7.1

Introduction

Modern digital systems including personal computer are designed around the central processing unit, like a microprocessor, by interconnecting them with a number of Input/Output (I/O) devices. An I/O device needing a processor's attention can interrupt the processor by sending a hardware signal. The processor in response to that signal provides the necessary service to an 1/O device by execut ing the special service routine designed for the device. The same situation might arise for executing a special service routine due to an internal condition within the microprocessor or due to execution of some software instruction. The above-mentioned situations fall under the events called interrupts, and special service routines are called Interrupt Service Routines. The 8086 supports all kinds of interrupts

Including hardware, exception and software interrupts The processor uses 1/O instructions to communicate directly with the 1/O devices. The 8086 supports IN and OUT instructions for communicating with 1/O devices. The IN instruction is used to read the data from the input device, whereas the OUT instruction is used to write the data to the output device. Reading the data from keyboard and sending the data to the printer are examples of input and output respectively.

7.2

What is an Interrupt?

An interrupt is an event caused by sending a signal from an external device or due to an exceptional code tion resulting from the execution of some specific instructions or due to execution of a software instruction Whenever such event occurs, the processor upends in normal activities, saves its current status and jumps to special service routine called Interrupt Service Routine (ISR), Alter executing the ISR, the original scams of the processor is used and the suspended activity is resumed from the point from where it was supended,

Various procenon support the interrupt mechanism in various forms. Generally, on the basis of the interrupt source, we can classify interrupts in three different categories as follows:

1. Hardware interrupts.

2 Exceptions. 3. Software interrupts

A hardware interrupt is caused by external devices by sending a signal through the system bus to a proces or on one of its interrupt pins. Processor support one or more interrupt pins. This is an asynchronous cvent which occurs at any time while the processor is performing any normal activity. The processor in response to a hardware interrupt cecutes the ISR to provide the required service to the device and then

resume the normal activity.

An our prion is a condition that occurs while executing some special instruction. For cumple, an exception occurs when the division instruction performs the division by 0. In that case, the processor jumps to a special service routine (in the form of ISR) designed to handle the situation. After providing the required service, the processor resumes its normal activities. An exception occurs only if a specific condi ton occurs at the time of executing a specific instruction which is placed in a program at a known place A software interrupt occurs due to the execution of a program instruction designed to call a system service. The processor provides the required service by executing the ISR designed for it and then resumes its normal activities. The software interrupts are designed to provide system services and that is why sometimes they are also referred to as system calls. The software interrupt instructions are placed in a program at faxed points and once execute causes the interrupt. For that reason, a software interrupt is considered as an synchronous event.

Various processors also provide the facility to enable or disable the hardware interrupt facility using

the interrupt enable and disable instructions. Based on whether the interrupt can be enabled or disabled. we can classify hardware interrupts as follows:

1. Maskable interrupts: The maskable interrupts are masked (or disabled) by executing the interrupt disable instruction

2 Non-maskable interrupts: The non-maskable interrupt cannot be masked (or disabled) by any means and are normally used in emergency situations,

The processor normally checks for the interrupts at the end of each instruction cycle. If any valid interrupt is requested, then the processor performs the necessary steps to process the interrupt and executes the corresponding ISR. After executing the ISR. it resume its normal activities. Remember that the ISR is a special routine which performs the task necessary to provide the service for a requested interrupt. Sometimes, an ISR is also referred to as an interrupt handler.

We will learn about the interrupt mechanism supported by the 8086 processor in the next section

in detail.

7.3

The 8086 Interrupts

The 8086 processor supports all three kinds of interrupts discussed in the previous section through its

versatile interrupt mechanism. This section focuses on the details of how the 8086 interrupt mechanism works. It discusses different ways to interrupt the 8086 processor and enable disable instructions,

RET instruction to return from interrupt, the BOS6 mpoe to an interrupe thesgh predefined dhe vector table which provides the address for the 15Ra o be executed

Interrupting the 8086 fetent urces. The 80B6 processor contains

The spent supports hardware, accepting and of f which it pe pins for the hardware interrupt, namely NMI

wo

INTR. The NMI is a non-maskable interrupt, where the INTR madable meme An maldesi can send an interrupt signal to the 8086 on one of these pins Seme eor may occur while executing some specific

instructions. This is known as to temple, during the seduction of the DIV instruction, if the division occurs by then there called Die Overlo cun. The 8086 generates the type 0 interrupt in this case to handle them the quotient is too large to fit in the AL or AX during division operation, then also the 806 as the type 0 interrupt. The other 8086 instructions that can cause the exception interese IDIV and INTO The third type of the interrupt source for the 8O86 processor is the software interrupt. A software

interupt is cased by executing the INT instruction. The software interrupts ate normally ued to e the system services from the user programs and are aluo called mtem call. We have always nt functions of DOS service 21h in the programs in previous chapters

