UNIT - 3

Basic Input/Output

Basic Input/Output

Accessing I/O Devices

- A simple arrangement to connect I/O devices to a computer is to use a interconnection network
- **The components of a computer system communicate with each other through an interconnection network, as shown in Figure**

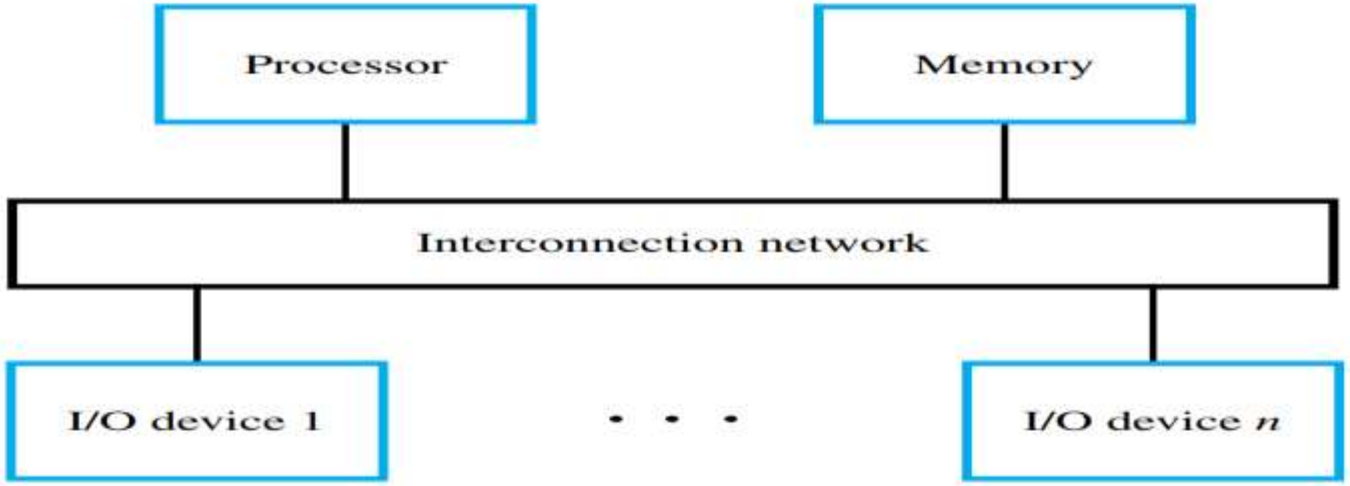


Figure 3.1 A computer system.

- **The interconnection network consists of circuits needed to transfer information between the processor, the memory unit, and a number of I/O devices.**
- Each I/O device is assigned a **unique set of addresses**. When the processor places a particular address on interconnection network

- **Load and Store instructions use addressing modes to generate effective addresses that identify the desired locations.**
- Some **addresses in the address space of the processor** are assigned to these I/O locations, rather than to the main memory.
- These **locations** are usually implemented as bit storage circuits (flip-flops) organized in the form of registers. It is customary to refer to them as I/O registers.
- Since the I/O devices and the memory share the same address space, this arrangement is called memory-mapped I/O. It is used in most computers.
- With 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.

I/O Device Interface

- **An I/O device is connected to the interconnection network by using a circuit, called the device interface.**
- Which provides the means for **data transfer** and for the exchange of status and control information needed to facilitate the data transfers and govern the operation of the device.
- The interface includes some registers that can be accessed by the processor.
- One register may serve as a buffer for data transfers, another may hold information about the current status of the device, and yet another may store the information that controls the operational behavior of the device.
- These **data, status, and control** registers are accessed by program instructions as if they were memory locations. Typical transfers of information are between I/O registers and the registers in the processor.

Figure illustrates how the keyboard and display devices are connected to the processor from the **software** point of view.

The connection for processor, keyboard, and display.

