THE STACK AND THE STACK POINTER

- Sometimes it is useful to have a region of memory for temporary storage, which does not have to be allocated as named variables.
- When we use subroutines and interrupts it will be essential to have such a storage region.
- Such a region is called a *Stack*.
- The *Stack Pointer* (`SP`) register is used to indicate the location of the last item put onto the stack.
- When you put something onto the stack (push onto the stack), the `SP` is decremented *before* the item is placed on the stack.
- When you take something off of the stack (pull from the stack), the `SP` is incremented *after* the item is pulled from the stack.
- Before you can use a stack you have to initialize the Stack Pointer to point to one value higher than the highest memory location in the stack.
- For the HC12 use a block of memory from about `$3B00` to `$3BFF` for the stack.
- For this region of memory, initialize the stack pointer to `$3C00`.
- Use the `LDS` (Load Stack Pointer) instruction to initialize the stack point.
- The `LDS` instruction is usually the first instruction of a program which uses the stack.
- The stack pointer is initialized only one time in the program.
- For microcontrollers such as the HC12, it is up to the programmer to know how much stack his/her program will need, and to make sure enough space is allocated for the stack. If not enough space is allocated the stack can overwrite data and/or code, which will cause the program to malfunction or crash.

The stack is an array of memory dedicated to temporary storage

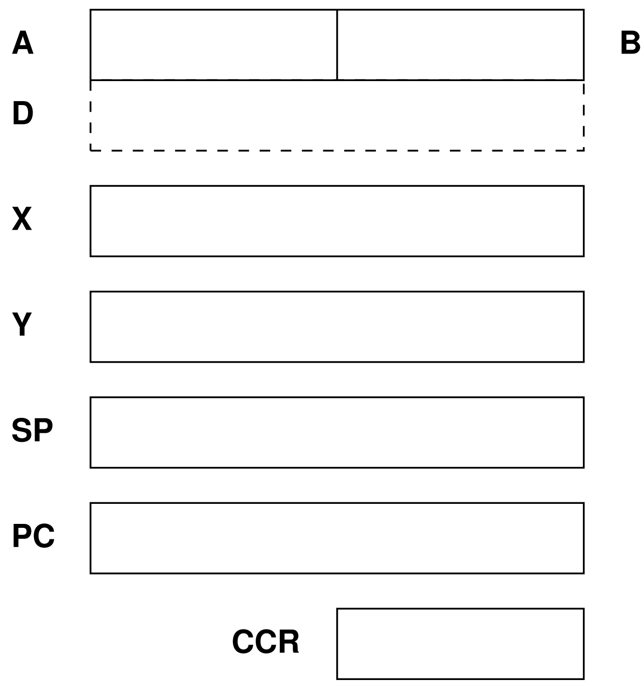
SP points to location last item placed in block

SP decreases when you put item on stack

SP increases when you pull item from stack

For HC12 EVBU, use 0x3C00 as initial SP:

```
STACK:   EQU   $3C00
         LDS   #STACK
```



An example of some code which uses the stack

Stack Pointer:

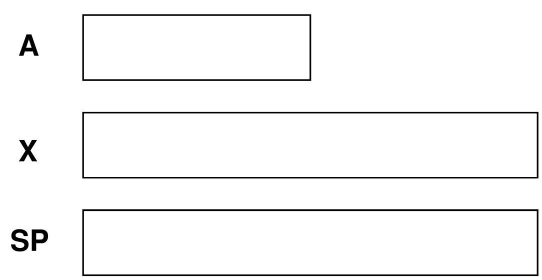
Initialize ONCE before first use (LDS #STACK)

Points to last used storage location

Decreases when you put something on stack

Increases when you take something off stack


```
STACK:  equ   $3C00

CODE:     section  .text
          org 0x1000

          lds      #STACK
          ldaa     #$2e
          ldx      #$1254
          psha
          pshx
          clra
          ldx       #$ffff

          CODE THAT USES A & X

          pulx
          pula
```



PSHA Push A onto Stack PSHA

Operation (SP) – \$0001 ⇒ SP
(A) ⇒ M_{SP}

Decrements SP by one and loads the value in A into the address to which SP points.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

