Input-Output Operations

Chapter 3

INTRODUCTION

To perform any useful task, the microcomputer must interact with the outside world. The input-output (I/O) devices or peripherals provide the necessary data communications link between the microprocessor and its environment. Typically, information is accepted from the input devices, it is processed and the results of the data processing are then sent to one or more output devices. In a microcomputer system, the input-output operations are particularly important since, in the majority of applications, the microprocessor spends the greatest part of its time interacting with the I/O devices.

The operation of the I/O devices is usually independent of that of the microprocessor, and a procedure must be adopted to synchronise program execution with their operation during data transmission. There are three basic types of input-output according to the method of controlling and synchronising data transfer:

(i) Program-controlled I/O
(ii) Interrupt-controlled I/O
(iii) Direct-memory-access I/O.

The type of input-output used in a particular application will depend on three main factors:

(i) The rate at which data must be transmitted.
(ii) The maximum time delay which can be accepted between the I/O device signalling its readiness to transmit or receive data and the data transfer actually taking place.

(iii) The feasibility of interleaving input-output and other microprocessor operations.

In this chapter, input-output operations using each of the three methods of controlling data transfer are described. The software techniques used to synchronise data transmission with program execution are explained, and the characteristics of each type of input-output are discussed. The hardware interconnection of the I/O devices and the microprocessor is described in the next chapter.

PROGRAM-CONTROLLED I/O

As shown in Figure 3-1, two basic types of information are transmitted between the microprocessor and the I/O device. These are *message data* and *control data.* The control data is used to synchronise the operation of the device with the execution of the program before transmission of the message data takes place. The input control data is called the *device status word.* The output control data is called the *device command word.* With program-controlled I/O, the input-output instructions are used to initiate and control the transfer of all types of data.

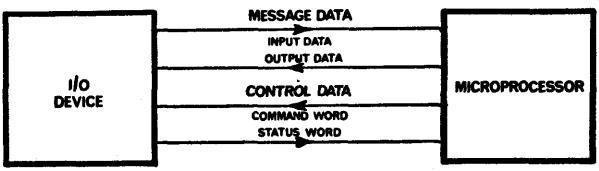


Figure 3-1. Information flow between I/O device and microprocessor

The status word is read into the microprocessor to determine the current state of the device. Each bit of the status word will indicate a particular device condition such as message data ready for transmission, device busy, device unavailable, or transmission error.

The command word is sent out from the microprocessor to control the operation of the device. Each bit of the command word has a particular function such as stop motor, increment feed, or change transmission rate.

The input-output instructions can be organised in one of the three ways:

(i) A unique instruction is provided for each kind of I/O data transfer using a single *device address* to define each I/O device. Typically, the four instructions are:

1) Read data (input message data)
2) Write data (output message data)
3) Send command (output command word)
4) Accept status (input status word)

(ii) Two I/O instructions, one for input and one for output, are provided for both message and control data transfer. Two device addresses are used to differentiate between transmission of message or control data for a particular I/O device. Typically, the two instructions are:

1) Read data (input either message data or status word)
2) Write data (output either message data or command word)

(iii) No separate I/O instructions are provided. The memory data transfer instructions are also used to communicate with the I/O devices by assigning a block of unused memory addresses as the device addresses. The approach is called *memory-mapped I/O* and is common in microprocessor systems which have a unified bus structure (see Chapter 4). Although the available memory address area is made smaller, this approach can reduce both program storage requirements and program execution times. The equivalent I/O instructions are typically:

1) Load data (input message data or status word)
2) Store data (output message data or command word)

In most microprocessors both message and control data are sent or received via the accumulator or some other working register. Some systems have special-purpose registers to deal with the control data or use the main processor status register for this purpose. Device status testing is simplified in the latter case since the conditional branch instructions can check directly the condition of the individual bits of the status register.

The control data is used to synchronise data transfer under program control in the following way:

(i) A command word is written out to the I/O device to request transfer of message data.

(ii) The status word is read in from the I/O device.

(iii) The appropriate status bits are checked to test if message data transfer can take place.

(iv) If the device is not ready, steps (ii) and (iii) are repeated until the I/O device is ready for data transfer.

(v) The message data is read (or written) from (or to) the I/O device. This operation will reset the status of the I/O device.

Step (i) is omitted in applications where the decision to initiate data transfer originates from the I/O device itself. In this case, the device indicates its desire for data transfer by setting the appropriate status bits.

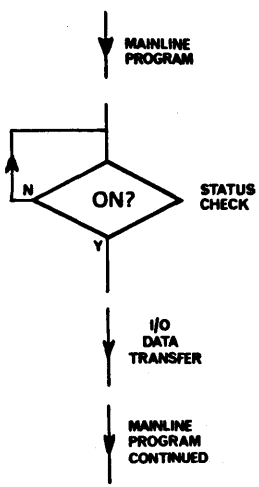


Figure 3-2a. Program controlled I/O (status loop)

In a simple program, the status check (steps (ii) and (iii)) is repeated continuously until the device is ready as shown in Figure 3-2a. The status check loop effectively halts the program execution and may cause an unacceptable waste of useful processing time. More sophisticated schemes, as shown in Figure 3-2b, where the status

