Chapter 7

Interrupt Structure

The NSC800 has a very versatile interrupt structure—probably the most versatile of any 8-bit microprocessor. It includes all of the features of the Z80's interrupt structure, plus some additional features similar to those found on the 8085. The six prioritized interrupt lines of the NSC800 allow simple hardware designs in medium-sized, interrupt-driven systems, since all of the necessary interrupt circuits are on the NSC800. This chapter will discuss all of the features of the NSC800 interrupt structure in detail.

INTERRUPTS

Interrupts play a very important role in many systems. The primary advantage of using interrupts—and it is a big one—is that it allows the processor to be doing useful work when not servicing a peripheral device, instead of constantly monitoring system peripheral devices until they require servicing. This, of course, results in much more efficient use of available processing time.

Interrupts are also used for other purposes. One common use is in multi-user systems. In multi-user systems, the processor spends a small amount of time (called a "time-slice") with each user in succession. A system timing device must generate an interrupt to the processor at the end of each time-slice, to inform the processor that it is time to go to the next user.

Another use for interrupts is in system emergency or error processing. Typical situations include power failures and illegal instruction traps. These types of situations generally employ the use of a non-maskable interrupt—an interrupt that cannot be disabled.

The six NSC800 interrupt lines are all active low inputs which allow

asynchronous interruption of normal processor program execution. The lines, along with their respective NSC800 pin numbers in order of priority from highest to lowest are:

Interrupt	Pin
$\overline{\text{RESET IN}}$	33
$\overline{\text{NMI}}$	21
$\overline{\text{RSTA}}$	22
$\overline{\text{RSTB}}$	23
$\overline{\text{RSTC}}$	24
$\overline{\text{INTR}}$	25

All of these interrupts, with the exception of $\overline{\text{NMI}}$, are low-level triggered; $\overline{\text{NMI}}$ is high-to-low edge triggered.

The $\overline{\text{RESET IN}}$ input, while truly being an interrupt, will not be covered in detail in this chapter, because its operation was discussed in Chapter 6. Any references to interrupts in this chapter, therefore, will not include $\overline{\text{RESET IN}}$, unless explicitly stated. The $\overline{\text{RESET IN}}$ interrupt has the highest priority in the system and will override any other input.

The five NSC800 interrupts are prioritized; that is, each interrupt has a designated priority. Higher priority interrupts may interrupt lower priority interrupts, but the lower priority interrupts may not interrupt higher priority interrupts. If two or more interrupts occur simultaneously, the interrupt having the highest priority will be accepted.

NONMASKABLE INTERRUPT ($\overline{\text{NMI}}$)

The nonmaskable interrupt ($\overline{\text{NMI}}$) has the highest priority of the NSC800 interrupts. Unlike the other NSC800 interrupts, this interrupt is not maskable; i.e., it cannot be disabled. It is generally reserved for system emergencies such as power failures or illegal instruction traps.

This interrupt is edge sensitive, and when triggered (with a high-to-low signal) causes a restart (call) to memory location 0066H. The $\overline{\text{NMI}}$ service routine must begin at this location. Regardless of when the $\overline{\text{NMI}}$ is triggered, it is not recognized or acted upon by the processor until the end of the currently-executing instruction. This is true for all of the NSC800 interrupts.

A nonmaskable interrupt disables the other interrupts, but, unlike the other NSC800 interrupts, saves the current interrupt-enable status so that it may be restored later. This will be discussed in more detail later.

The Z80 and 8085 microprocessors both also have a nonmaskable interrupt. This is $\overline{\text{NMI}}$ on the Z80, and TRAP on the 8085. The operation of the Z80's $\overline{\text{NMI}}$ interrupt is the same as that on the NSC800; the 8085's TRAP interrupt, however, is somewhat different. In the 8085, the TRAP line must go high and remain high until sampled (both the rising edge and the high level are necessary for the interrupt to be acknowledged). When

the TRAP interrupt is triggered, the 8085 performs a restart to memory location 0024H.

RESTART MASKABLE INTERRUPTS

The NSC800 has three restart interrupts: $\overline{\text{RSTA}}$, $\overline{\text{RSTB}}$ and $\overline{\text{RSTC}}$. These interrupts are individually maskable (that is, they can be selectively enabled or disabled) and provide direct calls (restarts) to dedicated memory addresses when activated. The three restart interrupts are shown here along with their respective restart addresses:

$\overline{\text{RSTA}}$	003CH
$\overline{\text{RSTB}}$	0034H
$\overline{\text{RSTC}}$	002CH

These interrupts are active low and generate a call to the respective restart address when activated. They are level triggered and are sampled at the end of each instruction. These interrupts are selectively masked by writing to the Interrupt Control Register (ICR), which will be discussed shortly.

