CpE 390 Microprocessor Systems

Lecture 6

Interrupts, Resets, and Operation Modes

Fundamental Concepts of Interrupts

What is an interrupt?

A special event that requires the CPU to stop normal program execution and perform some service related to the event. Examples of interrupts include I/O completion, timer time-out, illegal opcodes, arithmetic overflow, divide-by-0, and so on.

Functions of Interrupts

- Coordinating I/O activities and preventing CPU from being tied up
- Providing a graceful way to exit from errors
- Reminding the CPU to perform routine tasks (ex: timer, multi-task OS, etc.)

Interrupt Maskability

- Interrupts that can be ignored by the CPU are called *maskable interrupts*. A maskable interrupt must be *enabled* before it can interrupt the CPU. An interrupt is enabled by setting an enable flag.
- Interrupts that can't be ignored by the CPU are called *nonmaskable interrupts.*
- Two-level interrupt enabling capability: global interrupt mask and local interrupt mask

Interrupt priority

The order in which the CPU will service interrupts when all of them occur at the same time.

Interrupt Service

The CPU provides service to an interrupt by executing a program called the *interrupt service routine*.

A complete interrupt service cycle includes:

1. Saving the program counter value in the stack
2. Saving the CPU status (including the CPU status register and some other registers) in the stack
3. Identifying the cause of interrupt
4. Resolving the starting address of the corresponding interrupt service routine
5. Executing the interrupt service routine
6. Restoring the CPU status and the program counter from the stack
7. Restarting the interrupted program

When does CPU start to service an interrupt?

• For hardware-maskable interrupts, CPU waits till it completes the current instruction

• For nonmarkable interrupts, CPU may start the service without completing the current instruction

• For some SW interrupts which are caused by an error in an instruction execution, CPU starts the service without completing the current instruction.

Interrupt Vector

Starting address of the interrupt service routine

Interrupt Vector Table

A table where all interrupt vectors are stored.

Methods of Determining Interrupt Vectors

1. Predefined locations for starting address of service routines (8051 approach)
2. Fetching the vector from a predefined memory location for IVT (68HC12)
3. Executing an interrupt acknowledge cycle to fetch a *vector number* in order to locate the interrupt vector (68000 and x86 families)

Steps of Interrupt Programming

Step 1. Initializing the interrupt vector table

```
        ORG  IVT row for service 1
        FDB   service_1      ; starting address of interrupt service routine 1
        ORG  IVT row for service 2
        FDB   service_2
```

Step 2. Writing the interrupt service routine
Step 3. Enabling the interrupt

The Overhead of Interrupts

- Saving and restoring of CPU status and other registers. (68HC12 needs to save all CPU registers).
- The execution of the RTI instruction that will restore all the CPU registers.
- Execution time of instructions of the interrupt service routine.

Resets

- The initial values of some CPU registers, flip-flops, and the control registers in I/O interface chips must be established in order for the computer to function properly.
- The reset mechanism establishes these initial conditions for the computer system.
- There are at least two types of resets: *power-on reset* and *manual reset*.
- The power-on reset establishes the initial values of registers and I/O control registers.
- The manual reset without power-down allows the computer to get out of most error conditions if hardware does not fail.
- A reset is nonmaskable.

68HC12 Exceptions

- Maskable interrupts: including IRQ pin and all peripheral function interrupts.
- Nonmaskable interrupts: including XIRQ pin, SWI interrupt, and unimplemented opcode trap.
- Resets: including the power-on reset, reset pin manual reset, the COP reset (computer operate properly), and clock monitor reset.

Maskable Interrupts

- Different number of maskable interrupts.
- One of the maskable interrupts can be raised to the highest priority among the maskable interrupt group and receive quicker service. This is achieved by writing the low byte of the vector of the interrupt to the HPRIO register.

