

## Using the 68HC12 Java Simulator

---

### INSTALLING SIMHC12

The 68HC12 Java simulator is available on our website under Software/Docs or online at the following website:

<http://www.almy.us/68hc12.html>

Follow the installation instruction on this website. The help from the actual program is available at

[http://mil.ufl.edu/4744/software/simhc12\\_help/sim0001.html](http://mil.ufl.edu/4744/software/simhc12_help/sim0001.html)

There are several choices of programs in the Sim68HC12 package. You should choose "SimHC12 Expanded Narrow Mode."

### WHAT IS AN S19 FILE?

S19 files contain the binary machine code (represented in hexadecimal values) that is executed in the microprocessor. These files are generated by an assembler (such as MiniIDE/AS12) that takes ASM files as input. For example, the following is an excerpt from the example0.asm file, but will stand on its own as `sim_hc12.asm`. (Both these files are available in the example directory on our website and at the end of this document.

```
*****  
* Filename: sim_hc12.asm  
* The assembler directives below were modified slightly  
* from example0.asm for this example  
*****  
num2      org      $8001  
          dc.w     $1234  
          org      $4000  
results   ds.b     3  
*****  
          org      $1100  
          ldd     num2  
          std     results  
end2      bra     end2
```

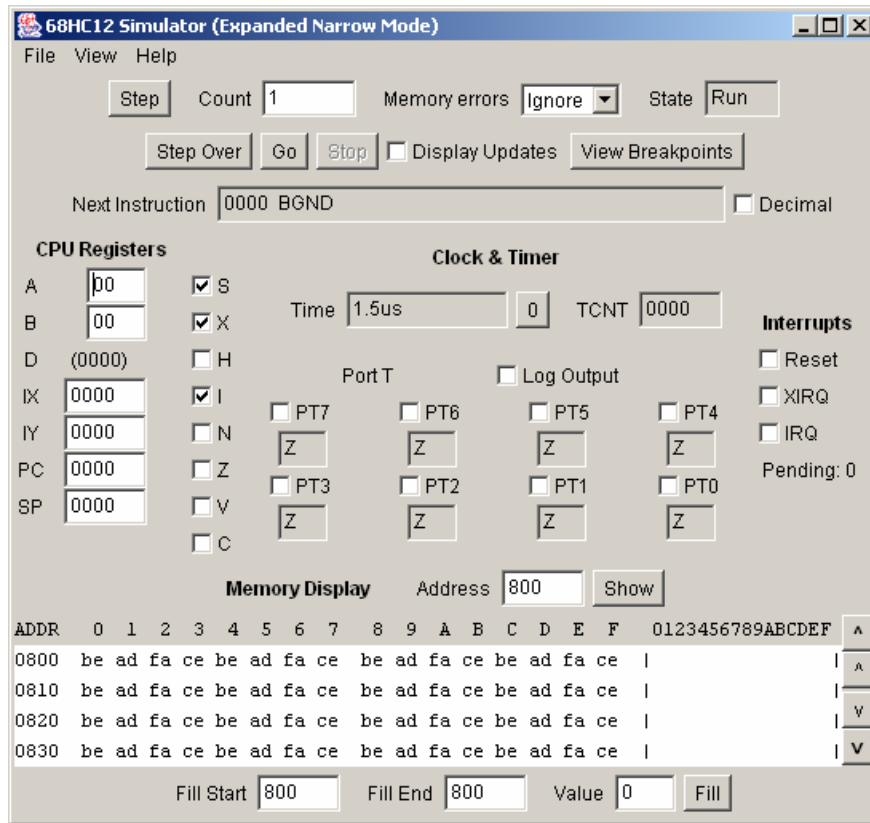
MiniIDE (using AS12) assembles the above statement into the following single line and places it in example0.s19.

