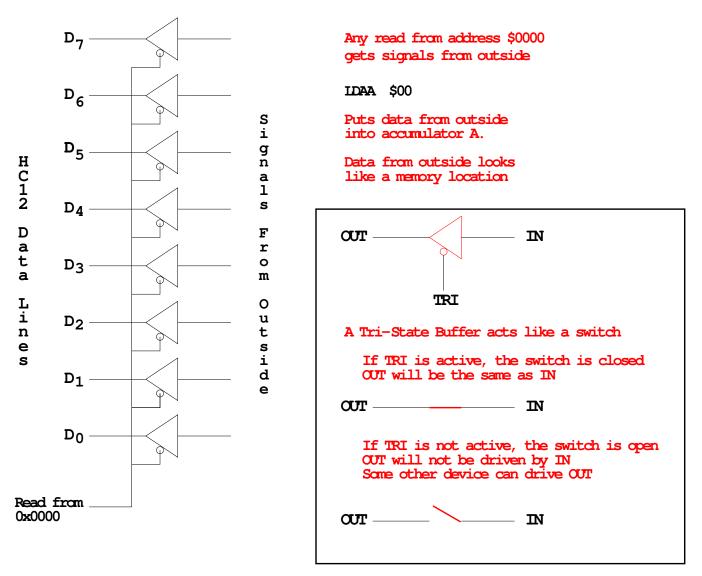
# Input and Output Ports

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





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

S i

g

ñ

a 1 s

т

ο

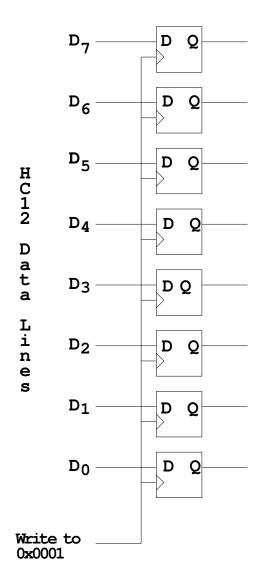
0

u

t s i

d

е



# SIMPLIFIED OUTPUT PORT

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

#### MC9S12DP256B Device User Guide — V02.13

VRH <−VRH VRH 256K Byte Flash EEPROM ATD0 VRL ATD1 VRL <−VRL VDDA VDDA 12K Byte RAM VSSA -VSSA VSSA PAD00 PAD08 AN0 AN0 4K Byte EEPROM AN1 ←PAD01 AN1 PAD09 AN2 AN2 4 PAD02 4 PAD10 PAD02 PAD10
PAD11
PAD11
PAD12 VDDR-AN3 • AN3 ADO AD 1 VSSR-AN4 AN4 **\*** <−PAD05 <−PAD06 VREGEN-Voltage Regulator 4 AN5 -PAD13 AN5 AN6 . -PAD14 VDD1,2 AN6 VSS1.2 AN7 <−PAD07 AN7 <−PAD15 PK0 XADDR14 PIX0 Single-wire Background Debug Module BKGD◀ CPU12 <> PK1 ¦ <> PK2 ¦ XADDR15 PPAGE PIX1 PIX2 -XADDR16 DDRK XFC-PTK <->PK3 XADDR17 →PK4 XADDR18 PIX3 ---VDDPLL-Clock and Reset Generation Module PIX4 PLL VSSPLL-Periodic Interrupt PIX5 APK5 XADDR19 EXTAL-COP Watchdog Clock Monitor ECS ↔ PK7 ECS XTAL 🔫 IOC0 <-> PT0 RESET -Breakpoints <→ PT1 <→ PT2 IOC1 PE0-XIRQ IOC2 PE1-> -IRQ Enhanced Capture < ► PT3 ► PT4 ► PT5 System IOC3 -DDR PE2-R/W E Integration Module Timer IOC4 DDRE PE3-\* \* \* LSTRB IOC5 PE4 ECLK (SIM) 10C6 <→ PT6 PE5 MODA ↔PT7 1007 PE6--MODB NOACC/XCLKS PE7 RXD PS0 SCI0 тхр <→PS1 TEST-> <→ PS2 -SCI1 \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* DDRS <-> PS2 <-> PS3 <-> PS4 PTS TXD MISO Signals shown in Bold are not available on the 80 Pin Package Multiplexed Address/Data Bus MOSI • <≻ PS5 SPI0 SCK <≻ PS6 \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \$ SS PS7 Ť V DDRA DDRB RXB BDLC (J1850) PTB PTA ► PM0 TXB Port Routing PB7 PB66 PB56 PB44 PB24 PB22 PB16 PB16 \* \* \* \* \* RXCAN <→ PM1 \$ \$ \$ CAN0 TXCAN → PM2 PA6 -PA6 -PA4 -PA4 -PA2 -PA1 -PA1 -DDRM <→ PM3 <→ PM4 RXCAN PTM CAN1 TXCAN ADDR15 ADDR14 ADDR13 ADDR12 ADDR12 ADDR11 ADDR10 ADDR9 ADDR8 ADDR6 ADDR6 ADDR5 ADDR4 ADDR3 ADDR3 ADDR2 ADDR2 ← PM5 RXCAN • CAN2 5 TXCAN <≻ PM6 Module 1 RXCAN PM7 CAN3 Mnltiblexed Bara115 DATA12 DATA12 DATA12 DATA12 DATA12 DATA12 DATA3 DATA3 DATA3 TXCAN DATA7 DATA6 DATA5 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0 CAN4 RXCAN TXCAN - - -KWJ0 PJ0 Multiplexed DATAS DATAS Narrow Bus DATAS DATAS DDRJ KWJ1 ЪТJ <→ PJ1 **∢≻**PJ6 SDA KWJ6 IIC SCL KWJ7 ► PJ7 Internal Logic 2.5V I/O Driver 5V - PPO PWM0 KWP0 VDDX PWM1 KWP1 <→ PP1 VSSX PWM2 KWP2 <→ PP2 PWM3 KWP3 <→ PP3 РТР DDRF PWM A/D Converter 5V & Voltage Regulator Reference <► PP4 PWM4 KWP4 -PLL 2.5V PWM5 -<→ PP5 KWP5 VDDPLL VDDA ----PWM6 KWP6 <>> PP6 -VSSPLL VSSA PWM7 KWP7 <→ PP7 1 MISO KWH0 <-> PH0 MOSI KWH1 **\* \* \*** <≻ PH1 Voltage Regulator 5V & I/O VDDR SPI1 SCK KWH2 <→ PH2 DDRH SS кwнз 🖘 VSSR \_ < → PH3 < → PH4 PTH -MISO KWH4 → PH5 MOSI -KWH5 -SPI2 SCK KWH6 <-> PH6 SS KWH7 PH7



