Notes - Unit 6

C LANGUAGE PROGRAMMING

DATA TYPES FOR HCS12

Signed integer (2's complement)	Unsigned integer	Number of bytes
`char`	`unsigned char`	1
`int`	`unsigned int`	2
`long int`	`unsigned long int`	4

C PROGRAM STRUCTURE FOR HCS12

- The main function usually looks like this:

```
#include <hidef.h>      /* common defines and macros: file found in CodeWarrior folder */
#include "derivative.h" /* derivative-specific definitions (e.g.: PORTA, DDDRK) */
#include <stdlib.h>     /* for malloc, calloc */

/* Global Variables: For Small Memory Model, they are place right after the Stack,
                     i.e., starting from $1100. Stack is empty at $1100*/
unsigned char a, b; // a, b: unsigned 8-bit numbers

/* Small Memory Model: Program starts at $C000 */

// Function declaration:
unsigned char myfun (unsigned char val); // Input: 1 byte. Output: 1 byte

/* Body: This is considered to be a Subroutine in the ASM Code */
void main(void) {
   /* Local Variables: */
   unsigned char i;
   ...
   ...
}

/* Functions:*/
unsigned char myfun (unsigned char val)
{
   unsigned char temp;
   ...
   ...
   return temp; // The value of 'temp' is returned
}
```

CODEWARRIOR TIPS

- **Memory Model:** Small (Program/Data fit into the 64 KB memory space).
- **Program and Data Location**: Program starts at `$C000` (16 KB Fixed Flash, Page `$3F`). By default, data starts at `$1100`. The Stack starts (is empty) at `$1100` (Stack Size = `0x100`).
- **Debug**: the main program in C is treated as a subroutine and the variables are local variables stored in the Stack. The windows `Data:1` and `Data:2` let us see how the variables change in real time.
  - ✓ Use *Step Over* to avoid entering the ASM code of every C instruction
  - ✓ Use *Step On* when you are about to execute a C function if you want to execute the C function line by line. Once inside the C function, use Step Over to avoid entering the ASM code of every C instruction.
- **Global variables**: These are defined outside the main program. Global variables will be placed starting the address `$1100`. Initializing a global variable requires instruction loading data, it is not the same as `dc.b, dc.w`.
- **Constants**: They do not occupy memory positions. They are equivalent to EQU.
- **Local variables:** Initializing them is akin to execute a series of instruction to load data on the Stack. It is not the same as dc.b, dc.w. The initial values are loaded using normal ASM instructions.
- **I/O registers**: They occupy the memory space from `$0000` to `$03FF`. We can use all the common names: `PORTA, PORTB, PORTP, PORTH, PORTM, DDRA, DDRB, DDRP, DDRH, DDRM`, etc.
- **Sign-extension**: In additions and subtractions, we do not have to worry about sign extension. If all the variables are signed and if we accumulate numbers into a larger data size, sign-extension will be taken care of automatically.

EXAMPLES

- Given 10 16-bit unsigned numbers in an array, compute the sum. Then add the sum to an initial value.
  The following topics are covered in this example:
  - ✓ Use of: i) a local variable defined as a constant, and ii) global variable with an initial value.
  - ✓ For-loop.
  - ✓ Inline Assembly Instructions: It is strongly suggested to use: i) CPU registers, ii) I/O registers, and iii) variables in C as memory locations. Example: `asm("ldd sum");` We load the value of the variable sum onto register `D`. Note that you might modify the execution of your program by modifying the registers. If you are not sure whether you will affect the execution of your program, you can do: `asm ("pshd"); asm ("ldd sum"); asm ("puld").`

  ASM Code: `unit6a.c`

- Compute the average of an array. Use the DIP Switch to do an OR operation between the computed average and the DIP Switch state. The result must appear on the LEDs.
  The following topics are covered in this example:
  - ✓ Use of functions in C (input parameters, output parameter)
  - ✓ Use of HCS12 I/O Registers
  - ✓ Use of bit-wise AND operator and OR operator.

  ASM Code: `unit6b.c`

- Generate the Fibonacci numbers from `F(0)` to `F(n)` given `n`.
  The following topics are covered in this example:
  - ✓ If-else statement.
  - ✓ Do-while statement

  ASM Code: `unit6c.c`

- For an array of numbers, find the maximum or minimum (depending on DIP Switch bit 0). If the bit is 0, the minimum appears on the LEDs. If the bit is 1, the maximum appears on the LEDs
  The following topics are covered in this example:
  - ✓ Function in C (array as input)
  - ✓ While statement

  ASM Code: `unit6d.c`

- For an array of numbers, compute the polynomial and store it on another array.
  The following topics are covered in this example:
  - ✓ Type-casting in C.
  - ✓ Functions in C: Input array, Output array.
  - ✓ Pointers.

  ASM Code: `unit6e.c`

MIXED C/ASSEMBLY PROGRAMMING:
The following topics are covered in these examples:
- ✓ Working with Mixed C/Assembly code in CodeWarrior.
- ✓ Creation the header and ASM files.
- ✓ Input parameters to an ASM Function: 1 byte (`B`), 2 bytes (`D`), 3 bytes (`B:X`), 4 bytes (`X:D`)
- ✓ Getting the return value of an ASM function as an accessible value in the C code.

- **Mixed C/Assembly Programming**: Add a number stored as a constant to another number.
  Here, we also use structures in C
  C Code: `unit6f.c mixasm_6f.h` **ASM Code**: `mixasm_6f.asm`

- **Mixed C/Assembly Programming**: Display a 4-byte number on the 7-segment displays
  C Code: `unit6g.c mixasm_6g.h` **ASM Code**: `mixasm_6g.asm`

- **Mixed C/Assembly Programming**: Read the KeyPad and display hex value on all 7-segment Displays.
  C Code: `unit6h.c mixasm_6h.h` **ASM Code**: `mixasm_6h.asm`