

TABLE 11-2 Priority Encoder Truth Table

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
1	x	x	x	0	0	1	$x = I_0' I_1'$ $y = I_0' I_1 + I_0' I_2'$ $(IST) = I_0 + I_1 + I_2 + I_3$
0	1	x	x	0	1	1	
0	0	1	x	1	0	1	
0	0	0	1	1	1	1	
0	0	0	0	x	x	0	

that $I_0 = 0$, regardless of the values of the other two lower-priority inputs. The output for I_2 is generated only if higher-priority inputs are 0, and so on down the priority level. The interrupt status IST is set only when one or more inputs are equal to 1. If all inputs are 0, IST is cleared to 0 and the other outputs of the encoder are not used, so they are marked with don't-care conditions. This is because the vector address is not transferred to the CPU when $IST = 0$. The Boolean functions listed in the table specify the internal logic of the encoder. Usually, a computer will have more than four interrupt sources. A priority encoder with eight inputs, for example, will generate an output of three bits.

The output of the priority encoder is used to form part of the vector address for each interrupt source. The other bits of the vector address can be assigned any value. For example, the vector address can be formed by appending six zeros to the x and y outputs of the encoder. With this choice the interrupt vectors for the four I/O devices are assigned binary numbers 0, 1, 2, and 3.

Interrupt Cycle

The interrupt enable flip-flop IEN shown in Fig. 11-14 can be set or cleared by program instructions. When IEN is cleared, the interrupt request coming from IST is neglected by the CPU. The program-controlled IEN bit allows the programmer to choose whether to use the interrupt facility. If an instruction to clear IEN has been inserted in the program, it means that the user does not want his program to be interrupted. An instruction to set IEN indicates that the interrupt facility will be used while the current program is running. Most computers include internal hardware that clears IEN to 0 every time an interrupt is acknowledged by the processor.

At the end of each instruction cycle the CPU checks IEN and the interrupt signal from IST . If either is equal to 0, control continues with the next instruction. If both IEN and IST are equal to 1, the CPU goes to an interrupt cycle. During the interrupt cycle the CPU performs the following sequence of micro-operations:

$SP \leftarrow SP - 1$ Decrement stack pointer
 $M[SP] \leftarrow PC$ Push PC into stack

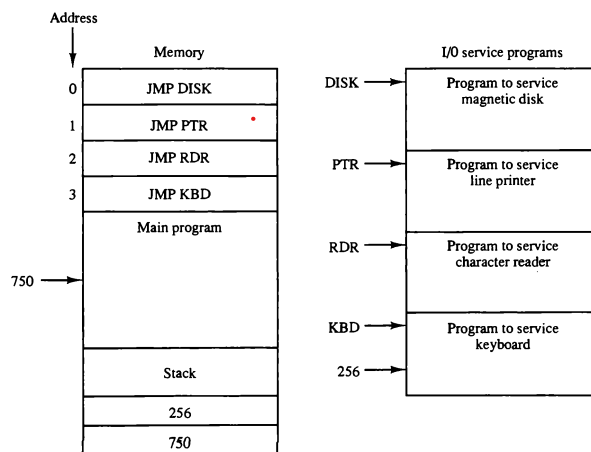
$INTACK \leftarrow 1$ Enable interrupt acknowledge
 $PC \leftarrow VAD$ Transfer vector address to PC
 $IEN \leftarrow 0$ Disable further interrupts
 Go to fetch next instruction

The CPU pushes the return address from PC into the stack. It then acknowledges the interrupt by enabling the $INTACK$ line. The priority interrupt unit responds by placing a unique interrupt vector into the CPU data bus. The CPU transfers the vector address into PC and clears IEN prior to going to the next fetch phase. The instruction read from memory during the next fetch phase will be the one located at the vector address.

Software Routines

A priority interrupt system is a combination of hardware and software techniques. So far we have discussed the hardware aspects of a priority interrupt system. The computer must also have software routines for servicing the interrupt requests and for controlling the interrupt hardware registers. Figure 11-15 shows the programs that must reside in memory for handling the

Figure 11-15 Programs stored in memory for servicing interrupts.



service program

interrupt system. Each device has its own service program that can be reached through a jump (JMP) instruction stored at the assigned vector address. The symbolic name of each routine represents the starting address of the service program. The stack shown in the diagram is used for storing the return address after each interrupt.

To illustrate with a specific example assume that the keyboard sets its interrupt bit while the CPU is executing the instruction in location 749 of the main program. At the end of the instruction cycle, the computer goes to an interrupt cycle. It stores the return address 750 in the stack and then accepts the vector address 00000011 from the bus and transfers it to *PC*. The instruction in location 3 is executed next, resulting in transfer of control to the KBD routine. Now suppose that the disk sets its interrupt bit when the CPU is executing the instruction at address 255 in the KBD program. Address 256 is pushed into the stack and control is transferred to the DISK service program. The last instruction in each routine is a return from interrupt instruction. When the disk service program is completed, the return instruction pops the stack and places 256 into *PC*. This returns control to the KBD routine to continue servicing the keyboard. At the end of the KBD program, the last instruction pops the stack and returns control to the main program at address 750. Thus, a higher-priority device can interrupt a lower-priority device. It is assumed that the time spent in servicing the high-priority interrupt is short compared to the transfer rate of the low-priority device so that no loss of information takes place.

Initial and Final Operations

Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system. Remember that the interrupt enable *IEN* is cleared at the end of an interrupt cycle. This flip-flop must be set again to enable higher-priority interrupt requests, but not before lower-priority interrupts are disabled. The initial sequence of each interrupt service routine must have instructions to control the interrupt hardware in the following manner:

1. Clear lower-level mask register bits.
2. Clear interrupt status bit *IST*.
3. Save contents of processor registers.
4. Set interrupt enable bit *IEN*.
5. Proceed with service routine.

The lower-level mask register bits (including the bit of the source that interrupted) are cleared to prevent these conditions from enabling the interrupt. Although lower-priority interrupt sources are assigned to higher-numbered bits in the mask register, priority can be changed if desired since the