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

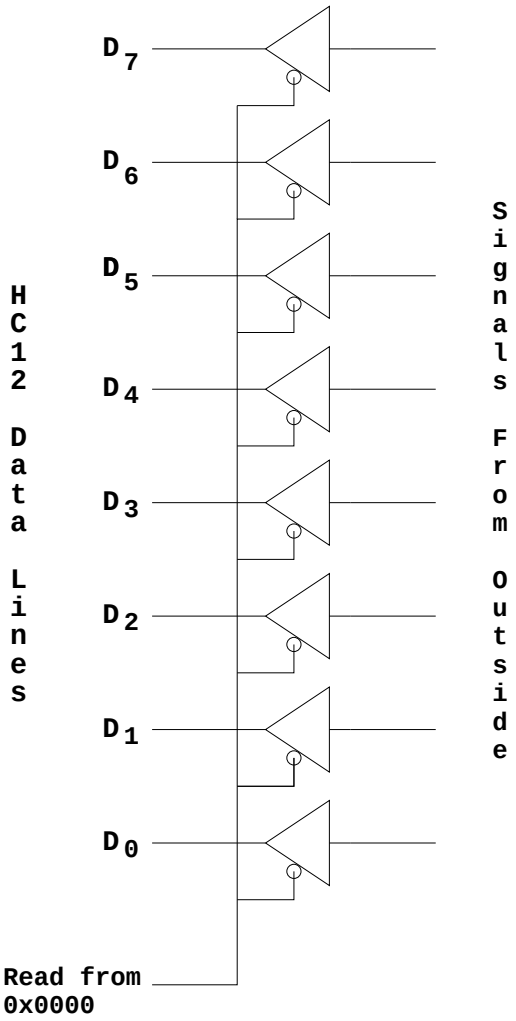
TIPS FOR WRITING PROGRAMS

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

Input and Output Ports

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

SIMPLIFIED INPUT PORT

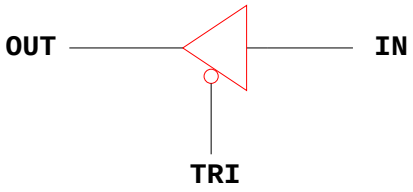


Any read from address $0000
gets signals from outside

```
LDAA  $00
```

Puts data from outside
into accumulator A.

Data from outside looks
like a memory location



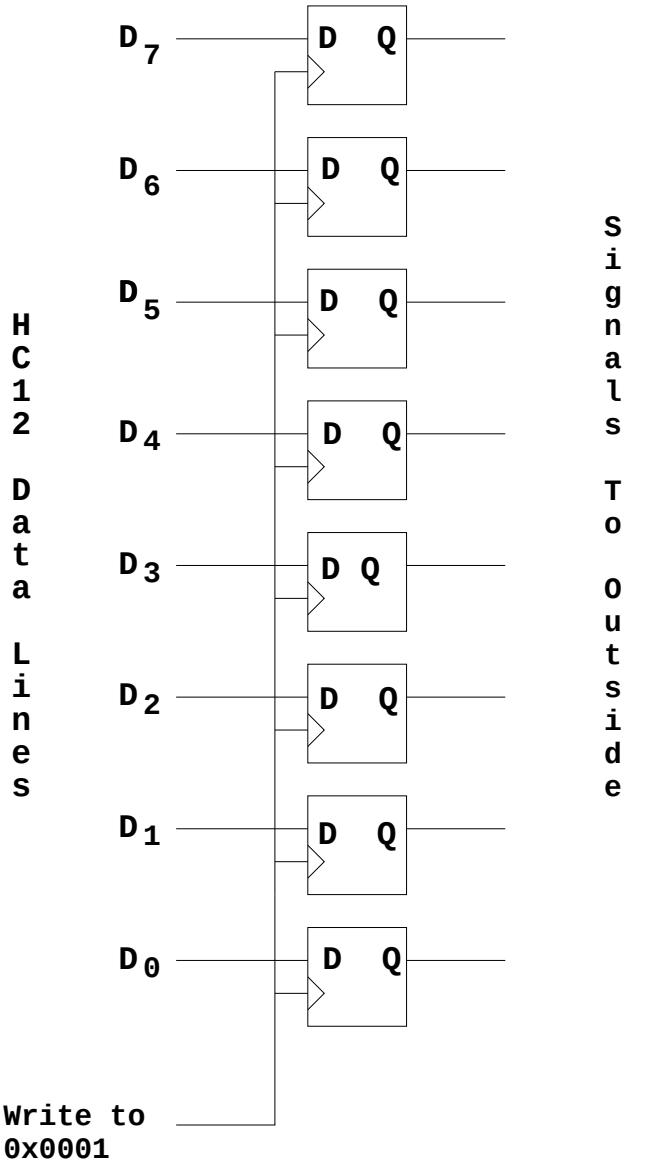
A Tri-State Buffer acts like a switch

If TRI is active, the switch is closed
OUT will be the same as IN

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 flip-flops, so data goes to external pins

```
STAA  $01
```