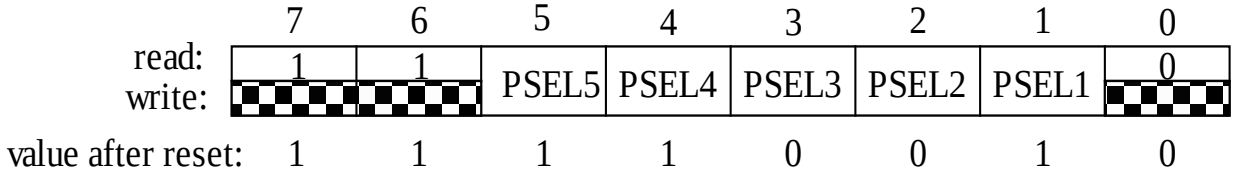

Figure 6.1 Highest priority I interrupt register

Table 6.1 Interrupt vector map

Vector address	Interrupt source	CCR mask	Local Enable	HPRIO value to elevate to highest I bit
$FFFE	Reset	none	none	-
$FFFC	Clock monitor reset	none	COPCTL(CME,FCME)	-
$FFFA	COP failure reset	none	COP rate selected	-
$FFF8	Unimplemented instruction trap	none	none	-
$FFF6	SWI	none	none	-
$FFF4	$\overline{\text{XIRQ}}$	X bit	none	-
$FFF2	$\overline{\text{IRQ}}$	I bit	INTCR(IRQEN)	$F2
$FFF0	Real time interrupt	I bit	RTICTL(RTIE)	$F0
$FFEE	Timer channel 0	I bit	TMSK1(C0I)	$EE
$FFEC	Timer channel 1	I bit	TMSK1(C1I)	$EC
$FFEA	Timer channel 2	I bit	TMSK1(C2I)	$EA
$FFE8	Timer channel 3	I bit	TMSK1(C3I)	$E8
$FFE6	Timer channel 4	I bit	TMSK1(C4I)	$E6
$FFE4	Timer channel 5	I bit	TMSK1(C5I)	$E4
$FFE2	Timer channel 6	I bit	TMSK1(C6I)	$E2
$FFE0	Timer channel 7	I bit	TMSK1(C7I)	$E0
$FFDE	Timer overflow	I bit	TMSK2(TOI)	$DE
$FFDC	Pulse accumulator overflow	I bit	PACTL(PAOVI)	$DC
$FFDA	Pulse accumulator input edge	I bit	PACTL(PAI)	$DA
$FFD8	SPI serial transfer complete	I bit	SP0CR1(SPIE)	$D8
$FFD6	SCI0	I bit	SC0CR2(TIE,TCIE,RIE,ILIE)	$D6
$FFD4	SCI1	I bit	SC1CR2(TIE,TCIE,RIE,ILIE)	$D4 (1,3,4)
$FFD2	ATD0 or ATD1	I bit	ATDxCTL2(ASCIE)	$D2
$FFD0	MSCAN 0 wakeup	I bit	C0RIER(WUPIE)	$D0 (1*,2,2*)
$FFCE	Key wakeup J or H	I bit	KWIEJ[7:0] and KWIEH[7:0]	$CE (1,3,4)
$FFCC	Modulus down counter underflow	I bit	MCCTL(MCZI)	$CC
$FFCA	Pulse accumulator B overflow	I bit	PBCTL(PBOVI)	$CA
$FFC8	MSCAN 0 errors	I bit	C0RIER(RWRNIE,TWRNIE, RERRIE,TERRIE,BOFFIE,OVRIE)	$C8 (2*,3,4)
$FFC6	MSCAN 0 receive	I bit	C0RIER(RXFIE)	$C6 (2*,3,4)
$FFC4	MSCAN 0 transmit	I bit	C0TCR(TXEIE[2:0])	$C4 (2*,3,4)
$FFC2	CGK lock and limp home	I bit	PLLCR(LOCKIE, LHIE)	$C2 (3,4)
$FFC0	IIC Bus	I bit	IBCR(IBIE)	$C0 (4)
$FFBE	MSCAN 1 wakeup	I bit	C1RIER(WUPIE)	$BE (4)
$FFBC	MSCAN 1 errors	I bit	C1RIER(RWRNIE,TWRNIE, RERRIE,TERRIE,BOFFIE,OVRIE)	$BC (4)
$FFBA	MSCAN 1 receive	I bit	C1RIER(RXFIE)	$BA (4)
$FFB8	MSCAN 1 transmit	I bit	C1TCR(TXEIE[2:0])	$B8 (4)
$FFB6	Reserved	I bit		$B6
$FF80-$FFB5	Reserved	I bit		$80-$B4

