- **Introduction to Programming the 9S12 in C**
- **Huang Sections 5.2 and 5.3**
  - Comparison of C and Assembly programs for the HC12
  - How to compile a C program using the GNU-C compiler
  - Using pointers to access the contents of specific addresses in C

**Exam 1**

• You will be able to use all of the Motorola data manuals on the exam.
• No calculators will be allowed for the exam.
• Numbers
– Decimal to Hex (signed and unsigned)
– Hex to Decimal (signed and unsigned)
– Binary to Hex
– Hex to Binary
– Addition and subtraction of fixed-length hex numbers
– Overflow, Carry, Zero, Negative bits of CCR
• Programming Model
– Internal registers – A, B, (D = AB), X, Y, SP, PC, CCR
• Addressing Modes and Effective Addresses
– INH, IMM, DIR, EXT, REL, IDX (Not Indexed Indirect)
– How to determine effective address
• Instructions
– What they do - Core Users Guide
– What machine code is generated
– How many cycles to execute
– Effect on CCR
– Branch instructions – which to use with signed and which with unsigned
• Machine Code
– Reverse Assembly
• Stack and Stack Pointer
– What happens to stack and SP for instructions (e.g., PSHX, JSR)
– How the SP is used in getting to and leaving subroutines
• Assembly Language
– Be able to read and write simple assembly language program
– Know basic assembler directives – e.g., equ, dc.b, ds.w
– Flow charts

## A simple C program and how to compile it

Here is a simple C program

```
#define COUNT 5
unsigned int i;
main()
{
        i = COUNT;
}
```

Details of compiling a program are discussed in detail in the text in **Section 5.10**. Here is an outline of the details:

**1.** Open the Embedded GNU (EGNU) IDE.

**2.** From the File menu, select the New Source File option. Type in your C program. Then from the File menu, select the Save unit as submenu, and save your file with an appropriate name and in an appropriate directory.

**3.** From the *File* menu, select the *New Project* option. Give the project an appropriate name and an appropriate directory. (Note: the project base name must be different from the C file names.) When the *Project Options* popup dialog appears, click the down arrow below *Hardware Profile*, and select *Dragon12*. Click the *Edit Profile* button, and make sure the following are set:

- **ioports from 0000, length 400**
- **eeprom from 400, length c00**
- **data from 1000, length 1000**
- **text from 2000, length 2000**
- **stack at 3c00**

Then click the OK button.

**4.** From the Project menu, select the Add to project option, and, in the pop-up dialog box, select the C file you entered in Step 2.

**5.** From the Build menu, select the Make option. Under the Compiler window at the bottom of the screen, you will hopefully see the message No errors or warnings. If not, you will need to fix the errors.

**6.** If all went well, you should be able to download the compiled file into the 9S12.

If the name of your project is ***Project1.prj***, the compiler will generate a file ***Project1.dmp***. Here is some of the output from ***Project1.dmp***. There are a couple of things you should note about this output:

• The first thing the C program does is load the stack pointer.

