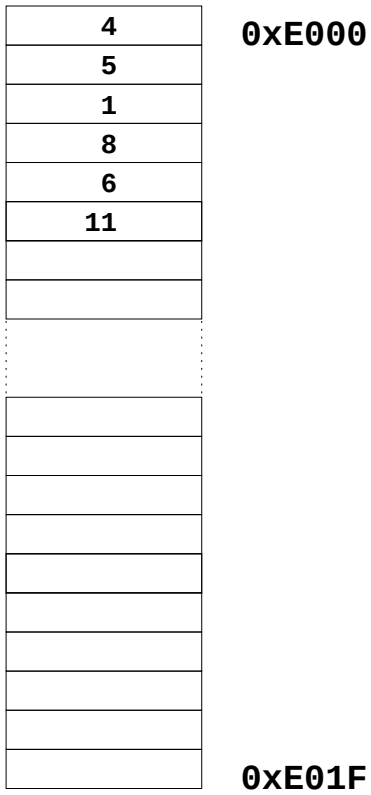
Another Example of an Assembly Language Program

- Add the odd numbers in an array of data.
- The numbers are 8-bit unsigned numbers.
- The address of the first number is `$E000` and the address of the final number is `$E01F`.
- Save the result in a variable called `answer` at address `$0900`.

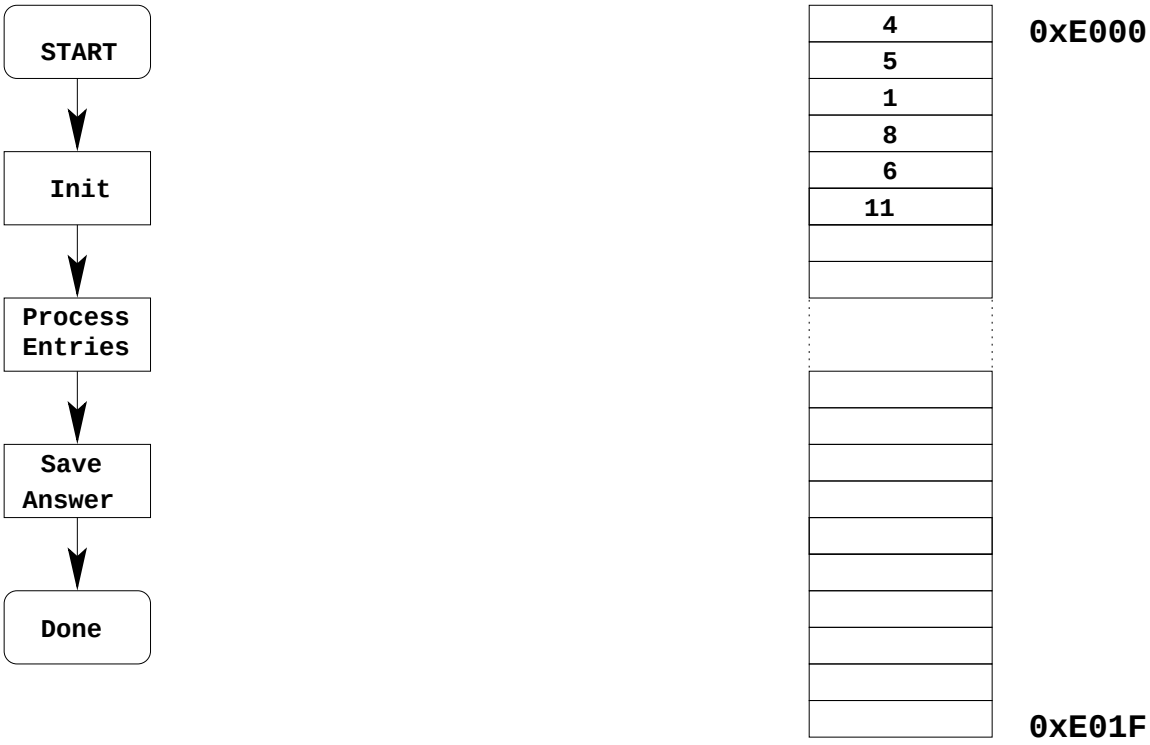
Start by drawing a picture of the data structure in memory:

```
SUM ODD NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f
Treat numbers as 8-bit unsigned numbers
```



