Freescale Semiconductor, Inc.

SECTION 8
PROCESSING STATES

The DSP is always in one of five processing states: normal, exception, reset, wait, and stop. These states are described in the following paragraphs.

8.1 NORMAL PROCESSING STATE

The normal processing state is associated with instruction execution. Details concerning normal processing of the individual instructions can be found in APPENDIX A INSTRUCTION SET DETAILS. Instructions are executed using a three-stage pipeline, which is described in the following paragraphs.

8.1.1 Instruction Pipeline

DSP56000/DSP56001 instruction execution is performed in a three-stage pipeline, allowing most instructions to execute at a rate of one instruction every instruction cycle. However, certain instructions require additional time to execute: instructions longer than one word, instructions using an addressing mode that requires more than one cycle, and instructions causing a control-flow change. In the latter case, a cycle is needed to clear the pipeline.

Instruction pipelining allows overlapping of instruction execution so that the fetch-decode-execute operations of a given instruction occur concurrently with the fetch-decode-execute operations of other instructions. Specifically, while an instruction is executed, the next instruction to be executed is decoded, and the instruction to follow the instruction being decoded is fetched from program memory. Only one word is fetched per cycle so that, if an instruction is two words in length, the additional word will be fetched before the next instruction is fetched. Table 8-1 demonstrates pipelining; F1, D1, and E1 refer to the fetch, decode, and execute operations, respectively, of the first instruction. The third instruction, which contains an instruction extension word, takes two instruction cycles to execute. The extension word will be either an absolute address or immediate data. Although it takes three instruction cycles for the pipeline to fill and the first instruction to execute, an instruction usually executes on each instruction cycle thereafter.

For More Information On This Product,
Go to: www.freescale.com

Table 8-1 Instruction Pipelining

Operation	Instruction Cycle									
	1	2	3	4	5	6	7	•	•	•
Fetch	F1	F2	F3	F3e	F4	F5	F6	•	•	•
Decode		D1	D2	D3	D3e	D4	D5	•	•	•
Execute			E1	E2	E3	E3e	E4	•	•	•

Each instruction requires a minimum of three instruction cycles (12 clock phases) to be fetched, decoded, and executed. This means that there is a delay of three instruction cycles on powerup to fill the pipe. A new instruction may begin immediately following the previous instruction. Two-word instructions require a minimum of four instruction cycles to execute (three cycles for the first instruction word to move through the pipe and execute and one more cycle for the second word to execute). A new instruction may start after two instruction cycles.

The pipeline is normally transparent to the user. However, it will affect program execution in some situations. These situations, which are instruction-sequence dependent, are best described by case studies. Most of these restricted sequences occur because 1) all addresses are formed during instruction decode, or 2) they are the result of contention for an internal resource such as the status register (SR). If the execution of an instruction depends on the relative location of the instruction in a sequence of instructions, there is a pipeline effect. To test for a suspected pipeline effect, compare between the execution of the suspect instruction 1) when it directly follows the previous instruction and 2) when four NOPs are inserted between the two. If there is a difference, it is due to a pipeline effect. The DSP56000/DSP56001 assembler (ASM56000) is designed to flag instruction sequences with potential pipeline effects so that the user can decide if the operation will be as expected.

Case 1: The following two examples show similar code sequences.

1. .No pipeline effect:

```
ORI #xx,CCR        ;Changes CCR at the end of execution time slot
Jcc xxxx           ;Reads condition codes in SR in its execution time slot
```

The Jcc will test the bits modified by the ORI without any pipeline effect in the code segment above.

2. Instruction that started execution during decode:

```
ORI #04,OMR        ;Sets DE bit at execution time slot
MOVE x:$100,a      ;Reads external RAM instead of internal ROM
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

A pipeline effect occurs in example 2 because the address of the MOVE is formed at its decode time before the ORI changes the DE bit (which changes the memory map) in the ORI's execution time slot. The following code produces the expected results of reading the internal ROM:

```
ORI #04,OMR          ;Sets DE bit at execution time slot
NOP                  ;Delays the MOVE so it will read the updated OMR
MOVE x:$100,a        ;Reads internal ROM
```

Case 2: One of the more common sequences where pipeline effects are apparent is as follows:

```
  •                  ;Move a number into register Rn (n=0; - 7).
  •
MOVE #xx,Rn
MOVE X:(Rn),A        ;Use the new contents of Rn to address memory.
  •
  •
```

In this case, before the first MOVE instruction has written Rn during its execution cycle, the second MOVE has accessed the old Rn, using the old contents of Rn. This is because the address for indirect moves is formed during the decode cycle. This overlapping instruction execution in the pipeline causes the pipeline effect. One instruction cycle should be allowed after a register has been written by a MOVE instruction before the new contents are available for use by another MOVE instruction. The proper instruction sequence is as follows:

```
  •                  ;Move a number into register Rn.
  •
MOVE X0,Rn
NOP                  ;Execute any instruction or instruction
  •                  ;sequence not using Rn.
  •
MOVE X:(Rn),A        Use the new contents of Rn.
```

Case 3: A situation related to Case 2 can be seen in the boot ROM code shown in APPENDIX A of the DSP56001 Advance Information Data Sheet (ADI1290). At the end of the bootstrap operation, the operation mode register (OMR) is changed to mode #2, and then the program that was loaded is executed. This process is accomplished in the last three instructions:

```
_BOOTEND    MOVEC    #2,OMR      ;Set the operating mode to 2
                                 ;(and trigger an exit from
                                 ;bootstrap mode).
            ANDI     #$0,CCR     ;Clear SR as if RESET and
                                 ;introduce delay needed for
                                 ;Op. Mode change.
            JMP      <$0         ;Start fetching from PRAM, P:$0000
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The JMP instruction generates its jump address during its decode cycle. If the JMP instruction followed the MOVEC, the MOVEC instruction would not have changed the OMR before the JMP instruction formed the fetch address. As a result, the jump would fetch the instruction at P:$0000 of the bootstrap ROM (MOVE #$FFE9,R2). The OMR would then change due to the MOVEC instruction, and the next instruction would be the second instruction of the downloaded code at P:$0001 of the internal RAM. However, the ANDI instruction allows the OMR to be changed before the JMP instruction uses it, and the JMP fetches P:$0000 of the internal RAM.

Case 4: An interrupt has two additional control cycles that are executed in the interrupt controller concurrently with the fetch, decode, and execute cycles (see 8.2 EXCEPTION PROCESSING STATE (INTERRUPT PROCESSING) and Figure 8-2). During these two control cycles, the interrupt is arbitrated by comparing the interrupt mask level with the interrupt priority level (IPL) of the interrupt and allowing or disallowing the interrupt. Therefore, if the interrupt mask is changed after an interrupt is arbitrated and accepted as pending but before the interrupt is executed, the interrupt will be executed, regardless of what the mask was changed to. The following examples show that the old interrupt mask is in effect for up to four additional instruction cycles after the interrupt mask is changed. All instructions shown in the examples here are one-word instructions; however, one two-word instruction can replace two one-word instructions except where noted.

1. Program flow with no interrupts after interrupts are disabled:

```
 •
 •
ORI #03,MR      ;Disable interrupts
INST 1
INST 2
INST 3
INST 4
 •
 •
```

2. The four possible variations in program flow that may occur after interrupts are disabled:

```
 •                  •                  •                •
 •                  •                  •                •
ORI #03,MR         ORI #03,MR         ORI #03,MR       ORI #03,MR
II (See Note 2)    INST 1             INST 1           INST 1
II+1               II                 INST 2           INST 2
INST 1             II+1               II               INST 3 (See Note 1)
INST 2             INST 2             II+1             II
INST 3             INST 3             INST 3           II+1
INST 4             INST 4             INST 4           INST 4
 •                  •                  •                •
 •                  •                  •                •
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

Note 1: INST 3 may be executed at that point only if the preceding instruction (INST 2) was a single-word instruction.

Note 2: II=Interrupt instruction from maskable interrupt.

The following program flow will not occur because the ORI instruction becomes effective after a pipeline latency of four instruction cycles:

```
 •
 •
ORI #03,MR          ;Disable interrupts.
INST 1
INST 2
INST 3
INST 4
II                  ;Interrupts disabled.
II+1                ;Interrupts disabled.
 •
 •
```

1. Program flow without interrupts after interrupts are re-enabled:

```
 •
 •
ANDI #00,MR         ;Enable interrupts
INST 1
INST 2
INST 3
INST 4
 •
 •
```

2. Program flow with interrupts after interrupts are re-enabled:

```
 •
 •
ANDI #00,MR         ;Enable interrupts
INST 1              ;Uninterruptable
INST 2              ;Uninterruptable
INST 3              ;II+1 fetched
INST 4              ;II+1 fetched
II
II+1
 •
 •
```

The DO instruction is another instruction that begins execution during the decode cycle of the pipeline. As a result, there are a number of restrictions concerning access contention with the program controller registers accessed by the DO instruction. The ENDDO instruction has similar restrictions. APPENDIX A INSTRUCTION SET DETAILS contains

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

additional information on the DO and ENDDO instruction restrictions.

Case 5: A resource contention problem can occur when one instruction is using a register during its decode while the instruction executing is accessing the same resource. One example of this is as follows:

```
MOVEC        X:$100,SSH
DO           #$10,END
```

The problem occurs because the MOVEC instruction loads the contents of X:$100 into the system stack high (SSH) during T3 of its execution cycle. The DO instruction that follows pushes the stack (LA → SSH, LC → SSL) during T3 of its decode cycle. Therefore, the two instructions try writing to the SSH simultaneously and conflict.

8.1.2 Summary of Pipeline-Related Restrictions

A summary of the instruction sequences that cause pipeline effects is given in the following paragraphs. Additional information concerning the individual instructions can be found in APPENDIX A INSTRUCTION SET DETAILS.

DO instruction restrictions:

The DO instruction must not be immediately preceded by any of the following instructions:

BCHG/BCLR/BSET LA, LC, SSH, SSL, or SP
MOVEC/MOVEM to LA, LC, SSH, SSL, or SP
MOVEC/MOVEM from SSH

Restrictions near the end of DO loops:

Proper DO loop operation is guaranteed if no instruction starting at address LA-2, LA-1, or LA specifies the program controller registers SR, SP, SSL, LA, LC, or (implicitly) PC as a destination register or specifies SSH as a source or a destination register. Also, SSH can not be specified as a source register in the DO instruction.

The restricted instructions at LA-2, LA-1, and LA are as follows:

DO
BCHG/BCLR/BSET LA, LC, SR, SP, SSH, or SSL
BTST SSH
JCLR/JSET/JSCLR/JSSET SSH
MOVEC/MOVEM/MOVEP from SSH
MOVEC/MOVEM/MOVEP to LA, LC, SR, SP, SSH, or SSL
ANDI/ORI MR

The restricted instructions at LA include the following:

Any two-word instruction

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Jcc, JMP, JScc, JSR,
REP, RESET, RTI, RTS, STOP, WAIT

Other restrictions are

DO SSH,xxxx
JSR/JScc/JSCLR/JSSET to LA, if loop flag is set

ENDDO instruction restrictions:

The ENDDO instruction must not be immediately preceded by any of the following instructions:

BCHG/BCLR/BSET LA, LC, SR, SSH, SSL, or SP
MOVEC/MOVEM to LA, LC, SR, SSH, SSL, or SP
MOVEC/MOVEM from SSH
ANDI/ORI MR

RTI and RTS instruction restrictions:

The RTI instruction must not be immediately preceded by any of the following instructions:

BCHG/BCLR/BSET SR, SSH, SSL, or SP
MOVEC/MOVEM to SR, SSH, SSL, or SP
MOVEC/MOVEM from SSH
ANDI MR, ANDI CCR
ORI MR, ORI CCR

The RTS instruction must not be immediately preceded by any of the following instructions:

BCHG/BCLR/BSET SSH, SSL, or SP
MOVEC/MOVEM to SSH, SSL, or SP
MOVEC/MOVEM from SSH

SP and SSH/SSL register manipulation restrictions:

In addition to all the above restrictions concerning SP, SSH, and SSL, the following instruction sequences are illegal:

1. BCHG/BCLR/BSET SP
2. MOVEC/MOVEM/MOVEP from SSH or SSL

and

1. MOVEC/MOVEM to SP
2. MOVEC/MOVEM/MOVEP from SSH or SSL

and

1. MOVEC/MOVEM to SP
2. JCLR/JSET/JSCLR/JSSET SSH or SSL

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

and

1. BCHG/BCLR/BSET SP
2. JCLR/JSET/JSCLR/JSSET SSH or SSL

Also the instruction MOVEC SSH,SSH is illegal.

Rn, Nn, and Mn register restrictions:

If an address register (R0 – R7, N0 – N7, or M0 – M7) is changed with a move-type instruction (LUA, Tcc, MOVE, MOVEM, MOVEC, or parallel move), the new contents will not be available for use as a pointer until the second following instruction. This restriction does not apply to registers updated as part of an indirect addressing mode.

Fast interrupt routines:

SWI, STOP, and WAIT may not be used in a fast interrupt routine.

8.2 EXCEPTION PROCESSING STATE (INTERRUPT PROCESSING)

The exception processing state is associated with interrupts that can be generated by conditions inside the DSP or from external sources. There are many sources for interrupts on the DSP56000/DSP56001; some of these sources can generate more than one interrupt. A prioritized interrupt vector scheme with 32 vectors is used to provide fast interrupt service. The following list outlines how interrupts are processed by the DSP56000/DSP56001:

9. A hardware interrupt is synchronized with the DSP clock, and the interrupt pending flag for that particular hardware interrupt is set. An interrupt source can have only one interrupt pending at any given time.
10. All pending interrupts (external and internal) are arbitrated to select which interrupt will be processed. The arbiter automatically ignores any interrupts with an IPL lower than the interrupt mask level in the SR and selects the remaining interrupt with the highest IPL.
11. The interrupt controller then freezes the program counter (PC) and fetches two instructions at the two interrupt vector addresses associated with the selected interrupt.
12. The interrupt controller jams the two instructions into the instruction stream and releases the PC, which is used for the next instruction fetch. The next interrupt arbitration is then begun.

If neither instruction is a change of program-flow instruction (i.e., a JSR), the state of the machine is not saved on the stack, and a fast interrupt is executed. A long interrupt is formed if one of the interrupt instructions fetched is a JSR instruction. The PC is immedi-

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

ately released, the SR and the PC are saved in the stack, and the jump instruction controls where the next instruction is fetched from. While either an unconditional jump or conditional jump can be used to form a long interrupt, they do not store the PC on the stack; therefore, there is no return path.

In digital signal processing, one of the main uses of interrupts is to transfer data between DSP memory or registers and a peripheral device. When such an interrupt occurs, a limited context switch with minimum overhead is often desirable. This limited context switch is accomplished by a fast interrupt. The long interrupt is used when a more complex task must be accomplished to service the interrupt..

The second and third activities require two additional control cycles, which effectively make the interrupt pipeline five levels deep.

8.2.1 Interrupt Sources

Exceptions may originate from any of the 32 vector addresses listed in Table 8-2. The corresponding interrupt starting address for each interrupt source is shown. These addresses are located in the first 64 locations of program memory. When an interrupt is serviced, the instruction at the interrupt starting address is fetched first. Because the program flow is directed to a different starting address for each interrupt, the interrupt structure of the DSP56000/DSP56001 is said to be vectored. A vectored interrupt structure has low overhead execution. If it is known a priori that certain interrupts will not be used, those interrupt vector locations can be used for program or data storage.

The 32 interrupts are prioritized into four levels. Level 3, the highest priority level, is not maskable. Levels 0 – 2 are maskable. The interrupts within each level are prioritized according to a predefined priority. The level-3 interrupts (reset, illegal instruction, nonmaskable interrupt (NMI), stack error, trace, and software interrupt (SWI) are discussed individually.

8.2.1.1 Hardware Interrupt Source

There are two types of hardware interrupts in the DSP: internal and external. The internal interrupts include all of the on-chip peripheral devices (host interface (HI), synchronous serial interface (SSI), and serial communications interface (SCI). Each internal interrupt source is latched and serviced if it is not masked. When it is serviced, the interrupt is cleared. Each internal hardware source has independent enable control.

The external hardware interrupts include $\overline{\text{RESET}}$, NMI, $\overline{\text{IRQA}}$, and $\overline{\text{IRQB}}$. The $\overline{\text{RESET}}$ interrupt, which is level sensitive, is the highest level interrupt (IPL 3). The $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$ interrupts can be programmed to be level sensitive or edge sensitive. Since the level-sensitive interrupts will not be cleared automatically when they are serviced, they must be cleared by other means to prevent multiple interrupts. The edge-sensitive inter-

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 8-2 Interrupt Sources

Interrupt Starting Address	IPL	Interrupt Source
$0000	3	Hardware $\overline{\text{RESET}}$
$0002	3	Stack Error
$0004	3	Trace
$0006	3	SWI
$0008	0 - 2	$\overline{\text{IRQA}}$
$000A	0 - 2	$\overline{\text{IRQB}}$
$000C	0 - 2	SSI Receive Data
$000E	0 - 2	SSI Receive Data With Exception Status
$0010	0 - 2	SSI Trasmit Data
$0012	0 - 2	SSI Transmit Data with Exception Status
$0014	0 - 2	SCI Receive Data
$0016	0 - 2	SCI Receive Data with Exception Status
$0018	0 - 2	SCI Transmit Data
$001A	0 - 2	SCI Idle Line
$001C	0 - 2	SCI Timer
$001E	3	NMI — Reserved for Hardware Development
$0020	0 - 2	Host Receive Data
$0022	0 - 2	Host Transmit Data
$0024	0 - 2	Host Command (Default)
$0026	0 - 2	Available for Host Command
$0028	0 - 2	Available for Host Command
$002A	0 - 2	Available for Host Command
$002C	0 - 2	Available for Host Command
$002E	0 - 2	Available for Host Command
$0030	0 - 2	Available for Host Command
$0032	0 - 2	Available for Host Command
$0034	0 - 2	Available for Host Command
$0036	0 - 2	Available for Host Command
$0038	0 - 2	Available for Host Command
$003A	0 - 2	Available for Host Command
$003C	0 - 2	Available for Host Command
$003E	3	Illegal Instruction

rupts are latched as pending on the high-to-low transition of the interrupt input and are

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

automatically cleared when the interrupt is serviced. $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$ can be programmed to one of three priority levels: 0, 1, or 2, all of which are maskable. Additionally, both of these interrupts have independent enable control.

When the $\overline{\text{IRQA}}$ or $\overline{\text{IRQB}}$ interrupts are disabled in the interrupt priority register, the pending request will be ignored, regardless of whether the interrupt input was defined as level sensitive or edge sensitive. Additionally, if the interrupt is defined as edge sensitive, its edge-detection latch will remain in the reset state as long as the interrupt is disabled; if the interrupt is defined as level sensitive, its edge-detection latch will remain in the reset state. If the level-sensitive interrupt is disabled while the interrupt is pending, the pending interrupt will be cancelled. However, if the interrupt has been fetched, it normally will not be cancelled.

Interrupt service, which begins by fetching the instruction word in the first vector location, is considered finished when the instruction word in the second vector location is fetched. In the case of an edge-triggered interrupt, the internal latch is automatically cleared when the second vector location is fetched. The fetch of the first vector location does not guarantee that the second location will be fetched. Figure 8-1 illustrates the one case where the second vector location is not fetched. In Figure 8-1, the SWI instruction discards the fetch of the first interrupt vector to ensure that the SWI vectors will be fetched. Instruction n4 is decoded as an SWI while ii1 is being fetched. Execution of the SWI requires that ii1 be discarded and the two SWI vectors (ii3 and ii4) be fetched instead.

