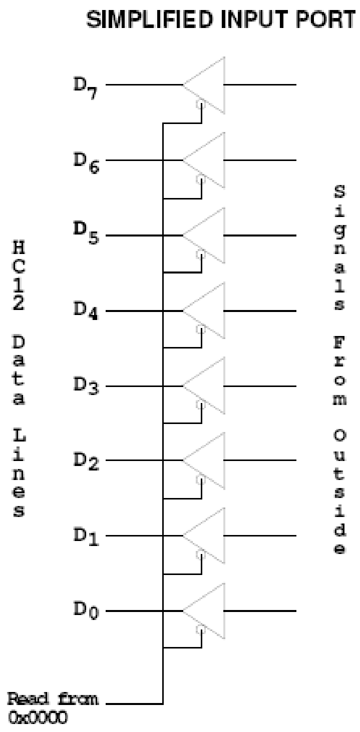
- **Another simple program in assembly language**
- **Using the stack and the stack pointer**
- Huang Section 4.3
    - A program to add all the odd numbers in a memory array
    - Flow charts
    - Assembly language program
    - Assembly listing file
    - Assembly map file
    - The Stack and the Stack Pointer
    - The stack is an area of memory used for temporary storage
    - The stack pointer points to the last byte pushed onto the stack
    - Some instructions which use the stack, and how data is pushed onto and pulled off of the stack.

Input and Output Ports

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

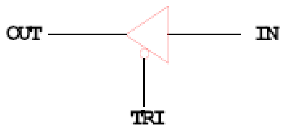


Any read from address $0000 gets signals from outside

LDAA $00

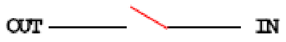
Puts data from outside into accumulator A.

Data from outside looks like a memory location.

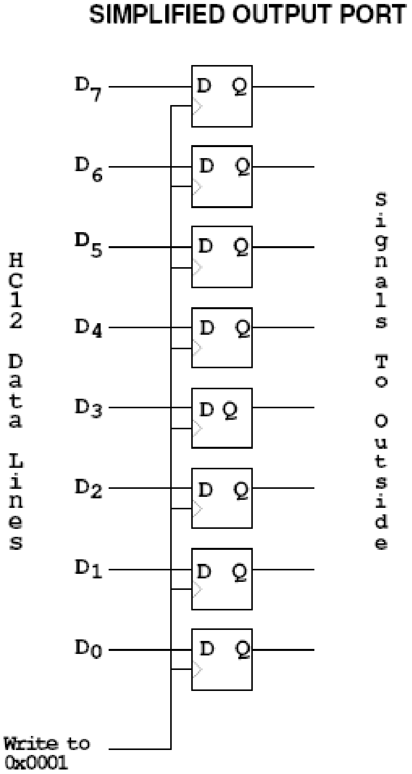


A Tri-State Buffer acts like a switch

If TRI is not active, the switch is open:
OUT will not be drived by IN
Some other device can drive OUT



