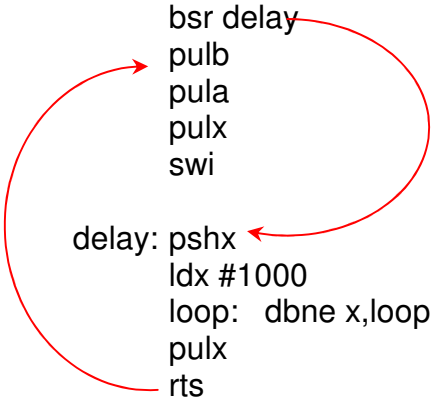
- **An Example of Using the Stack**
  - An example of using the stack
  - Including hcs12.inc in assembly language programs
  - Using a mask in assembly language programs
  - Using a subroutine with PORTA to make a binary counter on LEDs

Examples of Using the Stack

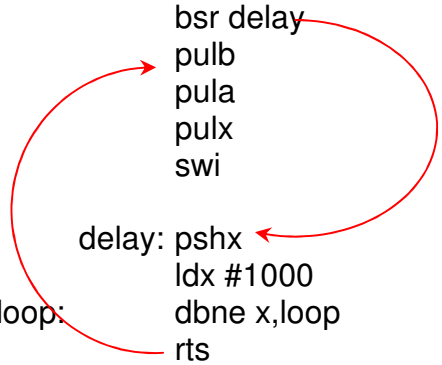
Consider the following:

```
2000                    org $2000
2000 cf 20 00           lds #$2000
2003 ce 01 23           ldx #$0123
2006 cc ab cd           ldd #$abcd
2009 34                 pshx
200a 36                 psha
200b 37                 pshb
200c 07 04              bsr delay
200e 33                 pulb
200f 32                 pula
2010 30                 pulx
2011 3f                 swi

2012 34           delay: pshx
2013 ce 03 e8           ldx #1000
2016 04 35 fd           loop:   dbne x,loop
2019 30                 pulx
201a 3d                 rts
```



The following does not work; the RTS goes to the wrong place

```
2000                        org $2000
2000 cf 20 00               lds #$2000
2003 ce 01 23               ldx #$0123
2006 cc ab cd               ldd #$abcd
2009 34                     pshx
200a 36                     psha
200b 37                     pshb
200c 07 04                  bsr delay
200e 33                     pulb
200f 32                     pula
2010 30                     pulx
2011 3f                     swi

2012 34               delay: pshx
2013 ce 03 e8               ldx #1000
2016 04 35 fd loop:         dbne x,loop
2019 3d                     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 "derivative.inc"

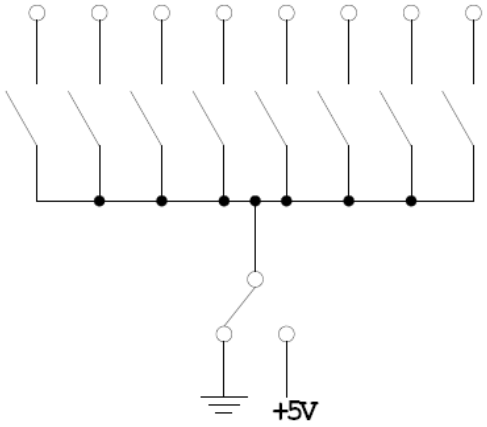
• Here is some of derivative.inc

```
;*** PORTA - Port A Register; 0x00000000 ***
PORTA: equ $0000 ;*** PORTA - Port A Register; 0x0000 ***
;*** PORTB - Port B Register; 0x0001 ***
PORTB: equ $0001 ;*** PORTB - Port B Register; 0x0001 ***
;*** DDRA - Port A Data Direction Register; 0x0002 ***
DDRA: equ $0002 ;*** DDRA - Port A Data Direction Register; 0x0002 ***
;*** DDRB - Port B Data Direction Register; 0x0003 ***
DDRB: equ $0003 ;*** DDRB - Port B Data Direction Register; 0x0003 ***
```

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

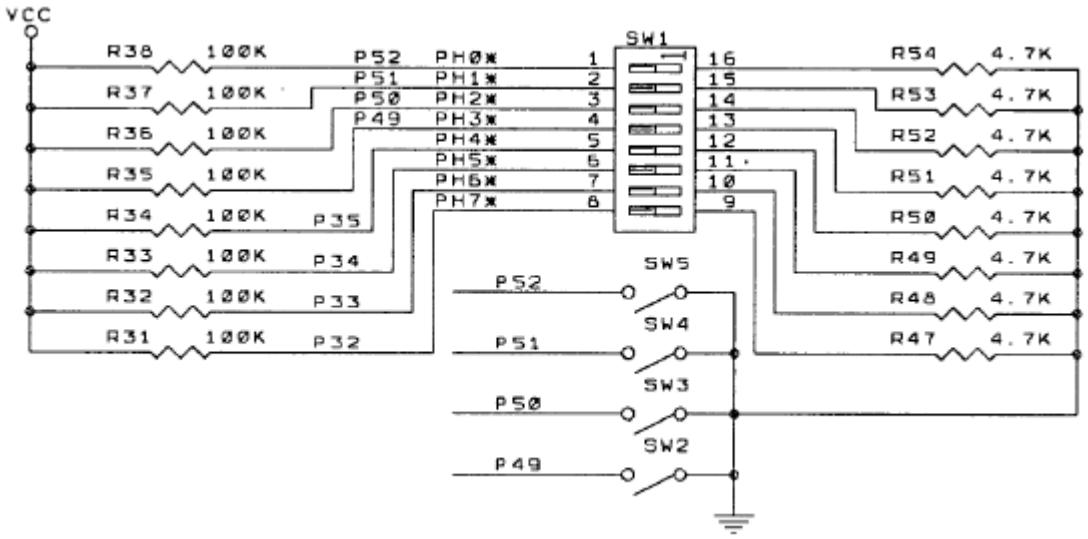
• Connect the other end of the resistor to +5 V

• 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 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)

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 PH3-PH0 is 0110

• Don't know or care what are on the other 4 pins (PH7-PH4)

• Here is the wrong way to do it:

```
ldaa PTH
cmpa #$06
beq task
```

• If PH7-PH4 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

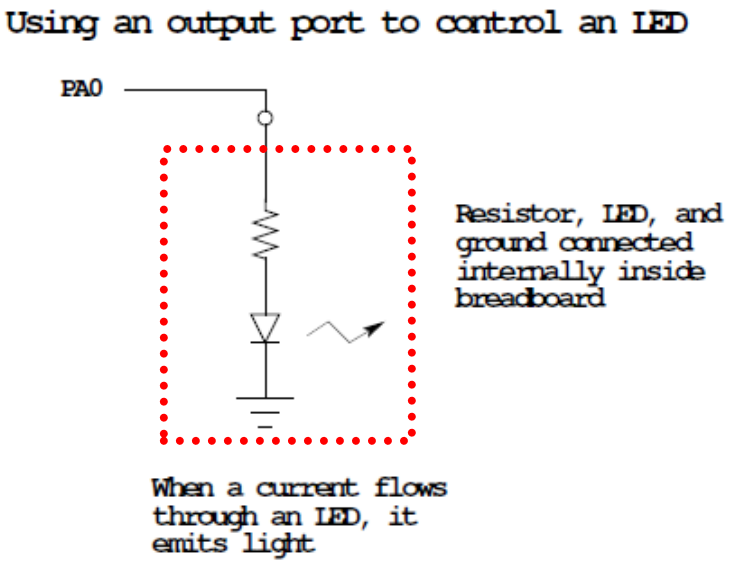
– To mask out don't care bits, AND the bits with a mask which has 0's in the don't
care bits and 1's in the bits you want to look at.

```
ldaa PTH
anda #$0F
cmpa #$06
beq task
```

• Now, whatever pattern appears on PH7-4 is ignored

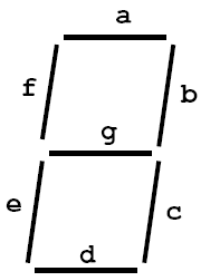
Using an HC12 output port to control an LED

• Connect an output port from the HC12 to an LED.



Making a pattern on a seven-segment LED

• Want to generate a particular pattern on a seven-segment LED:



• Determine a number (hex or binary) which 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 0011 1111, or $3f.

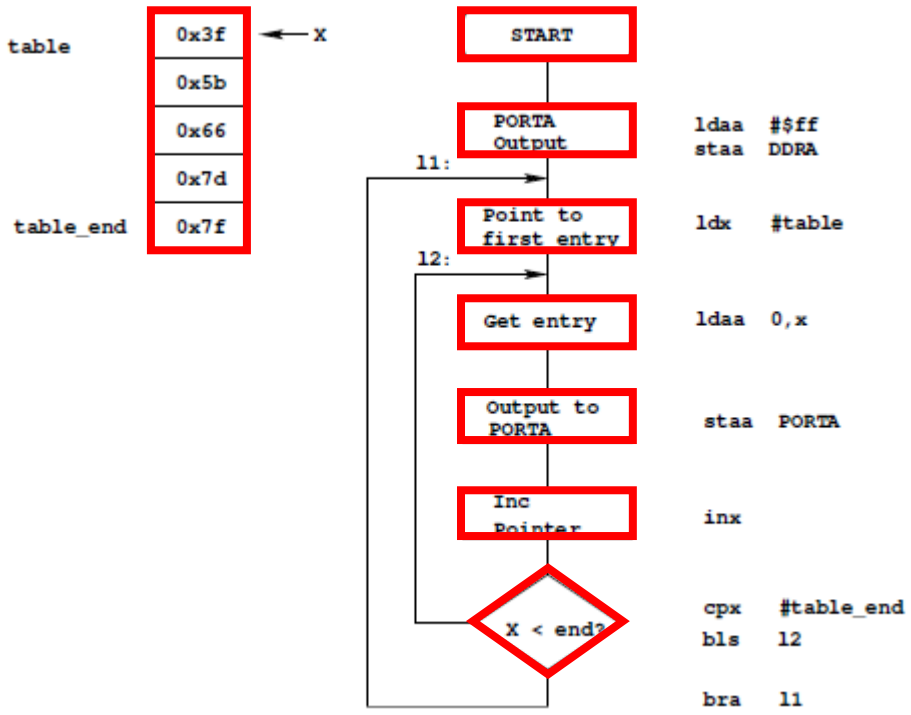
– To display 0 2 4 6 8, the hex numbers are $3f, $5b, $66, $7d, $7f.

• 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

Flowchart to display a pattern of lights on a set of LEDs



as12, an absolute assembler for Motorola MCU's, version 1.2h

```
; Program to display a pattern on a seven-segment LED display

                                #include "hcs12.inc"
2000                            prog: equ $2000
1000                            data: equ $1000
2000                            stack: equ $2000

0005            table_len:      equ (table_end-table)

2000                            org prog
2000 cf 20 00                       lds #stack      ; initialize stack pointer
2003 86 ff                      ldaa #$ff       ; Make PORTB output
2005 5a 03                      staa DDRB       ; 0xFF -> DDRB
2007 ce 10 00                   l1:     ldx #table      ; Start pointer at table
200a a6 00                  l2:     ldaa 0,x        ; Get value
200c 5a 01                      staa PORTB  ; Update LEDs
200e 07 08                      bsr delay       ; Wait a bit
2010 08                         inx             ; point to next
2011 8e 10 05                       cpx #table_end ; More to do?
2014 25 f4                      blo l2          ; Yes, keep going through table
2016 20 ef                      bra l1          ; At end; reset pointer
2018 36                     delay: psha
2019 34                         pshx
201a 86 64                      ldaa #100
201c ce 1f 40                   loop2:          ldx #8000
201f 04 35 fd                   loop1:          dbne x,loop1
2022 04 30 f7                       dbne a,loop2
2025 30                         pulx
2026 32                         pula
2027 3d                         rts

1000                            org data
1000 3f                     table:  dc.b $3f
1001 5b                         dc.b $5b
1002 66                         dc.b $66
1003 7d                         dc.b $7d
1004 7f                         dc.b $7F
1005            table_end:
```

Putting a program into EEPROM on the Dragon12-Plus

• EEPROM from 0x400 to 0xFFF

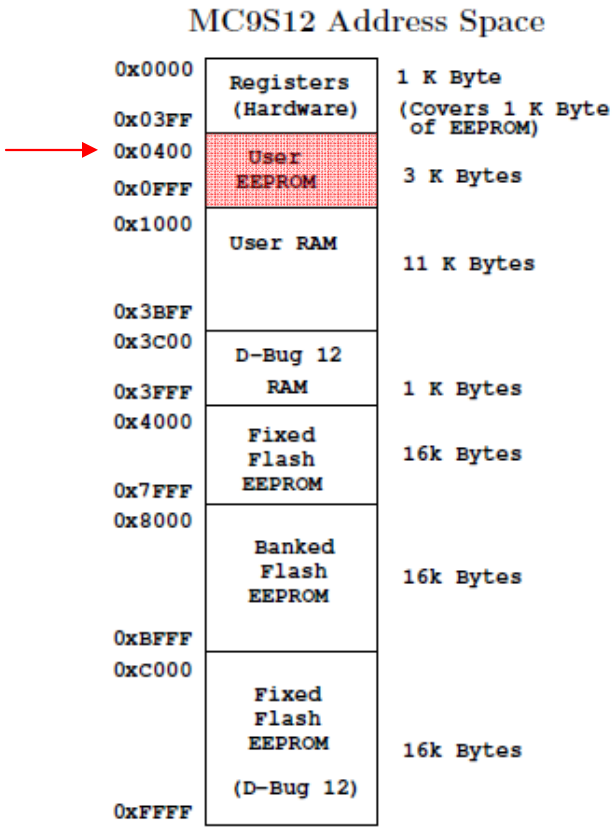
• Program will stay in EEPROM memory even after power cycle

– 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 you 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.



• Here is the above program with table put into EEPROM

• Also, I have included a variable var which I initialize to $aa in the program
– I don't use var in the program, but included it to show you how to use a RAMbased 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
            moveb #$aa,var      ; initialize var
            ldaa #$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
```