CCR Effects

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Code and CPU Cycles

Source Form	Address Mode	Machine Code (Hex)	CPU Cycles
PSHA	INH	36	Os

Subroutines

- A subroutine is a section of code which performs a specific task, usually a task which needs to be executed by different parts of a program.
- Example:
  - Math functions, such as *square root*
- Because a subroutine can be called from different places in a program, you cannot get out of a subroutine with an instruction such as

```
jmp     label
```

  because you would need to jump to different places depending upon which section of code called the subroutine.
- When you want to call the subroutine your code has to save the address where the subroutine should return to. It does this by saving the *return address* on the stack.
  - This is done automatically for you when you get to the subroutine by using the `JSR` (Jump to Subroutine) or `BSR` (Branch to Subroutine) instruction. This instruction pushes the address of the instruction following the `JSR (BSR)` instruction on the stack.
- After the subroutine is done executing its code it needs to return to the address saved on the stack.
  - This is done automatically for you when you return from the subroutine by using the `RTS` (Return from Subroutine) instruction. This instruction pulls the return address off of the stack and loads it into the program counter, so the program resumes execution of the program with the instruction following that which called the subroutine.

  The subroutine will probably need to use some HC12 registers to do its work. However, the calling code may be using its registers for some reason — the calling code may not work correctly if the subroutine changes the values of the HC12 registers.

- To avoid this problem, the subroutine should save the HC12 registers before it uses them, and restore the HC12 registers after it is done with them.

BSR Branch to Subroutine BSR

Operation

$(SP) - \$0002 \Rightarrow SP$
$RTN_H:RTN_L \Rightarrow M_{SP}:M_{SP+1}$
$(PC) + \$0002 + rel \Rightarrow PC$

Sets up conditions to return to normal program flow, then transfers control to a subroutine. Uses the address of the instruction after the BSR as a return address.

Decrements the SP by two, to allow the two bytes of the return address to be stacked.

Stacks the return address (the SP points to the high byte of the return address).

Branches to a location determined by the branch offset.

Subroutines are normally terminated with an RTS instruction, which restores the return address from the stack.

CCR Effects

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Code and CPU Cycles

Source Form	Address Mode	Machine Code (Hex)	CPU Cycles
BSR *rel8*	REL	07 rr	SPPP

RTS Return from Subroutine RTS

Operation

$(M_{SP}):(M_{SP+1}) \Rightarrow PC_H:PC_L$
$(SP) + \$0002 \Rightarrow SP$

Restores the value of PC from the stack and increments SP by two. Program execution continues at the address restored from the stack.

CCR Effects

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Code and CPU Cycles

Source Form	Address Mode	Machine Code (Hex)	CPU Cycles
RTS	INH	3D	UfPPP

Example of a subroutine to delay for a certain amount of time

```
; Subroutine to wait for 100 ms
```

```
delay:  ldaa    #250
loop2:  ldx     #800
loop1:  dex
        bne     loop1
        deca
        bne     loop2
        rts
```

- Problem: The subroutine changes the values of registers `A` and `X`

- To solve, save the values of `A` and `X` on the stack before using them, and restore them before returning.

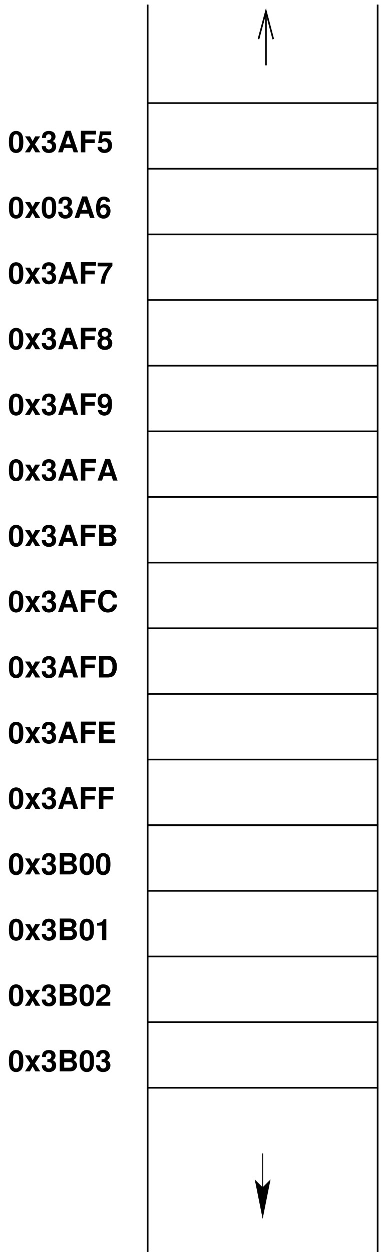
```
; Subroutine to wait for 100 ms

delay:  psha                ; Save regs used by sub on stack
        pshx
        ldaa    #250
loop2:  ldx     #800
loop1:  dex
        bne     loop1
        deca
        bne     loop2
        pulx                ; Restore regs in opposite
        pula                ; order
        rts
```