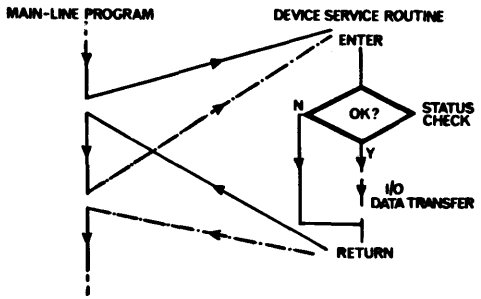


Figure 3-2b. Program controlled I/O (interleaved operation)

check operation is interleaved with other microprocessor operations, can use the processor time more efficiently. The problem is particularly significant in a microprocessor system which communicates with several I/O devices, since periodic status checks must be made on each of the devices. This *device polling* operation may also introduce a considerable time delay between a device indicating readiness for data transfer, and the program sensing that readiness and the data transfer actually taking place. In some microprocessors, the time spent in checking device status is

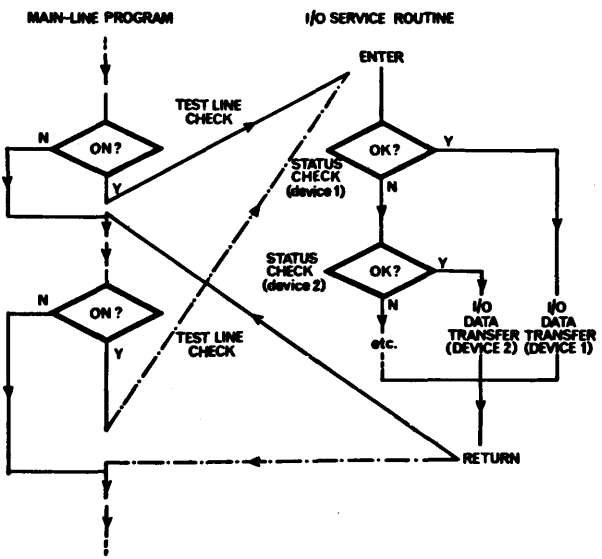


Figure 3-2c. Program controlled I/O (test line)

reduced by using a single test line. This line is common to all devices and may be used for signalling when any device requires attention. As shown in Figure 3-2c, the microprocessor can periodically and rapidly check the status of this one line and thus avoid polling the individual devices until one of the devices has signalled that attention is required. The time delay before servicing a device may still be considerable.

EXAMPLE OF PROGRAM-CONTROLLED I/O

A teletype is used as the output device for a microprocessor-based instrument. The results of data processing are first stored in a memory buffer and then written out to the teleprinter under

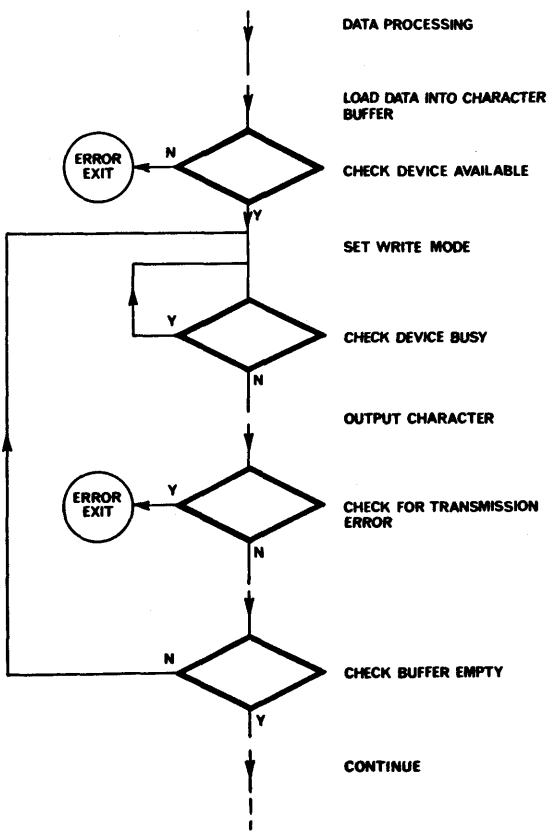
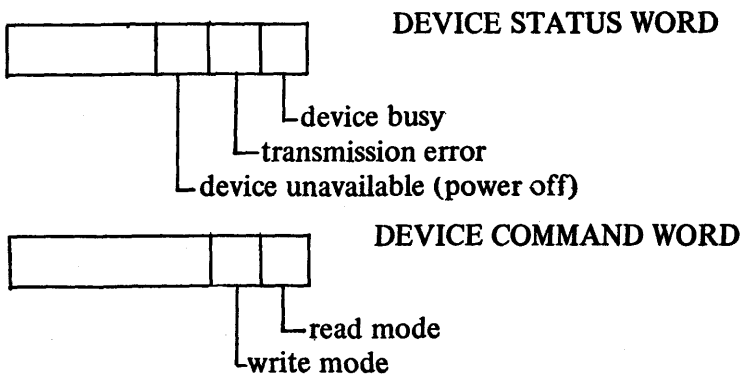


Figure 3-3. Program controlled output to teleprinter

program-control. A simplified flow-chart of the section of the program dealing with data output is given in Figure 3-3. The device status word and device command word for the teletype are given below:



INTERRUPT-CONTROLLED I/O

The major disadvantage of program-controlled I/O arises from the necessity for periodically leaving the main data processing section of the program to check whether any device is ready for data transfer. The checking procedure, which must occur whether or not any device is ready, can waste valuable processing time. The problem is overcome in many microprocessors by introducing an *interrupt system* which allows the I/O devices to break into (or interrupt) the main program execution when, and only when, they are ready for data transfer.