The Z80 does not have interrupts that are comparable to the NSC800's restart interrupts, but the 8085 has three interrupts that are very similar. The 8085 has three interrupts that are very similar. The 8085's RST 7.5, RST 6.5, and RST 5.5 restart interrupts function similar to the NSC800's $\overline{\text{RSTA}}$, $\overline{\text{RSTB}}$, and $\overline{\text{RSTC}}$ restart interrupts, respectively. The only significant difference is that the NSC800 restart interrupts are low-level triggered, while the 8085 restart interrupts are either low-to-high triggered (RST 7.5) or high-level triggered (RST 6.5 and RST 5.5).

MULTI-MODE MASKABLE INTERRUPT ($\overline{\text{INTR}}$)

The $\overline{\text{INTR}}$ interrupt line has the lowest priority of the NSC800 interrupts, and functions identical to the Z80's $\overline{\text{INT}}$ interrupt. The $\overline{\text{INTR}}$ interrupt is by far the most versatile of the NSC800 interrupts. This interrupt, like the restart interrupts described above, is maskable under software control.

The $\overline{\text{INTR}}$ interrupt is a *multi-mode* interrupt; it can function in any of three NSC800 interrupt modes. These interrupt modes—IM 0, IM 1, and IM 2—are software selectable and can be changed dynamically at any time. As mentioned in Chapter 6, the NSC800 is set to IM 0 upon reset. The operation of the three interrupt modes varies greatly and they will be discussed individually.

Interrupt Mode 0—8080A Compatible

Interrupt mode 0 is set either by a processor reset or by execution of the IM 0 instruction. This interrupt mode allows the NSC800 to be compatible with the operation of the 8080A INT interrupt (and the 8085 INTR interrupt). In this mode, the interrupting device must supply an instruction to the processor. While the instruction may be any instruction in the NSC800 instruction set, it is generally a one-byte restart instruction.

The instruction is gated onto the data bus during $\overline{\text{INTA}}$ (interrupt acknowledge); thus, $\overline{\text{INTA}}$ acts like a $\overline{\text{RD}}$ strobe for the interrupting device. If the instruction given to the CPU is a multiple-byte instruction, $\overline{\text{INTA}}$ will go low the necessary number of times to read in the complete instruction (e.g., a three-byte call instruction is generated by the Intel 8259A interrupt controller). The timing for $\overline{\text{INTR}}$/$\overline{\text{INTA}}$ will be discussed later in this chapter.

As mentioned above, in this mode the interrupting device usually provides a one-byte restart instruction to the processor. This is because the restart instructions are one-byte calls to dedicated page zero addresses. The service routines for the various interrupting devices must, therefore, begin at these dedicated addresses. Usually there is a three-byte jump at these locations to a service routine somewhere else in memory, since the eight restart addresses are only eight memory locations apart.

Because restart instructions are one byte in length, and differ by only three bits in their opcodes, they are very easy to generate in hardware. The bit patterns for the eight restart instructions are shown here, along with their respective call addresses:

Restart	Opcode	Address
00H	11000111	0000H
08H	11001111	0008H
10H	11010111	0010H
18H	11011111	0018H
20H	11100111	0020H
28H	11101111	0028H
30H	11110111	0030H
38H	11111111	0038H

The underlined bits indicate the bits that change between the eight restart instruction opcodes.

Figure 7-1 shows a circuit that will generate different restart instructions for different interrupting devices. In this circuit, a 74HC148 is used to generate the three unique bits of the restart instructions, based on which interrupt line is low. The 74HC244 makes sure that all other restart instruction bits are high, and places the restart instruction on the bus during $\overline{\text{INTA}}$.

Figure 7-2 illustrates the operation of mode 0 interrupts.

In using interrupt mode 0, most designers do not use the RST 00 restart instruction, since this causes execution to go to location 0000H, which is generally reserved for processor reset.