(A) MOTOROLA

21

## 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 \$OF 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
  - $\ast$  To make PTH an output, use MM to change the contents of address 0262 (DDRH) to an FF.
  - $\ast$  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 PORIA (address 0x0000). When you read from PORIA input pins will return the value of the signals on them  $(0 \Rightarrow 0V, 1 \Rightarrow 5V)$ ; output pins will return the value written to them.

PA7	PA6	PA5	PA4	DP3	PA2	PA1	PAO	\$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 : PORTB : DDRA : DDRB :	equ equ equ equ	\$00 \$01 \$02 \$03			
	org movb movb	prog #\$ff,DDRA #\$00,DDRB		PORTA PORTB	output input
	ldaa staa swi	PORTB PORTA			

• 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.
- $\bullet$  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.

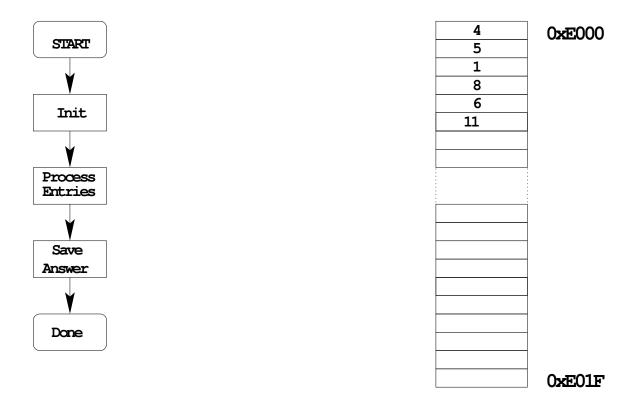
Start by drawing a picture of the data structure in memory:

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

Δ	0.000
	0xE000
5	
1	
8	
4 5 1 8 6	
11	
	0xE01F
	OVENTE

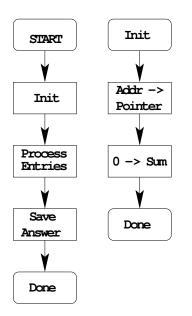
# Start with the big picture

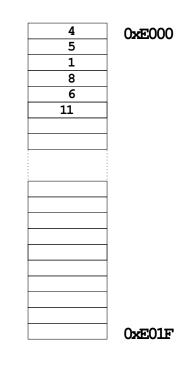
### SUM ODD 8-BIT NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f



## Add details to blocks

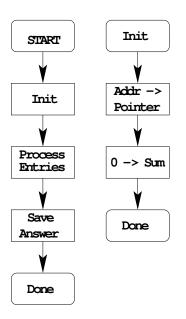
### SUM ODD 8-BIT NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f