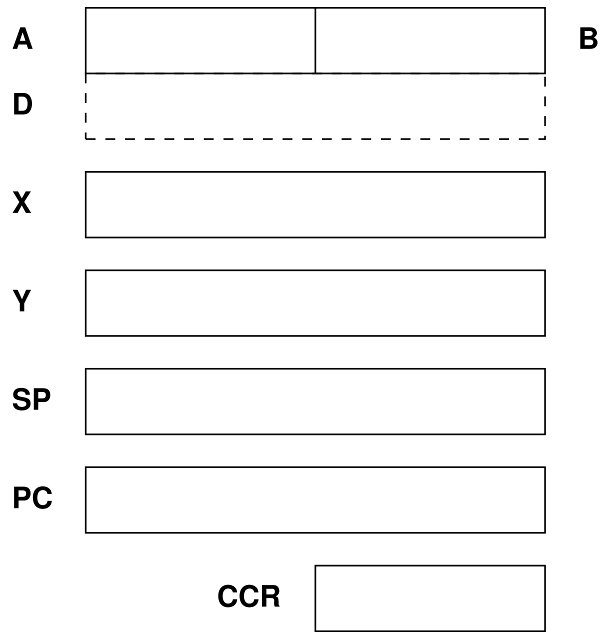
```
; Program to make a binary counter on LEDs
;
; The program uses a subroutine to insert a delay
; between counts

prog:   equ     $1000
STACK:  equ     $3C00       ;Stack ends of $3BFF
PORTA:  equ     $0000
PORTB:  equ     $0001
DDRA:   equ     $0002
DDRB:   equ     $0003


        org     prog

        lds     #STACK      ; initialize stack pointer
        ldaa    #$ff        ; put all ones into DDRA
        staa    DDRA        ; to make PORTA output
        clr     PORTA       ; put $00 into PORTA
loop:   jsr     delay       ; wait a bit
        inc     PORTA       ; add one to PORTA
        bra     loop        ; repeat forever


; Subroutine to wait for 100 ms

delay:  psha
        pshx
        ldaa    #250
loop2:  ldx     #800
loop1:  dex
        bne     loop1
        deca
        bne     loop2
        pulx
        pula
        rts
```

JSR and BSR place return address on stack

RTS returns to instruction after JSR or BSR

```
                     STACK:   EQU   $3C00
                              ORG   $1000
1000 CF 3C 00                 LDS   #STACK
1003 16 10 07                 JSR   MY_SUB
1006 7F                       SWI
1007 CE 12 34        MY_SUB:  LDX   #$1234
100A 3D                       RTS
```





Another example of using a subroutine

Using a subroutine to wait for an event to occur, then take an action.

- Wait until bit 7 of address `$00C4` is set.
- Write the value in ACCA to address `$00C7`.

```
; This routine waits until the HC12 serial
; port is ready, then sends a byte of data
; to the HC12 serial port

putchar:      brclr    $00CC,#$80,putchar
              staa     $00CF
              rts
```

