Exam a week from Monday

Exam Monday February 20

You will be able to use all the Motorola manuals on the exam
No calculators will be allowed for the exam
Numbers
  - Decimal (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
  - Inherent (INH), Immediate (IMM), Direct (DIR), Extended (EXT), Relative (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
Exam a week from Monday

Exam Monday February 20

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 programs
- Know basic assembler directives – e.g. equ, dc.b, ds.w
- Flow charts
Using DIP switches to get data into the HC12

DIP switches make or break a connection (usually to ground)
Using DIP switches to get data into the HC12

To use DIP switches, connect one end of each switch to a resistor

Connect the other end of the resistor to +5V

Connect the junction of the DIP switch and the resistor to an input port on the HC12

When the switch is open, the input port sees a logic 1 (+5V)

When the switch is closed, the input sees a logic 0 (0V)
Looking at the state of a few input pins

Want to look for a particular pattern on 4 input pins

- For example want to do something if pattern on PB3-PB0 is 0110

Don’t know or care what are on the other 4 pins (PB7-PB4)

Here is the wrong way to do it:

\[
\begin{align*}
\text{ldaa} & \quad \text{PORTB} \\
\text{cmpa} & \quad \#b0110 \\
\text{beq} & \quad \text{task}
\end{align*}
\]

If PB7-PB4 are anything other than 0000, you will not execute the task.

You need to mask out the Don’t Care bits before checking for the pattern on the bits you are interested in

\[
\begin{align*}
\text{ldaa} & \quad \text{PORTB} \\
\text{anda} & \quad \#b00001111 \\
\text{cmpa} & \quad \#b00000110 \\
\text{beq} & \quad \text{task}
\end{align*}
\]

Now, whatever pattern appears on PB7-4 is ignored
Using an HC12 output port to control an LED

Connect an output port from the HC12 to an LED.

Using an output port to control an LED

When a current flows through an LED, it emits light

Resistor, LED, and ground connected internally inside breadboard
Making a pattern on a 7-segment LED

Want to make a particular pattern on a 7-segment LED.

Determine a number (hex or binary) that will generate each element of the pattern

- For example, to display a 0, turn on segments a, b, c, d, e, and f, or bits 0, 1, 2, 3, 4, and 5 of PTH. The binary pattern is 00111111, or $3f$

- To display 0, 2, 4, 6, 8, the hex numbers are $3f$, $5b$, $66$, $7d$, $7f$.

Put the numbers in a table

Go through the table one by one to display the pattern

When you get to the last element repeat the loop
Flow chart to display the patterns on LEDs

Start

Port A Output

Point to First entry

Get entry

Output to PORT A

Inc pointer

L1:

ldaa #$ff
staa DDRA

L2:

ldx #table

ldaa 0,x

staa PORTA

inx

cpx #end_table
bls L2
bra L1

X < end

Table:

| X | 0x3f | 0x5b | 0x66 | 0x7d | 0x7f |
Program to display the patterns on LEDs

; Program to display patterns

prog:   equ $1000
data:   equ $2000
stack:  equ $3C00
PORTA:  equ $0000
DDRA:   equ $0002

org prog
data: table: table_end: org prog
stack: dc.b #3f
data: dc.b #5b

lds #stack
da #ff
staa DDRA

L1: idx #table
L2: ldaa 1,x+
staa PORTA
jsr delay
cpx #table_end
bls L2
bra L1
Subroutine “delay”

; Subroutine to wait for 100 ms

delay:
    psha          ; 2 cycles
    pshx          ; 2 cycles
    ldaa          ; 1 cycle
    #250
    loop2:
        ldx        ; 2 cycles
        #3200
    loop1:
        dbne       ; 3 cycles
        x,loop1
        dbne       ; 3 cycles
        a,loop2
        pulx       ; 3 cycles
        pula       ; 3 cycles
        rts         ; 5 cycles

Inner loop takes 3 cycles; is executed 3200 times

Outer loop takes (2+3X+3) cycles; is executed 250 times

Total number of cycles: 2+2+1+250*(2+3*3200+3)+3+3+5 = 2,401,266 cycles

This takes 100 ms with a 24 MHz clock
Programming the HC12 in C

Typical C program

- **#header**
  - Preprocessor instructions (include another files)

- **main()**
  - "main()" is always the first function called

- **Function a()**
  - Functions are the building blocks of C

- **Function b()**
  - Declarations
  - Assignments
  - Functions

Advantages of C over other languages:

- Natural for the users to use top-down planning.
- Structured programming.
- Modular design.
Programming the HC12 in C

A comparison of some assembly language and C constructs

<table>
<thead>
<tr>
<th>Assembly</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>; Use a name instead of a num</td>
<td>/* Use a name instead of a num*/</td>
</tr>
<tr>
<td>COUNT: EQ 5</td>
<td>#define COUNT 5</td>
</tr>
<tr>
<td>; Start a program</td>
<td>/* To start a program */</td>
</tr>
<tr>
<td>org $1000</td>
<td>main()</td>
</tr>
<tr>
<td>lds #$3c00</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Note that in C, the starting location of the program is defined when you compile the program, not in the program itself.
Programming the HC12 in C

Note that C always uses the stack, so C automatically loads the stack pointer for you.

