8086/8088 INTERRUPTS AND DOS SYSTEM PROGRAMMING

1.1 Interrupt Sources

The 8086/88 microprocessors allow normal program execution to be interrupted by external signals or by special instructions embedded in the program code. When the microprocessor is interrupted, it stops executing the current program and **calls** a procedure which **services** the interrupt. At the end of the **interrupt service routine**, the code execution sequence is returned to the original, interrupted program.

An interrupt can be generated by one of three sources. First, an interrupt can be generated as a result of a processor state violation, called an exception. An example would be a divide-by-zero interrupt produced when the DIV instruction is interpreted to have a zero divisor. Program execution is automatically interrupted and control transferred to an interrupt handler. Conditional interrupts such as this are referred to as internal interrupts.

An interrupt can also be generated by an external device requesting service. This happens when a device signals its request on either the non-maskable interrupt (NMI) or on the INTR interrupt input lines of the processor. The NMI interrupt is generally used to signal the occurrence of a catastrophic event, such as the immanent loss of power. The INTR interrupt is used by all other devices. An interrupt caused by a signal applied to either the NMI or INTR input is referred to as a **hardware interrupt.**

Since there is only one INTR input, and multiple devices may have an interrupt capability, an Intel 8259A Programmable Interrupt Controller (PIC) can be used to manage multiple interrupt requests. The PIC receives requests from peripheral equipment, decides which request has the highest priority and issues an interrupt request to the CPU.

Finally, interrupts may be generated as a result of executing the INT instruction. This is referred to as a **software interrupt.**

1.2 Interrupt Vectoring

Interrupt service routines are called in a manner similar to FAR procedures. Two **16** bit data words are used to specify the location of the interrupt service routine. one word is used to load the CS register and points to the **base address** of the code segment containing the service routine. The second word is used to load the IP with the **offset value** for the desired routine within the specified code segment.

The base and offset words for all interrupt types are grouped together in an **interrupt vector table**. Figure 1 shows the vector table for the 86/88 processors. Note that interrupt vectors are assigned a location based on the **type** of the interrupt. Although some of the 256 possible interrupt vectors are reserved by Intel, most are available to the programmer.

Finally, in the IBM PC (and of course, like-wise in the ATT 6300) some of the interrupt TYPES are reserved for PC/MS-DOS system service routines. Figure 2 shows which interrupt types are reserved for this case.

Interrupt				Interrupt			
Dec	Hex	Address	Use	Dec	Hex	Address	Use
0	0	0000	Generated by CPU when division by zero is attempted	26	1A	0068	Invokes time and date services in BIOS
1	1	0004	Used to single-step through programs (as with DEBUG)	27	1B	006C	Interrupt generated on keyboard break under BIOS; a routine is invoked if we create it
2	2	0008	Non-maskable interrupt; in PCir, NMI has some special uses	28	1C	0070	Interrupt generated at each clock tick; a routine is invoked if we create it
3	3	000C	Used to set break-points in programs (as with DEBUG)	29	1D	0074	Points to table of video control parameters
4	4	0010	Generated when arithmetic result overflows	30	1E	0078	Points to disk base table
5	5	0014	Invokes print-screen service routine in BIOS	31	1F	007C	Points to high video graphics characters
8	8	0020	Generated by hardware clock tick	32	20	0080	Invokes program-terminate

			service in DOS				
9	9	0024	In most models, generated by keyboard action; simulated on PCjr for model compatibility	33	21	0084	Invokes all function-call services in DOS
13	D	0034	Generated during CRT vertical retrace for video control	34	22	0088	If we create it, an interrupt routine is invoked at program end under DOS
14	E	0038	Signals diskette attention (e.g. to signal completion)	35	23	008C	If we create it, an interrupt routine is invoked on keyboard break under DOS
15	F	003C	Used in printer control	36	24	0090	If we create it, an interrupt routine is invoked at critical error under DOS
16	10	0040	Invokes video display services in BIOS	37	25	0094	Invokes absolute diskette read service in DOS
17	11	0044	Invokes equipment-list service in BIOS	38	26	0098	Invokes absolute diskette write service in DOS
18	12	0048	Invokes memory-size service in BIOS	39	27	009C	Ends program, but keeps it in memory-under DOS
19	13	004C	Invokes diskette services in BIOS	68	44	0110	Points to low video graphics characters; only on PCjr
20	14	0050	Invokes communications services in BIOS	72	48	0120	Invokes program to translate PCjr keyboard into PC keyboard
21	15	0054	Invokes cassette tape services in BIOS	73	49	0124	Points to translation table for keyboard-supplement devices
22	16	0058	Invokes standard keyboard services in BIOS				
23	17	005C	Invokes printer services in BIOS				
24	18	0060	Activates ROM-BASIC language, or override for it				
25	19	0064	Invokes boot-strap start-up routine in BIOS				

Figure 2 The interrupt TYPES reserved for use by the IBM family of personal computers. A comparison of the reserved locations in Figures 1 and 2 show that some of the locations are actually reserved by the processor manufacturer for specified exceptions (e.g., TYPES 0-4).

