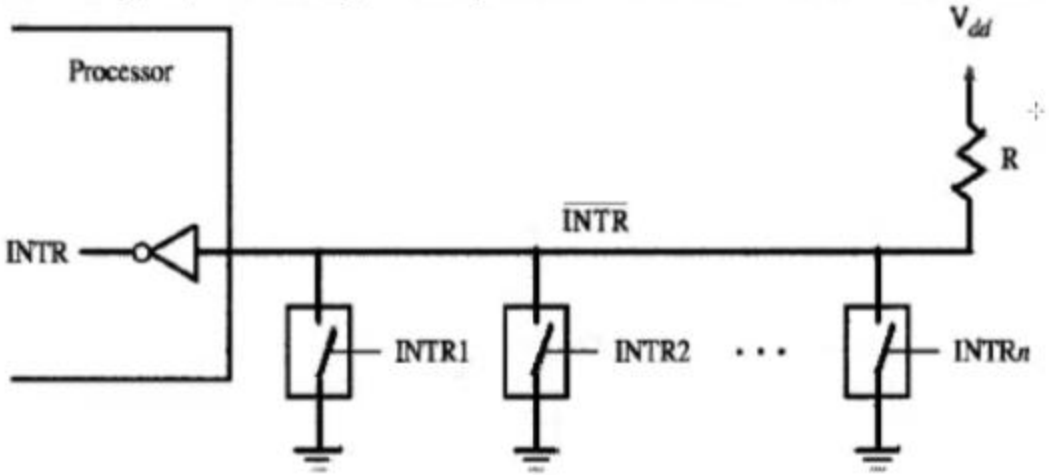


Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

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.

Interrupts (contd..)

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed.
  - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as *Interrupt-enable* and *Interrupt-disable* for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
  - First instruction of an interrupt service routine can be Interrupt-disable.
  - Last instruction of an interrupt service routine can be Interrupt-enable.

Interrupts (contd..)

- Saving and restoring information can be done automatically by the processor or explicitly by program instructions.
- Saving and restoring registers involves memory transfers:
  - Increases the total execution time.
  - 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.
- In order to reduce the interrupt latency, most processors save only the minimal amount of information:
  - This minimal amount of information includes Program Counter and processor status registers.
- Any additional information that must be saved, must be saved explicitly by the program instructions at the beginning of the interrupt service routine.

Interrupts (contd..)

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed.
  - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as *Interrupt-enable* and *Interrupt-disable* for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
  - First instruction of an interrupt service routine can be Interrupt-disable.
  - Last instruction of an interrupt service routine can be Interrupt-enable.

Interrupts (contd..)

- Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.
  - Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?
- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed?
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?
- If two interrupt-requests are received simultaneously, then how to break the tie?

COMPUTER ORGANIZATION

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 ent
  - SIN = 0 → when the character is read by processor

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

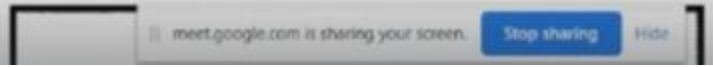
- Information needed to determine whether device is requesting interrupt is available in status-register
- Following condition-codes are used:
  - DIRQ → Interrupt-request for display.
  - KIRQ → Interrupt-request for keyboard.
  - KEN → keyboard enable.
  - DEN → Display Enable.
  - SIN, SOUT → status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

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.



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.

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

INTERRUPT NESTING

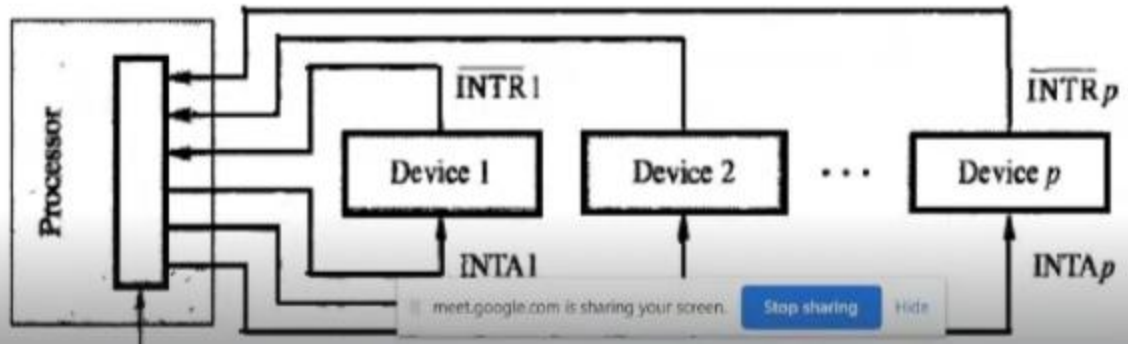
- A multiple-priority scheme is implemented by using separate INTR & INTA lines for each device
- Each INTR line is assigned a different priority-level (Figure 4.7).
- Priority-level of processor is the priority of program that is currently being executed.
- Processor accepts interrupts only from devices 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**.



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 instru[illegible] program being debugged is executed