<table>
<thead>
<tr>
<th>Assembly</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>; Allocate 2 bytes for a signed number</td>
<td>/* Allocate 2 bytes for a signed number*/</td>
</tr>
<tr>
<td>org $2000</td>
<td>int i;</td>
</tr>
<tr>
<td>i: ds.w 1</td>
<td>int j = 0x1a00;</td>
</tr>
<tr>
<td>j: dc.w $1a00</td>
<td>unsigned int i;</td>
</tr>
<tr>
<td>; Allocate 2 bytes for an unsigned number</td>
<td>/* Allocate 2 bytes for an unsigned number*/</td>
</tr>
<tr>
<td>i: ds.w 1</td>
<td>unsigned int i;</td>
</tr>
<tr>
<td>j: dc.w $1a00</td>
<td>unsigned int j = 0x1a00;</td>
</tr>
</tbody>
</table>
Programming the HC12 in C

---

<table>
<thead>
<tr>
<th>Assembly</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>; Allocate 1 byte for a signed number</td>
<td>/* Allocate 1 byte for a signed number*/</td>
</tr>
<tr>
<td>i:</td>
<td>signed char i;</td>
</tr>
<tr>
<td>ds.b</td>
<td>signed char j = 0x1f;</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>; Get a value from an address and put of contents</td>
<td>/* Get value form an address and put*/</td>
</tr>
<tr>
<td>; of address $E000 into variable i</td>
<td>/* contents of address 0xe000 into i */</td>
</tr>
<tr>
<td>i:</td>
<td>unsigned char i;</td>
</tr>
<tr>
<td>ds.b</td>
<td>i = *(unsigned char *) 0xE000;</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ldaa $E000</td>
<td></td>
</tr>
<tr>
<td>staa i</td>
<td></td>
</tr>
<tr>
<td>*/</td>
<td></td>
</tr>
<tr>
<td>/* Use a variable as a pointer</td>
<td></td>
</tr>
<tr>
<td>unsigned char *ptr, i;</td>
<td></td>
</tr>
<tr>
<td>ptr = (unsigned char *) 0xE000;</td>
<td></td>
</tr>
<tr>
<td>i = *ptr;</td>
<td></td>
</tr>
<tr>
<td>*ptr = 0x55</td>
<td></td>
</tr>
</tbody>
</table>

In C, the construct *(num) says to treat num as an address, and to work with the contents of that address.
Programming the HC12 in C

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 2 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 address 0xE000.
- `*(int *) 0xE000 = 0xaa;` tells C to write the number 0x00aa to 2 bytes starting at address 0xE000.
Programming the HC12 in C

<table>
<thead>
<tr>
<th>Assembly</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>; To call a subroutine</td>
<td>/* To call a function */</td>
</tr>
<tr>
<td>ldab i</td>
<td>sqrt(i);</td>
</tr>
<tr>
<td>jsr sqrt</td>
<td></td>
</tr>
</tbody>
</table>

; To return from a subroutine
<table>
<thead>
<tr>
<th>Assembly</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldab j</td>
<td>return j;</td>
</tr>
<tr>
<td>rts</td>
<td></td>
</tr>
</tbody>
</table>

; Flow control
<table>
<thead>
<tr>
<th>Assembly</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>blo</td>
<td>/* Flow control */</td>
</tr>
<tr>
<td>blt</td>
<td>if (i &lt; j)</td>
</tr>
<tr>
<td>bhs</td>
<td>if (i &gt;= j)</td>
</tr>
<tr>
<td>bge</td>
<td></td>
</tr>
</tbody>
</table>

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

```assembly
org $2000  
i: ds.b 1  
org $1000  
lds #$3c00  
ldaa #16  
jsr div  
staa i  
swi  
div: asra  
rts

signed char i;

signed char div (signed char j);

main()  
{  
i=div(16);  
}  

signed char div (signed char j)  
{  
return j >> 1;  
}  
```
Here is a simple C program

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

Details of compiling of 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. From the File -> New Source File option. Type in your C program. Then from the File -> Save unit save your file in an appropriate directory.
2. 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 form 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:
   - iports from 0000, length 400
   - eeprom from 400, length c00
   - data from 1000, length 1000
   - text from 2000, length 2000
   - stack at 3c00
A simple program in C and how to compile it

Then click **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.
A simple program in C and how to compile it

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.
A simple program in C and how to compile it

If the name of the project is Project.prj, the compiler will generate a file Project.dmp. Here is some of the output from The 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.

```
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>
```
A simple program in C and how to compile it

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 3ce 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
A simple program in C and how to compile it

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  

00002041 <_etext>:  
  2041:  a7   nop
Pointers in C

To access a memory locations:

*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 = 0xaa; /* Put 0xaa into address */
ptr = ptr+2; /* Point two further into table */
x = *ptr; /* Read form address 0x2002 */
Pointers in C

To set aside ten locations for a table:

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

Can access the third element in the table as:

```c
table[2];
```

or as

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

To set up a table of constant data:

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

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

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 the beginning of the header file iodp256.h. You can find the complete file on the EE 308 homepage. Here are a few entries from the header file:

/* IO DEFINITIONS FOR MCS912DP256
 * Copyright (c) 2000 by COSMIC Software */
#ifndef _BASE
#define _BASE 0
#endif
#define _IO(x) @(_BASE)+(x)
#if _BASE == 0
#define _PORT @dir
#else #define _PORT
#endif
#define uint unsigned int

/* MEBI Module */
_PORT volatile char PORTA _IO(0x00); /* port A */
_PORT volatile char PORTB _IO(0x01); /* port B */
_PORT volatile char DDRA _IO(0x02); /* data direction port A */
_PORT volatile char DDRB _IO(0x03); /* data direction port B */
"Interface problems"

"Susan! ...are you trying to tell me we have an interface problem?"