Note. 1. Available in 812 A4
2. Used as BDLC interrupt vector for 912B32, 912BC32
3. Available in D60
4. Available in DG128 (DT128)

1*. Used as wake up key J in 812A4
2*. Available in 912BC32

- **IRQ Pin Interrupt**
- The only external maskable interrupt for the 68HC12.
- IRQ interrupt can be edge-triggered or level-triggered.
- IRQ interrupt has a local enable mask in the interrupt control (INTCR) register.
- The IRQ interrupt may be delayed for 4096 E clock cycles after exiting the stop mode.
- The IRQ interrupt is configured by programming the INTCR register.
- The contents of the INTCR register are shown in Figure 6.2.

address: $1E	7	6	5	4	3	2	1	0
read: write:	IRQE	IRQEN	DLY	0	0	0	0	0
value after reset:	0	1	1	0	0	0	0	0

IRQE -- $\overline{\text{IRQ}}$ edge sensitive only bit

IRQE can be written once in normal mode. In special modes , it can be written any time, but the first write is ignored.

1 = $\overline{\text{IRQ}}$ pin responds only to falling edge

0 = $\overline{\text{IRQ}}$ pin responds to low level.

IRQEN -- IRQ enable bit

IRQEN bit can be written any time in all modes. The $\overline{\text{IRQ}}$ pin has an internal pullup.

1 = $\overline{\text{IRQ}}$ pin interrupt enabled

0 = $\overline{\text{IRQ}}$ pin interrupt disabled

DLY -- Oscillator startup delay on exit from stop mode

DLYcan be written once in normal modes. In special modes, DLY can be written anytime.

1 = Stabilization delay on exit from stop mode

0 = No stabilization delay on exit from stop mode

Figure 6.2 Interrupt control register (INTCR)

Making IRQ Level-Sensitive

Pros:

Multiple interrupt sources can be tied to this pin.

Cons:

Need to make sure that the IRQ signal has become inactive before the IRQ service routine is complete if there is only one interrupt request pending.

Making IRQ Edge-Sensitive

Pros:

No need to control the duration of the IRQ pulse.

Cons:

Not suitable for noisy environment because every falling edge caused by noise will be recognized as an interrupt.

When does the MCU recognize interrupt requests?

The MCU recognizes the interrupt request when it completes the execution of the current instruction unless the current instruction is a fuzzy logic instruction. For fuzzy logic instructions, the 68HC12 recognizes the interrupt immediately.

The Stack Order on Entry of an Interrupt

- The 68HC12 saves all CPU registers on an interrupt.
- The order of saving CPU registers is shown in Figure 6.3.

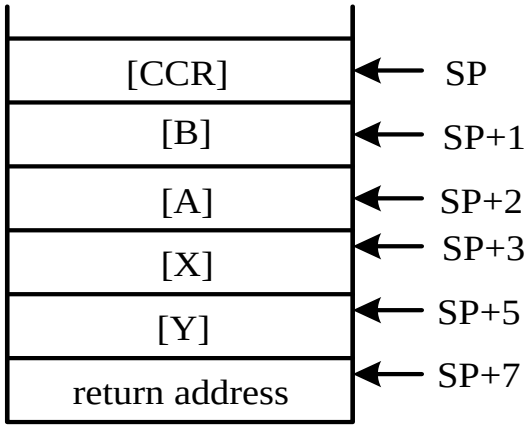

Figure 6.3 Stack order on entry to interrupts

The RTI Instruction

- RTI is used to terminate interrupt service routines.
- RTI will restore CPU registers from the stack.
- The 68HC12 will continue to execute the interrupted program unless there is another pending interrupt.

