- **An Example of Using the Stack**
- **Introduction to Programming the MC9S12 in C**
    - An example of using the stack
    - Including hcs12.inc in assembly language programs
    - Using a mask in assembly language programs
    - Using the DIP switches on the Dragon12
    - Putting a program into the MC9S12 EEPROM
    - Displaying patterns from a table on the Dragon12 LEDs
    - Comparison of C and Assembly language programs

Example Using the Stack; why the following program would not work properly?

Consider the following:

```
            org   $2000
            lds   #$2000
            ldx   #$0123
            ldd   #$abcd
            pshx
            psha
            pshb
            bsr   delay
            pulb
            pula
            pulx
            swi

delay:      pshx
            ldx    #1000
loop:       dbne  x,loop
            rts
```

Using Registers in Assembly Language

- The DP256 version of the MC9S12 has lots of hardware registers

- To use a register, you can use something like the following:

PORTB: equ $0001

- It is not practical to memorize the addresses of all the registers

- Better practice: Use a file which has all the register names with their addresses

#include "hcs12.inc"

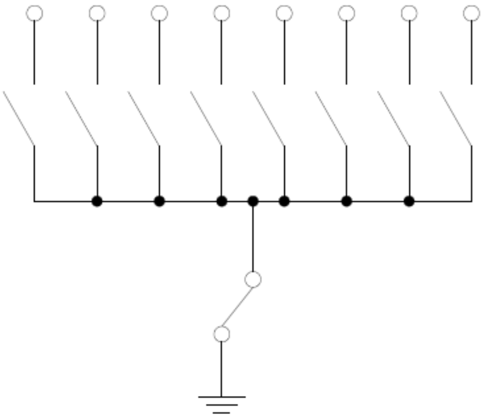
- Here is some of hcs12.inc

```
;
***********************************************************************
*
; Prepared by Dr. Han-Way Huang
; Date: 12/31/2004
; HC12SDP256 I/O register locations
; HCS12 peripheral bits definitions
; D-Bug12 I/O functions calling address
; D-Bug12 SRAM interrupt vector table
; Flash and EEPROM commands
;
***********************************************************************
*

PORTA           equ     0       ; port a = address lines a8 - a15
PTA             equ     0       ; alternate name for PORTA
PORTB           equ     1       ; port b = address lines a0 - a7
PTB             equ     1       ; alternate name for PORTB
DDRA            equ     2       ; port a direction register
DDRB            equ     3       ; port a direction register

....
```

Using DIP switches to get data into the MC9S12

- DIP switches make or break a connection (usually to ground)

DIP Switches on Breadboard



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

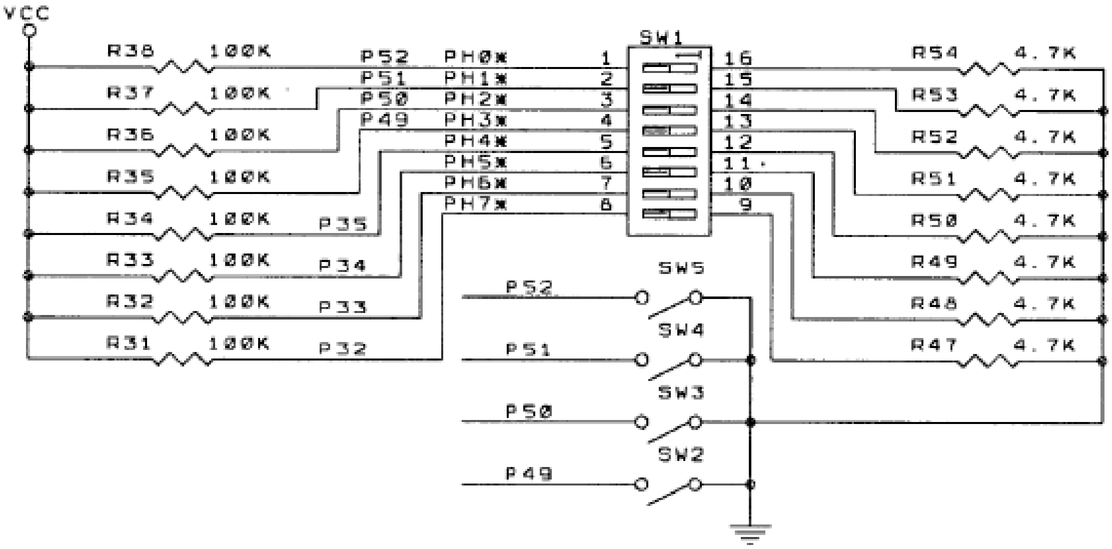
- Connect the other end of the resistor to +5 V

- Connect the junction of the DIP switch and the resistor to an input port on the MC9S12

- The Dragon12-Plus has eight dip switches connected to Port H (PTH)

- The four least significant bits of PTH are also connected to push-button switches.