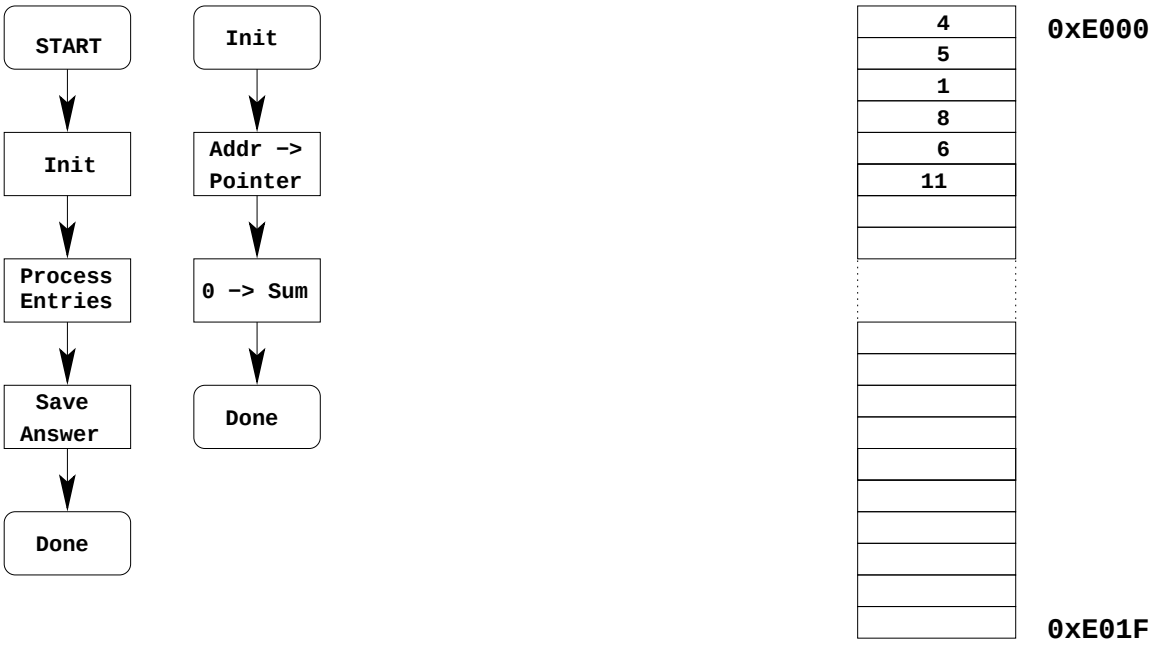
Start with the big picture

SUM ODD 8–BIT NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f



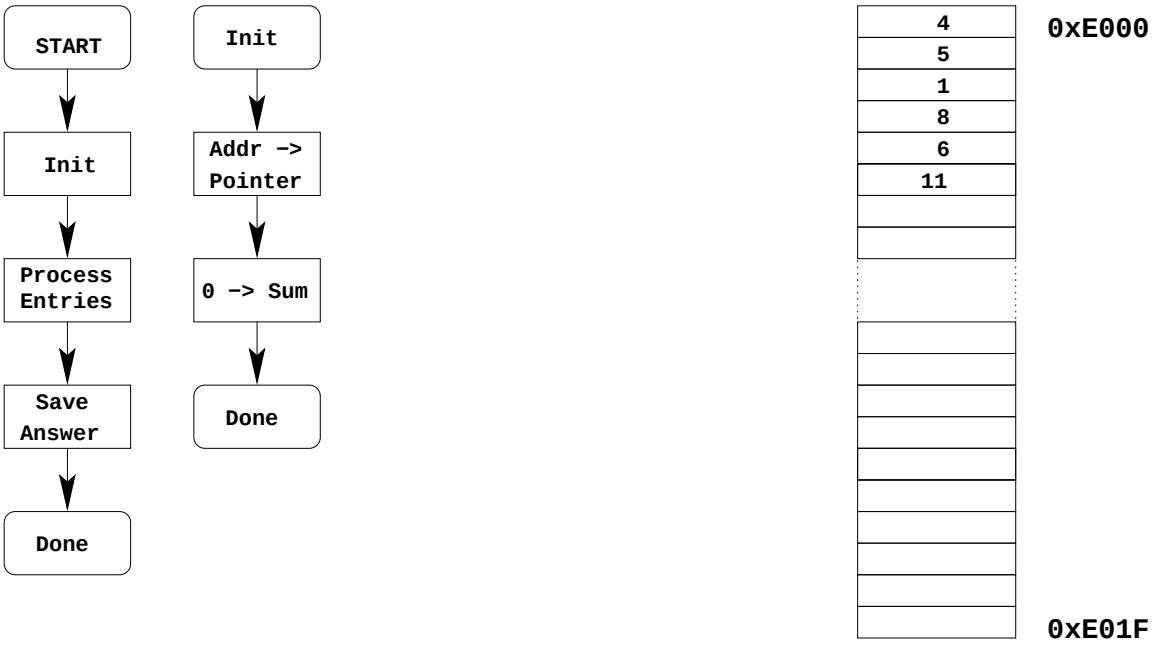
Add details to blocks

SUM ODD 8-BIT NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f