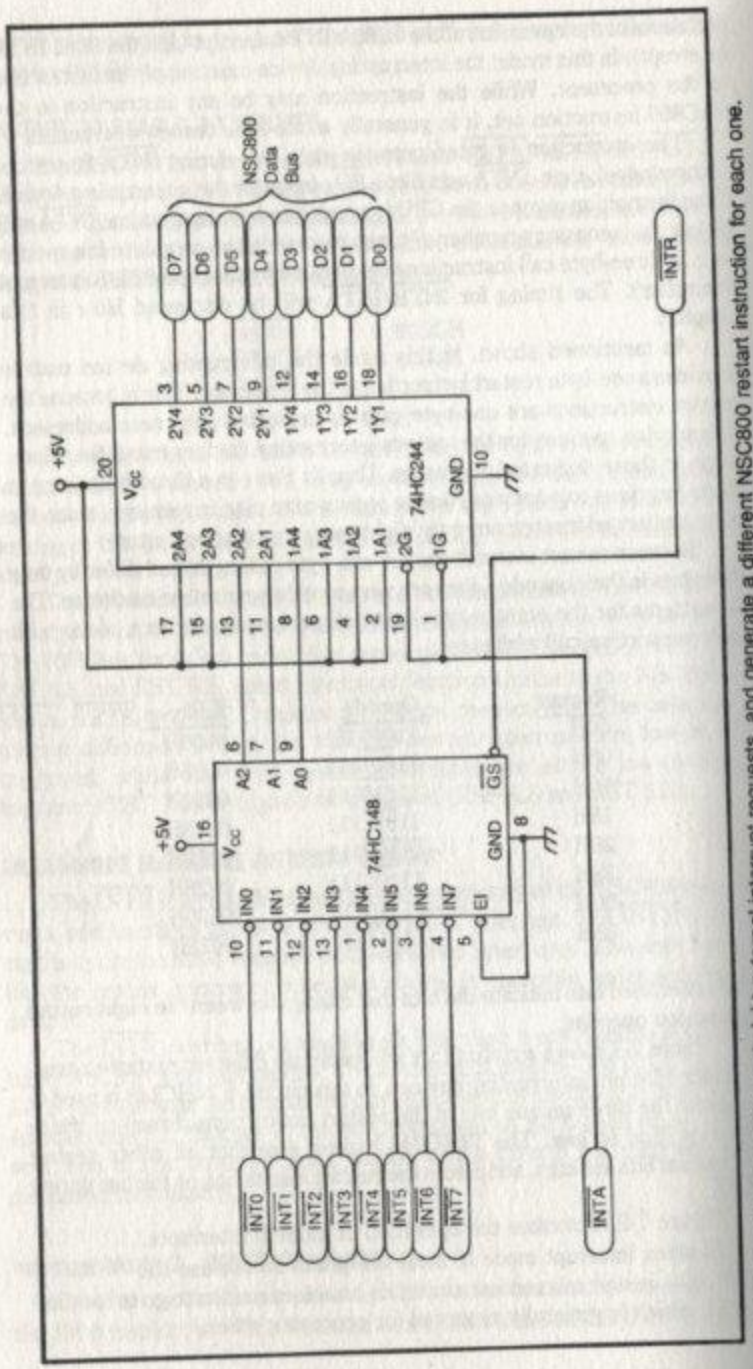


Fig. 7-1. This circuit will accept eight external interrupt requests, and generate a different NSC800 restart instruction for each one.

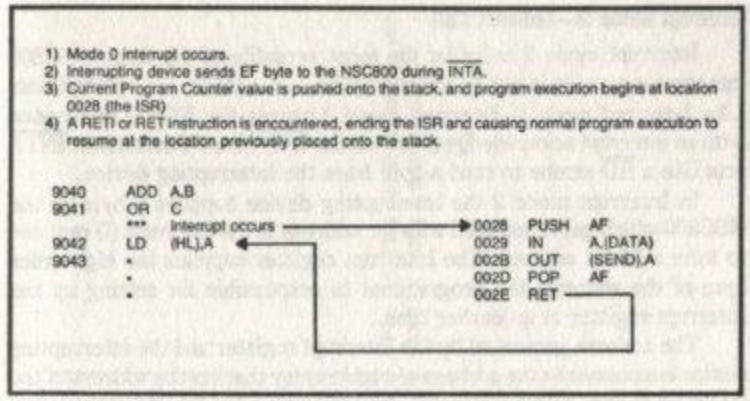


Fig. 7-2. Typical sequence of events for a mode 0 interrupt.

Interrupt Mode 1—Restart Input

Interrupt mode 1 is set by executing the IM 1 instruction. This interrupt mode is the simplest of the three NSC800 interrupt modes and, consequently, requires the least amount of hardware or software support of the available interrupt modes.

When the NSC800 is set to IM 1 , a low level on $\overline{INTR}$ causes the NSC800 to execute a restart to location 0038H—the same address as the highest restart instruction (RST 38H).

The response of the NSC800 to $\overline{INTR}$, while in this mode, is very similar to the NSC800's response to the NMI, except for the call address ($\overline{INTR}$=0038H, $\overline{NMI}$=0066H). Another difference, as we shall see later, is in the time required to execute the call— two wait states are added to the interrupt.

This interrupt mode is particularly useful in situations where less hardware support is desired, and greater performance is not necessary.

Figure 7-3 illustrates the operation of mode 1 interrupts.