In the simplest type of interrupt system, only one I/O device is connected to a *single interrupt request line.* The occurrence of a signal on this line causes the microprocessor to automatically initiate the following minimal sequence of operations:

(i) Complete execution of the current program instruction.

(ii) Store the current contents of the program counter.

(iii) Load the program counter with a predefined program memory address.

(iv) Inhibit interrupts and resume normal program execution according to the new contents of the program counter.

Thus recognition of an interrupt request signal causes a jump from the main-line program to a predetermined location in program memory (*the interrupt trap address*). In a simple system, with only one I/O device capable of generating interrupts, the *device service program* which controls the actual data transfer is loaded from the interrupt trap address. As shown in Figure 3-4, after completing the device service program, the previously stored contents of the program counter provide the *return address* to link back and continue execution of the main-line program. Interrupts are automatically inhibited before the start of execution of the device service program to prevent multiple interruption by the same interrupt request signal. In some microprocessors, the instruction causing the jump back to the main-line program also re-enables interrupts.

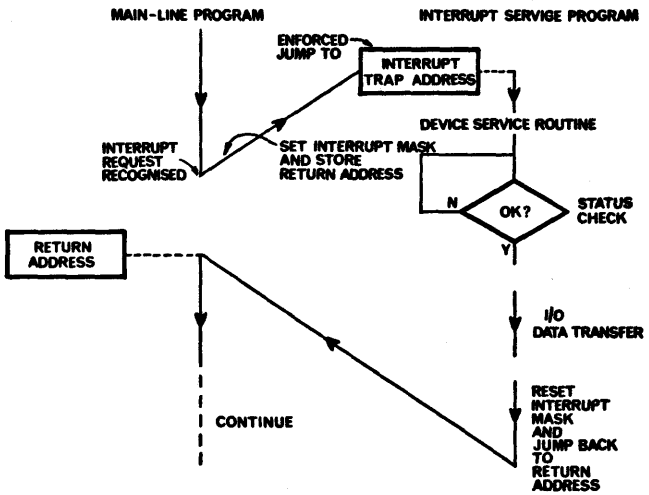


Figure 3-4. Interrupt controlled I/O

It is noted that the time delay before servicing a single I/O device under interrupt control can be longer than that occurring under program control, since the interrupt recognition, the hardware enforced jump and the store sequence require several machine cycles for completion.

Interrupts are inhibited by setting an *interrupt mask bit* within the

microprocessor. In most systems the mask bit is part of the main processor status register and can also be set or reset by software. The mask bit is frequently used to prevent the interruption of certain sections of program which must be executed without a break (e.g., a section of data processing which must be completed before the next input-output operation can take place).

Many microprocessors use a slightly different interrupt system which employs a form of indirect addressing to link with the device service program. The interrupt trap address contains the address of the first instruction of the service program rather than

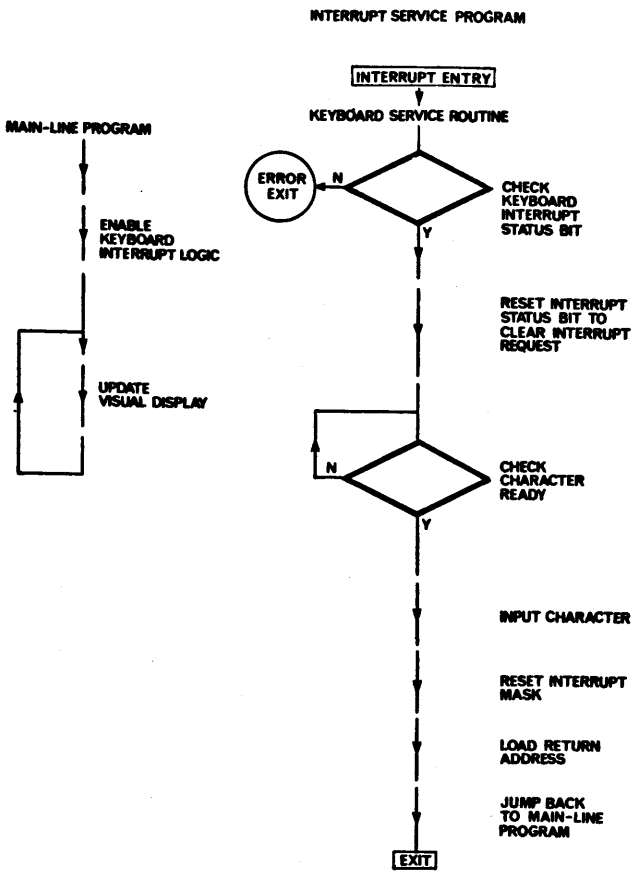
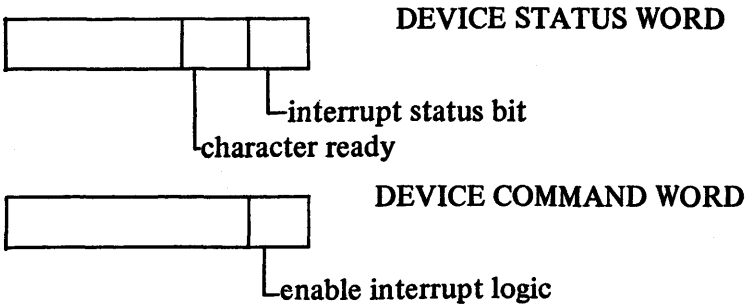


Figure 3-5. Interrupt controlled input from a keyboard

the instruction itself. The interrupt hardware automatically loads this address into the program counter before resuming normal program execution as described before. Indirect addressing allows the service program to be located at any arbitrary position in program memory. If the interrupt trap address refers to a location in read/write memory, the service program entry point can be changed during execution of the main-line program and the response of the microprocessor to an interrupt request varied accordingly.