Decide on how to use CPU registers for processing data

SUM ODD 8-BIT NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f



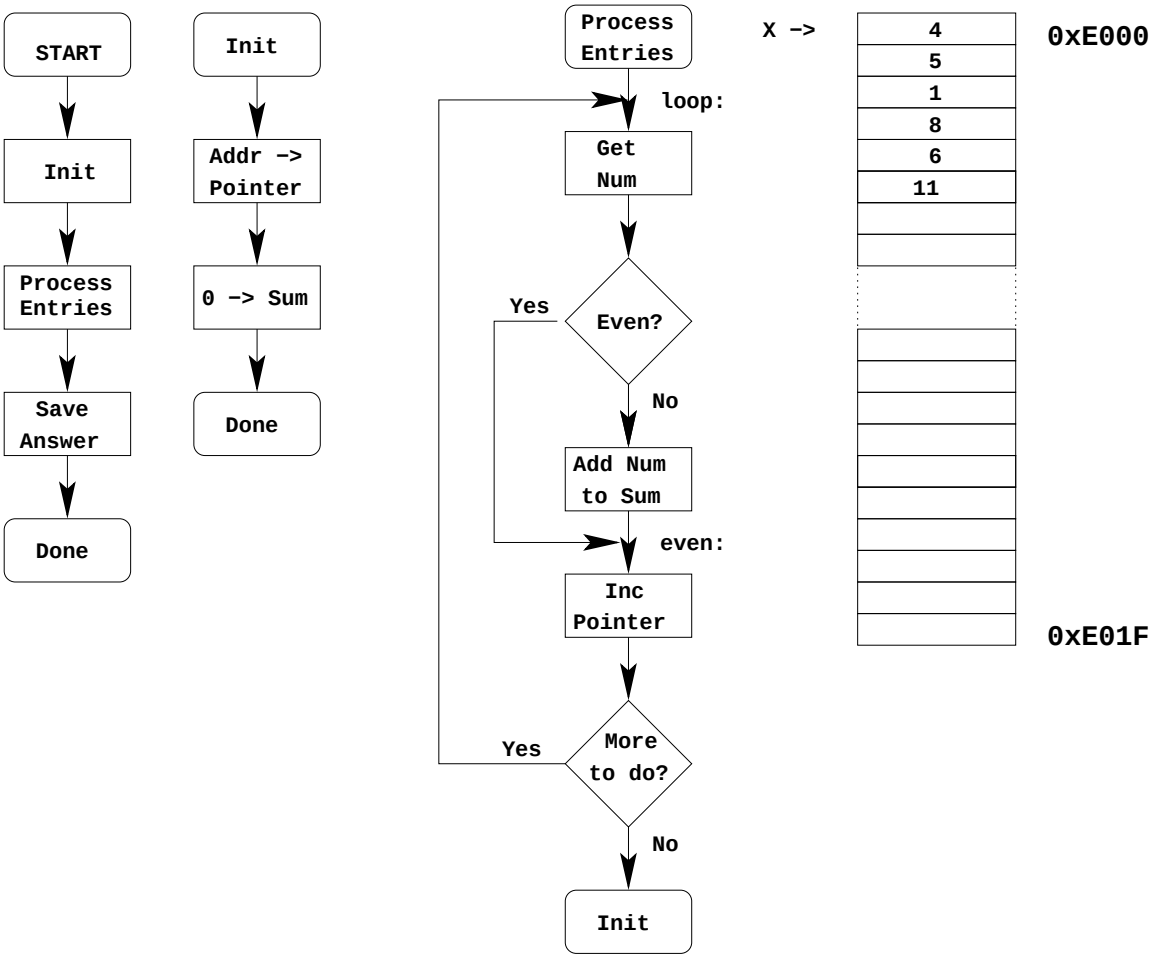
Pointer: X or Y -- use X

Sum: 16-bit register
D or Y

No way to add 8-bit number to D
Can use ABY to add 8-bit number to Y