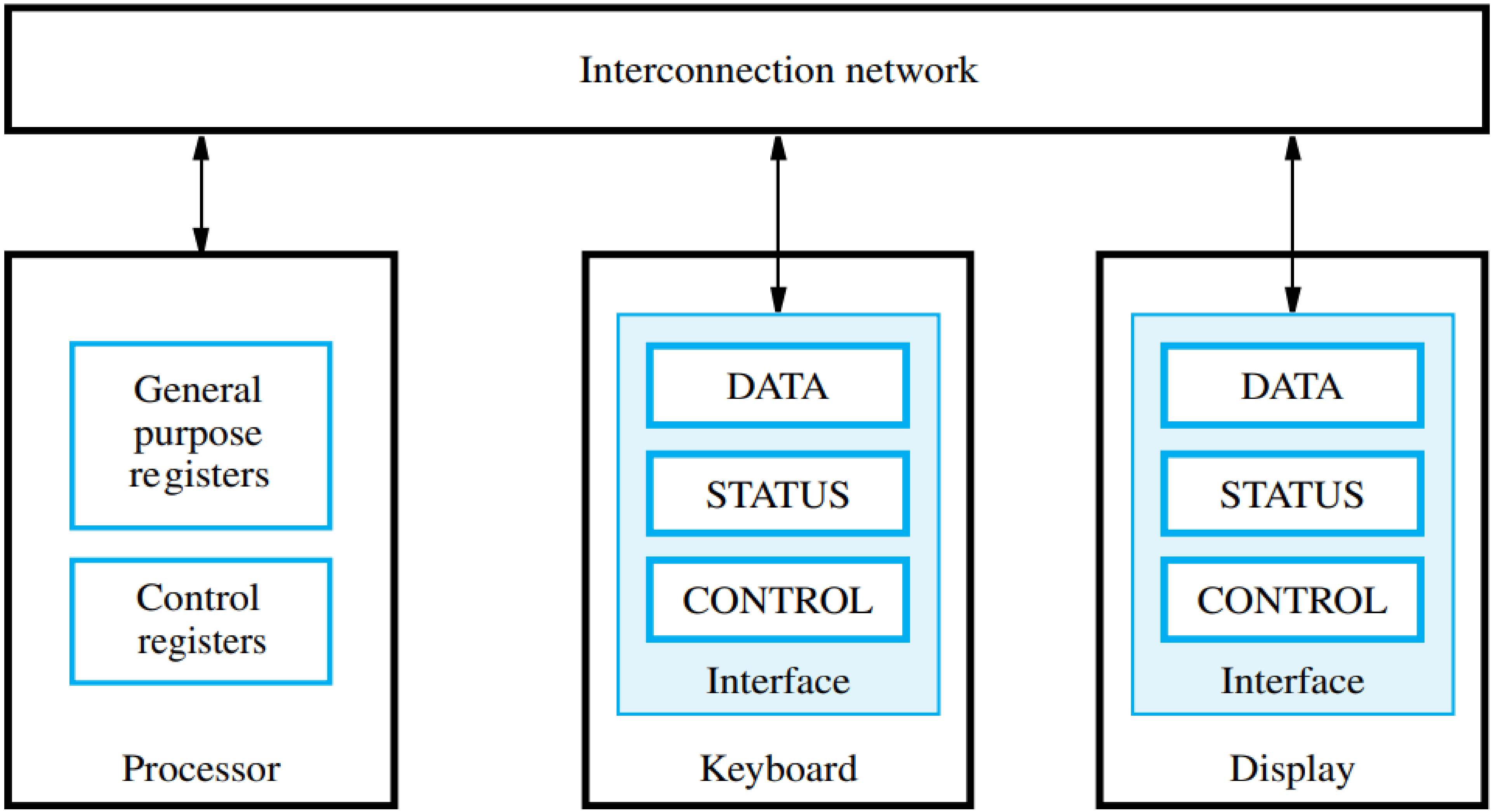


Figure 3.2 The connection for processor, keyboard, and display.