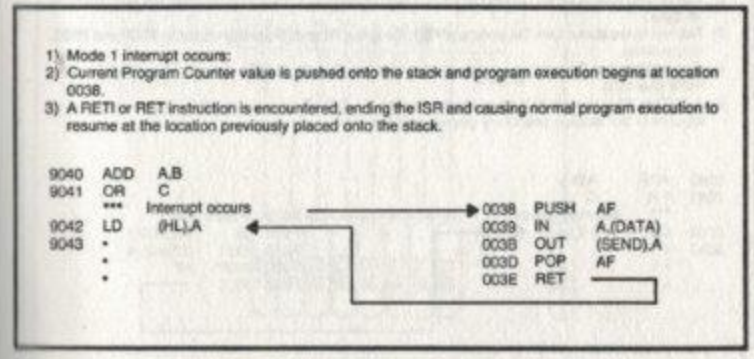


Fig. 7-3. Typical sequence of events for a mode 1 interrupt.

Interrupt Mode 2—Indirect Call

Interrupt mode 2 is by far the most versatile of the three NSC800 interrupt modes; it is set by executing the IM 2 instruction. Like Interrupt mode 0, Interrupt mode 2 causes the NC800 to respond with an interrupt acknowledge (a low on the $\overline{\text{INTA}}$ line). Remember, $\overline{\text{INTA}}$ acts like a $\overline{\text{RD}}$ strobe to read a byte from the interrupting device.

In Interrupt mode 2 the interrupting device supplies a byte to the NSC800 which gets combined with the address in the Interrupt (I) register to form a 16-bit address. The Interrupt register supplies the high-order byte of the address; the programmer is responsible for setting up the Interrupt register at an earlier time.

The address generated by the Interrupt register and the interrupting device is a pointer to the address of a table entry that has the address of the interrupt service routine. Because, by convention, the byte from the interrupting device must be even (bit 0 = 0), all table entries will be at even addresses. Figure 7-4 illustrates the operation of Interrupt mode 2.

The Interrupt register merely holds the page address of the interrupt table. Since a memory page holds 256 bytes, and two bytes are required for each interrupt service routine address entry, up to 128 independent service routines can be specified.

The use of Interrupt mode 2 provides the programmer with a great deal of flexibility in dynamically altering the system memory allocation, or changing the location of the interrupt handling routines. Since the Interrupt register can also be changed, the table itself can also be dynamically moved around in memory.

To use this interrupt mode effectively, the designer must either use peripheral chips that are designed to supply interrupt vector bytes (such as the Z80 support peripherals), or include specialized circuitry for this purpose. An example of such a circuit is shown in Fig. 7-5. Note the similarity to the circuit of Figure 7-1 for mode 0 interrupts.

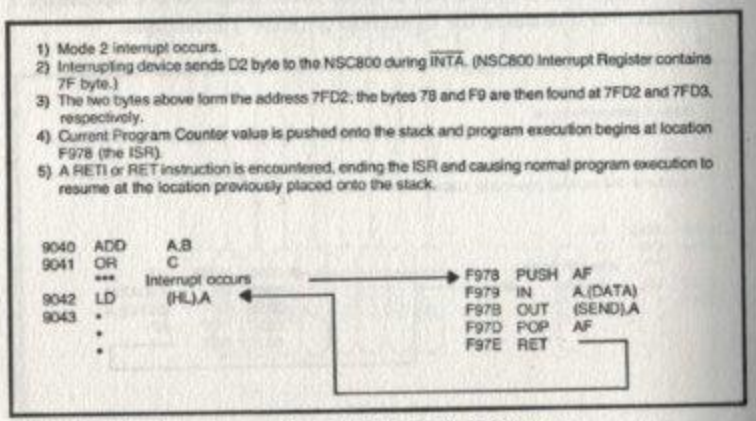
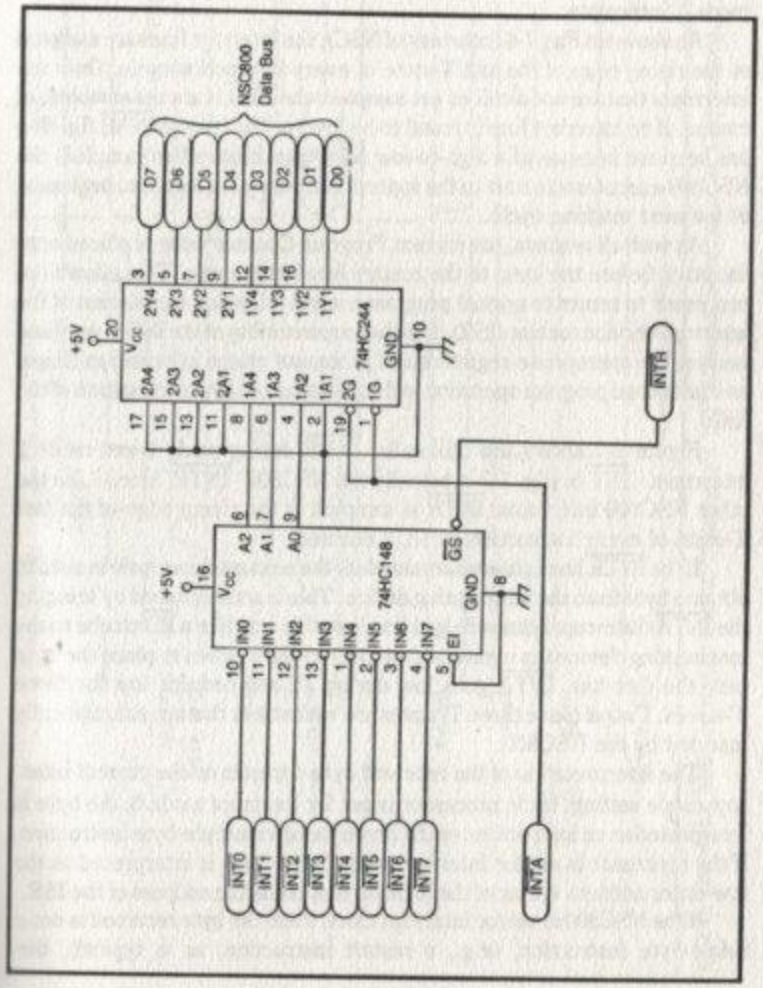