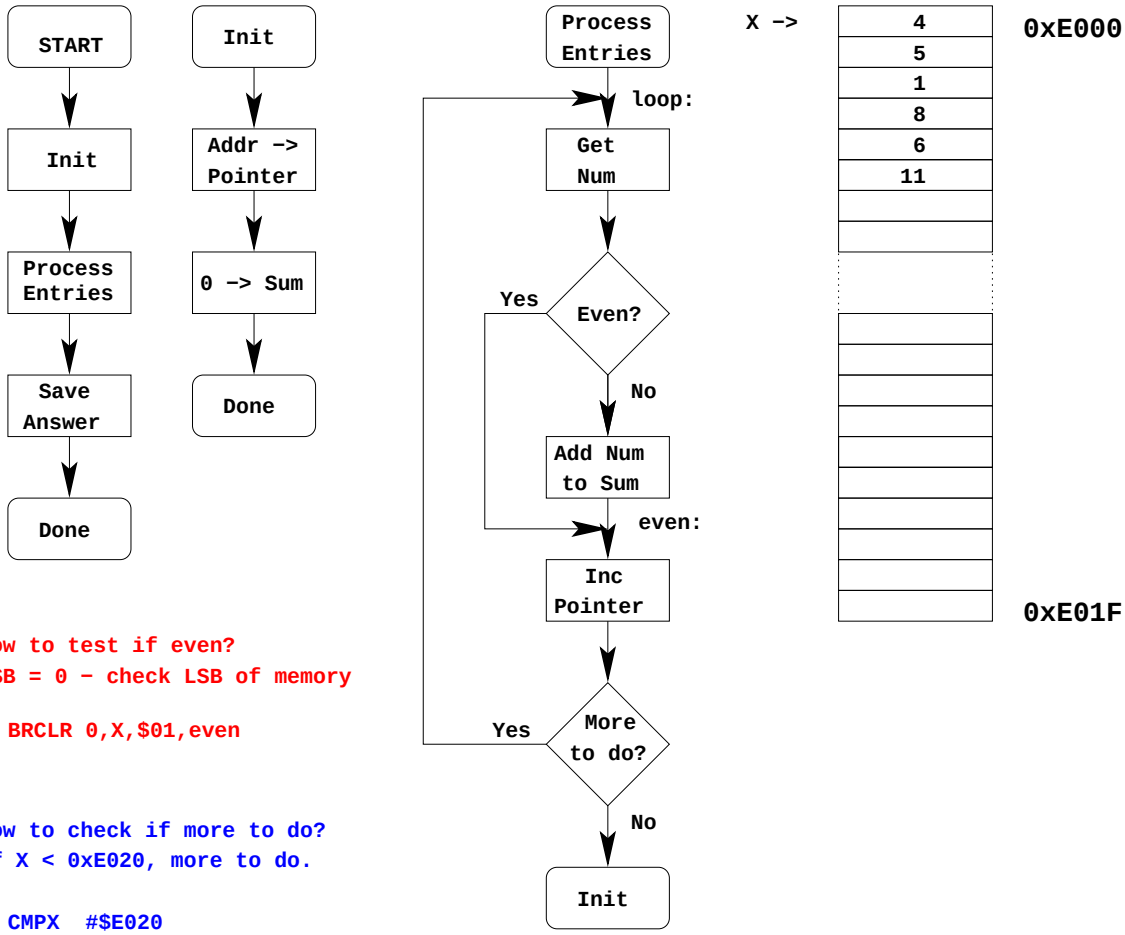
Add more details: Expand another block

SUM ODD 8–BIT NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f



More details: How to tell if number is odd, how to tell when done

SUM ODD 8-BIT NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f



How to test if even?