- Program to send the word `hello` to the HC12 serial port

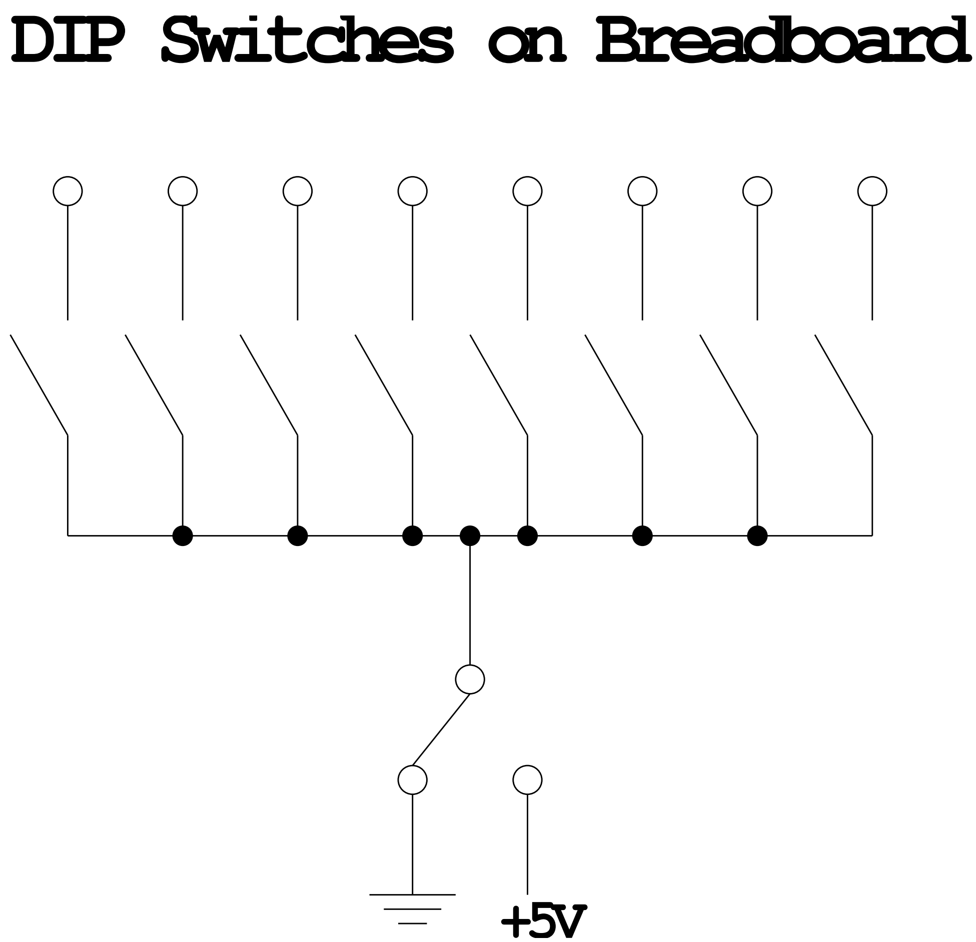
```
; Program fragment to write the word "hello" to the
; HC12 serial port

             ldx       $str
loop:        ldaa      1,x+     ; get next char
             beq       done     ; char == 0 => no more
             jsr       putchar
             bra       loop
             swi

str:         dc.b      "hello"
             fc.b      $0A,$0D,0     ; CR LF
```

Here is the complete program to write a line to the screen:

```
prog:    equ     $1000
data:    equ     $2000
stack:   equ     $3c00

         org     prog
         lds     #stack
         ldx     #str
loop:    ldaa    1,x+ ; get next char
         beq     done ; char == 0 => no more
         jsr     putchar
         bra     loop
done:    swi

putchar: brclr   $00CC,$80,putchar
         staa    $00CF
         rts

         org     data
str:     fcc     "hello"
         dc.b    $0a,$0d,0 ; CR LF
```