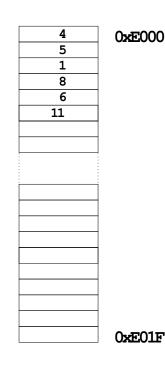




Decide on how to use CPU registers for processing data





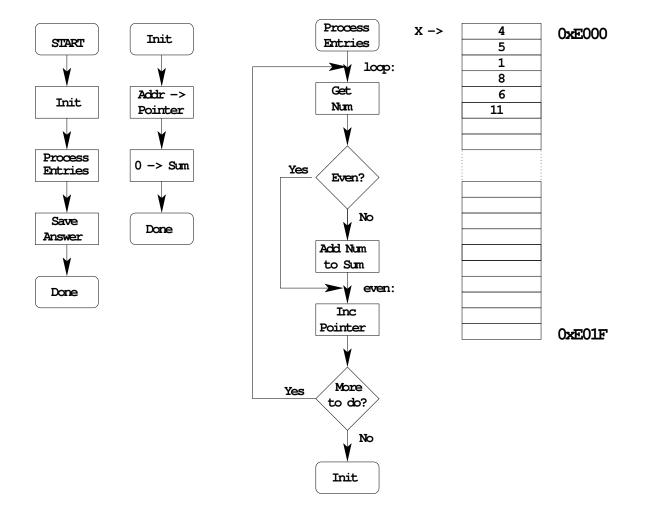


Pointer: X or Y -- use X

Sum: 16-bit register D or Y

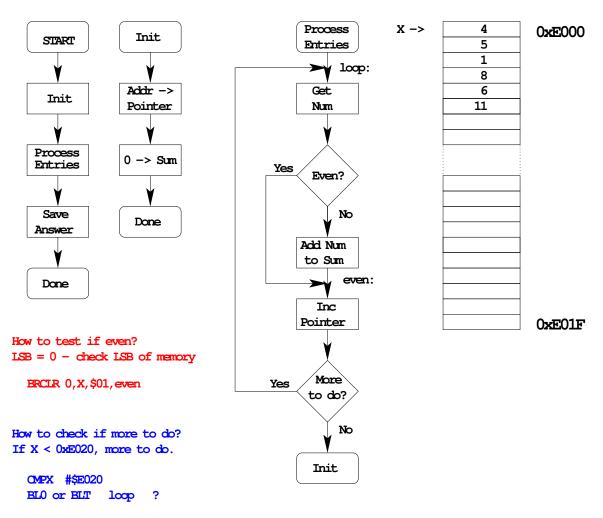
> No way to add 8-bit number to D Can use ABY to add 8-bit number to Y

## 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

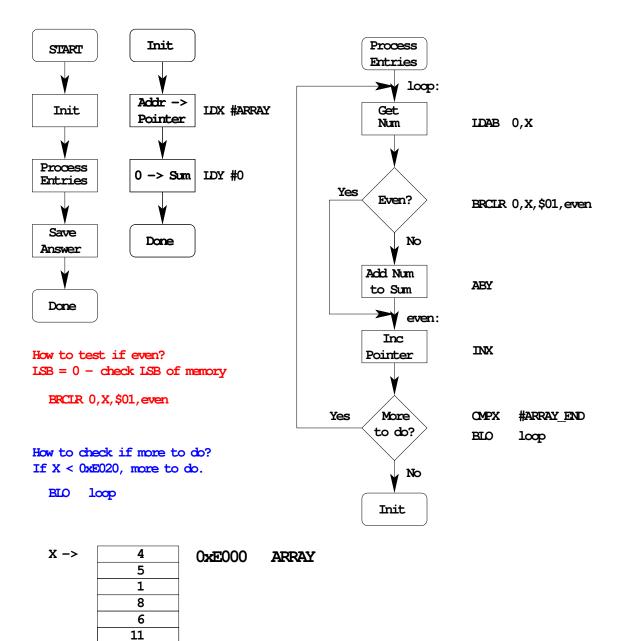


#### SUM ODD 8-BIT NUMBERS IN ARRAY FROM 0xE000 TO 0xE01f

Address in unsigned, use unsigned compare BLO loop

## Convert blocks to assembly code





ARRAY END

 $0 \times E01F$ 

## Write program

;Progra	m to sum	odd numbers in a memory	array
	equ equ	\$1000 \$2000	
array: len:	equ equ	\$E000 \$20	
	org	prog	
_	ldx ldy	#array #0	; initialize pointer ; initialize sum to O
loop:	ldab brclr aby	0,x 0,x,\$01,skip	; get number ; skip if even ; odd - add to sum
skip:	inx cpx blo sty swi	#(array+len) loop answer	; point to next entry ; more to process? ; if so, process ; done save answer
answer:	org ds.w	data 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 2000 e000 0020 1000 1000 ce e0 00 1003 cd 00 00 1006 e6 00 1008 0f 00 01 02 100c 19 ed 100c 19 ed 100c 8 100f 8e e0 20 1012 25 f2 1014 7d 20 00 1017 3f	prog: data: array: len: loop: skip:	equ equ equ org ldx ldy ldab brclr aby inx cpx blo sty swi	-	<pre>; initialize pointer ; initialize sum to 0 ; get number ; skip if even ; odd - add to sum ; point to next entry ; more to process? ; if so, process ; done save answer</pre>
1017 3f 2000 2000	answer:	swi org ds.w	data 1	; reserve 16-bit word for answer

And here is the .s19 file:

S012000046696C653A20746573742E61736D0ADA S1131000CEE000CD0000E6000F00010219ED088ECD S10B1010E02025F27D20003FE1 S9030000FC