Computer Organization & Architecture

7-3 Interrupt Concepts & Processing

Guohua Wang

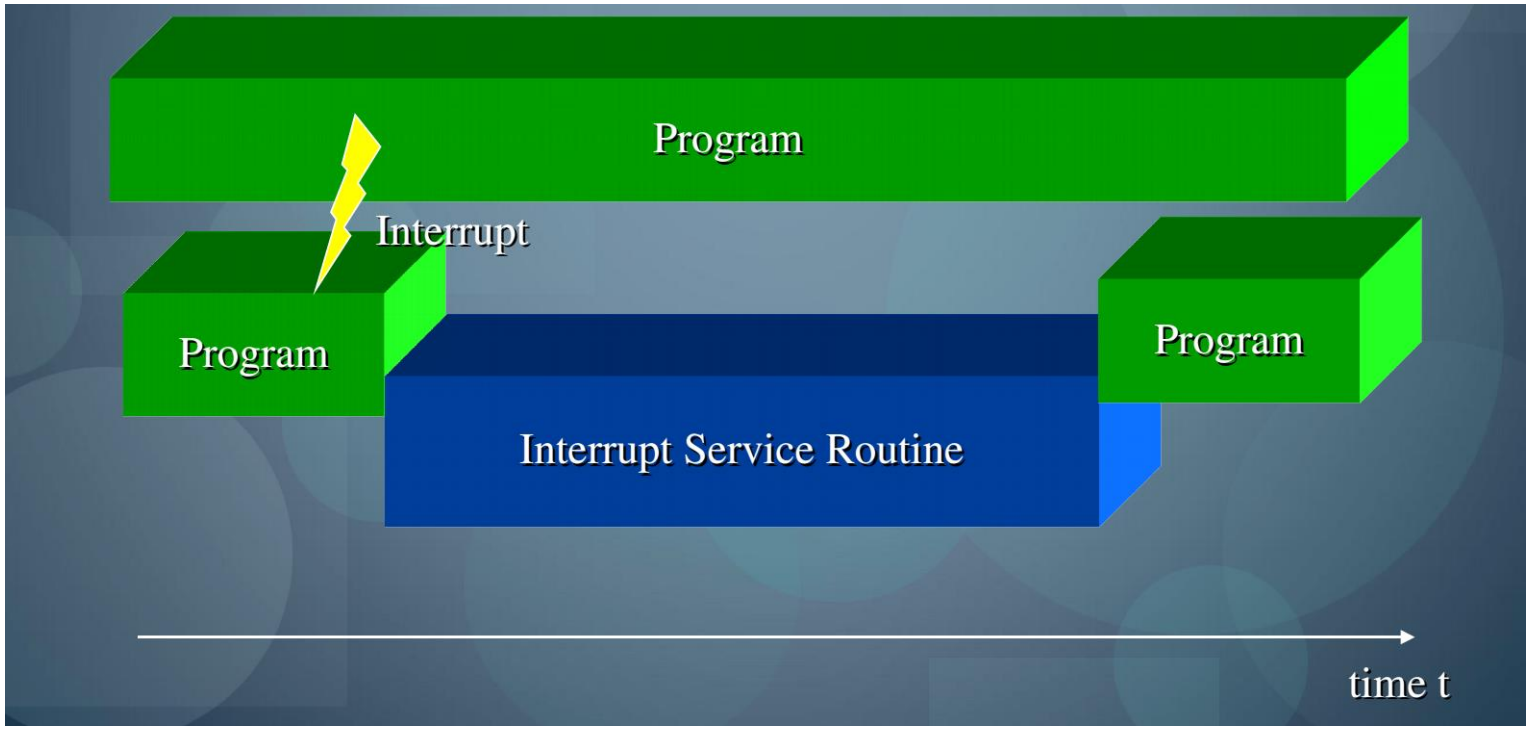
School of Software Engineering

Contents of this lecture

- What is An Interrupt
- Interrupt Example
- Advantages of Interrupt-Driven I/O
- Concepts of Interrupt
- Interrupt Processing
- Enabling and Disabling Interrupts

What is An Interrupt

- An interrupt is an event that causes the processor to stop its current program execution and switch to performing an interrupt service routine.



