EE4390 Microprocessors

Lessons 23, 24
- Exceptions -
Resets and Interrupts

- Exceptions -
Resets and Interrupts

- Polling vs. Interrupts
- Exceptions: Resets and Interrupts
- 68HC12 Exceptions
  - Resets
  - Interrupts: Maskable and Non-maskable
- 68HC12 Interrupt Response
- Exception Vector
- Exception Priority
- Programming an Interrupt Service Routine

- Exceptions -
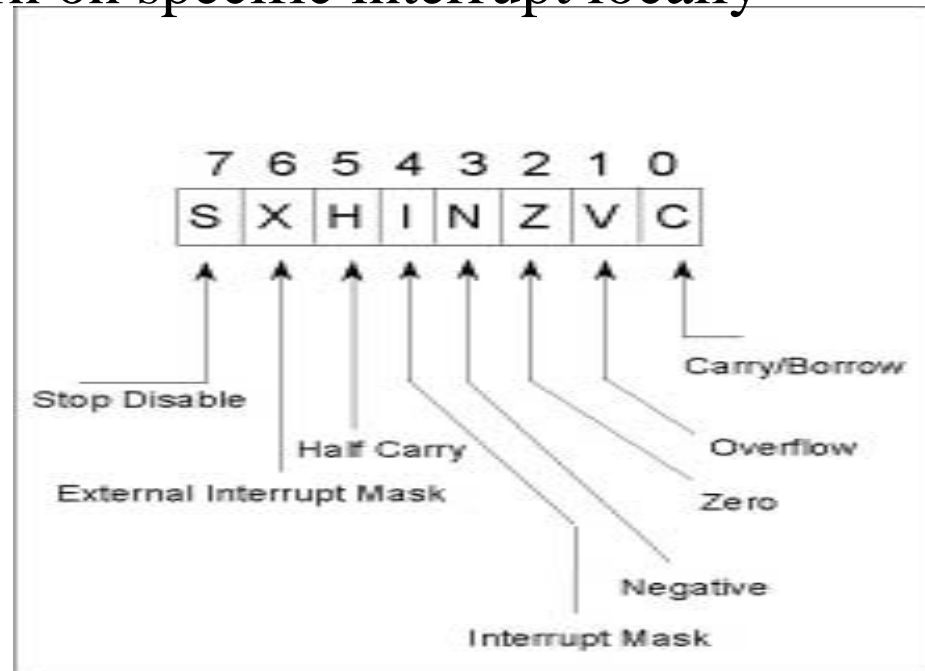# Resets and Interrupts

- Polling versus interrupts
  - polling: constantly monitoring for flag to set
    - program is tied up waiting for flag
    - inefficient use of processor
  - interrupt: processor tells program when event has occurred
    - program can be executing other tasks
    - efficient use of processor
  - EX] sequentially ask question vs you ask me

