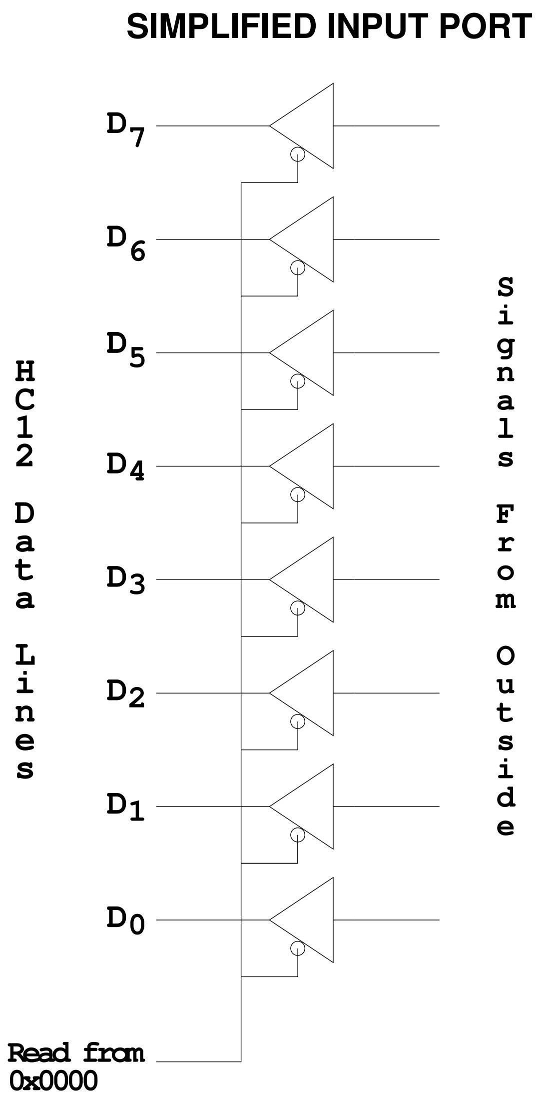
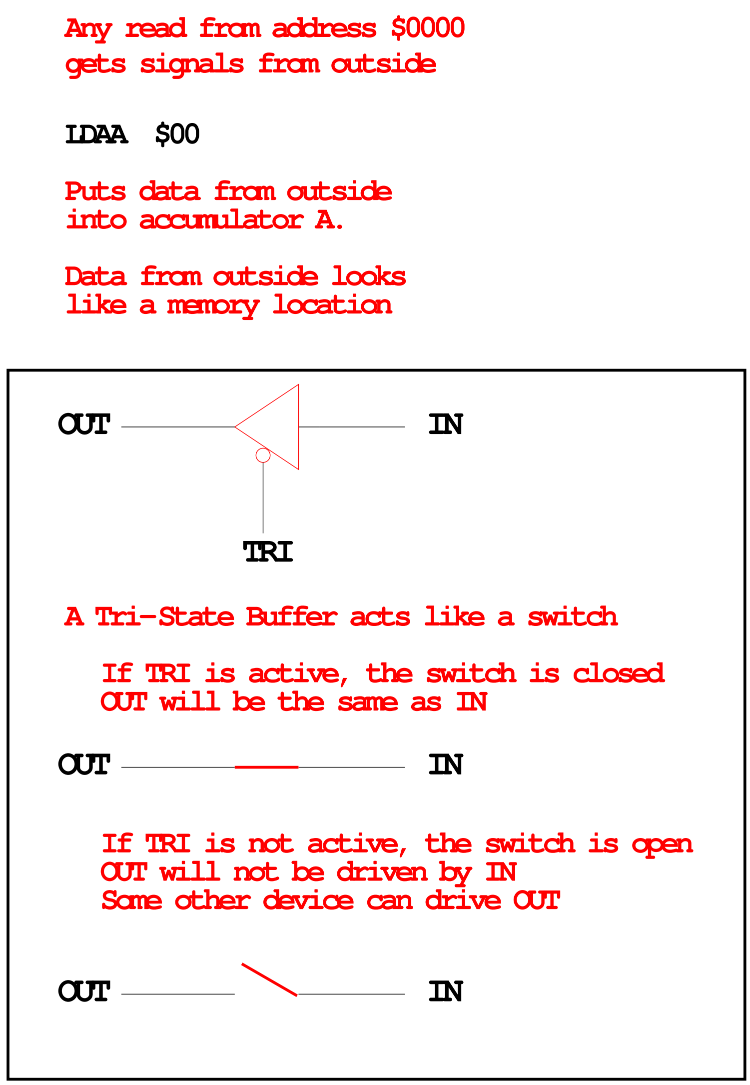