LSB = 0 – check LSB of memory

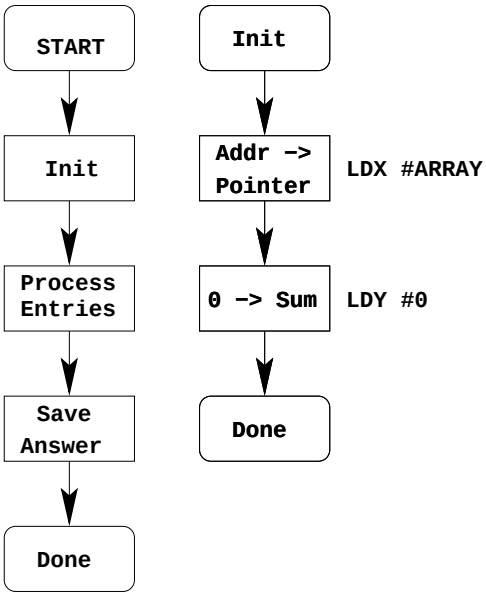
```
BRCLR 0,X,$01,even
```

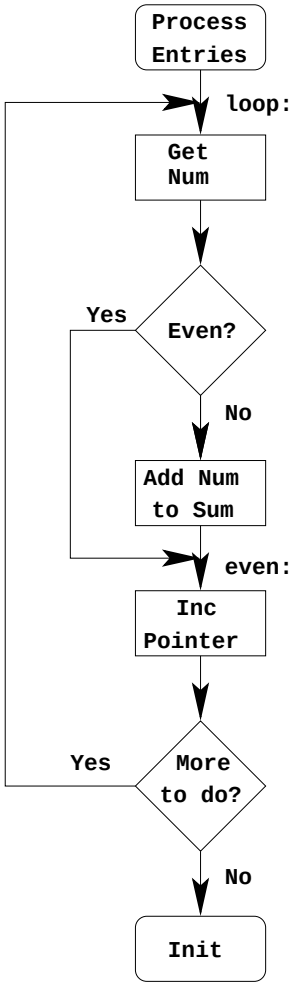
How to check if more to do?
If X < 0xE020, more to do.

```
CMPX  #$E020
BL0 or BLT   loop   ?

Address in unsigned, use unsigned compare
BLO loop
```