- Exceptions -
Resets and Interrupts

- Resets: returns 68HC12 to known, well-defined state after detected fault

  – power-on reset

  – Computer Operating Properly (COP) reset

  – Clock Monitor reset

  – External reset

- Interrupts - planned, but unscheduled high priority event

  – non-maskable: may not be turned off by user

  – maskable: turned on and off by user with "I" bit in CCR

- Exceptions -
Resets and Interrupts

- "I" bit controlled with CLI and SEI command
  - CLI: Clear Interrupt Mask - turns interrupt system on
  - SEI: Set Interrupt Mask - turns interrupt system off
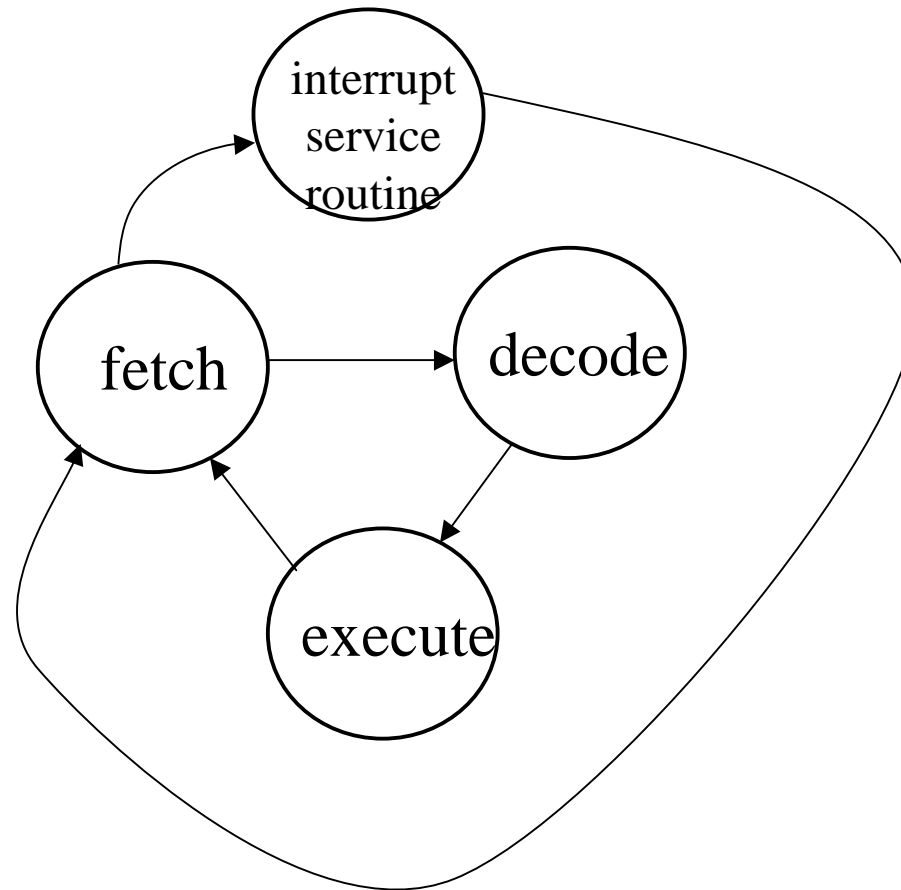- Need to turn on specific interrupt locally
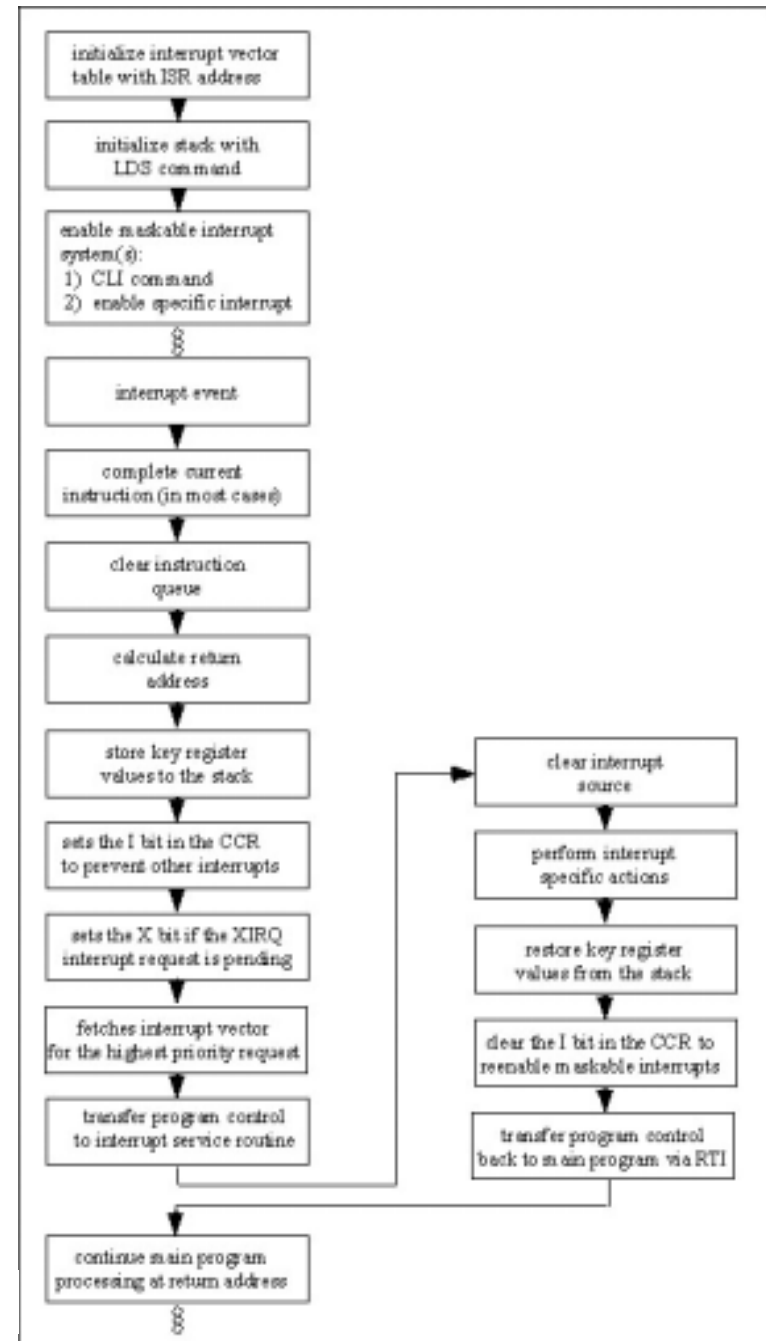
- Exceptions -
Resets and Interrupts (cont)
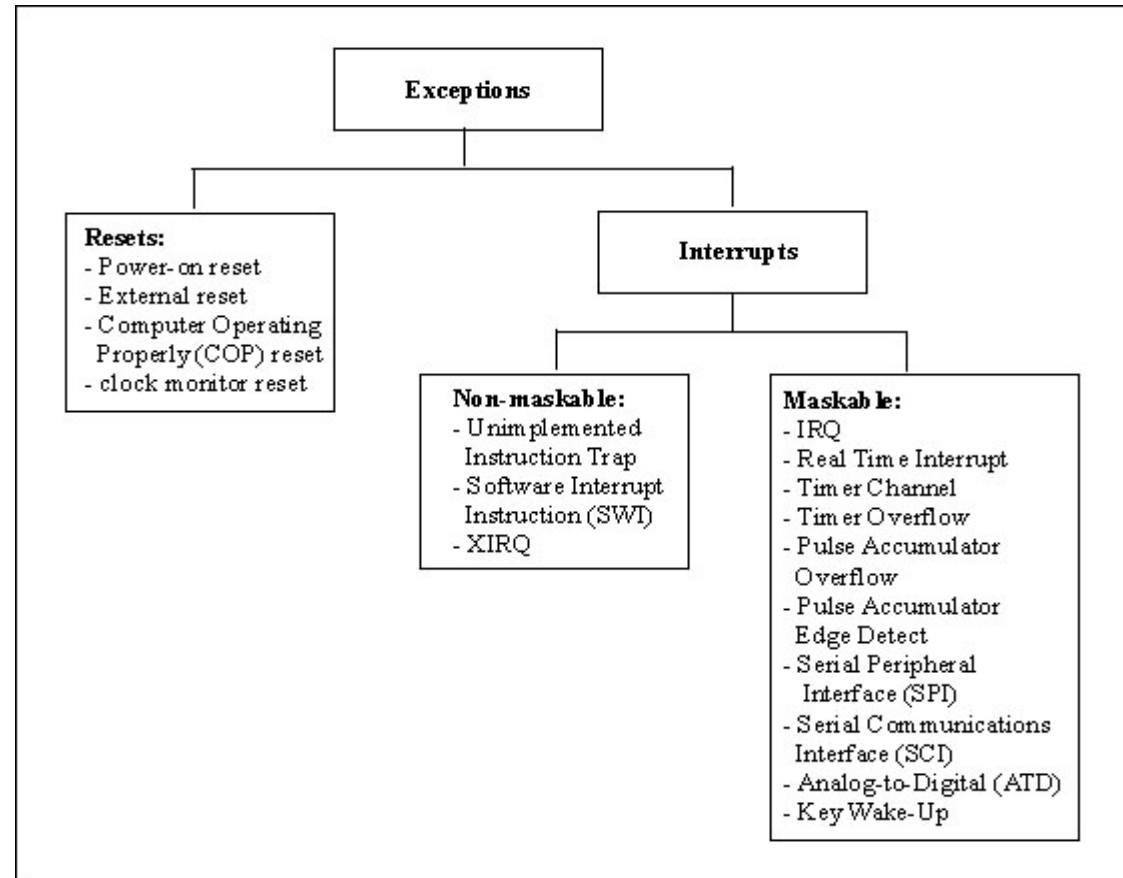
68HC12 Interrupt Response