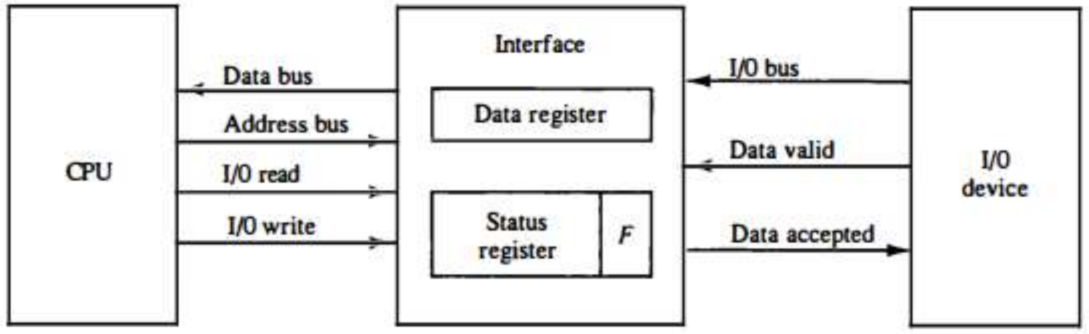
Program-Controlled I/O

- Consider a task that **reads characters typed on a keyboard**, stores these data in the memory, and displays the same characters on a display screen.
- A simple way of implementing this task is to write a program that performs all functions needed to realize the desired action. **This method is known as program-controlled I/O.**
- In addition to transferring each character from the keyboard into the memory, and then to the display, it is necessary to ensure that this happens at the right time.
- An input character must be read in response to a key being pressed. For output, a character must be sent to the display only when the display device is able to accept it.
- The rate of data transfer from the keyboard to a computer is limited by the typing speed of the user, which is unlikely to exceed a few characters per second. The rate of output transfers from the computer to the display is much higher.
- It is determined by the rate at which characters can be transmitted to and displayed on the display device, typically several thousand characters per second. However, this is still much slower than the speed of a processor that can execute billions of instructions per second.
- The difference in speed between the processor and I/O devices creates the need for mechanisms to synchronize the transfer of data between them.

One solution to this problem involves a **signaling protocol.** On output, the processor sends the first character and then waits for a signal from the display that the next character can be sent.

It then sends the second character, and so on. An input character is obtained from the keyboard in a similar way.

The processor waits for a signal indicating that a key has been pressed and that a binary code that represents the corresponding character is available in an I/O register associated with the keyboard. Then the processor proceeds to read that code.



F = Flag bit

The transfer of each byte requires three instructions:

1. Read the status register.

2. Check the status of the flag bit and branch to step 1 if not set or to step

3 if set. Read the data register.

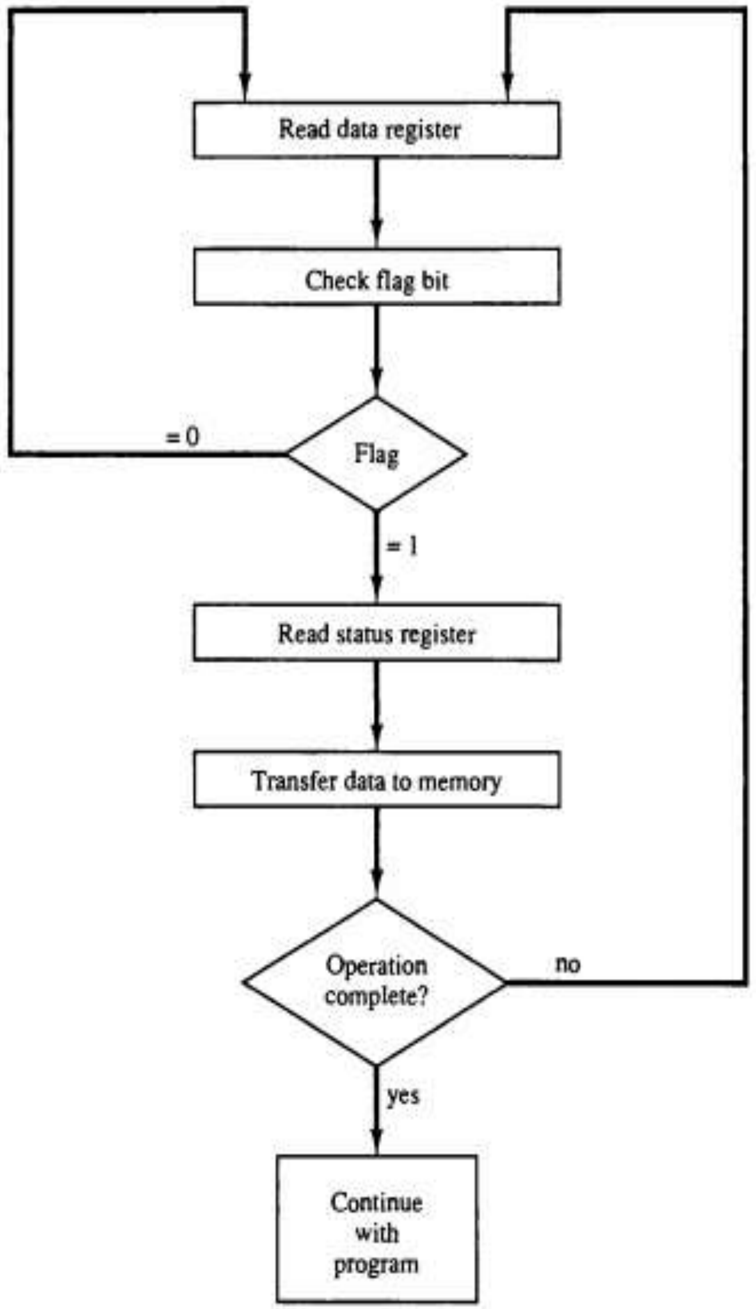
- **Let KBD_DATA be the address label of an 8-bit register that holds the generated character**.