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



Any write to address $01 latches data into FF, so data goes to external pints

MOVB #$AA,$01

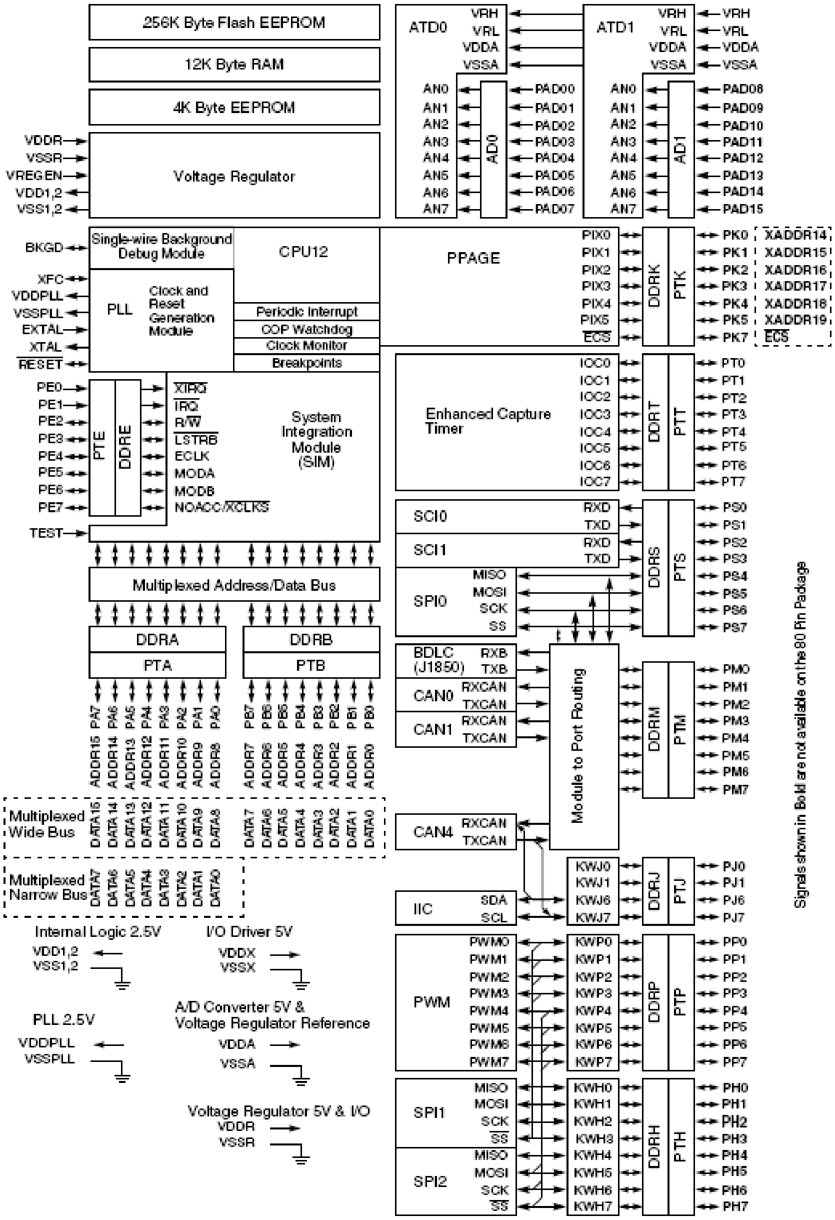
Puts $AA on the external pins

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

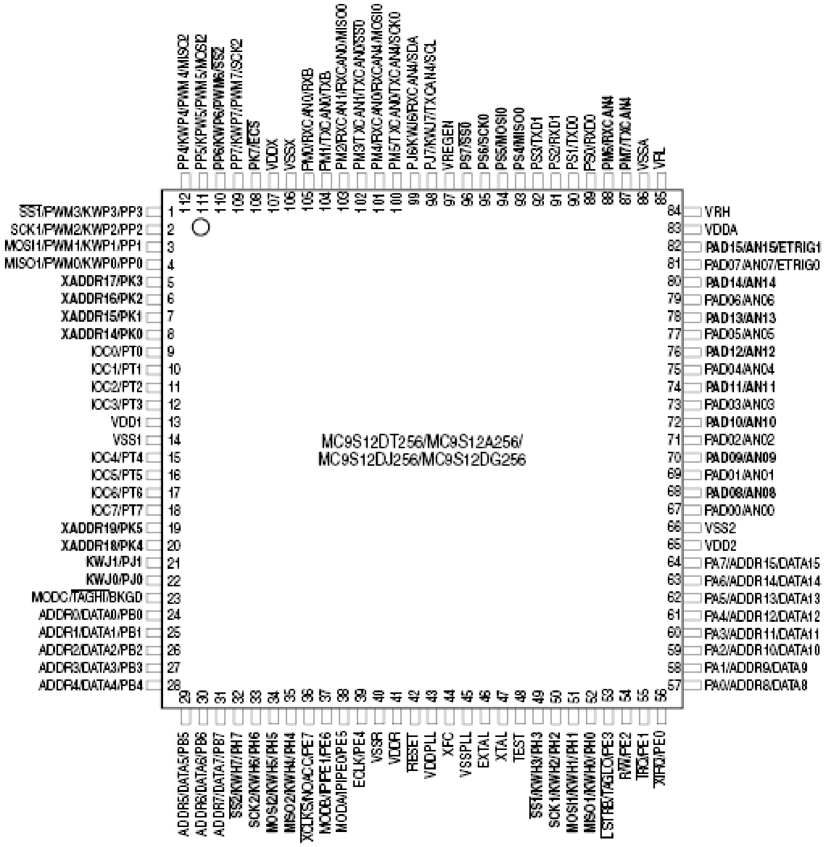
Accumulator A will have $AA after this

Figure 1-1 MC9S12DT256 Block Diagram


256K Byte Flash EEPROM
12K Byte RAM
4K Byte EEPROM
Voltage Regulator
VDDR
VSSR
VREGEN
VDD1,2
VSS1,2
ATD0
ATD1
VRH
VRL
VDDA
VSSA
AN0 AN1 AN2 AN3 AN4 AN5 AN6 AN7
AD0
AD1
PAD00 PAD01 PAD02 PAD03 PAD04 PAD05 PAD06 PAD07
PAD08 PAD09 PAD10 PAD11 PAD12 PAD13 PAD14 PAD15
BKGD
Single-wire Background Debug Module
CPU12
PPAGE
PIX0 PIX1 PIX2 PIX3 PIX4 PIX5 ECS
DDRK
PTK
PK0 PK1 PK2 PK3 PK4 PK5 PK7
XADDR14 XADDR15 XADDR16 XADDR17 XADDR18 XADDR19 ECS
XFC
VDDPLL
VSSPLL
EXTAL
XTAL
RESET
PLL
Clock and Reset Generation Module
Periodic Interrupt
COP Watchdog
Clock Monitor
Breakpoints
PE0 PE1 PE2 PE3 PE4 PE5 PE6 PE7
PTE
DDRE
XIRQ
IRQ
R/W
LSTRB
ECLK
MODA
MODB
NOACC/XCLKS
System Integration Module (SIM)
TEST
Enhanced Capture Timer
IOC0 IOC1 IOC2 IOC3 IOC4 IOC5 IOC6 IOC7
DDRT
PTT
PT0 PT1 PT2 PT3 PT4 PT5 PT6 PT7
SCI0
SCI1
RXD
TXD
SPI0
MISO
MOSI
SCK
SS
DDRS
PTS
PS0 PS1 PS2 PS3 PS4 PS5 PS6 PS7
Multiplexed Address/Data Bus
DDRA
PTA
DDRB
PTB
PA7 PA6 PA5 PA4 PA3 PA2 PA1 PA0
PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0
ADDR15 ADDR14 ADDR13 ADDR12 ADDR11 ADDR10 ADDR9 ADDR8
ADDR7 ADDR6 ADDR5 ADDR4 ADDR3 ADDR2 ADDR1 ADDR0
