CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test - II

| Sub: | Computer Organization and Architecture | | | | | | | Code: | BEC306C |
|---|---|---|---|---|---|---|---|---|---|
| Date: | 19/01/2024 | Duration: | 90 mins | Max Marks: | 50 | Sem: | 3rd | Branch: | ECE |

Answer Any FIVE FULL Questions

| | | Marks | OBE | |
|---|---|---|---|---|
| | | | CO | RBT |
| 1. | With an example and blockdiagram, discuss the basic operational concepts of computer. | [10] | CO2 | L2 |
| 2. | What are assembler directives? Explain any four assembler directive. | [10] | CO2 | L2 |
| 3. | With a neat block diagram describe the input and output operations. | [10] | CO2 | L2 |
| 4. | Define Subroutine. Explain subroutine linkage using a link register. | [10] | CO2 | L2 |
| 5. | Explain various types of rotate instructions. | [10] | CO2 | L2 |
| 6. | Explain how I/O devices can be interfaces with a block diagram. | [10] | CO2 | L2 |

Solutions: 1

## Basic Operational Concepts

To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be used as operands are also stored in the memory. A typical instruction may be

*Add LOCA, R0*

This instruction adds the operand at memory location LOCA to the operand in a register in the processor, R0, and places the sum in the register R0. The original contents of location LOCA are preserved whereas those of R0 are overwritten. First the instruction is fetched from the memory into the processor. Next the operand at LOCA is fetched and added to the contents of R0. Finally the resulting sum is stored in register R0.

Transfers between memory and processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data is transferred to or from the memory. The memory and processor connection is shown in Fig 1.2.

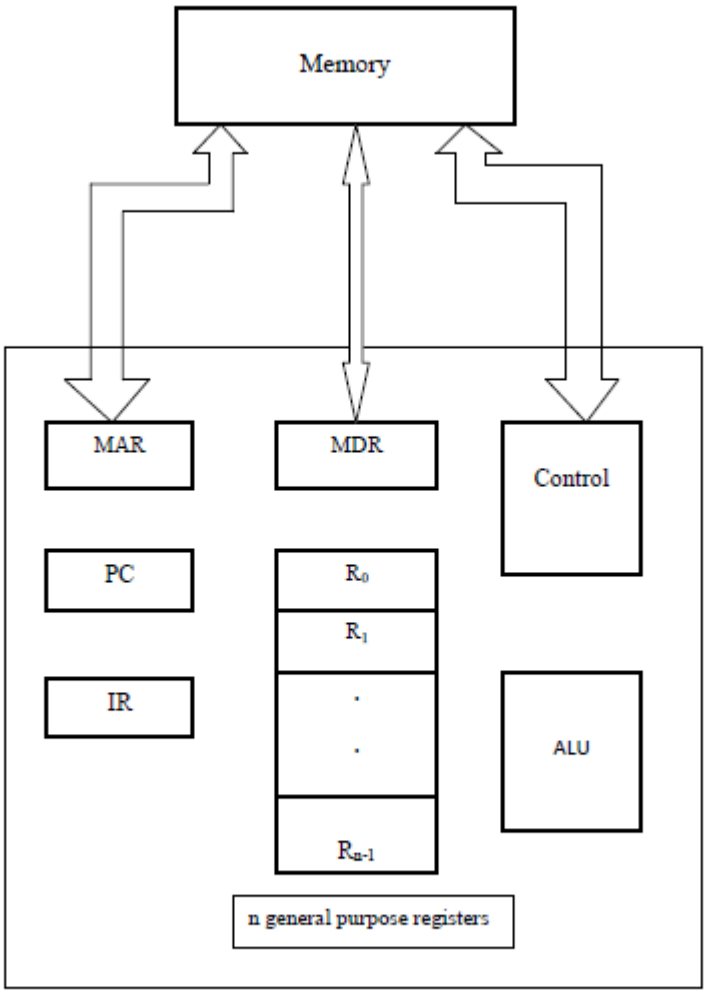



**Fig 1.2** Connections between the processor and memory

The Instruction register (IR) holds the instruction that is currently being executed. Its output is available to control circuits which generate the timing signals that control various processing elements involved in executing the instruction.