Convert blocks to assembly code

SUM ODD 8-BIT NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f



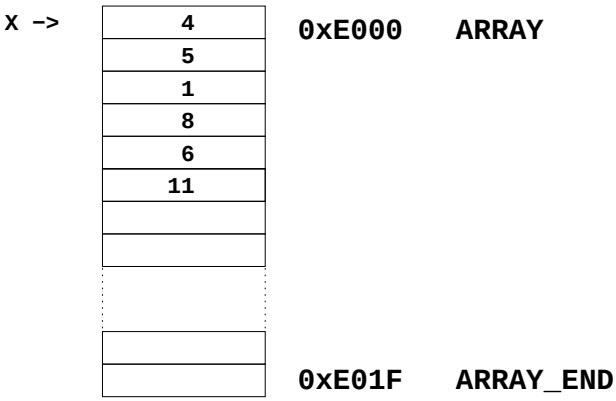


How to test if even?
LSB = 0 – check LSB of memory

```
BRCLR 0,X,$01,even
```

How to check if more to do?
If X < 0xE020, more to do.

```
BLO   loop
```



Write program

```
;Program to sum odd numbers in a memory array

prog:   equ     $0800
data:   equ     $0900

array:  equ     $E000
len:    equ     $20

CODE:   section .text
        org     prog

        ldx     #array                  ; initialize pointer
        ldy     #0                      ; initialize sum to 0
loop:   ldab    0,x                     ; get number
        brclr   0,x,$01,skip            ; skip if even
        aby                             ; odd - add to sum
skip:   inx                             ; point to next entry
        cpx     #(array+len)            ; more to process?
        blo     loop                    ; if so, process
        sty     answer                  ; done -- save answer
        swi

DATA:   section .data
        org     data
answer: ds.w    1                       ; reserve 16-bit word for answer
```

- Important: Comment program so it is easy to understand.

The assembler output for the above program

- Note that the assembler output shows the op codes which the assembler generates for the HC12.
- For example, the op code for `brclr  0,x,$01,skip` is `0f 00 01 02`

```
 1        00000800      prog:   equ     $0800
 2        00000900      data:   equ     $0900
 3
 4        0000e000      array:  equ     $E000
 5        00000020      len:    equ     $20
 6
 7                      CODE:   section .text
 8  0800                        org     prog
 9
10  0800 cee000                 ldx     #array
11  0803 cd0000                 ldy     #0
12  0806 e600           loop:   ldab    0,x
13  0808 0f000102               brclr   0,x,$01,skip
14  080c 19ed                   aby
15  080e 08             skip:   inx
16  080f 8ee020                 cpx     #(array+len)
17  0812 25f2                   blo     loop
18  0814 7d0900                 sty     answer
19  0817 3f                     swi
20
21                      DATA:   section .data
22  0900                        org     data
23  0900 0000           answer: ds.w    1
```

The map file for the above program

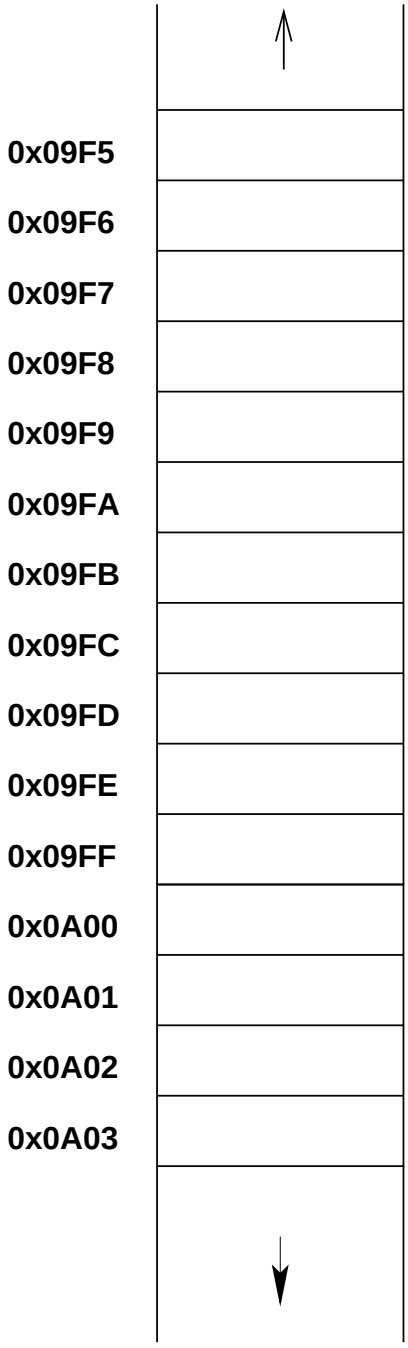
- Note that the map file shows you where your code and data will go, and how much room they will take.
- The value of all names are shown.
- The addresses of all labels are shown.

```
Map of sumodds.h12 from link file sumodds.lkf - Thu Jan 31 21:19:43 2002


Segments:

start 00000800 end 00000800 length     0 segment .text
start 00000900 end 00000900 length     0 segment .data
start 00000900 end 00000902 length     2 segment DATA
start 00000800 end 00000818 length    24 segment CODE
start 00000000 end 0000011e length   286 segment .debug

Modules:

sumodds.o:
start 00000000 end 0000011e length   286 section .debug
start 00000800 end 00000818 length    24 section CODE
start 00000900 end 00000902 length     2 section DATA

.11      00000000
.12      00000008
.13      00000013
answer   00000900
array    0000e000
data     00000900
len      00000020
loop     00000806
prog     00000800
skip     0000080e

Symbols:
```