**S10B1100FC80017C400020FE8C**

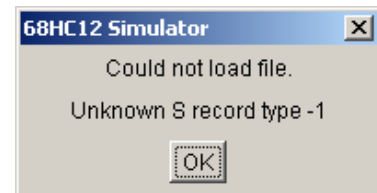
### USING S19 FILES IN THE 68HC12 JAVA SIMULATOR

The following is the SIMHC12 main window.

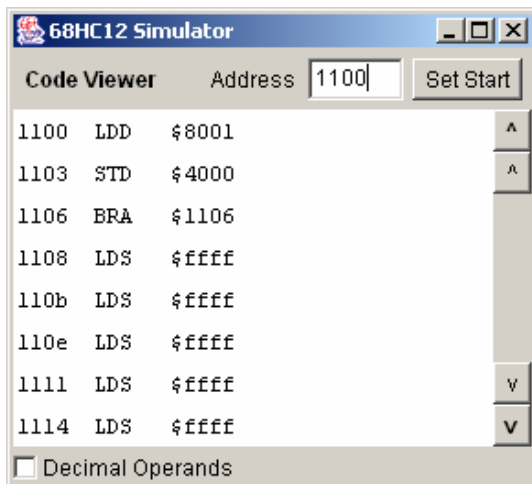
Using the 68HC12 Java Simulator



In order to load an S19 file, click on **File → Load...** on the pull-down menus; then, browse to the appropriate directory and select your S19 file. If it is a valid S19 file, the simulator will just wait for you to begin simulation. However, if the file is not recognized, the following message will appear.



If there are no error windows, the S19 load can be verified by selecting **View → Code Viewer...** from the pull-down menus. It will open a window asking for an address. Using example0.asm, it is known that the sample code shown previously is located in memory location \$1100 (because of the ORG statement). Therefore, typing 1100 in the address field and clicking on **Set Start** should show the same code (using actual addresses instead of labels). The window is shown below.



---

## Using the 68HC12 Java Simulator

---

Notice that **num2**, **results**, and **end2** have been replaced by their actual addresses. Remember that labels are only for programming convenience and not implemented internally at all by the microprocessor.

Now that the S19 load has been verified, it is time to initialize the registers for execution. The simulator defaults to a value of 00 for all of the registers. This is not a problem with registers A, B, D, IX, and IY. However, **registers PC and SP MUST be initialized properly**. Register **SP** is used for managing the 68HC12 stack and is normally initialized by the program using the LDS instruction. It will be ignored for this example, but in any real program, it must be used.

Register **PC** is the Program Counter. It contains the address of the next instruction to be executed. The file `example0.asm` contains 8 small programs located at memory locations \$1000, \$1100, \$1200, \$1300, \$1400, \$1500, \$1600, and \$1700. Using the program `sim_hc12.asm` from above, we would like to run the program located at memory \$1100. This can be done by typing 1100 in the **PC** register field of the simulator.