INTERRUPT CONTROL CYCLE 1		i		i*							
INTERRUPT CONTROL CYCLE 2			i		i*						
FETCH	n3	n4	n5	ii1		ii3	ii4	sw1	sw2	sw3	sw4
DECODE	n2	n3	SWI	—	—	—	JSR	—	sw1	sw2	sw3
EXECUTE	n1	n2	n3	SWI	NOP	NOP	NOP	JSR	—	sw1	sw2
INSTRUCTION BEING DECODED	1										

i = INTERRUPT REQUEST
i* = INTERRUPT REQUEST GENERATED BY SWI
ii1 = FIRST VECTOR OF INTERRUPT i
ii3 = FIRST SWI VECTOR (ONE-WORD JSR)
ii4 = SECOND SWI VECTOR
n = NORMAL INSTRUCTION WORD
n4 = SWI
sw = INSTRUCTIONS PERTAINING TO THE SWI LONG INTERRUPT ROUTINE

Figure 8-1 Interrupting an SWI

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

CAUTION

On all level-sensitive interrupts, the interrupt must be externally released before interrupts are internally re-enabled, or the processor will be interrupted repeatedly until the interrupt is released.

The edge-sensitive NMI is generated on the first transition to 10 V on the $\overline{\text{IRQB}}$ pin after the last time the NMI interrupt was serviced or the DSP was reset. The NMI is a priority 3 interrupt and cannot be masked. Only $\overline{\text{RESET}}$ and illegal instruction have higher priority than NMI. NMI is reserved for hardware development and should not be used in an application. Repeated use may damage the integrated circuit.

8.2.1.2 Software Interrupt Source

There are two software interrupt sources — illegal instruction interrupt (III) and SWI. The III is a nonmaskable interrupt (IPL 3), which is serviced immediately following the execution of the illegal instruction or the attempted execution of an illegal instruction (any undefined operation code). IIIs are fatal errors. Only a long interrupt routine should be used for the III routine; RTI or RTS should not be used at the end of the interrupt routine since return from the III to the main code should not be attempted. During the III service, the JSR located in the III vector will normally stack the address of the illegal instruction (this is the reason why return should not be attempted (see Figure 8-2). The user may examine the stack (using MOVE SSH,dest) to locate the offending illegal instruction. The illegal instruction is useful for triggering the illegal interrupt service to see if the III routine is capable of recovery from illegal instructions.

There are two cases in which the stacked address will not point to the illegal instruction:

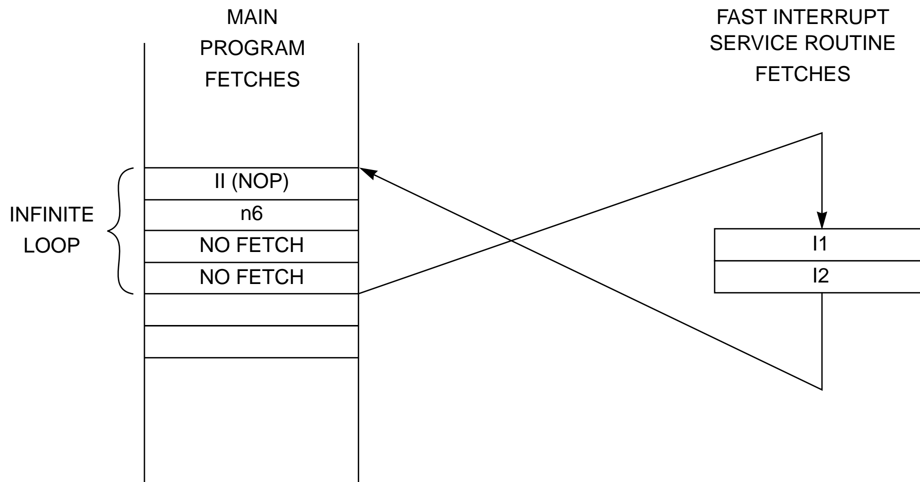
1. If the illegal instruction is one of the two instructions at an interrupt vector location and is fetched during a regular interrupt service, the processor will stack the address of the next sequential instruction in the normal instruction flow (the regular return address of the interrupt routine that had the illegal opcode in its vector).

2. If the illegal instruction follows an REP instruction (see Figure 8-3), the DSP will effectively execute the illegal instruction as a repeated NOP and the interrupt vector will then be inserted in the pipeline. The next instruction will be fetched but will not be decoded or executed. The processor will stack the address of the next sequential instruction, which is two instructions after the illegal instruction.

In DO loops, if the illegal instruction is in the loop address (LA) location and the instruction preceding it (i.e., at LA-1) is being interrupted, the loop counter (LC) will be decremented as if the loop had reached the LA instruction. When the interrupt service ends and the instruction flow returns to the loop, the illegal instruction will be refetched (since

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



(a) Instruction Fetches from Memory

ILLEGAL INSTRUCTION INTERRUPT RECOGNIZED AS PENDING (cycle 8)

ILLEGAL INSTRUCTION INTERRUPT RECOGNIZED AS PENDING (cycle 14)

INTERRUPT CONTROL CYCLE 1								i						
INTERRUPT CONTROL CYCLE 2									i					
FETCH		n1	n2	n3	n4	n5	n6	—	—	ii1	ii2	n5		
DECODE			n1	n2	n3	n4	II	—	—	—	ii1	ii2	II	
EXECUTE				n1	n2	n3	n4	NOP	—	—	—	ii1	ii2	NOP
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11	12	13	14

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
II = ILLEGAL INSTRUCTION
n = NORMAL INSTRUCTION WORD

(b) Program Controller Pipeline

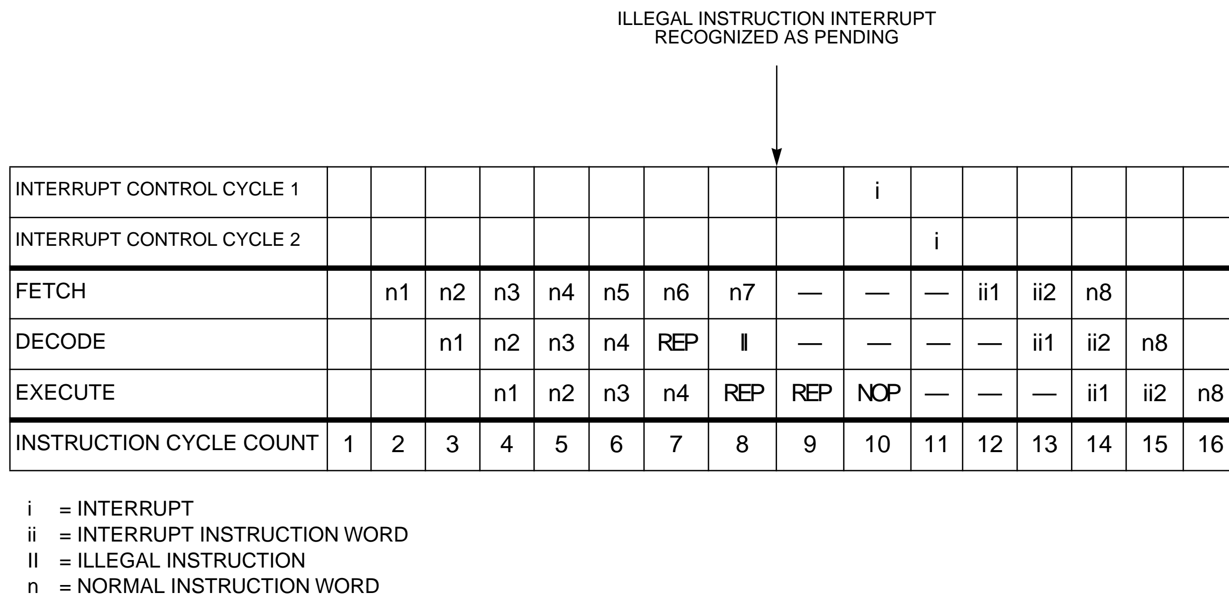
Figure 8-2 Illegal Instruction Interrupt Serviced by a Fast Interrupt

it is the next sequential instruction in the flow). The loop state machine will again decrement LC because the LA instruction is being executed. At this point, the illegal instruction will trigger the III. The result is that the loop state machine decrements LC twice in one loop due to the presence of the illegal opcode at the LA location.

SWI is a nonmaskable interrupt (IPL 3), which is serviced immediately following the SWI

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



INTERRUPT CONTROL CYCLE 1										i						
INTERRUPT CONTROL CYCLE 2											i					
FETCH		n1	n2	n3	n4	n5	n6	n7	—	—	—	ii1	ii2	n8		
DECODE			n1	n2	n3	n4	REP	II	—	—	—	—	ii1	ii2	n8	
EXECUTE				n1	n2	n3	n4	REP	REP	NOP	—	—	—	ii1	ii2	n8
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
II = ILLEGAL INSTRUCTION
n = NORMAL INSTRUCTION WORD

Figure 8-3 Repeated Illegal Instruction

instruction execution. A long interrupt service routine is usually used. The difference between an SWI and a JSR instruction is that the SWI sets the interrupt mask to prevent interrupts below IPL 3 from being serviced. Masking out lower level interrupts makes the SWI very useful for setting breakpoints in monitor programs. The JSR instruction does not affect the interrupt mask.

8.2.1.3 Other Interrupt Sources

Other interrupt sources include the stack error interrupt and trace interrupt (IPL3 interrupts).

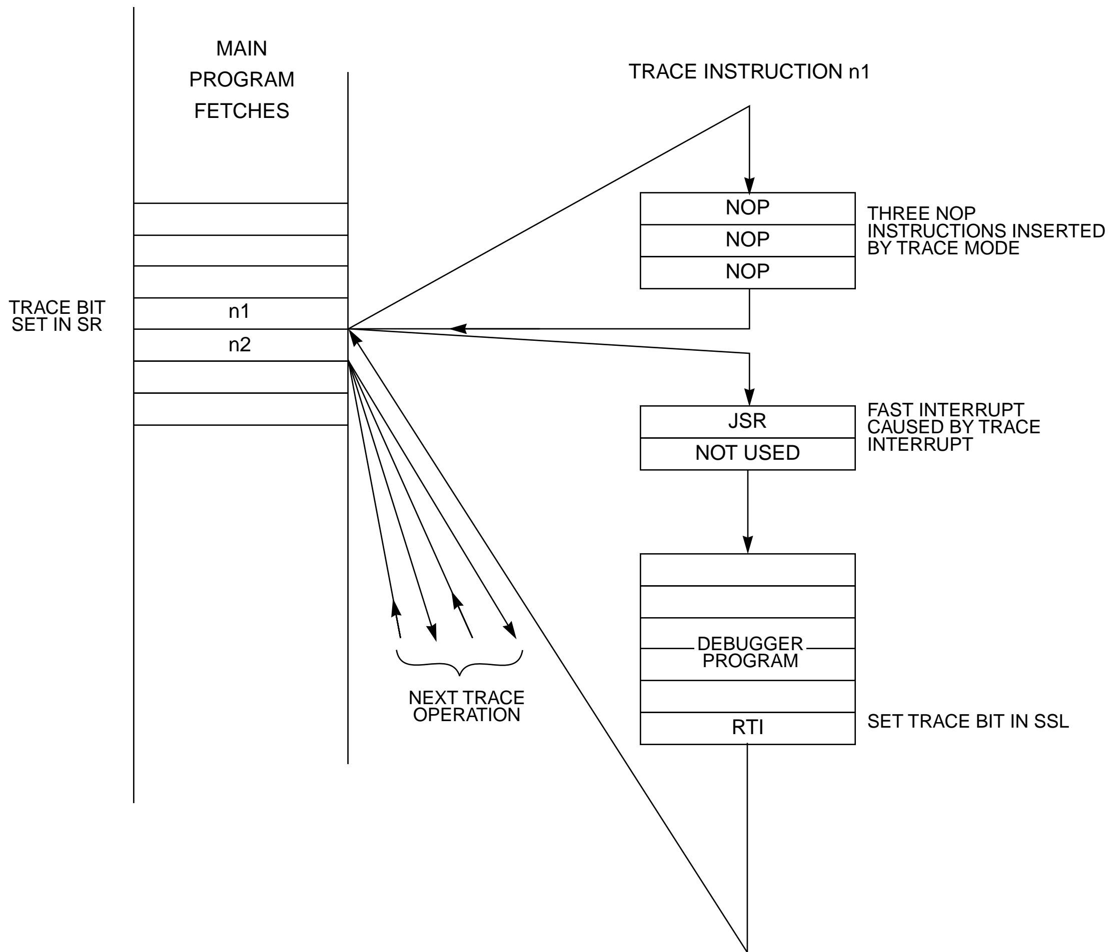
An overflow or underflow of the system stack (SS) causes a stack error interrupt (see SECTION 6 PROGRAM CONTROL UNIT for additional information on the stack error flag). The stack error interrupt is caused by a nonrecoverable error condition and is vectored to P:$0002. Since the stack error is nonrecoverable, a long interrupt should be used to service the interrupt, and the service routine should not end in an RTI. Executing an RTI instruction ‘‘pops’’ the stack, which has been corrupted.

The DSP56000/DSP56001 includes a facility for instruction-by-instruction tracing as a program development aid. This trace mode (entered by setting the trace bit in the SR) generates a trace exception after each instruction executed (see Figure 8-4), which can be used by a debugger program to monitor the execution of a program.

The trace mode is entered by setting the trace bit in the SR. A trace exception is generated after executing each instruction executed while the trace bit is set. When servicing

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



(a) Instruction Fetches from Memory

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING

INTERRUPT CONTROL CYCLE 1	i												i					
INTERRUPT CONTROL CYCLE 2		i													i			
FETCH		n1	NOP	NOP	NOP	JSR	—	TRACE PROGRAM			RTI	—	n2	NOP	NOP	NOP		
DECODE			n1	NOP	NOP	NOP	JSR	NOP	TRACE PROGRAM			RTI	NOP	n2	NOP	NOP	NOP	
EXECUTE				n1	NOP	NOP	NOP	JSR	NOP	TRACE PROGRAM			RTI	NOP	n2	NOP	NOP	NOP
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
II = ILLEGAL INSTRUCTION
n = NORMAL INSTRUCTION WORD

(b) Program Controller Pipeline

Figure 8-4 Trace Exception

the trace exception, it is expected that a JSR will be encountered in the trace vector loca-

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

tions, thereby forming a long interrupt routine. The JSR causes the SR to be stacked and the trace bit in the SR to be cleared (clearing the trace bit in the SR prevents tracing while executing the trace exception service routine). This service routine should end with an RTI instruction, which restores the SR (with the trace bit set) from the SS, causing the next instruction to be traced. The pipeline must be flushed to allow each sequential instruction to be traced. Three instruction cycles are appended by the tracing facility to the end of each instruction traced (these are the three NOP instructions shown in Figure 8-4) flushing the pipeline and allowing the next trace interrupt to follow the next sequential interrupt.

During tracing, the REP instruction and the instruction being repeated are considered a single two-word instruction. That is, only after executing the REP instruction and all the repeats of the next instruction will the trace exception be generated.

Fast interrupts can not be traced because they are uninterruptable. Long interrupts will not be traced (unless the trace mode is entered in the subroutine) because the SR is pushed on the stack and the trace bit is cleared. Tracing is resumed upon returning from a long interrupt because the trace bit is restored when the SR is restored. Interrupts are not likely to occur during tracing because only an interrupt with a higher IPL can interrupt during a trace operation. While executing the program being traced, the trace interrupt will always be pending and will win the interrupt arbitration. During the trace interrupt, the interrupt mask is set to reject interrupts below IPL3.

8.2.2 Interrupt Priority Structure

Four levels of interrupt priority are provided. IPLs numbered 0, 1, and 2 are maskable (level 0 is the lowest level). Level 3 (highest level) is nonmaskable. The only IPL 3 interrupts are reset, III, NMI, stack error, trace, and SWI. The interrupt mask bits (I1, I0) in the SR reflect the current processor priority level and indicate the IPL needed for an interrupt source to interrupt the processor (see Table 8-3). Interrupts are inhibited for all priority levels less than the current processor priority level. However, level 3 interrupts are not maskable and therefore can always interrupt the processor.

Table 8-3 Status Register Interrupt Mask Bits

I1	I0	Exceptions Permitted	Exceptions Masked
0	0	IPL 0, 1, 2, 3	None
0	1	IPL 1, 2, 3	IPL 0
1	0	IPL 2, 3	IPL 0, 1
1	1	IPL 3	IPL 0, 1, 2

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.2.2.1 Interrupt Priority Levels

The IPL for each on-chip peripheral device (HI, SSI, SCI) and for each external interrupt source ($\overline{IRQA}$, $\overline{IRQB}$) can be programmed under software control. Each on-chip or external peripheral device can be programmed to one of the three maskable priority levels (IPL 0, 1, or 2). IPLs are set by writing to the interrupt priority register shown in Figure 8-5. This read/write register specifies the IPL for each of the interrupting devices (HI, SSI, SCI, $\overline{IRQA}$, $\overline{IRQB}$). In addition, this register specifies the trigger mode of both external interrupt

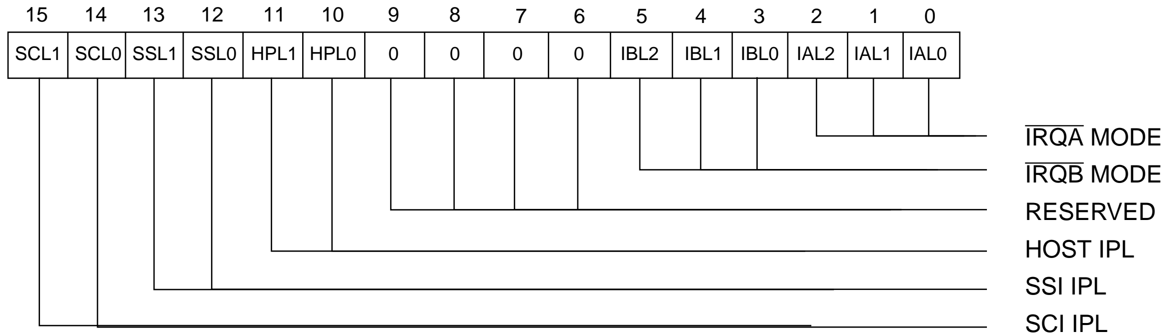


Figure 8-5 Interrupt Priority Register (Addr X:$FFFF)