• The main() function is the assembly language for the C program you entered.

```
Disassembly of section .text:

00002000 <_start>:
    2000:   cf 3c 00          lds   #3c00 <_stack>
    2003:   16 20 38          jsr   2038 <__premain>

00002006 <__map_data_section>:
    2006:   ce 20 42          ldx   #2042 <__data_image>
    2009:   cd 10 00          ldy   #1000 <__data_section_start>
    200c:   cc 00 00          ldd   #0 <__data_section_size>
    200f:   27 07             beq   2018 <__init_bss_section>

00002011 <Loop>:
    2011:   18 0a 30 70       movb  1,X+, 1,Y+
    2015:   04 34 f9          dbne  D,2011 <Loop>

00002018 <__init_bss_section>:
    2018:   cc 00 02          ldd   #2 <__bss_size>
    201b:   27 08             beq   2025 <Done>
    201d:   ce 10 00          ldx   #1000 <__data_section_start>

00002020 <Loop>:
    2020:   69 30             clr   1,X+
    2022:   04 34 fb          dbne  D,2020 <Loop>

00002025 <Done>:
    2025:   16 20 31          jsr   2031 <main>

00002028 <fatal>:
    2028:   16 20 3c          jsr   203c <_exit>
    202b:   20 fb             bra   2028 <fatal>
    202d:   20 06             bra   2035 <main+0x4>
    202f:   20 18             bra   2049 <__data_image+0x7>

00002031 <main>:
    2031:   18 03 00 05       movw  #5 <__bss_size+0x3>, 1000
<__data_section_start>
    2035:   10 00
    2037:   3d                rts

00002038 <__premain>:
    2038:   87                clra
    2039:   b7 02             tap
    203b:   3d                rts

0000203c <_exit>:
    203c:   10 ef             cli
    203e:   3e                wai
    203f:   20 fb             bra   203c <_exit>

00002041 <_etext>:
    2041:   a7                nop
```

## 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 0x2000:

```
x = *(unsigned char *)0x2000;
```

– To write an 0xaa55 to a sixteen-bit signed number at memory locations 0x2010 and 0x2011:

```
*(signed int *)0x2010 = 0xaa55;
```

• If there is an address which is used a lot:

```
#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 *)0x2000;
*ptr = 0xaa;        /* Put 0xaa into address 0x2000 */
ptr = ptr+2;        /* Point two further into table */
x = *ptr;           /* Read from address 0x2002 */
```

• 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 which has the registers predefined. Here is a sample of the **hcs12.h**. You can find the complete file on the EE 308 homepage.

Here are a few entries from the header file:

```
#define     IOREGS_BASE  0x0000

#define     _IO8(off)    *(unsigned char  volatile *)(IOREGS_BASE + off)
#define     _IO16(off)   *(unsigned short volatile *)(IOREGS_BASE + off)

#define     PORTA        _IO8(0x00)     //port a = address lines a8 - a15
#define     PORTB        _IO8(0x01)     //port b = address lines a0 - a7
#define     DDRA         _IO8(0x02)     //port a direction register
#define     DDRB         _IO8(0x03)     //port a direction register

#define     PORTE        _IO8(0x08)     //port e = mode, irq and control signals
#define     DDRE         _IO8(0x09)     //port e direction register
#define     PEAR         _IO8(0x0A)     //port e assignments
#define     MODE         _IO8(0x0B)     //mode register
#define     PUCR         _IO8(0x0C)     //port pull-up control register
#define     RDRIV        _IO8(0x0D)     //port reduced drive control register
#define     EBICTL       _IO8(0x0E)     //stretch control
```

**Here is a program which uses hcs12.h to write a 0x55 to PORTA:**

```
#include "hcs12.h"
main()
{
        DDRA = 0xff;          // Make PORTA output
        PORTA = 0x55;         // write a 0x55 to PORTA
        asm(" swi");
}
```

**Setting and Clearing Bits using Assembly and C**

• To put a specific number into a memory location or register (e.g., to put 0x55 into PORTA):

– In assembly:
  ldaa #$55
  staa PORTA

– In C:
  PORTA = 0x55;

• To set a particular bit of a register (e.g., set Bit 4 of PORTA) <u>while leaving the other bits unchanged</u>:

– In assembly, use the bset instruction with a mask which has 1's in the bits you want to set:

  bset PORTA,#$10

– In C, do a bitwise OR of the register with a mask which has 1's in the bits you want to set:

```
PORTA = PORTA | 0x10;
```

• To clear a particular bit of a register (e.g., clear Bits 0 and 5 of PORTA) while leaving the other bits unchanged:

– In assembly, use the blcr instruction with a mask that has 1's in the bits you want to clear:

```
bclr PORTA,#$21
```