Enabling and Disabling Interrupts

The 106 contains the interrupt flag (IF) as part of its flag register. If the interrupt flag IF is the hardware, interrupt received on the maskable interrupt pin INTR are disabled and the 806 sje che interrupt requests received on the INTR pin, In order to receive the interrupts on the INTR pin. we have to enable it by making IF 1. The 8086 provides the CLI and STI instructions for enabling nd disabling the maskable interrupt (INTR). These instructions will not affect the non-maskable mp (NMI)

Cu Instruction

It wkown as the clear interrupt flag instruction. The syntax of the CLI instruction is as folloWS CLI

IF 0. clear Interrupt flag This instruction sets the interrupt flag IF to 0. This disables the maskable interrupts in S06, implying the interrupt request on the INTR pin is not accepted. This does not affect the non maskabLe inter rupu received on the NMI pin

STI Instruction

It is known as the set interrupt Hag instruction. The syntax of the STI instruction in film STI 1 IF + 1, set interrupt flag T is ititruction sets the interrupe flag IF to 1. Setting the interrape lag to l enables the interrupts on the

INTR pin of the 8086. The B0S6 starts accepting the interrupts on the INTR pin from the instruction

IRET Instruction

This is known as the Interrupt Return (IRET instruction. The IRET instruction is used from the ISR to the interrupted pncess, and it is written at the end of the ISR. The syntax of the 18

insurrection is as lows

ZRET (20) ISS:SP). SP SP + 2

(CS) (S SSP). SP SE 2 (Flags) - (SS:SP), SP - SP + 2

return from the interrupt

pop the IP

pop the cs pop the flag

IRET pops the IRS and lags which were saved by the 8086. The execution of IRET causes the e ton to be resumed back from the interrupted point in the interrupted

highlights the use of the IRET instruction in more detail.

process. The next subscene

Nete Remember that the RET instruction is used at the end of the procedure. The IRET instruction at the end of the ISR The RET instruction can be near or far. The RET instruction only per IP (for near RET and IP and CS (for far RET, whereas the IRET instruction pops IP, CS and flag

Steps to Respond an Interrupt In general, microprocessors perform the following steps to process a valid interrupt request:

1. Stop the current process and save the current status.

2 Jump to an ISR using its address Execute the ISR process.

4 Restore the status, return from the ISR and resume the

interrupted The current status includes the flags, return address (address of the instruction from where the interrupted

process will resume) and the registers. In general, a part of status is saved and retrieved automatically by the processor itself and the rest is left to the user to perform as part of writing the ISR. The minimum part of the status all the processors save and retriee automatically is the return address. For the rest of the status, different processors take different approaches. The computation or acquiring the ISR address also

save from processor to processor. Let us understand these steps specific to the 8086 processor

The 8086 performs the following six steps when a valid interrupt is received. The steps are lised with reasons

1. Decrement abe stack pointer (SP) h 2 and push the flag register on the sac

SP SP - 2 (SS:SP) Flags

The Bags are part of the current status and the 8086 automatically saves them onto that the interrupted process resumes with the same status of the flags. The IRET instruction *

the stack

the end of the ISR restores the flags 2 Cear she app is the flag register to double the interminar on the INTR pin.

IF 0 The clearing of IF disables the interrupts on the INTR pin. The 8086 automatically disable the maskable interrupts by making IF = 0. This step is performed to allow the ISR to Executed without being interrupted by the other lower priority interrupt. The IRET restores the original lags which enable the maskable interrupts again after completing the ISR. If you want to allow the other interrupt to interrupt the current ISR. enable the interrupts on the INTR by executing the STI instruction at the start of the current ISR. Another reason for disabling the maskable interrupt is to avoid the racing. The racing means interrupting again and again. If a device holds the interrupt signal on the INTR pin continuously, the 8086 is

interrupted again and again. 3. Clear the trap flag in the flag register to stop single-step execution.

TF 0 The trap flag is used by the debugging tools to allow single-step execution. The ISRs are well-tested routines and need not be executed in the single-step mode. The 8086 automatically disables the single-step mode by making the TF = 0. The IRET restore the original flags

4. Decrement the stack pointer (SP) by 2 and push the content of the CS register onto the stack

SP - SP- 2 (SS:SP) (CS)

The details for both steps 4 and 5 are covered together in step 5. 5. Decrement the stack pointer (SP) by 2 and push the contents of the IP register onto the stack.