sources and is used to enable or disable the individual external interrupts. This register is cleared on $\overline{RESET}$ or by the reset instruction. Table 8-4 defines the IPL bits. Table 8-5 defines the external interrupt trigger mode bits.

Table 8-4 Interrupt Priority Level

Table 8-5 External Interrupt

xxL1	xxL0	Enabled	IPL
0	0	No	—

8.2.2.2 Exception Priorities within an IPL

If more than one exception is pending when an instruction is executed, the interrupt with the highest priority level is serviced first. When multiple interrupt requests having the same IPL are pending, a second fixed-priority structure within that IPL determines which interrupt is serviced. The fixed priority of interrupts within an IPL and the interrupt enable bits for all interrupts are shown in Table 8-6. The interrupt enable bits for the HI, SSI, and SCI are located in the control registers associated with their respective on-chip peripherals.

8.2.3 Instructions Preceding the Interrupt Instruction Fetch

The following one-word instructions are aborted when they are fetched in the cycle preceding the fetch of the first interrupt instruction word — REP, STOP, WAIT, RESET, RTI, RTS, Jcc, JMP, JScc, and JSR.

Two-word instructions are aborted when the first interrupt instruction word fetched will replace the fetch of the second word of the two-word instruction. Aborted instructions are refetched again when program control returns from the interrupt routine. The PC is adjusted appropriately before the end of the decode cycle of the aborted instruction.

If the first interrupt word fetch occurs in the cycle following the fetch of a one-word instruction not previously listed or the second word of a two-word instruction, that instruction will complete normally before the start of the interrupt routine.

The following cases have been identified where service of an interrupt might encounter an extra delay:

1. If a long interrupt routine is used to service an SWI, then the processor priority level is set to 3. Thus,all interrupts except other level-3 interrupts are disabled until the SWI service routine terminates with an RTI (unless the SWI service routine software lowers the processor priority level).

2. While servicing an interrupt, the next interrupt service will be delayed according to the following rule: after the first interrupt instruction word reaches the instruction decoder, at least three more instructions will be decoded before

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 8-6 Exception Priorities within an IPL

Priority	Exception	Enabled By	Bit No.	X Data Memory Address
Level 3 (Nonmaskable)				
Highest	Hardware $\overline{\text{RESET}}$	—	—	—
	III	—	—	—
	NMI	—	—	—
	Stack Error	—	—	—
	Trace	—	—	—
Lowest	SWI	_	—	—
Levels 0, 1, 2 (Maskable)				
Highest	$\overline{\text{IRQA}}$ (External Interrupt)	$\overline{\text{IRQA}}$ Mode Bits	0 and 1	$FFFF
	$\overline{\text{IRQB}}$ (External Interrupt)	$\overline{\text{IRQB}}$ Mode Bits	3 and 4	$FFFF
	Host Command Interrupt	HCIE	2	$FFE8
	Host Receive Data Interrupt	HRIE	0	$FFE8
	Host Transmit Data Interrupt	HTIE	1	$FFE8
	SSI RX Data with Exception Interrupt	RIE	15	$FFED
	SSI RX Data Interrupt	RIE	15	$FFED
	SSI TX Data with Exception Interrupt	TIE	14	$FFED
	SSI TX Data Interrupt	TIE	14	$FFED
	SCI RX Data with Exception Interrupt	RIE	11	$FFF0
	SCI RX Data Interrupt	RIE	11	$FFF0
	SCI TX Data Interrupt	TIE	12	$FFF0
	SCI Idle Line Interrupt	ILIE	10	$FFF0
Lowest	SCI Timer Interrupt	TMIE	13	$FFF0

decoding the next first interrupt instruction word. If any one pair of instructions-being counted is the REP instruction followed by an instruction to be repeated, then the combination is counted as two instructions independent of the number of repeats done. Sequential REP combinations will cause pending interrupts to be rejected and can not be interrupted until the sequence of REP

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

combinations ends.

3. The following instructions are not interruptable: SWI, STOP, WAIT, and RESET.
4. The REP instruction and the instruction being repeated are not interruptable.
5. If the trace bit in the SR is set, the only interrupts that will be processed are the hardware RESET, III,NMI, stack error, and trace. Peripheral and external interrupt requests will be ignored. The interrupt generated by the SWI instruction will be ignored.

During an interrupt instruction fetch, two instruction words are fetched — the first from the interrupt starting address and the second from the interrupt starting address +1 locations.

8.2.4 Interrupt Types

Two types of interrupt routines may be used: fast and long. The fast routine consists of the two automatically inserted interrupt instruction words. These words can contain any unrestricted, single two-word instruction or any two one-word instructions (see A.8 INSTRUCTION SEQUENCE RESTRICTIONS for a list of restrictions). Fast interrupt routines are never interruptable.

CAUTION

Status is not preserved during a fast interrupt routine; therefore, instructions that modify status should not be used at the interrupt starting address and interrupt starting address +1.

If one of the instructions in the fast routine is a JSR, then a long interrupt routine is formed. The following actions occur during execution of the JSR instruction when it occurs in the interrupt starting address or in the interrupt starting address +1:

1. The PC (containing the return address) and the SR are stacked.
2. The loop flag is reset.
3. The scaling mode bits are reset.
4. The IPL is raised to disallow further interrupts at the same or lower levels (except that hardware $\overline{\text{RESET}}$, NMI, stack error, trace, and SWI can always interrupt).
5. The trace bit in the SR is cleared.

The long interrupt routine should be terminated by an RTI. Long interrupt routines are interruptable by higher priority interrupts.

8.2.5 Interrupt Arbitration

External interrupts are internally synchronized with the processor clock (takes up to three T cycles) before their interrupt-pending flags are set. Each external interrupt and internal interrupt has its own flag. After each instruction is executed, all interrupts are arbitrated — i.e., all hardware interrupts that have been latched into their respective interrupt-pending flags and all internal interrupts. During arbitration, each interrupt's IPL is compared with the interrupt mask in the SR, and the interrupt is either allowed or disallowed. The remaining interrupts are prioritized according to the priority shown in Table 8-6, and the highest priority interrupt is chosen. The interrupt vector is then calculated so that the program interrupt controller can fetch the first interrupt instruction. Interrupt arbitration and control, which occurs concurrently with the fetch-decode-execute cycle, takes two instruction cycles. Interrupts from a given source are not buffered. The interrupt-pending flag for the chosen interrupt is not cleared until the second interrupt vector of the chosen interrupt is being fetched. A new interrupt from the same source will not be accepted for the next interrupt arbitration until that time.

The internal interrupt acknowledge signal is used to clear the edge-triggered interrupt flags, the HC bit in the host port, the SCI timer interrupt, and the internal latches of the stack error, NMI, SWI, and trace interrupts. Peripheral interrupt requests that need a read/write action to some register do not receive this signal, and those interrupts will remain pending until their registers are read/written. Also, level-triggered interrupts will not be cleared. The acknowledge signal will be generated after generation of the interrupt vectors, not before.

8.2.6 Interrupt Instruction Fetch

The interrupt controller generates an interrupt instruction fetch address, which points to the first instruction word of a two-word interrupt routine. This address is used for the next instruction fetch, instead of the contents of the PC, and the interrupt instruction fetch address +1 is used for the subsequent instruction fetch. While the interrupt instructions are being fetched, the PC is inhibited from being updated. After the two interrupt words have been fetched, the PC is used for any subsequent instruction fetches.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

After both interrupt vectors have been fetched, they are guaranteed to be executed. This is true even if the instruction that is currently being executed is a change-of-flow instruction (i.e., JMP, JSR, etc.) that would normally ignore the instructions in the pipe. After the interrupt instruction fetch, the PC will point to the instruction that would have been fetched if the interrupt instructions had not been inserted.

8.2.7 Interrupt Instruction Execution

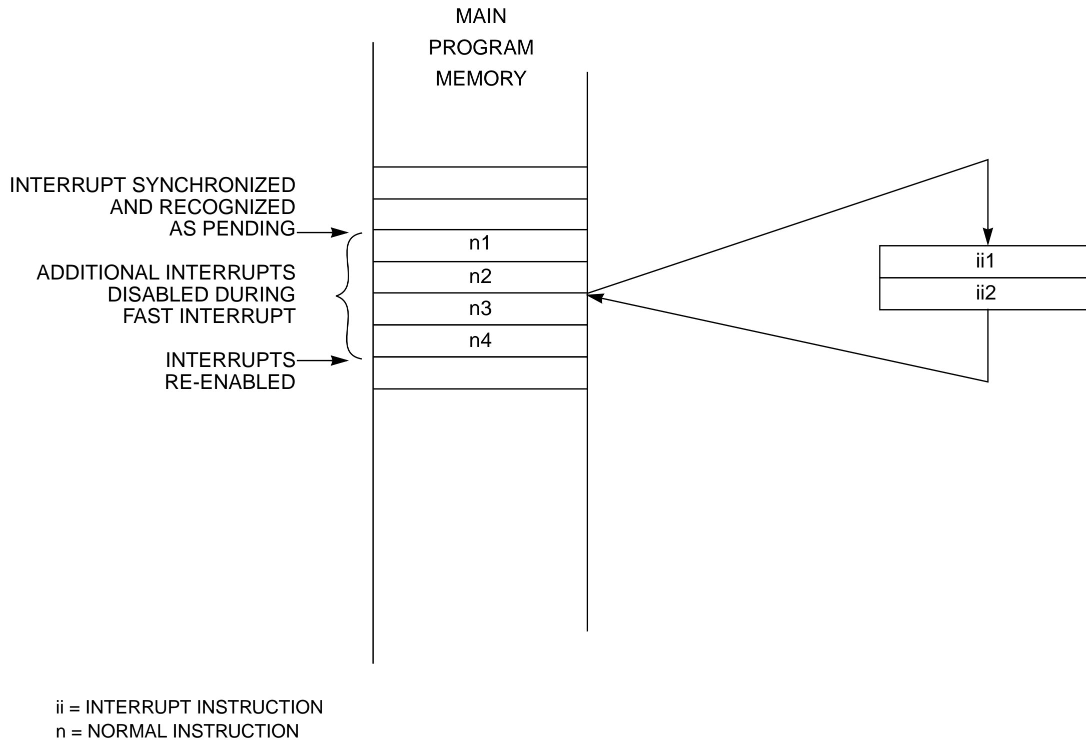
Interrupt instruction execution is considered ''fast'' if neither of the instructions of the interrupt service routine cause a change of flow. A JSR within a fast interrupt routine forms a long interrupt, which is terminated with an RTI instruction to restore the PC and SR from the stack and return to normal program execution. Reset is a special exception, which will normally contain only a JMP instruction at the exception start address. At the programmer's option, almost any instruction can be used in the fast interrupt routine. The restricted instructions include SWI, STOP, and WAIT. Figure 8-6 and Figure 8-8 show the fast and the long interrupt service routines. The fast interrupt executes only two instructions and then automatically resumes execution of the main program; whereas, the long interrupt must be told to return to the main program by executing an RTI instruction.

Figure 8-6 illustrates the effect of a fast interrupt routine in the stream of instruction fetches.

Figure 8-7 shows the sequence of instruction decodes between two fast interrupts. Four decodes occur between the two interrupt decodes (two after the first interrupt and two preceding the second interrupt). The requirement for these four decodes establishes the

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



(a) Instruction Fetches from Memory

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING

INTERRUPTS RE-ENABLED

INTERRUPT CONTROL CYCLE 1	i							
INTERRUPT CONTROL CYCLE 2		i						
FETCH	n1	n2	ii1	ii2	n3	n4		
DECODE		n1	n2	ii1	ii2	n3	n4	
EXECUTE			n1	n2	ii1	ii2	n3	n4
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
n = NORMAL INSTRUCTION WORD

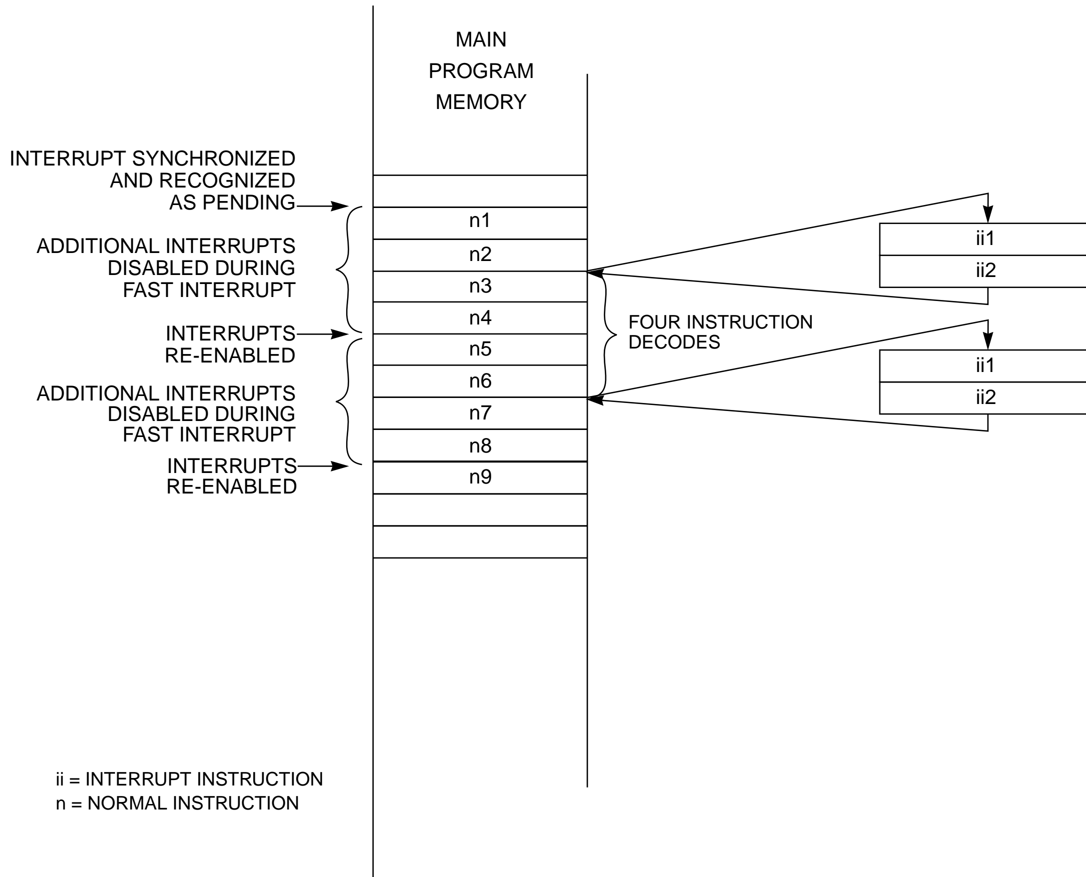
(b) Program Controller Pipeline

Figure 8-6 Fast Interrupt Service Routine

maximum rate at which the DSP56000/DSP56001 will respond to interrupts — namely, one interrupt every six instructions (six instruction cycles if all six instructions are one instruction cycle each). Since some instructions take more than one instruction cycle, the

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



(a) Instruction Fetches from Memory

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING

INTERRUPTS RE-ENABLED

6 I_{cyc}

INTERRUPT CONTROL CYCLE 1	i						i					
INTERRUPT CONTROL CYCLE 2		i						i				
FETCH	n1	n2	ii1	ii2	n3	n4	n5	n6	ii1	ii2		
DECODE		n1	n2	ii1	ii2	n3	n4	n5	n6	ii1	ii2	
EXECUTE			n1	n2	ii1	ii2	n3	n4	n5	n6	ii1	ii2
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11	12

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
n = NORMAL INSTRUCTION WORD

(b) Program Controller Pipeline

Figure 8-7 Two Consecutive Fast Interrupts

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

MAIN PROGRAM FETCHES

LONG INTERRUPT SERVICE ROUTINE FETCHES (STARTS WITH A FAST INTERRUPT)

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING

n1
n2
n3
n4

ii1
ii2

JSR CAN BE IN EITHER LOCATION TO FORM A LONG INTERRUPT

ii3
ii4
INTERRUPT ROUTINE
ii7
RTI

PROGRAM COUNTER RESUMES OPERATION

INTERRUPTS RE-ENABLED

EXPLICIT RETURN FROM INTERRUPT (SHOULD BE RTI)

(a) Instruction Fetches from Memory

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING

INTERRUPTS RE-ENABLED

INTERRUPT CONTROL CYCLE 1	i														
INTERRUPT CONTROL CYCLE 2		i													
FETCH	n1	n2	ii1	ii2	ii3	ii4	ii5	ii6	ii7	RTI	—	n3	n4		
DECODE		n1	n2	ii1	ii2	ii3	ii4	ii5	ii6	ii7	RTI	NOP	n3	n4	
EXECUTE			n1	n2	ii1	ii2	ii3	ii4	ii5	ii6	ii7	RTI	NOP	n3	n4
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
n = NORMAL INSTRUCTION WORD

(b) Program Controller Pipeline

Figure 8-8 Long Interrupt Service Routine

minimum number of instructions between two interrupts may be more than six instruction cycles.

For More Information On This Product, Go to: www.freescale.com
Freescale Semiconductor, Inc.

Execution of a fast interrupt routine always conforms to the following rules:

1. A JSR to the starting address of the interrupt service routine is not located at one of the two interrupt vector addresses.
2. The processor status is not saved.
3. The fast interrupt routine may (but should not) modify the status of the normal instruction stream.
4. The fast interrupt routine may contain any single two-word instruction or any two one-word instructions except SWI, STOP, and WAIT.
5. The PC, which contains the address of the next instruction to be executed in normal processing remains unchanged during a fast interrupt routine.
6. The fast interrupt returns without an RTI.
7. Normal instruction fetching resumes using the PC following the completion of the fast interrupt routine.
8. A fast interrupt is not interruptable.
9. A JSR instruction within the fast interrupt routine forms a long interrupt routine.
10. The primary application is to move data between memory and I/O devices.

Execution of a long interrupt routine always adheres to the following rules:

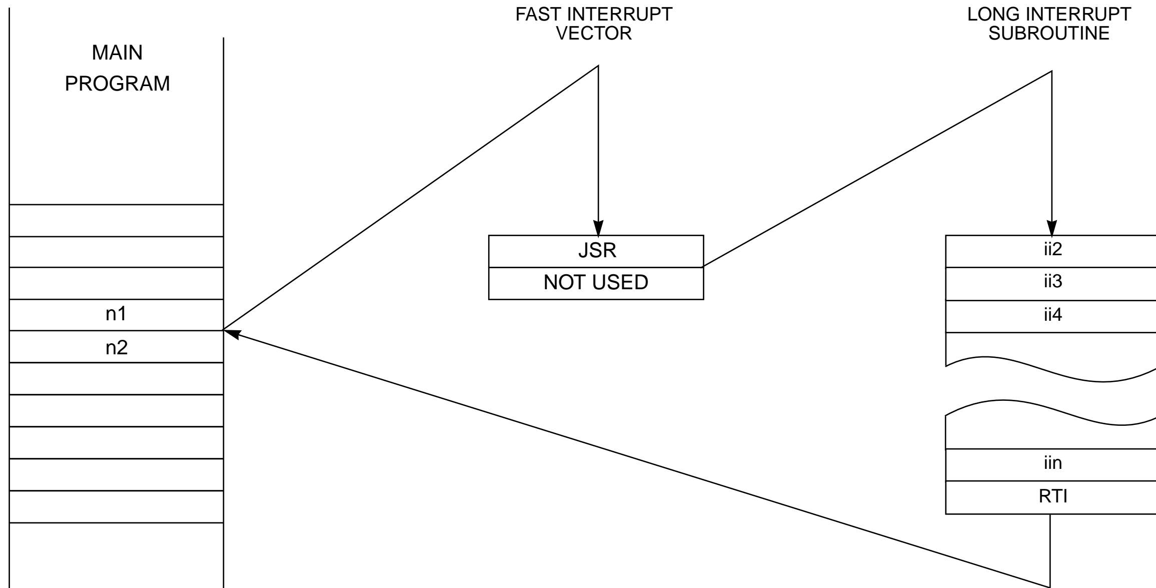
1. A JSR to the starting address of the interrupt service routine is located at one of the two interrupt vector addresses.
2. During execution of the JSR instruction, the PC and SR are stacked. The interrupt mask bits of the SR are updated to mask interrupts of the same or lower priority. The loop flag, trace bit, and scaling mode bits are reset.
3. The first instruction word of the next interrupt service (of higher IPL) will reach the decoder only after the decoding of at least four instructions following the decoding of the first instruction of the previous interrupt.
4. The interrupt service routine can be interrupted — i.e., nested interrupts are supported.
5. The long interrupt routine, which can be any length, should be terminated by an RTI, which restores the PC and SR from the stack.

Figure 8-8 illustrates the effect of a long interrupt routine on the instruction pipeline. A short JSR (a JSR with 12-bit absolute address) is used to form the long interrupt routine. For this example, word 6 of the long interrupt routine is an RTI. The point at which interrupts are re-enabled and subsequent interrupts are allowed is shown to illustrate the noninterruptable nature of the early instructions in the long interrupt service routine.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Either one of the two instructions of the fast interrupt can be the JSR instruction that forms the long interrupt. Figure 8-9 and Figure 8-10 show the two possible cases. If the first fast interrupt vector instruction is the JSR, the second instruction is never used.



(a) Instruction Fetches from Memory

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING

INTERRUPTS RE-ENABLED

INTERRUPT CONTROL CYCLE 1	i												
INTERRUPT CONTROL CYCLE 2		i											
FETCH		n1	JSR	—	ii2	ii3	ii4	iin	RTI	—	n2		
DECODE			n1	JSR	NOP	ii2	ii3	ii4	iin	RTI	NOP	n2	
EXECUTE				n1	JSR	NOP	ii2	ii3	ii4	iin	RTI	NOP	n2
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11	12	13

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
n = NORMAL INSTRUCTION WORD

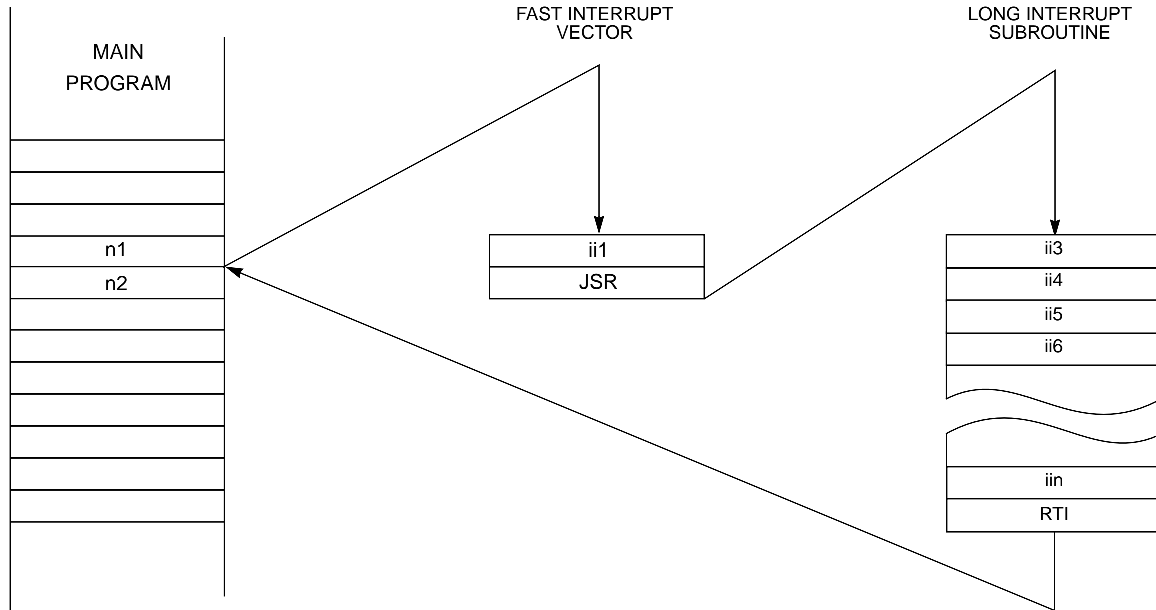
(b) Program Controller Pipeline

Figure 8-9 JSR First Instruction of a Fast Interrupt

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

An REP instruction is treated as a single two-word instruction, regardless of how many times it repeats the second instruction of the pair. Instruction fetches are suspended and will be reactivated only after the LC is decremented to one (see Figure 8-11). During the execution of n2 in Figure 8-11, no interrupts will be serviced. When LC finally decrements to one, the fetches are reinitiated, and pending interrupts can be serviced.



(a) Instruction Fetches from Memory

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING

INTERRUPTS RE-ENABLED

NTERRUPT CONTROL CYCLE 1	i														
NTERRUPT CONTROL CYCLE 2		i													
ETCH		n1	ii1	JSR	—	ii3	ii4	ii5		iin	RTI	—	n2		
ECODE			n1	ii1	JSR	NOP	ii3	ii4	ii5	ii6	iin	RTI	NOP	n2	
XECUTE				n1	ii1	JSR	NOP	ii3	ii4	ii5	ii6	iin	RTI	NOP	n2
NSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

= INTERRUPT
i = INTERRUPT INSTRUCTION WORD
= NORMAL INSTRUCTION WORD

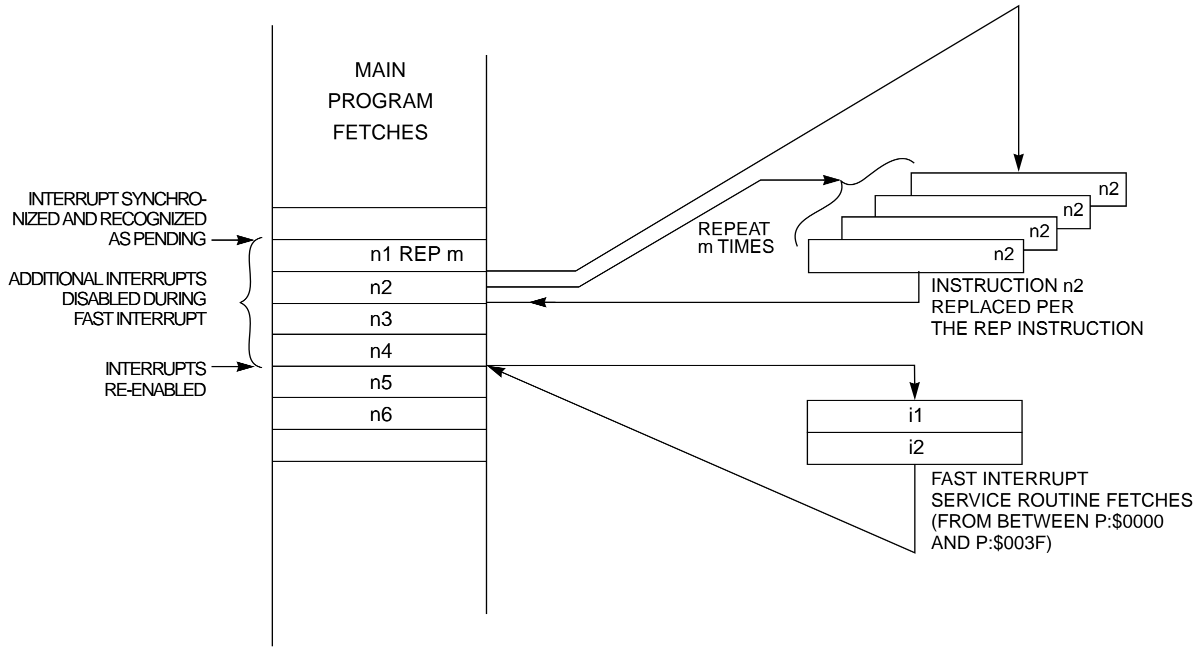
(b) Program Controller Pipeline

Figure 8-10 JSR Second Instruction of a Fast Interrupt

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



i = INTERRUPT INSTRUCTION
n = NORMAL INSTRUCTION

(a) Instruction Fetches from Memory

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING (cycle 1)

INTERRUPTS RE-ENABLED (cycle 7)

INTERRUPT CONTROL CYCLE 1	i						i					
INTERRUPT CONTROL CYCLE 2		i%						i				
FETCH	REP	n2	n3					n4	ii1	ii2	n5	n6
DECODE		REP	NOP	n2	n2	n2	n2	n3	n4	ii1	ii2	n5
EXECUTE			REP	NOP	n2	n2	n2	n2	n3	n4	ii1	ii2
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11	12

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
n = NORMAL INSTRUCTION WORD
i% = INTERRUPT REJECTED

(b) Program Controller Pipeline

Figure 8-11 Interrupting an REP Instruction

Sequential REP packages will cause pending interrupts to be rejected until the sequence

For More Information On This Product,
Go to: www.freescale.com

of REP packages ends. REP packages are not interruptable because the instruction being repeated is not refetched. While that instruction is repeating, no instructions are fetched or decoded, and an interrupt can not be inserted. For example, in Figure 8-12, if n1, n3, and n5 are all REP instructions, no interrupts will be serviced until the last REP instruction (n5 and its repeated instruction, n6) completes execution.

8.3 RESET PROCESSING STATE

The reset processing state is entered in response to the external RESET pin being asserted (a hardware reset). Upon entering the reset state (see Figure 8-13): 1) internal peripheral devices are reset, and their pins revert to general-purpose I/O pins; 2) the modifier registers are set to $FFFF; 3) the interrupt priority register is cleared; 4) the BCR is set to $FFFF, thereby inserting 15 wait states in all external memory accesses; 5) the stack pointer is cleared; 6) the scaling mode, trace mode, loop flag, and condition code bits of the SR are cleared, and the interrupt mask bits of the SR are set; 7) the data ROM enable bit, the stop delay bit, and the memory strobe bit are cleared; and 8) the DSP remains in the reset state until RESET is deasserted. Upon leaving the reset state 9), the chip operating mode bits of the OMR are loaded from the external mode select pins (MODA, MODB), and 10) program execution begins at program memory address $E000 in normal expanded mode or at $0000 in all other operation modes. The first instruction must be fetched and then decoded before executing. Therefore, the first instruction execution is two instruction cycles after the first instruction fetch.

Figure 8-14 is a copy of the output from the DSP56000/DSP56001 simulator showing all of the DSP56000/DSP56001 registers before the hardware reset and showing only the registers that were written by the hardware reset after the reset occurred. The instructions executed are as follows:

1. Reset s — Resets the simulator.
2. Change OMR 0 — Puts the DSP56000/DSP56001 in mode 0.
3. Display all — Displays all registers. Note that OMR=$00.
4. Reset d — Is a hardware reset.
5. Display w — Causes the display command to only display the registers that were written in the last instruction.
6. Display — Displays the contents of the registers that were written by the hardware reset.

The OMR changed from $00 to $02, which is mode 2, because the MODA/$\overline{\text{IRQA}}$ and MODB/$\overline{\text{IRQB}}$ pins are set to a one and zero, respectively (binary 2) in the simulator. If the DSP had been in any other mode, the result would have been the same. The X: memory locations written to are the memory locations of the peripheral registers. The internal peripheral registers are memory mapped between X:$FFC0 and X:$FFFF.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

MAIN PROGRAM FETCHES

INTERRUPT PENDING — INTERRUPT REJECTED — REP m n1
n2
INTERRUPT PENDING — INTERRUPT REJECTED — REP m n3
n4
INTERRUPT PENDING — INTERRUPT REJECTED — REP m n5
INTERRUPT PENDING — n6
n7
n8
n9

REPEAT m TIMES: n2, n2, n2
REPEAT m TIMES: n4, n4, n4
REPEAT m TIMES: n6, n6, n6
ii1
ii2

(a) Instruction Fetches from Memory

INTERRUPT SYNCHRONIZED AND RECOGNIZED AS PENDING

INTERRUPTS RE-ENABLED

INTERRUPT CONTROL CYCLE 1	i															i						
INTERRUPT CONTROL CYCLE 2		i%															i					
FETCH	REP	n2	REP				n4	REP				n6	n7				n8	ii1	ii2	n9		
DECODE		REP	NOP	n2	n2	n2	REP	NOP	n4	n4	n4	REP	NOP	n6	n6	n6	n7	n8	ii1	ii2	n9	
EXECUTE			REP	NOP	n2	n2	n2	REP	NOP	n4	n4	n4	REP	NOP	n6	n6	n6	n7	n8	ii1	ii2	n9
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
n = NORMAL INSTRUCTION WORD
i% = INTERRUPT REJECTED

(b) Program Controller Pipeline

Figure 8-12 Interrupting Sequential REP Instructions

The internal peripheral devices (HI, SSI, SCI, and ports A, B, and C) can be reset by sev-

For More Information On This Product, Go to: www.freescale.com
Freescale Semiconductor, Inc.

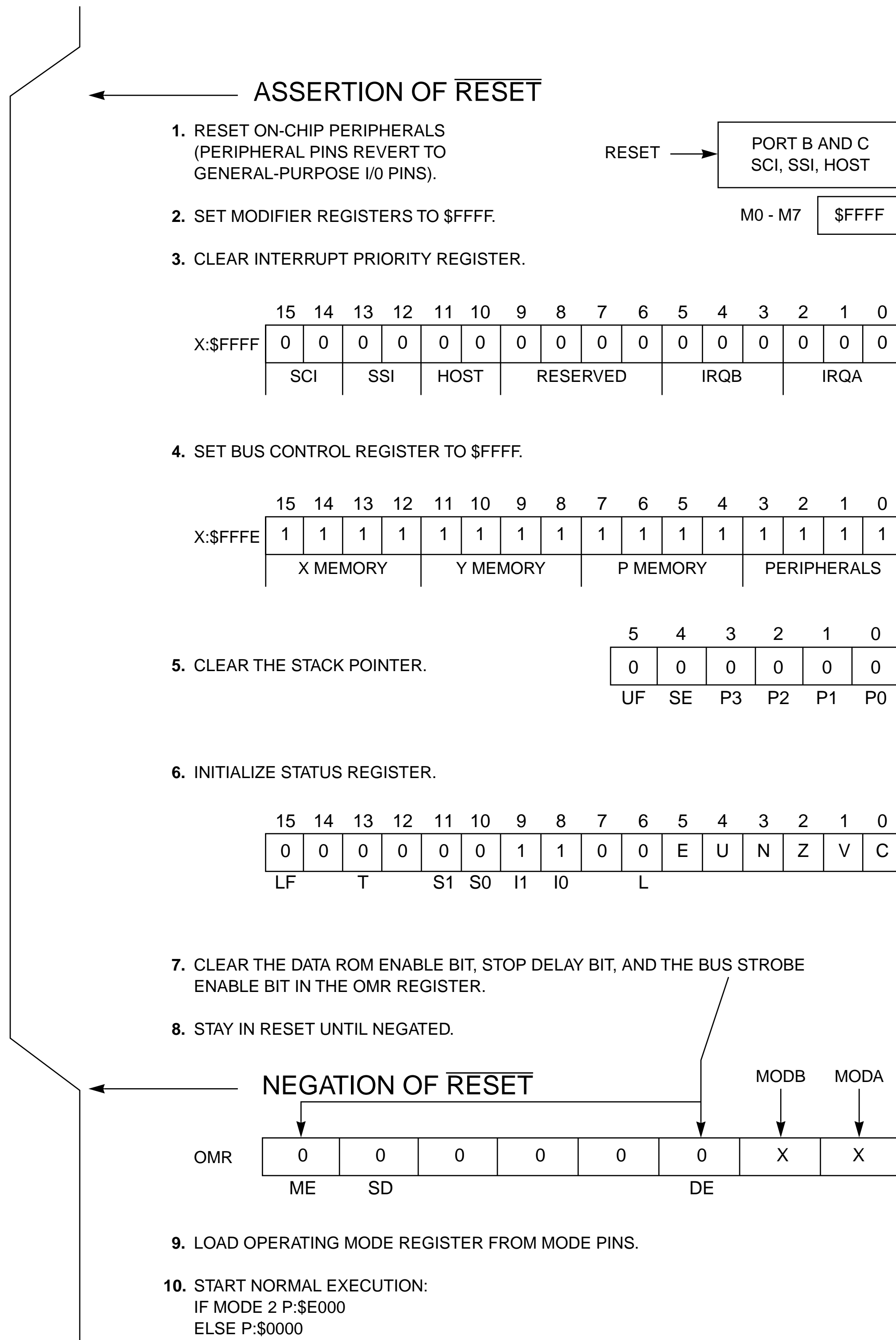


Figure 8-13 Reset Sequence

eral methods — hardware (HW) reset, software (SW) reset, individual (I) reset, and stop

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