Interrupt Example (1)

- Example 3.1
  - Consider a task that requires continuous extensive computations to be performed and the results to be displayed on a display device.
  - The displayed results must be updated every ten seconds.
    - The 10- second intervals can be determined by a simple timer circuit, which generates an appropriate signal.
    - The processor treats the timer circuit as an input device that produces a signal that can be interrogated.
    - The timer circuit raise an interrupt request once every 10 seconds. In response, the processor displays the latest results.

Interrupt Example (2)

- Example 3.1(ctd.)
  - The task can be implemented with a program that consists of two routines, COMPUTE and DISPLAY.
    - The processor continuously executes the COMPUTE routine.
    - When it receives an interrupt request from the timer, it suspends the execution of the COMPUTE routine and executes the DISPLAY routine which sends the latest results to the display device.
    - Upon completion of the DISPLAY routine, the processor resumes the execution of the COMPUTE routine.

Since the time needed to send the results to the display device is very small compared to the 10-second interval, the processor in effect spends almost all of its time executing the COMPUTE routine.

Interrupt Example (3)

- Example 3.1 (ctd.)

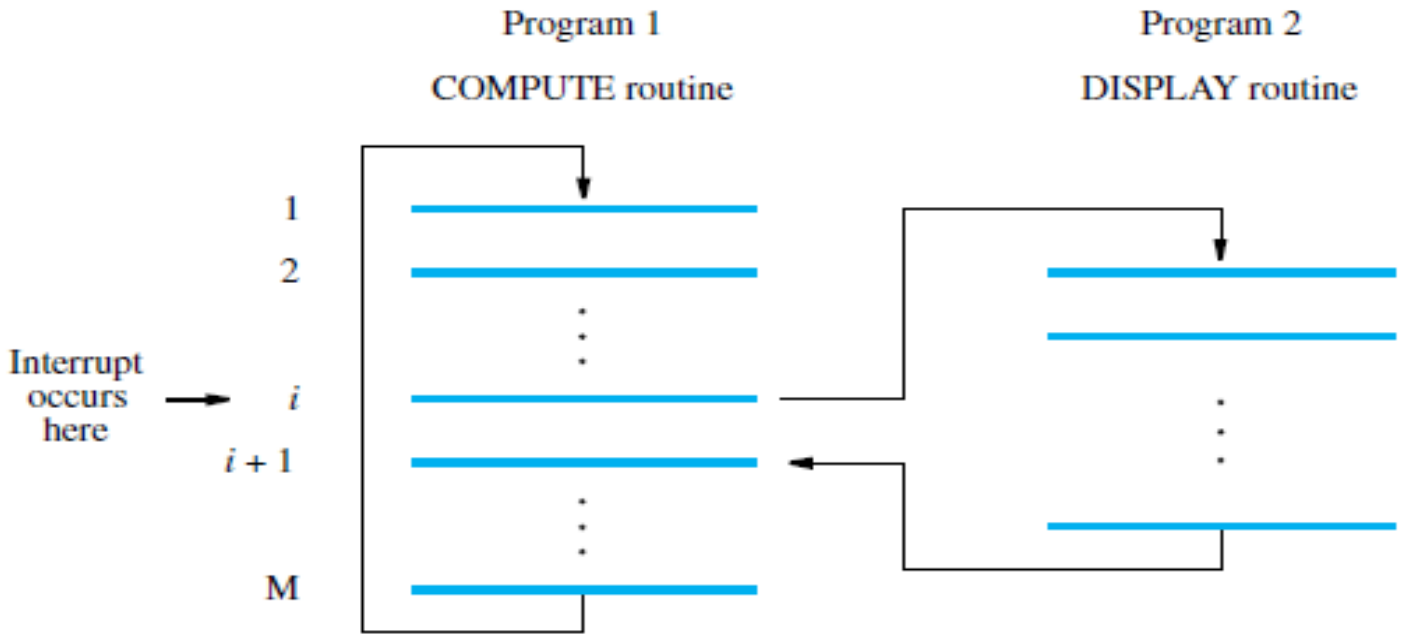


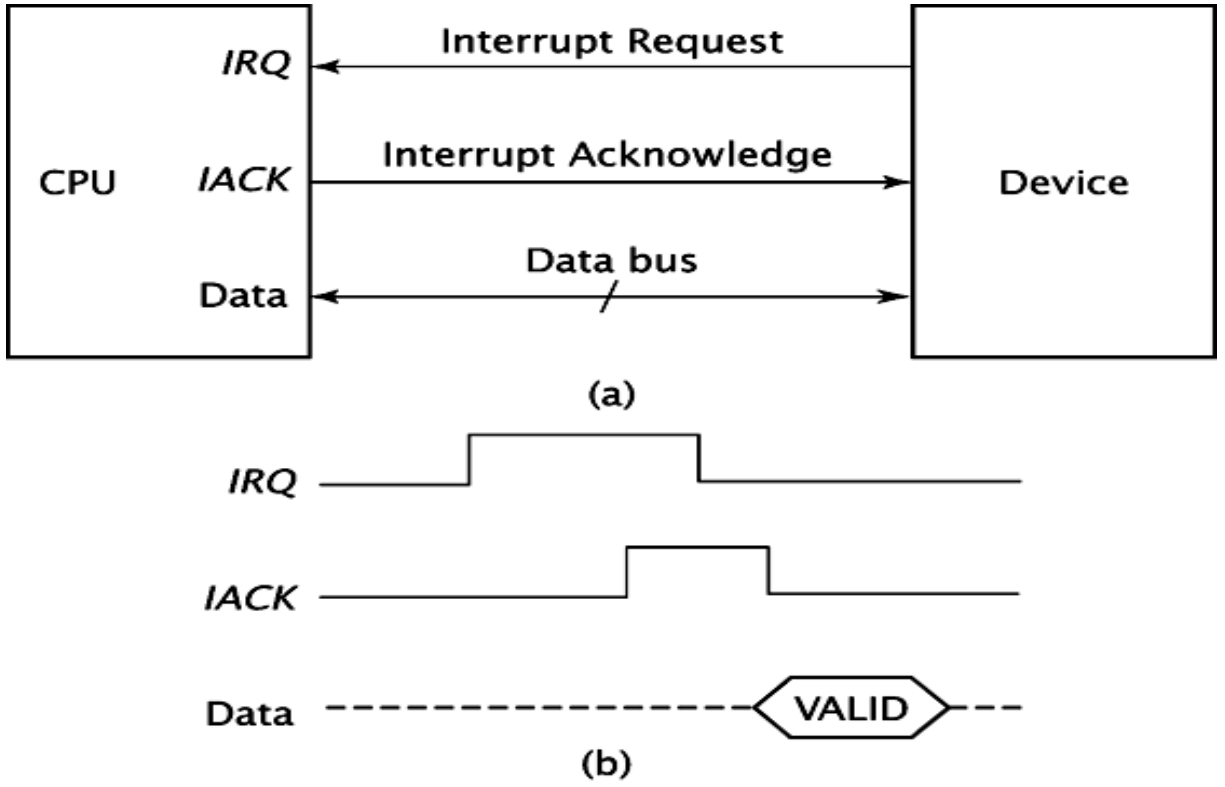
Figure 3.6 Transfer of control through the use of interrupts.

Advantages of Interrupt-Driven I/O

- No repeated CPU checking of device
  - In the examples of Figures 3.4 and 3.5, the program enters a wait loop in which it repeatedly tests the device status.
  - During this period, the processor is not performing any useful computation.
- Overcomes CPU waiting
- I/O module interrupts when ready
  - By sending a hardware signal called *interrupt request* to the processor

Concepts of Interrupt (1)

- Interrupt Request
  - A signal that an I/O device sends to the processor through one of the bus control lines.
- Interrupt Acknowledge
  - The CPU issue the signal to acknowledges the interrupt.



Concepts of Interrupt (2)

- Interrupt-Service Routine (Interrupt Handler)
  - The routine executed in response to an interrupt request is called the interrupt-service routine.
- Interrupt Latency
  - The delay between the time an interrupt request is received and the start of execution of the interrupt-service routine.