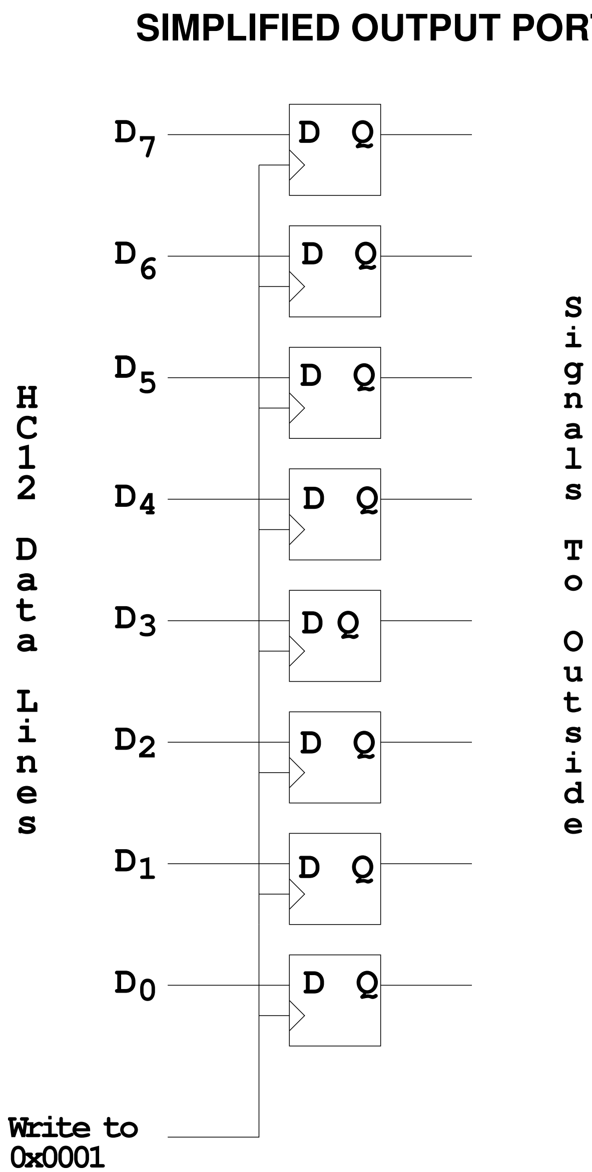
Input and Output Ports

