Programming the HC12 in C

- A comparison of some assembly language and C constructs

```
Assembly                          |   C
----------------------------------|-------------------------------------
; Use a name instead of a num     |   /* Use a name instead of a num */
COUNT:  EQU  5                    |   #define COUNT 5
;----------------------------------|   /*-------------------------------*/
;start a program                  |   /* To start a program */
CODE:  section  .text             |   main()
       org   $0800                |   {
       lds   #0x0A00              |   }
;----------------------------------|   /*-------------------------------*/
```

- Note that in C, the starting location of the program is defined when you compile the program, not in the program itself.
- Note that C always uses the stack, so C automatically loads the stack pointer for you.

```
Assembly                          |   C
----------------------------------|-------------------------------------
;allocate two bytes for           |   /* Allocate two bytes for
;a signed number                  |    * a signed number */
                                  |
DATA: section .data               |
      org     $0900               |
i:    ds.w    1                   |   int i;
j:    dc.w    $1A00               |   int j = 0x1a00;
;----------------------------------|   /*-------------------------------*/
;allocate two bytes for           |   /* Allocate two bytes for
;an unsigned number               |    * an unsigned number */
i:    ds.w    1                   |   unsigned int i;
j     dc.w    $1A00               |   unsigned int j = 0x1a00;
;----------------------------------|   /*-------------------------------*/
;allocate one byte for            |   /* Allocate one byte for
;an signed number                 |    * an signed number */
                                  |
i:    ds.b    1                   |   signed char i;
j:    dc.b    $1F                 |   signed char j = 0x1f;
```

```
Assembly                          |   C
;-------------------------------- |   /*-------------------------------*/
;Get a value from an address      |   /* Get a value from an address */
; Put contents of address         |   /*  Put contents of address    */
; $E000 into variable i           |   /*  0xE000 into variable i     */
                                  |
i:  ds.b   1                      |   unsigned char i;
                                  |
    ldaa   $E000                  |   i = * (unsigned char *) 0xE000;
    staa   i                      |
;-------------------------------- |   /*-------------------------------*/
                                  |   /* Use a variable as a pointer
                                  |      (address) */
                                  |
                                  |   unsigned char *ptr, i;
                                  |
                                  |   ptr = (unsigned char *) 0xE000;
                                  |   i = *ptr;
                                  |   *ptr = 0x55;
;-------------------------------- |   /*-------------------------------*/
```

- In C, the construct `*(num)` says to treat `num` as an address, and to work with the contents of that address.
- Because C does not know how many bytes from that address you want to work with, you need to tell C how many bytes you want to work with. You also have to tell C whether you want to treat the data as signed or unsigned.
  - `i = * (unsigned char *) 0xE000;` tells C to take one byte from address `0xE000`, treat it as unsigned, and store that value in variable `i`.
  - `j = * (int *) 0xE000;` tells C to take two bytes from address `0xE000`, treat it as signed, and store that value in variable `j`.
  - `* (char *) 0xE000 = 0xaa;` tells C to write the number `0xaa` to a single byte at addess `0xE000`.
  - `* (int *) 0xE000 = 0xaa;` tells C to write the number `0x00aa` to two bytes starting at addess `0xE000`.

```
Assembly                        |   C
;-----------------------------|   /*-----------------------------*/
;To call a subroutine          |   /* To call a function */
  ldaa  i                       |   sqrt(i);
  jsr   sqrt                    |
;-----------------------------|   /*-----------------------------*/
;To return from a subroutine   |   /* To return from a function */
  ldaa  j                       |   return j;
  rts                           |

;-----------------------------|   /*-----------------------------*/
;Flow control                  |   /* Flow control */
    blo                         |   if (i < j)
    blt                         |   if (i < j)
                                |
    bhs                         |   if (i >= j)
    bge                         |   if (i >= j)
;-----------------------------|   /*-----------------------------*/
                                |
```

- Here is a simple program written in C and assembly. It simply divides 16 by 2. It does the division in a function.

```
ASSEMBLY                      | C
------------------------------|----------------------
                              |
DATA: section .data           |  signed char i;
      org     $0900           |
i:    ds.b    1               |
                              |
CODE: section .text           |
                              |  signed char div(signed char j);
      org     $0800           |  main()
      lds     #$0A00          |  {
      ldaa    #16             |     i = div(16);
      jsr     div             |  }
      staa    i               |
      swi                     |
                              |
div:  asra                    |  signed char div(signed char j)
      rts                     |  {
                              |     return j >> 1;
                              |  }
```

