C Programming for Microcontrollers
Why Use C?

• C is generally easier to use than assembler, therefore a programmer can be more productive

• C will not usually produce code that is as efficient as hand coded assembler, but modern optimizing compilers are pretty good

• C is standard (ANSI C) and therefore much more portable than assembler which is tied to a specific processor

• However, to use C effectively on a given microcontroller, you still need to know about (and use) bits and registers etc.
Quick Example

```c
#include <IO6811.H> /* internal register defs */

void main(void)
{
    DDRC = 0x00; /* make Port C an input port */
    while (1)
    {
        PORTB = PORTC; /* read the DIP switches (PORTC) and */
        /* write to the LEDs (PORTB) */
    }
}
```
Some C Issues

• C programs tend to use a lot of stack space, which can lead to stack overflow

• Assembly can (should?) be used where speed is critical. You can almost always write better assembly (machine) code than a compiler

• Mixed C and Assembly programs are very often used

• Remember that floating point arithmetic (float, double) can take up to 100 times longer to execute than integer arithmetic on the 68HC11
C Compiler

- The C Compiler processes one `.c` file at a time. The file is parsed, and translated to relocateable machine code, known as an object file.

- Have a look at the list file `.lst` to see how the compiler is translating your C statements into machine code.

- The object code is not targeted at any particular memory location.
Linker

• “a linker is a program that combines object modules to form an executable program” (from www.webopedia.com)

• “In addition to combining modules, a linker also replaces symbolic addresses with real addresses. Therefore, you may need to link a program even if it contains only one module.”

• Have a look at the file ‘icc6811cl.bat’ in the ‘c:\bat’ directory to see how this is done for a single source file

• You will need to write your own ‘bat’ file for multi-file projects
Process Diagram
Register Operations

• There are many ways to write to or read from a register.
• For Bitwise write operations:
  – Calculate the 8 bit value you wish to write to a register,
    \( \text{writeValue} = 0x45, \) or
  – Define bit fields, e.g. BIT1 = 0x01; BIT2 = 0x02, BIT3 = 0x04 etc., then
    \( \text{writeValue} = \text{BIT7|BIT3|BIT1}; \) to set individual bits, or
  – Use a macro, e.g.  
    \[
    \text{#define bit(x) \ (1} \ll \ (x)\)
    \]
    \( \text{writeValue} = \text{bit(7)|bit(3)|bit(1)}; \)
  – Etc.

• To test for bits:
  – If( \( \text{readValue} \& 0x02 \)), or
  – If( \( \text{readValue} \& \text{BIT2} \)), or
  – If( \( \text{readValue} \& \text{bit(2)} \)).
Using Interrupts

• The icc6811 compiler provides a fairly simple way to use interrupts in your C program

• First, include the ‘int6811.h’ file in your code (its in the ‘INC’ directory of the compiler). This provides function prototypes for all the interrupts in the interrupt vector.

• Then, simply write the function (prefixed with the ‘interrupt’ keyword)

• In your main code, unmask the interrupt
Interrupt Example

#include <int6811.H> /* ISR prototype defs */
#include <IO6811.H> /* internal register defs */

interrupt void TO_interrupt()
{
    if (PORTA & 1) do_something();
}
Debugging in C

• Understand and use the .sym file
  – For in-depth debugging, use the .LST files as-well.

• Use LEDs, Serial and 7-Segs
  – If these modules are correctly designed and well tested, they become debugging tools.

• Make sure you understand the expectation...