- How do you get data into a computer from the outside?





- How do you get data out of computer to the outside?



Any write to address $01 latches data into flip-flops, so data goes to external pins

```
MOVB #$AA,$01
```

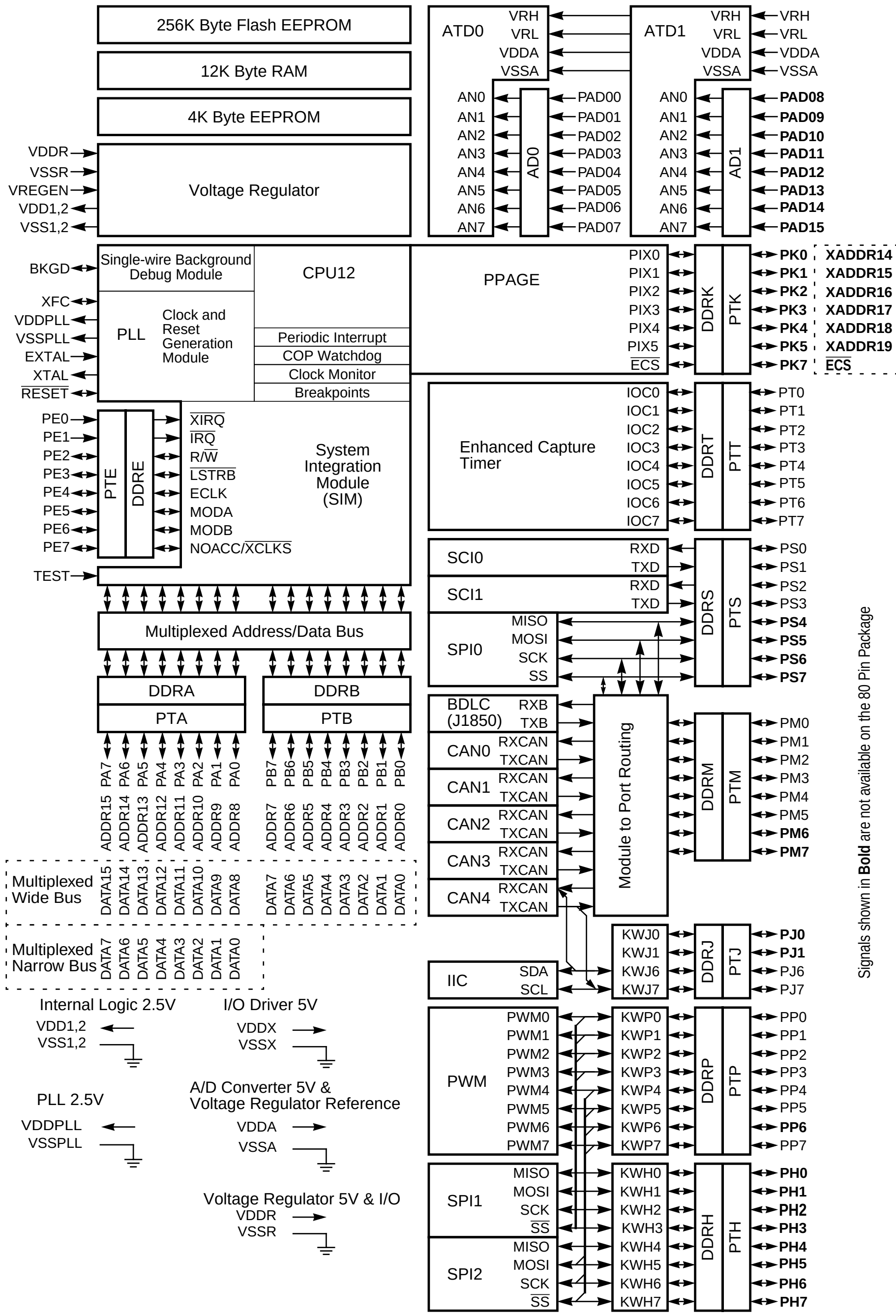
puts $AA on the external pins

When a port is configured as output and you read from that port, the data you read is the data which was written to that port:

```
MOVB #$AA,$01
LDAA   $01
```

Accumulator A will have $AA after this

Figure 1-1 MC9S12DP256B Block Diagram



Ports on the HC12

- How do you get data out of computer to the outside?

- A **Port** on the HC12 is device the HC12 uses to control some hardware.
- Many of the HC12 ports are used to communicate with hardware outside of the HC12.
- The HC12 ports are accessed by the HC12 by reading and writing memory locations `$0000` to `$03FF`.
- Some of the ports we will use in this course are *PORTA*, *PORTB* and `PTH`
- `PORTA` is accessed by reading and writing address `$0000`.
- `PORTB` is accessed by reading and writing address `$0001`.
- `PTH` is accessed by reading and writing address `$0260`.
- You can connect signals from the outside by connecting wires to pins 39 to 46 (`PORTA`), 18 to 25 (`PORTB`), and to pins 32 to 35 and 49 to 52 (`PTH`).
  - On the MiniDRAGON+ EVB, a seven-segment LED is connected to `PTH`.
- When you power up or reset the HC12, `PORTA`, `PORTB` and `PTH` are input ports.
- You can make any or all bits of `PORTA`, `PORTB` and `PTH` outputs by writing a 1 to the corresponding bits of their *Data Direction Registers*.
  - The Data Dirction Register for `PORTA` is located at memory addres `$0002`. It is called `DDRA`. To make all bits of `PORTA` output, write a `$FF` to `DDRA`. To make the lower four bits of `PORTA` output and the upper four bits of `PORTA` input, write a `$0F` to `DDRA`.
  - The Data Dirction Register for `PORTB` is located at memory addres `$0003`. It is called `DDRB`. To make all bits of `PORTB` output, write a `$FF` to `DDRB`.
  - The Data Dirction Register for `PTH` is located at memory addres `$0262`. It is called `DDRH`. To make all bits of `PTH` output, write a `$FF` to `DDRH`.