2.0 PROGRAMMING

Figure 3 presents, in flowchart form, how the 86/88 processors handle internal, nonmaskable (NMI), single step and INTR interrupts.

If an INTR interrupt occurs, the **CPU** checks to see if this kind of interrupt is **enabled** (through the use of the STI instruction) and if so, acknowledges the interrupt request **by** generating an **interrupt** acknowledge (IAK) bus cycle. **As a result of the IAK** bus cycle**,** the uP reads an 8 bit TYPE number provided by the interrupting device. This TYPE number is used as an index into the interrupt vector table to get the address of the interrupt handler.

If Trap Flag (TF) is set, the single step operation is activated. After every instruction, a Type I interrupt is generated and a single step interrupt handler is called. If several interrupts occur simultaneously, the single step function is temporarily disabled during the execution of the INTR vectored interrupt service routine.

2.1 INT Instruction

Software interrupts produced by the INT assembler instruction have many uses. For example, one use would be for the testing of the various interrupt service routines. You could use an INT 2 instruction to start the execution of an NMI interrupt service procedure. This would allow you to test the NMI procedure without needing to apply an external signal to the processors NMI input line.

Software interrupts can also be used to call commonly used procedures from many different programs. The **Basic Input/Output System (BIOS)** procedures of an

IBM computer or compatible are a good example of this use of the INT instruction. The specific functions and use of these procedures will be covered in detail in Section 3, below.

Figure 3 The interrupt recognition and processing sequence for the 8086/88 family of processors is shown in flowchart form.

In general, INT activates the interrupt procedure specified by the instruction's TYPE operand. To return from an interrupt service routine, the IRET instruction is used. IRET is used to exit any interrupt procedure, whether activated by hardware or software. The specific action of INT and IRET can be found in any reference on the 8086/88 instruction set.

2.2 Loading and Examining IR Pointers

To be able to vector to your own interrupt service routines, you have to load the interrupt pointer table with the complete two-word start address of your interrupt hander corresponding to the desired interrupt TYPE number. The following example shows how this is done using interrupt type 36. Good programming practices dictate that you should first store the original contents of the part of the pointer table you want to change. The following code segment does this operation.

```
xor ax, ax ; first clear the AX register

mov es, ax ; load ES with segment address

; of interrupt pointer table

; (use 0000h for this example)

mov bx, 36 ; load BX with interrupt type

mov ax, word ptr es:[bx] ; load AX with the offset

; for interrupt type 36

push ax ; save the old interrupt pointer

; table content

mov ax, word ptr es:[bx+2] ; Load AX with the segment

; address of interrupt type 36

push ax ; store the old interrupt pointer

; table content
```

Now, the interrupt table can be loaded with the new vector locations for the handler you have written.

```
xor ax, ax ; first clear AX

mov es, ax ; load ES with the segment address

; for the interrupt pointer table

mov bx, 36 ; load BX with interrupt type

cli ; disable interrupts during the

; changes to the interrupt table

mov word ptr es:[bx], BBBB ; load the lower interrupt pointer

; table word with the base address

; (BBBBH) of the user's interrupt

; service routine

mov word ptr es:[bx+2],SSSS ; load the higher interrupt pointer

; table word with the segment address

; (SSSSH) of the user's interrupt

; service routine

sti ; re-enable interrupts
```

3.0 THE ROM BIOS

3.1 What is the BIOS?

The ROM **BIOS** (Basic Input Output System) is part of the ROM based control -system of an IBM PC or compatible that both **defines the architecture** of the computer to the software, and **provides the fundamental I/O services** that are needed for the operation of the computer.

The BIOS is actually a collection of procedures. Each procedure performs a specific function such as reading a character from the keyboard, writing characters to the screen, or reading information from disk. The ROM BIOS in the AT&T PC 6300 is located starting at physical address FC000 (F000:C000).