Concepts of Interrupt (3)

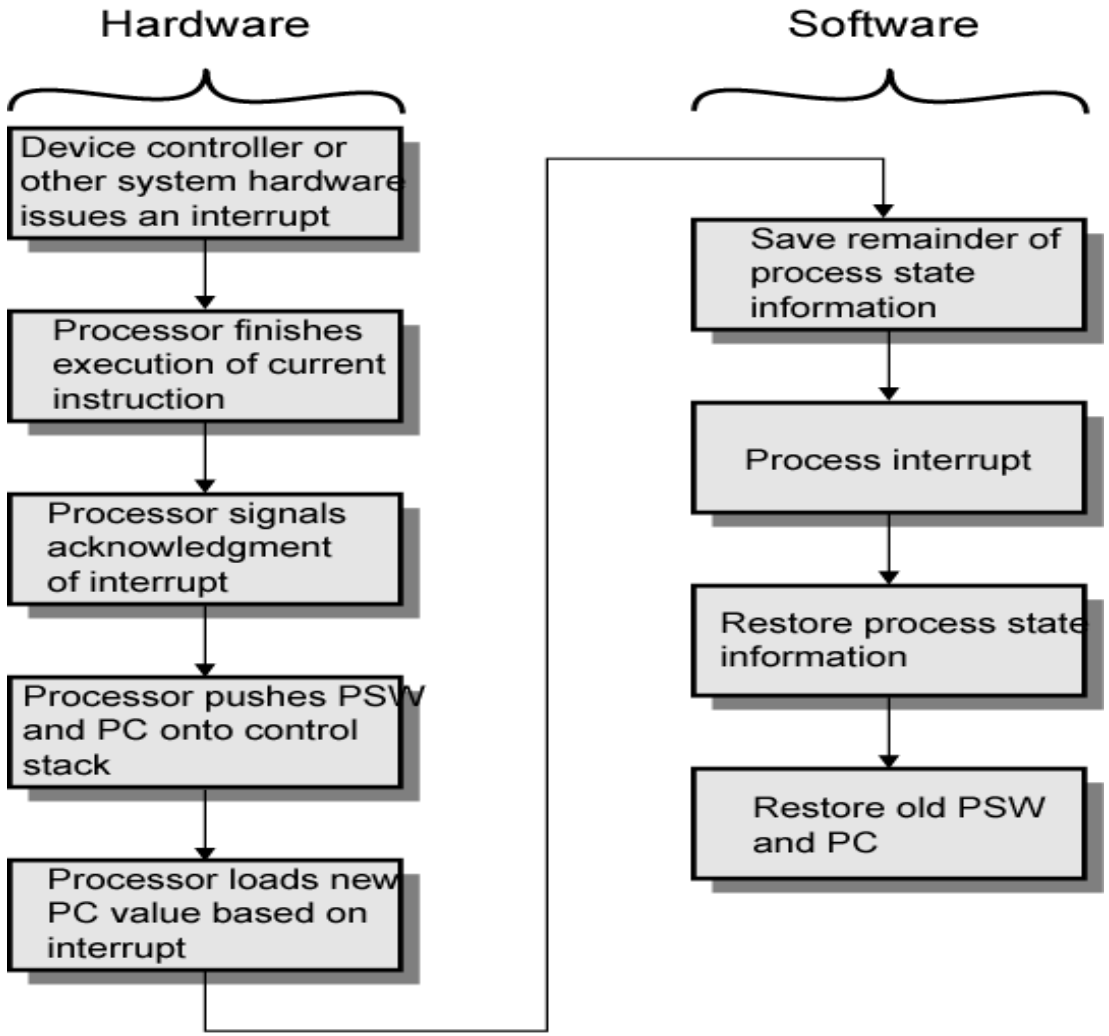
- Difference between subroutine and interrupt-service routine
  - A subroutine performs a function required by the program from which it is called. As such, potential changes to status information and contents of registers are anticipated.
  - An interrupt-service routine may not have any relation to the portion of the program being executed at the time the interrupt request is received. Therefore, before starting execution of the interrupt service routine, status information and contents of processor registers that may be altered in unanticipated ways during the execution of that routine must be saved. This saved information must be restored before execution of the interrupted program is resumed.

Interrupt Processing (1)

- Interrupt Processing
  - When an interrupt occurs (and is accepted), the execution of the current program is suspended.
  - Must save PC, Registers in Process Control Block (PCB).
  - Interrupt service routine executes to service the interrupt.