The Program Counter (PC) holds the address of the next instruction to be fetched and executed. During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed. MAR and MDR facilitate communication with the memory.

MAR (Memory Address Register) hold the address of the location to be accessed and MDR (Memory Data Register) contains data written into or read out of the addressed location.

If some devices require urgent servicing then they raise the interrupt signal interrupting the normal execution of the current program. The processor provides the requested service by executing the appropriate interrupt service routine.

2.

## Assembler Directives

The assembly language allows the programmer to specify other information needed to translate the source program to object program. Suppose the name SUM is used to represent the value 200. This fact may be conveyed to the assembler program through a statement such as

```
SUM  EQU  200
```

This statement does not denote the instruction that will be executed when the object program is run. It informs the assembler that the name SUM should be replaced by the value 200 wherever it appears in the program. Such statements are *assembler directives* (or commands) are used by the assembler when it translates the source program in to a object program.

**ORIGIN** is a directive that tells the assembler program where in the memory to place the data block.

**DATAWORD** directive is used to inform the assembler to place the data in the address.

**RESERVE** directive declares a memory block and does not cause any data to be loaded in these locations.

**ORIGIN** directive specifies that the instructions of an object program are to be loaded in the memory starting at an address.

3.

## Basic Input/Output Operations

Consider a task that reads in a character input from a keyboard and produces a character output on a display. A simple way of performing such tasks is to use methods known as *program-controlled I/O*. The difference in speed between the processor and I/O devices creates the need for mechanisms to synchronize the transfer of data between them.

Consider the problem of moving a character code from the keyboard to the processor. Striking a key store the corresponding character code in an 8-bit buffer register associated with the keyboard. Let us call this register DATAIN as shown in Fig 2.3. To inform the processor that a valid character is in DATAIN, a status control flag, SIN, is set to 1. A program monitor SIN, and when SIN is set to 1, the processor reads the contents of DATAIN. When the character is transferred to processor, SIN is automatically cleared to 0. If the second character is entered at the keyboard, SIN is again set to 1 and the process repeats.

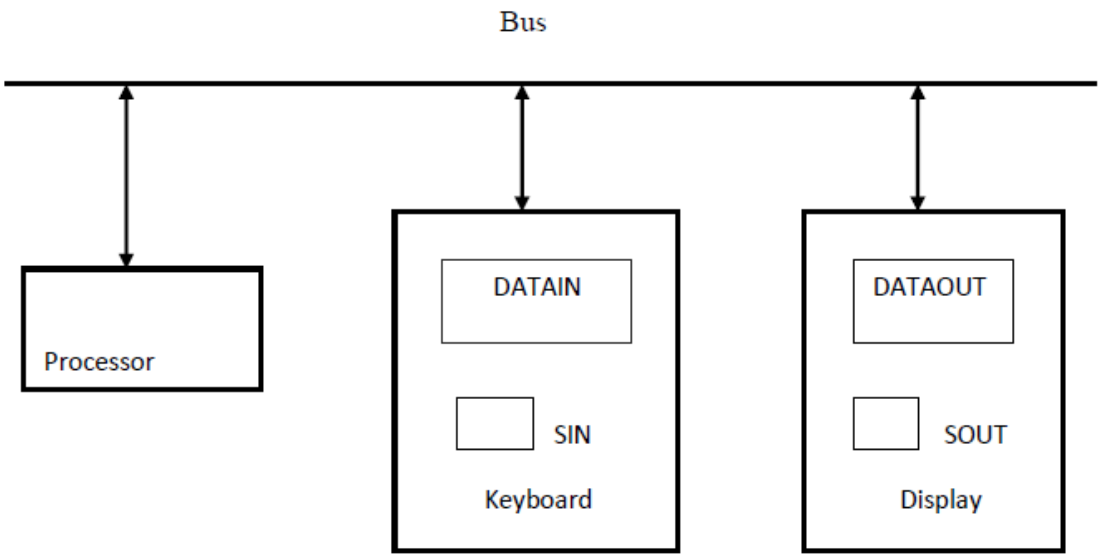



**Fig 2.3:** Bus connection for processor, keyboard and display

An analogous process takes place when the characters are transferred from the processor to display. A buffer register, DATAOUT and a status control register SOUT are used for this transfer. When SOUT equals 1, the display is ready to receive a character. Under program control, the processor monitors SOUT and when SOUT is set to 1, the processor transfers a character code to DATAOUT. The transfer of character to DATAOUT clears SOUT to 0, when the display is ready to receive a second character; SOUT is set again to 1. The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry known as *device interface.*