EXAMPLE OF INTERRUPT-CONTROLLED I/O

An interactive computer system is based on a visual-display-terminal linked to a microcomputer. The main-line program communicates with the operator by writing out, under program control, information onto the display screen. At any time the operator can modify the program flow and change the information presented on the screen by entering control characters at the keyboard. An interrupt request is generated whenever a key is depressed. Data input takes place under interrupt-control. A simplified flow-chart of the interrupt driven section of the program is shown in Figure 3-5. The device status word and device command word for the keyboard are given below:



The similarity between the interrupt service sequence and the execution of a jump to subroutine instruction should be noted. Both cause the contents of the program counter to be saved and then restored to allow the return to the main-line program.

When the service program uses or modifies the internal working

registers of the microprocessor (accumulators, index registers, status register, etc.), the main-line program execution could be upset since, unless the service program makes provision to save and restore the contents of these registers, they would be altered following the interrupt service. More sophisticated interrupt systems use the stack to automatically save the contents of the important internal registers as well as the program counter in response to an interrupt. Before the microprocessor resumes main-line program execution, the contents of the registers are automatically restored to their original values by the return jump instruction. The more registers automatically stored in this way, the longer will be the *interrupt response time*, i.e., the time between initial recognition of an interrupt request and the execution of the first instruction in the service program.

A more rapid interrupt response time is provided by microprocessors which have an architecture specially designed to facilitate interrupt operation. Some have two sets of internal registers. The main-line program is executed using one set whilst the service program uses the other set so as to prevent interference. Other microprocessors use locations in data memory to replace some of the usual internal registers. A single internal register, the work space pointer register, defines the memory locations to be used. The interrupt hardware stores and modifies the contents of the pointer register before executing the service program and thus defines a different workspace in data memory to that used by the main-line program. The pointer register is restored to its original value following completion of the service program.

REAL-TIME OPERATION

In many applications, the input-output is required to take place at a particular instant in time or periodically with a given time interval. In these cases, the operation of the I/O device and the execution of the program controlling data transfer must be synchronised in real-time.

The synchronisation is achieved by connecting an external pulse generator of known and constant frequency to the interrupt

request line. The program is interrupted periodically with a known time between interrupts. The input-output operations can be synchronised to "real-time" by counting the interrupt requests and controlling program flow accordingly. Used in this way, the external pulse generator, which usually consists of a high frequency oscillator (typically about 1MHz) feeding a chain of frequency dividers, is called a *real-time clock*. Programmable real-time clock chips, which also allow software control of the clock interrupt rate, are provided in some microprocessor systems.

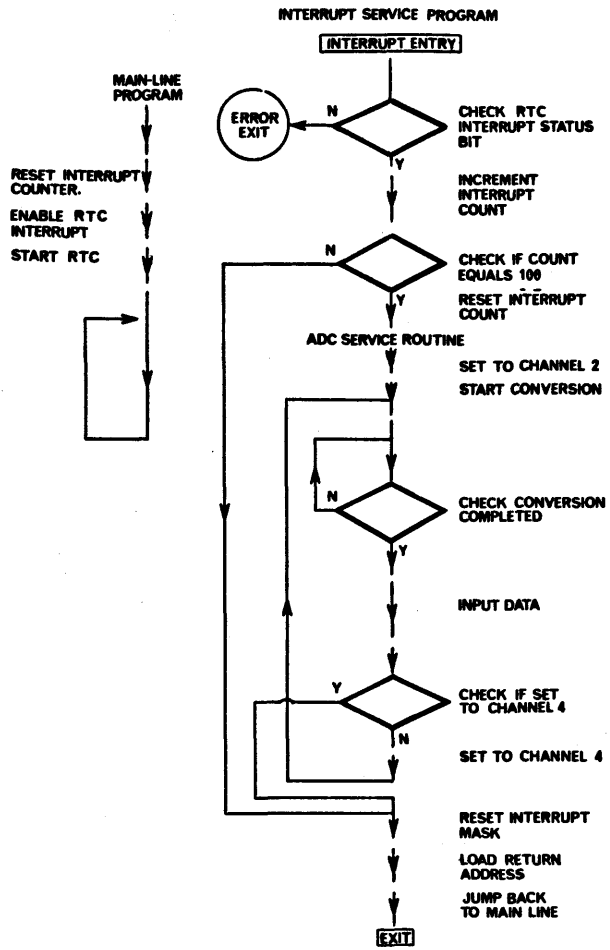
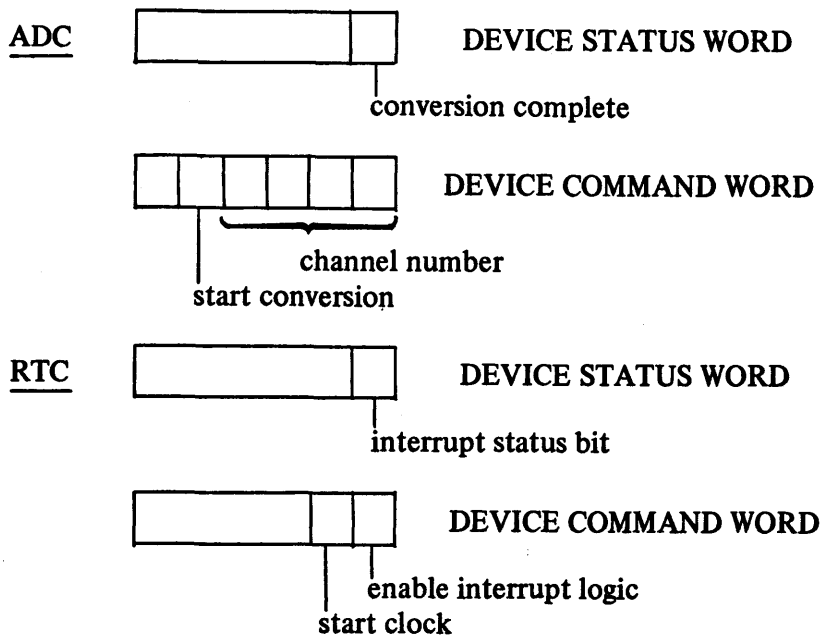