Multiplexed Wide Bus
DATA15 DATA14 DATA13 DATA12 DATA11 DATA10 DATA9 DATA8
DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
Multiplexed Narrow Bus
DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
BDLC (J1850)
RXB
TXB
CAN0
CAN1
CAN4
RXCAN
TXCAN
Module to Port Routing
DDRM
PTM
PM0 PM1 PM2 PM3 PM4 PM5 PM6 PM7
IIC
SDA
SCL
KWJ0 KWJ1 KWJ6 KWJ7
DDRJ
PTJ
PJ0 PJ1 PJ6 PJ7
PWM
PWM0 PWM1 PWM2 PWM3 PWM4 PWM5 PWM6 PWM7
KWP0 KWP1 KWP2 KWP3 KWP4 KWP5 KWP6 KWP7
DDRP
PTP
PP0 PP1 PP2 PP3 PP4 PP5 PP6 PP7
SPI1
SPI2
KWH0 KWH1 KWH2 KWH3 KWH4 KWH5 KWH6 KWH7
DDRH
PTH
PH0 PH1 PH2 PH3 PH4 PH5 PH6 PH7
Internal Logic 2.5V
VDD1,2
VSS1,2
I/O Driver 5V
VDDX
VSSX
PLL 2.5V
VDDPLL
VSSPLL
A/D Converter 5V & Voltage Regulator Reference
VDDA
VSSA
Voltage Regulator 5V & I/O
VDDR
VSSR
Signals shown in Bold are not available on the 80 Pin Package

Ports on the HC12

- How do you get data out of computer to the outside?
- A Port on the HC12 is a device that 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 57 to 64 (PORTA), 24 to 31 (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 Direction Register for PORTA is located at memory address $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 Direction Register for PORTB is located at memory address $0003. It is called **DDRB**. To make all bits of PORTB output, write a $FF to DDRB.
  - The Data Direction Register for PTH is located at memory address $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 I/O 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.