The processor can monitor the keyboard status flag SIN and transfer a character from DATAIN to register R! By the following sequence of operations:

**READWAIT Branch to READWAIT if SIN=0**

**Input from DATAIN to R1**

The first instruction tests the status flag and the second performs the branch. The processor monitors the status flag by executing a short *wait loop* and proceeds to transfer the input data when SIN is set to 1 as a result of key being struck. The input operation resets SIN to 0. The sequence of operations are used for transferring the output to display are

**WRITEWAIT Branch to WRITEWAIT if SOUT=0**

**Output from R1 to DATAOUT**

Many computers use an argument called *memory mapped I/O* in which some memory address values are used to refer to the peripheral device buffer registers such as DATAIN and DATAOUT. The keyboard character buffer DATAIN can be transferred to register R1 in the processor by the instruction

MoveByte DATAIN, R1

Similarly the contents of R1 can be transferred to DATAOUT by the instruction

MoveByte R1, DATAOUT

4.

## Subroutines

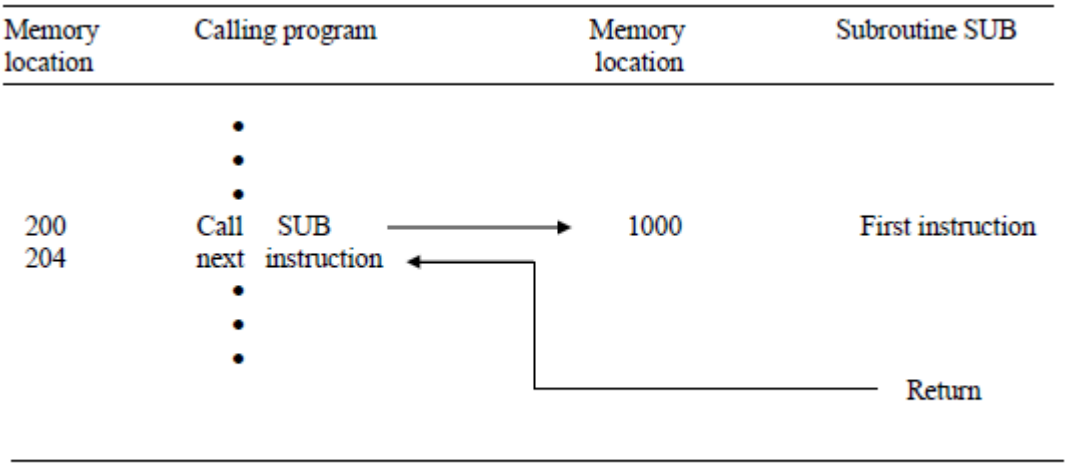
It is often necessary to perform a particular subtask many times on different data values. Such subtask is called *subroutine*. When a program branches to a subroutine we call that it is *calling* a subroutine. The instruction that performs this branch operation is called a Call instruction. The subroutine is said to *return* to program that called it by executing a Return instruction. The location where the calling program resumes execution is the location pointed by the updated PC while the Call instruction being executed. Hence the contents of the PC must be saved by the Call instruction to enable correct return to the calling program. This way in which the computer makes it possible to call and return from subroutines is referred to as *subroutine linkage method*.

The Call instruction is a special branch instruction that performs the following operations:

1. Store the contents of PC in the link register.
2. Branch to the target address specified by the instruction.

The Return instruction is a special branch instruction that performs the operation:
Branch to the address contained in the link register.

Fig 2.6 illustrates this procedure.




5.