3.2 Using BIOS Procedures

System I/O procedures are called with the INT instruction (Section 2). The advantage to calling procedures using this instruction is that the programmer need not know the absolute address of the procedure being called or how to link the procedures into the calling program. All you have to know is the interrupt TYPE for the procedure and the **format** for the parameter(s) that have to be passed to the procedure.

There are twelve BIOS interrupts in all, falling into five groups (Figure 4). For example with INT 10h you can access the video display services. This interrupt includes 20 subroutines. Obviously, one of the INT 10h parameters is a data value indicating which one of the twenty subroutines is required. In this case, the AH Register is loaded with the number of the subroutine. In addition, the AL, BX, CX and DX registers are used to provide the parameters for this subroutines.

Interrupt		
Dec	**Hex**	**Use**
Peripheral Devices Services		
16	10	Video-display services
19	13	Diskette services
20	14	Communications services
21	15	Cassette-tape services
22	16	Standard keyboard services
23	17	Printer services
Equipment Status Services		
17	11	Equipment-list service
18	12	Memory-size service
Time/Date Service		
26	1A	Time and date services
Print-Screen Service		
5	5	Print-screen service
Special Services		
24	18	Activate ROM-BASIC language
25	19	Activate bootstrap start-up routine

Figure 4 The twelve BIOS service routines supported by the IBM PC (and compatibles).

As a simple example of the use of the BIOS routines, let us assume that we want to set the AT&T PC 6300 monitor to the super high graphic resolution mode 64Ox4OO pixel. Normally the is a tricky programming job if you program the video display controller hardware directly. However, when using the BIOS routines, the control change is extremely simple.

mov ah,00h ; load AH with service

; number 0 = "Set video model"

mov al,48h ; load AL with mode number

; 48 = "Super high resolution model"

int 10h ; call the BIOS procedure to set

; the desired video mode

As a second example we will write some pixels to the screen and create graphics. The code shown below will turn ON one pixel at screen position (0,639):

mov ah,OCh ; load number OCh = "write

; pixel dot"

mov al,O1h ; set code for pixel color

```
mov dx,0000h ; load DX with row number = 0
mov cx,027Fh ; Load CX with column number = 639d
int 10h ; call the BIOS procedure to set
; one pixel on the screen
```

Finally, examine the organization and operation of a program that will send characters to the printer. Note that the BIOS interrupt 17h is used.

```
; this program initializes the
; printer port and sends a character
; string to the printer
code segment ; defines that this is a code segment
org 100h ; starting at offset 100h
assume cs:code, ds:code ; and assuming the registers CS and
; DS are loaded with the segment
; address for this routine
; for more information, see the
; TURBO-ASSEMBLER manual!
enter proc far; ; this procedure (ENTER) will simply
call print ; call the PRINT routine written below
ret ; info: see TURBO-ASSEMBLER manual
enter endp
message db 'Hallo printer, how are you?'
db 0dh, 0ah ; a carriage return & line feed is used
; to terminate string messages
print proc near
mov ah, 01h ; Service number 01 = "Initialize
; printer port"
mov dx, 00h ; Use printer port 0
int 17h ; Call procedure to initialize
; printer
mov si, offset message ; Load SI with offset of string
mov cs,29 ; Load CX with length of string
again: mov ah, 00h ; service number 01 = "Print
; character"
move al, byte ptr [si] ; Load character to be sent in AL
int 17h ; Call procedure to print character
cmp ah, 01h ; If character not printed then AH=1
jnz next
jmp exit
next: inc si ; Address of next character
loop again ; Send next character
exit: ret
print endp
```

code ends

end

Note: If you use BIOS Interrupts, make sure, that you save all registers which could be affected by the BIOS routine. Reference [4] provides more details on the operation of the various BIOS routines.

4. SYSTEM CALLS

4.1 Comparing DOS and BIOS Services

When you turn on your PC there are several jobs to do. One is to load the operating system from the system disk. If you use MS-DOS (MicroSoft - Disk Operating System), three system files are loaded; **IBMBIO.COM, COMMAND.COM** and **IBMDOS.COM.**

The file IBMDOS.COM contains DOS service routines. The DOS services, like the BIOS services, can be called by programs through a set of interrupts whose vectors are placed in the interrupt vector table (Section 1.2). The ROM-BIOS routines can be thought of as the lowest-level system software available, performing the most fundamental and primitive input and output operations. The DOS service routines provide more sophisticated and efficient control over the I/O operations than the BIOS routines do, particularly for disk file operations.