	DDRA7	**DDRA6**	**DDRA5**	**DDRA4**	**DDRA3**	**DDRA2**	**DDRA1**	**DDRA0**	
Reset	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**$0002**

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**	**PA3**	**PA2**	**PA1**	**PA0**	
Reset	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**$0000**

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

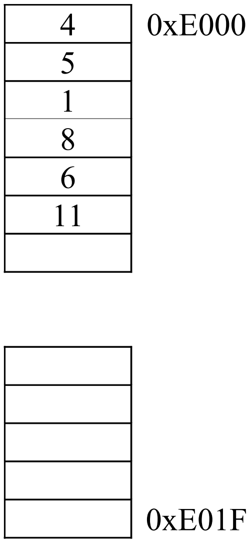
1. Think about how data will be stored in memory.
    - Draw a picture
2. Think about how to process data
    - Draw a flowchart
3. Start with big picture. Break into smaller parts until reduced to individual instructions
    - Top-down design
4. Use names instead of numbers

Another Example of an Assembly Language Program

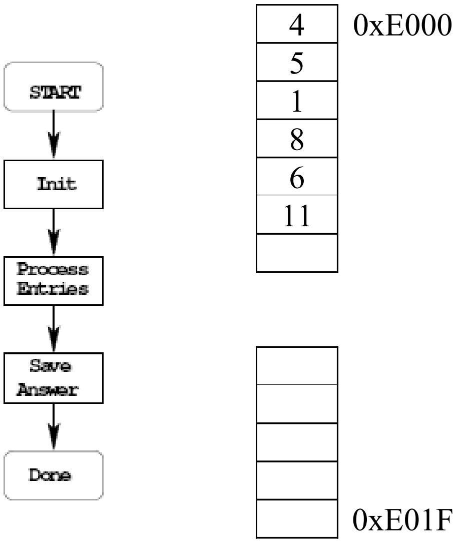
- Add the odd numbers in an array of data.
- The numbers are 8-bit unsigned numbers.
- The address of the first number is $E000 and the address of the final number is $E01F.
- Save the result in a variable called answer at address $2000.

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

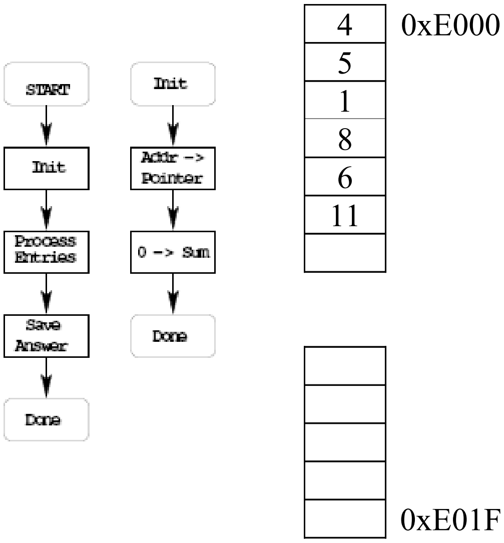
SUM ODD NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f
Treat numbers as 8–bit unsigned numbers