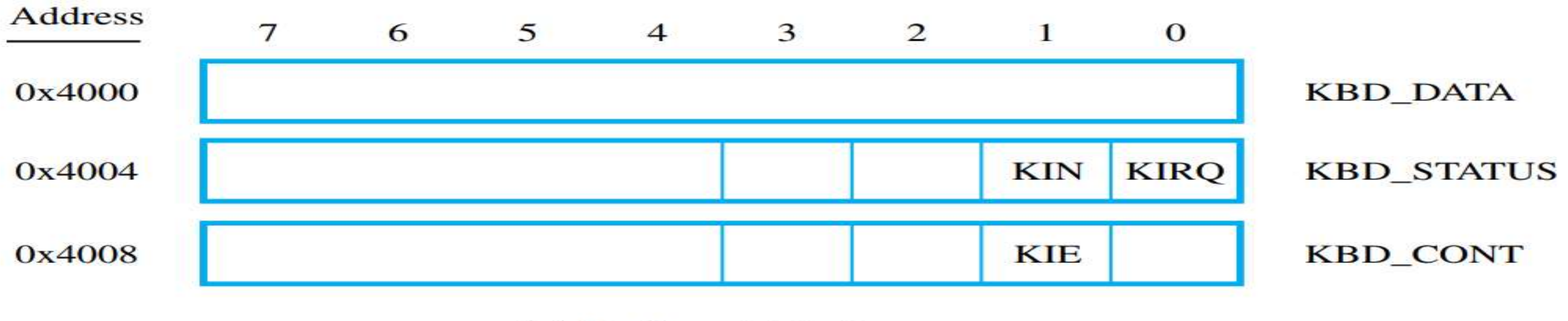
Also, let a signal indicating that a key has been pressed be provided by setting to 1 a flip-flop called KIN, which is a part of an eight-bit status register, KBD_STATUS.

The processor can read the status flag KIN to determine when a character code has been placed in KBD_DATA. When the processor reads the status flag to determine its state.

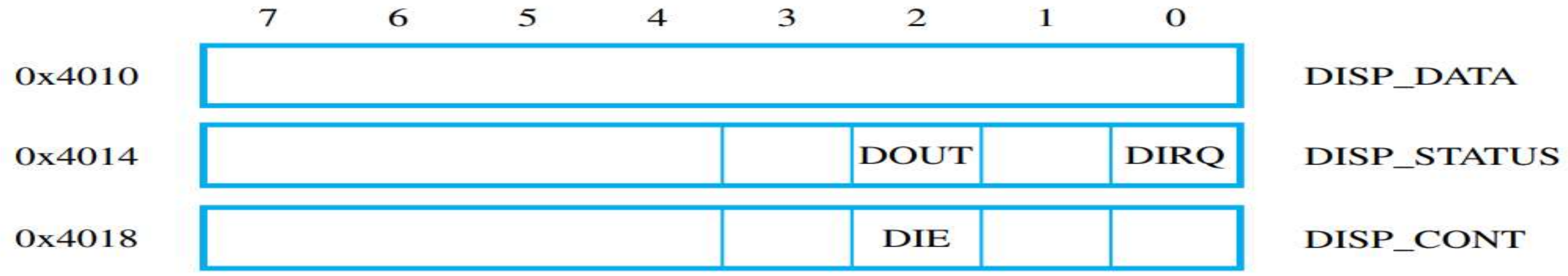
- The display includes an 8-bit register, which we will call DISP_DATA, used to receive characters from the processor.

It also must be able to indicate that it is ready to receive the next character; this can be done by using a status flag called DOUT, which is one bit in a status register, DISP_STATUS.





(a) Keyboard interface



(b) Display interface

- Let us consider the details of the input process. When a key is pressed, the keyboard circuit places the ASCII-encoded character into the KBD_DATA register.
- At the same time, the circuit sets the KIN flag to 1. Meanwhile, the processor is executing the I/O program which continuously checks the state of the KIN flag.