– In C, do a bitwise AND of the register with a mask that has 0's in the bits you want to clear:

```
PORTA = PORTA & 0xDE;
```

or

```
PORTA = PORTA & ~0x21;
```

## Waiting for a bit to be set or cleared in Assembly and C

• You often have to wait for an event to occur before taking some action.
• For example, wait for the "wash" cycle to finish before starting the "rinse" cycle.
• Usually, when an event occurs, a bit is either set or cleared.
• Here is how to wait until Bit 3 of PORTB is set:

– In assembly:

```
l1:  brclr    PORTB,#$08,l1
```

– In C:

```
while ((PORTB & 0x08) == 0) ;
```

• Here is how to wait until Bit 3 of PORTB is cleared:

– In assembly:

```
l1:  brset    PORTB,#$08,l1
```

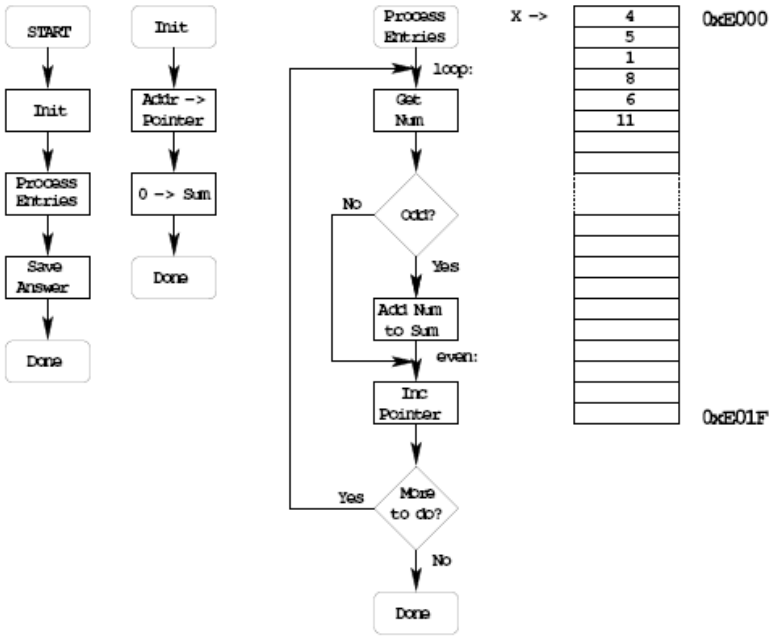
– In C:

```
while ((PORTB & 0x08) != 0) ;
```

## Sum the odd 8-bit numbers in an array

• Write a program to sum all the odd numbers in an array of data.
• The numbers in the array should be treated as unsigned 8-bit numbers.
• The array starts at address 0xE000 and ends at address 0xE01F.
• This is the same program which was done in assembly language before.

Sum odd 8-bit numbers in array from 0xE000 to 0xE01f




## Convert to C

How to check if odd?
Divide by 2, if REM = 1 odd, Modulo (%) in C returns REM

```
main(){
        ptr = (unsigned char *) 0xe000;
        sum = 0;
        do{
                x = *ptr;
                if ((x % 2) = = 1) {
                        sum = sum + x;
                }
                ptr = ptr + 1;
        }
        while (ptr <= (unsigned char *) 0xe01f);
}
```

```
/* Program to sum the odd numbers in an array
* The numbers are unsigned characters
* The array starts at address 0xE000 and
* is 0x20 bytes long
*/
#include "DBug12.h"

#define     START 0xE000
#define     LEN 0x20
#define     END (START+LEN-1)

main()
{
        unsigned int sum;           /* Keep the running sum (need 16-bit number) */
        unsigned char *ptr;         /* Pointer to array element */
        unsigned char x;            /* Character from array */

        ptr = (unsigned char *) START;
        sum = 0;
        do
        {
                x = *ptr;           /* Get entry */
                if ((x % 2) == 1)   /* Is number odd? */
                {
                        sum = sum + x; /* Odd: add to sum */
                                    /* C automatically makes x 16-bits */
                }
                ptr = ptr + 1;      /* Advance to next */
        }
        while (ptr <= (unsigned char *) END); /* Done? */
        DB12FNP -> printf("sum = %d\r\n",sum);
        asm(" swi");
}
```

## A software delay

• To enter a software delay, put in a nested loop, just like in assembly.