- You can use DBug-12 to easily manipulate the IO ports on the 68HCS12
  * To make `PTH` an output, use `MM` to change the contents of address `$0262 (DDRH)` to an `$FF`.
  * You can now use `MM` to change contents of address `$0260 (PTH)`, which changes the logic levels on the `PTH` pins.
  * If the data direction register makes the port an input, you can use `MD` to display the values on the external pins.

Using Port A of the 68HC12

To make a bit of Port A an output port, write
a 1 to the corresponding bit of DDRA (address 0x0002).
To make a bit of Port A an input port, write a 0 to
the corresponding bit of DDRA.

On reset, DDRA is set to $00, so Port A is an
input port.

	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	$0002
RESET	0	0	0	0	0	0	0	0	

For example, to make bits 3–0 of Port A input, and
bits 7–4 output, write a 0xf0 to DDRA.
To send data to the output pins, write to
PORTA (address 0x0000). When you read from PORTA
input pins will return the value of the signals on them
(0 => 0V, 1 => 5V); output pins will return the value
written to them.

	PA7	PA6	PA5	PA4	DP3	PA2	PA1	PA0	$0000
RESET	—	—	—	—	—	—	—	—	

Port B works the same, except DDRB is at address 0x0003
and PORTB is at address 0x0001.

```
;A simple program to make PORTA output and PORTB input,
;then read the signals on PORTB and write these values
;out to PORTA

prog:       equ     $1000

PORTA:      equ     $00
PORTB:      equ     $01
DDRA:       equ     $02
DDRB:       equ     $03

            org     prog
            movb    #$ff,DDRA   ; Make PORTA output
            movb    #$00,DDRB   ; Make PORTB input

            ldaa    PORTB
            staa    PORTA
            swi
```

- Because `DDRA` and `DDRB` are in consecutive address locations, you could make `PORTA` and output and `PORTB` and input in one instruction:

```
           movw     #$ff00,DDRA ; FF -> DDRA, 00 -> DDRB
```

GOOD PROGRAMMING STYLE

1. Make programs easy to read and understand.
   - Use comments
   - Do not use tricks
2. Make programs easy to modify
   - Top-down design
   - Structured programming – no spaghetti code
   - Self contained subroutines
3. Keep programs short BUT do not sacrifice items 1 and 2 to do so

TIPS FOR WRITING PROGRAMS

1. Think about how data will be stored in memory.
   - Draw a picture
2. Think about how to process data
   - Draw a flowchart
3. Start with big picture. Break into smaller parts until reduced to individual instructions
   - Top-down design
4. Use names instead of numbers

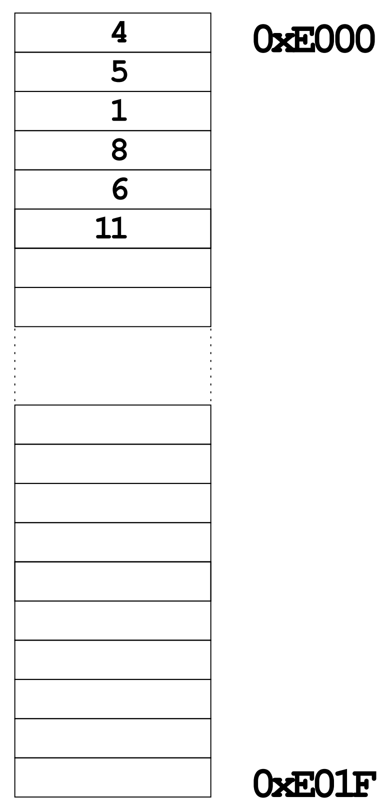
Another Example of an Assembly Language Program

- 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 `$2000`.