A simple C program and how to compile it

Here is a simple C program

```
#define COUNT 5

unsigned int i;

main()
{
    i = COUNT;
}
```

To compile the program, make a linker file simple.lkf:

```
#    link command file for test program
#
+seg .text -b 0x0800 -n .text    # program start address
+seg .const -a .text             # constants follow code
+seg .data -b 0x0900             # data start address
crts.o                           # startup routine
simple.o                         # application program
+seg .const -b 0xffce            # vectors start address
+def __memory=@.bss              # symbol used by library
+def __stack=0x0a00              # stack pointer initial value
```

You need a C startup file crts.s

```
;   C STARTUP FOR MC68HC12
;
;
    xdef    _exit, __stext
    xref    _main, __memory, __stack
;
    switch  .bss
__sbss:
    switch  .text
__stext:
    lds #__stack    ; initialize stack pointer
    jsr _main       ; execute main
_exit:
    swi             ; Return to D-Bug 12
;
    end
```

You can use the following batch file cc.bat to compile the program, create a simple.h12 file to load into the Zap simulator, and a simple.s19 file to load into the HC12:

```
cx6812 -vl -p -xx -a -xx crts.s %1.c
echo Linking ...
clnk -o %1.h12 %1.lkf
echo Converting ...
chex -o %1.s19 %1.h12
echo Generating absolute listing ...
clabs %1.h12
```

Here is the machine code generated by the compiler:

```
 1                      ; C Compiler for MC68HC12 [COSMIC Soft
 2                      ; Version V4.2g - 13 Oct 1998
29                      ; 5 main()
29                      ; 6 {
30                         switch   .text
31  0815                _main:
35                      ; 7    i = COUNT;
37  0815 cc0005            ldd #5
38  0818 7c0000            std _i
39                      ; 8 }
42  081b 3d                rts
65                         xdef    _main
66                         switch  .bss
67  0000                _i:
68  0000 0000              ds.b    2
69                         xdef    _i
70                         end
```

Pointers in C

- To access a memory location:

```
*address
```

- You need to tell compiler whether you want to access 8-bit or 16 bit number, signed or unsigned:

```
*(type *)address
```

  - To read from an eight-bit unsigned number at memory location `0x0900`:

```
x = *(unsigned char *)0x0900;
```

  - To write an `0xaa55` to a sixteen-bit signed number at memory locations `0x0910` and `0x0911`:

```
*(signed int *)0x0910 = 0xaa55;
```

- If there is an address which is used alot:

```
#define PORTA (* (unsigned char *) 0x0000)

x = PORTA;      /* Read from address 0x0000 */
PORTA = 0x55;   /* Write to address 0x0000 */
```

- To access consecutive locations in memory, use a variable as a pointer:

```
unsigned char *ptr;

ptr = (unsigned char *)0x0900;
*ptr = 0xaa;          /* Put 0xaa into address 0x0900 */
ptr = ptr+2;          /* Point two further into table */
x = *ptr;             /* Read from address 0x0902     */
```

- To set aside ten locations for a table:

```
unsigned char table[10];
```

- Can access the third element in the table as:

```
table[2]
```

  or as

```
*(table+2)
```

- To set up a table of constant data:

```
const unsigned char table[] = {0x00,0x01,0x03,0x07,0x0f};
```

  This will tell the compiler to place the table of constant data with the program (which might be placed in EEPROM) instead of with regular data (which must be placed in RAM).

- There are a lot of registers (such as `PORTA` and `DDRA`) which you will use when programming in C. Rather than having to define the registers each time you use them, you can include a header file for the HC12 wihch has the registers predefined. Here is the beginning of the header file `hc12b32.h`. You can find the complete file on the EE 308 homepage.

```
/*
* C include file for EE 308
* Spring, 2002
*/

#ifndef __HC12_H
#define __HC12_H    1

/* base address of register block, change this if you reloca
 * block.
 */

#define _BASE    0x0000

#define PORTA  (* (unsigned char *)(_BASE+0x00))
#define PORTB  (* (unsigned char *)(_BASE+0x01))
#define DDRA   (* (unsigned char *)(_BASE+0x02))
#define DDRB   (* (unsigned char *)(_BASE+0x03))
```