Figure 3-6. Program synchronisation using a real-time clock

EXAMPLE OF I/O USING A REAL-TIME CLOCK

An on-line data acquisition system is based on a microprocessor-controlled multichannel analog-to-digital converter. A 50Hz real-time clock is used to control the sampling rate of the system. In one application, the analog signals on channel 2 and channel 4 are sampled, digitized and stored in memory at two-second time intervals. The simplified flow-charts shown in Figure 3-6 illustrate the program flow during data collection. The device status words and device command words for the analog-to-digital converter (ADC) and real-time clock (RTC) are given below:



INTERRUPT SERVICING IN A MULTIPLE INTERRUPT SYSTEM

The simple interrupt servicing procedure described above is only appropriate in systems which have a single I/O device generating interrupts. Many microprocessor systems have more than one source and more than one type of interrupt. Three main types of interrupt may be defined as:

(i) *External interrupts* generated from one or more I/O devices.

(ii) *Internal interrupts* generated by the microprocessor system itself to indicate the occurrence of particular conditions or errors (e.g., power failure, system malfunction, transmission error).

(iii) *Simulated interrupts* generated by software to assist in program debugging or interrupt service testing.

The different sources of interrupt will have different servicing requirements. Some will require immediate attention; others will accept a delay whilst the task in hand is completed. The interrupt service procedure must therefore:

(i) Differentiate between the various interrupt sources.

(ii) Determine the order in which interrupts are serviced should more than one source of interrupt require attention at the same time.

(iii) Save and restore the contents of the registers of the microprocessor to assure program continuity during the servicing of multiple interrupts.

Recognising the source of interrupt. Some microprocessors have several interrupt request lines, each with its own unique interrupt trap address. The recognition problem may be solved by assigning only one source of interrupt to each line. This approach is commonly used to differentiate between internal, external and simulated interrupts.

In the majority of microprocessors several I/O devices must use the same interrupt request line. In these systems, two methods of recognising the source of interrupt are commonly used:

(i) *Device polling.* The interrupt causes a jump to the interrupt service program via the interrupt trap address as described earlier. The initial section of the service program checks the status word of each I/O device in turn to determine which has caused the interrupt. Figure 3-7 illustrates the program flow of a typical service program for three I/O devices. The *interrupt status* bit, which indicates whether an I/O device has generated an interrupt

request, is checked for each device in turn. The device status word is read into the status register of the microprocessor and a jump is made to the associated device service program if the bit is set. Device recognition by device polling is performed in software.

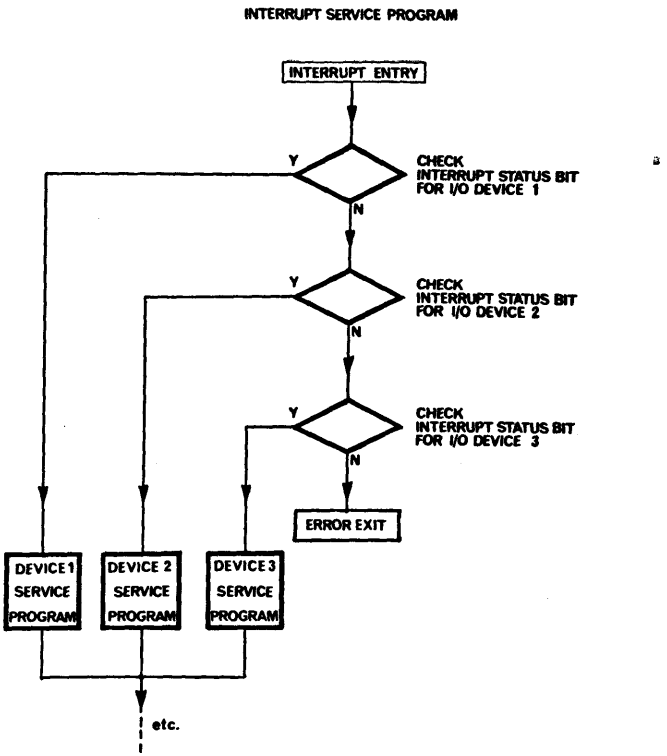


Figure 3-7. Device polling

(ii) *Vectored interrupts.* In the vectored interrupt microprocessor system, the interrupt control logic within the processor recognises the interrupting I/O device. Each I/O device is assigned a unique *device interrupt address* (not to be confused with the device address defined previously). On recognising an interrupt request, the interrupt control logic requires the interrupting I/O device to transmit its device interrupt address to the microprocessor. This address is then used to generate a unique interrupt trap address for the device. The trap addresses are usually located sequentially in program memory and form the *interrupt vector.*