SP + SP - 2

(SS:SP) (IP)

The ISR is a far procedure as it is outside the 64 KB range of the process which is interrupted. In order to return back correctly to the interrupted process, we need to store both the segment base and the offset value of the instruction in the interrupted process from where execution is to be resumed after completing the ISR. The 8086 automatically saves both CS and IP to store the return address. The IRET instruction restores them to resume back from the same point. 6. Make an indirect far jump to an Interrupt Service Routine by loading CS and IP by its address from the rector table

After completing steps I to 5, the next important step is to get the address of the ISR and load it in the CS and IP to jump to the ISR so that the execution can begin from the first instruction of the ISR. The 8086 has a special data structure called the interrupt vector table which contains the addresses of ISRs. This is discussed in the next subsection,

An ISR implements the desired service to be provided in response to the requested interrupt. In general. the ISR at the start stores the registers onto the stack as part of the current status as the registers are not automatically saved by the 8086. Then it performs the desired tasso provide the service. Finally it restore the registers back and returns to the interrupted process using the IRET instruction Figure I thows the entire 8086 interrupt processing which includes all the steps discussed above.

Vector Table

When a valid interrupt comes, the 8086 performs the series of steps in which the last step is to load he starting address (IP and CS) of the corresponding ISR to make a far jump to the ISR. The data structure which stores the addresses of the SRs is called interrupt vector table or interrupe printer table. We will now onwards refer to it simply as rector table. Each entry in the vector table consists of four bytes - 2 bytes for IP and 2 bytes for CS. An address stored in an entry of the vector table is call interrupt vector or interrupt pointer. The addresses in the vector table include both IP and CS. s the ISRS are far procedures and we need to perform a far jump to jump to an ISR. The total number of entries in the vector table is 256 as the 8086 recognizes 256 interrupts. Each interrupt in the 8086 is identified by an 8-bit number N called interrupt type and ranges from 0 to 255. The vector table occupies first 1 KB memory as the total size of the vector table is 256 x 4 = 1024. Figure 2 shows the

8086 interrupt vector table.

The interrupt type number Is multiplied by 4 to get the starting position in the vector table from where the four bytes give the starting address of an ISR. The first 2 bytes give the IP and the next 2 bytes give the CS. Let us take an example. The NMI is the type 2 interrupt and hence when a valid signal comes on the NMI pin, the 8086 saves the flags, clears IF and TF, saves IP and CS, and multi plies interrupt type N = 2 by 4 which results in 00008h which gives the starting position in the vector table to get the address of ISR for the NMI interrupt. The locations 00008h and 00009h give the IP and 0000Ah and 0000Bh give the CS. The interrupt types are discussed in more detail later.

Check Points

We know that

an interrupt in an en caused by a hardware w alan enor condition or the execution of a program instruction. the IDIV and INTO instruction cause an exception (e, internal interrupts The software internals are caused by accor

can

ing the INT instruction. • the STI and CLI instructions are used to unumak

and mask the interrupt on the INTR pin. the IRET instruction is used at the end of the ISR and returns from the interupt.

when a valid interrupt comes, the BO86 save the flag register, clears the IF and TF, saves the P and CS and makes a far jump to the ISR the vector table stores the starting address of the ISRs and occupies first 1 KB memory

interrupts are categorized as hardware, excel toni and software interrupts. The hardware interrupts can be maskable or non-maskable Interrupt Service Routine ISR) is far pr procedure executed in response to a valid interrupt interrupts.

the

frequent. It implements the serve to be pro video and n also known as interrupt handler. • the 8OS6 supports hardware, exception and

software interrupts. It has two interrupt pins, the INTR and NMI pin to receive hardware

7.4 Writing a Simple ISR

The ISRs implement the various services to be offered by means of hardware, exception and software intel rupe. Normally, in a digital system based on a processor like 8086, these services are loaded at the time ef

the w oman up and their addresses are in the vector table Then by means of sending an intn rupe the required service is executed. How can we write our cwn ISR and test it! le is a simple at writing and calling a procedure. Let us understand the writing and executing an ISR through an cumple The ISR a written as a far procedure with an IRET instruction at the end of the procedure. We cannot execute the ISR by just calling it Ide a simple far procedure. We have to attach the ISR to an interrupt type number and load its address including the w ent base and the offset into the vector table at the position starting from 4 interrupt type number. Then by ercating the INT

imitruction with the card type number, we can cut our 15. Program 7.1

As can be seen from Program 7.1, the ISR is written as far procedure with name prt_mes. The ISR simply prints the message stored in the data segment. Observe that the ISR stores and retrieves only the AX and DX registers. The last instruction in the ISR is the IRET which returns to the interrupted program. The mainline program first initializes the data and stack segments. The program associates the ISR to the interrupt type number 5, hence we need to save the offset address of the ISR at the loca tion 00014h and the segment address of the ISR at the location 00016h in the vector table. The vector table starts from location 00000h, so the segment address of the vector table is 0000h. The program initializes the ES register with the 0000h, which means the vector table is defined as the extra c The offset address of the ISR and the segment address of the ISR are saved at offset 0014h (

20 = 14h) and 0016h (4 x5 +2= 22 = 16h), respectively, in the extra segment (vector table) bra following instructions:

MOV WORD PTR ES:0014h, OFFSET prt mes MOV WORD PTR ES:0016h, SEG prt_mes

Once these instructions are executed, the ISR is linked to the interrupt type number 5, as its adi stored in the vector table at the position for the interrupt type 5. The next instruction in the mine program is INT 05h which uses the software interrupt. The 8086 saves the flag register, den and TE saves the CS and IP of the next instruction and multiplies type number 5 by 4 reultine in 20 (14h). It loads the IP from the 0000: 0014600014 h) and CS from the 0000:0016h (OĞLGH) and makes the far jump to new values of CS and IP The new values of CS and IP point to the IS prt mes. The ISR prt_mes executes and prints the message and it returns back to the maint program which then is terminated. After assembling and linking the Execution of the above program results into the printing of the message Interrupt is processed. The appearance of this message indicates that our ISR is successfully linked to the interrupt type number, and the vector table

entry for interrupt type is pointing to ISR prt_mes We have studied the procedures in the last chapter. Now, we compare the procedures and ISR

1. The procedure can be near or far, whereas the ISR is always far. 2. The procedure ends with the RET instruction, whereas the ISR ends with the IRET instruction 3. The procedure is called using the CALL instruction, whereas the ISR is called by causing the

hardware, exception or software interrupts.

4. The CALL instruction itself contains the address of the procedure, is available from the vector table.

once the vector table entry for the ISR made, the ISR can be executed by cucuting the INT instruction with the appropriate type number

procedures and ISRs differ in their calling and returning mechanism

whereas the address of the ISR

5. The procedure does not need initialization, whereas the ISR needs initialization of the vector table

Check Points

We now know that

an ISR is written as a far procedure with IRET at the end to return from an interrupt.

the ISR should be associated with the inter

rupt type number

the office and the segment address of the ISR are stored into the vector table at the position

corresponding to the interrupt type number.

7.5 Software versus Hardware Interrupts

We know that the 8086 supports all three kinds of interrupts: hardware, exception and software. they are caused by external hardware device by and

hardware interrupts are external interrupts

as

ing signal. The exception and software interrupts are internal interrupts as they are caused in

hasuch instruction section. The exception interrupts are very similar to the software interrupts an hey are also caused by program instruction. The difference is that a software interrupt always when the INT instruction executes, whereas an exception interrupt occurs only when a specific condi tion like divide by 0 ia met during the execution of a specific intction like DIV. f the constitution not me, the instruction performa les operation in normal course. What happens if the processor receives more than one interrupt request at a time The fact is that the processor can handle only one interrupt at a time. In the case of multiple interrupts, the solution is to handle them one by one. But in what order multiple interrupts will be handled To resolve this issue, the processor supports a priority order. This section discusses the software and hardware interrupts and the interrupt priorities with respect to the 8086 processor

Software Interrupts

The 8086 provides the INT instruction for causing the software interrupts which are used to all the system services written in the form of ISR. We have already used the various functions of service 21h of the DOS. Before going into further discussion, let us understand the functioning of the INT instruction

INT Instruction

The INT instruction is used to generate the software interrupt. The syntax of the INT instruction is as follows:

INT N

generate the interrupt type number N

Here N denotes the interrupt type number and it ranges from 0 to 255 (8-bit), When an INT Instruction is executed, the 8086 pushes the flag register onto the stack, clear the IF and TF pushes the CS and IP of the next instruction and makes a far jump to the ISR for the interrupt type number N by getting its address from the vector table. IP is loaded from location 0000:Nx4 and CS is loaded from location 0000:Nx4+2 in the vector table. The following example clarifies the concept

INT 05h PUSHF

IF + 0, TF + 0

PUSH CS PUSH IP :

; (IP) word from location 0000:0014h (5 x 4 20 =14h) : (CS) word from location 0000:0016h (5 x 4 + 2 =22 16h]

The locations 0000:0014h and 0000:0016h in the vector table contain the address (IP and CS) of the

ISR for the interrupt type number 5. The execution begins from the first instruction of the ISR and completes when the IRET instruction is executed. It pops the return address and Hage, and the care tion resumes from the next instruction in the interrupted process The software interrupts are used normally to access the services provided by the system. The IBM PC compatible machines use DOS as the operating system and the 8088 is the processor. The 8088 is the 8-bit version (the external data bus is 8-bit, but the internal

the 8086 and is completely software compatible to the 8086 processor. The high-end PC uses the Intel higher processors in 80X86 series like 80286, 80386, 80486 and Pentium. They all are down ward compatible to the 8086 processor. This means that we can run the 8086 program on any PC machine supporting DOS. The system services are provided an PC machines in the form of BIOS (Basic Input Output System) and DOS services. We can use these services using the INT instruction with appropriate type number assigned to each service. For example, the video service is assigned number

in the BIOS. Using service 10h, we can access the various video functions. The service 21h is the major DOS service. The BIOS and DOS services are discussed with examples in the next section