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 `$09C0` to `$09FF` for the stack.
- For this region of memory, initialize the stack pointer to `$0A00`.
- 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



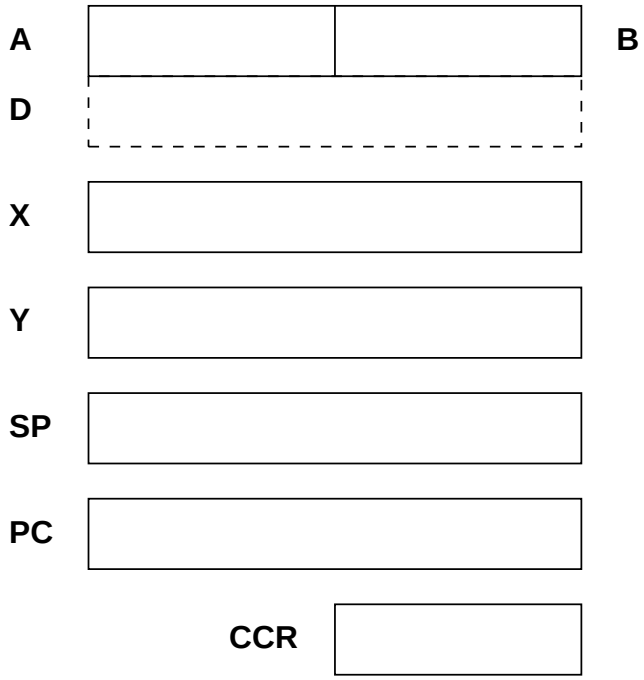
SP points to location last item placed in block

SP decreases when you put item on stack

SP increases when you pull item from stack

For HC12 EVBU, use 0x0A00 as initial SP:

```
STACK:  EQU  $0A00
        LDS  #STACK
```



An example of some code which uses the stack

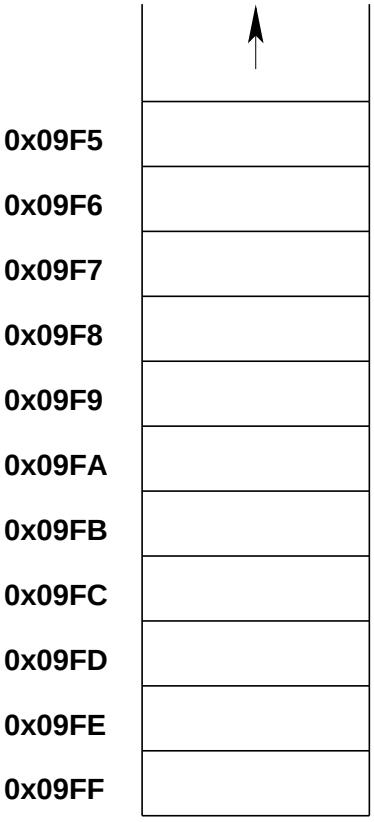
Stack Pointer:

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   $0A00

CODE:   section  .text
        org 0x0800

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

        CODE THAT USES A & X

        pulx
        pula
```