## Shift and Rotate Instructions

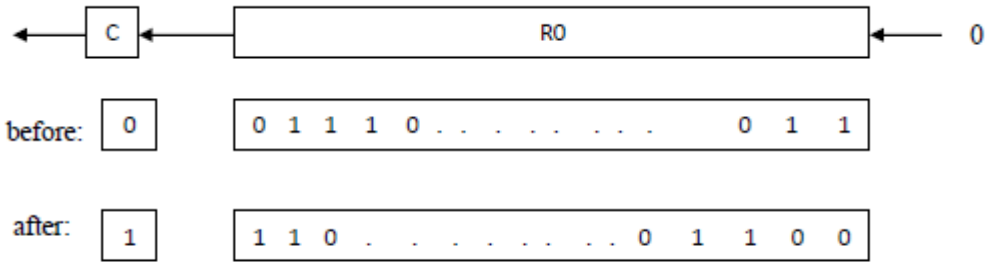
There are applications that require bits of an operand to be shifted to the right or left some specified number of bit positions. For general operands we use a logical shift. For a number we use an arithmetic shift which preserves the sign of the number.

### Logical Shifts

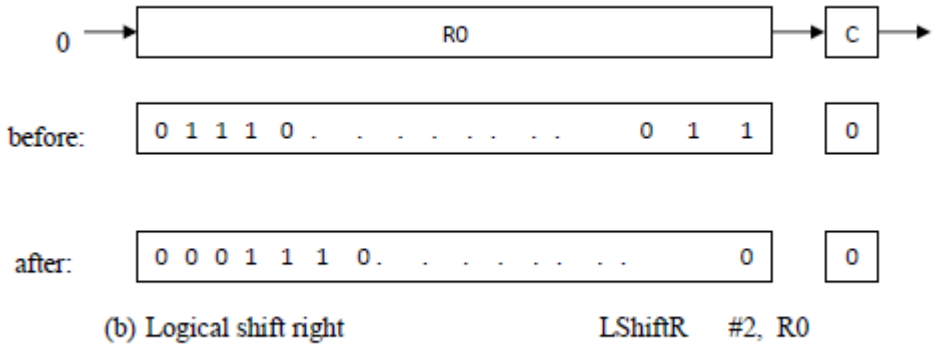
Two logical shift instructions are needed, one for shifting left (LShiftL) and another for shifting right (LShiftR). These instructions shift an operand over a number of bit positions specified in a count operand contained in the instruction. The general form of logical left shift instruction is

LShiftL count, dst

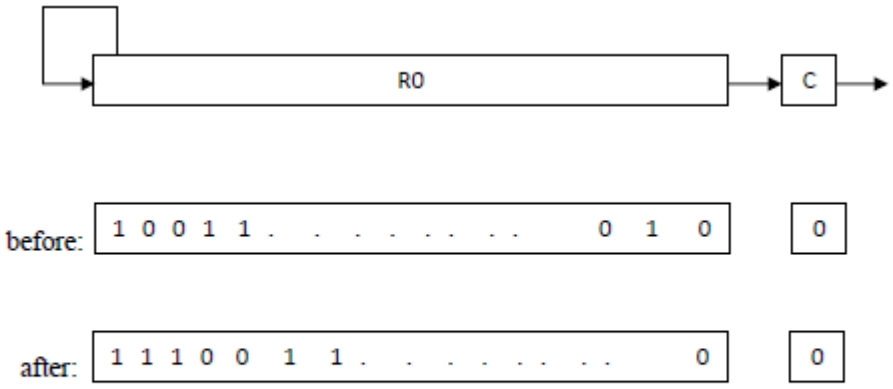
The count operand may be given as an immediate operand or it may be contained in the processor register. Vacated positions are filled with zeros, and the bits shifted out are passed through the Carry flag C, and then dropped. Involving the C flag in shifts is useful in arithmetic operations on large numbers that occupy more than one word. Fig 2.10 illustrates all the shift operations.