Hardware Interrupts The 8086 contains two interrupt input pins, namely INTR and NMI. A device in the system

send a signal on one of these pins to cause a hardware interrupt. The interrupt on the INTR pin can be disabled using the CLI instruction as it is maskable, whereas the NMI pin is non-maskable and cannot be disabled by software or hardware means The NMI pin is normally used for the emergency services like power failure. The power failure is detected by the internal circuitry in the system which sends a signal on the NMI pin which executes

the ISR for type 2 to save the important data. This takes very little time during which the power is supplied from the internal capacitors as the capacitor can store the charge for some time after the power failure Digital systems like PCs support multiple hardware devices and hence they are not directly can nected to the INTR pin for sending the interrupts. A programmable interrupt controller chip, like 8259A, is used to manage the interrupts from multiple device. Each device is allocated the IRQ (Interrupt Request) line to which the device sends the interrupt to get the service. The controller supports 8 or more such lines. The 8259A supports 8 such lines and using them in the cascading mode the number of lines can be extended to 64. The controller manages the interrupt requests coming from the various devices connected on IRQ lines and using priority resolution sends one of them to the 8086 using the INTR pin. If the interrupts on INTR are enabled then the 8086 generates two interrupt acknowledgement cycle by inserting a signal on the interrupt acknowledge (INTA) pin. The first cycle informs the controller to become ready and during the second cycle controller sends the interrupt type number to the 8086, The 8086 uses type number to get the address of the ISR for the device from the vector table. The IBM PC-compatible machines reserve certain type numbers for the services to the hardware devices. We will learn about them in the next

section in detail.

One important point to be noted is that the same ISR can be executed by using both hardware and software interrupts in many casc. For example, the NMI is non-maskable interrupt and is assigned type 2 in the PCs. The ISR for type 2 can be executed by sending a signal on the NMI pin or executing the INT 02h instruction. This facilitates the testing of the ISR for the hardware interrupts using the software interrupts. The ISR is first tested using the software interrupt by calling the INT instruction, and if it works correctly then it can be installed in the system permanently to provide the service using the hardware interrupt.

Interrupt Priorities

more than one interrupt request comes by means of hardware, exception or software, the 8086 uses the If

internal priority to resolve the issue. The interrupts are served according to priority levels. The highest priority first and the lowest priority last. The 8086 interrupt priority levels are given in Table L.

Table 1 Interrupt priority levels in 8086

Interrupt Type

SExceptions (DIV and

Software (INTERN)

Non-maskable (NMI) Maskable (INTR) Single step (TF = 1)

Priority Level

Highst

Lowest

As can be seen from Table 1, the internal interrupts (exception and software) are highest priority interrupts followed by the non-maskable interrupts, the maskable interrupts and the single-step inter rupts with the lowest priority. The exception and software interrupts are at the same priority as they can never occur at the same time. The reason for this is that both are internal and are generated by program instructions. The processor can execute only one instruction at a time. The priorities are assigned such that a lower-priority interrupt cannot interrupt the serving of a higher-priority

interrupt as the 8086 disables the maskable interrupt and the single-step interrupt by clearing IF and TF while responding to the interrupt. The IRET at the end of the ISR restore the IF and TF which enables them again to allow the lower-priority interrupt to be served which occurred while serving the higher

priority interrupt. The 8086 checks the interrupts at the end of every instruction cycle and serves them in the order of priority. If only one of them is present then the 8086 completes the current instruction, performs the predefined steps, executes the ISR for the requested interrupt and resume back the original process. If more than one interrupt are present then the 8086 serves them in order of priority Let us understand this situation by an example. Assume that while executing the INT instruction, the interrupt request is received on the INTR pin which is currently enabled. The software interrupt has higher priority than the interrupt on INTR. Hence, the 8086 performs the predefined steps and jumps to the ISR for serv ing the software interrupt. This automatically disables the interrupt on INTR and does not allow the interrupt on INTR to disturb the service. When the service for the software interrupt is completed, the IRET instruction restores the flags which enable the interrupt on INTR again. The 8086 then serves the interrupt on INTR by performing the same process. This needs the external device to hold the signal on INTR until it gets acknowledged.

Remember thar the NMI is used for emergency service and is non-maskable. It should be executed immediately to take quick action. Even though it is given lower priority s compared to internal inter rupts, the above purpose is not violated as the clearing of IF while responding to an interrupt does not disable the interrupt on NMI