- When it detects that KIN is set to 1, it transfers the contents of KBD_DATA into a processor register.
- Once the contents of KBD_DATA are read, KIN must be cleared to 0, which is usually done automatically by the interface circuit.
- If a second character is entered at the keyboard, KIN is again set to 1 and the process repeats.

The desired action can be achieved by performing the operations:

```
READWAIT                Read the KIN flag
                        Branch to READWAIT if KIN = 0
                        Transfer data from KBD_DATA to R5
```

which reads the character into processor register R5.

- An analogous process takes place when characters are transferred from the processor to the display. When DOUT is equal to 1, the display is ready to receive a character.
- Under program control, the processor monitors DOUT, and when DOUT is equal to 1, the processor transfers an ASCII-encoded character to DISP_DATA.
- The transfer of a character to DISP_DATA clears DOUT to 0. When the display device is ready to receive a second character, DOUT is again set to 1.

This can be achieved by performing the operations:

```
WRITEWAIT     Read the DOUT flag
              Branch to WRITEWAIT if DOUT = 0
              Transfer data from R5 to DISP_DATA
```

The wait loop is executed repeatedly until the status flag DOUT is set to 1 by the display when it is free to receive a character.

Then, the character from R5 is transferred to DISP_DATA to be displayed, which also clears DOUT to 0.

In computers that use memory-mapped I/O, in which some addresses are used to refer to registers in I/O interfaces, data can be transferred between these registers and the processor using instructions such as Load, Store, and Move.

For example, the contents of the keyboard character buffer KBD_DATA can be transferred to register R5 in the processor by the instruction

```
LoadByte R5, KBD_DATA
```

Similarly, the contents of register R5 can be transferred to DISP_DATA by the instruction

```
StoreByte R5, DISP_DATA
```

The Read operation :

```
READWAIT:                 LoadByte R4, KBD_STATUS
                          And R4, R4, #2
                          Branch_if_[R4]=0
                          READWAIT LoadByte R5, KBD_DATA
```

Similarly, the Write operation may be implemented as:

```
WRITEWAIT:                LoadByte R4, DISP_STATUS
                          And R4, R4, #4
                          Branch_if_[R4]=0
                          WRITEWAIT StoreByte R5, DISP_DATA
```

Interrupts

- Data transfer between the CPU and the peripherals is initiated by the CPU. **But the CPU cannot start the transfer unless the peripheral is ready to communicate with the CPU**. When a device is ready to communicate with the CPU, it generates an interrupt signal.
- A number of input-output devices are attached to the computer and each device is able to generate an interrupt request.
- The main job of the interrupt system is to identify the source of the interrupt.
- There is also a possibility that several devices will request simultaneously for CPU communication.
- Then, the interrupt system has to decide which device is to be serviced first.

Priority Interrupts

- It is a system responsible for selecting the priority at which devices generating interrupt signals simultaneously should be serviced by the CPU.
- **High-speed transfer** devices are generally given high priority, and slow devices have low priority. And, in case of multiple devices sending interrupt signals, the device with high priority gets the service first.

Interrupt Service Routine:" An ISR (also called an interrupt handler) **is a software process invoked by an interrupt request from a hardware device.** It handles the request and sends it to the CPU, interrupting the active process. When the ISR is complete, the process is resumed.