- If you want to use the push-button switches, make sure the first 4 DIP switches are in the OFF position.



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

• When the switch is closed, the input sees a logic 0 (0.22 V)

Putting a program into EEPROM on the Dragon12-Plus

- EEPROM from 0x400 to 0xFFF

- Program will stay in EEPROM memory even after power cycle

  – Data will not stay in RAM memory (!)

- If you put the above program into EEPROM, then cycle power, you will display a sequence of patterns on the seven-segment LED, but the pattern will be whatever junk happens to be in RAM.

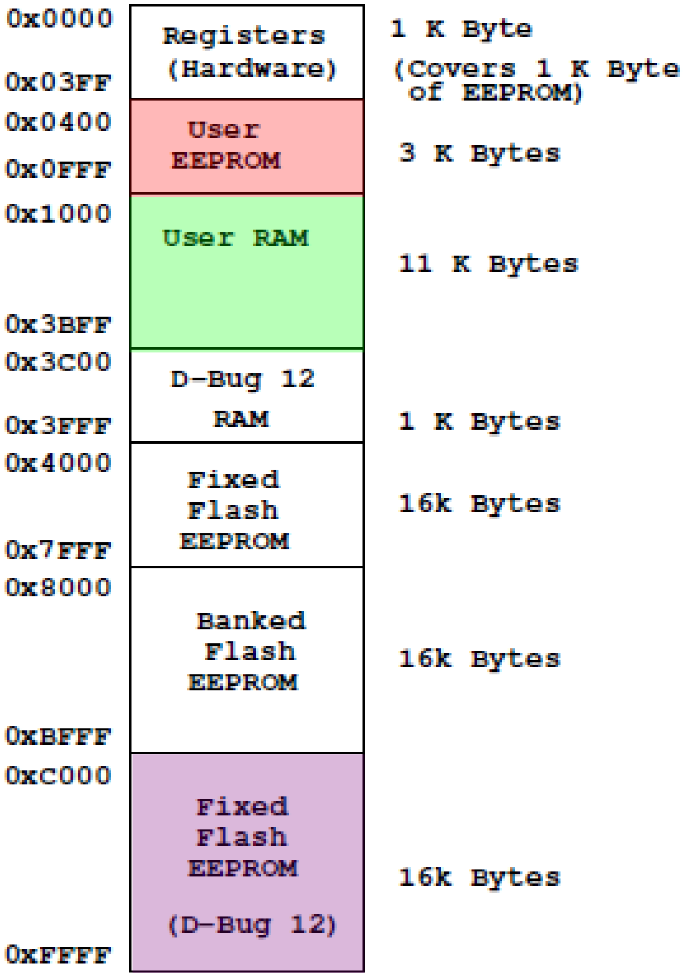
- To make sure you retain your patterns, put the table in the text part of your program, not the data part.

- If you use a variable which needs to be stored in data, be sure you initialize that variable in your program and not by using dc.b.

- The Dragon12 board uses an 8 MHz clock. The MC9S12 has an internal phase-locked loop which can change the clock speed. DBug12 increases the clock speed from 8 MHz to 48 MHz.

- When you run a program from EEPROM, DBug12 does not run, so your program will run six times slower that it would using DBug12. The lab has instructions on how to increase the MC9S12 clock from 8 MHz to 48 MHz so your program will run with the same speed as under DBug12.

MC9S12 Address Space



0x0000	0x03FF	Registers (Hardware)	1 K Byte (Covers 1 K Byte of EEPROM)
0x0400	0x0FFF	User EEPROM	3 K Bytes
0x1000	0x3BFF	User RAM	11 K Bytes
0x3C00	0x3FFF	D-Bug 12 RAM	1 K Bytes
0x4000	0x7FFF	Fixed Flash EEPROM	16k Bytes
0x8000	0xBFFF	Banked Flash EEPROM	16k Bytes
0xC000	0xFFFF	Fixed Flash EEPROM (D-Bug 12)	16k Bytes

• Here is the above program with table put into EEPROM

• Also, we have included a variable *var* which we initialize to $aa in the program

- We don't use var in the program, but included it to show you how to use a RAM-based variable

```
#include "hcs12.inc"
prog:           equ     $0400
data:           equ     $1000
stack:          equ     $2000
table_len:      equ      (table_end-table)

                org     prog
                lds     #stack          ; initialize stack pointer
                movb    #$aa,var        ; initialize var
                ldaa    #$ff            ; Make PORTB output
                staa    DDRB            ; 0xFF -> DDRB
l1:             ldx      #table         ; Start pointer at table
l2:             ldaa    0,x             ; Get value
                staa    PORTB           ; Update LEDs
                bsr     delay           ; Wait a bit
                inx                     ; point to next
                cpx     #table_end      ; More to do?
                blo     l2              ; Yes, keep going through table
                bra     l1              ; At end; reset pointer

delay:          psha
                pshx
                ldaa    #100
loop2:          ldx     #8000
loop1:          dbne    x,loop1
                dbne    a,loop2
                pulx
                pula
                rts
```

```
table:          dc.b $3f
                dc.b $5b
                dc.b $66
                dc.b $7d
                dc.b $7F
table_end:

                org data
var:            ds.b 1                  ; Reserve one byte for var
```

Programming the MC9S12 in C

- A comparison of some assembly language and C constructs

Assembly	C
; Use a name instead of a num	/* Use a name instead of a num */
COUNT: EQU 5	#define COUNT 5
;---	/*-----------------------------*/
;start a program	/* To start a program */
org $1000	main()
lds #$3C00	{
	}
;---	/*-----------------------------*/

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

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

Assembly	C
; allocate two bytes for	/* Allocate two bytes for
; a signed number	* a signed number */
org $2000	
i: ds.w 1	int i;
j: dc.w $1A00	int j = 0x1a00;

Assembly

C

```
;-----------------------------------------
; allocate two bytes for
; an unsigned number

i:        ds.w   1
j:        dc.w   $1A00
```

```
/*-----------------------------*/
/* Allocate two bytes for
 * an unsigned number */

 unsigned int i;
 unsigned int j = 0x1a00;
```

```
; allocate one byte for
; a signed number

i:        ds.b   1
j:        dc.b   $1F
```

```
/* Allocate one byte for */
/* a signed number */

signed char i;
signed char j = 0x1f;
```

```
;------------------------------------------
; Get a value from an address
; Put contents of address
; $E000 into variable i

i:        ds.b   1

         ldaa     $E000
         staa     i
```

```
/*-----------------------------*/
/* Get a value from an address */
/* Put contents of address */
/* 0xE000 into variable i */

unsigned char i;

i = * (unsigned char *) 0xE000;
```

```
/*-----------------------------------*/
/* Use a variable as a pointer
(address) */

unsigned char *ptr, i;

ptr = (unsigned char *) 0xE000;
i = *ptr;
```

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

• 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 two 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 addess 0xE000.

- * (int *) 0xE000 = 0xaa; tells C to write the number 0x00aa to two bytes starting at address 0xE000.

Assembly	C
;---------------------------------	/*-----------------------------*/
; To call a subroutine	/* To call a function */
ldaa i	sqrt(i);
jsr sqrt	
;--------------------------------	/*-----------------------------*/
; To return from a subroutine	/* To return from a function */
ldaa j	return j;
rts	
;--------------------------------	/*-----------------------------*/
; Flow control	/* Flow control */
blo	if (i < j)
blt	if (i < j)
bhs	if (i >= j)
bge	if (i >= j)
;---------------------------------	/*-----------------------------*/

- 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     $1000
i:      ds.b    1


        org     $2000
        lds     #$3C00
        ldaa    #16
        jsr     div
        staa    i
        swi

div:    asra
        rts
```

C

```
unsigned char i;


unsigned char div(unsigned char j);
main()
 {
        i = div(16);
 }


 unsigned char div(unsigned char j)
 {
        return j >> 1;
 }
```