(a) Logical shift Left LShiftL #2, R0




(b) Logical shift right LShiftR #2, R0

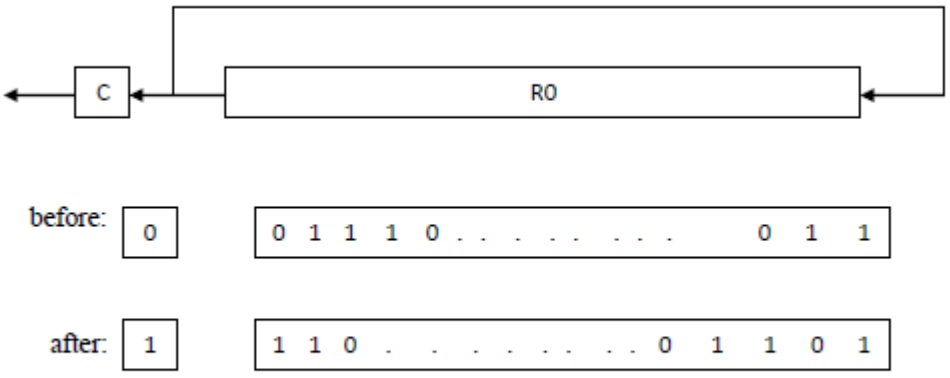



(c) Arithmetic shift right AShiftR #2, R0

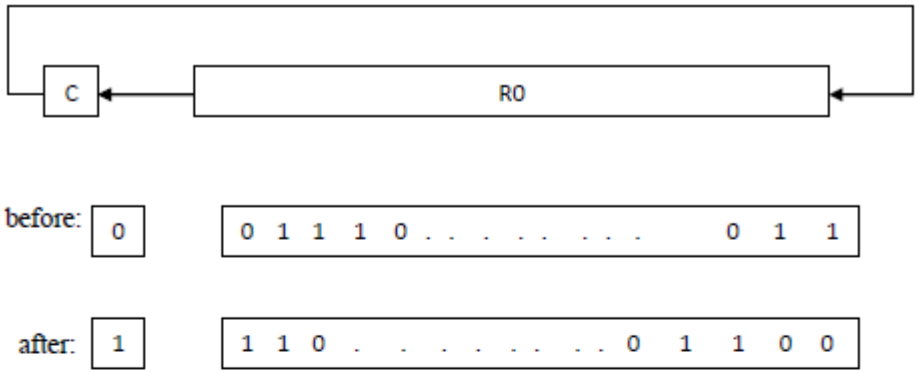
**Fig 2.10:** Logical and arithmetic shift instructions

## Rotate Operations

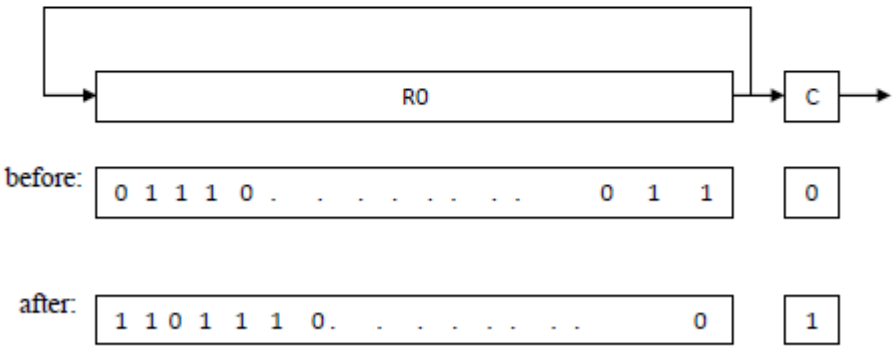
To preserve all the bits, a set of rotate can be used. They move the bits that are shifted out of one end of the operand back in to the other end. Two versions of both left and right rotate instructions are provided. In one version, bits are of the operand are simply rotated. In the other version, the rotation includes the C flag. Figure 2.11 shows the left and right rotate operations with and without C flag being include in the rotation. Note that when C flag is not included in the rotation, it still retains the last bit shifted out of the end of the register.