```
reset s
change omr 0
display all

  x=        $000000000000  y=        $000000000000
  a=      $00000000000000  b=      $00000000000000
             x1=  $000000  x0=   $000000  r7=    $0000   n7= $0000   m7= $FFFF
             y1=  $000000  y0=   $000000  r6=    $0000   n6= $0000   m6= $FFFF
a2=   $00 a1=  $000000  a0=   $000000  r5=    $0000   n5= $0000   m5= $FFFF
b2=   $00 b1=  $000000  b0=   $000000  r4=    $0000   n4= $0000   m4= $FFFF
                                        r3=    $0000   n3= $0000   m3= $FFFF
pc=  $E00     sr=  $0300 omr=     $00  r2=    $0000   n2= $0000   m2= $FFFF
la= $0000     lc=  $0000               r1=    $0000   n1= $0000   m1= $FFFF
ssh= $0000   ssl=  $0000  sp=     $00  r0=    $0000   n0= $0000   m0= $FFFF
pbc=    $0 pbddr=  $0000 pbd=   $0000  pcd=   $0000 pcddr= $0000  pcc= $FFFF
ipr= $0000   bcr=  $FFFF htx= $000000  hrx= $000000   hsr=   $02  hcr=   $00
icr=   $00   cvr=    $12 isr=     $06  ivr=     $0F
rxh=   $00   rxm=    $00 rxl=     $00  txh=     $00   txm=   $00  txl=   $00
ssr=   $03   scr=  $0000 stx=     $00  srx=     $00  sccr= $0000  stxa=  $00
tsr=   $00 ssisr=    $40  tx= $000000  rx=  $000000   cra= $0000  crb= $0000
cyc=000000  ictr= 000000 cnt1=  000000 cnt2=  000000  cnt3=000000 cnt4=000000
P:$E000 000000        =NOP