Check Points

We now know that

. the exception and software interrupts are called internal interrupts as both are generated

internally by the program instructions. the software interrupts are used to call the system services using the INT instruction with the interrupt type number

the IBM PC-compatible machines provide

BIOS and DOS services accessible using soft

ware interrupts the hardware interrupts are received on one of

the 8086 pins-NMI and INTR

7.6

Interrupt Types

• the programmable interrupt

controller chip

manages the interrupts from multiple device and sends them on the INTR pin of the 8086 one at a time. The interrupt type number 1 provided by the controller chip. the processor check the presence of interrupt at the end of every instruction cycle and pro cesses them in the order of priority if multiple

interrupts are present

The 8086 recognizes a total of 256 interrupts which are given the The interrupt type number is used to get the address of the ISR from the vector table. The hardware

interrupt type numbers from 0 to 255

interrupts are used for services to external devices, exceptions are used for handling special conditions and software interrupts are used to access system services. The 8086 (and 8088) divides the 256 interrupts

into three different sections as shown in Table 2

Table 2 The 8086 interrupt categories

Interrupt Range

0-4

5-31 32-255

Category

Dedicated functions

Reserved User service

The first five interrupts (0-4) are used by the 8086 processor itself for the dedicated functions which indude exceptions and non-maskable interrupts. The interrupt types 5-31 are reserved and 32-335 are used for the user services. The IBM PC-compatible machines use the 80X86 series of processors which use these interrupt categories for two purposes, namely BIOS and DOS services. The ROM BIOS provides device drivers for various hardware devices like keyboard, video, printer, serial ports, etc. These devices access the services from ROM BIOS using the interrupt types in the range 5-31. The range 5-31 includes both the hardware interrupts and the BIOS services accessible as software interrupes. For sample, type 09h is the keyboard hardware interrupt whereas type 16h is the keyboard 1/0 software interrupt. Whenever a key is poured on the keyboard, it sends the hardware interrupt type 09h on the INTR pin and its ISR reads the keystroke and puts its code into the keyboard buffer. The program waiting for reading the input from the keyboard uses the software interrupt type 16h to read that key code from the keyboard buffer. The operating system DOS used on the IBM PC.compatible system uses the range 32-127 from the last category for providing DOS services in the form of software inter rupts. Table 3 lists the dedicated functions, hardware interrupts and some of the BIOS and DOS inter- rupt rypes. The complete list and their uses are beyond the scope of this book.

Table 3 Selected interrupt list

Interrupt Type Number

Dedicated Functions

O1h

02h

03h

04h

Hardware Interrupts

O5h

08h 09h

Oah-Odh

BIOS Services

10h

13h 14h

16h

17h

DOS Services

20h 21h

25h

26h 27h

Function

Divide by zero

Single step

Non-maskable Break point

Overflow

Print screen

Timer

Keyboard

Hardware interrupts

Video services

Disk 1/0

Serial port I/O Keyboard 1/0 Printer 1/0

Program termination DOS services

Absolute Disk Read Absolute Disk Write

Terminate and Stay Resident

The rest of this section discusses exceptions, non-maskable interrupts and some hardware interrupts and then demonstrates some important BIOS and DOS services through programming examples.

Hardware and Exception Interrupts The interrupt types 0- defining dedicated functions and the type 5 interrupt for print screen are aplained in the following subsections.

Type 0: Divide by Zero Interrupt This is known as the divide overflow or divide by zero interrupt. We know that when the 8086

CCcutes the DIV or IDIV instruction and if the quotient is too large to fit in the AL register for 16-bit by 8-bit division or in the AX register for 32-bit by 16-bit division, then the 8086 automati ally generates the type 0 interrupt. The 8086 saves the flags, clears the IF and IF use the CS and of the next instruction and jumps to the ISR for the type 0 interrupt by getting the address of the ISR from the vector table. The locations 0000:0000h and 0000:0002h in the vector table give the IP nd CS of the service routine for the type 0 interrupt, respectively. The following division generates the type

the above cox, the service routine for the type o print the message "Divide overflow

DIV BL

Type 1: Single-Step Interrupt

The 8086 microprocessor provides the facility to execute a program in the single-step modk, that instruction by instruction, and allow the user to see the contents of registers and memory locations for car e each instruction. This feature is normally used by the debugging tools. We have a sed the

trace mode in che "debug utility of the DOS. The 8086 does it by uaing the trap flag TE Whenever the 8086 executes the instruction with TF = 1, it automatically generates the type 1 in upt after the function of the instruction. The 8086 save the flaps, clean IF and TF saves the CS zd IP of the next instruction and jumps to the ISR for the type map by getting the address of the ISR fram the vector table. The locations 00000004 h and 0000 000 in the vector table pive the IP and CS of the service routine for the type 1 interrupt. respectively. The ISR for the type 1 interrupt allows the user to examine the contents of registers and memory locations. The implementation of the type 1 service depends on the software tool. In order to provide this service, the software tool needs to set the trap flag to 1 before auction of cach instruction. The 8086 does not provide instructions to set or reset the trap flag, Wec have already discussed how to set and rea

the trap fLag in Chapter 4 while discussing the processor control instruction. The 8086 generates the type 2 interrupt when gets a valid signal on the NMI pin. The 8086 saves the flags, clears IF and TE saves the CS and IP of the next instruction and jumps to the ISR for the type 2 interrupt by getting the address of the ISR from the vector table. The locations 0000 00045 and 0000-000Ah in the vector table give the IP and CS of the service routine for the type 2 interrupt. respectively. The NMI is non-maskable and is used for emergency services.