Nonmaskable Interrupts

- There are three nonmaskable interrupts: XIRQ pin, SWI instruction, and unimplemented instruction opcode trap.

XIRQ Pin Interrupt

- XIRQ interrupt is disabled during a system reset and upon entering the service routine of another XIRQ interrupt.
- After minimal system initialization, software can clear the X bit of the CCR register to enable the (using the **andcc #$BF** instruction) XIRQ interrupt. Software cannot reset the X bit once it has been set.
- When a nonmaskable interrupt is recognized, both the X and I bits are set after CPU registers are saved.
- The execution of an RTI instruction at the end of the XIRQ service routine will restore the X and I bits to the pre-interrupt request state.

Unimplemented Opcode Trap

- There are 202 unimplemented opcode on page 2 (16-bit opcode).
- These unimplemented opcode share the same vector $FFF8:$FFF9.

Software Interrupt Instruction (SWI)

- Execution of the SWI instruction causes an interrupt without an interrupt request signal.
- The SWI instruction is commonly used in the debug monitor to implement *breakpoints* and to transfer control from a user program to the debug monitor.
- A **breakpoint** in a user program is a memory location where we want program execution to be stopped and information about instruction execution (in the form of register contents) to be displayed.

Resets

There are four possible sources of resets:

- Power-on reset (POR)
- External reset (RESET pin)
- COP (computer operate properly) reset
- Clock monitor reset

Power-On Reset

- The 68HC12 has circuitry to detect a positive transition in the V_{DD} supply and initialize the microcontroller by asserting the reset signal internally.
- The reset signal is released after a delay that allows the device clock generator to stabilize.

External Reset (RESET pin)

- The 68HC12 has circuit to distinguish internal and external resets.
- The on-chip EEPROM may be corrupted if the power supply drops below the required level. When the power supply drops below the required level, the RESET signal should be pulled low to prevent instruction execution.
- A low-voltage inhibit circuit such as the Motorola MC34064 can be used to protect against the EEPROM corruption.

COP Reset

- The computer operate properly (COP) system is designed to protect against software failure.
- If software was written correctly, it should follow certain sequence of execution and the execution time can also be predicted.
- When the COP is enabled, software must write $55 and $AA (in this order) to the COPRST register to keep a watchdog timer from timing out.
- If our software was not written properly, then it may not write $55 and $AA to the COPRST (located at $17) before the COP times out and the CPU will be reset. The software problems can therefore be detected.

Clock Monitor Reset

The clock monitor reset circuit uses an internal RC circuit to determine whether the clock frequency is above a predetermined limit.

If no EXTALi clock edges are detected within this time delay, the clock monitor can optionally generate a system reset.

Low Power Modes

- Power consumption is unavoidable in normal operation for an embedded system.
- The microcontroller may not always perform useful operations. It would be ideal to reduce the power consumption to minimum whenever the microcontroller is idle.
- The 68HC12 has two low power modes: **wait** and **stop** modes.

The Wait Instruction

- The wait instruction pushes all CPU registers (except the stack pointer) and the return address into the stack and enters a wait state.
- During the wait state, CPU clocks are stopped (clock signals that drive the ALU and register file), but other clocks in the microcontroller (clock signals that drive peripheral functions) continue to run.

The 68HC12 will leave the wait state when one of the following events occurs:

- maskable interrupts that are not masked
- nonmaskable interrupts
- resets

Upon leaving the wait state, the CPU sets the appropriate interrupt mask bits and fetches the vector corresponding the exception sensed, and instruction execution continues at the location the vector points to.

The STOP Instruction

- When the S bit in the CCR register is cleared and a **stop** instruction is executed, the 68HC12 saves all CPU registers (except the stack pointer) in the stack, stops all system clocks, and puts the microcontroller in standby mode.
- Standby operation minimizes system power consumption. The contents of registers and the states of I/O pins remain unchanged
- Standby mode ends by asserting RESET, XIRQ, and IRQ signals.