Each location in the vector contains the start address of a device service program. The contents of the interrupt vector defined by the particular interrupt trap address are loaded into the program counter and program control is automatically transferred to the correct device service program. The process is simplified in some microprocessors by using the interrupt trap addresses as the device interrupt addresses.

In some systems, instead of transmitting an address, the I/O device is required to transmit a single byte instruction to the microprocessor after the interrupt request has been acknowledged. The interrupt control logic automatically loads the instruction code into the instruction register and normal microprocessor operation resumes by executing this instruction. Interrupt vectoring is achieved by using a special-purpose single byte jump instruction which derives the jump address from a part of the instruction code itself. Using the device interrupt address to specify this bit field, a unique jump address is defined for each I/O device. Device recognition by interrupt vectoring is performed in hardware.

Interrupt priority schemes. With several sources of interrupt there is always the possibility of one or more interrupt requests occurring during the servicing of an earlier interrupt request. In the simpler interrupt systems, the interrupt mask bit is automatically set when the first request is recognised. Subsequent interrupt requests join a queue. They wait until the service of the first interrupt has been completed before they can in turn be recognised and serviced. The order in which the queued interrupts are recognised is a critical factor in determining the time delay before service. The order or *priority* is dictated either by software or by hardware.

(i) *Software priority*. After recognising an interrupt request, the service program polls the I/O devices in an order which determines the interrupt priority of each device. Thus the highest priority devices, which are polled first, are serviced first. Figure 3-8 illustrates the program flow during interrupt servicing with priority determined by software.

(ii) *Hardware priority*. The interrupt control logic of the microprocessor sends out an external signal to control the interrupt

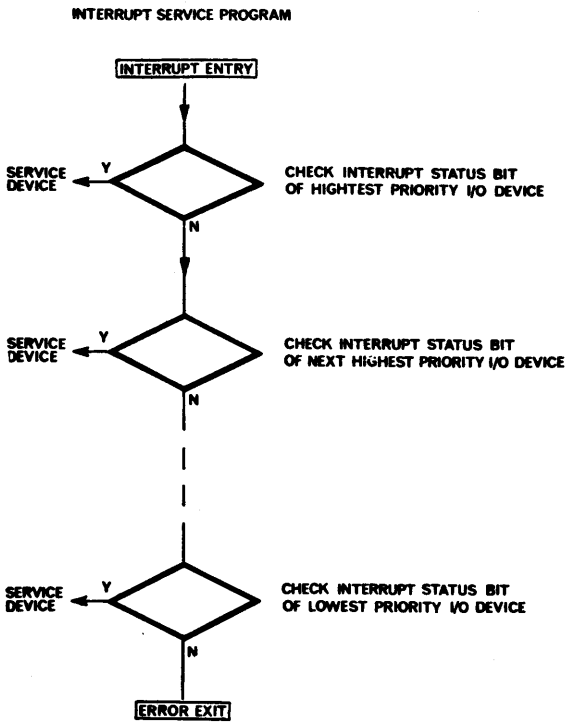


Figure 3-8. Software interrupt priority scheme

request logic in each of the I/O devices. The control signal, which always reflects the state of the interrupt mask bit, passes through each device in turn as shown in Figure 3-9 (the daisy-chain structure is discussed in Chapter 4). If the mask is set, the signal will prevent all devices from generating interrupt requests. If the mask is reset and the signal arrives at a device which has no interrupt request pending, the signal is simply passed on to the next device. When the signal arrives at a device which is waiting for interrupt service, the interrupt logic in the device prevents the signal from passing on to the next device and generates an interrupt request itself. The position of a device along the control line will determine its interrupt priority. Thus, when more than one device awaits interrupt service, the device which receives the control signal first will be serviced first. Figure 3-10 illustrates the program flow during interrupt service.