Type 2: Non-Maskable Interrupt

Type 3: Breakpoint Interrupt The 8086 generates the type 3 interrupt whenever an INT 03h instruction is executed. A breakpoint

is useful in executing the program up to a certain point and then stop. Breakpoints are also wed

for debugging the assembly programs. The g command in DOS debug implements the breakpoint

facility. Actually, the software inserts the INT 03h instruction for each breakpoint

breakpoint automatically type 3 interrupt is generated and the execution stops temporarily. When

breakpoint is encountered, the 8086 saves the flags, clears the IF and TF, saves the CS and IP of the

so that for cach

next instruction and jumps to the ISR for the type 3 interrupt by getting the address of the ISR from

the vector table The locations 0000:000Ch and 0000000 Eh in the vector table give the IP and

the service routine for the type 3 interrupt respectively.

Type 4: Overflow Interrupt The overflow flag OF is used to store the status of the signed addition in the 8086 processor. The 8086 the OF to I if the signed addition results into overflow. For example, if the result of an 8-bit signed ion is less than -128 or greater than +127 then OF becomes 1. The 8086 INTO instruction d o ws to generate the type 4 interrupt if the OF is set to I after signed addition:

ADD AL, BL add signed number in AL to signed

number in BL

;if OF = 1, do type 4 interrupt,

otherwise NOP

INTO

The ADD instruction sets the OF to 1 if the result is out of the range. If OF = 1, the INTO instruction generates the type 4 interrupt. The 8086 saves the flags, clears the IF and TF, saves the CS and IP of the next instruction and jumps to the ISR for the type 4 interrupt by getting the address of the ISR from the vector table. The locations 0000:0010h and 0000:0012h in the vector table give the IP and CS of the service routine for the type 4 interrupt, respectively. If OF =0 then INTO simply acts as the NOP instruction and does nothing

The use of the INTO instruction and generation of the type 4 interrupt if OF = 1 is one of the ways to handle the signed overflow. Another way is to use the JO Jump if OF = 1) instruction after the ADD instruction to transfer the control to a place where the code for processing the situation is written

Type 5: Print Screen Interrupt The IBM PC compatible machines implement the screen dump utility in their ROM BIOS to print the

contents of the screen. The pressing of the PrtScr key on the keyboard generates the INT 09h which in turn (if key is PrtScn) calls this utility by generating the type 5 interrupt. The 8086 saves the flags, clears the IF and TF saves the CS and IP of the next instruction and jumps to the ISR for the type 5 interrupt by getting the address of the ISR from the vector table. The locations 0000:0014h and 0000:0016h in the vector table give the IP and CS of the service routine for the type 5 interrupt, respectively.

BIOS Services

The BIOS services include the I/O services for devices like video, disk, keyboard, printer, serial ports, and many more. For each service, it provides a number of functions. We will take the examples of only one or two functions for video and keyboard to get an idea of BIOS services BIOS provides the video services as the INT 10h. The INT 10h provides a number of functions

related to both the text mode and the graphics mode. Program 7.2 demonstrates the use of video ser

vices by displaying the message "Hello" in the middle of the screen in the MDA (Monochrome Display Adapter) mode with 40 x 25 text mode. As can be seen from Program 7.2, the program first uses the video function 0 to set the video mode. The input to the function is given in the AL register which in our case is Denoting the MDA 40

mode. Then the video function 2 is used to set the cursor to the middle of the screen of 40 column (0 to 39) and 25 rows (0 to 24). The parameters to the function are BH = page number=0, AND= two number 12 and DL column number 19. Then function 9 of DOS service 21h prints the mes sage "Hello" at the new cursor position. Finally, function I of the DOS service 21h is used to stop the program to visualize the effect. The pressing of any key then terminates the program. The BIOS provides keyboard I/O as INT 16h. The function 0 of the service 16h reads the char actress from the keyboard without echoing it on the screen and gives its ASCII value in the AL register. Program 7.3 demonstrates its usc.

As can be seen from Program 7.3. the data segment defines the buffer named buf of size 10 with 11" character as 's'. Within the code segment, the function 0 of service 16h is used to read the charac ters in a loop with each character read in the AL register placed at the next position in buf. Finally, the characters stored in buf are printed. Note that while reading the characters, they are not visible as the function of BIOS service 16h provides the character reading without echo.

DOS Services

The DOS uses the interrupt types 20h-7Fh (32-127) for providing its services. However, the DOS provides major functionality through its well-known service 21h. This service includes the func

size 10 with nationalities for Character I/O, file operations, record operations, directory operations, disk manage ment, process management, memory management, network functions, time and date operations, and other system functions. We have already described functions 1, 2, 9 and 10 from the character I/O of the DOS service 21h for character and string input/output in Chapter 3 and used in pro

the charac

Finally, the

sible as the

grams in the previous chapters. Let us explore the DOS service 21h by using some more functions

in programs.