4.2 Using DOS interrupts

There are nine DOS interrupt services and they are listed in Figure 5. Five of them, interrupts 20h, 25h through 27h and 2fh are "true" DOS interrupt services, each one having a specifically-defined task associated with it.

Interrupt		
Dec	**Hex**	Description
32	20	Program terminate: come to normal ending
33	21	Function-call umbrella interrupt
34	22	Terminate address
35	23	Break address
36	24	Critical error-handler address
37	25	Absolute disk read
38	26	Absolute disk write
39	27	Terminate-but-stay-resident
47	2F	Print spool control (DOS-3 versions only)

Figure 5 The Nine DOS Interrupts.

In Section 4.4 we will examine the DOS service routine called by INT 21h. This routine provides under one "umbrella" a set of universal functions we can use in our programs.

4.3 DOS Interrupts Except INT 21h

INT 20h "Program Terminate"

This interrupt terminates the current process and returns control back to the parent process. For example, if you run a com.file program, INT 20 terminates your program and returns to DOS.

INT 22h, INT 23h, INT 24h

This three interrupts are used to hold segmented addresses. our programs set these addresses to point to special routines. Then, when the appropriate circumstances arise, DOS invokes the routines located at these addresses through these three address interrupts. This interrupts are used for advanced programming. More information about these interrupts are provided in references [1] and [4].

INT 25h, INT 26h

These two interrupts are used to read and write specific disk sectors. They are the only DOS services that ignore the logical structure of a disk and work with individual sectors.

INT 27h "Terminate But Stay Resident"

The Terminate But Stay Resident call is used to establish a section of code as resident in the system after its termination. This system call is often used to install device drivers in memory.

4.4 The Universal Functions of DOS INT 23Lh

All of the DOS function calls are invoked by INT 21h. Individual functions are selected in the same way as BIOS functions, placing the function number in the AH-Register.

To see how easy the use of DOS functions are, compare the following sample programs with the programs in Section 2.2.

Function "Get Interrupt Vector"

mov ah,35h ; function number in AH-Register

mov al,36 ; interrupt type in AL-Register

int 21h ; get interrupt vector

push bx ; save offset address

push es ; save segment address

Function "Set Interrupt Vector

mov dx,xxxx ; offset address in DX-Register

mov ds,yyyy ; segment address in DS-Register

mov ah,25h ; function number in AH-Register

mov al,36 ; interrupt type in AL-Register

int 21h ; set interrupt vector

The program "Stopwatch" shows you the use of some other INT 21h functions. More information about these functions is provided in reference [1] and [4].

	Range		
Description	**PCjr**	PC/XT	AT
DMA controller (8237A-5)	n/a	000-00F	000-01F
Interrupt controller (8259A)	020-027	020-021	020-03F
Timer (8253-5; 8254.2 in AT)	040-047	040-043	040-05F
PPI (8255A-5; 8255-5 in PCir)	060-067	060-063	n/a
Keyboard (8042)	n/a	n/a	060-06F
DMA page register (74LS612)	n/a	080-083	080-09F
NMI (non-maskable interrupt) mask register	0A0-0A7	0A	070-07F
Interrupt controller 2 (8259A)	n/a	n/a	0A0-0BF
Sound generator (SX76496N)	0C0-0C7	n/a	n/a
DMA controller 2 (8237A-5)	n/a	n/a	0C0-0DF
Clear/reset math coprocessor	n/a	n/a	0F0-0F1
Math coprocessor	n/a	n/a	0F8-0FF
Joystick (game controller)	200-207	200-20F	200-207
Expansion unit	n/a	210-217	n/a
Parallel printer (secondary)	n/a	n/a	278-27F
Serial port (primary)	2F8-2FF	3F8-33FF	3F8-3FF
Serial port (secondary)	n/a	2F8-2FF	2F8-2FF
Prototype card	n/a	300-31F	300-31F
Fixed disk	n/a	320-32F	1F0-1F8
Parallel printer (primary)	n/a	378-37F	378-37F
SDLC (secondary bisynchronous communications in AT only)	n/a	380-38F	380-38F
Bisynchronous communications (primary)	n/a	n/a	3A0-3AF
Monochrome adapter/printer	n/a	3B0-3BF	3B0-3BF
Color/graphics adapter	n/a	3D0-3DF	3D0-3DF
Diskette controller	0F0-0FF	3F0-3F7	3F0-3F7