Fig. 7-4. Typical sequence of events for a mode 2 interrupt.

Fig. 7-5. This circuit generates vectors for mode 2 interrupts.



The circuit shown in Figure 7-5 generates eight different interrupt vectors, depending on which of eight interrupt lines is low. The vectors generated range from 00H to 0EH, in increments of two (i.e., 00H, 02H, 04H, . . .). This results in using the first eight entries (16 bytes) of the table (page) designated by the value in the Interrupt register.

TIMING

The timing involved with the NSC800 is fairly simple, making it easy to interface with user hardware. This can be seen by noting the simplicity of the circuits in Figs. 7-1 and 7-5.

Figure 7-6 shows the timing for the $\overline{\text{NMI}}$, and the restart interrupts ($\overline{\text{RSTA}}$, $\overline{\text{RSTB}}$ and $\overline{\text{RSTC}}$); this is also the timing for $\overline{\text{INTR}}$ when set for mode 1 interrupts.

As shown in Fig. 7-6 (courtesy of NSC), the interrupt lines are sampled at the rising edge of the last T-state of every instruction cycle. Only the interrupts that are not disabled get sampled; the $\overline{\text{NMI}}$ is always sampled, of course. If an interrupt line is found to be low (or the internal $\overline{\text{NMI}}$ flip-flop has been set because of a high-to-low $\overline{\text{NMI}}$ transition) when sampled, the NSC800 executes a restart to the appropriate page zero location, beginning at the next machine cycle.

As with all restarts, the current Program Counter value is placed onto the stack before the jump to the restart location is made. This allows the processor to return to normal program control following completion of the interrupt service routine (ISR). It is the responsibility of the ISR to save and restore the appropriate registers and processor status information (flags) so that normal program operation will not be affected by the execution of the ISR.

Figure 7-7 shows the timing for $\overline{\text{INTR}}$ during mode 0 and mode 2 interrupts. ($\overline{\text{INT}}$ in Fig. 7-7 is actually the NSC800 "$\overline{\text{INTR}}$" line.) Like the other NSC800 interrupts, $\overline{\text{INTR}}$ is sampled at the rising edge of the last T-state of every instruction cycle, if enabled.

If the $\overline{\text{INTR}}$ line is low when sampled, the next machine cycle is used to obtain a byte from the interrupting device. This is accomplished by bringing the $\overline{\text{INTA}}$ (interrupt acknowledge) line low; this acts like a $\overline{\text{RD}}$ strobe to the interrupting device (as mentioned earlier) to tell it when to place the byte onto the data bus. $\overline{\text{INTA}}$ goes low during T2 and remains low for three T-states. Two of these three T-states are wait states that are automatically inserted by the NSC800.

The interpretation of the received byte depends on the current interrupt mode setting. If the processor is set for Interrupt mode 0, the byte is interpreted as an instruction, or the first byte of a multiple-byte instruction. If the processor is set for Interrupt mode 2, the byte is interpreted as the low-order address vector of the location that holds the address of the ISR.

If the NSC800 is set for Interrupt mode 0 and the byte received is not a single-byte instruction, (e.g., a restart instruction, as is typical), the