reset d
display w
display

                                                                  m7= $FFFF
                                                                  m6= $FFFF
                                                                  m5= $FFFF
                                                                  m4= $FFFF
                                                                  m3= $FFFF
pc= $E000     sr=  $0300 omr=     $02                             m2= $FFFF
                                                                  m1= $FFFF
                          sp=     $00                             m0= $FFFF
pbc=    $0 pbddr=  $0000                              pcddr= $0000  pcc= $0000
ipr= $0000   bcr=  $FFFF                                hsr=   $02  hcr=   $00
icr=   $00   cvr=    $12 isr=     $06  ivr=     $0F
ssr=   $03   scr=  $0000                               sccr= $0000
           ssisr=    $40                                cra= $0000  crb= $0000
X:$FFE3     $000000
X:$FFE8     $000000  $000002
X:$FFEC     $000000  $000000  $000040
X:$FFE0     $000000  $000003  $000000
X:$FFFF     $000000
P:$E000 000000        =NOP
```

Figure 8-14 Reset When OMR=0

(ST) reset. Depending on the type of reset, the registers of these devices will be affected differently (see SECTION 9 PORT A, SECTION 10 PORT B, and SECTION 11 PORT C for additional information on the internal peripherals). Tables 8-7 – 8-11 show how each bit in these registers is affected by the various resets. The HI is programmed for both the DSP56000/DSP56001 side of the interface and the host processor side of the interface.

The symbols used are as follows:

HW – Hardware reset is caused by asserting the external pin $\overline{\text{RESET}}$.
SW – Software reset is caused by executing the RESET instruction.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

I – Individual reset is caused by all of the I/O pins for a given internal I/O device being configured for general-purpose I/O. These I/O devices are the HI, SSI, and SCI. The conditions for these resets are:

1. SSI individual reset occurs when port C control register bits 3 – 8 are set to zero.
2. SCI individual reset occurs when port C control register bits 0 – 2 are set to zero.
3. HI individual reset occurs when port B control register bit 0 is set to zero.

ST – Stop reset is caused by executing the STOP instruction.

1 – The bit is set during the xx reset.

0 – The bit is clear during the xx reset.

— – The bit is not changed during the xx reset.

The definitions for individual reset for the ports A, B, and C register settings during indi-

Table 8-7 HI Reset Effects — DSP56000/ DSP56001Programming Model

Register Name	Register Data Bits	HW Reset	SW Reset	I Reset	ST Reset
HCR X:$FFE8	HF(3-2)	0	0	—	—
	HCIE	0	0	—	—
	HTIE	0	0	—	—
	HRIE	0	0	—	—
	DMA	0	0	0	0
	HF (1-0)	0	0	0	0

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 8-8 HI Reset Effects — Host Processor Programming Model

Register Name	Register Data Bits	HW Reset	SW Reset	I Reset	ST Reset
ICR $0	INIT	0	0	0	0
	HM(1-0)	0	0	0	0
	TREQ	0	0	0	0
	RREQ	0	0	0	0
	HF(1-0)	0	0	0	0
CVR $1	HC	0	0	0	0
	HV (4-0)	$12	$12	$12	$12
ISR $2	HREQ	0	0	0	0
	DMA	0	0	0	0
	HF(3-2)	0	0	—	—
	TRDY	1	1	1	1
	TXDE	1	1	1	1
	RXDF	0	0	0	0
IVR $3	IV(7-0)	$0F	$0F	—	—
RX $5, 6, 7	RXH(23-16)	—	—	—	—
	RXM(15-8)	—	—	—	—
	RXL	—	—	—	—
TX $5, 6, 7	TXH(23-16)	—	—	—	—
	TXM(15-8)	—	—	—	—
	TXL(7-0)	—	—	—	—

vidual reset are shown in Table 8-11.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

Table 8-9 SSI Reset Effects

Register Name	Register Data Bits	HW Reset	SW Reset	I Reset	ST Reset
CRA X:$FFEC	WL(2-0)	0	0	—	—
	PSR	0	0	—	—
	DC(4-0)	0	0	—	—
	PM(7-0)	0	0	—	—
CRB X:$FFED	RIE	0	0	—	—
	TIE	0	0	—	—
	RE	0	0	—	—
	TE	0	0	—	—
	MOD	0	0	—	—
	GCK	0	0	—	—
	SYN	0	0	—	—
	FSL0	0	0	—	—
	FSL1	0	0	—	—
	SCKD	0	0	—	—
	SCD(2-0)	0	0	—	—
	OF(1-0)	0	0	—	—
SR X:$FFEE	RDF	0	0	0	0
	TDE	1	1	1	1
	ROE	0	0	0	0
	TUE	0	0	0	0
	RFS	0	0	0	0
	TFS	0	0	0	0
	IF(1-0)	0	0	0	0
RX X:$FFEF	RDR(23-0)	—	—	—	—
TX X:$FFEF	TDR(23-0)	—	—	—	—
SRSR*	RDR(23-0)	—	—	—	—
STSR**	RDR(23-0)	—	—	—	—

*SRSR—SSI serial receive shift register
**STSR—SSI serial transmit shift register

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 8-10 SCI Reset Effects

Register Name	Register Data Bits	HW Reset	SW Reset	I Reset	ST Reset
SCR X:$FFF0	SCKP	0	0	—	—
	TMIE	0	0	—	—
	TIE	0	0	—	—
	RIE	0	0	—	—
	ILIE	0	0	—	—
	TE	0	0	—	—
	RE	0	0	—	—
	WOMS	0	0	—	—
	RWU	0	0	—	—
	WAKE	0	0	—	—
	SBK	0	0	—	—
	SSFTD	0	0	—	—
	WDS(2-0)	0	0	—	—
SSR X:$FFF1	R8	0	0	0	0
	FE	0	0	0	0
	PE	0	0	0	0
	OR	0	0	0	0
	IDLE	0	0	0	0
	RDRF	0	0	0	0
	TDRE	1	1	1	1
	TRNE	1	1	1	1
SCCR X:$FFF2	TCM	0	0	—	—
	RCM	0	0	—	—
	SCP	0	0	—	—
	COD	0	0	—	—
	CD(11-0)	0	0	—	—
SRX X:$FFF4 X:$FFF5 X:$FFF6	STX(23-0) LOW MID HIGH	—	—	—	—
STX X:$FFF4 X:$FFF5 X:$FFF6 X:$FFF3	SRX(23-0) LOW MID HIGH STXA	—	—	—	—
SRSH*	SRSH(23-0)	—	—	—	—
STSH**	STSH(23-0)	—	—	—	—

*SRSH—SCI serial receive shift register
**STSH—SCI serial transmit shift register

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 8-11 Ports A, B, and C Reset Effects

Register Name	Register Data Bits	HW Reset	SW Reset	I Reset	ST Reset	Comments

8.4 WAIT PROCESSING STATE

The wait processing state is a low power-consumption state entered by execution of the WAIT instruction. In the wait state, the internal clock is disabled from all internal circuitry except the internal peripherals (e.g., the interrupt controller, the SCI, SSI, and HI). All internal processing is halted until an unmasked interrupt occurs or until the DSP is reset. The $\overline{BR}/\overline{BG}$ circuits remain active during the wait state.

The wait state is one of two low power-consumption states. As a general rule, the normal operating current for the DSP56000/DSP56001 is typically less than 100 ma for a 20.5-MHz clock. The current is typically reduced to less than 10 ma (for a 20.5-MHz clock) in the wait state and to less than 1.0 ma (independent of the clock frequency) in the stop state. See the DSP56001 Advance Information Data Sheet (ADI1290) for exact figures. There are several other ways that power can be reduced. Power consumption varies linearly with both clock frequency and power-supply voltage. Changing clock frequency from 20 MHz[lz to 4 MHz can reduce power consumption 75 percent (i.e., linearly with decreasing frequency). Changing the memory wait states from 0 to 15 can reduce power consumption by more than half during external memory accesses.

Figure 8-15 shows a WAIT instruction being fetched, decoded, and executed. It is fetched as n3 in this example and, during decode, is recognized as a WAIT instruction. The following instruction (n4) is aborted, and the internal clock is disabled from all internal circuitry except the internal peripherals. The processor stays in this state until an interrupt or reset is recognized. The response time is variable due to the timing of the interrupt with respect to the internal clock. Figure 8-15 shows the result of a fast interrupt bringing the pro-

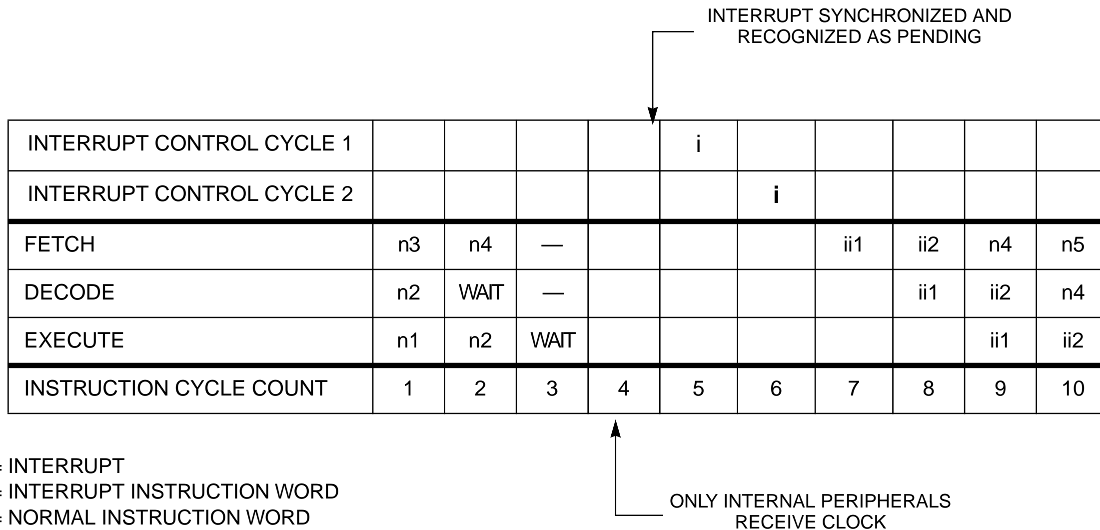
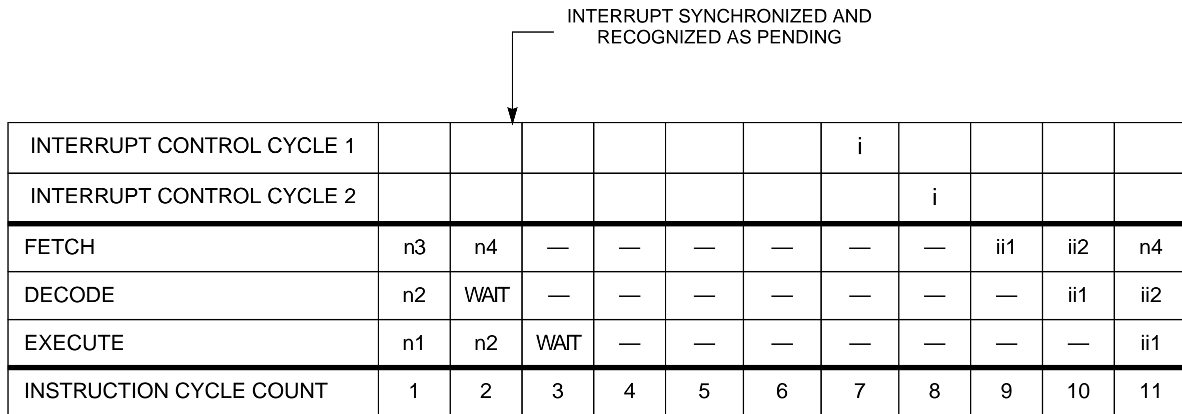


Figure 8-15 Wait Instruction Timing

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

cessor out of the wait state. The two appropriate interrupt vectors are fetched and put in the instruction pipe. The next instruction fetched is n4, which had been aborted earlier. Instruction execution proceeds normally from this point.

Figure 8-16 shows an example of the WAIT instruction being executed at the same time that an interrupt is pending. Instruction n4 is aborted as before. There is a five-instruction-cycle delay caused by the WAIT instruction; then the interrupt is processed normally.



INTERRUPT CONTROL CYCLE 1							i				
INTERRUPT CONTROL CYCLE 2								i			
FETCH	n3	n4	—	—	—	—	—	—	ii1	ii2	n4
DECODE	n2	WAIT	—	—	—	—	—	—	—	ii1	ii2
EXECUTE	n1	n2	WAIT	—	—	—	—	—	—	—	ii1
INSTRUCTION CYCLE COUNT	1	2	3	4	5	6	7	8	9	10	11

i = INTERRUPT
ii = INTERRUPT INSTRUCTION WORD
n = NORMAL INSTRUCTION WORD

EQUIVALENT TO EIGHT NOPs

Figure 8-16 Simultaneous Wait Instruction and Interrupt

The internal clocks are not turned off, and the net effect is that of executing eight NOP instructions between the execution of n2 and ii1.

During the wait state, the BR/BG circuits remain active. Before BR is asserted (see Table 8-12), all port A signals are driven. While the port is inactive, the control signals are deasserted, the data signals are inputs, and the address signals remain as the last address

Table 8-12 BR/BG During WAIT

Signal	Before BR Asserted	While BG Asserted	After BR Deasserted	After Return to Normal State	After First External Access
PS	Driven	Three-state	Three-state	Driven	Driven
DS	Driven	Three-state	Three-state	Driven	Driven
X/Y	Driven	Three-state	Three-state	Driven	Driven
RD	Driven	Three-state	Driven	Driven	Driven
WR	Driven	Three-state	Driven	Driven	Driven
Data	Driven	Three-state	Three-state	Three-state	Driven
Address	Driven	Three-state	Three-state	Three-state	Driven

read or written. The signal timing during a read or write is given in the timing diagrams in the DSP56001 Advance Information Data Sheet (ADI1290). When BG is asserted, all signals are three-stated (high impedance). Immediately after BR is deasserted, the RD and WR signals are driven and are deasserted; all other signals remain in the high-impedance state. During the first T0 clock state following the exit from the wait state, control signals PS, DS, and X/Y are again driven; the data and address signals remain in the high-impedance state. During the first external access, all signals return to their normal operating mode.

8.5 STOP PROCESSING STATE

