# Lab-6: Timer Subsystem -Flag Polling and Interrupts
# ECE-3731

## HCS12 INTERRUPT SYSTEM

The interrupt system on the HCS12 is used for communicating with external hardware as well as for timing procedures. When an interrupt line is activated, the interrupt system stops the main program, stores it registers on the stack and then services the interrupt (runs the interrupt subroutine). Once an interrupt has been served, it will return back to the main program. You can think of the interrupt system like an alarm clock – when some condition is met (a certain time, for instance), the alarm sounds, and you immediately attend to the alarm clock. Because the interrupt system is automatic and immediate, it results in better timing control and less software work.

When an interrupt occurs, it is serviced with an interrupt service routine. An ISR is a separate piece of code, similar to a subroutine, which executes when an interrupt occurred. The ISR performs any work that needs to be done to service the interrupt, such as reading data from a parallel port. After the ISR is finished, the main program resumes.

Here's what actually happens during an interrupt. First, the HCS12 saves all the registers (A, B, X, Y, the CCR, and the PC) to the stack, so we don't need to worry about disturbing them during the ISR. This makes sense, because we don't know exactly when an interrupt is going to happen. Second, the HCS12 prevents any more interrupts from happening until the ISR is finished. This is good, because we don't want a second interrupt happening until the first one is completed. Third, the ISR executes. Finally, when the ISR is finished, the HCS12 restores the registers from the stack, and resumes the main program. The instruction that signifies the end of the ISR is the RTI instruction. *Every interrupt service routine must end with the RTI instruction.*

How does the HCS12 know where to find the ISR? When an interrupt occurs, the HCS12 looks to the jump vector table for that specific interrupt line. A jump vector is simply a space in memory that contains the address of the ISR.

Naturally, we don't want any random interrupt going off during our program, so we can control which interrupts are active or enabled. The I (interrupt) bit in the CCR serves as a master control switch for the interrupt system. When the I bit is one, all interrupts are disabled. When the I bit is zero, the interrupt system is enabled. By default, the I bit is set to one, so we have to set the I bit to zero when we want to use any interrupt. The CLI instruction clears the I bit to zero. If we want to later disable the interrupt system, the SEI instruction sets the I bit to one. Each separate interrupt line also has its own individual control bit that enables or disables it. In this way, we can control whether any interrupts are enabled, and then selectively enable only those interrupt lines that we need. This control system is similar to a circuit breaker box in a lab. There is usually a master on/off switch that needs to be turned on before anything can get power, and then each light or area has its own individual circuit breaker.

Please see the class notes and textbook for more in-depth information and examples.

## HCS12 TIMER SUBSYSTEM

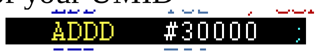
The HCS12 has a standard timer module that is built around a 16-bit timer counter which is clocked by submultiple of the bus clock, using the prescaler.
It provides 8 channels of input capture or output compare. It also has a pulse accumulator and can do pulse width modulation.
In this lab we are mainly concerned with output compare and timer overflow.

Output Compare
-waits for the 16-bit timer (TCNT register) to be equal to a value in a register (TOC time of compare register) -an output compare flag is set when this happens
-optionally generates an interrupt

Timer Overflow
-a flag is set (TOF timer overflow flag) whenever the 16-bit TCNT register overflows.
-optionally generates an interrupt

## ASSIGNMENT

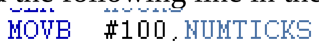
a) Understand the working of code given in canvas under Lab section with name "Lab6.zip", sketch the complete/detailed flowchart. The flowchart should be made in software.
b) In the sample code instead of adding 30,000 in CPU register D, add the last 4 digits of your UMID



```
ADDD    #30000   ;
```

explain the behavior of clock., also provide calculations on paper. Does your clock behavior agree to your calculations?

c) Find the following line in the original sample code,



```
MOVB  #100,NUMTICKS
```

In NUMTICKS move a value equal to the last 2 digits of UMID and explain the behavior of clock, also provide calculations on paper. Does your clock behavior agree to your calculations?

## Post lab Question

Write assembly code to display the clock on the terminal. The value of pre-scalar to be used is 32. You should do this by polling the timer over flow flag.