Exception Programming for the 68HC12

Initializing the Interrupt Vector Table

Setup by using assembler directives

```
        org     $FF80
        …
        fdb     pbov_isr        ; pulse accumulator B overflow interrupt vector
        …
        fdb     atd_isr         ; A/D conversion complete interrupt service routine
        …
        fdb     pai_isr         ; PAI interrupt service routine
        …
        fdb     c7_isr          ; output compare channel 7 interrupt service routine
        fdb     c6_isr          ; input capture channel 6 interrupt service routine
        …
```

Table 6.4 D-Bug12 Monitor Exception Vector Numbers

Vector number	D-Bug12 Version 1.xxx	D-Bug12 Version 2.xxx
7	Port H key Wakeup	Port H key Wakeup
8	Port J key Wakeup	Port J key Wakeup
9	Analog-to-Digital converter	Analog-to-Digital converter
10	Serial Communication Interface 1	Serial Communication Interface 1
11	Serial Communication Interface 0	Serial Communication Interface 0
12	Serial Peripheral Interface 0	Serial Peripheral Interface 0
13	Timer Channel 0	Pulse Accumulator Edge
14	Timer Channel 1	Pulse Accumulator Overflow
15	Timer Channel 2	Timer Overflow
16	Timer Channel 3	Timer Channel 7
17	Timer Channel 4	Timer Channel 6
18	Timer Channel 5	Timer Channel 5
19	Timer Channel 6	Timer Channel 4
20	Timer Channel 7	Timer Channel 3
21	Pulse Accumulator Overflow	Timer Channel 2
22	Pulse Accumulator Edge	Timer Channel 1
23	Timer Overflow	Timer Channel 0
24	Real Time Interrupt	Real Time Interrupt
25	$\overline{\text{IRQ}}$ interrupt	$\overline{\text{IRQ}}$ interrupt
26	$\overline{\text{XIRQ}}$ interrupt	$\overline{\text{XIRQ}}$ interrupt
27	SWI instruction	SWI instruction
28	Unimplemented Instruction Trap	Unimplemented Instruction Trap
-1	Return to the starting address of the RAM vector table	Return to the starting address of the RAM vector table

Examples of Interrupt Service Routine

In assembly, an example for TC7 (output compare 7, to be discussed in chapter 8)

```
oc7_isr   bclr   TFLG1,$7F     ; clear the C7F flag
          dec    oc7_cnt       ; decrement a counter
          rti
```

How to clear a timer flag bit?
In normal mode, write a 1 to the flag bit to be cleared
Method 1. Use the BCLR instruction with a 0 at the bit position (s) corresponding to the flag (s) to be cleared. For example,

```
BCLR  TFLG1, $7F
```

will clear the C7F flag.

Method 2. Use the **movb** instruction with a 1 at the bit position (s) corresponding to the flag (s) to be cleared. For example,

```
movb    #$80,TFLG1
```

will clear the C7F flag.

Enabling/disabling Interrupts

- Need to set/clear local interrupt enable bit
- Need to clear/set the I flag of the CCR register

To enable TC7 interrupt in assembly,

```
bset    TMSK1,$80    ; enable TC7 interrupt locally
cli                  ; enable interrupt globally
```

In C language,

```
TMSK1 |= 0x80;
INTR_ON();           ; same as asm("cli");
```

To disable interrupt **in assembly**,

```
bclr    TMSK1,$80    ; local disable
sei                  ; global disable
```

In C language,

```
TMSK1 &= 0x7F;       /* local disable */
INTR_OFF ();         /* global disable */
```

Example 6.2 Generate external interrupts to the 68HC12 by

1. Using the 555 timer chip to generate a digital waveform with frequency equal to approximately 1 Hz. The circuit connection of the 555 timer is illustrated in Figure 6.6. Connect the 555 timer output (pin 3) to the PT7 pin (pulse accumulator input) of the 68HC912BC32.

2. Configuring the pulse accumulator so that it operates in event-counting mode and interrupts the CPU whenever rising edges arrive at the PAI (PT7) pin.

3. Writing a service routine for the PAI interrupt that increments the interrupt count and outputs the message **interrupt x,** where **x** can be from one to ten.