The function 2Ch (42) of DOS service 21h is "Get time" which returns the system time. It returns hours in the CH register, minutes in the CL register and seconds in the DH register. Program 7.4 dem

onstrates it by displaying the time in hh:mm:ss format.

As can be seen from Program 7.4, it contains two procedures: bin_bcd and print. The bin_ bcd accepts the 8-bit binary number in the AL register as input and converts it into packed BCD and returns it to the AL register itself. The procedure print accepts the packed BCD in the AL register and prints it by converting its individual decimal digits into the ACSII. The mainline program calls the function 2Ch of the DOS service 21h and then prints the output given in the registers CH (hours), CL (minutes) and DH (seconds) in the format hh:mm:ss using the procedures bin_bcd and print.

The output of Program 7.4 looks as follows: Current Time is 10:07:13

The interested readers are advised to refer the IBM PC and DOS manuals or online resources to get the description of the BIOS and DOS services.

7.7 Input/Output Instructions

• 243

s6 identifies the hardware devices using port number called port addresses. The port addresses The O communicate with hardware devices using the IN and OUT instructions juit like memory ses are used to communicate with memory. Lt un uke an example of keyboard which is an derive. The system assigns a particular port address to the keyboard. Whenever a kry in pressed, Nudware interrupt 09h is generated whose service routine uses the keyboard port address with the Instruction to read the key code and place it into the keyboard buffet. The software waiting for intuit then reads that key code from the buffer using INT 16h. This means that the interrupt aike routines for the hardware interrupts internally use the port addresses to transfer the data in nuget direction. The input devices to the IN instruction to read the data from the device into the register and the output devices use the OUT instruction to write the data from the register to the device.

The 8086 supports 8-bit and 16-bit port addresses. For the 8-bit port address, the system can support maximum 256 ports, whereas for the 16-bit port address, the system support maximum 65536 ports. The 8086 port can read or write the byte or word. Let us understand the IN and OUT Instructions

IN Instruction

The I instruction is used to read a byte or word from an input device connected to a specified port address into the AL or AX register. The IN instruction is either fixed-port or variable-port. No flaps are affected by the IN instruction

The fixed-port IN instruction uses the 8-bit port address. The syntax of the fixed-port IN intre

tion is as follows:

IN AL or AX, ports copy a byte or word from 8-bit port address porta

to AL or AX The fixed-port IN instruction reads a byte or word from an input device connected to a specified 8-bit

port address pers into the AL or AX register. Following are some examples:

IN AL, 80h read a byte into AL from port address 80h. , read a word into a

IN AX, BIH

from port

address 81h

The variable-port IN instruction uses the 16-bit port address which is loaded in the DX register. The syntax of the variable-port IN instruction is as follows:

IN AL or AX, DX I copy a byte or word from 16-bit port address in the DX register to ALL or AX

given

enable-port IN instruction reads a byte or word from an input device, connected to a specified Air port address given in the DX register into the AlL or AX register, Following are some examples:

MOV DX, 8000h load the port address

IN AL, DX

read a byte into AL

MOV DX, 8080h load the port address

IN AX, DX

read a word into AX

The advantage of the variable-port IN instruction is that we can use the port address computed dynam ically in the program and loaded into the DX register. This is useful specifically when the same pro cedure is used to read from different ports in a range. In that case, only the port number is passed as parameter to the procedure.

OUT Instruction

The OUT instruction is used to write a byte or word from the AL or AX register to an output device connected to a specified port address. The OUT instruction is either fixed-port or variable-port. No fags are affected by the OUT instruction. The fixed-port OUT instruction uses the 8-bit port address. The syntax of the fixed-port OUT

instruction is as follows

OUT port8. AL or AX

copy a byte or word from AL or AX to 8-bit port

address porte

The fixed-port OUT instruction writes a byte or word from the AL or AX register to an output device connected to a specified 8-bit port address port8. Following are some examples:

OUT 80h, AL

OUT 81h, AX

write a

byte from AL to port

address 80h write a word from AX to port

; address 81h

The variable-port OUT instruction uses the 16-bit port address which is syntax of the variable-port OUT instruction is as follows:

OUT DX, AL or AX

loaded in the DX register. The

copy a byte or word from AL Or AX to 16-bit address given inDEX register

port

The variable-port OUT instruction writes a byte or word from the AL or AX register to an output device

connected to a specified 16-bit port address given in the DX register. Following are some cxampla:

HOV DX,

B000h

OUT DX, AL

MOV DX, 80B0h

OUT DX, AX

;load the port address

port

write a byte to the load the port address

write a word to the port

The variable-port OUT instruction has the same advantage as the variable-port

IN instruction.