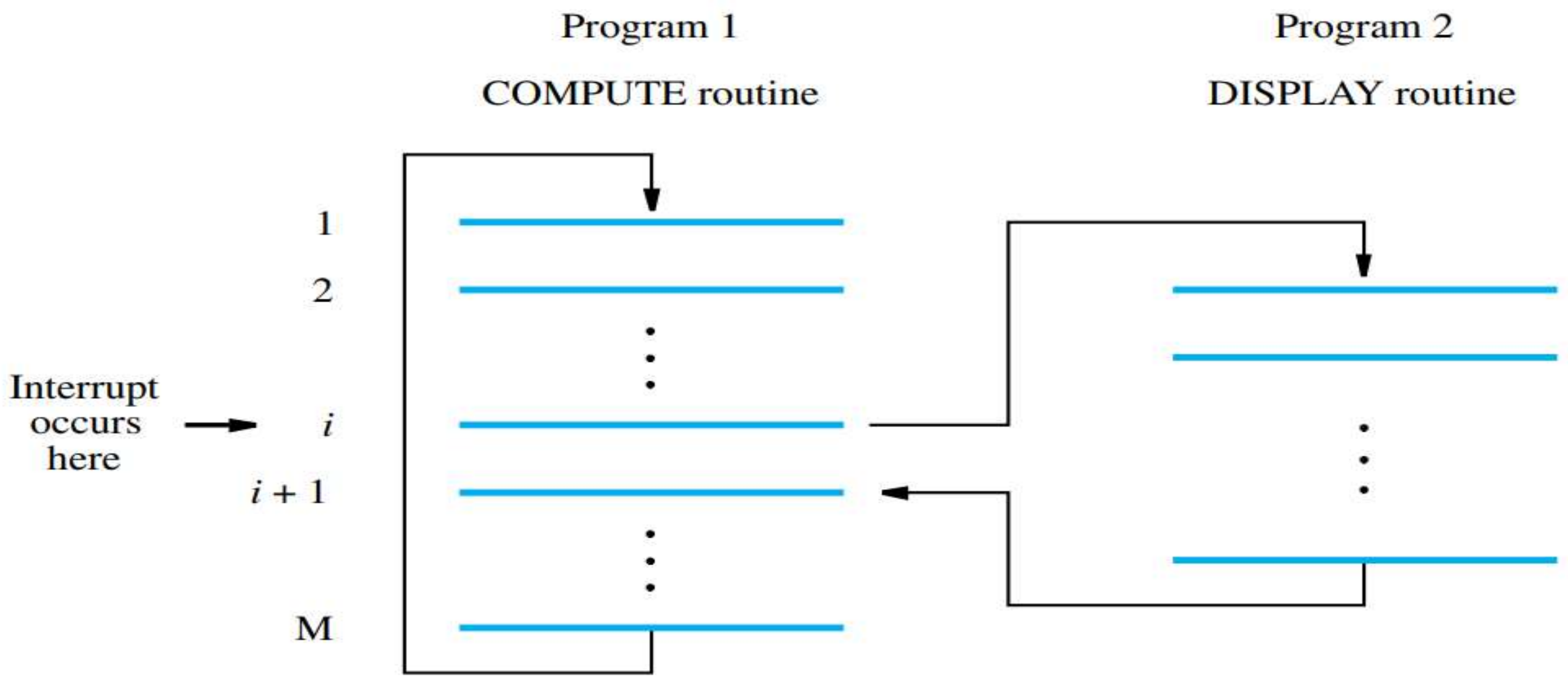


Figure 3.6 Transfer of control through the use of interrupts.

The processor first completes execution of instruction i. Then, it loads the program counter **with the address of the first instruction of the interrupt-service routine.**

For the time being, let us assume that this address is hardwired in the processor. After execution of the interrupt-service routine, the processor returns to instruction i + 1.

Therefore, when an interrupt occurs, **the current contents of the PC**, which point to instruction i + 1, must be put in temporary storage in a known location.

A Return-from-interrupt instruction at the end of the interrupt-service routine reloads the PC from that temporary storage location, causing execution to resume at instruction i + 1.

The return address must be saved either in a designated **general-purpose register or on the processor stack.**

- The processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal. This can be accomplished by means of a special control signal, called interrupt acknowledge
- Interrupt latency is the time between the occurrence of an interrupt, and the time the system has properly responded to that interrupt(ISR). This goes for all computers, systems it’s typically very important to react quickly.
- A different set of registers can be used by the interrupt-service routine, thus eliminating the need to save and restore registers. The duplicate registers are sometimes called the shadow registers.

Enabling and Disabling Interrupts

- Modern computers have facilities to enable or disable interrupts. A programmer must have control over the events during the execution of the program.
- **There are many situations in which the processor should ignore interrupt requests.**
- In another case, 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. For these reasons, some means for enabling and disabling interrupts must be available to the programmer.
- It is convenient to be able to enable and disable interrupts at both the processor and I/O device ends.
- The processor has a status register (PS), which contains information about its current state of operation. Let one bit, IE, of this register be assigned for enabling/disabling interrupts.
- Then, the programmer can set or clear IE to cause the desired action.
- 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.

- Let us now consider the specific case of a single interrupt request from one device. When a device activates 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, perhaps until an instruction is reached that accesses the device in question.
- 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.
- A good choice is to have the processor automatically disable interrupts before starting the execution of the interrupt-service routine.
- The processor saves the contents of the program counter and the processor status register.

let us summarize the sequence of events involved in handling an interrupt request from a single device.