68HC12 Interrupt Response

- Interrupt Vector
  - location of ISR
  - located in upper 128 bytes
    of memory
  - user must tie Vector to ISR
- Interrupt Priority
  - determines order of
    execution when multiple
    interrupts occur
- Interrupt Service Routine
  (ISR) - user written
  response routine to
  interrupt event

initialize interrupt vector
table with ISR address

initialize stack with
LDS command

enable maskable interrupt
system(s):
1) CLI command
2) enable specific interrupt

interrupt event

complete current
instruction (in most cases)

clear instruction
queue

calculate return
address

store key register
values to the stack

sets the I bit in the CCR
to prevent other interrupts

sets the X bit if the XIRQ
interrupt request is pending

fetches interrupt vector
for the highest priority request

transfer program control
to interrupt service routine

clear interrupt
source

perform interrupt
specific actions

restore key register
values from the stack

clear the I bit in the CCR to
reenable maskable interrupts

transfer program control
back to main program via RTI

continue main program
processing at return address

68HC12 Interrupt Response

- Interrupt Priority -

Vector Address	Interrupt Source	CCR Mask	Local Enable	HPRIO Value to Elevate
$FFFE, $FFFF	Reset	none	none	–
$FFFC, $FFFD	COP Clock Monitor Fail Reset	none	CME, FCME	–
$FFFA, $FFFB	COP Fail Reset	none	cop rate selected	–
$FFF8, $FFF9	Unimplemented Instruction Trap	none	none	–
$FFF6, $FFF7	SWI	none	none	–
$FFF4, $FFF5	XIRQ	x bit	none	–
$FFF2, $FFF3	IRQ or Key Wake Up D	1 bit	IRQEN, KWIED[7:0]	$F2
$FFF0, $FFF1	Real Time Interrupt	1 bit	RTIE	$F0
$FFEE, $FFEF	Timer Channel 0	1 bit	TC0	$EE
$FFEC, $FFED	Timer Channel 1	1 bit	TC1	$EC
$FFEA, $FFEB	Timer Channel 2	1 bit	TC2	$EA
$FFE8, $FFE9	Timer Channel 3	1 bit	TC3	$E8
$FFE6, $FFE7	Timer Channel 4	1 bit	TC4	$E6
$FFE4, $FFE5	Timer Channel 5	1 bit	TC5	$E4
$FFE2, $FFE3	Timer Channel 6	1 bit	TC6	$E2
$FFE0, $FFE1	Timer Channel 7	1 bit	TC7	$E0
$FFDE, $FFDF	Timer Overflow	1 bit	TOI	$DE
$FFDC, $FFDD	Pulse Accumulator Overflow	1 bit	PAOVI	$DC
$FFDA, $FFDB	Pulse Accumulator Input Edge	1 bit	PAII	$DA
$FFD8, $FFD9	SPI Serial Transfer Complete	1 bit	SPI0E	$D8
$FFD6, $FFD7	SCI 0	1 bit	TIE0, TCIE0, RIE0, ILIE0	$D6
$FFD4, $FFD5	SCI 1	1 bit	TIE1, TCIE1, RIE1, ILIE1	$D4
$FFD2, $FFD3	ATD	1 bit	ADIE	$D2
$FFD0, $FFD1	Key Wakeup J (stop wakeup)	1 bit	KWIEJ[7:0]	$D0
$FFCE, $FFCF	Key Wakeup H (stop wakeup)	1 bit	KWIEH[7:0]	$CE
$FF80–$FFCD	Reserved	1 bit	·	$B0–$CC

Revised: Aug 1, 2003

Programming an
Interrupt Service Routine

- Determine how interrupt is enabled
  - global: CLI
  - local enable bit
- Initialize Vector Table
  - directive approach
  - EVB SetUserVector
- Initialize Stack
- Enable interrupt
- Write the specific ISR

Programming an
Interrupt Service Routine - an example

Initialize the microprocessor for the interrupt.