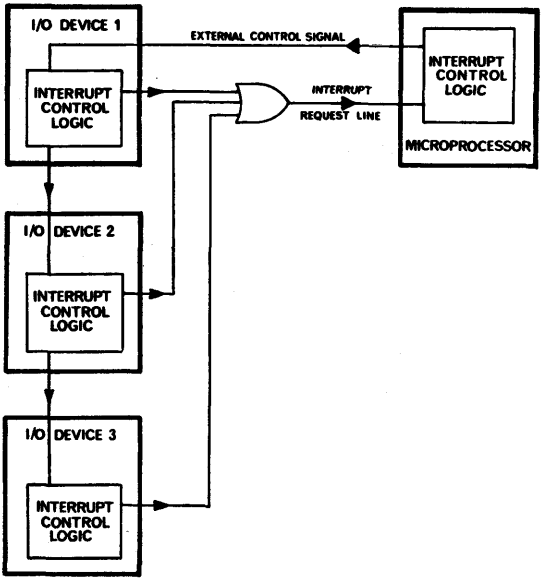


Figure 3-9. Hardware interrupt priority scheme

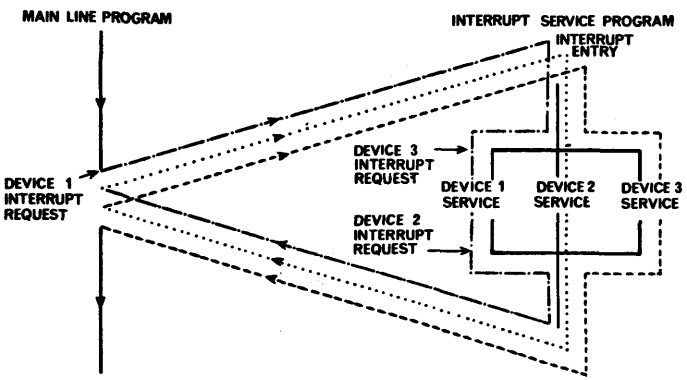


Figure 3-10. Program flow during hardware priority interrupt servicing

Both of these simple interrupt priority schemes are slow to respond to a high priority interrupt if it occurs during the servicing of a low priority interrupt.

More sophisticated schemes for software control of interrupt priority are used in microprocessors which have separate interrupt

mask bits for each of several interrupt request lines or for each of the interrupting I/O devices. In the latter case, the mask bits are often included in the interrupt logic of the I/O device itself. By setting and resetting the individual mask bits under program control, a number of schemes are possible in which interrupt priorities are changed during program execution.

Many microprocessors have two interrupt request lines. One line has a conventional software controlled mask bit whilst the other is permanently enabled. This *non-maskable interrupt request line* has the highest interrupt priority and is used in applications which require immediate service at all times under all circumstances. An example of this would be the orderly shut-down of the microprocessor system following detection of a power failure. The simulated interrupt also has no mask bit, but since it is generated by a program instruction it has the lowest interrupt priority.

In some vectored interrupt systems, the interrupt priorities are automatically defined and controlled by the interrupt control logic of the microprocessor. After an interrupt request has occurred and the device interrupt address has been transmitted to the microprocessor, the address is compared with a multi-bit interrupt enabling mask. If the device interrupt address is equal to or less than the mask, the interrupt request is recognised, the mask is

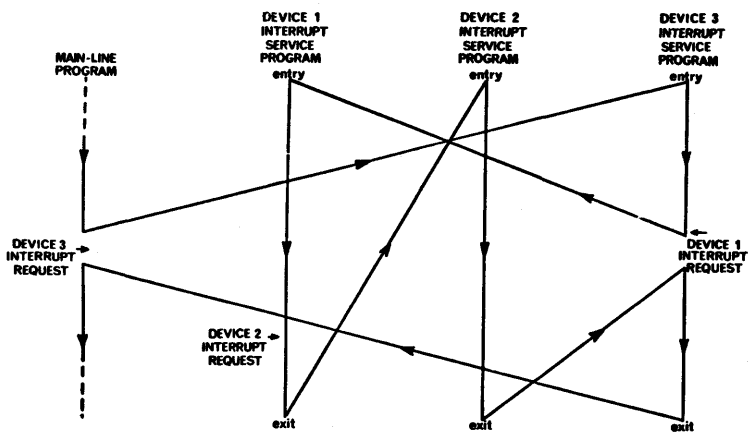


Figure 3-11. Program flow during vectored automatic priority interrupt servicing

forced to a value that is one less than the device interrupt address, and device servicing begins. If the device address is greater than the enabling mask, the interrupt request is queued. Only interrupts from a device with a lower device interrupt address to that of the device currently being serviced are recognised. Thus the device interrupt address determines the device interrupt priority; the lower the address, the higher the priority. The enabling mask can be initialised or modified under program control. Figure 3-11 illustrates the program flow during interrupt servicing. Here the interrupts are nested to ensure rapid service of high priority interrupts even if they occur during the servicing of a low priority interrupt.

MAINTAINING PROGRAM CONTINUITY DURING MULTIPLE INTERRUPT SERVICE

In general, the problem of maintaining program continuity in a multiple interrupt environment is similar to but more complex than that described previously for the single interrupt case. In particular, where nesting of interrupts occurs, provision must be made to save (and subsequently restore) the contents of all important microprocessor registers, including the return address held in the program counter, for each of the different interrupt requests which are currently being serviced. Each level of interrupt service requires its own unique storage locations in which the information can be saved. The save and restore operations are implemented in one of three basic ways:

(i) *Program-controlled save and restore.* The information is transferred to a unique area of memory under program control in a non-interruptable section at the beginning of each interrupt service program before the interrupt mask is reset to allow recognition of further higher priority interrupts. The information is similarly restored in a non-interruptable section at the conclusion of each service program. In some microprocessors, the stack pointer is automatically advanced when an interrupt request is recognised and the programming can be simplified by using the stack as the storage area.

(ii) *Automatic stacking.* The interrupt control logic auto-

matically stores the information onto the memory stack and advances the stack pointer whenever an interrupt request is recognised. On completing the interrupt service, a special-purpose "return from interrupt" instruction restores the information and decrements the stack pointer appropriately.

