• More on programming in assembly language
• Introduction to Ports on the HC12
  o Input and Output Ports
    ▪ Simplified input port
    ▪ Simplified output port
    ▪ Simplified input/output port
    ▪ PORTA, PORTB, DDRA, DDRB
      ▪ A simple program to use PORTA and PORTB
  o Good programming style
  o Tips for writing programs
  o A program to average the numbers in a memory array
  o Flow charts
  o Assembly language program
  o Assembly listing file
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 driven by IN
Some other device can drive OUT
• How do you get data out of computer to the outside?

**SIMPLIFIED OUTPUT PORT**

- Any write to address $01 latches data into FF, so data goes to external pins
  - MOVB #$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
• Most I/O ports on MC9S12 can be configured as either input or output

SIMPLIFIED INPUT/OUTPUT PORT

• A write to address 0x0000 writes data to the flip-flop
  A read from address 0x0000 reads data on pin

• If Bit 7 of DDRA is 0, the port is an input port. Data written to FF does not get to pin though tri-state buffer

• If Bit 7 of DDRA is 1, the port is an output port. Data written to flip-flop does get to pin though tri-state buffer

• DDRA (Data Direction Register A) is located at 0x0002
Figure 1-1 MC9S12DT256 Block Diagram
**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, PTJ** and **PTP**:

  - **PORTA** is accessed by reading and writing address $0000$.
    - DDRA is accessed by reading and writing address $0002$.

  - **PORTB** is accessed by reading and writing address $0001$.
    - DDRB is accessed by reading and writing address $0003$.

  - **PTJ** is accessed by reading and writing address $0268$.
    - DDRJ is accessed by reading and writing address $026A$.

  - **PTP** is accessed by reading and writing address $0258$.
    - DDRP is accessed by reading and writing address $025A$.

- On the DRAGON12-Plus EVB, eight LEDs and four seven-segment LEDs are connected to PTB.

  - Before you can use the eight individual LEDs or the seven-segment LEDs, you need to enable them.

  - Bit 1 of PTJ must be low to enable the eight individual LEDs

  - Bits 3-0 of PTP are used to enable the four seven-segment LEDs
    * A low PTP0 enables the left-most (Digit 3) seven-segment LED
    * A low PTP1 enables the second from the left (Digit 2) seven-segment LED
    * A low PTP2 enables the third from the left (Digit 1) seven-segment LED
    * A low PTP3 enables the right-most (Digit 0) seven-segment LED

  – To use the eight individual LEDs and turn off the seven-segment LEDs, write ones to Bits 3-0 of PTP:
BSET #$0F,DDRP ; Make PTP3 through PTP0 outputs
BSET #$0F,PTP ; Turn off seven-segment LEDs

• On the DRAGON12-Plus EVB, the LCD display is connected to PTK

• When you power up or reset the HC12, PORTA, PORTB, PTJ and PTP are input ports.

• You can make any or all bits of PORTA, PORTB PTP and PTJ outputs by writing a 1 to the corresponding bits of their Data Direction Registers.

  – You can use DBug-12 to manipulate the IO ports on the 68HCS12
    * To make PTB an output, use MM to change the contents of address $0003 (DDRB) to an $FF.

    * You can now use MM to change contents of address $0001 (PORTB), which changes the logic levels on the PORTB 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.

<table>
<thead>
<tr>
<th>DDRA7</th>
<th>DDRA6</th>
<th>DDRA5</th>
<th>DDRA4</th>
<th>DDRA3</th>
<th>DDRA2</th>
<th>DDRA1</th>
<th>DDRA0</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$0002</td>
</tr>
</tbody>
</table>

For example, to make bits 7–4 output and bits 3–0 of Port A input, 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.

<table>
<thead>
<tr>
<th>PA7</th>
<th>PA6</th>
<th>PA5</th>
<th>PA4</th>
<th>PA3</th>
<th>PA2</th>
<th>PA1</th>
<th>PA0</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>$0000</td>
</tr>
</tbody>
</table>

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 |
| PROTA: 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

• Find the average of the 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. There are 32 numbers.

• Save the result in a variable called answer at address $2000.

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

FIND AVERAGE OF 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

Find average of 8-bit numbers in array from 0xE000 to 0xE01f

Sum: 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

```
\[ X \Rightarrow \begin{array}{c}
4 \\
5 \\
1 \\
8 \\
6 \\
11 \\
0xE000 \\
0xE01F
\end{array} \]
```
More details: How to tell if number is odd, how to tell when done

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

   BLT or BLO?

   Addresses are unsigned, so BLO

How to find average? Divide by LEN
To divide, use IDIV
   TFR Y,D ; divide in D
   LDX #LEN ; divisor in X
   IDIV
Convert blocks to assembly code

```
X => 4 0xE000
  5
  1
  8
  6
  11

0xE01F
```
Write program

; Program to average 32 numbers in a memory array

prog:  equ  $1000
data:  equ  $2000
array:  equ  $E000
len:  equ  $32

org  prog

ldx  #array
ldy  #0
loop:  ldab  0,x
aby
inx

; get number
; odd - add to sum
; point to next entry

; more to process?

blo  loop

; if so, process

tfr  y,d

; to divide, need dividend in D
idiv

; D/X quotient in x, remainder in D

stx  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

<table>
<thead>
<tr>
<th>Abs.</th>
<th>Rel.</th>
<th>Loc</th>
<th>Obj. code</th>
<th>Source line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>;Program to average 32 numbers in a memory array</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0000 2000</td>
<td></td>
<td>prog: equ $2000</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0000 1000</td>
<td></td>
<td>data: equ $1000</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0000 E000</td>
<td></td>
<td>array: equ $E000</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0000 0020</td>
<td></td>
<td>len: equ 32</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
<td>org prog</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>a002000 CEE0 00</td>
<td></td>
<td>ldx #array ; initialize pointer</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>a002003 CD00 00</td>
<td></td>
<td>ldy #0 ; initialize sum to 0</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>a002006 E600</td>
<td></td>
<td>loop: ldab 0,x ; get number</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>a002008 19ED</td>
<td></td>
<td>aby ; odd - add to sum</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>a00200A 80</td>
<td></td>
<td>inx ; point to next entry</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>a00200B 8EE0 20</td>
<td></td>
<td>cpx #(array+len) ; more to process?</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>a00200E 25F6</td>
<td></td>
<td>blo loop ; if so, process</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>a002010 B764</td>
<td></td>
<td>tfr y,d ; To divide, need dividend</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>a002012 CE00 20</td>
<td></td>
<td>ldx #len ; To divide, need divisor</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>a002015 1810</td>
<td></td>
<td>idiv ; D/X quotient in X, remainder</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>a002017 7E10 00</td>
<td></td>
<td>stx answer ; done -- save answer</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>a00201A 3F</td>
<td></td>
<td>swi</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td></td>
<td></td>
<td>org data</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>a001000</td>
<td></td>
<td>answer: ds.w 1 ; reserve 16-bit word for 27 27</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Here is the .s19 file:

```
S1 1E2000CEE000CD0000E60019ED088EE02025F6B764CE002018107E10003FAB
S9030000FC
```