- Initialize the stack - this is done through compiler settings

- Initialize any other necessary systems on the HC12.

- Initialize the interrupt vector table. You will need to use a special header file and code.

  – Header file is on your computer and is called **abbie.h** (change name).

- Code to set up your function to be an interrupt service routine will be similar to the following:

   This part declares your function as an interrupt service routine.
   #pragma interrupt_handler toggle_isr

Programming an
Interrupt Service Routine - an example

- This part fills the appropriate vector with the address of your
interrupt service routine.

```
#pragma abs_address: 0x0B2A
void (*Timer_Channel_2_interrupt_vector[])()={toggle_isr};
#pragma end_abs_address
```

- Make sure the interrupt you will be using is cleared to start.
- Initialize local interrupts.
- Initialize the interrupt system using the CLI command (With this header file
use CLI();). Do this step last so that you aren't inadvertently setting off
interrupts before you finish initializing the system.
- Write the interrupt service routine to handle the interrupt.

Programming an ISR - an example

Table 1. RAM Interrupt Vectors

Interrupt Name	RAM Vector Location
BDLC (Key Wakeup J)	$0B10, $0B11
ATD	$0B12, $0B13
SCI	$0B16, $0B17
SPI	$0B18, $0B19
Pulse Accumulator Input Edge	$0B1A, $0B1B
Pulse Accumulator Overflow	$0B1C, $0B1D
Timer Overflow	$0B1E, $0B1F
Timer Channel 7	$0B20, $0B21
Timer Channel 6	$0B22, $0B23
Timer Channel 5	$0B24, $0B25
Timer Channel 4	$0B26, $0B27
Timer Channel 3	$0B28, $0B29
Timer Channel 2	$0B2A, $0B2B
Timer Channel 1	$0B2C, $0B2D
Timer Channel 0	$0B2E, $0B2F
Real Time Interrupt	$0B30, $0B31
IRQ	$0B32, $0B33
XIRQ	$0B34, $0B35
SWI	$0B36, $0B37
Unimplemented Instruction Trap	$0B38, $0B39
COP Failure	$0B3A, $0B3B
COP Clock Monitor Fail Reset	$0B3C, $0B3D
Reset	$0BEF, $0BFF

Revised: Aug 1, 2003

Programming an
Interrupt Service Routine - an example

- Example] In this task you will need to simultaneously generate two square waves with different frequencies. For one wave use the month and day of your birthday and for the second use the month and day of your Lab TA's birthday. Verify that the waves are being generated simultaneously and that they have different frequencies with the oscilloscope.

Programming an
Interrupt Service Routine - an example

```
#include <abbie.h>
void toggle1_isr(void);                    //function prototype
void toggle2_isr(void);
#pragma interrupt_handler toggle1_isr   //define as interrupt
#pragma interrupt_handler toggle2_isr

#pragma abs_address: 0x0B28
void (*Timer_Channel_3_interrupt_vector[])()={toggle2_isr};
void (*Timer_Channel_2_interrupt_vector[])()={toggle1_isr};
#pragma end_abs_address
```

Programming an
Interrupt Service Routine - an example

```c
void initialize(void);              // Define function initialize


void main(void){
    initialize();                   // Initialize the timer system
    TMSK1 = 0x0C;
    TFLG1 = 0xFF;
    CLI();                          // Initialize interrupts

    while(1)                        // Continuous loop
    {                               // Wait for interrupts
        ;
    }
}
```

Programming an
Interrupt Service Routine - an example

```
/* Function: initialize: enables the timer and sets up the M-
   Clk */

 void initialize(){
   CLKCTL = 0x02;    // Set M-clock to divide by 4 (2 MHz)
                     // CPU master clock divider ($0047)
   TMSK2 = 0x00;     // Disable TOI, Prescale = 0;
   TIOS = 0x0C;      // Make OS2 output compare
   TSCR = 0x80;      // Enable the timer
   TCTL2 =0x50;
 }
```

Programming an
Interrupt Service Routine - an example

```
void toggle1_isr(void){
    TFLG1 = 0x04;
     TC2 += 9091;
 }


 void toggle2_isr(void)
 {
   TFLG1 = 0x08;
   TC3 += 4854;
 }
```