(iii) *Special-purpose architecture.* The microprocessor is provided with several sets of processor registers and a pointer register to indicate which set is to be used during the current program execution. The pointer register is automatically incremented on recognising an interrupt request and decremented on completing the interrupt service. In some microprocessors, these sets of registers are internal to the microprocessor chip. In others, the microprocessor uses different workspaces in memory instead of different sets of internal registers.

The speed of response to an interrupt request, once it is recognised, is mainly determined by the time required to perform these save operations. Those microprocessors which avoid the need for information transfer provide the most rapid interrupt service response.

An example of a microcomputer using multiple-priority interrupt-controlled input-output is given in Chapter 9.

DIRECT-MEMORY-ACCESS I/O

Some I/O devices require data transfer at rates which are too rapid to permit any type of input-output which is under control of the microprocessor itself. In these cases, the information must be transferred directly between the I/O device and the memory of the microprocessor system without microprocessor intervention. The technique is called *direct-memory-access* or *DMA*. The data transfer is controlled by a dedicated high-speed logic circuit (the direct-memory-access controller) which is capable of operating at higher speeds than the microprocessor. During DMA data transfer, the microprocessor must relinquish control of its memory and allow the DMA controller to take over. There are several ways in which the DMA controller can gain control of the memory:

(i) *Processor halt.* An external control line initiates an orderly halt in the operation of the microprocessor following

completion of the current instruction. Since the memory control signals of the microprocessor are disabled in the halt state, the DMA controller can take over and initiate data transfer. After the DMA input-output has been completed, the controller resets the halt control line, the microprocessor resumes normal operation and execution of the next instruction commences.

(ii) *Cycle-steal.* External control lines initiate a pause in microprocessor operation by suspending instruction execution within the instruction cycle. The processor clock is halted and the memory control lines of the microprocessor are disabled. The DMA controller takes over and "steals" several machine cycles to allow data transfer to take place. On completing the data transfer the pause control lines are reset, the clock restarts, and the instruction cycle continues to complete the execution of the instruction. The only result of "interrupting" the instruction execution is to extend the apparent execution time.

Microprocessors which use dynamic memory (see Chapter 6) on the processor chip have a limit to the number of machine cycles which may be stolen in this way if no loss of internal status is to occur. Input or output of a long block of data may require several separate cycle-steal operations for completion.

(iii) *Memory-sharing.* The memory is only accessed by the microprocessor at specific times during the basic machine cycle. At other times it is available for use by other devices. By synchronising the DMA controller operation with the processor clock, DMA data transfers can be interleaved with the normal microprocessor/memory data transfers within the basic machine cycle. Interleaved DMA has no effect on the operating speed of the microprocessor.

EXAMPLE OF DMA INPUT-OUTPUT

A microcomputer is linked to a large computer system via a high-speed communication channel. Blocks of data are transferred from the memory of the microcomputer to the communication channel by direct-memory-access. The main-line program, which communicates with the DMA controller using program-controlled I/O,

specifies the data block and initiates the DMA operation. The DMA controller then halts the microprocessor and organises the data transmission from the memory to the channel. The simplified flow charts in Figure 3-12 explain the sequence of operations leading to the transfer of a block of data.

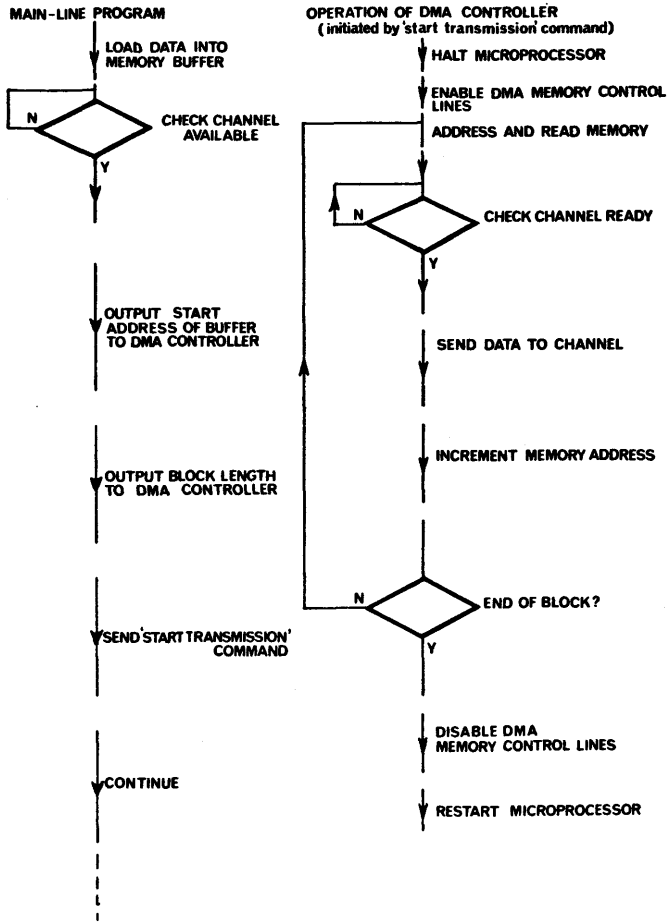


Figure 3-12. DMA data transfer

SUMMARY

The input-output scheme of a microcomputer is of great importance in the overall system design: a good scheme can give significant savings in both hardware and software. For this reason,

a thorough understanding of I/O techniques is one of the cornerstones of successful microcomputer design. Most of the material covered in this chapter is applied in a more practical sense in Chapter 8.