- Assuming that interrupts are enabled in both the processor and the device, the following is a typical scenario:

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

- Consider the situation that the processor is connected to multiple devices each of which is capable of generating the interrupt. Now as each of the connected devices is functionally independent of each other, there is no certain ordering in which they can initiate interrupts.
- Let us say device X may interrupt the processor when it is servicing the interrupt caused by device Y. Or it may happen that multiple devices request interrupts simultaneously. These situations trigger several questions like:

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?

- How these situations are handled vary from computer to computer. Now, if multiple devices are connected to the processor where each is capable of raising an interrupt the how will the processor determine which device has requested an interrupt.
- The solution to this is that whenever a device request an interrupt it set its interrupt request bit (IRQ) to 1 in its status register. Now the processor checks this IRQ bit of the devices and the device encountered with IRQ bit as 1 is the device that has to raise an interrupt.
- But this is a time taking method as the processor spends its time checking the IRQ bits of every connected device. The time wastage can be reduced by using a vectored interrupt.

Vectored Interrupt

- The devices raising the vectored interrupt identify themselves directly to the processor. So instead of wasting time in identifying which device has requested an interrupt the processor immediately start executing the corresponding interrupt service routine for the requested interrupt.
- Now, to identify themselves directly to the processors either the device request with its own interrupt request signal or by sending a special code to the processor which helps the processor in identifying which device has requested an interrupt.
- **Usually, a permanent area in the memory is allotted to hold the starting address of each interrupt service routine. The addresses referring to the interrupt service routines are termed as *interrupt vectors* and all together they constitute an *interrupt vector table*.**

Now how does it work?

- The device requesting an interrupt sends a specific interrupt request signal or a special code to the processor. This information act as a pointer to the interrupt vector table and the corresponding address (address of a specific interrupt service routine which is required to service the interrupt raised by the device) is loaded to the program counter.

Controlling I/O Device Behaviour

I/O devices vary in complexity from simple to quite complex. Simple devices, such as a keyboard, require little in the way of control.

Complex devices may have a number of possible modes of operation, which must be controlled. A commonly used approach is to provide a control register in the device interface, which holds the information needed to control the behavior of the device.

This register is accessed as an addressable location, just like the data and status registers.

One bit in the register serves as the interrupt-enable bit, IE. **When it is set to 1** by an instruction that writes new information into the control register, the device is placed into a mode in which it is allowed to interrupt the processor whenever it is ready for an I/O transfer.

Figure (Program-Controlled I/O) shows the registers that may be used in the interfaces of keyboard and display devices. Since these devices transfer character-based data, handling one character at a time, it is appropriate to use an eight-bit data register.

We have assumed that the status and control registers are also eight bits long. Only one or two bits in these registers are needed in handling the I/O transfers. The remaining bits can be used to specify other aspects of the operation of the device, or ignored if they are not needed. The keyboard status register includes bits KIN and KIRQ.

The KIRQ bit is set to 1 if an interrupt request has been raised, but not yet serviced. The keyboard may raise interrupt requests only when the interrupt-enable bit, KIE, in its control register is set to 1.

Thus, when both KIE and KIN bits are equal to 1, an interrupt request is raised and the KIRQ bit is set to 1. Similarly, the DIRQ bit in the status register of the display interface indicates whether an interrupt request has been raised.

Bit DIE in the control register of this interface is used to enable interrupts. Observe that we have placed KIN and KIE in bit position 1, and DOUT and DIE in position 2.

Processor Control Registers

- To deal with interrupts it is useful to have some other control registers. Figure shows one possibility, where there are four processor control registers.
- The status register, PS, includes the interrupt-enable bit, IE, in addition to other status information. Recall that the processor will accept interrupts only when this bit is set to 1.
- The IPS register is used to automatically save the contents of PS when an interrupt request is received and accepted.
- At the end of the interrupt-service routine, the previous state of the processor is automatically restored by transferring the contents of IPS into PS.

Since there is only one register available for storing the previous status information, it becomes necessary to save the contents of IPS on the stack if nested interrupts are allowed.