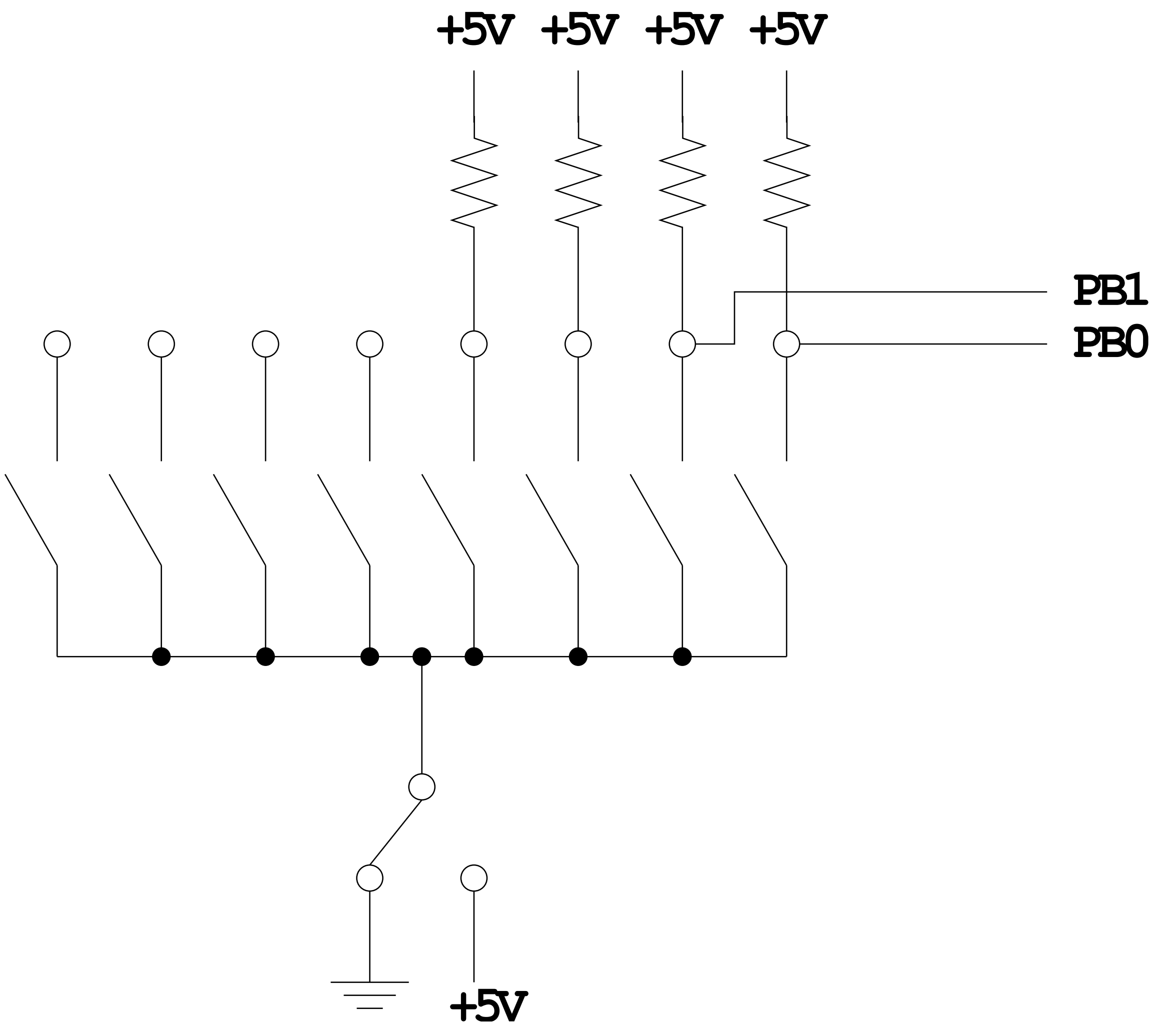
Using DIP switches to get data into the HC12

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



- 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 HC12

Using DIP Switches



- 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 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 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:

```
ldaa   PORTB
cmpa   #b0110
beq    task
```

- 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

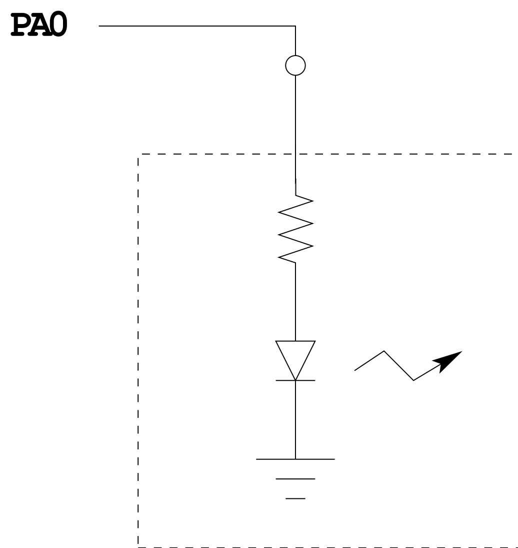
```
ldaa    PORTB
anda    #b00001111
cmpa    #b00000110
beq     task
```