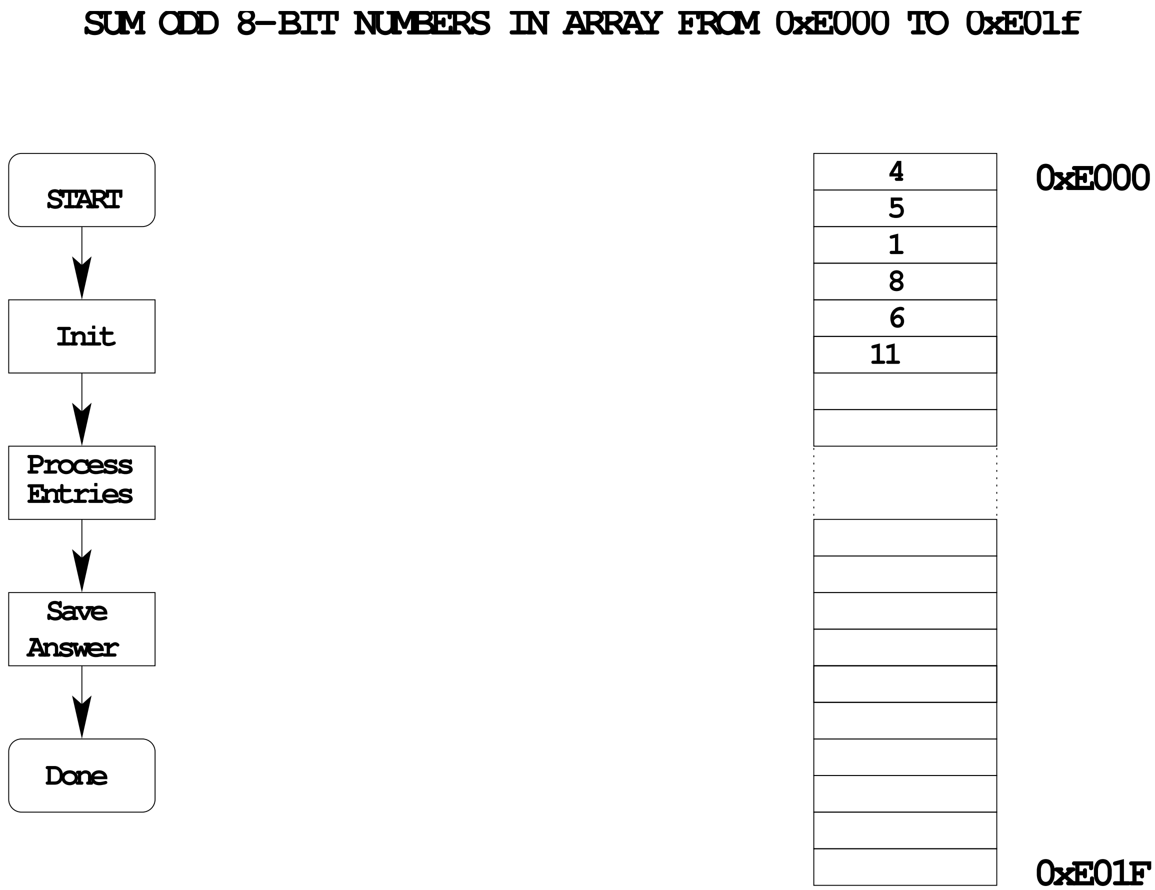
Start by drawing a picture of the data structure in memory:

SUM ODD NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f

Treat numbers as 8-bit unsigned numbers



Start with the big picture

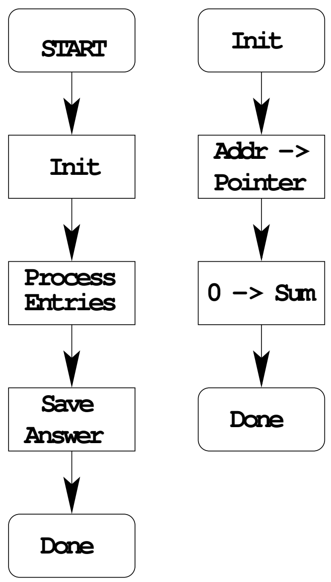


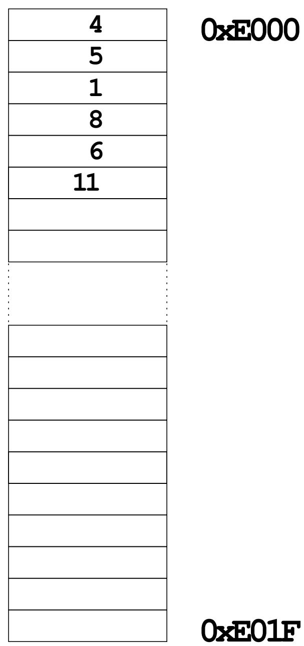
Add details to blocks

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



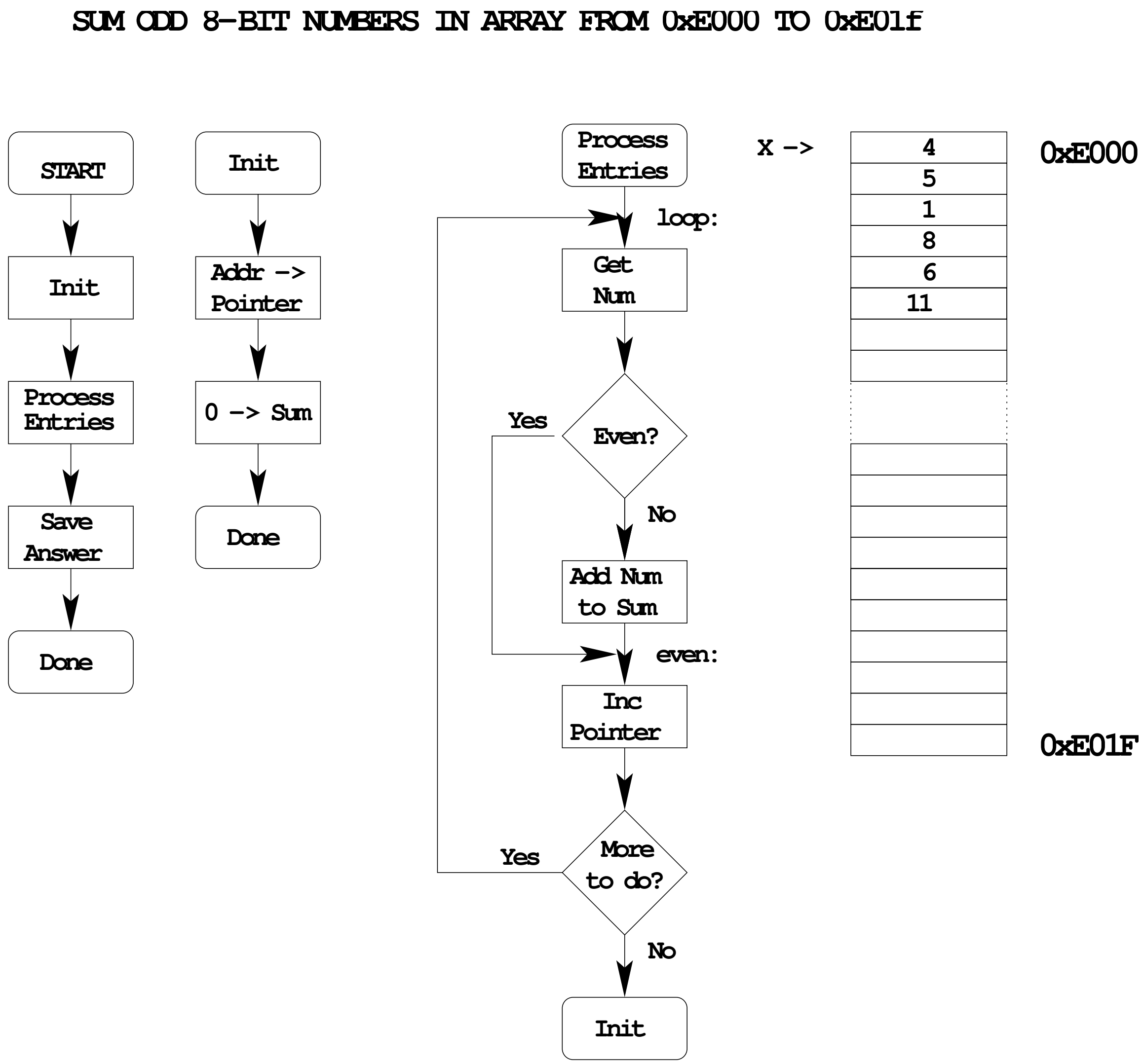


Pointer: X or Y -- use X