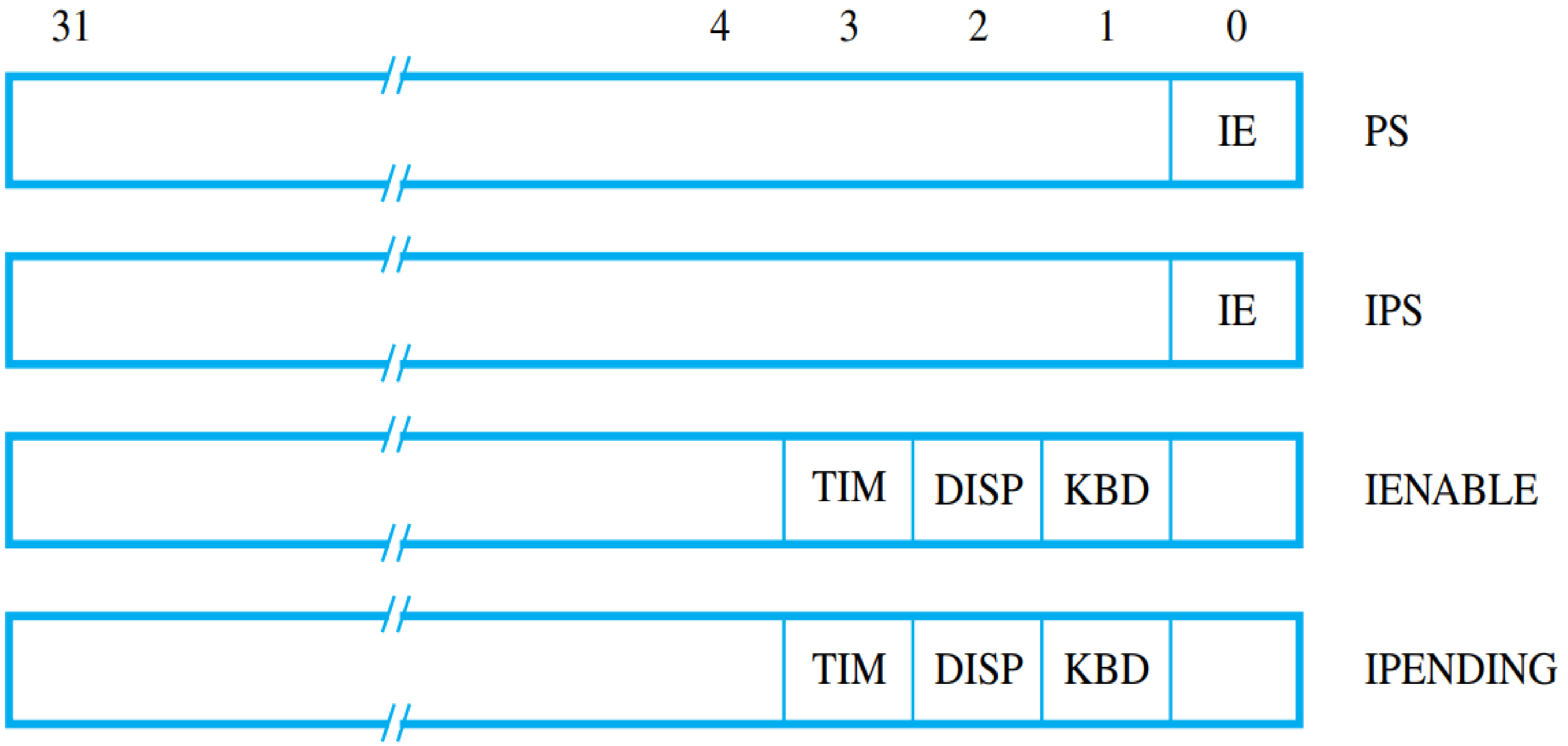


Figure 3.7 Control registers in the processor.

- The IENABLE register allows the processor to selectively respond to individual I/O devices. A bit may be assigned for each device, as shown in the figure for the keyboard, display, and a timer circuit.
- When a bit is set to 1, the processor will accept interrupt requests from the corresponding device.
- The IPENDING register indicates the active interrupt requests. This is convenient when multiple devices may raise requests at the same time. Then, a program can decide which interrupt should be serviced first.
- In a 32-bit processor, the control registers are 32 bits long. Using the structure in Figure 3.7, it is possible to accommodate 32 I/O devices in a straightforward manner.
- But, these registers cannot be accessed in the same way as the general-purpose registers. They cannot be accessed by arithmetic and logic instructions.
- They also cannot be accessed by Load and Store instructions that use the encoding format, because a five-bit field is used to specify a source or a destination register in these instructions, which makes it possible to specify only 32 general-purpose registers.
- Special instructions or special addressing modes may be provided to access the processor control registers.

In a RISC-style processor, the special instructions may be of the type

```
MoveControl R2, PS
```

which loads the contents of the program status register into register R2, and

```
MoveControl IENABLE, R3
```

which places the contents of R3 into the IENABLE register. These instructions perform transfers between control and general-purpose registers.