Start with the big picture



Add details to blocks

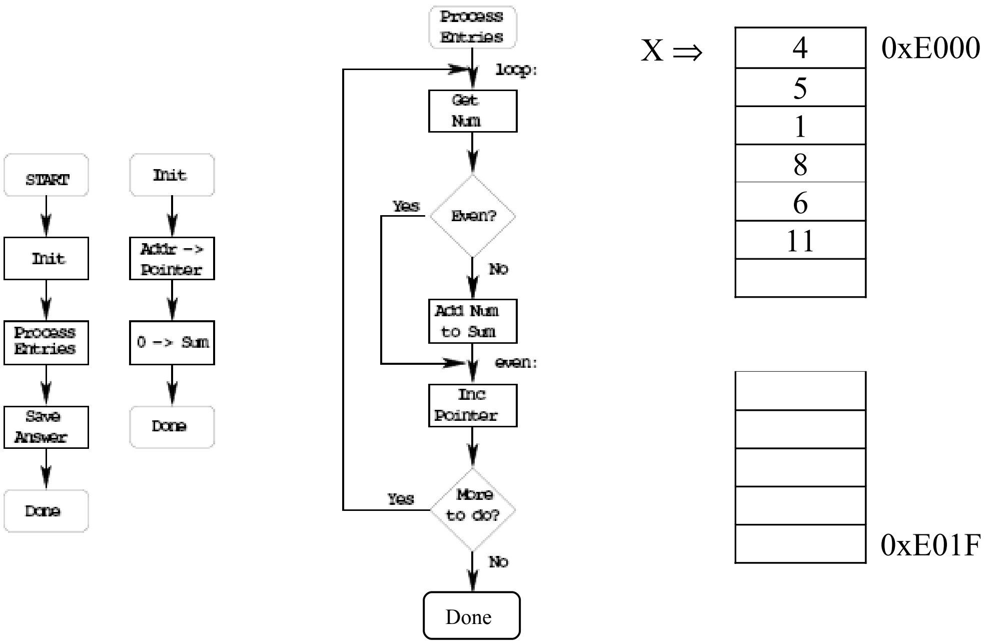


Decide on how to use CPU registers for processing data

Pointer: X or Y –– Let us use X
Sum: 8-bit or 16–bit register
Can use 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
```

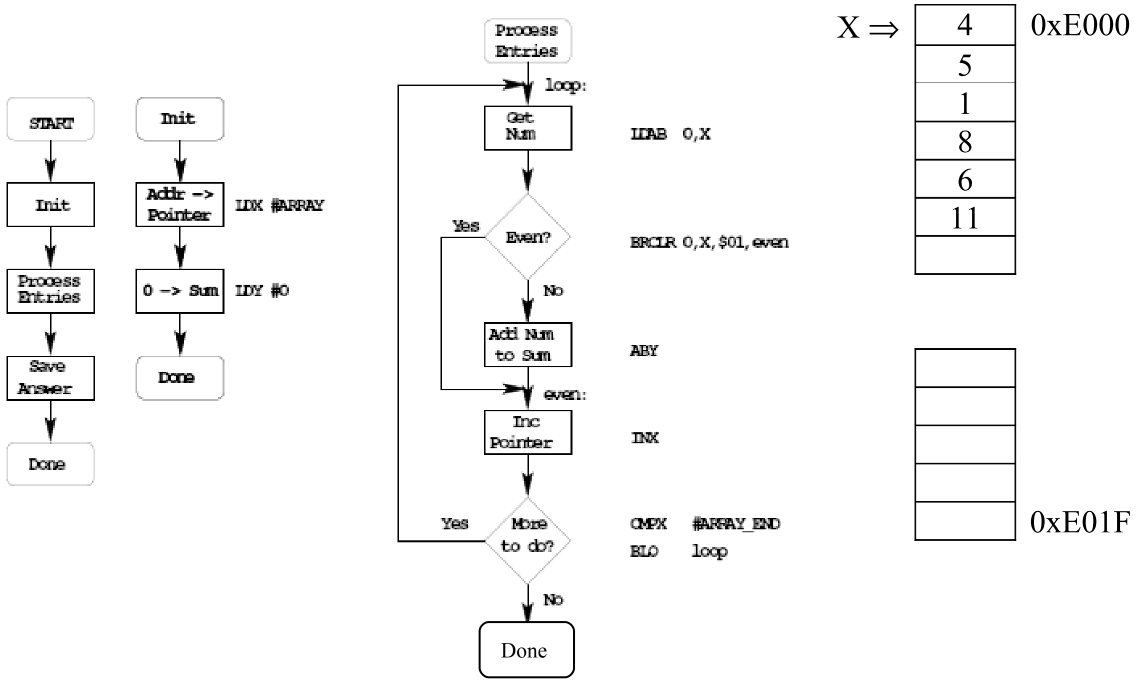
How to check if more to do?
If X < 0xE020, more to do.

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

Address in unsigned, use unsigned compare

```
        BLO loop
```

Convert blocks to assembly code



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,even    ; skip if even
        aby                    ; odd - add to sum
even:   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,even is 0f 00 01 02

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

1000                    prog: equ    $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,even; skip if even
100c 19 ed                    aby                ; odd - add to sum
100e 08                 even: 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

Executed: Sun Jan 20 10:00:02 2008
Total cycles: 36, Total bytes: 24
Total errors: 0, Total warnings: 0
```

Here is the .s19 file:

```
S011000046696C653A2074657374332E730A76
S1131000CEE000CD0000E6000F00010219ED088ECD
S10B1010E02025F27D20003FE1
S9030000FC
```