Interrupts

Interrupts are what make modern computing possible. With all our I/O devices, the OS would have to poll each in succession. Polling is a bad thing as the number of devices increases. Interrupts, on the other hand, allow for asynchronous system events to occur.

ISR

Special piece of code that executes to service an interrupt that has occurred.

Interrupt Vector Table Entries

Normal expanded, normal single-chip, special expanded (special test), special single-chip (special boot)

	Normal Expanded	Normal Single-chip	Special Expanded	Special Single-Chip
RTI (Real Time)	0xFFF0 – 0xFFF1		0xBFF0-0xBFF1	0x00EB – 0x00ED
TOC2	0xFFE6 – 0xFFE7		0xBFE6 – 0xBFE7	

All the interrupt vector assignments are available on page.

In normal mode, the vector table entries are on the FF page of memory. In special test mode, they're on the BF page of memory.

In everything but bootstrap mode, the interrupt vector table entries are two bytes long. The vector table location holds the starting address of the ISR. Because we have 16-bit wide addresses on the HC11, these are two bytes.

When an interrupt occurs, the HC11 reads from the vector table entry and inserts the value at that memory location into the program counter.

Example

Assume our RTIISR begins at 0x8B94.

Up in our main/init/boot, we would do:

```
LDX     #RTIISR
STX     0xFFF0
STX     0xBFF0
```

0x8B94 gets written to FFF0 – FFF1 and BFF0 – BFF1.

Go to the label RTIISR, convert that to a number and load it into IX. It does **not** say go to the memory location pointed to by RTIISR and load into IX.

The pound sign is an operator that converts what follows it to a number.

The original Motorola assembly language used $ before numbers to denote that it is a hex value. More recent assemblers also allow for the use of 0x, which is used by higher level programming languages.

Technically, for *our* robot, this is not applicable. You can include it; it won't hurt anything, but since the bot never comes up in normal mode, these vector table entries are ignored.

When you bring the robot up in bootstrap mode, the on chip boot ROM is suddenly on and present. It is in the memory map at the BF location. But, the BF page locations cannot be written to because the ROM is mapped to it.

When an interrupt occurs in bootstrap mode, the processor sets the PC to the location of the jump table entry. For example, for the RTI, it sets the PC to 00EB. There are therefore three bytes – one to allow the JMP command, and then two for the address to which the processor should jump. Not JSR – that messes with the stack – but JMP.

```
LDAA            #0x7E           ; 7E is the JMP op code
STAA            0x00EB
LDX             #RTIISR
STX             0x00EC
```

Since our robot can boot up in special test mode, the BF page location should also be written to from within the bootstrap code.

When the program is successfully downloaded from the computer to the robot, JBug11 and WinBug switch the robot back to special test mode. Interrupts work because it copies into ram the information needed about the vector table entries.

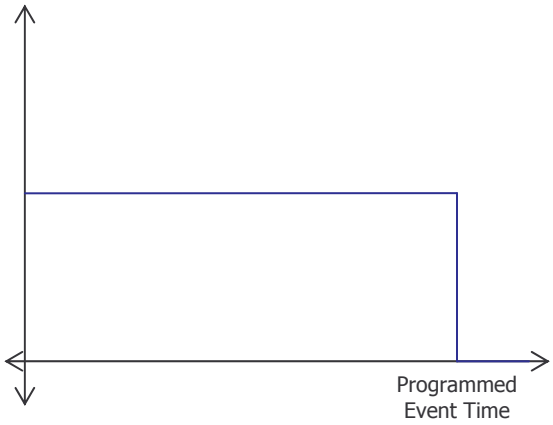
Reset Vector Table Entry

In the vector table, there is a reset vector. This is the first line of code that you would like the processor to execute when coming out of reset. In our system, the first line of code we want to execute is the bootstrap code of the operating system.

Since the memory never loses power as long as the batteries are plugged in, the program remains in memory even when the processor is turned off.

Timer Subsystem

1. **Real Time Interrupt (RTI)**
    a. Periodic interrupt. Once you set it up, you don't have to dink with it again. It just does its thing. Right now, it's the heartbeat for our operating systems.
2. **Input Capture (IC)**
    a. Grabs the time that a single edge occurs.
3. **Output Compare (OC)**
    a. If the current time compares to your program time, then an edge event will occur on the output.



4. **Pulse Accumulation**
    a. Very useful in metering the flow of fluids. They are a classic example of things that pulse. Being able to count the number of pulses that occur over a period of time is a good way to measure the rate at which it is flowing.
    b. It counts the number of pulses that occur in a time period. OR it can count the distance between edges. In other words, if the pulse goes high, it begins accumulating counts before it falls. It counts how long the pulse was there.

These four functions together are fundamental in timing. They can make many more functions we might need to generate things such as electronic fuel injection control.

The timer is the most complicated device on the HC11 microcontroller, but is probably the most useful.

Timing's Usefulness in Lab

The speed of a DC motor is proportional to the applied voltage and therefore the duty cycle of its voltage waveform (the high time divided by the period).

If I attach the motor to a battery, the input voltage to the motor is a constant flat line. When that occurs, the motor will run at its maximum speed.

Suppose, though, I begin switching the battery in and out at different rates. From calculus, we know that the average voltage being applied to the motor is the integral of the period.

Using output compare, we can generate a duty cycle wave.