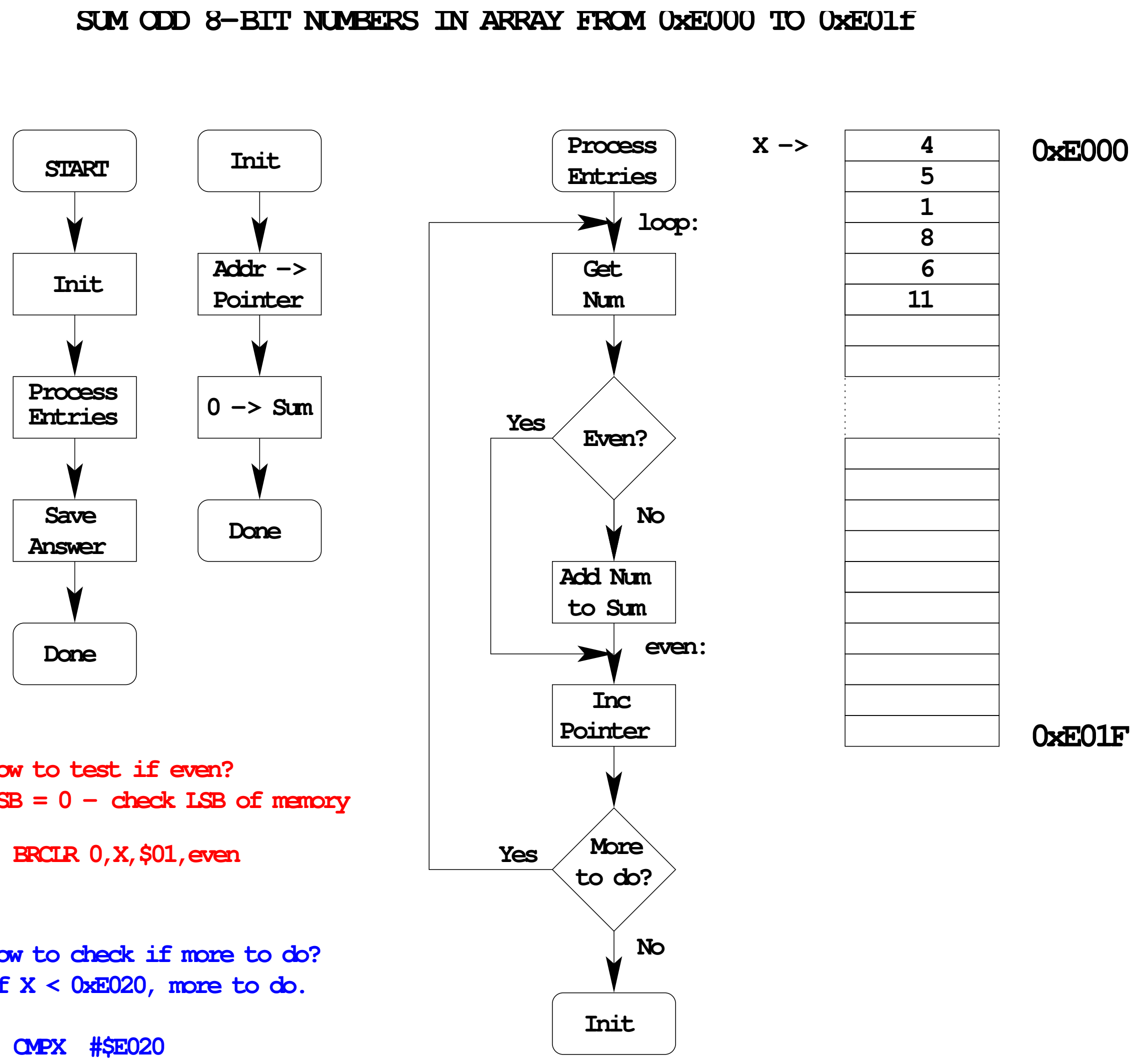
Sum: 16-bit register
D or Y

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



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



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

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

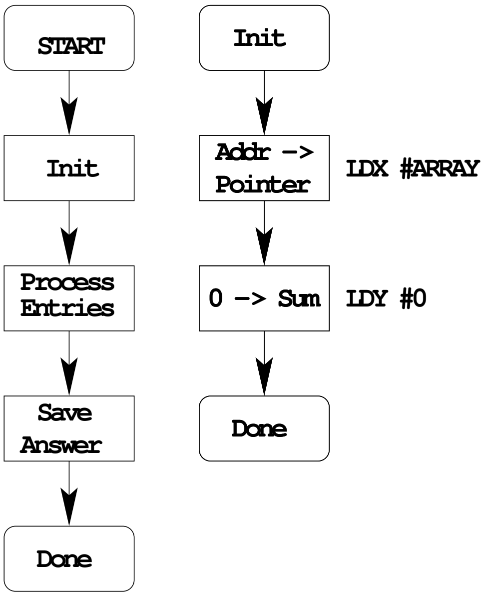
How to check if more to do?
If X < 0xE020, more to do.