The stop processing state, which is the lowest power-consumption state, is entered by the execution of the STOP instruction. In the stop state, the clock oscillator is gated off; whereas, in the wait mode, the clock oscillator remains active. The chip clears all peripheral interrupts (HI, SSI, and SCI) and external interrupts (IRQA, IRQB, and NMI) when entering the stop state. Trace or stack errors that were pending, remain pending. The priority levels of the peripherals remain as they were before the STOP instruction was executed. The SCI, SSI, and HI are held in their respective individual reset states while in the stop state.

All activity in the processor is halted until one of the following actions occurs:

1. A low level is applied to the IRQA pin.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

2. A low level is applied to the $\overline{\text{RESET}}$ pin.

Either of these actions will gate on the oscillator, and, after a clock stabilization delay, clocks to the processor and peripherals will be re-enabled. The clock stabilization delay period is determined by the stop delay (SD) bit in the OMR.

The stop sequence is composed of eight instruction cycles called stop cycles. These are differentiated from normal instruction cycles because the fourth cycle is stretched an indeterminant period of time while the four-phase clock is turned off.

The STOP instruction is fetched in stop cycle 1 of Figure 8-17, decoded in stop cycle 2 (which is where it is first recognized as a stop command), and executed in stop cycle 3. The next instruction (n4) is fetched during stop cycle 2 but is not decoded in stop cycle 3

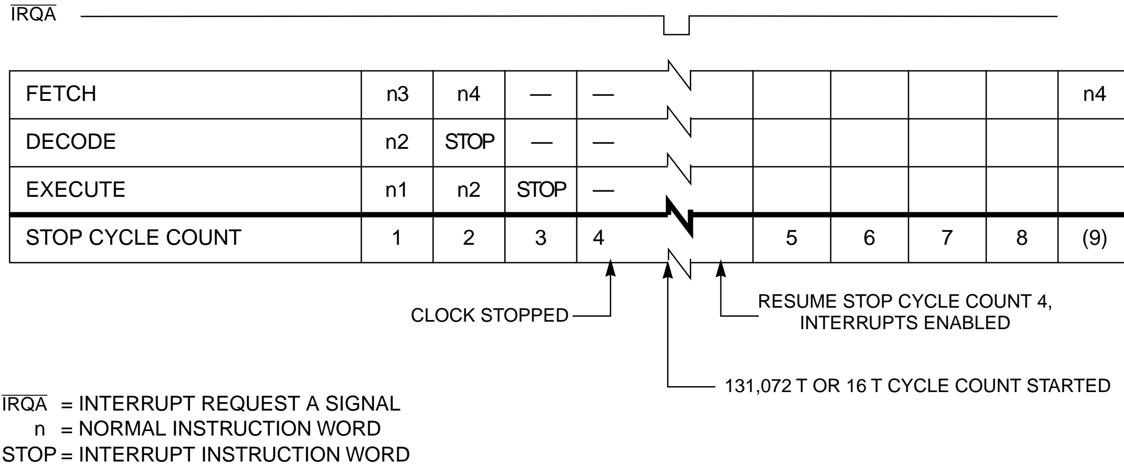


Figure 8-17 Simultaneous Wait Instruction and Interrupt

because, by that time, the STOP instruction prevents the decode. The processor stops the clock and enters the stop mode. The processor will stay in the stop mode until it is restarted.

Figure 8-18 illustrates restarting the system by asserting the IRQA signal. If the exit from stop state was caused by a low level on the $\overline{\text{IRQA}}$ pin, then the processor will service the highest priority pending interrupt. If no interrupt is pending, then the processor resumes

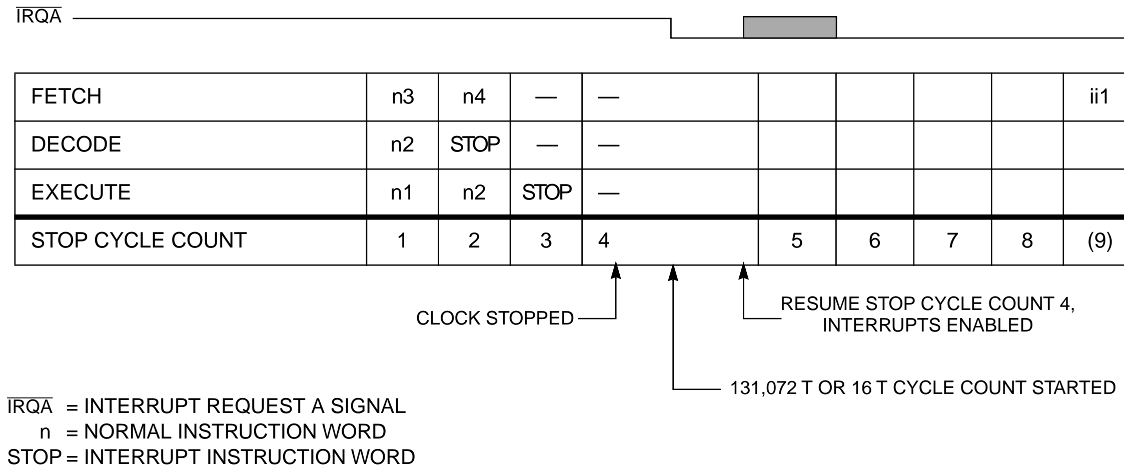


Figure 8-18 STOP Instruction Sequence

at the instruction following the STOP instruction that caused the entry into the stop state.

An $\overline{\text{IRQA}}$ deasserted before the end of the stop cycle count will not be recognized as pending. If $\overline{\text{IRQA}}$ is asserted when the stop cycle count completes, then an $\overline{\text{IRQA}}$ interrupt will be recognized as pending and will be arbitrated with any other interrupts.

Specifically, when $\overline{\text{IRQA}}$ is asserted, the internal clock generator is started and begins a delay determined by the SD bit of the OMR. If the internal clock oscillator is used, the SD bit should be set to zero, which enables a delay count of 128K T cycles (131,072 T cycles) to allow the clock oscillator to stabilize. If a stable external clock is used, the SD bit may be set to one, which enables a 16 T cycle delay.

The following description assumes that SD=0 (the 128K T counter is used). During the 128K T count, interrupts are ignored until the last few count cycles. At this time, the interrupts are synchronized. At the end of the 128K T cycle delay period, the chip restarts instruction processing, stop cycle 4 is completed (interrupt arbitration occurs at this time), and stop cycles 5, 6, 7, and 8 are executed (it takes 17T from the end of the 128K T delay to the first instruction fetch). If the $\overline{\text{IRQA}}$ signal is released (pulled high) after a minimum of 4T but less than 128K T cycles, no $\overline{\text{IRQA}}$ interrupt will occur, and the instruction fetched after stop cycle 8 will be the next sequential instruction (n4 in Figure 8-18). An $\overline{\text{IRQA}}$ interrupt will be serviced (as shown in Figure 8-18) if 1) the $\overline{\text{IRQA}}$ signal had previously

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

been initialized as level sensitive, 2) $\overline{\text{IRQA}}$ is held low from the end of the 128K T cycle delay counter to the end of stop cycle count 8, and 3) no interrupt with a higher interrupt level is pending. If $\overline{\text{IRQA}}$ is not asserted during the last part of the STOP instruction sequence (6, 7, and 8) and if no interrupts are pending, the processor will refetch the next sequential instruction (n4). Since the $\overline{\text{IRQA}}$ signal is asserted (see Figure 8-18), the processor will recognize the interrupt and fetch and execute the instructions at P:$0008 and P:$0009 (the $\overline{\text{IRQA}}$ interrupt vector locations).

To ensure servicing $\overline{\text{IRQA}}$ immediately after leaving the stop state, the following steps must be taken before the execution of the STOP instruction:

1. Define $\overline{\text{IRQA}}$ as level sensitive.
2. Define $\overline{\text{IRQA}}$ priority as higher than the other sources and higher than the program priority.
3. Ensure that no stack error or trace interrupts are pending.
4. Execute the STOP instruction and enter the stop state.
5. Recover from the stop state by asserting the $\overline{\text{IRQA}}$ pin and holding it asserted for the whole clock recovery time. If it is low, the IRQA vector will be fetched. Also, the user must ensure that NMI will not be asserted during these last three cycles; otherwise, NMI will be serviced before $\overline{\text{IRQA}}$ because NMI priority is higher.
6. The exact elapsed time for clock recovery is unpredictable. The external device that asserts $\overline{\text{IRQA}}$ must wait for some positive feedback, such as specific memory access or a change in some predetermined I/O pin, before deasserting $\overline{\text{IRQA}}$.

The STOP sequence totals 131,104 T cycles (if SD=0) or 48 T cycles (if SD=1) in addition to the period with no clocks from the stop fetch to the $\overline{\text{IRQA}}$ vector fetch (or next instruction). However, there is an additional delay if the internal oscillator is used. An indeterminant period of time is needed for the oscillator to begin oscillating and then stabilize its amplitude. The processor will still count 131,072 T cycles (or 16 T cycles), but the period of the first oscillator cycles will be irregular; thus, an additional period of 19,000 T cycles should be allowed for oscillator irregularity (the specification recommends a total minimum period of 150,000 T cycles for oscillator stabilization). If an external oscillator is used that is already stabilized, no additional time is needed.

If the STOP instruction is executed when the $\overline{\text{IRQA}}$ signal is asserted, the clock generator will not be stopped, but the four-phase clock will be disabled for the duration of the 128K T cycle (or 16 T cycle) delay count. In this case, the STOP looks like a 131,072 + 35 T cycle (or 51 T cycle) NOP, since the STOP instruction itself is eight instruction cycles long (32 T) and synchronization of $\overline{\text{IRQA}}$ is 3T which equals 35T.

A trace or stack error interrupt pending before entering the stop state is not cleared and will remain pending. During the clock stabilization delay, all peripheral and external interrupts are cleared and ignored (includes all SCI, SSI, HI, $\overline{\text{IRQA}}$, $\overline{\text{IRQB}}$, and NMI interrupts, but not trace or stack error). If the SCI, SSI, or HI have interrupts enabled in 1) their respective control registers and 2) in the interrupt priority register, then interrupts like SCI transmitter empty will be immedi-

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

ately pending after the clock recovery delay and will be serviced before continuing with the next instruction. If peripheral interrupts must be disabled, the user should disable them with either the control registers or the interrupt priority register before the STOP instruction is executed.

If $\overline{\text{RESET}}$ is used to restart the processor (see Figure 8-19), the 128K T cycle delay counter

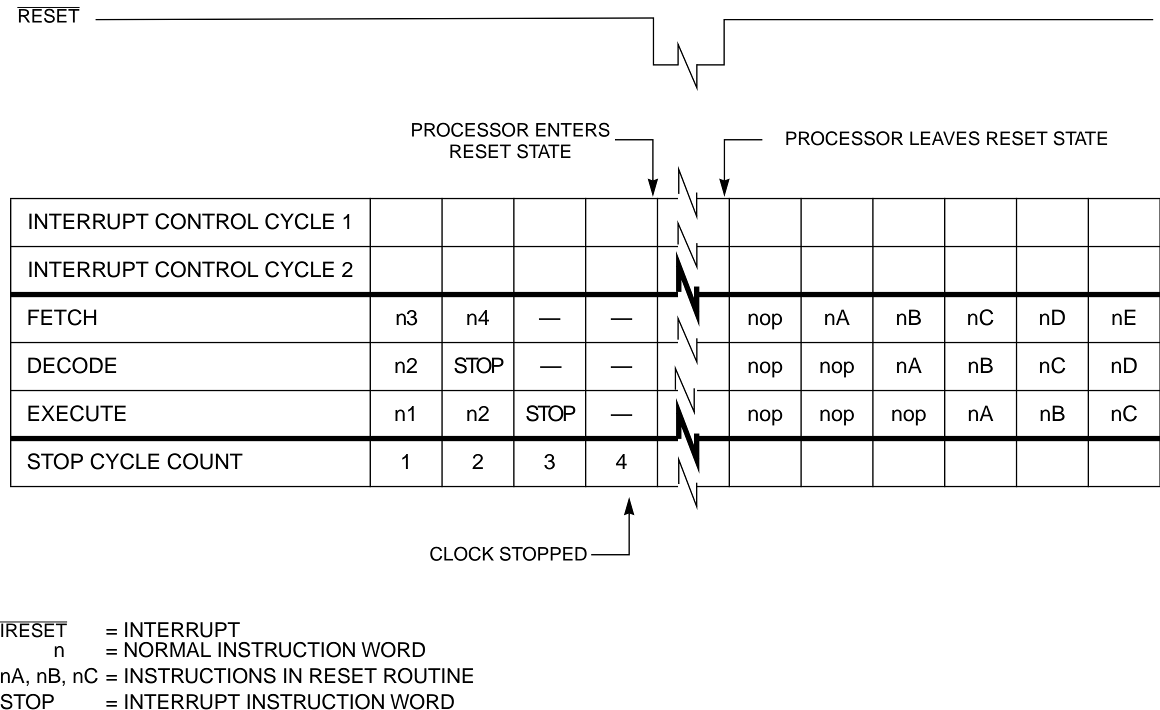


Figure 8-19 STOP Instruction Sequence Recovering with $\overline{\text{RESET}}$

would not be used, all pending interrupts would be discarded, and the processor would immediately enter the reset processing state as described in **8.3 RESET PROCESSING STATE**. The recommended stabilization time suggested in the data sheet for the clock ($\overline{\text{RESET}}$ should be asserted for this time) is only 50 T for a stabilized external clock but is the same 150,000 T for the internal oscillator. These stabilization times are recommended times but are not imposed by internal timers or time delays. The DSP fetches instructions immediately after exiting reset. If the user wishes to use the 128K T (or 16 T) delay counter, it can be started by asserting $\overline{\text{IRQA}}$ for a short time (about two clock cycles).

During the stop mode, the port A bus is frozen. The state of each pin immediately before executing the STOP instruction will be held until the DSP leaves the stop state. Port A is not three-stated, and the $\overline{\text{BR}}$/$\overline{\text{BG}}$ circuits are not operational. However, port A will remain three-stated if $\overline{\text{BG}}$ was asserted before the STOP instruction was executed. One way to release the port A bus for use while the DSP is in the stop state is to use a port B or port C pin to initiate a bus request before executing the STOP instruction.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com