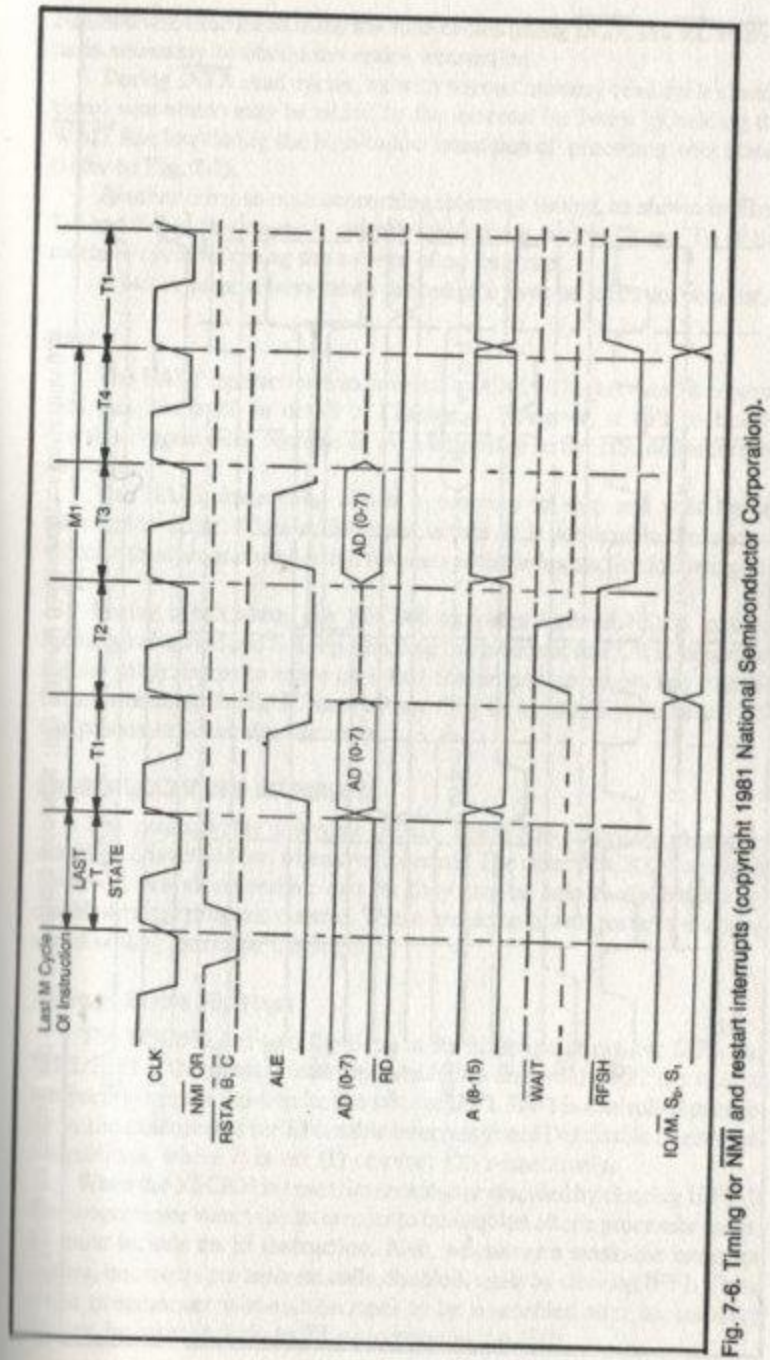


Fig. 7-6. Timing for $\overline{\text{NMI}}$ and restart interrupts (copyright 1981 National Semiconductor Corporation).