- 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

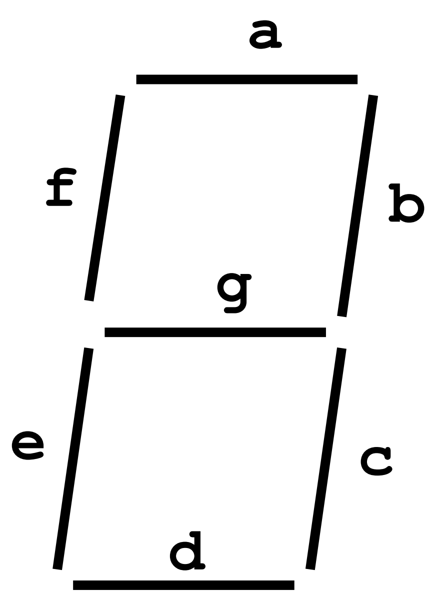


Resistor, LED, and ground connected internally inside breadboard

When a current flows through an LED, it emits light

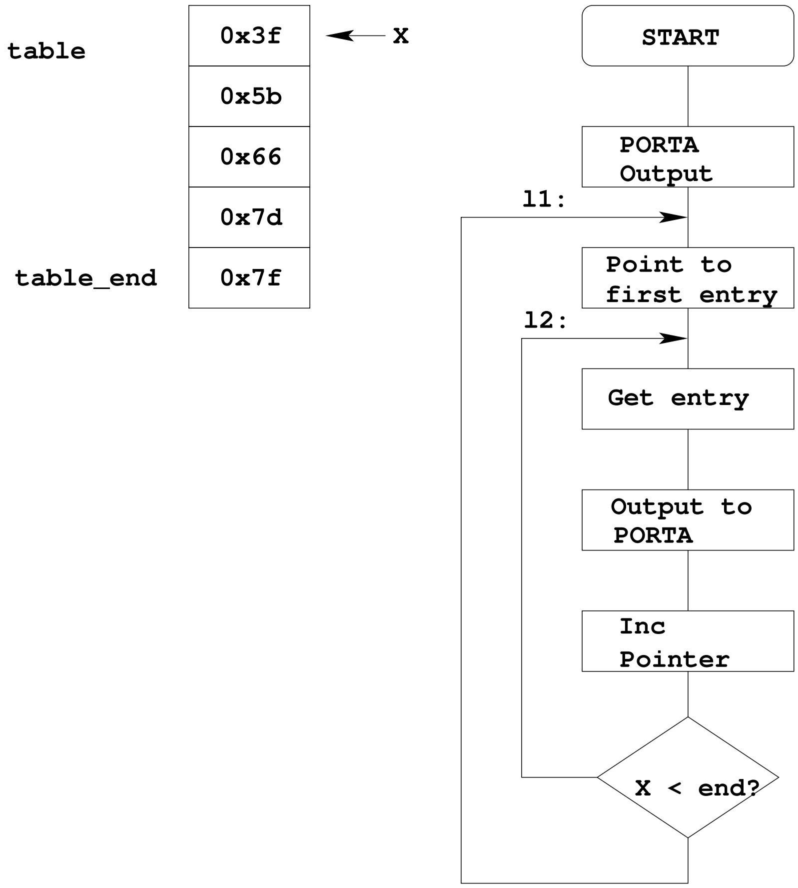
Making a pattern on a seven-segment LED

- 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 `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

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




```
ldaa   #$ff
staa   DDRA

ldx    #table

ldaa   0,x

staa   PORTA

inx

cpx    #table_end
bls    l2

bra    l1
```

```
; Program using subroutine to make a time delay

prog:       equu    $1000
data:       equ     $2000
stack:      equ     $3C00
PTH:        equ     $0260
DDRH:       equ     $0262


            org     prog

            lds     #stack      ; initialize stack pointer
            ldaa    #$ff        ; Make PTH output
            staa    DDRH        ;   0xFF -> DDRH
l1:         ldx     #table      ; Start pointer at table
l2:         ldaa    1,x+        ; Get value; point to next
            staa    PTH         ; Update LEDs
            jsr     delay       ; Wait a bit
            cpx     #table_end  ; More to do?
            bls     l2          ; Yes, keep going through table
            bra     l1          ; At end; reset pointer

delay:      psha
            pshx
            ldaa    #250
loop2:      ldx     #8000
loop1:      dex
            bne     loop1
            deca
            bne     loop2
            pulx
            pula
            rts

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