

## Introduction of C Compiler

### HMS800C Coding Know-How

#### DESCRIPTION

In the embedded system, the code size is one of the most important factor specially in coding with C. Some programmer who does not encounter embedded C either experienced embedded C not our HMS800C should consider following these rules. I strongly recommend you should follow these rules in our HMS800C. But these recommended methods are not absolute methods.

This application note describes how to reduce the code size and how to change C code to our assembler which will reduce the code size and is necessary in application which requires critical time. Our HMS800C is perfectly not mature. so you will encounter some problems which are repaired by using the other algorithm or using inline assembler. These problems will be figured out by upgrading HMS800C in a few months.

#### How to Access The Speical RAM

Some of the HMS800 series has speical display RAM for LCD or VFD. To access the display RAM, there are two methods. One is the using pointer, and the other is the using array. Actually, both of mothods are same way. Below examples describe how to write display RAM in the HMS87C2248 which has display RAM from 0x400 to 0x46F. The code size of example 3 is less than example 1 or example 2. If you want to reduce the code size, We recommend example 3 method.

```
#define VFD_data ((unsigned char *) 0x400)

void function(void)
{
    *(VFD_data) = 0xff;
    *(VFD_data+1) = 0xff;
    *(VFD_data+2) = 0xff;
}
```

**Example 1. Access Display RAM by Pointer**

```
unsigned char VFD_data[0x6f] PAGE4;

void function(void)
{
    VFD_data[0] = 0xff;
    VFD_data[1] = 0xff;
    VFD_data[2] = 0xff;
}
```

**Example 2. Access Display RAM by Array**

```
#define VFD_data ((unsigned char *) 0x400)
void function(void)
{
    asm("
        lda    #0ffh
        sta    !_VFD_data
        sta    !_VFD_data+1
        sta    !_VFD_data+2
    ");
}
```

**Example 3. Access Display RAM in the Inline-asm**

#### How to Call Function in the Inline-asm

```
void function(void) { }
void test(void)
{
    asm("
        call   !_function
    ");
}
```

**Example 4. Call Function in the Inline-asm**

#### How to Use Array

```
unsigned char Data[3];
void function(void)
{
    Data[0] = 0xff;
    Data[1] = 0xff;
    Data[2] = 0xff;
}
```

**Example 5. Use of Array**

```
unsigned char Data[3];

void function(void)
{
    asm("
        lda    #0ffh
        sta    !_Data
        sta    !_Data+1
        sta    !_Data+2
    ");
}
```

**Example 6. Use of Array in The Inline-asm**

## How to Use Struct And Write Struct Data to ROM Area

Below example 7 describes how to store struct data in the ROM and shows how to read struct data in the function\_read(). But the function\_write(), which writes data in the ROM area, is a bad example because cook\_data pointer indicates the ROM area. Example 8 describes struct copy function which copies struct data in the ROM to struct variable in the RAM. It will be need in the embedded system which has small RAM size. And example 9 describes use of struct in the inline-asm.

When you use array of struct and a variable which indicate a location of array, it will increase the ROM size remarkably. So, if you reduce a ROM size, you would better change C to ASM like example 10. It will reduce a ROM size dramatically. But if the locational variable of array is a constant, ROM size of C is as same as one of ASM.

```
typedef struct
{
    unsigned char model;
    unsigned char power_level[2];
    unsigned char disp_data[2];
}cook;

cook RICE[] __attribute__((section(".text"))) = {
    {1,100,100,4,5},
    {2,100,100,3,4},
    {3,100,100,2,3}
};

cook *cook_data;

cook data_ram;

unsigned char model, power, disp;

void function_read(void)
{
    cook_data = &RICE[1];
    model = cook_data->model;
    power = cook_data->power_level[0];
    disp = cook_data->disp_data[0];
}

// bad example
void function_write(void)
{
    cook_data = &RICE[1];
    cook_data->model = model;
    cook_data->power_level[0] = power;
    cook_data->disp_data = disp;
}
```

**Example 7. Use of Struct**

```
// struct size : 5bytes
void struct_copy(cook data_ram, cook cook_data)
{
    unsigned char i;
    for ( i=0 ; i<5 ; i++ )
    {
        *(data_ram+i) = *(cook_data+i);
    }
}
```

**Example 8. Struct Copy function**

```
void function_read(void)
{
    asm("
        lda    !_cook_data
        sta    !_model
        lda    !_cook_data+1
        sta    !_power
        lda    !_cook_data+3
        sta    !_disp
    ");
}

void function_write(void)
{
    asm("
        lda    !_model
        sta    !_data_ram
        lda    !_power
        sta    !_data_ram+1
        lda    !_disp
        sta    !_data_ram+3
    ");
}
```

**Example 9. Use of Struct in The Inline-asm**

```
unsigned char temp, power;
void function_C(void)
{
    cook_data_pwr_level[temp] = power;
}

void function_A(void)
{
    asm("
        clrc
        lda    !_temp
        adc    #2
        tay
        lda    !_value
        sta    !_cook_data+y
    ");
}
```

**Example 10. Use of Array of Struct**

### How to Flag change to the Inline-asm

Example 11 describes how to use a flag in the inline-asm. The function\_C() shows use of a flag in the C, and the function\_A() shows use of flag in the ASM. If there are a lot of flags, change them to the inline-asm. It will reduce your ROM size remarkably.

```

struct flag0
{
    char f0:1;
    char f1:1;
    char f2:1;
    char f3:1;
} f;

void function_C(void)
{
    f.f0 = 1;
    f.f1 = 1;
    f.f2 = 1;
    f.f3 = 1;
}

void function_A(void)
{
    asm("
        set1    _f.0
        set1    _f.1
        set1    _f.2
        set1    _f.3
    ");
}
    
```

**Example 11. Use of Flag in The Inline-asm**

### Comparison Switch-Statement with If-Statement

If there are a lot of conditions, which condition-statement is better to reduce the code size?. It is depend upon number of conditions. Below the table describes comparison of code size as the number of conditions.

Number of Condition	Better Condition Statement
1	if-else or switch-case
2 ~ 19	if-else
20 ~	switch-case

**Table 1: Comparison of Condition-Statement**

### Define MACRO with the Inline-asm

Example 12 describes how to use macro with the inline-

asm which has some arguments. The argument must be global variable in the inline-asm. And to access the variable, you should attach '\_' in front of each variable. This macro multiplies unsigned short type variable 'A' by unsigned char type variable 'B'. The quotient is stored to unsigned short type 'R' and the remainder is thrown away.

```

#define MUL16_8(A,B,R)
    asm("
        lda    _A+1
        ldy    _B
        mul
        sta    !_R+1
        lda    _A
        ldy    _B
        mul
        sta    !_R
        tya
        clrc
        adc    !_R+1
        sta    !_R+1
    ");
    
```

**Example 12. Use of Flag in The Inline-asm**

*Author: June Choi  
 MCU Application Team  
 e-mail: jungmin1.choi@hynix.com*