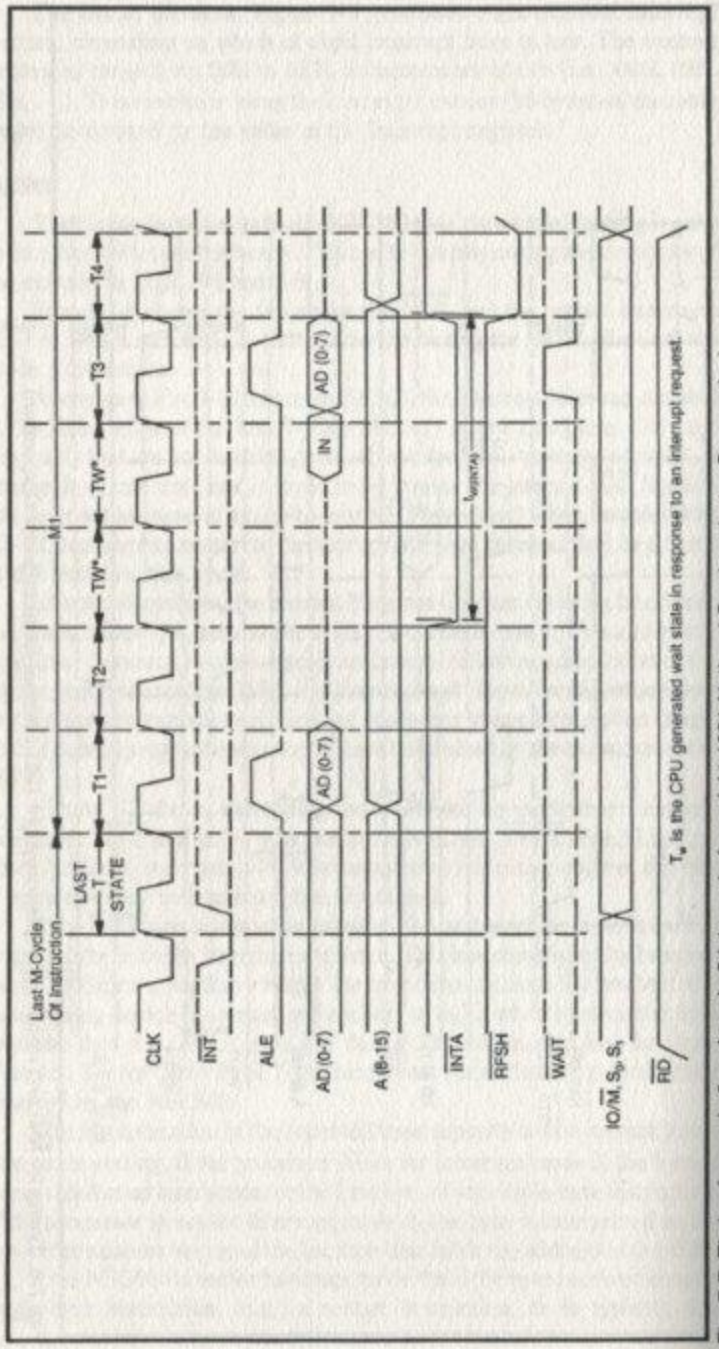


Fig. 7-7. Timing for mode 0 and mode 2 interrupts (copyright 1981 National Semiconductor Corporation).

NSC800 will execute as many machine cycles (using $\overline{INTA}$ as a $\overline{RD}$ strobe) as is necessary to obtain the entire instruction.

During $\overline{INTA}$ read cycles, as with normal memory read cycles, additional wait states may be added by the external hardware by holding the $\overline{WAIT}$ line low during the high-to-low transition of preceding wait states (refer to Fig. 7-7).

Another thing to note concerning interrupt timing, as shown in Figs. 7-6 and 7-7 is that memory refresh takes place during T3 and T4 of the machine cycle following the receipt of an interrupt.

A bus request always takes precedence over an interrupt request.

HALT

The HALT instruction was covered briefly in Chapter 4 and its operation was discussed in detail in Chapter 6. However, it will be briefly discussed again here because of its relationship to the NSC800 interrupt structure.

The HALT instruction allows a program to stop and wait for an interrupt to occur. When an interrupt occurs, it is serviced in the normal manner, then program execution resumes with the instruction following the HALT.

During a halt state, the NSC800 executes internal NOPs to keep memory refreshed and to keep sampling the interrupt lines. It is important for the programmer to make sure that the proper interrupts are enabled before executing the HALT instruction. Only by an interrupt (or reset) will the processor leave the halt state.

ENABLING/DISABLING INTERRUPTS

The nonmaskable interrupt ($\overline{NMI}$), being non-maskable, cannot be disabled; it is acted upon whenever it occurs. The other NSC800 interrupts, however, are all maskable; that is, they can be selectively enabled or disabled under program control. There are actually two parts to enabling and disabling maskable interrupts.

Interrupt Enable Flip-Flops

The NSC800 has two flip-flops in its interrupt structure: IFF1 and IFF2. IFF1 is the primary interrupt enable flip-flop, while IFF2 is a special temporary-storage flip-flop for the state of IFF1. IFF1 is controlled primarily by the execution of the EI (enable interrupts) and DI (disable interrupts) instructions, where it is set (1) or reset (0), respectively.

When the NSC800 is reset, interrupts are disabled by clearing IFF1. If the programmer wants the interrupts to be enabled after a processor reset, he must include an EI instruction. Also, whenever a maskable interrupt occurs, interrupts are automatically disabled, again by clearing IFF1. Thus, if the programmer wishes interrupts to be re-enabled after an interrupt occurs, he must include an EI instruction in the ISR.

If the programmer wishes to disable interrupts at any time, he needs merely to execute the DI instruction; this will clear IFF1, disabling the maskable NSC800 interrupts.

The interrupt flip-flop IFF2 is used during nonmaskable interrupts. During a nonmaskable interrupt, the NSC800 disables the maskable interrupts by clearing IFF1. Before this, however, the NSC800 stores the value of IFF1 in IFF2, so that IFF1 may be restored to its original pre-interrupt value after completion of the $\overline{\text{NMI}}$ service routine.

A special instruction, RETN (return from nonmaskable interrupt), has been provided for this purpose. The RETN instruction is placed at the end of a $\overline{\text{NMI}}$ service routine; when the RETN instruction is executed, the value in IFF2 is placed into IFF1, as the return is being done. Figure 7-8 illustrates the operation of IFF1 and IFF2.

The value of IFF1 may be viewed at any time by executing either the LD A, I instruction or the LD A, R instruction. When these instructions are executed, the value of IFF1 is placed into the P/V flag.