4. Writing a loop to check the interrupt count. Exit the loop when the interrupt count reaches 10.

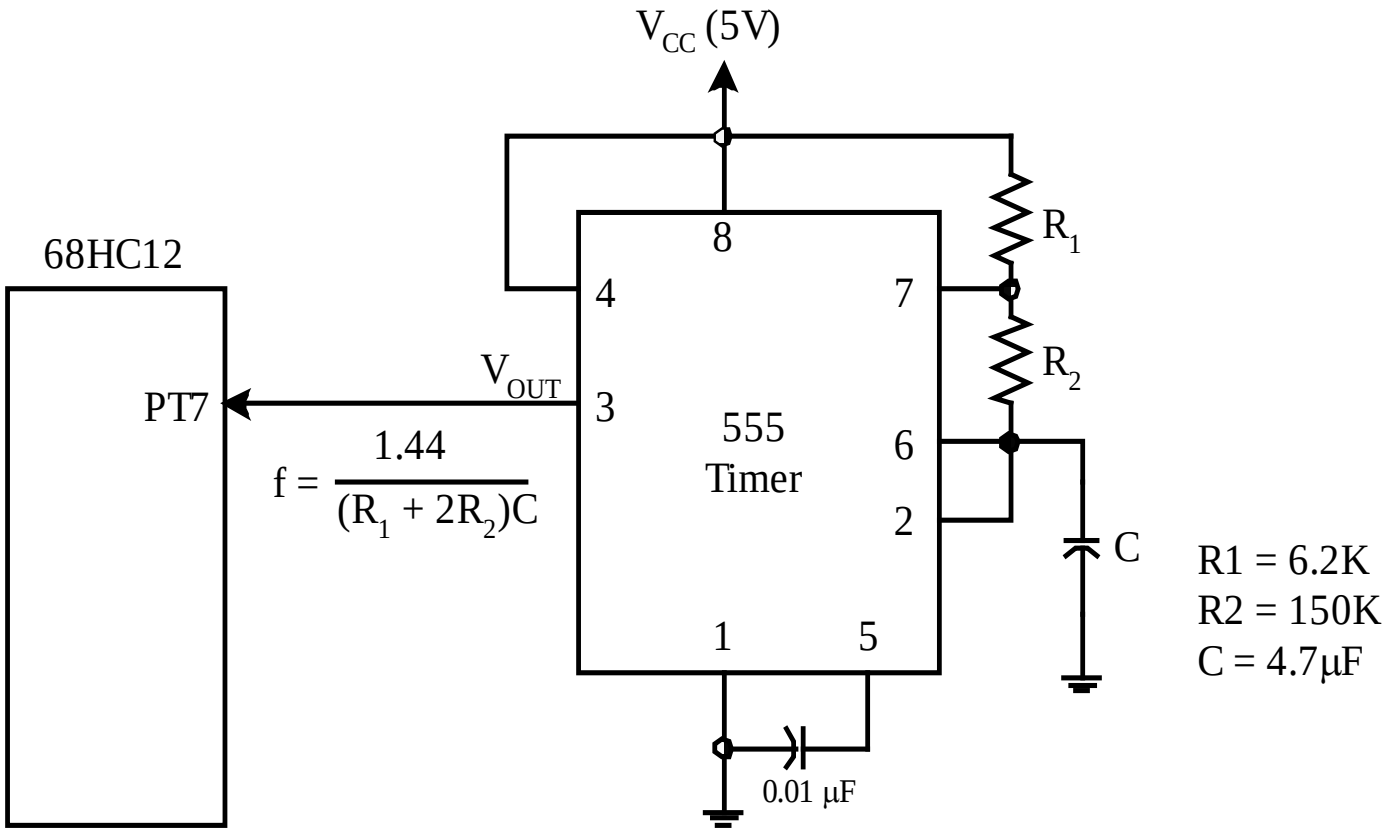

Figure 6.6 68HC12 PAI interrupt circuit

Solution:

The assembly program is as follows:

```
CR              equ     $0D             ; ASCII code of carriage return
LF              equ     $0A             ; ASCII code of line feed
pactl           equ     $A0             ; pulse accumulator control register
paflg           equ     $A1             ; pulse accumulator flag register
printf          equ     $F686           ; memory location to hold the starting address of printf()
setuservector equ       $F69A           ; memory location to hold the starting address of
                                        ; setuservector()
pa_vect_no      equ     13              ; vector number of PAI edge interrupt

                org     $800            ; starting address of internal SRAM
total           rmb     1               ; number of seconds passed
                org     $1000           ; starting address of the program
                lds     #$8000          ; initialize the stack pointer
                sei                     ; disable interrupt
; the next instruction enable PA function, select event counting mode,
; and select rising edge as the active edge
                movb    #$50,pactl      ; initialize pa function
; the following 6 instructions set up PAI interrupt vector
                ldd     #pai_isr
                pshd
```

```
            ldab    #pa_vect_no   ; interrupt vector number of PAI pin
            clra
            ldx     setuservector
            jsr     0,x
            puld                  ; clean up the stack
            ldaa    #01
            staa    paflg         ; clear the PAIF flag
            ldaa    #0            ; initialize the second count to 0
            staa    total         ;           "
; the following two instructions enable PAI interrupt
            bset    pactl,$01     ; enable PAI interrupt
            cli                   ; enable interrupt globally
; the outer loop makes sure 10 second delays are created
wait_loop   ldaa    total
            cmpa    #10
            beq     done
            jmp     wait_loop
done        swi
; ***************************************************************************
; the PAI interrupt service routine cleared the PAIF flag, increment the interrupt output
; count, and the message "interrupt x", where x can be 1, 2, 3, ...
; ***************************************************************************
pai_isr     ldaa    #01           ; clear PAIF flag
```

```
            staa    paflg               ;           "
            inc     total               ; one second delay has been created
            clra                        ; print out the message
            ldab    total               ;           "
            pshd                        ;           "
            ldd     #time_msg           ;           "
            ldx     printf              ;           "
            jsr     0,x                 ;           "
            leas    2,sp                ; clean up the stack
            rti
time_msg    db      " interrupt %d ",CR,LF,0
            end
```

In summary, three conditions must be met simultaneously for an interrupt service to occur:

- The device is armed, which means the signal is connected to an input of the microcomputer that can generate an interrupt (such as SPI, SCI, RTI)
- The interrupt is enabled (both locally and globally)
- An interrupt event occurs that sets the flag in one of the I/O status registers of the microcomputer

Operation Modes

- The 68HC12 can operate in 8 different modes.
- The states of the BKGD, MODB, and MODA pins when the reset signal is low determine the operating mode after the CPU leaves the reset state. The mode selections are listed in Table 6.5.
- The SMODN, MODB, and MODA bits in the MODE register show the current operation mode and provide limited mode switching during the operation.
- Two basic types of operation modes are:

  1.Normal modes-- Some registers and bits are protected against accidental changes.

  2.Special modes-- Greater access for special purposes such as testing and emulation to protected control registers and bits are allowed .
- The background debug mode (BDM) is a system development and debug feature and is available in all modes. In special single-chip mode, BDM is active immediately after reset.

Table 6.5 68HC12 Mode Selection

BKGD	MODB	MODA	Mode	Port A	Port B
0	0	0	Special single chip	general-purpose I/O	general-purpose I/O
0	0	1	Special expanded narrow	ADDR[15:8]DATA[7:0]	ADDR[7:0]
0	1	0	Special peripheral	ADDR/DATA	ADDR/DATA
0	1	1	Special expanded wide	ADDR/DATA	ADDR/DATA
1	0	0	Normal single chip	General-purpose I/O	General-purpose I/O
1	0	1	Normal expanded narrow	ADDR[15:8]DATA[7:0]	ADDR[7:0]
1	1	0	Reserved (forced to peripheral)	--	--
1	1	1	Normal expanded wide	ADDR/DATA	ADDR/DATA

Homework #4

- See course website: http://390.revan.us
  - click homework tab
- Please submit a hard copy