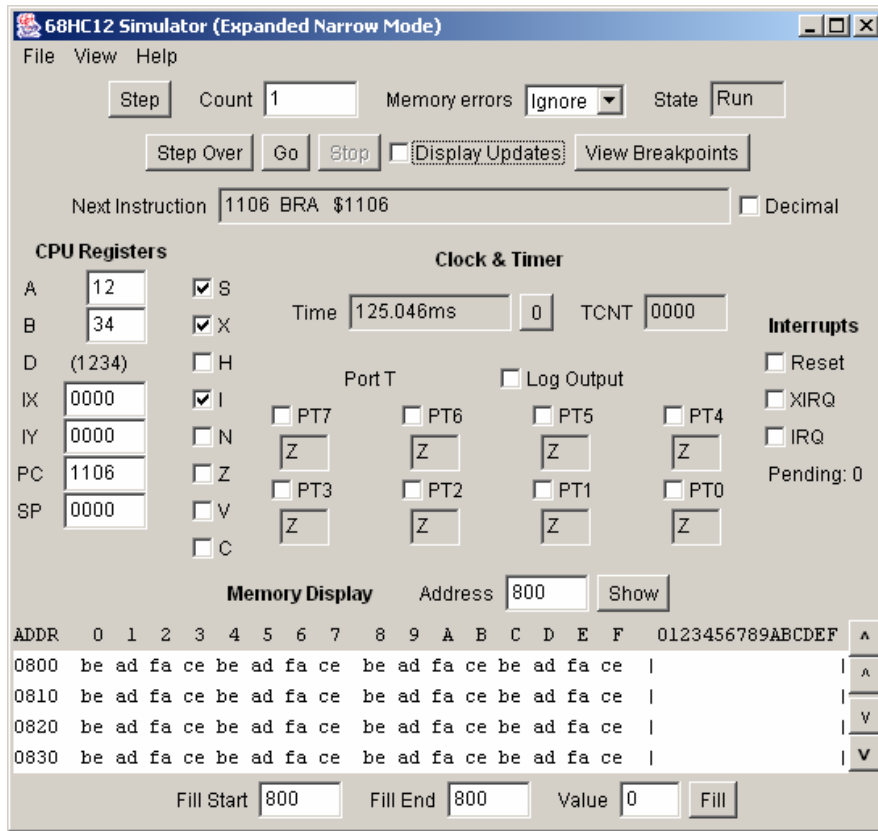
The program can now be run in two ways: **Step** or **Go**. **Step** allows the user to view the contents in the registers and memory by executing one instruction (or any number of instructions) at a time. **Go** runs successive instructions until either a breakpoint or a software interrupt is reached. All 8 of the small programs in `example0.asm` end with infinite loops. Therefore, hitting **Go** will make the simulation run forever. It can be stopped by clicking on the **Stop** button. Notice that the **Stop** button is only available while performing a **Go** operation.

The program `sim_hc12.asm` performs the following operations.

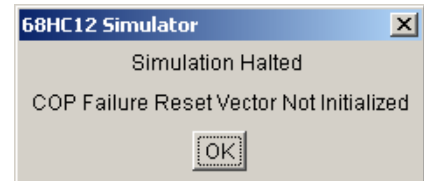
- Load the 2-byte value at memory locations \$8001 - \$8002 into register D.
- Store the 2-byte value into memory locations \$4000 - \$4001.
- Branch to memory location \$1106.

The final state of the simulator is shown on the next page (where I hit the **Stop** button very shortly after I hit the **Start** button).

Using the 68HC12 Java Simulator



It should be noted that if you just let this program run, eventually you will get **COP Failure** message. By default the COP (computer operating properly) system is on, so eventually the COP timer will run out and a COP interrupt will be generated. To turn off the COP in the simulator, add the following code at the start of your program:



COP Failure message

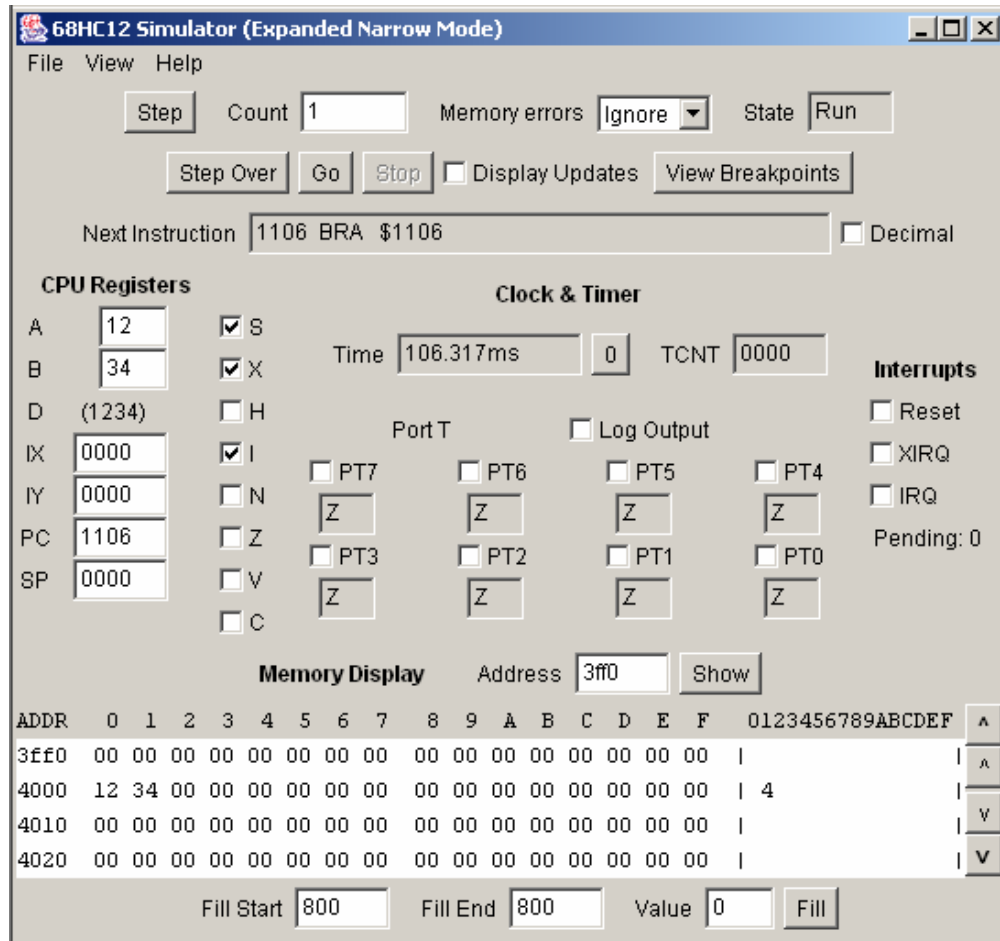
```

COPCTL equ $0016 ; COP Control Register
        ldaa #0 ; Turns of the COP for the
        staa COPCTL ; simulator.
    
```

With the COP turned off, the code will run indefinitely. Since on our UF development board, our monitor program will turn off the COP by default, this is not necessary except in the simulator.

The **Memory Display** can be used to look at the contents of memory. In the below figure, I have entered 3FF0 in the blank following **Memory Display**. A window into the contents of memory starting at address \$3ff0 will appear. Each of the four lines shown has 16-bytes of data. This shows that the contents of register D (\$1234) were successfully stored at memory locations \$4000 - \$4001 since memory locations \$4000 - \$4001 show the values of \$12 and \$34.

Using the 68HC12 Java Simulator



Notice that the **PC** register points to the branch instruction at memory location \$1106. It will always point there due to the infinite loop. Also, notice that **Next instruction** displays the assembly code for the next instruction (the one at the location given by the **PC**).

The **View** pull-down menu of the Java simulator provides many other useful tools for 68HC12 simulation. To learn more, please consult the **Help → Home Page** pull-down menu.

**sim\_hc12.asm**

```

*****
* Filename: sim_hc12.asm
* The assembler directives below were modified slightly
* from example0.asm for this example.
*****
    org     $8001
num2    dc.w  $1234
        org     $4000
results ds.b  3
*****
        org     $1100
        ldd    num2
        std    results
    
```

## Using the 68HC12 Java Simulator

---

```
end2    bra    end2
```

### sim\_hc12\_no\_cop.asm

```
*****
* Filename: sim_hc12_no_cop.asm
* The assembler directives below were modified slightly
* from example0.asm for this example. The COP has been
* disabled.
*****
COPCTL  equ    $0016          ; COP Control Register
        org    $8001
num2    dc.w   $1234
        org    $4000
results ds.b   3
*****
        org    $1100
        ldaa  #0              ; Turns of the COP for the
        staa  COPCTL          ; simulator.

        ldd   num2
        std   results
end2    bra    end2
```