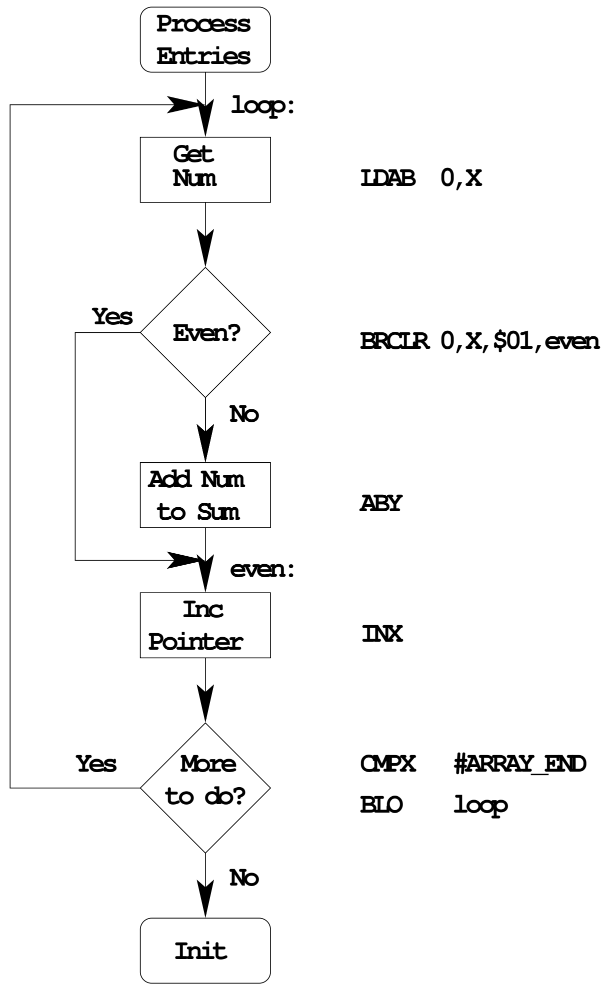
```
CMPX  #$E020
BLO or BLT   loop   ?
```

Address in unsigned, use unsigned compare
BLO loop

Convert blocks to assembly code

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



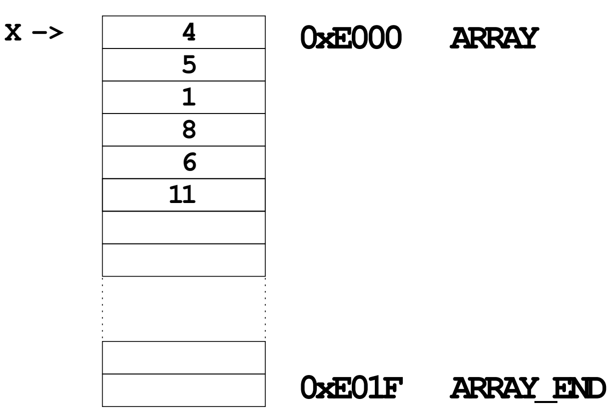


How to test if even?
LSB = 0 - check LSB of memory

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     $1000
data:   equ     $2000

array:  equ     $E000
len:    equ     $20

        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

        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`

```
1000                prog:    equu    $1000
2000                data:    equ    $2000
e000                array:   equ    $E000
0020                len:     equ    $20
1000                         org    prog
1000 ce e0 00                ldx    #array        ; initialize pointer
1003 cd 00 00                ldy    #0            ; initialize sum to 0
1006 e6 00          loop:    ldab   0,x           ; get number
1008 0f 00 01 02             brclr  0,x,$01,skip ; skip if even
100c 19 ed                   aby                  ; odd - add to sum
100e 08             skip:    inx                  ; point to next entry
100f 8e e0 20                cpx    #(array+len) ; more to process?
1012 25 f2                   blo    loop          ; if so, process
1014 7d 20 00                sty    answer        ; done -- save answer
1017 3f                      swi

2000                         org    data
2000                answer:  ds.w   1             ; reserve 16-bit word for answer
```

And here is the `.s19` file:

```
S012000046696C653A20746573742E61736D0ADA
S1131000CEE000CD0000E6000F00010219ED088ECD
S10B1010E02025F27D20003FE1
S9030000FC
```