– Write a function **delay**(**num**) which will delay for *num milliseconds*

```
void delay(unsigned int num)
{
        unsigned int i;
        while (num > 0){
                x = y;
                                        /* ---------------------------- */
                while (x > 0){          /*Want inner loop to delay   */
                        x = x - 1;      /* for 1ms                    */
                }                       /*                            */
                num = num - 1;
        }
}
```

• What should X be to make a 1 ms delay?

• Try using x = 1000 (0x3e8).

• Look at assembly listing generated by compiler:

```
void delay(unsigned int num);

int main(){
      unsigned int i;
      i = 1;
      delay(i);
      asm("swi");
}

void delay(unsigned int num)
{
      unsigned int x;
      while(num > 0){
                  x = 1000;
                  while(x > 0){
                        x = x - 1;
                  }
                  num = num - 1;
       }
}
```

```
00002031 <main>:
   2031:    cc 00 01    ldd    #1 <__bss_size+0x1>
   2034:    07 02       bsr    2038 <delay>
   2036:    3f          swi
   2037:    3d          rts

00002038 <delay>:
   2038:    04 44 0c    tbeq   D,2047 <delay+0xf>
   203b:    ce 03 e8    ldx    #3e8 <__bss_size+0x3e8>
   203e:    1a e1 e7    leax   -25,X
   2041:    04 75 fa    tbne   X,203e <delay+0x6>
   2044:    04 34 f4    dbne   D,203b <delay+0x3>
   2047:    3d          rts
```

Inner Loop

The inner loop takes **5 clock cycles** (leax (2) + tbne (3)).

• One millisecond takes 24,000 cycles

(24,000,000 cycles/sec × 0.001 sec = 24,000 cycles)

• Need to execute inner loop 24,000 cycles / 5 cycles = 4**800 times to delay for 1 millisecond**

```
void delay(unsigned int num)
{
        unsigned int x;
        while (num > 0){
                x = 4800;
                                        /* ---------------------------- */
                while (x > 0){          /*Want inner loop to delay   */
                        x = x - 1;      /* for 1ms                    */
                }                       /*                            */
                num = num - 1;
        }
}
```

**Program to increment LEDs connected to PORTA, and delay for 50 ms between changes**

```
#include "hcs12.c"
#define delay1ms (24000/5)          // Inner loop takes 5 cycles
                                    // Need 24,000 cycles for 1 ms
void delay(unsigned int num);
main() {
        DDRA = 0xff;                // Make PORTA output
        PORTA = 0;                  // Start with all off
        while(1){
                PORTA = PORTA + 1;
                delay(50);
        }
}

void delay(unsigned int num){
        unsigned int x;
        while (num > 0){
                x = delay1ms;
                while (x > 0){
                        x = x - 1;
                }
                num = num - 1;
        }
}
```

**Program to display a particular pattern of lights on PORTA**

```
#include "hcs12.c"
#define delay1ms (24000/5)          // Inner loop takes 5 cycles
                                    // Need 24,000 cycles for 1 ms
#define TABLEN 8

void delay(unsigned int num);
main()
{
        const char table[] = {0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
        int i;

        DDRA = 0xff;                // Make PORTA output
        PORTA = 0;                  // Start with all off
        i = 0;
        while(1)
```

```
        {
                PORTA = table[i];
                delay(50);
                i = i + 1;
                if (i >= TABLEN) i = 0;        // Start over when
                                               // end is reached
        }
}
```