EE 308 – LAB 5

C Programming Language

Introduction

The C programming language is used extensively in programming microprocessors. In this lab you will write some simple C programs which do the things you did in assembly language in the last lab.

For example, the following C program increments Port B:

```
/* A C Language Program to Increment Port B on a 68HC12 */
#include <iodp256.h>         /* Get the HCS12DP256 definitions */
#define D_1MS (24000/6)      /* Inner delay loop takes 6 cycles */
                             /* 24000 cycles = 1 ms for 24 MHz clock */
#define TRUE  1              /* A normal C define */
void delay(unsigned int ms);

void main()                  /* The main program */
{
    DDRB = 0xff;             /* Make all bits of Port B output */
    PORTB = 0;
    while (TRUE)             /* Do forever */
    {
        PORTB = PORTB + 1;   /* Increment Port B */
        delay(100);          /* Wait 100 ms */
    }
}

/* Function to delay ms milliseconds */
void delay(unsigned int ms)
{
    int i;

    while (ms > 0)
    {
        i = D_1MS;
        while (i >0)
        {
            i = i - 1;
        }
        ms = ms - 1;
    }
}
```

Figure 1: A C program to increment Port B.

PreLab

For the pre-lab write the programs for Part 4 of this lab.

The Lab

1. Type in the above C program (or download it from the web) and give it the name `inc.c`. Open a Command window. Compile the program as described in the **TIPS** section at the end of this lab.

   You should now have the files `inc.la`, `inc.h12`, `inc.s19` and `inc.map` in your directory.

   (a) `inc.la` is the assembly language listing generated by the C compiler. Look at the file and try to understand what it does. Note that there may be some things which do not make sense to you. At the very least, find the assembly language code which increments Port B. (Note that the C compiler produces assembly code in decimal rather than hexadecimal.)

   (b) Look at the file `inc.map`. This shows the addresses of the start of the functions in the program, as well as the addresses of any global variables. (Since the `inc.c` program does not use any global variables, none will appear in the `inc.map` file. The local variables used in `inc.c` are allocated on the stack when they are needed.)

   Note that the function and variable names are preceded by an underscore. Note also that there is a function `_exit`. Find the address of this function.

   (c) Look at the file `inc.s19`. This contains the op codes that will be loaded into the HC12. Reverse assemble the `_exit` function. What does this do?

2. Load the file `inc.h12` into your `ZAP` simulator and run it. Note that you can see both the assembly code which the HC12 will execute and the C code used to generate the assembly code.

3. Load the file `inc.s19` into your HC12 and run it. Verify that Port B increments.

4. Using the program `inc.c` as a model, write a C program to implement the program from Lab 4.

5. Compile and run your program. Have an instructor verify that it works.

6. Look at the `lab05.map` file for this week's lab, and determine how many bytes the program takes (the length of the `.text` segment). Compare this to the length of last week's program written in assembly.

7. Put your program in the EEPROM at address `0x0400`. Remember, when you put code into EEPROM you need to do some setup which DBug12 normally does for you. It is easiest to do this in assembly language. You could add this to the `crts.s` file and have a special startup file for whenever you want to load code into EEPROM. Alternatively, you could add the following as the first few instructions of your C program:

```
_asm("    ldaa    #$55");
_asm("    staa    $003f");
_asm("    coma");
_asm("    staa    $003f");
_asm("    clr     $003c");
_asm("    ldab    #$11");
_asm("    nop");
_asm("    stab    $0010");
_asm("    ldab    #$00");
_asm("    stab    $0035");
_asm("    ldab    #$05");
_asm("    stab    $0034");
_asm("    nop");
_asm("    nop");
_asm("    nop");
_asm("    nop");
_asm("l1: brclr   $0037,#$08,l1");
_asm("    bset    $0039,#$80");
```

Also, note that you will want the array which stores the turn signal patterns into the EEPROM (so the array will not disappear when you turn off power). You will want variables which will change as the program is executed to be placed in RAM. You can tell the compiler to put an array in EEPROM by defining the array as type `const`, and telling the linker to put the `const` section in EEPROM following the `text` section. An example of setting up an array of type `const` is

```
const char table[] = {0xaa, 0xbb, 0xcc};
```

For more information on putting your C program into EEPROM, read the **TIPS** section below.

TIPS:

- To compile a C program a startup file called `crts.s` is needed. The file we will use will clear initialized global variables to zero, load the stack pointer and jump to the `main()` function of the C program. After the main program finishes, it returns to `crts.s`; the `swi` instruction if `crts.s` returns control to DBug-12. The `crts.s` file which you should use can be found in `C:\CX32\CRTS.S`. Here are the contents of that file:

```
;   C STARTUP FOR MC68HC12
;   Copyright (c) 1996 by COSMIC Software
;
    xdef    _exit, __stext
    xref    _main, __memory, __stack
;
    switch   .bss
__sbss:
```

```
    switch   .text
__stext :
    clra                ; reset the bss
    clrb
    ldx       #__sbss   ; start of bss
    bra       loop      ; start loop
zbcl:
    std       2,x+      ; clear word
loop:
    cpx       #__memory ; up to the end
    blo       zbcl      ; and loop
    lds       #__stack  ; initialize stack pointer
    jsr       _main     ; execute main
_exit:
    swi                 ; return to DBug12
;
   end
```

- Copy `c:\cx32\crts.s` into your directory. Create a linker file `lab05.lkf` like this:

```
# link command file for test program
#
+seg .text -b 0x1000 -n .text # program start address
+seg .const -a .text          # constants follow code
+seg .data -b 0x2000          # data start address
crts.o                        # startup routine
lab05.o                       # application program
+def __memory=@.bss           # symbol used by library
+def __stack=0x3C00           # stack pointer initial value
```

  To put the program into EEPROM change the location of the `.text` segment from `0x1000` to `0x0400`.

- To make compiling programs easier make a batch file called, e.g., `cc.bat`. An example of a `cc.bat` file is:

```
c:\cx32\cx6812 -vl -ax +debug crts.s %1.c
c:\cx32\clnk -o %1.h12 -m %1.map %1.lkf
c:\cx32\chex -o %1.s19 %1.h12
c:\cx32\clabs %1.h12
```

  Then to compile the file `lab05.c`, just give the command `cc lab05`.

- If you have a program which uses initialized static variables, please read the section on static variables in the Cosmic compiler manual (available from the instructor). It may be simpler at this point to use global variables which are initialized in the main program rather than static initialized variables.