### EXAMPLE0.ASM

```
* file = example0.asm
* Quick examples of Assembler directives & 6812 code
* Dr. Karl Gugel, 12/2001
* Dr. Eric Schwartz 21May2002

* Sim12 is based on the M68HC12AEVB. It assumes the following memory map:
* 16K External RAM at $4000-$7FFF
* 32K External ROM at $8000-$FFFF
* Internal RAM at $800-$BFF
* Internal EEPROM at $1000-$1FFF
* Note: You can load in programs anywhere in the above mentioned areas but
should
*      only write to RAM (i.e. assume ROM and EEPROM are read only).

* Assemble using the "MiniIDE" 6812 Assembler found on our web site.
* Inside of MiniIDE go to "Build" => "Options" and make sure the "Generate
Listing
* File" option is checked.
*****

* ASSEMBLER EQUATE (Define) Statements
CONST  EQU    $8000  ;starting address for constants in memory
RAM1    EQU    $4000  ;starting address for variables in memory
RAM2    EQU    $5000  ;starting address for additional variables in memory

* DATA ALLOCATION SECTION - constants & variables
* Data can go before or after your program but should
* not in the middle of a program for clarity reasons.
        org    CONST  ;these are all assembler directives
num1    dc.b   $CD    ;define byter, single byte placed in memory starting at
$1000
```

---

## Using the 68HC12 Java Simulator

---

```
num2    dc.w    $1234 ;define word, double byte placed in memory
char    dc.b    'E'   ;define string, ASCII characters (one byte each)
        dc.b    'E'
        dc.b    'L'
        dc.b    '4'
        dc.b    '7'
        dc.b    '4'
        dc.b    '4'
```

```
        org     RAM1
results ds.b    3      ;locations $4000-$4002 reserved for future use
```

```
        org     RAM2
value1  dc.b    27     ;put 27 decimal in mem @ $5000
value2  dc.b    30     ;put 30 decimal in mem @ $5001
sum     ds.b    1      ;reserve a byte for a future sum
```

```
* Special Note: Assembler directives are used to place data
*               and variables into memory. They are not 6811
*               instructions and thus are not executed at run
*               time. When this program is loaded into memory,
*               the data (created above by the assembler) is also
*               copied down into memory.
```

\* Data Transfer Examples

\* Program #1 - moving a byte to a new location

```
        org     $1000 ;we will fill in comments in class
        ldaa   num1   ;
        staa   results ;
end1    bra     end1
```

\* Program #2 - moving a word (two bytes) to a new location

```
        org     $1100
        ldd   num2   ;
        std   results ;
end2    bra     end2
```

\* Program #3 - load immediatedata, transfer to another register & store in mem

```
        org     $1200
        ldaa   #%10101110 ;
        tab   ;
        stab  results ;
end3    bra     end3
```

\* Program #4 - move data using an index register

```
        org     $1300
        ldx   #num1   ;
        ldy   #results ;
        ldaa  0,x     ;
        staa  2,y     ;
end4    bra     end4
```

---

### Using the 68HC12 Java Simulator

---

\* ARITHMETIC & LOGIC EXAMPLES

\* Program #5 - adding (2) byte size numbers & saving the result

```
    org    $1400
    ldaa   value1      ;
    ldab   value2      ;
    aba
    staa   sum
end5    bra    end5
```

\* Program #6 - subtracting a byte in memory from an immediate data byte

```
    org    $1500
    ldab   #$20        ;
    subb   value2      ;
    ldx    #sum        ;
    stab   0,x
end6    bra    end6
```

\* Program #7 - summing words

```
    org    $1600
    ldd    num2        ;
    addd   num2        ;
    ldy    #results    ;
    std    9,y         ;
end7    bra    end7
```

\* Program #8 - bit wise ANDing example

```
    org    $1700
    ldaa   num1
    nega
    anda   value1      ;
    staa   sum
end8    bra    end8
```