Interrupt Processing (2)

- Interrupt Processing Flowchart



Interrupt Processing (3)

- Details of Interrupt Processing
  - Processor Acknowledgement (part of handling interrupt)
    - Method 1: Processor issues interrupt-acknowledge signal.
    - Method 2: The execution of an instruction in the interrupt-service routine that access a status or data register in the device interface implicitly informs the device that its interrupt request has been recognized.

Interrupt Processing (4)

- Details of Interrupt Processing (ctd.)
  - Save and Restore Information
    - The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine.
    - The task of saving and restoring information can be done automatically by the processor or by program instructions.

Interrupt Processing (5)

- Details of Interrupt Processing (ctd.)
  - Save and Restore Information (ctd.)
    - Most modern processors save only the minimum amount of information needed to maintain the integrity of program execution. (Typically the contents of PC and PSW)
    - Any additional information that needs to be saved must be saved by program instructions at the beginning of the interrupt-service routine and restored at the end of the routine.

Enabling and Disabling Interrupts (1)

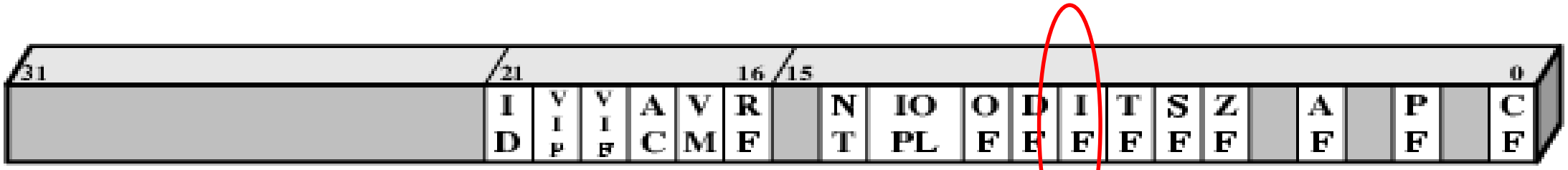
- Enabling and disabling interrupts are fundamental to all computers
  - The interruption of program execution must be carefully controlled because interrupts can arrive at any time, they may alter the sequence of events from that envisaged by the programmer.
  - It may be necessary to guarantee that a particular sequence of instructions is executed to the end without interruption because the interrupt service routine may change some of the data used by the instructions in question.

Enabling and Disabling Interrupts (2)

- Interrupt Control at Processor End
  - Set Interrupt-enable bit in the Program Status register
    - When IE=1, interrupt requests from I/O devices are accepted and serviced by the processor
    - When IE=0, the processor simply ignores all interrupt requests from I/O devices
      - Have the processor automatically disable interrupts (clear the Interrupt-enable bit) before starting the execution of the interrupt-service routine. (IE=0)
      - When a Return-from-interrupt instruction is executed, the contents of the PS are restored from the stack, setting the Interrupt-enable bit back to 1. (IE=1)
  - Suitable for a simple processor with only one interrupt-request line.

Enabling and Disabling Interrupts (3)

- Interrupt Control at Processor End (ctd.)
  - Example: Interrupt enable flag of x86 architecture



ID = Identification flag
VIP = Virtual interrupt pending
VIF = Virtual interrupt flag
AC = Alignment check
VM = Virtual 8086 mode
RF = Resume flag
NT = Nested task flag
IOPL = I/O privilege level
OF = Overflow flag

DF = Direction flag
IF = Interrupt enable flag
TF = Trap flag
SF = Sign flag
ZF = Zero flag
AF = Auxiliary carry flag
PF = Parity flag
CF = Carry flag

Enabling and Disabling Interrupts (4)

- Interrupt Control at I/O Device End
  - 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.
  - An interrupt-enable bit in a control register determines whether the device is allowed to generate an interrupt request.

Enabling and Disabling Interrupts (5)

- Interrupt Control at I/O Device End (ctd.)