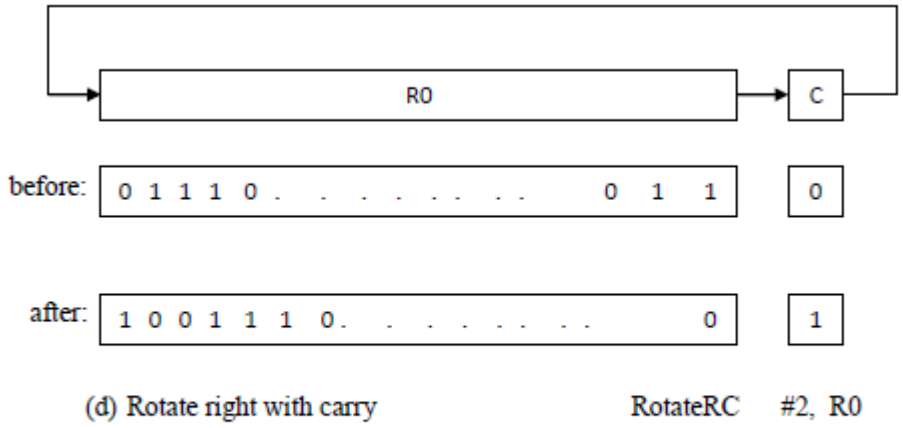

(a) Rotate left without carry RotateL #2, R0




(b) Rotate left with carry RotateLC #2, R0




(c) Rotate right without carry RotateR #2, R0




(d) Rotate right with carry RotateRC #2, R0

**Fig 2.11:** Rotate instructions

6.

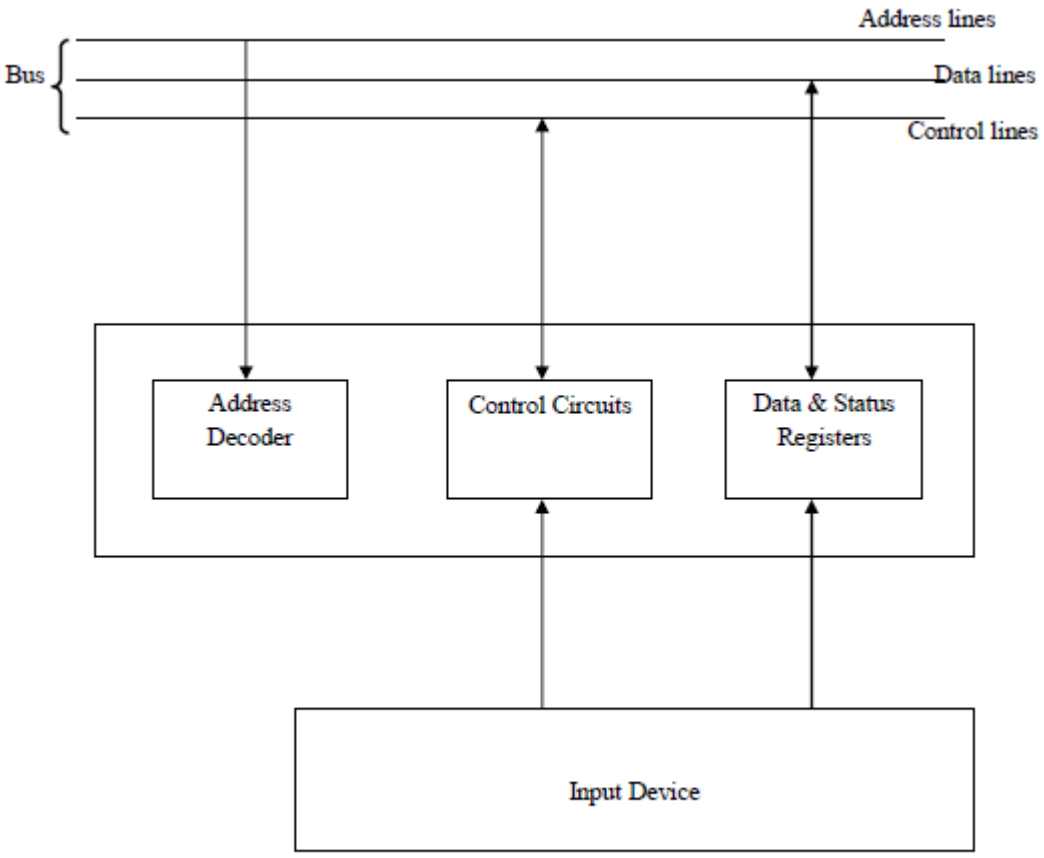



**Fig 2**: I/O interface of an input device

Fig 2 illustrates the hardware required to connect the I/O device to the bus. The address decoder enables the device to recognize its address when its address appears on the address lines. The data register holds the data being transferred to or from the processor. The status register contains the information relevant to the operation of I/O device. Both status and data registers are connected to the data bus and assigned unique addresses.

The address decoder, data & status registers and the control circuitry required to coordinate I/O transfers constitutes the device interface circuit.