Puts contents of accumulator A on external pins

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

Ports on the HC12

- 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 `$01FF`.
- Two of the simplest ports to use are *PORTA* and *PORTB.*
- `PORTA` is accessed by reading and writing address `$0000`.
- `PORTB` is accessed by reading and writing address `$0001`.
- You can connect signals from the outside to `PORTA` and `PORTB` by connecting wires to pins 39 to 46 (`PORTA`) and 18 to 25 (`PORTB`).
- When you power up or reset the HC12, both `PORTA` and `PORTB` are input ports.
- You can make any or all bits of `PORTA` and `PORTB` 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`.

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     $0800

PORTA:      equ     $0000
PORTB:      equ     $0001
DDRA:       equ     $0002
DDRB:       equ     $0003

CODE:       section .text
            org     prog
            ldaa    #$ff        ; Make PORTA output
            staa    DDRA        ; $FF -> DDRA

            ldaa    PORTB
            staa    PORTA
            swi
```

Subroutines

- A subroutine is a section of code which performs a specific task, usually a task which needs to be executed by different parts of a program.
- Example:
  - Math functions, such as *square root*
- Because a subroutine can be called from different places in a program, you cannot get out of a subroutine with an instruction such as

  ```
  jmp    label
  ```

  because you would need to jump to different places depending upon which section of code called the subroutine.
- When you want to call the subroutine your code has to save the address where the subroutine should return to. It does this by saving the *return address* on the stack.
  - This is done automatically for you when you get to the subroutine by using the `JSR` (Jump to Subroutine) instruction. This instruction pushes the address of the instruction following the `JSR` instruction on the stack.
- After the subroutine is done executing its code it needs to return to the address saved on the stack.
  - This is done automatically for you when you return from the subroutine by using the `RTS` (Return from Subroutine) instruction. This instruction pulls the return address off of the stack and loads it into the program counter, so the program resumes execution of the program with the instruction following that which called the subroutine.

  The subroutine will probably need to use some HC12 registers to do its work. However, the calling code may be using its registers for some reason — the calling code may not work correctly if the subroutine changes the values of the HC12 registers.
  - To avoid this problem, the subroutine should save the HC12 registers before it uses them, and restore the HC12 registers after it is done with them.

Example of a subroutine to delay for a certain amount of time

```
; Subroutine to wait for 100 ms

delay:  ldaa    #250
loop2:  ldx     #800
loop1:  dex
        bne     loop1
        deca
        bne     loop2
        rts
```

- Problem: The subroutine changes the values of registers A and X

- To solve, save the values of A and X on the stack before using them, and restore them before returning.

```
; Subroutine to wait for 100 ms

delay:  psha            ; Save regs used by sub on stack
        pshx
        ldaa    #250
loop2:  ldx     #800
loop1:  dex
        bne     loop1
        deca
        bne     loop2
        pulx            ; Restore regs in opposite
        pula            ; order
        rts
```

```
; Program to make a binary counter on LEDs
;
; The program uses a subroutine to insert a delay
; between counts

prog:   equ     $0800
STACK:  equ     $0A00       ;Stack ends of $09FF
PORTA:  equ     $0000
PORTB:  equ     $0001
DDRA:   equ     $0002
DDRB:   equ     $0003


CODE:   section .text
        org     prog

        lds     #STACK      ; initialize stack pointer
        ldaa    #$ff        ; put all ones into DDRA
        staa    DDRA        ; to make PORTA output
        clr     PORTA       ; put $00 into PORTA
loop:   jsr     delay       ; wait a bit
        inc     PORTA       ; add one to PORTA
        bra     loop        ; repeat forever


; Subroutine to wait for 100 ms

delay:  psha
        pshx
        ldaa    #250
loop2:  ldx     #800
loop1:  dex
        bne     loop1
        deca
        bne     loop2
        pulx
        pula
        rts
```