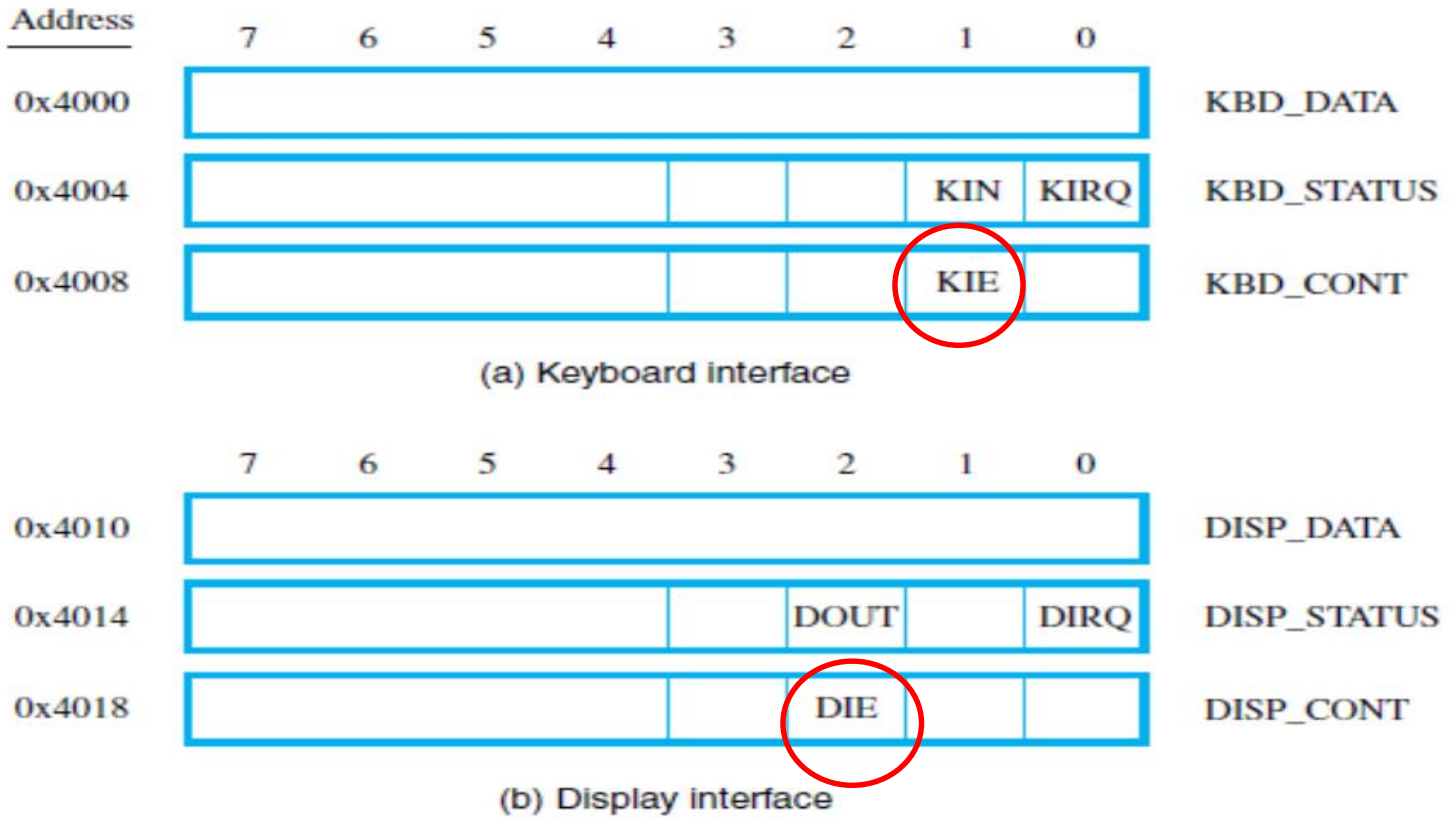


Figure 3.3 Registers in the keyboard and display interfaces.

Quiz (1)

1. In an interrupt process, the usage of saving PC is ____.

   A. to make CPU find the entry address of the interrupt service routine

   B. to continue from the program breakpoint when returning from interrupt

   C. to make CPU and peripherals working in parallel

   D. to enable interrupt nesting

Quiz (2)

2. What is interrupt? What advantage does Interrupt-Driven I/O have over Program-Controlled I/O?

Solution:

An interrupt is an event that causes the processor to stop its current program execution and switch to performing an interrupt service routine.

Interrupt-Driven I/O allows the computer to process other tasks while waiting for I/O.