Earlier in this chapter it was mentioned that the NSC800 interrupts are sampled during the last T-state of every instruction cycle. While this is true for the most part, there is an exception to the rule. There are two NSC800 instructions during which the interrupts are not sampled: EI and DI.

By not sampling the interrupt lines during these instructions, a potential problem is avoided. It is not unusual to desire further interrupts to be disabled while servicing an interrupt. Usually, however, it is desirable to have the interrupts re-enabled after the completion of the ISR. This is accomplished by executing an EI instruction immediately before returning from the ISR.

If the NSC800 did sample interrupts during the EI instruction (during the last T-state), another NSC800 interrupt could occur before the ISR return instruction was executed. This would cause another return address

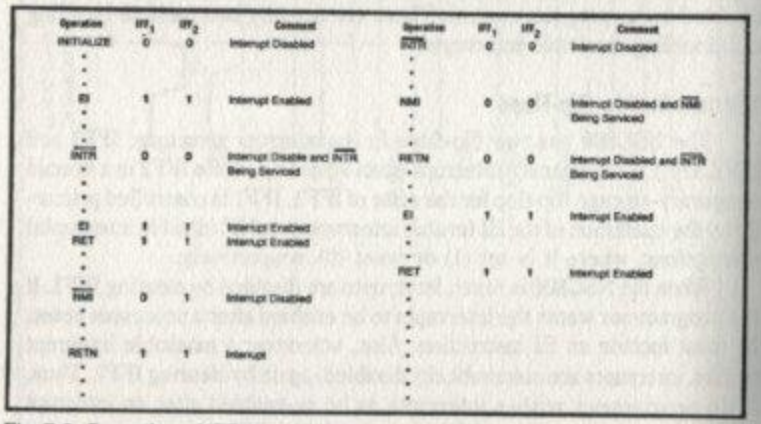


Fig. 7-8. Operation of IFF1 and IFF2 interrupt flip-flops. (copyright 1981 NSC.)

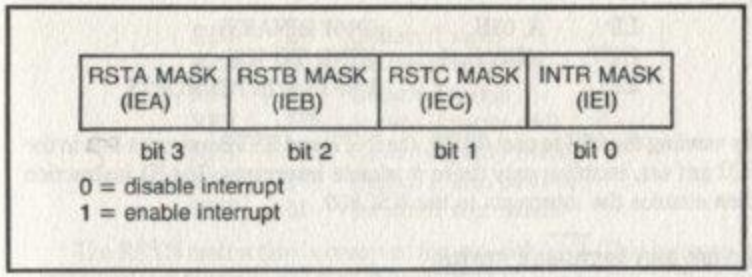


Fig. 7-9. NSC800 Interrupt Control Register (ICR).

to be placed onto the stack and a new ISR to be executed (maybe even the same one!). This process could occur over and over, until the stack overflowed.

By not sampling the interrupts until the instruction following the EI instruction, the ISR return instruction (RET or RETI) can be executed before another interrupt is accepted by the processor. This, in turn, keeps the stack from piling up an excess number of return addresses.

Interrupt Control Register (ICR)

The Interrupt Control Register (ICR) is an on-chip, 4-bit write only register, accessed as an output port at location 0BBH. Figure 7-9 shows the bit arrangement of the ICR. The ICR offers the programmer a second level of maskable control over the NSC800's maskable interrupts.

The ICR is secondary to IFF1; thus, if IFF1 is reset, no maskable interrupt will be accepted by the NSC800, regardless of the settings of the ICR bits. If IFF1 is set, enabling maskable interrupts, the bits of the ICR may be set or cleared as desired, enabling or disabling the respective maskable interrupts.

During processor reset, the ICR is set to 01H, enabling $\overline{\text{INTR}}$ but disabling $\overline{\text{RSTA}}$, $\overline{\text{RSTB}}$ and $\overline{\text{RSTC}}$. Also during reset, IFF1 is reset, disabling all maskable interrupts. When interrupts are enabled (with the EI instruction), however, the NSC800 will accept $\overline{\text{INTR}}$ interrupts, but none of the other maskable interrupts.

The reason the NSC800 sets up the ICR in this manner during reset is to maintain Z80 compatibility. The Z80 does not have interrupts equivalent to the NSC800's restart interrupts, so they are disabled during reset. Therefore, like the Z80, only $\overline{\text{INTR}}$ (or "INT" on the Z80) is enabled on the NSC800 after the first execution of an EI instruction, unless the ICR is previously changed by the programmer.

Maskable interrupts are enabled by setting the respective bit in the ICR. They are, likewise, disabled by clearing the respective ICR bit. For interrupts to be enabled to the NSC800, IFF1 must also be set (using the EI instruction). For example, if we wished to enable only the $\overline{\text{INTR}}$ and $\overline{\text{RSTB}}$ interrupts, we might execute the following instructions: