

The THRSim11 68HC11 Simulator and Debugger

This pdf document is generated from the original .hlp file. The links in this document are not active.

This file contains information on the THRSim11 68HC11 microcontroller simulator and debugger. Select one of the following topics:



A quick introduction to get started. Read this before you try to use the THRSim11 68HC11 simulator or debugger.



Working with the simulator or target windows. Read this when you want to use the THRSim11 68HC11 simulator or debugger.



Editor and Assembler. Read this when you want to use the build in editor and THRAss11 assembler.



Debugging C/C++ programs which are compiled with the GNU Development Chain for 68HC11. Read this when you want to develop C/C++ programs for the 68HC11.



Debugging GNU as programs which are assembled with the GNU Development Chain for 68HC11. Read this when you are an experienced 68HC11 assembler programmer.



Overview of the buttons and menu items. Search here if you want to know the purpose of a button or a menu item of the THRSim11 68HC11 simulator and debugger.



Command line help. Search here if you want to know the purpose of a command-line command of the THRSim11 68HC11 simulator.



External boxes. Read this when you want to use some of the simulated external hardware.



Example programs. Here are some example assembler programs you can use to try the simulator.



Example C/C++ programs. Here are some example C/C++ programs you can use to try the simulator.



Example GNU as programs. Here are some example GNU as programs you can use to try the simulator.

EVB



Communication with your EVM or EVB (compatible) target board. You can use THRSim11 to download and debug programs on your target board.



Frequently asked questions.

Frequently asked questions.

A couple of frequently asked questions is answered on the following internet page: <http://www.hc11.demon.nl/thrsim11/faq.htm>

If you have other questions don't hesitate to ask me: harry@hc11.demon.nl (Harry Broeders).

More information.

There is more information available on the world wide web:

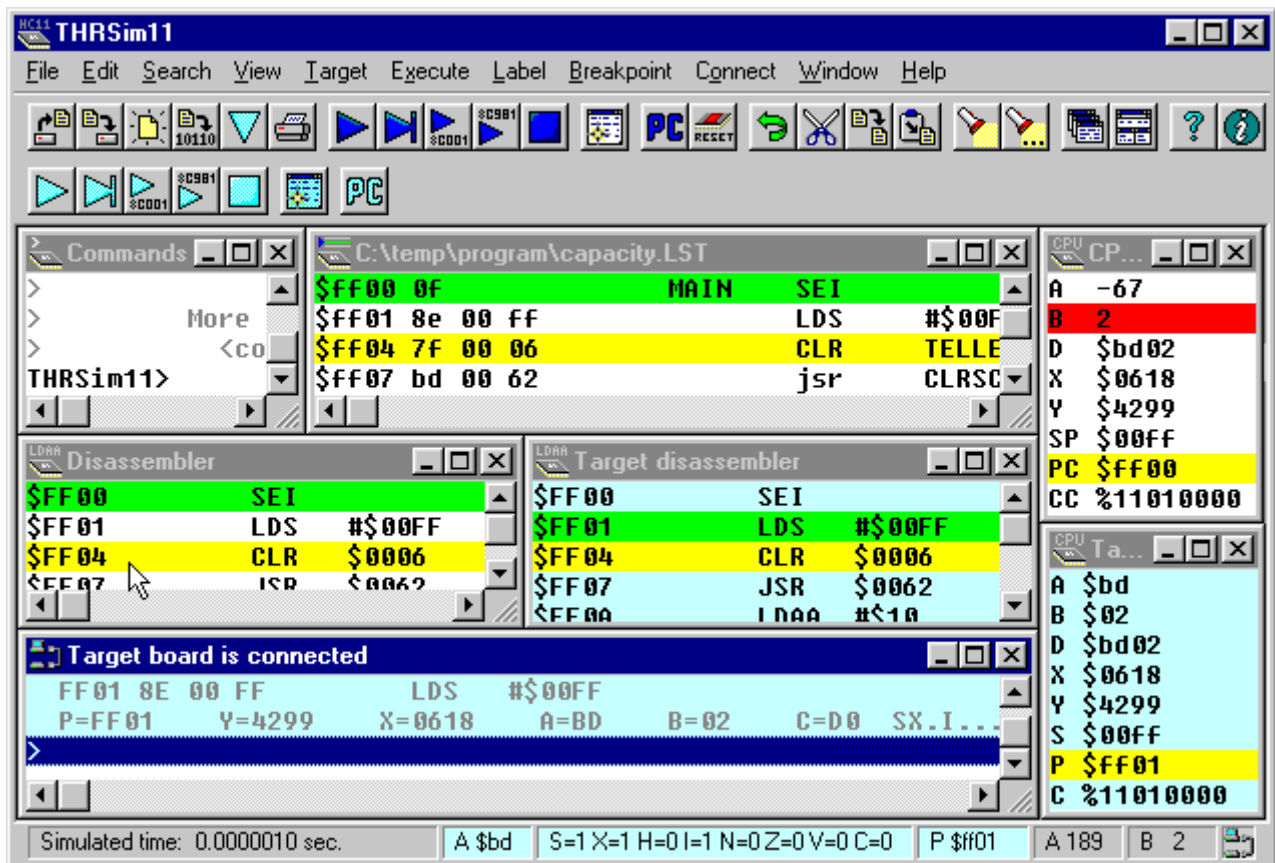
- You can visit my home page at <http://www.hc11.demon.nl/index.htm>
- You can visit the THRSim11 home page at <http://www.hc11.demon.nl/thrsim11/thrsim11.htm>

If you have any questions or problems please don't hesitate to contact me:

- You can send me e-mail at: harry@hc11.demon.nl (Harry Broeders).

Overview of buttons and menu items.

This part of the help file contains an overview of the buttons and menu items of the simulator program. The figure below displays a part of the THRSim11 window. The various buttons and menu items of the figure can be selected with the left mouse button to display the concerning information. An overview of all the menu items is also provided.



Open

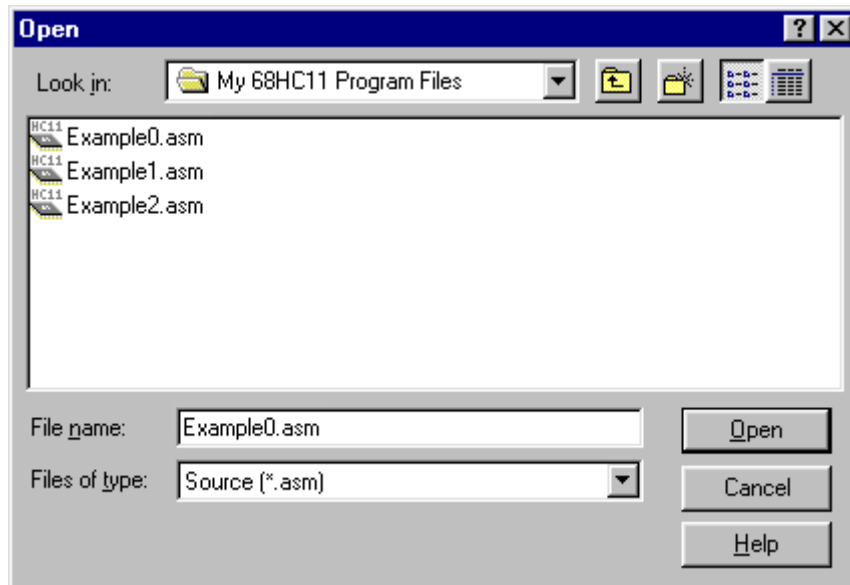
Using the  button or the Open item from the File menu the following file types can be loaded into THRSim11:

- .S19 files (machine code in Motorola S format)
- .ASM and .INC files (THRAss11 assembler source code in ASCII format).
- .LST files (the list file generated by the THRAss11 assembler).
- .CMD files (simulator commands).

When THRSim11 C support is installed you can also load the following file types:

- .C, .CPP and .H files (GNU gcc source code in ASCII format).
- .S files (GNU as assembler source code in ASCII format).
- makefile (GNU makefile).
- .LD (GNU ld script).
- .OUT or .ELF (GNU ELF executable code with DWARF debug information).







To open a file the following dialog box appears.




After selecting a file the OK button must be pressed to load the file.


The actions THRSim11 takes when you open a file depends on the file type and is described below. For most types THRSim11 loads the file into an external editor or into an edit window.

You can specify which editor you want to use in the THRSim11_options.txt file.


- A .S19 file will be directly loaded into the simulated memory of the 68HC11 microcontroller. This loaded program can be viewed in a disassembler window.
- An .ASM or .INC file will be loaded into an external editor or into an edit window. To Assemble the program (converting to 68HC11 machine language) which is loaded into the internal editor the  button or the Assemble item from the File menu can be used. When you use an external editor you can use the LSRC command to assemble and load the program. If you use the external editor SciTE you can just press F5 to assemble and load the program which you are editing. An assembled and loaded program can be executed with the  button or the Run item from the Execute menu.
- When a .LST file is loaded a dialog box is displayed giving you the choice to load the labels from the list file's label table in the label window or to load the entire list file in a read-only edit window.
- When a .CMD file is loaded a dialog box is displayed giving you the choice to execute the simulator commands from file or to load the file in an edit window. You can also use the external editor to edit .CMD files.
- A .C, .CPP or .H file will be loaded into an external editor or into an edit window. To Compile the program (converting to 68HC11 machine language) which is loaded into the internal editor the  button or the Compile item from the File menu can be used. When you use an external editor you can use the LELF command to load the compiled program. If you use the external editor SciTE you can just press F5 to compile and load the program which you are editing. An compiled and loaded program can be executed with the  button or the Run item from the Execute menu.
- A .S will be loaded into an external editor or into an edit window. To Assemble the program (converting to 68HC11 machine language) which is loaded into the internal editor the  button or the Assemble item from the File menu can be used. When you use an external editor you can use the LELF command to load the assembled program. If you use the external editor SciTE you can just press F5 to assemble and load the program which you are editing. An assembled and loaded program can be executed with the  button or the Run item from the Execute menu.
- A .LD file is loaded into an external editor or into an edit window.
- When an .OUT or .ELF file is loaded a C/C++ list window and C/C++ variables window is opened to

show the program source and variables. A loaded program can be executed with the  button or the Run item from the Execute menu.


Save



With the  button or the Save item from the File menu a file can be stored on disk under a specific name. (The name must be specified with the Save as item from the File menu) When no name is specified you will be asked to type (or select) a new name.




Save as

Using the Save As item from the File menu the contents of the edit window can be stored on disk under a specific name. The name of the file will be shown in the title of the edit window. After using this menu item the file can be quickly stored with the  button or the Save item from the File menu.

New

With the  button or the New item from the File menu the simulator will create a new edit window.

A 68HC11 assembler program can be written in this edit window. Using the Save as item from the File menu the contents of the edit window can be stored. To Assemble the program (converting to 68HC11 machine language) the  button or the Assemble item from the File menu can be used. Execute the program with the  button or the Run item from the Execute menu.

You can also write a C/C++ or GNU as program in this edit window. Using the Save as item from the File menu the contents of the edit window must be stored with the extension .C, .CPP or .S. After you have saved the file with the extension .C or .CPP you can use the  button or the Compile item from the File menu to compile and load the program. A simple makefile and ld script are automatically created when needed. After you have saved the file with the extension .S you can use the  button or the Assemble item from the File menu to assemble and load the program. A simple makefile and ld script are automatically created when needed. Execute the program with the  button or the Run item from the Execute menu.

A **simpler** way to create GNU gcc or as source code is to use the New GNU gcc or as source item from the File menu.


New GNU gcc or as source

With the New GNU gcc or as source item from the File menu the simulator will create a new GNU gcc or as source file. It also creates an appropriate makefile and ld script if needed.

[More information about THRSim11 GNU gcc support can be found here.](#)
[More information about THRSim11 GNU as support can be found here.](#)

[More help is available here \(in HTML\).](#)


Assemble or Compile

With the  button or the Assemble/Compile item from the File menu the contents of an edit window can be converted to machine language.

When the edit window contains a .ASM file THRAss11 will be called to assemble the file. When the file contains errors these will be displayed in a list window. Double clicking on an error message in this list window will select the source of the error in the edit window. When the file contains no errors a list window will be opened and the machine code will be loaded in the simulated memory. The program can now be executed or stepped. Depending on the assembler options you selected the listing and the machine codes are also saved on disk in a .LST and a .S19 file.

When the edit window contains a .C, .CPP, or .S file the !make command is called to compile or assemble the file. When the file contains errors these will be displayed in the command window. When the file contains no errors a list window will be opened and the machine code will be loaded in the simulated memory. A variables window will also be opened when appropriate. The program can now be executed or stepped.

Download to Target Board


With the  button or the Download to Target Board item from the File menu the contents of a S19 or ELF file can be loaded into your target board. THRSim11 can communicate with the Motorola EVM and EVB boards or with any other board running the BUFFALO monitor program. This monitor program can be downloaded (for free) from the Motorola website.

When there is currently no edit window or list window open a file open dialog is selected so you can select a S19 or ELF file from disk. If there is a THRAss11 list window or C/C++ list window open the corresponding S19 or ELF file is loaded into the target board. If there is an edit window open this edit window is assembled or compiled (which opens a list window) and the S19 or ELF file is loaded into the target board.

If the communication with the target fails an error message is displayed. You can monitor the communication with the target board by opening a Target Command Window from the Target menu.

If the communication fails and you are sure the serial cable is connected you can try to change the Target Communication Options from the Target menu.

Print

With the  button or the Print item from the File menu the contents of the selected edit window or list window will be printed on the selected printer. The formatting of the list file can be changed using the Assembler Options.

Options

The Options item from the File menu consists of five (or six) sub-items which are:

Assembler options

Simulator options

Target Communication options

Statusbar options

Font options

Memory Configuration options

Assembler options

The assembler **input** options are:

- **Default numeral system:** When a number is typed without numeral system indicator this numeral system is used by default. The default numeral system can be set to *decimal* or *hexadecimal*.
- **Allow strings in FCB directive:** When this option is checked the FCB directive may contain character strings. Every character in the string is converted to a byte and stored in memory. If this option is **not** checked the FCB directive may only contain single bytes and you have to use the FCC directive to store a string in memory.

The assembler **output** options are:

- **Listfile:** Select the *listfile* option to generate a file containing the listing.
- **Include label table:** If the *listfile* option is selected you can select the *include label table* option to include a table containing all labels used in the program.
- **Expand included files:** If the *listfile* option is selected you can select the *expand included files* option to include the (listing of) the included files in this listing.
- **S19-file:** Select the *S19-file* option to generate a file containing the machine codes in Motorola S19 format.

The More listfile options button can be used to set some format options for the listfile if the *listfile* option is selected.

These options are self explanatory

The More S19-file options button can be used to set some format options for the S19-file if the *S19-file* option is selected.

These options are also self explanatory

Simulator options

These options are:

- **Remove currently set labels before loading:** When this option is selected the currently set labels are removed when a 68HC11 program is loaded. This makes it possible to only use labels that are defined in the last loaded 68HC11 program.
- **Reset 68HC11 after loading:** After loading a 68HC11 program the simulator directly resets the simulated 68HC11 chip and therefore loads the Program Counter with the address contained in the reset vector. This is very handy if your program sets the reset vector to the start of your program.
- **Look for a .LST or .MAP file to load new labels:** Automatically loads a .MAP or .LST file when a 68HC11 machine code program (.S19 file) is loaded. (Instead of asking for confirmation.) When this option is enabled the appropriate labels are loaded if they are available in a .LST or .MAP file. (.MAP files are generated by the IASM assembler that was formerly used at the TH Rijswijk and are probably of no interest to you.)
- **Confirm reset:** If this option is selected the simulator asks for confirmation when you try to reset the simulated 68HC11 chip.
- **Give warning if writing to ROM:** If this option is selected the simulator gives a warning if the program tries to write to ROM (Read Only Memory).
- **Give warning if writing to unused memory:** If this option is selected the simulator gives a warning if the program tries to write to an address where no memory is located. Use the Memory, Memory Map menu item from the View menu to display the current memory map.
- **Give warning if reading from unused memory:** If this option is selected the simulator gives a warning if the program tries to read from an address where no memory is located. Use the Memory, Memory Map menu item from the View menu to display the current memory map.

- [Give warning if reading from uninitialised vector](#): If this option is selected the simulator gives a warning if the program reads the address \$FFFF from an interrupt vector location. You probably forgot to initialise this vector address.
-

Target Communication options

THRSim11 can communicate with the Motorola EVM and EVB boards or with any other board running the BUFFALO monitor program.

You can set the following communication options:

- [Name](#): The name of the communication port to use can be selected from this drop down list.
- [Baudrate](#): The speed of the communication can be selected from this drop down list. Most EVB boards work at 9600 Baud. The EVM board can work at 19200 Baud. You don't need to make any changes to your EVM board to use this faster baudrate because the EVM board automatically detects the baudrate.
- [Parity](#): The parity used in communication can be selected from this drop down list.
- [Stop bits](#): The number of stop bits used in communication can be selected from this drop down list.
- [Character size](#): The character size used in communication can be selected from this drop down list.

Other target options are:

- [Remove currently set labels before downloading](#): When this option is selected the currently set labels are removed when a 68HC11 program is downloaded into the target board. This makes it possible to only use labels that are defined in the last downloaded 68HC11 program.
 - [Look for a .LST or .MAP file to load new labels](#): Automatically loads a .MAP or .LST file when a 68HC11 machine code program (.S19 file) is downloaded. (Instead of asking for confirmation.) When this option is enabled the appropriate labels are loaded if they are available in a .LST or .MAP file.
-

Statusbar options

With this submenu item the statusbar can be configured. If you choose Menu Font the font used to display the statusbar will be the same as the font used to display the menu. If you want to change the font which is used to display the statusbar you have to choose Window Font. If you select Window Font the statusbar font will be the same as the font used in all THRSim11 windows, this font can be changed by using the [Font options](#).

Font options

With this submenu item the currently used font can be altered. Only so called fixed pitch fonts can be used.

Memory configuration options

THRSim11 is closed when the Memory Configuration Tool is started. THRSim11 is (re)started when the Memory Configuration Tool is closed.

These options are saved in the THRSim11_options.txt file. [A description of all memory configuration options can be found here.](#)

Exit

With the Exit item from the [File menu](#) the simulator can be closed.

Run

You first have to [assemble or compile](#) a program or open the [disassembler](#) before you can run a program.

By selecting the Run item from the [Execute menu](#) the 68HC11 simulator runs the program starting at the address pointed to by the Program Counter. There are several ways to start the simulator:

- Using the [button on the toolbar](#).
 - By using the [command line prompt](#). Typing the command [run](#) will cause the simulator to run the program.
 - Using the menu [Execute](#). And selecting the item Run.
 - Using the pop up menu ([right mouse click or Alt+F10](#)) of the [list window](#). And selecting the item Run.
 - Just press F9.
-

Step

You first have to [assemble or compile](#) a program or open the [disassembler](#) before you can step through a program.

With this command the simulator executes a single instruction (pointed to by the Program Counter). After executing the instruction the Program Counter will be set to the next instruction. If the [THRAss11 list window](#) is active then the step function will execute one 68HC11 **instruction**. If the [C/C++ list window](#) is active the step function will execute one **line** of C/C++ code. If you want to step through each 68HC11 instruction when the C/C++ list window is active you can open the C/C++ line by clicking on the + sign preceding the line. You can also open the [disassembler window](#) and step through each 68HC11 instruction there.

There are several ways to execute the step function:

- Using the [button on the toolbar](#).
 - Using the [command line prompt](#). And typing the command [step](#).
 - Using the menu [Execute](#). And selecting the item Step.
 - Using the pop up menu ([right mouse click or Alt+F10](#)) of the [list window](#). And selecting the item Step.
 - Press the Enter key when the [list window](#) is active.
 - Just press F7 or the space bar.
-

Run until

You first have to assemble a program or open the disassembler before you can run a program.

By selecting the Run Until item from the Execute menu a dialog window will appear where you can enter an address. The simulation will run until the Program Counter equals the entered address. There are several ways to perform this task:

- Using the [button on the toolbar](#).
 - Using the [command line prompt](#). And typing the command rununtil.
 - Using the menu Execute. And selecting the item Run Until.
 - Using the pop up menu (right mouse click or Alt+F10) of the list window. And selecting the item Run Until.
 - Just press F4.
-

Run from

You first have to assemble a program or open the disassembler before you can run a program.

By selecting the Run From item from the Execute menu a dialog window will appear where you can enter an address. The simulation will be started from the entered address. There are several ways to perform this task.:

- Using the [button on the toolbar](#).
 - By using the [command line prompt](#). Typing the command run followed by an address.
 - Using the menu Execute. And selecting the item Run From.
 - Just press F6.
-

Stop

By selecting the Stop item from the Execute menu the simulator will be stopped. There are several ways to perform this task:

- Using the [button on the toolbar](#).
 - Using the [command line prompt](#). By typing the command stop
 - Using the menu Execute. And selecting the item Stop.
 - Using the pop up menu (right mouse click or Alt+F10) of the list window. And selecting the item Stop.
 - Just press F2.
-

Set PC

The [button](#) makes it possible to give a new value to the Program Counter. After pressing this button a new Program Counter value can be assigned.

Target Go

You first have to [download](#) a program before you can execute it on your target board.

By selecting the Target Go item from the [Execute menu](#) the program runs on the target board starting at the address pointed to by the target board's Program Counter. There are several ways to start a program on the target board:

- [Using the button on the target toolbar.](#)
 - [By using the target command window.](#) Typing the [EVM command G](#) or [EVB command G](#) will cause the target board to execute the program.
 - [Using the menu Execute.](#) And selecting the item Target Go.
 - [Using the pop up menu \(right mouse click or Alt+F10\) of the list window.](#) And selecting the item Target Go.
 - [Just press Ctrl+F9.](#)
-

Target Step

You first have to [download](#) a program before you can execute it on your target board.

With this command the target board executes a single instruction (pointed to by the target board's Program Counter). After executing the instruction the Program Counter will be set to the next instruction. If the [THRAss11 target list window](#) is active then the step function will execute one 68HC11 **instruction**. If the [C/C++ target list window](#) is active the step function will execute one **line** of C/C++ code. If you want to step through each 68HC11 instruction when the C/C++ list window is active you can open the C/C++ line by clicking on the + sign preceding the line. You can also open the [target disassembler window](#) and step through each 68HC11 instruction there.

There are several ways to execute the target step function:

- [Using the button on the target toolbar.](#)
 - [By using the target command window.](#) Typing the [EVM command T](#) or [EVB command T](#) will cause the target board to execute one instruction.
 - [Using the menu Execute.](#) And selecting the item Target Step.
 - [Using the pop up menu \(right mouse click or Alt+F10\) of the list window.](#) And selecting the item Target Step.
 - [Press the Ctrl+Enter key when the list window is active.](#)
 - [Just press Ctrl+F7 or the space bar.](#)
-

Target Go until

You first have to download a program before you can execute it on your target board.

By selecting the Target Go Until item from the Execute menu a dialog window will appear where you can enter an address. The program will run on the target board until the Program Counter equals the entered address. There are several ways to perform this task:

- Using the [button on the target toolbar](#).
 - Using the menu Execute. And selecting the item Target Go Until.
 - Using the pop up menu (right mouse click or Alt+F10) of the list window. And selecting the item Run Until.
-

Target Go from

You first have to download a program before you can execute it on your target board.

By selecting the Target Run From item from the Execute menu a dialog window will appear where you can enter an address. The program will be started on your target board from the entered address. There are several ways to perform this task.:

- Using the [button on the target toolbar](#).
 - Using the menu Execute. And selecting the item Target Run From.
-

Target Stop

A program on your target board can **only** be stopped by pressing RESET or ABORT on your target board! When a program is running on the target board THRSim11 waits until it stops. The program on the target board can stop for example because a breakpoint is reached.

While the program is running on the target the following dialog box is shown:

Or when the target command window is open this message is displayed:

By selecting the Target Stop item from the Execute menu the connection with the target will be closed but the program will continue to run on the target board!. There are several ways to perform this task:

- Using the [button on the target toolbar](#).
 - Using the [target command window](#). By typing enter twice.
 - Using the menu Execute. And selecting the item Target Stop.
 - Using the pop up menu (right mouse click or Alt+F10) of the list window. And selecting the item Target Stop.
 - Just press Ctrl+F2.
-

Set Target PC

The [button](#) makes it possible to give a new value to the target board's Program Counter. After pressing this button a new Program Counter value can be assigned.

Editor and assembler

The THRSim11 simulator contains an integrated editor and assembler. This makes it needless to separately use a simulator, assembler and editor. 68HC11 assembly programs can be written, assembled and tested all in the simulator environment.

You can also use an external editor in combination with THRSim11. If you installed THRSim11 with C support the external editor SciTE will be installed on default. This editor will be used on default to edit .C, .CPP, .H, .S, .LD, and makefile files. All other files will be automatically opened with the internal editor. If you want to change this default behaviour or if you want to use an other editor you have to change the editor options in the THRSim11_options.txt file. More information about the external editor can be found [here](#) (in HTML).

You can use THRAss11 to assemble 68HC11 source codes but you can also use the GNU assembler as.

When to use the internal assembler THRAss11 or the GNU as assembler?

The internal assembler THRAss11 is an absolute assembler. This means that you have to provide the addresses where the program and data are to be located in the assembler source code (using the ORG directive). The GNU assembler as is a relocatable assembler. This means that you need a linker (the GNU linker ld) to generate executable code and you need to provide a linker script to define the addresses where the program and data are to be located in memory.

THRAss11 is much simpler than GNU as and is therefore recommended for programmers starting to learn 68HC11 assembler programming. THRAss11 has also better error diagnostics. GNU as has many more features than THRAss11 and is therefore recommended for more experienced 68HC11 assembler programmers.

When to use the internal or internal editor for THRAss11?

The internal editor is more appropriate when you are just starting to learn the 68HC11 assembler language. When you are a more experienced developer you probably need more advanced edit functions that are not provided by the internal editor of THRSim11.

When to use the internal or internal editor for GNU source files?

GNU source files can also be edited with the internal editor but when you compile or assemble a .GNU gcc or as source file which contains errors there is no easy way to locate the offending lines. Most external editors (e.g. SciTE) provide support for GNU gcc and as and provide links between the error report and the source codes. Therefore I recommend using an external editor to edit GNU gcc or as source files. But if you prefer the internal editor for some reason you can use it. The internal editor recognises GNU gcc and as source files and calls the !make command to compile or assemble the file and load the resulting ELF file (a.out) into the simulated memory.

Editor

The internal editor contains all commonly used commands like:

Undo
Cut
Copy

Paste
Delete

Find
Replace
Find next

When the New or Open (.ASM or .INC file) item of the File menu is selected a file will be opened. After the program is typed in or altered it can be stored with the Save as item from the File menu. When you have assigned a name to the file it can be stored quickly with the Save item from the File menu.

Currently the editor can not handle files containing more than 30,000 characters.

If you want to use source files that are bigger than 30,000 characters you have 2 options:

- Use an external editor and assemble (and load) the file with the Load source command
- Divide your source in multiple source files and use the #INCLUDE directive to include all subfiles into one main file.

Edit

Copy

This button or the Copy item from the Edit menu places selected text from the edit window on the clipboard. (The old contents of the clipboard will be overwritten.) You can also use the Ctrl+C key to copy text.

Paste

This button or the Paste item from the Edit menu inserts the contents of the clipboard at the current cursor position. This button and menu item will be disabled when the clipboard is empty. You can also use the Ctrl+V key to paste text.

Cut

This button or the Cut item from the Edit menu clears the selected area and places it on the clipboard. You can also use the Ctrl+X key to cut text.

Undo

This button or the Undo item from the Edit menu undoes the last edit action. You can also use the Ctrl+Z key to undo the last edit action.

Delete

The Delete item from the Edit menu deletes the selected text. (It will not be stored on the clipboard, text deleted this way can by no means be restored.) You can also use the Del key to delete text.

Search

Using the Search buttons or Search menu it is possible to search for a specific piece of text. The Search menu items are:

Find

This button or the Find item from the Search menu locates a specific piece of text.

Replace

The Replace item from the [Search menu](#) finds a specific text and replaces it with an other text.

Find Next

This button or the Find Next item from the [Search menu](#) finds the next occurrence of the last searched for text.

Assembler

The build in assembler assembles all 68HC11 instructions with their appropriate addressing modes. [Click here for a description of the assembler syntax and the assembler directives](#)

There are several ways to assemble the program:

- The button on the [toolbar](#)
- The item [Assemble](#) from the [File menu](#).
- The item Assemble on the menu that pops up when the mouse pointer is positioned above the edit window and the right mouse button is clicked. The following pop-up menu will appear:

Select the pop-up menu item Assemble.

When the program is assembled there are several possibilities:

- [The program contains errors.:](#)

Program errors are displayed in a [list window](#). To find a specific error in the editor select the error message in the [list window](#) and press return. The editor will automatically position the cursor on the error, so you can correct it and assemble again. Depending of the [assembler options](#) you selected the listing is also saved on disk in a .LST file.

- [The program contains no errors:](#)

A [list window](#) will be opened and the machine code will be loaded in the simulated memory. The program can now be [executed](#) or [stepped](#). Depending of the [assembler options](#) you selected the listing and the machine codes are also saved on disk in a .LST and a .S19 file.

Disassembler

If you use the build in assembler you will not use this window. You use the [list window](#) instead.

If you don't use the build in assembler but directly load machine codes (in Motorola S format) you will be using this window. With the Disassembler item from the [View menu](#) the [disassembler window](#) can be opened. This window contains the disassembled code of the memory contents. For example:

The current position of the Program Counter is displayed with a green bar. The Program Counter can be altered in different ways.

- [Within the disassembler window.](#) [Select a line](#) within the disassembler window. Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.)

Select the menu item Place PC on.

- [On the command line prompt](#). Typing a new value for the Program Counter. For example: "PC C000"
 - [By clicking the button](#). In the dialog box a new value can be assigned to the Program Counter.
 - [Within the CPU registers window](#). Double clicking on the row containing the Program Counter let you enter a new value. You can also just select the row containing the Program Counter and type in a new value.
-

Target Disassembler

If you use the build in assembler you will not use this window. You use the [list window](#) instead.

If you don't use the build in assembler but directly download machine codes (in Motorola S format) into your target board you will be using this window. With the Target Disassembler item from the [Target menu](#) the [target disassembler window](#) can be opened. This window contains the disassembled code of the memory contents of the target board. For example:

The current position of the target board's Program Counter is displayed with a green bar. The Program Counter can be altered in different ways.

- [Within the target disassembler window](#). [Select a line](#) within the target disassembler window. Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Place Target PC on.
 - [On the target command line prompt](#). Typing a new value for the Program Counter. For example: "PC C000"
 - [By clicking the button](#). In the dialog box a new value can be assigned to the Program Counter.
 - [Within the target CPU registers window](#). Double clicking on the row containing the target Program Counter let you enter a new value. You can also just select the row containing the target Program Counter and type in a new value.
-

Label

A **label** is a symbolic name you can associate to a value or address to make it easier to remember or identify. A label must start with a letter, underscore(_), or dot(.) and can further contain letters, digits, underscores (_), and dots (.).

The following label menu items are available:

[Set](#)
[Remove](#)
[Remove all](#)
[Set standard label](#)
[Window](#)

The following label commands are available:

[Set label](#)
[Delete label](#)
[Set standard labels](#)
[Delete standard labels](#)
[Delete all labels](#)
[Print label](#)
[Print all labels](#)

Set a label

Use the Set item from the Label menu to add a new label to the label table. After choosing this menu item a dialog box will appear where a name and a value for the new label can be entered. Labels can also be added directly in the label window.

Remove a label

Use the Remove item from the Label menu to remove one or more labels from the label table. After choosing this menu item a dialog box will appear where a number of labels can be selected. Labels can also be removed directly in the label window. You can remove all labels at once by using the Remove All item from the Label menu.

Remove all labels

Use the Remove All item from the Label menu to remove all existing labels from the label table. If you want to remove some but not all labels use the Remove item from the Label menu.

Standard labels

Use the Set Standard Labels item from the Label menu to let the simulator add the standard labels to the label table. Standard labels (names for I/O registers etc.) are used in the official 68HC11 documentation by Motorola.

Label window

This window gives an overview of all the labels currently present in the label table. It is possible to change the value of a label. Just select the label and type in a new value. It is also possible to add a new label by using the Ins key or by using the menu that pops up when you press the right mouse button. You can remove one or more labels by selecting them and press the Del key or by using the menu that pops up when you press the right mouse button. If all the labels are deleted the label window is closed and the Window item of the Label menu is disabled.

Breakpoint

A breakpoint is a sort of Boolean condition. If this condition **becomes** true the simulator stops executing instructions.

The condition consists of:
<name> <operator> <value>.

The simulator stops executing instructions if the condition <name> <operator> <value> **becomes** true.

A breakpoint can also optionally use a <count>. If you specify a count value. A counter will be incremented each time the breakpoint condition **becomes** true. Only when this counter reaches the entered count value the simulation of instructions will be stopped.

Examples of breakpoint conditions:

A == \$03	stops executing if register A becomes equal to \$03.
A > \$03	stops executing if register A becomes higher than \$03.
A == \$03 count 5	stops executing if register A becomes equal to \$03 for the fifth time.
A & \$03	stops executing if register A AND \$03 becomes not equal to \$00. Or formulated differently: stops if bit1 and/or bit0 of register A is being set.
M \$0EBD = \$a	stops executing when \$0A is written to memory location \$0EBD.

The following breakpoint menu items are available:

Set

Remove

Remove all

Window

The following breakpoint commands are available:

Set breakpoint

Delete breakpoint

Delete all breakpoints

Print breakpoint

Print all breakpoints

B / SB

DB

LB

See also:

Set a breakpoint on a line.

Set a breakpoint in a list window.

Set a breakpoint in a disassembler window.

Remove a breakpoint from a line.

Remove a breakpoint from a list window.

Remove a breakpoint from a disassembler window.

Target Breakpoint

In the target board several breakpoints can be set on the Program Counter. If the Program Counter **becomes** equal to one of these breakpoints (addresses) the target board stops executing the downloaded program. In the simulator breakpoints can also be set on other registers and even on memory locations and pins. This is **not** possible in the target board.

A target breakpoint can **not** use a <count>. If you have an EVM board you can use the EVM P command to set a breakpoint <count>.

The following target breakpoint menu items are available:

Target Set

Target Remove

Target Remove all

Target List

The following EVM breakpoint commands are available:

EVM BR command to set a target breakpoint or list all target breakpoints.

EVM NOBR command to remove one or all target breakpoints.

EVM P command to proceed after a target breakpoint.

The following EVB breakpoint commands are available:

EVB BR command to set or remove a target breakpoint or list all target breakpoints.

EVB P command to proceed after a target breakpoint occurred.

See also:

Set a breakpoint in a list window.

Set a breakpoint in a disassembler window.

Remove a breakpoint from a list window.

Remove a breakpoint from a disassembler window.

Set a breakpoint

Use the Set item from the Breakpoint menu to add a new breakpoint. After choosing this menu item a dialog box will appear where a new breakpoint and its conditions can be entered. Breakpoints can also be added directly in the breakpoint window.

Remove a breakpoint

Use the Remove item from the Breakpoint menu to remove one or more breakpoints. After choosing this menu item a dialog box will appear where a number of breakpoints can be selected. Breakpoints can also be removed directly in the breakpoint window. You can remove all breakpoints at once by using the Remove All item from the Breakpoint menu.

Remove all breakpoints

Use the Remove All item from the Breakpoint menu to remove all existing breakpoints. If you want to remove some but not all breakpoints use the Remove item from the Breakpoint menu.

Breakpoint window

This window gives an overview of all the breakpoints currently set. It is possible to change the value or conditions of a breakpoint. Just select the breakpoint and type in a new value or condition. It is also possible to add a new breakpoint by using the Ins key or by using the menu that pops up when you press the right mouse button. You can remove one or more breakpoints by selecting them and press the Del key or by using the menu that pops up when you press the right mouse button. If all the breakpoints are deleted the breakpoint window is closed and the Window item of the Breakpoint menu is disabled.

Set a target breakpoint

Use the Target Set item from the Breakpoint menu to add a new target breakpoint. After choosing this menu item a dialog box will appear where a new breakpoint address can be entered.

Remove a target breakpoint

Use the Target Remove item from the Breakpoint menu to remove one or more target target breakpoints. After choosing this menu item a dialog box will appear where a number of breakpoints can be selected. You can remove all target breakpoints at once by using the Target Remove All item from the Breakpoint menu.

Remove all target breakpoints

Use the Target Remove All item from the Breakpoint menu to remove all existing target breakpoints. If you want to remove some but not all breakpoints use the Target Remove item from the Breakpoint menu.

Target breakpoint list

Use the Target List item from the Breakpoint menu to get an overview of all the target breakpoints currently set in the target board.

Windows arranging

Cascade

This button or the Cascade item from the Window menu cascades the open windows. (The windows are arranged as overlapping windows.) You can also use the Shift+F5 key to cascade the windows.

Tile

This button or the Tile item from the Window menu tiles the open windows. (The windows are arranged vertically as non overlapping windows.) You can also use the Shift+F4 key to tile the windows.

Tile Horizontal

The Tile horizontal item from the Window menu tiles the open windows horizontally. (The windows are arranged horizontally as non overlapping windows.)

Arrange Icons

The Arrange Icons item from the Window menu neatly arranges the minimised (iconised) windows.

Close All

The Close All item from the Window menu closes all open windows.

Help

This help file is opened when you click the button or select the Contents item from the Help menu.

Information

Information about the simulator is displayed when you click the [button](#) or select the About item in the [Help menu](#).

Toolbar

The toolbar can be hidden by selecting the Toolbar option in the [View menu](#). The toolbar is normally positioned at the top of the window but it can be dragged to an other position. Click on the buttons in the following figure to get more information about the tools.

Target Toolbar

The target toolbar can be hidden by selecting the Target Board Toolbar option in the [Target menu](#). The target toolbar is normally positioned beneath the regular [toolbar](#) but it can be dragged to any other position. Click on the buttons in the following figure to get more information about the target tools.

Statusbar

The statusbar can be hidden by selecting the Statusbar option in the [View menu](#). The toolbar is positioned at the bottom of the window.

The statusbar provides short help messages when you browse the main menu or toolbar.

The statusbar also displays several CPU registers. You can set a new [value](#) to one of these registers by double clicking the displayed value.

When THRSim11 has an active connection with a target board the CPU registers from the target can also be shown in the statusbar. These target registers have a light blue background. There is also a small icon on the statusbar showing the state of the communication with the target board. You can click this icon to connect or disconnect to the target board.

You select the CPU registers you want to see on the statusbar by using the [Options, Statusbar](#) menu item from the [File menu](#).

Double clicking the CCR flags in the statusbar will open the following dialog:

A checked box indicates that the corresponding flag is set.

Memory Map

The Memory, Memory Map menu item from the [View menu](#) will display the current memory map. [The memory map can be changed by altering the THRSim11_options.txt initialisation file.](#)

Memory Warning

This dialog appears when the program writes to ROM (Read Only Memory) or when the program reads from or writes to an address where no memory is located. Use the Memory, [Memory Map](#) menu item from the [View menu](#) to display the current memory map. [The memory map can be changed by altering the THRSim11_options.txt initialisation file.](#)

You can check the box to disable this specific warning. You can enable this warning again by using the Options, [Simulator Options](#) menu item.

THRAss11 Assembler

THRAss11 was designed and implemented by former THR student (graduated in 1996) Rob van Beek. The assembler is part of THRSim11. The assemblers accept options from the dialog box opened via the [File|Options|Assembler](#) menu item.

The generated machine code is loaded into the simulated memory. The S1 formatted object file is placed in file filename.S19 if the generate S19 file option is enabled. The listing and error messages are written to a specific [list window](#). The listing and error messages are also placed in the file filename.lst if the generate list file option is on.

The listing file contains the address and bytes assembled for each line of input followed by the original input line (unchanged, but moved over to the right somewhat). If an input line causes more than 5 bytes to be output (for example, a long FCC directive), additional bytes are listed on succeeding lines with no address preceding them.

[THRAss11 syntax.](#)

[THRAss11 errors.](#)

[THRAss11 implementation notes.](#)

Assembler Syntax

The general format of a line assembler code is as follows:

LABEL MNEMONIC OPERAND COMMENT

Assembler Label Syntax

A label is defined as a string of characters with a non-initial digit. The string of characters may be from the set: [a-z][A-Z]_[0-9] (. and _ count as non-digits). All characters of a symbol are significant, with upper and lower case characters being non-distinct. There is no maximum number of characters in a symbol. The symbol table has no limits (only the amount of RAM you have).

A symbol starting in the first column is a label and may be followed by a :. This : is not part of the label and is replaced by a blank by the THRSim11 assembler. A label may appear on a line by itself and is then interpreted as: Label EQU *

Assembler Mnemonic Syntax

A symbol preceded by at least one whitespace character is a mnemonic. Upper case characters in this field are converted to lower case before being checked as a legal mnemonic. Thus nop, NOP and even NoP are recognised as the same mnemonic.

Note that register names that sometimes appear at the end of a mnemonic (e.g., NEGA or STX) must not

be separated by any whitespace characters. Thus CLRA means clear accumulator A, but that CLR A means clear memory location A.

All the valid 68HC11 instruction mnemonics can be used.

Assembler directives (also called pseudo-ops) can also be used and will cause the assembler to perform a specific action. The only pseudo-ops supported are: ORG, FCC, FDB, FCB, EQU, RMB, END, and #INCLUDE.

Assembler directives

The build in assembler supports the following assembler directives:

<u>END</u>	end.
<u>EQU</u>	equate.
<u>FCB</u>	form constant bytes.
<u>FCC</u>	form constant characters.
<u>FDB</u>	form double bytes.
<u>ORG</u>	origin.
<u>RMB</u>	reserve memory bytes.
<u>#INCLUDE</u>	include source file.

It also supports the following aliases:

DB	define bytes	is an alias for <u>FCB</u> .
DW	define words	is an alias for <u>FDB</u> .
DS	define storage	is an alias for <u>RMB</u> .
\$INCLUDE	include	is an alias for <u>#INCLUDE</u> .

An **address** is a 16 bits value that identifies a memory location within the 68HC11 memory map (\$0000 - \$FFFF)

A **label** is a symbolic name you can associate to a value or address to make it easier to remember or identify. A label must start with a letter, underscore(_), or dot (.) and can further contain letters, digits, underscores (_), and dots (.).

A **value** can be:

- a **literal constant**:

% binary	e.g. %00101011	= 43 decimal
@ octal	e.g. @53	= 43 decimal
\$ hexadecimal	e.g. \$2B	= 43 decimal
& decimal	e.g. &43	= 43 decimal
' ASCII	e.g. 'A	= 41 hexadecimal
" double ASCII	e.g. "AB	= 4142 hexadecimal

the default (binary, octal, decimal or hexadecimal) depends on the context and on the assembler option you choose.

- a **register name, pin name or label**.
- an **expression**:

- (expression) or {expression} can be used to change the build in operator precedence. For example:

2+3*4	= 14 decimal
(2+3)*4	= 20 decimal
{2+3}*4	= 20 decimal

- **unary operators:** (highest precedence):

+	positive
-	negative = two's complement
~	NOT = one's complement

- **binary operators:** (from high to low precedence)

* / \	times DIV MOD
+ -	plus minus
&	AND
^	EXOR
! or	OR

ORG directive

Syntax: [<label>] ORG <address>

The ORG (origin) directive sets the starting location <address> where the instructions or directives that follow this directive are placed in memory. The use of <label> is optional. If a label is used its value is set to <address>. You can use as many ORG directives as you like in one assembler program. If you don't use an ORG directive the starting location depends on an assembler option.

END directive

Syntax: [<label>] END

The END directive marks the end of the assembler code. All text following this directive is ignored by the assembler. The use of <label> is optional. You can use only one END directive. If you don't use an END directive the assembler assembles the entire file.

EQU directive

Syntax: <label> EQU <value>

The EQU (equates) defines a <label> with its <value>. If you use this label in the following instructions or directives this label is replaced by its value when the assembler assembles the file. The use of <label> is mandatory.

FCB directive

Syntax: [<label>] FCB <value>,<value>,...,<value>

The FCB (form constant bytes) directive forms a table with constant byte values in memory. You can provide one or more values separated by commas (without a space). Each <value> should fit into 8 bits (a byte). The

use of `<label>` is optional. If a `label` is used its value is set to the starting `<address>` of the table. The `DB` (define bytes) directive is an alias for this directive.

If the `assembler` option "Allow strings in FCB directive" is checked the list of values may also contain character strings. Each character from the string is saved in a memory byte. The begin and end of each string used in a FCB directives should be marked with `"`. (Not with an `'`)

E.g.

```
LABEL FCB "Hello", $20, "World", 0
```

Will fill the memory starting on address LABEL with \$48 \$65 \$6c \$6c \$6f \$20 \$57 \$6f \$72 \$6c \$64 \$00 if the "Allow strings in FCB directive" option is checked. But will generate an error "The value of this expression must fit in 8 bits" when the "Allow strings in FCB directive" option is unchecked. Because by default the `"` token is used to specify double byte constants which will not fit in a single byte.

FDB directive

Syntax: [`<label>`] FDB `<value>`,`<value>`,...,`<value>`

The FDB (form double bytes) directive forms a table with constant double byte values in memory. You can provide one or more values separated by commas (without a space). Each `<value>` should fit into 16 bits (a word or double byte). The use of `<label>` is optional. If a `label` is used its value is set to the starting `<address>` of the table. The `DW` (define words) directive is an alias for this directive.

FCC directive

Syntax: [`<label>`] FCC '`<string>`' or: [`<label>`] FCC "`<string>`"

The FCC (form constant characters) directive forms a table with constant character ASCII values in memory. You can provide one or more characters as a `<string>`. A string is a row of characters. The begin and end of the string are marked with `'` or `"`. These begin and end mark are not included in the table. Each 7 bit ASCII character is put in b6 to b0 of a byte and b7 is always zero. The use of `<label>` is optional. If a `label` is used its value is set to the starting `<address>` of the table.

RMB directive

Syntax: [`<label>`] RMB `<value>`

The RMB (reserve memory bytes) directive forms an empty variable or an empty table in memory. You can provide one `<value>` to set the size of the variable or table measured in bytes. The use of `<label>` is optional. If a `label` is used its value is set to the starting `<address>` of the variable or table. The `DS` (define storage) directive is an alias for this directive.

#INCLUDE directive

Syntax: #INCLUDE "`<filename>`" or: #INCLUDE '`<filename>`'

The #INCLUDE directive includes the contents of the given filename in the current file. You must provide the `filename` of the file to be included between single or double quotes. The \$INCLUDE directive is an alias for this directive.

It is common practice to include a file that contains `EQUates` for all the internally addressable registers and bits for the 68HC11.

Assembler Operand Syntax

The mnemonic can be followed by the operand separated by at least one whitespace character. The operand specifies the addressing mode. Not every mnemonic can be combined with every addressing mode.

Available addressing modes.

Immediate	#expression	LDAA #3 LDD #\$ABCD
Direct	expression	LDX \$34 LDAB <PIET
Extended	expression	LDX \$1034 LDAB >PIET
Indexed	expression,X expression,Y	LDAA 0,X LDAB 3,Y
Relative	address	BRA LABEL

For indexed addressing, the comma is required before the register; INC X and INC ,X are not the same.

The force direct addressing operator '<' forces direct addressing mode when the label equals a value greater than \$00FF an error is generated. The force extended addressing operator '>' forces extended addressing mode.

For the BCLR and BSET instructions the addressing mode is followed by a mask. For the BRCLR and BRSET instructions the addressing mode is followed by a mask and a label. Bit manipulation operands can be separated by blanks or by commas. Before the mask you may use a # symbol.

Expressions may consist of symbols, constants or the character '*' (denoting the current value of the program counter) joined together by one of the operators: +-*/^!~&|.

Available operators.

+	add
-	subtract
*	multiply
/	divide
\	remainder after division
~	bitwise not
&	bitwise and
	bitwise or
!	bitwise or
^	bitwise exor

The operators have the following priority:

* / \	highest priority
+ -	
&	
^	
!	lowest priority

Operators with equal priority are evaluated left to right and parenthesised expressions can be used to change the evaluation order. '(' and ')' or '{' and '}' can be used to parenthesise an expression. Arithmetic is carried out in signed twos-complement integer precision (32 bits).

Constants are constructed with the following syntax:

'	followed by ASCII character
"	followed by two ASCII characters
\$	followed by hexadecimal constant
&	followed by decimal constant
@	followed by octal constant
%	followed by binary constant
digit	decimal constant or hexadecimal constant depending on the <u>option</u> you selected

Assembler Comment Syntax

Any text following the operand for a given mnemonic or, a line beginning with * or, an empty line are all ignored by the assembler and can be used as comments.

Assembler Errors

Error diagnostics are placed in the listing file just below the line containing the error. Format of the error line is:

```
TEST STAA #2
E          ^ Immediate addressing mode is not allowed here.
```

Error messages are meant to be self-explanatory. Some errors are classified as fatal and cause an immediate termination of the assembly. Generally these errors occur when a temporary file cannot be created or is lost during the assembly. Consult your local guru if this happens.

THRAss11 Implementation Notes

THRAss11 is a classic 2-pass assembler. Pass 1 establish the symbol table and pass 2 generates the code. Equates that have forward references can thus cause so called "phasing errors" in pass 2.

THRAss11 has the following differences with AS11 (the freeware assembler from Motorola):

- The THRAss11 assembler does NOT support the following directives or pseudo-ops: BSZ, ZMB, FILL, PAGE, and OPT.
- The following pseudo-ops are recognised but ignored on AS11, the THRAss11 assembler generates an error when you use these directives: SPC, TTL, NAM[E]
- Bit manipulation operands are separated by blanks in AS11. In the THRAss11 assembler you can use both blanks or commas.
- Labels can contain the character \$ In AS11. THRAss11 generates an error when you use a \$ within a label.
- Labels are case sensitive in AS11. THRAss11 translates all labels to uppercase. So LABEL and label are two different labels for AS11 but they are the same for THRAss11.

Command window help

This part of the help file gives an overview of the command window commands. The appropriate parameters are mentioned with each command.

You can use 3 forms of input:

<name>	Displays the value of <name> .
<name> <value>	Assign the value of <value> to <name> .
<command>	Execute <command> .

General commands:

Run
Go
Stop
Step
Run until
Step until
Assembler
Disassemble

Commands for timing:

Clock cycles
Print simulated time
Print E clock period
Set E clock period

Commands for loading files:

Edit
Load and assemble ASM source
Load ELF including debug info
Load S19 machine code
Load ELF machine code
Load commands
Load map
Load list

Commands concerning labels:

Set label
Delete label
Set standard labels
Delete standard labels
Delete all labels
Print label
Print all labels

Commands concerning breakpoints:

Set breakpoint
Delete breakpoint
Delete all breakpoints
Print breakpoint
Print all breakpoints

Commands concerning memory:

Print name
Print all names
Print memory

[Print stack](#)
[Print registers](#)
[Print pins](#)
[Set memory](#)
[Fill memory](#)
[Clear memory](#)
[Map memory](#)

Commands concerning numerical systems:

[Hexadecimal](#)
[Signed decimal](#)
[Unsigned decimal](#)
[Octal](#)
[Binary](#)
[ASCII](#)
[Hexadecimal all](#)
[Signed decimal all](#)
[Unsigned decimal all](#)
[Octal all](#)
[Binary all](#)
[ASCII all](#)

Commands for general information:

[About](#)
[Info](#)
[Help](#)

External commands:

[!command](#)

Verification commands:

[Verify](#)
[Verify contains](#)
[Verify not](#)
[Verify contains not](#)

Other commands:

[Record](#)
[Sleep](#)
[Target](#)
[Send windows message](#)

Name

As [<name>](#) you can use:

A	register A.
B	register B.
C, CC or CCR	condition code register.
D	register D.
P	program counter.
S or SP	stack pointer.
X or IX	index register X.

Y or IY	index register Y.
CLOCK	number of clock cycles.
M <address> memory location <address>. PA0 to PA7	port A pins.
PB0 to PB7	port B pins.
PC0 to PC7	port C pins.
PD0 to PD5	port D pins.
PE0 to PE7	port E pins (analog voltage in mV).
STRA STRB	STRA STRB pins.
RESET	RESET pin.
IRQ XIRQ	IRQ XIRQ interrupt.
Vrl Vrh	Reference voltage low end high(AD converter).

Or one of the following "standard" labels:

PORTA, PIOC, PORTC, PORTB, PORTCL, DDRC, PORTD, DDRD, PORTE, CFORC, OC1M, OC1D, TCNTh, TCNTl, TICxh, TICxl, TOCxh, TOCxl, TCTLx, TMSKx, TFLGx, PACTL, PACNT, SPCR, SPSR, SPDR, BAUD, SCCR1, SCCR2, SCSR, SCDR, ADCTL, ADRx, OPTION, COPRST, PPROG, HPRI0, INIT, TEST1 or CONFIG.

You can also define your own labels (see SetLabel command).

A label must start with a letter, underscore (_), or dot (.) and can further contain letters, digits, underscores (_), and dots (.).

Every C/C++ variable which is **in scope** can be used as :<name>.

simple variables (also enums)	:count
array variable	:codes
array elements	:codes[2]
struct variable	:date
struct members	:date.month
pointers (also to functions)	:p
pointees	:*p
member of pointer to struct	:p->month
combinations of all of the above	:b[12]->e.a[2][2]

If there is no other <name> with the same name the : may be omitted.

Every <name> (except the pin names) may be preceded by ` to denote a <name> on the target board (including C/C++ variables which are in scope on the target board). For example:

`A	register A on the target board.
`B	register B on the target board.
`:count	C/C++ variable count on the target board.

Operator

As <operator> you can use:

= or ==	is equal to.
<> or !=	is not equal to.
>	is higher than (unsigned).

>=	is higher than (unsigned) or equal to.
<	is lower than (unsigned).
<=	is lower than (unsigned) or equal to.
&	bit set (as in the BRSET instruction).
!&	bit clear (as in the BRCLR instruction).

Address

16 bits value that identifies a memory location within the 68HC11 memory map (\$0000 - \$FFFF)

Label

A label is a symbolic name you can associate to a value or address to make it easier to remember or identify. You can define your own labels (see SetLabel command). A label must start with a letter, underscore (_), or dot (.) and can further contain letters, digits, underscores (_), and dots (.).

Filename

THRSim11 support filenames that can be up to 255 characters long. This is known as a long filename. Do not use the following characters in filenames, because they are reserved for Windows: < > : " / \ |

Run/Go

Syntax: Run [<address>] **or:** Go [<address>]

This command executes instructions starting from <address>. The execution of instructions can be stopped by the button or by using the Stop command.

Default <address> is PC.

Stop

Syntax: STOP

This command stops the execution of instructions.

Step

Syntax: sTep [<number>]

This command executes <number> instructions and prints each instruction. The execution of instructions can be stopped by the button or by using the Stop command.

Default <number> is 1.

Run until/Go until

Syntax: RununTil <address> **or:** GounTil <address>

This command executes instructions until PC=<address>. The execution of instructions can be stopped by the button or by using the Stop command.

Instead of RT or GT you may also use STOPAT or ST.

Step until

Syntax: sTepunTil <address>

This command executes instructions until PC=<address> and prints each instruction when executing it. The execution of instructions can be stopped by the button or by using the Stop command.

Assemble

Syntax: Assemble [<address>]

With this command you can edit assembler instructions one by one starting from address <address>. These assembler instructions are assembled immediately.

Default <address> = PC.

Disassemble / Print / List

Syntax: DisAssemble [<address1>] [<address2>] [<filename>] or: Print [<address1>] [<address2>] [<filename>] or: List [<address1>] [<address2>] [<filename>]

This command disassembles and prints the contents of memory from address <address1> to address <address2> to the file <filename> or to the screen if no filename is given.

Default < address1 > is PC.

Default < address2 > is <address1> + 16 instructions.

Default < filename > is the screen.

If you want to use THRSim11 to disassemble a **S19** file into an **ASM** file there is a command line tool available which you can download for **free** at: <http://www.hc11.demon.nl/thrsim11/faq.htm>

For example:

1. Download the file findlabels.zip
 2. Load a s19 file into THRSim11. For example capacity.s19 (delivered with the installation). You can do this in several ways e.g. Type: ls19 into the command window and choose a s19 file.
 3. Type into the command window: dis ff00 ffbc test.dis
 4. Unzip the findlabels.zip into the directory where test.dis is located. Open a DOS Window, go to the directory which contains test.dis and type: findlabels test.dis test.asm
 5. You can now load test.asm into your favorite assembler. You probably have to add some directives (ORG) but this should not cause any problems. Just to test you can load test.asm into THRSim11 and assemble it again.
-

Clock cycles

Syntax: CLOCKcycles [<value>]

This command prints the number of simulated clock cycles when no <value> is provided. Otherwise the number of simulated clock cycles is set to <value>.

Print simulated time

Syntax: PrintSimulatedTime

This command prints the simulated time in seconds. This is calculated using the number of simulated clock cycles and the E clock period time. The E clock period time can be set by using the SetEClockPeriod command.

Print E clock period

Syntax: PrintEClockPeriod

This command prints the E clock period in ns. The E clock period time can be set by using the SetEClockPeriod command.

Set E clock period

Syntax: SetEClockPeriod <value>

This command sets the E clock period time in ns to <value>.

Edit

Syntax: EDit [-<value>] [<filename>]

This command the file with name <filename> into an external editor or into an edit window. You can change the editor options by editing the THRSim11_options.txt file.

When you provide a <value> the file will be opened on that line. The default value is 1. If you do not provide a <filename> a dialog box will appear where you can select a file.

Load and assemble ASM source

Syntax: LoadASM [<filename>]

This command reads a source file named <filename> and assembles it. The generated S19 and list file are automatically loaded. If you do not provide a <filename> a dialog box will appear where you can select a file.

Load ELF including debug information

Syntax: LoadELF [<filename>]

This command loads an ELF executable file <filename> into memory. A C/C++ list window and C/C++ variables window is opened to show the program source and variables. The ELF file must contain 68HC11 code with DWARF debug information generated by the GNU Development Chain for 68HC11. If you do not provide a <filename> a dialog box will appear where you can select a file.

Load S19 machine code

Syntax: LoadS19 [<filename>]

This command loads the contents of the file <filename> in memory. The file must contain S19 records. If you do not provide a <filename> a dialog box will appear where you can select a file.

Load ELF machine code

Syntax: LoadMemoryElf [<filename>]

This command loads the contents of the ELF executable file <filename> in memory. The file must contain ELF records. If you do not provide a <filename> a dialog box will appear where you can select a file.

Load commands

Syntax: LoadCoMmanDs [<filename>]

This command reads a script file <filename> and executes the THRSIM11 command line commands that it contains. If you do not provide a <filename> a dialog box will appear where you can select a file.

Load map

Syntax: LoadMAP [<filename>]

This command reads the map file <filename> and adds the labels found in this map file to the label table. If you make a <name>.map file starting with:

```
THRSIM11 MAPFILE
```

and on every next line a label followed by an expression. All labels are loaded automatically when you open the file <name>.s19. If you do not provide a <filename> a dialog box will appear where you can select a file.

Load list

Syntax: LoadLIST [<filename>]

This command reads the list file <filename> and adds the labels found in this list file to the label table. If you do not provide a <filename> a dialog box will appear where you can select a file.

Set label

Syntax: SetLabel <label> [<value>]

This command adds the label <label> with value <value> to the label table or prints the value of <label>. The label that is defined in this way can be used in expressions.

Delete label

Syntax: DeleteLabel [<label>]

This command removes the label <label> from the label table.

Default <label> is * (all labels)

Set standard labels

Syntax: SetStandardLabels

This command loads the standard labels for I/O registers (as used in the Motorola documentation) in the label table.

The "standard" labels are:

PORTA, PIOC, PORTC, PORTB, PORTCL, DDRC, PORTD, DDRD, PORTE, CFORC, OC1M, OC1D, TCNTTh, TCNTI, TICxh, TICxl, TOCxh, TOCxl, TCTLx, TMSKx, TFLGx, PACTL, PACNT, SPCR, SPSR, SPDR, BAUD, SCCR1, SCCR2, SCSR, SCDR, ADCTL, ADRx, OPTION, COPRST, PPROG, HPRI0, INIT, TEST1 or CONFIG.

Delete standard labels

Syntax: DeleteStandardLabels

This command removes the standard labels for I/O registers (as used in the Motorola documentation) from the label table.

Delete all labels

Syntax: DeleteAllLabels

This command removes all labels from the label table.

Print label

Syntax: PrintLabel [<label>]

This command prints label <label> with its associated value.

Default <label> is * (all labels)

Print all labels

Syntax: PrintAllLabels

This command prints all labels with their associated values.

Set breakpoint

Syntax: SetBreakpoint <name> [<operator>] <value> [<count>]

This command sets a breakpoint on the condition <name> <operator> <value>.

This means that the simulator stops executing instructions if the condition <name> <operator> <value>

becomes true. The <count> is optional. If you enter a count value. A counter will be incremented each time the breakpoint condition **becomes** true. If this counter reaches the entered count value the simulation of instructions will be stopped.

Default <operator> is =.

Default <count> is 1.

Examples:

SB A \$03	stops executing if register A becomes equal to \$03.
SB A > \$03	stops executing if register A becomes higher than \$03.
SB A \$03 5	stops executing if register A becomes equal to \$03 for the fifth time.
SB A & \$03	stops executing if register A AND \$03 becomes not equal to \$00. Or formulated differently: stops if bit1 and/or bit0 of register A is being set.
SB M \$0EBD \$a	stops executing when \$0A is written to memory location \$0EBD.

Instead of SB you may also use BR or BS.

Delete breakpoint

Syntax: DeleteBreakpoint [<name>] [<operator>] [<value>]

This command removes the breakpoint with condition <name> <operator> <value>.

Default <operator> is =.

Default <name> is * (all breakpoints).

Default <value> is all breakpoints on <name>

Examples:

DB M \$0EBD \$a	removes the breakpoint with value \$a on memory location \$0EBD.
DB A < \$2	removes the breakpoint with the condition A < \$02.
DB Y	removes all breakpoints on register Y.
DB	removes all breakpoints.

Delete all breakpoints

Syntax: DeleteAllBreakpoints
 This command removes all breakpoints.

Print breakpoint

Syntax: PrintBreakpoint [<name>]
 This command prints a list of the breakpoints set on <name>.

Print all breakpoints

Syntax: PrintAllBreakpoints
 This command prints a list of all breakpoints.

B<n> / SB<n>

Syntax: B<number> <value> or: SB<number> <value>
 This command is provided to be compatible with an other simulator that was formerly used at the TH Rijswijk and this command is probably of no interest to you. You better use the Set breakpoint command.

This command sets breakpoint<number> on the condition PC = <value>. This means that the simulator stops executing instructions if the condition PC = <value> becomes true.
 As <number> you can use: 1, 2, 3 or 4.

DB<n>

Syntax: DB<number>
 This command is provided to be compatible with an other simulator that was formerly used at the TH Rijswijk and this command is probably of no interest to you. You better use the Delete breakpoint command.

This command removes breakpoint<number> on PC.
 DB <number> you can use: 1, 2, 3 or 4.

PB<n>

Syntax: PB<number>
 This command is provided to be compatible with an other simulator that was formerly used at the TH Rijswijk and this command is probably of no interest to you. You better use the Print breakpoint command.

This command prints the value of breakpoint<number>.

As <number> you can use: 1, 2, 3 or 4.

Print name

Syntax: PrintName [<name>]

This command prints name <name> with its associated value.

Default <name> is * (all names)

Print all names

Syntax: PrintAllNames

This command prints all names with their associated values.

Print memory

Syntax: PrintMemory [<address1>] [<address2>] [<filename>]

This command prints the contents of memory locations. The area from address <address1> to address <address2> is printed in hexadecimal notation to the file <filename> or to the screen if no filename is given.

Default < address1 > is start of RAM.

Default < address2 > is <address1> + \$80.

Default < filename > is the screen.

Instead of PM you may also use MD or DUMP.

Print stack

Syntax: PrintStack

This command prints the contents of 80 memory locations surrounding the memory location the SP is pointing to in hexadecimal notation.

Print registers

Syntax: PrintRegisters

This command prints the contents of all registers on one line in hexadecimal notation.

Instead of PR you may also use REG, RD, or RP.

Print pins

Syntax: PrintPins

This command prints the value of all pins.

Instead of PP you may also use PIN.

Set memory

Syntax: SetMemory [<address >] or: SetMemory <address > <value> [<value>] [<value>] [<value>] ...

If you do not provide one or more <value>s a dialog box is opened. With this command the contents of memory locations starting from address <address> can be changed. The default address is the first address of RAM.

Subcommands (only available in dialog box):

<return>	Opens the next memory location. (Same as OK button.)
=	Opens the same memory location.
^	Opens the previous memory location.
.	End of SetMemory command. (Same as Cancel button.)

Instead of SM you may also use MM or MS.

Fill memory

Syntax: FillMemory [<address1>] [<address2>] [<value>]

This command fills the memory from <address1> upto and including <address2> with <value>. If you do not provide all parameters an appropriate dialog box pops up.

Clear memory

Syntax: ClearMemory [<value>]

This command fills all RAM and ROM locations with <value>. The default value is \$FF.

Map memory

Syntax: MAPmemory

This command prints the memory map.

Hexadecimal

Syntax: HEXadecimal [<name>]

This command causes the simulator to print the value of <name> in the hexadecimal notation.

Signed decimal

Syntax: signedDECimal [<name>]

This command causes the simulator to print the value of <name> in the signed decimal notation (two's complement).

Unsigned decimal

Syntax: UnsignedDECimal [<name>]

This command causes the simulator to print the value of <name> in the unsigned decimal notation.

Octal

Syntax: OCTal [<name>]

This command causes the simulator to print the value of <name> in the octal notation.

Binary

Syntax: BINary [<name>]

This command causes the simulator to print the value of <name> in the binary notation.

ASCII

Syntax: ASCii [<name>]

This command causes the simulator to print the value of <name> in ASCII notation. This is only possible for 8 bits and 16 bits numbers. E.g. \$31 is shown as '1' and \$4161 is shown as "Aa. Nonprintable characters are shown as ¨.

Hexadecimal all

Syntax: HEXadecimalAll

This command causes the simulator to print all numbers in hexadecimal notation.

Signed decimal all

Syntax: signedDECimalAll

This command causes the simulator to print all numbers in signed decimal notation (two's complement).

Unsigned decimal all

Syntax: UnsignedDECimalAll

This command causes the simulator to print all numbers in unsigned decimal notation.

Octal all

Syntax: OCTalAll

This command causes the simulator to print all numbers in octal notation.

Binary all

Syntax: BINaryAll

This command causes the simulator to print all numbers in binary notation.

ASCII all

Syntax: ASCiIall

This command causes the simulator to print all 8 and 16 bits numbers in ASCII notation.

About

Syntax: ABOUT

This command gives general information about this simulator.

Info

This program is designed and written by:

Rob van Beek	(former) student TH Rijswijk.
Arjan Besjis	(former) student TH Rijswijk.
Wilbert Bilderbeek	(former) student TH Rijswijk.
Robert Brilmayer	(former) student TH Rijswijk.
Harry Broeders	(former) teacher and (former) student TH Rijswijk.
Bart Grootendorst	(former) student TH Rijswijk.
Philip Heppe	(former) student TH Rijswijk.
Alex van Rooijen	(former) student TH Rijswijk.

This is version 5.xx. This version simulates the 68HC11A8 microcontroller from Motorola.

- Instructionset
- I/O ports A to E
- Interrupt system
- Timer system
- Handshake system
- AD converter
- Serial interface

1994-2004 Harry Broeders.

For further information contact:

Harry Broeders
harry@hc11.demon.nl

Help

Syntax: Help

This command displays an overview of all commands available in the command window of THRSim11.

External command

Syntax: !<command>

THRSim11 can start an external command and show the output of this command in the command window. The external commands which you can start from THRSim11 are the commands from the GNU Development Chain (**m6811-elf-gcc**, **m6811-elf-as**, **m6811-elf-ld**, etc.) for 68HC11 and the GNU Utilities for Windows (**make**, **rm**, etc). These commands are located in <THRSim11 directory>\gcc\bin and <THRSim11 directory>\utils. <THRSim11 directory> is the directory you choosed when you installed THRSim11 (C:\Program Files\THRSim11 on default). You can point THRSim11 to other directories by changing the GNU Tools options in the THRSim11_options.txt file.

Example:

```
!make
```

You can for example generate a Intel hex file using the command:

```
!m6811-elf-objcopy -O ihex a.out a.hex
```

The m6811-elf- prefix may be omitted:

```
!objcopy -O ihex a.out a.hex
```

Most external commands provide their own help info with the **--help** option.

For example:

```
!make --help
```

[More information about the GNU Development Chain for 68HC11 can be found here \(in HTML\).](#)

Help for most of the GNU Utilities for Windows can be found on the internet: <http://www.gnu.org/software/coreutils/manual/>

Help for the GNU make utility can also be found on the Internet: http://www.gnu.org/software/make/manual/html_chapter/make.html.

There is also one build-in command: !cd that you can use to change the current working directory.

Verify

Syntax: VErify [-<value>] <string>

This command is ment to be used from CMD files which can be loaded and executed with the LCMD command. The verify command compares the last output line with the <string>. If the comparison finds a difference the execution of the CMD file is stopped and a message is printed. If the <string> **exactly** matches the output line nothing will happen. You can provide a <value> to compare <string> with other output lines. For example: If you use the value 1 one line before the last line is compared with <string>. The default value is 0.

The following CMD file, for example, tests one of the example programs:

```
# load the program to test
lasm c:\program files\thrsim11\examples\example1.asm
# provide input value ($bd = 189)
sm input $bd
# set breakpoint at label loop (end of program)
sb pc loop
run
# check output
m output
ver OUTPUT, M $0001 = $31.
m output+1
ver M $0002 = $38.
m output+2
ver M $0003 = $39.
```

The next CMD file does the same but places all verify commands at the end of the CMD file:

```
lasm c:\program files\thrsim11\examples\example1.asm
sm input $bd
sb pc loop
```

```
run
m output
m output+1
m output+2
ver -2 OUTPUT, M $0001 = $31.
ver -1 M $0002 = $38.
ver -0 M $0003 = $39.
```

The <string> given in the verify command must **exactly** match the output line otherwise the verify will not succeed. It is often more convenient to use the VerifyContains command.

Verify contains

Syntax: VerifyContains [-<value>] <string>

This command is ment to be used from CMD files which can be loaded and executed with the LCMD command. The verifycontains command compares the last output line with the <string>. If the output line does not contain the given <string> the execution of the CMD file is stopped and a message is printed. If the output line **contains** the <string> nothing will happen. You can provide a <value> to compare <string> with other output lines. For example: If you use the value 1 one line before the last line is compared with <string>. The default value is 0.

The following CMD file, for example, tests one of the example programs:

```
# load the program to test
lasm c:\program files\thrsim11\examples\example1.asm
# provide input value ($bd = 189)
sm input $bd
# set breakpoint at label loop (end of program)
sb pc loop
run
# check output
m output
vc $31.
m output+1
vc $38.
m output+2
vc $39.
```

The next CMD file does the same but places all verify commands at the end of the CMD file:

```
lasm c:\program files\thrsim11\examples\example1.asm
sm input $bd
sb pc loop
run
m output
m output+1
m output+2
ver -2 $31.
ver -1 $38.
ver -0 $39.
```

Verify not

Syntax: VerifyNot [-<value>] <string>

This command is ment to be used from CMD files which can be loaded and executed with the LCMD command. The verifynot command compares the last output line with the <string>. If the <string> **exactly** matches the output line the execution of the CMD file is stopped. When the comparison finds a difference nothing will happen. You can provide a <value> to compare <string> with other output lines. For example: If you use the value 1 one line before the last line is compared with <string>. The default value is 0.

Verify contains not

Syntax: VerifyContainsNot [-<value>] <string>

This command is ment to be used from CMD files which can be loaded and executed with the LCMD command. The verifycontainsnot command compares the last output line with the <string>. If the output line does contain the given <string> the execution of the CMD file is stopped and a message is printed. If the output line does **not contains** the <string> nothing will happen. You can provide a <value> to compare <string> with other output lines. For example: If you use the value 1 one line before the last line is compared with <string>. The default value is 0.

The following CMD file, for example, tests one of the example programs and checks if the program is correctly loaded before further action is taken:

```
# load the program to test
lasm c:\program files\thrsim11\examples\example1.asm
# check if program is correctly loaded
vcn Error:
# provide input value ($bd = 189)
sm input $bd
# set breakpoint at label loop (end of program)
sb pc loop
run
# check output
m output
vc $31.
m output+1
vc $38.
m output+2
vc $39.
```

Record

Syntax: RECoRD [<filename>]

This command saves all commands that are typed into the in the command window into the file <filename>. When no <filename> is provided and there is no recording going on a dialog box appears which you can use to type or select a <filename>. When the record command is used without <filename> when a recording is going on the recording stops.

All input lines are preceded with < and all output files are preceded by > in the recording.

Here is an example of a recording:

```
> Recording of commands started.
< a
> A = $bd.
< rec output2.txt
> Error: Already recording commands.
< rec
```

Sleep

Syntax: SLEEP [<value>]

This command is ment to be used from CMD files which can be loaded and executed with the LCMD command. This command pauses the execution of commands from the CMD file for <value> milliseconds. The default <value> is 1000.

Target command

Syntax: TArget <command>

This command is ment to be used from CMD files which can be loaded and executed with the LCMD command. This

command sends the <command> to the target board and displays the result in the [command window](#). If you just want to send a command to the target board you better use the [target command window](#). But this command is needed if you want to control the target board from a CMD file. This command is also useful when you want to make a recording with the [record command](#).

Send windows message

Syntax: sendWindowsMessage <message> [<parameters>]

This command is meant to be used from CMD files which can be loaded and executed with the [LCMD command](#). This command sends a windows message to THRSim11. You can use this to simulate keypresses and mouse movements when executing CMD files.

You may use raw windows messages but this command also understands symbolic constants as defined by Microsoft. All WM_COMMAND parameters are also defined as symbolic constants <menu>::<menu-option>.

The following CMD file, for example, opens a new edit window and types a simple program:

```
wm WM_COMMAND File::New
wm WM_CHAR 'N
wm WM_CHAR 'i
wm WM_CHAR 'c
wm WM_CHAR 'e
wm WM_CHAR VK_TAB
wm WM_CHAR 'N
wm WM_CHAR 'i
wm WM_CHAR 'c
wm WM_CHAR 'e
wm WM_CHAR VK_NEWLINE
sleep 500
wm WM_KEYDOWN VK_UP
sleep 800
wm WM_KEYDOWN VK_RIGHT
sleep 600
wm WM_KEYDOWN VK_RIGHT
sleep 400
wm WM_KEYDOWN VK_RIGHT
sleep 200
wm WM_KEYDOWN VK_RIGHT
sleep 100
wm WM_CHAR VK_TAB
wm WM_CHAR 'n
wm WM_CHAR 'o
wm WM_CHAR 'p
wm WM_CHAR VK_NEWLINE
wm WM_CHAR VK_TAB
wm WM_CHAR 'b
wm WM_CHAR 'r
wm WM_CHAR 'a
wm WM_KEYDOWN VK_BELL
sleep
wm WM_COMMAND File::Assemble
sleep
wm WM_KEYDOWN VK_F7
sleep
wm WM_KEYDOWN VK_F7
sleep
wm WM_CLOSE
wm WM_KEYDOWN VK_BELL
sleep
wm WM_CLOSE
```

External boxes.

Within the simulator several external boxes are present in the View menu. A box simulates an external circuit or process which is connected to the THRSim11 simulator. Within the simulator several boxes are available, a box is automatically connected to the simulator by selecting the menu box from the View menu.

The following boxes are available:

I/O box

Sliders E port

Resistance measurement

Capacity and frequency measurement

Serial receiver

Serial transmitter

Additional components can be installed in the THRSim11 simulator. These components will be available in the Connect menu. For example LED, switch, 7 segment display, square wave generator, logic analyzer, etc..

For information on available components see <http://www.hc11.demon.nl/thrsim11/comp.htm> or contact harry@hc11.demon.nl (Harry Broeders).

The I/O box.

This box contains:

Switches connected to the 68HC11 pins on PC0-PC7.

LED's connected to the 68HC11 pins on PB0-PB7.

LED's connected to the 68HC11 pins on PD0-PD5.

STRA switch + LED connected to the 68HC11 pin STROBE A.

4 x 20 rows LCD display. (When controlling this display use the 'simlcd.inc' include file in your program.)

A switch can be toggled by clicking on it with the mouse or by typing 0 to 7 for PC0 to PC7 or A for STRA.

The switches are normally used when port C is configured as input port (DDRCx = 0). When port C is configured as output port (DDRCx = 1) the switches represent the value written to port C. Real switches will not follow this behavior but will cause a short circuit :-).

The LCD display is controlled through 2 memory mapped registers. The LCD's data input is mapped at address \$1040. This can be changed in the thrsim11_options.txt initialization file. At the Rijswijk Institute of Technology the address \$3000 is normally used.) Writing a printable ASCII to this location causes the LCD display to print this character.

For example:

```
LDAA  #$41
```

```
STAA  $1040
```

```
STAA  $1040
```

will put AA on the LCD display.

The following non printable characters have a special meaning:

\$00 clear the display.

\$0D go to the first position of the next line.

\$05 get the cursor position. This position can be read from the control register mapped at address \$1041.

(This can be changed in the thrsim11_options.txt initialization file. At the Rijswijk Institute of Technology the address \$2000 is normally used.) The positions on the display are numbered from the upper left corner (0) to the bottom right corner (79).

For example:

```
LDAA  #$05
```

```
STAA  $1040
```

```
LDAB  $1041
```

will load the current cursor position in register B.

\$06 set the cursor position. This position must first be written to the control register mapped at address \$1041. (This can be changed in the [thrsim11_options.txt](#) initialization file. At the Rijswijk Institute of Technology the address \$2000 is normally used.) The positions on the display are numbered from the upper left corner (0) to the bottom right corner (79).

For example:

```
LDAA #30
STAA $1041
LDAA #$06
STAA $1040
```

will set the cursor position to the middle of the second line.

All other non printable characters are printed as a dot except \$04 and \$FF which are completely ignored.

Sliders E port.

The slider panel contains 8 potentiometers for controlling the analogue voltage on the E pins of the 68HC11 controller. The voltage can be controlled by positioning the sliders with the mouse pointer. If you want to type in a value just click the displayed value. When you type in a value (or the value of the pin changes for another reason) the slider moves to a new position according to the new value.

Resistance measurement.

This box contains a resistance measurement circuit. By controlling the pins PD3-PD5 a voltage divider will be created. The voltages on the several pins are displayed in the 7-segments displays. You can toggle the PD3-PD5 pins by clicking the displayed value.

Capacity and frequency measurement.

This box contains an electronic circuit for capacity and frequency measurement. When the simulator is running, the circuit delivers a continuing square wave (using a NE555 timer). The frequency of this square wave can be altered by varying the value of the capacitor Cx. To alter the value of the 'unknown' capacity Cx click on it.

The box is connected to the following ports:

PA7 Frequency measurement can be done by the 68HC11 controller using the pulse accumulator system.
PA2 Period measurement and C determination can be performed by using the input capture system of the 68HC11 microcontroller.

Here are the relevant parts from the NE555 datasheets:

The serial receiver.

This box contains the serial receiver. The box is connected to the PD1 = TxD pin of the simulated 68HC11. So the serial signals send by the 68HC11 microcontroller are displayed in this box. You have to write and run a

program that uses the 68HC11 serial controller to send a serial signal to the serial receiver.

Printable characters are normally displayed. Non printable characters are displayed as <ddd> where ddd is the decimal ASCII value of the received non printable character. Four non printable characters: CR, LF, BEL, and NUL are handled in a special way which can be controlled using the [thrsim11_options.txt](#) file. On default a CR character followed by a LF character causes the next output to start at the beginning of a new line. All other CR and LF characters are ignored on default. The BEL character is not displayed but generates a beep from the PC speaker. The NUL character is ignored on default.

There are three command buttons.

Settings Setting baud rate, parity, etc. You probably want to match the setting of the receiver with the settings you use in your program which is sending the data. But you can create a mismatch to simulate what will happen in this error situation. (You probably create a parity and/or framing error.) The current settings are displayed in the status bar at the bottom of the serial receive window. These settings are saved in the [thrsim11_options.txt](#) initialization file so if you exit the simulator and run it again later these settings are preserved.

Clear Clears all received characters currently displayed. This doesn't reset the receiver.

Reset Resets the receiver. Parity and/or framing errors will be removed.

The two gray squares at the top of the serial receiver window display the framing and parity errors. To clear these communication error flags press the reset button.

The serial transmitter.

This box contains the serial transmitter. The box is connected to the RxD pin of the simulated 68HC11. So the generated serial signal is send to the 68HC11 microcontroller. You have to write and run a program that uses the 68HC11 serial controller to receive a serial signal from the serial transmitter. You can type in the characters you want to send to the 68HC11 microcontroller.

Printable characters can be normally typed. Non printable characters must be typed as <ddd> where ddd is the decimal ASCII value of the non printable character to send. When you type an "enter" on the keyboard a new line in the serial transmitter window is started and a CR character is send (when you click the send button) on default. Which character is send at the start of each new line can be controlled using the [thrsim11_options.txt](#) file.

There are three buttons in this box.

Settings Setting baud rate, parity, etc. You probably want to match the setting of the transmitter with the settings you use in your program that is receiving the data. But you can create a mismatch to simulate what will happen in this error situation. (You probably create a parity and/or framing error.) The current settings are displayed in the status bar at the bottom of the serial transmit window. These settings are saved in the [thrsim11_options.txt](#) initialization file so if you exit the simulator and run it again later these settings are preserved.

Send The selected text in the edit window will be send to the 68HC11 microcontroller. If you didn't select a part of the text the entire contents of the window is automatically selected and send. (Note: the simulator must be running to perform this task.)

Stop This button ends the transmission of data to the microcontroller.

The serial settings.

This dialog appears when you press the Settings button in the serial receiver box or serial transmitter box. The setting of receiver and transmitter are not equal but both settings use a similar dialog box.

You can use this dialog to select the Baudrate, Mode (ten or eleven bits of data per character), and Parity (none, even, or odd). An even parity means that the number of ones send in the character is always even. Even and odd parity can only be used in Mode 11.

The current settings are displayed in the status bar at the bottom of the serial box. These settings are saved in the thrsim11_options.txt initialization file so if you exit the simulator and run it again later these settings are preserved.

External components.

Additional components can be installed in the THRSim11 simulator. Any installed external component will be shown in the Connect menu. (For example LED, switch, 7 segment display, square wave generator, logic analyzer, etc..)

Version 3.20 (and higher) of THRSim11 is delivered with a couple of components already installed:

- LED
- switch
- several 7 segment displays

For more information about these components see <http://www.hc11.demon.nl/thrsim11/comp320.htm>

For information on other available components see <http://www.hc11.demon.nl/thrsim11/comp.htm> or contact harry@hc11.demon.nl (Harry Broeders).

If you want to develop your own components you can download the THRSim11 Component Development Kit at: <http://www.hc11.demon.nl/thrsim11/cdk/documentation/index.htm>

Rescan for installed components.

On startup the THRSim11 program scans the window's registry for additional components. If you install additional components when THRSim11 is running you need to rescan the registry before you can use these components. Choose the Connect, Rescan for Installed Components menu option to rescan the registry.

Connect.

External components can be connected to any signal on the 68HC11 simulator using the connect dialog box.

When you open an external component by using the Connect menu the connect dialog box will automatically be opened.

Some components have default connections (these connections are automatically made when you open the component). You can use the Set default connections button to reset these connections. You can use the New node button to create external nodes. This can be useful when you want to connect several external components to each other.

You can open this dialog again to change the connections in several ways:

- Select an external component's window and choose the Connect, Connect... menu option.
- Open the window's system menu (click the icon) and choose the Connect... option.
- Open the window's context menu (click the right mouse button) and choose the Connect... option.

For information on available components see <http://www.hc11.demon.nl/thrsim11/comp.htm> or contact harry@hc11.demon.nl (Harry Broeders).

Overview of menu items

The THRSim11 menu consists of the following pull down menus:

<u>F</u> ile	Consists of menu items to control the assemble files and simulator program options.
<u>E</u> dit	Consists of menu items to edit text.
<u>S</u> earch	Consists of menu items to search for text.
<u>V</u> iew	Consists of menu items to open windows on registers, pins, memory, <u>l</u> abels, <u>b</u> reakpoints, external hardware, etc.
<u>T</u> arget	Consists of menu items to communicate with your target board.
<u>E</u> xecute	Consists of menu items to control the simulator.
<u>L</u> abel	Consists of menu items to view, set and remove <u>l</u> abels used in your program.
<u>B</u> reakpoint	Consists of menu items to view, set and remove <u>b</u> reakpoints used when simulating your program.
<u>C</u> onnect	Consist of menu items to open <u>e</u> xternal <u>c</u> omponents.
<u>W</u> indow	Consist of the standard menu items to manipulate the windows you opened.
<u>H</u> elp	Consist of the standard menu items to get help or information.

Menu File

This menu consists of menu items that controls the assemble files and simulator program.

New

New GNU gcc or as source This menu option is only available when you installed THRSim11 C support.

Open

Download to Target Board

Save

Save as&

Print&

Options

Assemble or Compile

Exit

The most recently used files are also displayed in this menu item.

Menu Edit

This menu consists of the following text edit menu items:

Undo

Cut

Copy

Paste

Delete

Menu Search

This menu contains the following search menu items:

Find

Replace

Search again

Menu View

This menu consists out of the following menu items:

Toolbar

Statusbar

Registers

CPU Registers

Ports

Timer

Serial

Pulse Accumulator

AD Converter

Handshake

Other registers

Pins

PA Pins

PB Pins

PC Pins

PD Pins

PE Pins

Other Pins

Memory

Memory List

Memory Dump

Stack

Memory Map

Memory Configuration

Number of Clock Cycles

High Level Language Variables

Custom Made Window

Command Window

[Label List](#)
[Breakpoint List](#)

[Disassembler](#)

[I/O Box](#)
[Sliders E Port](#)
[Resistance Measurement](#)
[Capacity Measurement](#)
[Serial Receiver](#)
[Serial Transmitter](#)

Menu Target

This menu consists out of the following menu items:

[Target Board Toolbar](#)

[Target Connect](#)
[Target Disconnect](#)
[Target Communication Options](#)

Target Registers

[Target CPU Registers](#)
[Target Ports](#)
[Target Timer](#)
[Target Serial](#)
[Target Pulse Accumulator](#)
[Target AD Converter](#)
[Target Handshake](#)
[Target Other registers](#)

Target Memory

[Target Memory List](#)
[Target Memory Dump](#)
[Target Stack](#)

[Target Download](#)

[Target High Level Language Variables](#)

[Target Custom Made Window](#)
[Target Command Window](#)

[Target Breakpoint List](#)

[Target Disassembler](#)

Target Copy

[Copy Target CPU Registers to Simulator](#)
[Copy Simulator CPU Registers to Target](#)
[Copy Target Memory to Simulator](#)
[Copy Simulator Memory to Target](#)
[Copy Simulator EEPROM Memory to Target](#)
[Copy Target Breakpoints to Simulator](#)
[Copy Simulator PC Breakpoints to Target](#)

Menu Execute

This menu controls the simulation of a program and consists out of the following menu items:

Run
Run until
Run from
Step

Stop
Reset 68HC11

Target Go
Target Go until
Target Go from
Target Step

Target Stop

Menu Label

The following label menu items are available:

Set
Remove
Remove All

Set standard label

Window

Menu Breakpoint

The following breakpoint menu items are available:

Set
Remove
Remove All

Window

Target Set
Target Remove
Target Remove All

Target List

Menu Connect

Info
Rescan for Installed Components

(Installed Components)

Connect

Menu Window

The following window management menu items are available:

Cascade

Tile

Tile Horizontal

Arrange Icons

Close All

Menu Help

The following help menu items are available:

Contents

More Info

About

Port A pins

This window contains the pins of port register A of the 68HC11 microcontroller.

Pins [PA0-PA2](#) Input pins.

Pins [PA3-PA6](#) Output pins.

Pin [PA7](#) Input/Output pin.

For an example on modifying pin values see [Working with register windows](#).

Port B pins

This window gives the pin values of port B of the 68HC11 microcontroller. These pins are output only pins.

For an example on modifying pins see [Working with register windows](#).

Port C Pins

This window gives an overview of the port C pins on the 68HC11 microcontroller. These pins can be configured as input or as output pins.

For an example on modifying pins see [Working with register windows](#).

Port D pins

This window gives an overview of the pins of port D on the 68HC11 microcontroller. These pins can be configured as input or output pins.

For an example on modifying pin values see [Working with register windows](#).

Port E pins

This window gives an overview of the pins of port E on the 68HC11 microcontroller. You can use these pins as analog or as digital inputs. When used as analog inputs the value of these pins is given in mV. To alter the value of the pins, double click on it and type a new value.

For an example on changing pin values see [Working with register windows](#).


Remaining pins

This window gives an overview of the remaining pins of the 68HC11 microcontroller. These are:

STRA pin	Alter by double-click or pressing the right mouse button.
STRB pin	Output only, not changeable.
IRQ pin	Alter by double-click or pressing the right mouse button.
XIRQ pin	Alter by double-click or pressing the right mouse button.
RESET pin	Alter by double-click or pressing the right mouse button.
Vrh pin	Reference voltage AD converter. Alter by double-click or pressing the right mouse button.
Vrl pin	Reference voltage AD converter. Alter by double-click or pressing the right mouse button.

For altering pin values see [Working with register windows](#).

Reset

Use the  button to reset the 68HC11 microcontroller. This action consists of placing a logical '0' on the RESET pin for a small period. After a reset the 68HC11 controller locates the start address from the reset vector at positions \$FFFE and \$FFFF.

A quick introduction to get started.

This is a quick introduction in using the THRSim11 68HC11 microcontroller simulator.

This introduction does **not** introduce the basics of the Motorola 68HC11 microcontroller itself, only the program that simulates it. You **should have elementary knowledge** about the 68HC11 register set and instruction set before you can use this simulator. It is perfectly possible to study the 68HC11 and learn to use this simulator in parallel. In fact this simulator was designed to give you a platform to experiment with the things you know or are learning about the 68HC11 microcontroller.

If you understand the following 68HC11 assembler program you are ready to start using the THRSim11 68HC11 microcontroller simulator program. If you don't understand the [assembler directives](#) you can click on them for more information.

* This is a very simple program that adds the contents of address
* \$0000 to the contents of address \$0001 and stores the resulting
* sum in the memory location at address \$0002. If there is an
* overflow the contents of address \$0003 is set to \$FF else
* address \$0003 is cleared.

```

    ORG $0000 start of RAM (data)
OPER1  RMB 1 reserve 1 memory byte for operand 1
OPER2  RMB 1 reserve 1 memory byte for operand 2
SUM     RMB 1 reserve 1 memory byte for sum
OVERFL  RMB 1 reserve 1 memory byte for overflow signal

FALSE  EQU $00 equate constant FALSE to $00

    ORG $FF00 start of ROM (program)
START  LDAB #FALSE be optimistic (assume no overflow occurs)
        LDAA OPER1 load operand 1 in accumulator A
        ADDA OPER1 add operand 2 to accumulator A
        BVC NOV branch if no overflow occurs
        COMB oops overflow: invert accumulator B
NOV     STAA SUM store result of addition
        STAB OVERFL store overflow signal
LOOP   BRA LOOP nice way to stop without running wild

    ORG $FFFE reset vector
    FDB START set to start of program
```

If you understand this program [click here](#).

If you don't understand this program you **can't** find any information about the 68HC11 register set, instruction set or input/output subsystems in this help file. You can find this information in another helpfile available for free at <http://www.hc11.demon.nl/thrsim11/68hc11>. You can also find links to several [books](#).

Books about the 68HC11 microcontroller.

If you don't know anything about the Motorola 68HC11 microcontroller you can use one of the following books (in random order):

M68HC11 Reference Manual, Motorola - literature reference M68HC11RM/AD

This FREE document is the "bible" of the 68HC11 and is a must-have for any serious 68HC11 programmer.

Microcontroller Technology, The 68HC11 4/e, Peter Spasov, Prentice Hall, 2002, 706 pp., ISBN 0-13-901240-0

Microcomputer Engineering 3/e, Gene H. Miller, Prentice Hall 2004, 592 pp., ISBN 0-13-142804-7

Microprocessors and Microcomputers: Hardware and Software 6/e, Ronald J. Tocci and Frank J. Ambrosio, Prentice Hall, 2003, 624 pp., ISBN 0-13-010494-9

Things move around a lot on the Internet. If the above links don't work look at: <http://www.hc11.demon.nl/thrsim11/68hc11> for an update.

Simulate or execute the Example0.asm program.

You can use THRSim11 to:

- **simulate** the program Example0.asm.
- **execute** the program Example0.asm on the Motorola EVM or EVB **target board** or on any other board running the BUFFALO monitor program.

You can go directly to one of these topics by clicking them or you can read them sequentially by using the >> button on the button bar.

Simulate the Example0.asm program.

To simulate the program **Example0.asm** you have to take the following actions:

- Load the source code in the editor.
- Assemble the source code to machine codes.
- Enter the appropriate input values.
- Run the program.
- Check the output values.
- Reset the 68HC11.

You can go directly to one of these topics by clicking them or you can read them sequentially by using the >> button on the button bar.

Load the source code in the editor.

You can load a file in the editor by using the button or the Open item from the File menu the Example0.asm assembler file can be loaded into an edit window. To select this file the following dialog box appears.

You can select the file Example0.asm by clicking on his name. After select the file the OK button must be pressed to load the file. If you double click the filename the file is immediately loaded.

The file Example0.asm will now be loaded into an edit window.

Assemble the source code to machine codes.

With the button or the Assemble item from the File menu the contents of the edit window can be converted to machine language. A list window will be opened and the machine code will be loaded in the simulated memory.

Depending on the assembler options you selected the listing and the machine codes are also saved on disk in a .LST and a .S19 file.

The green line shows you where the PC register is pointing to.

Enter the appropriate input values.

Before we can run the program we have to fill in the input values in memory. The program Example0.asm will add the contents of address \$0000 (labelled OPER1) to the contents of address \$0001 (labelled OPER2).

Lets for example try to add 5 and 6 together.

By using the Memory list item from the View menu you can open a memory list window to fill in these values. As you can see the labels used in the assembler program also appear in this window.

To enter the values 5 and 6 in memory locations \$0000 and \$0001 respectively you first have to select the line where you want to enter a value. If you press return a cursor will appear so you can enter a new value. It is not necessary to press return first you can immediate type in a new value after selecting a line. Press return after entering the value. If there is not enough room to type in an expression you can enlarge the window and try again.

Run the program.

With the Run button or the Run item from the Execute menu the 68HC11 simulator runs the program. The program ends with an endless loop:

```
LOOP BRA LOOP
```

This instruction doesn't stop the simulator but it causes the simulated 68HC11 to wait for a RESET or interrupt.

With the Stop button or the Stop item from the Execute menu the simulation can be stopped.

Check the output values.

The program writes the result to memory location \$0002 (labelled SUM). In the memory window you can see that the result of the addition of 5 and 6 is \$0B (hexadecimal notation for eleven).

If you find it easier to read out the result in decimal you can set the notation to decimal. To do this select the line where you want to change the notation and click the right mouse button or press Alt+F10. A pop up menu appears where you can select the decimal notation.

If you want to know how many clock cycles where used executing the program you can use the Number of clock cycles menu item from the View menu.

The number displayed is very high. This is caused by the continues running of the BRA instruction at the end of the program. If you just want to time the other instructions you must use a breakpoint to automatically stop the simulator when the BRA instruction is reached.

Reset the 68HC11.

If you want to add different numbers you can easily reset the simulated 68HC11 chip by clicking the

button. The content of the memory is **not** affected by a reset operation.

More things to learn.

Now you know how to assemble and simply simulate a simple 68HC11 assembler program it is time to learn the answers to the following questions:

- [What happens when the program contains errors?](#)
- [How can I enter my own program?](#)
- [How can I slowly step through the execution of the program?](#)
- [How can I execute the program until a certain condition?](#)
- [Can I add labels after the program is assembled?](#)
- [Can I minimise the number of windows I need?](#)

You can go directly to one of these questions by clicking them or you can read them sequentially by using the [>>](#) button on the button bar.

What happens when the program contains errors?

When you try to assemble a source file that contains errors these errors will be displayed in the [list window](#). The error messages are coloured red. Double clicking on an error message in this list window will select the source of the error in the [edit window](#). If you click the right mouse button on an error message in the list window or press Alt+F10 a pop up menu appears where you can select the first, last, next or previous error.

The programmer in the above example used the nonexisting instruction LDA instead of the correct instruction LDAA.

How can I enter my own program?

With the [button](#) or the [New](#) item from the [File menu](#) the simulator will create a new edit window. A 68HC11 assembler program can be written in this [edit window](#). Using the [button](#) or the [Save](#) item from the [File menu](#) the contents of the edit window can be stored.

Instead of starting out fresh you can also modify an existing program and save it under a different name using the [Save as](#) item from the [File menu](#).

How can I slowly step through the execution of the program?

There are many ways to do this. A few of the easy ways are:

- If the [list window](#) is active (selected) you can press F7, the space bar or the enter key to simulate one instruction. The green line that shows you where the PC register is pointing to will be placed on the next instruction to simulate. The window will automatically scroll so the green line will stay visible.
- If the [commands window](#) is active (selected) you can type "[STEP 4](#)" to simulate 4 instructions.

- If the list window is active (selected) you can simulate until a certain line in your program. To do this select the line where you want to stop the simulation and click the right mouse button or press Alt+F10. A pop up menu appears where you must select the Run until item.

To learn more ways see:

- The button.
- The button.
- Set a breakpoint on a line.

How can I execute the program until a certain condition is met?

To execute the program until a certain condition is met you can set a breakpoint.

You can use simple but also very complex conditions to stop the execution of a program. On this page 2 examples of breakpoints are given. One uses a simple condition and the other a complex one.

Breakpoint example 1.

If you want to determine the number of 68HC11 clock cycles it takes to execute the program Example0.asm. then you can use a breakpoint to stop the simulator at the end of the program.

First open the file Example0.asm and assemble the program. You can set a breakpoint at the BRA instruction by selecting the line in the list window and press F5.

If you want to know how many clock cycles where already used executing the program you can use the Number of clock cycles menu item from the View menu.

The number of clock cycles was set to 0 by the simulator when the program is loaded.

With the button or the Run item from the Execute menu the 68HC11 simulator runs the program until the breakpoint is reached.

The number of Clock cycles window shows the number of 68HC11 clock cycles used to run this program.

This number can also be calculated by using the timing information in the 68HC11 reference manual.

LDAB	takes 2 cycles (immediate addressing)
LDAA	takes 3 cycles (direct addressing)
ADDA	takes 3 cycles (direct addressing)
BVC	takes 3 cycles
COMB	is not executed because the above branch is taken!
STAA	takes 3 cycles (direct addressing)
STAB	takes 3 cycles (direct addressing)

Total 17 cycles

Breakpoint example 2.

You can also use complex conditions to stop the execution of the program. For example you can stop the execution of the program when a value less than 25 is written to memory location with label SUM. You can use the Set... item from the Breakpoint menu. After choosing this menu item a dialog box will appear where a new breakpoint and its conditions can be entered.

In the Name field the memory location with label SUM is selected. In the Operator field the operator < is selected. In the Value field the value 25 is entered. In the Count field the value 3 is entered. This means that the program stops when a value less than 25 is written to the memory location with label SUM for the third time.

When you run this program it stops at the breakpoint after you reset the 68HC11 twice.

Can I add labels after the program is assembled?

Yes, the most simple way is to use the label window.

Can I minimise the number of windows I need?

You can combine CPU registers, I/O registers, pins and memory locations in one custom made window.

Execute the Example0.asm program on the target board.

To execute the program **Example0.asm** on your EVM or EVB target board you have to take the following actions:

- Download the machine code into the target board.
- Enter the appropriate input values.
- Run the program.
- Stop the program.
- Check the output values.

You can go directly to one of these topics by clicking them or you can read them sequentially by using the >> button on the button bar.

Download the machine code into the target board.

You start with loading a file in the editor by using the button or the Open item from the File menu the Example0.asm assembler file can be loaded into an edit window. To select this file the following dialog box appears.

You can select the file Example0.asm by clicking on its name. After select the file the OK button must be pressed to load the file. If you double click the filename the file is immediately loaded.

The file Example0.asm will now be loaded into an edit window.

If you use a Motorola **EVM** compatible board you can download this program to your target board without modifications.

If you use a Motorola **EVB** compatible board (or any other board running the BUFFALO moniator program) you need to **modify** the program locations before you can download this program!

Please change the line:

```
ORG $00ff start of ROM (program)
into:
ORG $c000 start of program RAM
```

This is needed because the EVB memory map doesn't have free RAM at address \$ff00. Most EVB compatible boards have free RAM at address \$c000 but check your board documentation if you get memory errors while downloading.

Please change the last 2 lines:

```
ORG $FFFE reset vector
FDB START set to start of program
into:
* ORG $FFFE reset vector
* FDB START set to start of program
```

This comments out these 2 lines. In the THRSim11 (and also in the EVM) you can load the RESET vector. In an EVB board you can not change the RESET vector.

With the button or the Assemble item from the File menu the contents of the edit window can be converted to machine language. A list window will be opened and the machine code will be loaded in the simulated memory. Depending on the assembler options you selected the listing and the machine codes are also saved on disk in a .LST and a .S19 file.

When you use the **default version** of THRSim11 in combination with an EVB board and made the changes described above you will get the following error:

This message says that the program is not loaded into the simulator memory. So you can **not** simulate this program! But this is no problem in this case because you want to download this program to your target board.

You can adjust the memory map of THRSim11 any way you want and then you will be able to simulate this program.

Now make sure your target board is powered up and connected to a serial port of your PC. THRSim11 uses COM1 by default. You can change this by using the Target, Taget Communication Options... menu option.

Now you can use the button or the Download item from the File menu the machine language can be loaded into the target board.

The green line shows you where the target board's PC register is pointing to.

A special target toolbar pops up as soon as the program is successfully loaded.

Enter the appropriate input values on the target board.


Before we can run the program we have to fill in the input values in the target board's memory. The program Example0.asm will add the contents of address \$0000 (labelled OPER1) to the contents of address \$0001 (labelled OPER2).

Lets for example try to add 5 and 6 together.

By using the Target memory list item from the Target menu you can open a memory list window to fill in these values. As you can see the labels used in the assembler program also appear in this window.


To enter the values 5 and 6 in memory locations \$0000 and \$0001 respectively you first have to select the line where you want to enter a value. If you press return a cursor will appear so you can enter a new value. It is not necessary to press return first you can immediate type in a new value after selecting a line. Press return after entering the value. If there is not enough room to type in an expression you can enlarge the window and try again.

Run the program on the target board.

With the  button or the Target go item from the Execute menu the target board runs the program. The program ends with an endless loop:


```
LOOP BRA LOOP
```

This instruction causes the 68HC11 to wait for a RESET or interrupt.

With the  button or the Target stop item from the Execute menu the execution of the program can **not** be stopped.

The only way to stop the program is to press the (MASTER) RESET or ABORT button on your target board.

But there is a better way. Place a target breakpoint on the last instruction and the program will automatically stop when this instruction is reached. The simplest way to set a breakpoint us to select the last line and press F5.

When you now click the  button the program execution stops when the breakpoint is reached.

Stop the program on the target board.

When the program runs you can't stop the program because THRSim11 can't communicate with the target board while the program is running. The only way to stop the program is to press the (MASTER) RESET or ABORT button on your target board or to set a target breakpoint before you start the program.

Check the output values on the target board.

The program writes the result to memory location \$0002 (labelled SUM). If the program stops the target memory window is automatically refreshed and you can see that the result of the addition of 5 and 6 is \$0B (hexadecimal notation for eleven). The target memory window is not refreshed when the program runs because THRSim11 can't communicate with the target board while the program is running.

If you find it easier to read out the result in decimal you can set the notation to decimal. To do this select the line where you want to change the notation and click the right mouse button or press Alt+F10. A pop up menu appears where you can select the decimal notation.

More things to learn on the target board.

Now you know how to assemble and simply execute a 68HC11 assembler program it is time to learn the answers to the following questions:

- How can I slowly step through the execution of the program?
- How can I execute the program until a certain condition?

You can go directly to one of these questions by clicking them or you can read them sequentially by using the >> button on the button bar.

How can I slowly step through the execution of the program on the target board?

There are many ways to do this. A few of the easy ways are:

- If the target list window is active (selected) you can press Ctrl+F7 or space to execute one instruction. The green line that shows you where the target board's PC register is pointing to will be placed on the next instruction to execute. The window will automatically scroll so the green line will stay visible.
- If the target commands window is active (selected) you can type "T 4" to execute 4 instructions. See EVB T command or EVM T command.
- If the target list window is active (selected) you can execute until a certain line in your program. To do this select the line where you want to stop the execution and click the right mouse button or press Alt+F10. A pop up menu appears where you must select the Target go until item.

To learn more ways see:

- The button.
 - The button.
 - Set a target breakpoint on a line.
-

How can I execute the program on the target board until a certain address is reached?

Set a target breakpoint.

Custom made

The button or the custom made item in the View menu gives the possibility to create a custom window containing your own personal selection of registers, memory locations, pins, etc. When clicking this command a dialog window will appear where you can define the contents of the custom made window. You can also

create more than one custom made window. You can give each custom made window its own window title.

CPU registers

This item in the View menu (Registers submenu) shows the CPU registers of the 68HC11 microcontroller. (A, B, PC, etc..). To alter the value of a register double click it or select it and just type in a new value. (See Working with register windows)

Ports

This item in the View menu (Registers submenu) shows the value of the I/O registers of the 68HC11 microcontroller. To alter the value of an I/O register double click it or select it and just type in a new value. (See Working with register windows)

Timer

This item in the View menu (Registers submenu) shows the value of the timer registers. To alter the value of a timer register double click it or select it and just type in a new value. (See Working with register windows)

Serial

This item in the View menu (Registers submenu) shows the value of the serial port registers. To alter the value of a serial port register double click it or select it and just type in a new value. (See Working with register windows.)

Pulse accumulator

This item in the View menu (Registers submenu) shows the value of the pulse accumulator registers. To alter the value of a pulse accumulator register double click it or select it and just type in a new value. (See Working with register windows)

AD Converter

This item in the View menu (Registers submenu) shows the value of the AD converter registers. To alter the value of an AD converter register double click it or select it and just type in a new value. (See Working with register windows)

Handshake

This item in the View menu (Registers submenu) shows the value of the handshake registers. To alter the value of a handshake register double click it or select it and just type in a new value. (See Working with register windows)

Other Registers

This item in the View menu (Registers submenu) shows the value of the CONFIG, COPRST, HPRI0, INIT, OPTION, and PPROG register. To alter the value of a register double click it or select it and just type in a new value. (See Working with register windows)

Memory list

This item in the View menu (Memory submenu) shows the value of a number of locations in memory. The start address can be entered in a dialog box. To alter the value of a memory location double click it or select it and just type in a new value. (See Working with register windows)

A memory list window has special memory search options.

Memory dump

This item in the View menu (Memory submenu) shows the contents of memory locations starting from a specific address. The start address can be entered in a dialog box. To alter the value of a memory location double click it or select it and just type in a new value. You can also enter an ASCII string.

A memory dump window has special memory search options.

Stack

This item in the View menu (Memory submenu) shows the contents of memory locations around the stack pointer. To alter the value of a memory location double click it or select it and just type in a new value. (See Working with register windows)

A stack window has special memory search options.

Number of clock cycles

This item in the View menu (Memory submenu) shows the number of simulated clock cycles. To alter this value double click it or select it and just type in a new value. (See Working with register windows)

Target custom made

The button or the target custom made item in the Target menu gives the possibility to create a custom window containing your own personal selection of target registers, memory locations, etc. When clicking this command a dialog window will appear where you can define the contents of the target custom made window. You can also create more than one target custom made window. You can give each target custom made window its own window title.

Target CPU registers

This item in the Target menu (Target Registers submenu) shows the CPU registers of the 68HC11 microcontroller. (A, B, P, etc..) on the target board. To alter the value of a register double click it or select it and just type in a new value. (See Working with target register windows)

Target Ports

This item in the Target menu (Target Registers submenu) shows the value of the I/O registers of the 68HC11 microcontroller on the target board. To alter the value of an I/O register double click it or select it and just type in a new value. (See Working with target register windows)

Target Timer

This item in the Target menu (Target Registers submenu) shows the value of the timer registers on the target board. To alter the value of a timer register double click it or select it and just type in a new value. (See Working with target register windows)

Target Serial

This item in the Target menu (Target Registers submenu) shows the value of the serial port registers on the target board. To alter the value of a serial port register double click it or select it and just type in a new value. (See Working with target register windows.)

Target Pulse accumulator

This item in the Target menu (Target Registers submenu) shows the value of the pulse accumulator registers on the target board. To alter the value of a pulse accumulator register double click it or select it and just type in a new value. (See Working with target register windows)

Target AD Converter

This item in the Target menu (Target Registers submenu) shows the value of the AD converter registers on the target board. To alter the value of an AD converter register double click it or select it and just type in a new value. (See Working with target register windows)

Target Handshake

This item in the Target menu (Target Registers submenu) shows the value of the handshake registers on the target board. To alter the value of a handshake register double click it or select it and just type in a new value. (See Working with target register windows)

Target Other Registers

This item in the Target menu (Target Registers submenu) shows the value of the CONFIG, COPRST, HPRI0, INIT, OPTION, and PPROG register on the target board. To alter the value of a register double click it or select it and just type in a new value. (See Working with target register windows)

Target Memory list

This item in the Target menu (Target Memory submenu) shows the value of a number of locations in memory on the target board. The start address can be entered in a dialog box. To alter the value of a memory location

double click it or select it and just type in a new value. (See [Working with target register windows](#))

A target memory list window has special [memory search options](#).

Target Memory dump

This item in the [Target menu](#) (Target Memory submenu) shows the contents of memory locations on the target board starting from a specific address. The start address can be entered in a dialog box. To alter the value of a memory location double click it or select it and just type in a new value. You can also enter an ASCII string.

A target memory dump window has special [memory search options](#).


Target Stack

This item in the [Target menu](#) (Target Memory submenu) shows the contents of memory locations on the target board around the stack pointer. To alter the value of a memory location double click it or select it and just type in a new value. (See [Working with target register windows](#))

A target stack window has special [memory search options](#).


Memory search options.

The [Memory list window](#), [Memory dump window](#), [Stack window](#), [Target memory list window](#), [Target memory dump window](#), and [Target stack window](#) all have special search options.

When you click the  button, select the Find option from the [Search menu](#), or press Ctrl+F the following dialog is shown:

You can choose to search:

- a string within the window. This search is limited to the part of the memory which is shown in the window.
- for a data value in memory. This searches the complete memory map for the specified data value.
- an address. This search displays the specified address.

After you have searched for a string or for a data value you can repeat this search by using the  button, select the Search again option from the [Search menu](#), or press F3.

Working with simulator or target windows.

This part of the help file gives an explanation how to work with the various simulator and target windows. It consists of several parts each of which gives a step-by-step explanation on how to work with that window.

[Registers windows](#). You will always use register windows

[Edit window](#). You will be using this window if you are using the build in assembler.

[List window](#). You will be using this window if you are using the build in assembler. You can use this window to simulate your program.

Disassembler window. You will be using this window if you are **not** using the build in assembler but if you use the simulator to simulate S19 files.

Command window. You can use this window to enter simulator commands.

Target Registers window. You can use this window to inspect and modify the registers on your target board.

Target List window. You will be using this window if you are using the build in assembler. You can use this window to execute your program on your target board.

Target Disassembler window. You will be using this window if you are **not** using the build in assembler but if you run S19 files directly on your target board.

Target Command window. You can use this window to communicate with your target board.

Working with register windows.

The register windows contain different values of memory locations or registers, for example CPU, timer, Port A pins, etc. The windows can be opened by clicking on the appropriate submenu in the View menu in the menu bar. The CPU register window for example can be opened by selecting the submenu CPU registers within the submenu Registers of the View menu.

In the CPU window all CPU registers of the 68HC11 microcontroller are available. Every register name in combination with its value will be displayed on a "line" in the window. Within a register window several actions can be performed:

- Select a line.
 - Search for a line.
 - Select more than one line.
 - Alter the value of a line.
 - Set a breakpoint on a line.
 - Remove a breakpoint from a line.
 - Change the notation of a line.
 - Remove a line.
 - Add a removed line.
-

Working with target register windows.

The target register windows contain different values of target memory locations or target registers, for example CPU, timer, Port A pins, etc. The windows can be opened by clicking on the appropriate submenu in the Target menu in the menu bar. The CPU register window for example can be opened by selecting the submenu Target CPU registers within the submenu Target Registers of the Target menu.

To avoid confusing all the windows displaying information from the target board have a different (light blue) background colour. All windows displaying information from the simulator have a white background.

In the target CPU registers window all CPU registers of the 68HC11 microcontroller on the target board are available. Every register name in combination with its value will be displayed on a "line" in the window. Within a target register window several actions can be performed:

- Select a line.
- Search for a line.

- Select more than one line.
 - Alter the value of a line.
 - Change the notation of a line.
 - Remove a line.
 - Add a removed line.
-

Select a line.

You can select a line within a window in several way's:

- click on it with the left mouse button.
 - use the Up arrow key to select the line above the currently selected line.
 - use the Down arrow key to select the line below the currently selected line.
 - use the Ctrl+Home keys to select the first line.
 - use the Ctrl+End keys to select the last line.
 - use the Page Up key to scroll up a window.
 - use the Page Down key to scroll down a window.
 - search for a line.
-

Search for a line.

Search for a line using the [button](#).

There will appear a dialog box where you can enter your search string. The line containing this string will be selected. The search wraps around automatically.

Search for a line using the key [Ctrl+F](#).

Press Ctrl+F. There will appear a dialog box where you can enter your search string.

Search for a line using the [search menu](#).

Select the menu item Find from the [Search menu](#). There will appear a dialog box where you can enter your search string.

Search for a line using the [pop-up menu](#).

Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Find. There will appear a dialog box where you can enter your search string.

Once you have search for a specific string you can search for this string again.

Search again for a line using the [button](#).

The next line containing the last string searched for will be selected. The search wraps around automatically.

Search again for a line using the key [F3](#).

Press F3. The last search is repeated.

Search for a line using the [search menu](#).

Select the menu item Search again from the [Search menu](#). The last search is repeated.

Search for a line using the [pop-up menu](#).

Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Search again. The last search is repeated.

Select more than one line.

It is possible to select more than one line.

Using the CONTROL key:

Press the control key and hold it down. Click on a line. This line will be shown in a blue bar. Repeat this action on other lines. Release the control key.

Using the Shift key:

Select a line (I call it line A). Press the shift key and hold it down. Select an other line (I call it line B) All lines between line A and Line B will be selected (appear in a blue bar) including line A and line B. Release the shift key.

Using the Space bar.

Press the shift key together with the F8 key. After this you can select lines by pressing the space bar. You can go to the lines you want to select by using the up and down keys.

All selected lines appear in a blue bar. Every action on a selected line will work out on all selected lines. For example selecting a new notation will cause that all selected line will changes to this new notation.

Alter the value of a line.

There are four possibilities to alter the value of a line:

Double click on a value.

If the value is a bit its value will be inverted (toggled). If the value is a byte, a word or a long word a cursor will appear so you can enter a new value. Press return after entering the value. You can use all valid expressions and labels. If there is not enough room to type in an expression you can enlarge the window and try again.

Alter the value of a line with the keyboard.

Select the line. If you press return a cursor will appear so you can enter a new value. It is not necessary to press return first you can immediate type in a new value after selecting a line. Press return after entering the value. If there is not enough room to type in an expression you can enlarge the window and try again.

Increment the value.

Select the line. If you press Ctrl+Numpad+ (Hold the control key and press the plus key on your numeric keypad) the value will be incremented. You can also use the pop-up menu and choose Increment.

Decrement the value.

Select the line. If you press Ctrl+Numpad- (Hold the control key and press the minus key on your numeric keypad) the value will be decremented. You can also use the pop-up menu and choose Decrement.

Press the right mouse button and hold it down to open the pop-up menu. (If you don't have a mouse you can press Alt+F10.)

Enter a value.

A value can be:

- a literal constant:

% binary	e.g. %00101011	= 43 decimal
@ octal	e.g. @53	= 43 decimal
\$ hexadecimal	e.g. \$2B	= 43 decimal
& decimal	e.g. &43	= 43 decimal

' ASCII	e.g. 'A	= 41 hexadecimal
" double ASCII	e.g. "AB	= 4142 hexadecimal

the default (binary, octal, decimal or hexadecimal) depends on the context and on the assembler option you choose.

- **a register name, pin name or label.**
- **an expression:**
- (expression) or {expression} can be used to change the build in operator precedence. For example:

2+3*4	= 14 decimal
(2+3)*4	= 20 decimal
{2+3}*4	= 20 decimal

- **unary operators:** (highest precedence):

+	positive
-	negative = two's complement
~	NOT = one's complement

- **binary operators:** (from high to low precedence)

* / \	times DIV MOD
+ -	plus minus
&	AND
^	EXOR
! or	OR

Valid register names are:

A	register A.
B	register B.
CC or CCR	condition code register.
D	register D.
PC or P	program counter.
S or SP	stack pointer.
X or IX	index register X.
Y or IY	index register Y.
C or CLOCK	number of clock cycles.

Valid pin names are:

PA0 to PA7	port A pins.
PB0 to PB7	port B pins.
PC0 to PC7	port C pins.
PD0 to PD5	port D pins.
PE0 to PE7	port E pins (analog voltage in mV).
STRA STRB	STRA STRB pins.
RESET	RESET pin.

IRQ XIRQ	IRQ XIRQ interrupt.
Vrl Vrh	Reference voltage low end high(AD converter).

Standard labels are:

PORTA, PIOC, PORTC, PORTB, PORTCL, DDRC, PORTD, DDRD, PORTE, CFORC, OC1M, OC1D, TCNTh, TCNTI, TICxh, TICxI, TOCxh, TOCxI, TCTLx, TMSKx, TFLGx, PACTL, PACNT, SPCR, SPSR, SPDR, BAUD, SCCR1, SCCR2, SCSR, SCDR, ADCTL, ADRx, OPTION, COPRST, PPROG, HPRI0, INIT, TEST1 or CONFIG.

You can also define your own labels (see SetLabel command).

A label must start with a letter, underscore (_), or dot (.) and can further contain letters, digits, underscores (_), and dots (.).

The value of every C/C++ variable which is **in scope** can be used as :<name>.

simple variables (also enums)	:count
array variable	:codes
array elements	:codes[2]
struct variable	:date
struct members	:date.month
pointers (also to functions)	:p
pointees	:*p
member of pointer to struct	:p->month
combinations of all of the above	:b[12]->e.a[2][2]

If there is no other <name> with the same name the : may be omitted.

The value may be preceded by ` to denote a value on the target board (including C/C++ variables which are in scope on the target board). For example:

`A	register A on the target board.
`B	register B on the target board.
`:count	C/C++ variable count on the target board.

Set a breakpoint on a line

It is possible to set a breakpoint on a specific line. A breakpoint has a condition: name operator value. This means that the simulator stops executing instructions if the condition name operator value **becomes** true.

A line on which one or more breakpoints are set is displayed with a yellow background or with a yellow foreground when selected. When a breakpoint is reached the appropriate line is displayed with a red background or with a red foreground when selected.

As an example we illustrate setting a breakpoint on register D. The breakpoint will be valid when register D reaches the value \$1234.

Select register D by clicking it with the left mouse button. There will appear a blue bar.

Set the breakpoint using the function key F5.

Press F5. There will appear a dialog box where you can enter the breakpoint condition. This dialog box has the following input fields:

- **Name:** You can select a name here. The name of the line that you selected before pressing F5 is already selected but if you want you can select another name or use the Memory place button to set a breakpoint on the contents of a memory location.

- **Operator:** You can select the operator that will be used in the breakpoint condition here. The == (equal to) is already selected but if you want you can select an other operator.
- **Value:** You can enter the value to be used in the breakpoint condition here.
- **Count:** You can enter a count value here. A counter will be incremented each time the breakpoint condition **becomes** true. If this counter reaches the entered count value the simulation of instructions will be stopped.

Type \$1234 in the value input field. When the D register becomes equal to \$1234 the simulator will stop.

Set the breakpoint using the pop-up menu.

Press the right mouse button and hold it down. A pop-up menu will appear. (If you don't have a mouse you can press Alt+F10.) Select the menu item Set Breakpoint.

The dialog box where you can enter the breakpoint condition appears. Type \$1234 in the value input field. When the D register becomes equal to \$1234 the simulator will stop.

Remove a breakpoint from a line.

It is easy to remove a breakpoint from a line.

Remove a breakpoint using the function key F5.

Select the line and press F5. The breakpoint will be removed. If you have set more than one breakpoint on a line a dialog appears to let you choose which breakpoint(s) must be removed.

Remove a breakpoint using the pop-up menu.

Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Remove breakpoint(s).

If you have set more than one breakpoint on a line a dialog appears to let you choose which breakpoint(s) must be removed.

Change the notation of a line.

It is possible to change the notation of a specific line.

Select the line. Press the right mouse button and hold it down. (If you don't have a mouse you can press Alt+F10.) There will appear a pop-up menu. Select the new notation.

Remove a line.

It is possible to remove a specific line from a window.

Remove a line using the DEL key.

Select the line and press the DEL key.

Remove a line using the pop-up menu.

Select the line and press the right mouse button and hold it down. There will appear a pop-up menu. Select the menu item Remove line.

Add a removed line

It is possible to add a line that has been removed from a window (see [Removing a line](#)). So it is not possible to arbitrary insert a line. (Use a custom made window if you want to do this.)

Add a removed line using the INS key.

Press the INS key. The previously removed line is now inserted. If you have removed more than one line there will appear a selection window with the removed lines.

Add a removed line using the pop-up menu.

Press the right mouse button and hold it down. There will appear a pop-up menu. Select the menu option Add Removed line. If you have removed more than one line there will appear a selection window with the removed lines.

Update register and memory windows

All simulator register and memory windows are automatically updated. This slows down simulation! If you want to speed up the simulation you have to minimise or close all those windows. don't forget to hide the registers shown on the statusbar if you want maximum speed.

The target register and memory windows are also automatically updated. There is no communication between the THRSim11 and your target board when a program is running on your target board. The target register and memory windows are only updated when the program on the target stops running. If you have a lot of target register and memory windows open this will take quite some time. If you want to speed up this update time you have to minimise or close all those windows.

Working with the list window.

The list window contains the assembled code of the 68HC11 program. The list window opens automatically when a 68HC11 program is assembled using the build in assembler.

- Load a 68HC11 assembler file into an edit window using the button or the [Open](#) item from the [File menu](#). (For example open the file Example0.asm.)
- Assemble the program (converting to 68HC11 machine language) using the button or the [Assemble](#) item from the [File menu](#).
- A list window will be opened with the assembled code of the Example0.asm program.

Every address in combination with its code will be placed on a line in the list window.

When using the list window to simulate a program several actions can be performed.

- [Select a line.](#)
- [Search for a line.](#)
- [Set a breakpoint on a line.](#)
- You can press F9 to simulate the program until a breakpoint is found.
- [Remove a breakpoint from a line.](#)
- [Assign a new value to the program counter.](#)
- You can press F7, the space bar, or the enter key to simulate one instruction.
- You can simulate until a certain line in your program. To do this [select the line where you want to stop the simulation](#) and click the right mouse button or press Alt+F10. A pop up menu appears where you must select the Run until item.

Depending on its status a line can be displayed in several colour combinations. [The Meaning of line colours](#) will be explained below.

Working with the target list window.

The target list window contains the assembled code of the 68HC11 program. You can use this window to run a program on your target board.

- Load a 68HC11 assembler file into an edit window using the button or the Open item from the File menu. (For example open the file Example0.asm.)
- Assemble the program (converting to 68HC11 machine language) using the button or the Assemble item from the File menu.
- Download the program to your target board using the button or the Download item from the Target menu.
- A target list window will be opened with the assembled code of the Example0.asm program. The position of the target board's program counter is shown as a line with a green background.

When using the target list window to run a program on your target board several actions can be performed.

- Select a line.
- Search for a line.
- Set a breakpoint on a line.
- You can press Ctrl+F9 to run the program until a breakpoint is found.
- Remove a breakpoint from a line.
- Assign a new value to the program counter on the target board.
- You can press Ctrl+F7, space, or enter to execute one instruction.
- You can run until a certain line in your program is reached. To do this select the line where you want to stop the execution and click the right mouse button or press Alt+F10. A pop up menu appears where you must select the Target Go Until item.

Depending on its status a line can be displayed in several colour combinations. The Meaning of line colours will be explained below.

Set a breakpoint in the list window.

You can set a simple breakpoint in a list window and/or a target breakpoint on a line in the target list window. The simulation or execution of instructions will stop if this line of the program is reached. It is (of course) only possible to set a breakpoint in the list window on a piece of code, so don't try to perform this action on an empty line or a comment line. A line on which a breakpoint is set is displayed with a yellow background or with a yellow foreground when selected. When a breakpoint is reached the appropriate line is displayed with a red background or with a red foreground when selected.

Set the breakpoint using the function key F5.

Select the line and press F5. The breakpoint will be set.

Set the breakpoint using the pop-up menu.

Press the right mouse button and hold it down. A pop-up menu will appear. (If you don't have a mouse you can press Alt+F10.) Select the menu item Set Breakpoint or Set Target Breakpoint.

Remove a breakpoint from the list window.

It is easy to remove a breakpoint and/or target breakpoint from a line.

Remove a breakpoint using the function key F5.

Select the line and press F5. The breakpoint will be removed.

Remove a breakpoint using the pop-up menu.

Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Remove Breakpoint or Remove Target Breakpoint.

Set the program counter in the list window.

Select a line within the list window (Containing code). Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Place PC on.

Set the target board's program counter in the list window.

Select a line within the target list window (Containing code). Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Place Target PC on.

Working with the disassembler window.

The disassembler window contains the disassembled code of the 68HC11 program which is loaded in the simulator. The window can be opened by clicking the Disassembler item from the View menu.

This window shows a disassembled part of the simulator's memory (possibly a program). The disassembler begins disassembling instructions starting at the current value of the program counter. This line will be the first line in the disassembler window. Because 68HC11 instructions are of variable length and the beginning of a machine instruction is not marked in machine code memory can not be disassembled backwards! So you can not scroll up past the firstly disassembled line. You can scroll down as far as you want. Every address in combination with its code will be placed on a line of the disassembler window. Our disassembler doesn't display labels because labels can have the same value and in this case it is impossible to make the right decision on which label to show. If you want to use labels you should use the list window. Within the disassembler window several actions can be performed.

- Select a line.
- Search for a line.
- Alter the code of a line.
- Set a breakpoint on a line.
- You can press F9 to simulate the program until a breakpoint is found.
- Remove a breakpoint from a line.
- You can press F7 or the space bar to simulate one instruction.
- You can simulate until a certain line in your program. To do this select the line where you want to stop the simulation and click the right mouse button or press Alt+F10. A pop up menu appears where you must select the Run until item.
- Assign a new value to the program counter.

Depending on its status a line can be displayed in several colour combinations. The Meaning of line colours will be explained below.

Working with the target disassembler window.

The target disassembler window contains the disassembled code of the downloaded 68HC11 program. The window can be opened by clicking the Target Disassembler item from the Target menu.

This window shows a disassembled part of the target board memory (possibly a program). The target disassembler begins disassembling instructions starting at the current value of the target board's program counter. This line will be the first line in the target disassembler window. Because 68HC11 instructions are of variable length and the beginning of a machine instruction is not marked in machine code memory can not be disassembled backwards! So you can not scroll up past the firstly disassembled line. You can scroll down as far as you want. Every address in combination with its code will be placed on a line of the target disassembler window. Our disassembler doesn't display labels because labels can have the same value and in this case it is impossible to make the right decision on which label to show. If you want to use labels you should use the list window. Within the target disassembler window several actions can be performed.

- Select a line.
- Search for a line.
- Alter the code of a line.
- Set a breakpoint on a line.
- You can press Ctrl+F9 to run the program until a breakpoint is found.
- Remove a breakpoint from a line.
- You can press Ctrl+F7 or Ctrl+space to execute one instruction.
- Remove a breakpoint from a line.
- You can run until a certain line in your program is reached. To do this select the line where you want to stop the execution and click the right mouse button or press Alt+F10. A pop up menu appears where you must select the Target Go Until item.
- Assign a new value to the program counter.
- You can update the window by pressing the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Update Window.

Depending on its status a line can be displayed in several colour combinations. The Meaning of line colours will be explained below.

Alter the code in the disassembler window.

Select the line. If you press return or double click the line a cursor will appear so you can enter a new instruction in assembler code. It is possible to use labels in the code you type in but the disassembler will display the numerical value. It is not necessary to press return first you can also immediate type in a new assembler instruction after selecting a line. Press return after entering the instruction. If there is not enough room to type in an instruction you can enlarge the window and try again.

Set a breakpoint in the disassembler window.

You can set a simple breakpoint on a line in the disassembler window or a target breakpoint on a line in the target disassembler window. The simulation or execution of instructions will stop if this line of the program is reached. A line on which a breakpoint is set is displayed with a yellow background or with a yellow foreground when selected. When a breakpoint is reached the appropriate line is displayed with a red

background or with a red foreground when selected.

Set the breakpoint using the function key F5.

Select the line and press F5. The breakpoint will be set.

Set the breakpoint using the pop-up menu.

Press the right mouse button and hold it down. A pop-up menu will appear. (If you don't have a mouse you can press Alt+F10.) Select the menu item Set Breakpoint or Set Target Breakpoint..

Remove a breakpoint from the disassembler window.

It is easy to remove a breakpoint or a target breakpoint from a line.

Remove a breakpoint using the function key F5.

Select the line and press F5. The breakpoint will be removed.

Remove a breakpoint using the pop-up menu.

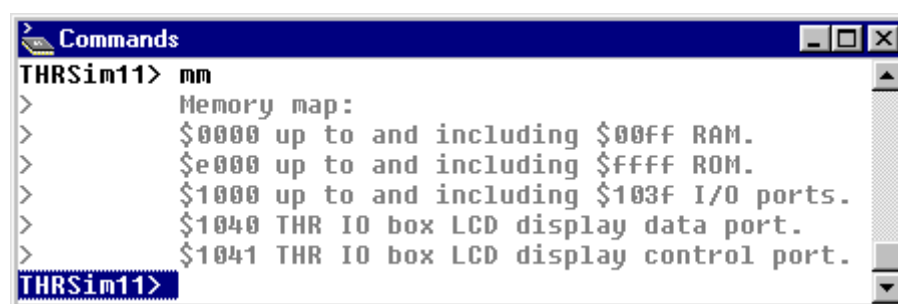
Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Remove Breakpoint or Remove Target Breakpoint.

Set the program counter in the disassembler window.

Select a line within the disassembler window or target disassembler window. Press the right mouse button and hold it down. There will appear a pop-up menu. (If you don't have a mouse you can press Alt+F10.) Select the menu item Place PC on or Place Target PC on.

Working with the command window.

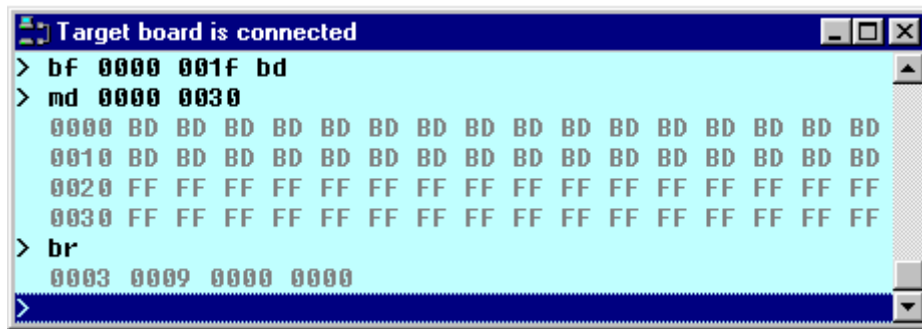
The command window can be used to enter commands. The window can be opened by clicking the Command Window item from the View menu.



Within the command window you can enter commands. You can also select a command you entered before and execute it again. It is also possible to edit a command you entered before.

Working with the target command window.

The target command window can be used to communicate with the target board. The window can be opened by clicking the Target Command Window item from the Target menu.



Within the command window you can enter target commands. You can also select a command you entered before and execute it again. It is also possible to edit a command you entered before.

Meaning of different list line colours.

Within the list windows several lines can have different colours.

The meanings of the different colours in the **simulator** list window are:

\$c000 86 01	start	ldaa	#01	Normal line.
\$c000 86 01	start	ldaa	#01	This line is selected.
\$c000 86 01	start	ldaa	#01	This line contains a <u>breakpoint</u> .
\$c000 86 01	start	ldaa	#01	This line contains a <u>breakpoint</u> and is selected.
\$c000 86 01	start	ldaa	#01	A breakpoint hit occurred.
\$c000 86 01	start	ldaa	#01	A breakpoint hit occurred and this line is selected.
\$c000 86 01	start	ldaa	#01	The PC has reached this line.
\$c000 86 01	start	ldaa	#01	The PC has reached this line and this line is selected.

The meanings of the different colours in the **target** list window are:

\$c000 86 01	start	ldaa	#01	Normal line.
\$c000 86 01	start	ldaa	#01	This line is selected.
\$c000 86 01	start	ldaa	#01	This line contains a <u>breakpoint</u> .
\$c000 86 01	start	ldaa	#01	This line contains a <u>breakpoint</u> and is selected.
\$c000 86 01	start	ldaa	#01	A breakpoint hit occurred.
\$c000 86 01	start	ldaa	#01	A breakpoint hit occurred and this line is selected.
\$c000 86 01	start	ldaa	#01	The PC has reached this line.
\$c000 86 01	start	ldaa	#01	The PC has reached this line and this line is selected.

Meaning of different disassembler line colours.

Within the disassembler windows several lines can have different colours.

The meanings of the different colours in the **simulator** disassembler are:

\$C000	LDAA #01	Normal line.
\$C000	LDAA #01	This line is selected.
\$C000	LDAA #01	This line contains a <u>breakpoint</u> .
\$C000	LDAA #01	This line contains a <u>breakpoint</u> and is selected.
\$C000	LDAA #01	A breakpoint hit occurred.
\$C000	LDAA #01	A breakpoint hit occurred and this line is selected.
\$C000	LDAA #01	The PC has reached this line.
\$C000	LDAA #01	The PC has reached this line and this line is selected.

The meanings of the different colours in the **target** disassembler are:

\$C000	LDAA #01	Normal line.
\$C000	LDAA #01	This line is selected.
\$C000	LDAA #01	This line contains a <u>breakpoint</u> .
\$C000	LDAA #01	This line contains a <u>breakpoint</u> and is selected.
\$C000	LDAA #01	A breakpoint hit occurred.
\$C000	LDAA #01	A breakpoint hit occurred and this line is selected.
\$C000	LDAA #01	The PC has reached this line.
\$C000	LDAA #01	The PC has reached this line and this line is selected.

Example programs

With this simulator three example programs are included. You can use these programs to try the assembler and simulator.

- [Example0.asm](#)
- [Example1.asm](#)
- [Example2.asm](#)

Example program Example0.asm

This is a very simple program that adds the contents of address \$0000 to the contents of address \$0001 and stores the resulting sum in the memory location at address \$0002. If there is an overflow the contents of address \$0003 is set to \$FF else address \$0003 is cleared.

To simulate the program **Example0.asm** you have to take the following actions:

- [Load the source code in the editor.](#)
- [Assemble the source code to machine codes.](#)
- [Enter the appropriate input values.](#)
- [Run the program.](#)
- [Check the output values.](#)
- [Reset the 68HC11.](#)

To execute the program **Example0.asm** on your EVM or EVB target board you have to take the following actions:

- [Download the machine code into the target board.](#)
- [Enter the appropriate input values.](#)
- [Run the program.](#)

- Stop the program.
- Check the output values.

Example program Example1.asm

This program calculates the decimal value (in ASCII code) of an 8 bits unsigned binary number.

Example program Example2.asm

This program calculates the decimal value (in ASCII code) of an 8 bits unsigned binary number present at external pins PC7 to PC0. The result will be displayed on the LCD display of the I/O box. Each conversion is triggered by a rising edge of the Strobe A pin. The program uses the IRQ interrupt.

THRSim11_options.txt

Some options of THRSim11 are saved in the file THRSim11_options.txt. Users can edit this file to change the memory map or to set the CONFIG register.

Changing the memory map.

Setting the CONFIG register.

Editor option.

GNU tools options.

Serial window options.

Target connect option.

Other options.

Important note: You have to **close THRSim11 before you edit** the **THRSim11_options.txt** file because this file will be overridden with the old values when THRSim11 closes.

Changing the memory map

The memory map can be configured **much easier** with the Memory Configuration Tool

Memory map.

You can use the MAP command to review the memory mapping and you can write to the INIT register within the first 64 clock cycles after RESET to reposition the internal 256-byte RAM and/or 64-byte register space to any 4K page boundary in the 64K-byte memory map.

Initial memory map.

The memory map of THRSim11 **can** be changed. You can use the MAP command to review the memory mapping and you can write to the INIT register within the first 64 clock cycles after RESET to reposition the internal 256-byte RAM and/or 64-byte register space to any 4K page boundary in the 64K-byte memory map. The memory map after installing THRSim11 looks like this:

\$x000-\$x0FF	256 bytes RAM, x is initially 0 but can be changed by using the INIT register.
\$y000-\$y03F	I/O registers, y is initially 1 but can be changed by using the INIT register.
\$B600-\$B7FF	512 bytes RAM. Used to simulate the internal EEPROM memory.
\$E000-\$FFFF	8K bytes ROM.

This memory map can be configured with the Memory Configuration Tool.

User defined memory map.

The memory map is created with information which is read on program start-up time from the file `thrsim11_options.txt`. The **easy** way to change the memory map is using the Memory Configuration Tool but you can also edit the following options. You can use the MAP command to review the memory mapping.

The memory map is created with information which is read on program start-up time from the file `thrsim11_options.txt`.

Important note: You have to **close THRSim11 before you edit the THRSim11_options.txt** file because this file will be overridden with the old values when THRSim11 closes.

The following memory configuration options can be used:

- **IOStart address**

Defines the first *address* of the IO register block. The last three hexadecimal digits of *address* are ignored and are always read as 0. This is done to make the IO block start at the beginning of a 4K-bytes memory page. The IO register block is always 0x40 bytes long. The first hexadecimal digit of *address* is shown in the last hexadecimal digit (low nibble) of the INIT register. You can write to the INIT register within the first 64 clock cycles after RESET to reposition the 64-byte register space to any 4K page boundary in the 64K-byte memory map.

- **RAMnSize size**

Defines the *size* of RAM block number *n* (*n*=0..3). The last two hexadecimal digits of *size* are ignored and are always read as 0. This is done to make the RAM block an integer number of 256-bytes memory pages. If RAM0Size is set lower than 0x100 RAM0Size is forced to 0x100. Because a system without RAM is useless. RAM1Size, RAM2Size and RAM3Size may be set to 0x0.

- **RAM0Start address**

Defines the first *address* of the RAM block number 0. The last three hexadecimal digits of *address* are ignored and are always read as 0. This is done to make the first RAM block start at the beginning of a 4K-bytes memory page. The first hexadecimal digit of *address* is shown in the first hexadecimal digit (high nibble) of the INIT register. You can write to the INIT register within the first 64 clock cycles after RESET to reposition the first RAM block to any 4K page boundary in the 64K-byte memory map. If $\text{RAM0Start} + \text{RAM0Size} - 1 > \FFFF then the end of RAM Block 0 is forced to 0xFFFF.

- **RAMnStart address**

Defines the first *address* of RAM block number *n* (*n*=1..3). The last two hexadecimal digits of *address* are ignored and are always read as 0. This is done to make these RAM blocks start at the beginning of a 256-bytes memory page. If $\text{RAMnStart} + \text{RAMnSize} - 1 > \FFFF then the end of RAM block *n* is forced to 0xFFFF.

- **ROMnSize size**

Defines the *size* of ROM block *n* (*n*=0..3). The last two hexadecimal digits of *size* are ignored and are always read as 0. This is done to make the ROM blocks an integer number of 256-bytes memory pages. ROMnSize may be set to 0x0.

- **ROMnEnd address**

Defines the last *address* of ROM block *n* (*n*=0..3). The last two digits should be FF. If this is not the case the nearest lower number that fulfils this need is taken. This is done to make the ROM blocks end at the ending of a memory page. If $\text{ROMnEnd} - \text{ROMnSize} + 1 < \0000 then the start of ROM block *n* is forced to 0x0000. If ROMnSize is set to zero ROMnEnd is ignored.

If there is no memory mapped in the memory page \$FF00 - \$FFFF THRSim11 maps an **extra** block of ROM in this area because the interrupt vectors are positioned on that (last) memory page.

You can use the INIT register to reposition the first RAM block and/or the 64-byte register space to any 4K page boundary in the 64K-byte memory map. See section 3.3.1 of [M68HC11RM/AD](#).

Memory mapped devices.

At the moment there is one memory mapped device (an [LCD display](#)) that can be used within THRSim11. The memory map is created with information which is read on program start-up time from the file `thrsim11_options.txt`.

Important note: You have to **close THRSim11 before you edit the THRSim11_options.txt** file because this file will be overridden with the old values when THRSim11 closes.

The following memory configuration options can be used to map the LCD display in the memory map:

- [LCDDisplayEnabled](#) 1

This enables the memory mapping of an LCD display in the address space of the 68HC11.

- [LCDDisplayDataRegister](#) *address*

Defines the *address* of the LCD Display Data register. This value is ignored if DisplayEnabled=0.

- [LCDDisplayControlRegister](#) *address*

Defines the *address* of the LCD Display Control register. This value is ignored if DisplayEnabled=0.

Memory map inspection.

Use the [MAP command](#) or [Memory Map menu option](#) to inspect the current memory mapping. If certain areas overlap the priority is as follows:

- IO has higher priority than RAM. So if IO and RAM overlap you can only use the IO registers.
- RAM has a higher priority than ROM.
- ROM has a higher priority than Memory mapped devices (e.g., LCD display).

Setting the CONFIG register

Because the EEPROM is not simulated the value of the EE-PROM CONFIG location is stored as an option. You can change this option by editing the `THRSim11_options.txt` file:

Important note: You have to **close THRSim11 before you edit the THRSim11_options.txt** file because this file will be overridden with the old values when THRSim11 closes.

- [EEPROMConfig](#) *value*

Value is loaded into the CONFIG register when the simulator is started. Only bit2 of *value* will be used (as value for NOCOP). When THRSim11 comes out of the factory the CONFIG register is set to \$0E.

The CONFIG register contains 4 control bits:

- [NOSEC](#) EEPROM Security Disabled

Because THRSim11 doesn't support EEPROM this bit always read as 1 and can not be changed.

- [NOCOP](#) COP Watchdog System Disabled

The default erased state of this bit corresponds to COP system off. 1 = The COP system is disabled and does not generate system resets. 0 = The COP system is enabled as the MCU comes out of reset. **The 68HC11 COP system is fully simulated in THRSim11 version 3.20 and higher.**

- [ROMON](#) Enable On-Chip ROM

The ROMON control bit in the EEPROM-based CONFIG register is overridden in normal single-chip mode to force the internal 8-Kbytes ROM on. This procedure is required because there must be a valid reset vector for the MCU to operate in a logical manner. **Because THRSim11 only simulates the single-chip mode this bit**

is always read as 1 and can not be cleared.

- [EEON](#) Enable On-Chip EEPROM

The default erased state of this bit corresponds to EEPROM enabled. 1 = The 512-byte on-chip EEPROM memory enabled at locations \$B600-\$B7FF. 0 = The 512-byte EEPROM is disabled and takes no space in the memory map. **Because the EEPROM is not simulated this bit is always read as 0 and can not be set.**

Editor options

You can use the internal editor of THRSim11 or your own favorite editor to edit source files. You can change the editor options by editing the THRSim11_options.txt file.

Important note: You have to **close THRSim11 before you edit the THRSim11_options.txt** file because this file will be overridden with the old values when THRSim11 closes.

- [HLLEditorExternalPath](#) and [HLLEditorExternalPathsRelativePath](#)

You should set these options to specify the directory where the external editor you want to use is located.

Example 1:

[HLLEditorExternalPath](#) C:\Program Files\THRSim11\SciTE\

[HLLEditorExternalPathsRelativePath](#) 0

Example 2:

[HLLEditorExternalPath](#) ..\SciTE\

[HLLEditorExternalPathsRelativePath](#) 1

When [HLLEditorExternalPathsRelativePath](#) is set to 1 the path is relative to the ???\THRSim11\program directory.

You should also set [HLLEditorExternalExeFile](#) and [AlwaysUseExternalEditor](#) or [HLLEditorUseExternal](#) before THRSim11 will use the external editor.

- [HLLEditorExternalExeFile](#)

You should set this option to specify the executable filename of the external editor you want to use.

Example:

[HLLEditorExternalExeFile](#) SciTE.exe

You should also set [HLLEditorExternalPath](#) and [AlwaysUseExternalEditor](#) or [HLLEditorUseExternal](#) before THRSim11 will use the external editor.

- [AlwaysUseExternalEditor](#)

When you set this option to 1 THRSim11 uses the external editor (specified by the [HLLEditorExternalPath](#) and [HLLEditorExternalExeFile](#) options) for **all** file types.

- [HLLEditorUseExternal](#)

When you set this option to 1 THRSim11 uses the external editor (specified by the [HLLEditorExternalPath](#) and [HLLEditorExternalExeFile](#) options) only for **.C**, **.CPP**, **.H**, **.S**, **.LD**, and, **makefile** files. The internal editor is used for all other types (except when [AlwaysUseExternalEditor](#) is set to 1).

- [HLLEditorUseExternalGotoLineOption](#) and [HLLEditorExternalGotoLineOption](#)

When you set the [HLLEditorUseExternalGotoLineOption](#) option to 1 THRSim11 uses the [HLLEditorExternalGotoLineOption](#) to open a file on a specific line when appropriate.

Example:

[HLLEditorExternalGotoLineOption](#) -goto:

[HLLEditorUseExternalGotoLineOption](#) 1

- [HLLEditorUseExternalGotoColumnOption](#) and [HLLEditorExternalGotoColumnOption](#)

When you set the [HLLEditorUseExternalGotoColumnOption](#) option to 1 THRSim11 uses the [HLLEditorExternalGotoColumnOption](#) to open a file on a specific column when appropriate.

Example:

[HLLEditorExternalGotoColumnOption](#) ,

[HLLEditorUseExternalGotoColumnOption](#) 1

The [HLLEditorExternalGotoColumnOption](#) is placed immediate after the [HLLEditorExternalGotoLineOption](#) without a separating space.

- [SpacesPerTabForHLL](#) 3

With this option you can specify the TAB stops used in the internal editor for **.C**, **.CPP**, and **.H** files.

- [SpacesPerTabForAS](#) 8

With this option you can specify the TAB stops used in the internal editor for **.S** files.

- [SpacesPerTabForASM](#) 8

With this option you can specify the TAB stops used in the internal editor for **.ASM** files.

- [SpacesPerTabForCMD](#) 8

With this option you can specify the TAB stops used in the internal editor for **.CMD** files.

- [SpacesPerTabForOthers](#) 8

With this option you can specify the TAB stops used in the internal editor for all **other** files (not mentioned above).

GNU tools options

You can use THRSim11 in combination with the GNU Development Chain for 68HC11 and the GNU Utilities for Windows. If you install THRSim11 with C support these tools are automatically installed and these options are defined automatically. You only need to define these options by hand if you installed the GNU Development Chain for 68HC11 or the GNU Utilities for Windows in some other directory.

Important note: You have to **close THRSim11 before you edit** the **THRSim11_options.txt** file because this file will be overridden with the old values when THRSim11 closes.

- [HLLGNUGCCPath](#) and [HLLGNUPathsAreRelativePaths](#)

You should set these options to specify the directory where the GNU Development Chain for 68HC11 executables (**m6811-elf-gcc**, **m6811-elf-as**, **m6811-elf-ld**, etc) are located.

Example 1:

[HLLGNUGCCPath](#) C:\Program Files\THRSim11\gcc\bin\

[HLLGNUPathsAreRelativePaths](#) 0

Example 2:

[HLLGNUGCCPath](#)..\gcc\bin\

[HLLGNUPathsAreRelativePaths](#) 1

When [HLLGNUPathsAreRelativePaths](#) is set to 1 the path is relative to the ???\THRSim11\program directory.

- [HLLGNUUtilitiesPath](#) and [HLLGNUPathsAreRelativePaths](#)

You should set these options to specify the directory where the GNU Utility tools for Windows executables (**make**, **rm**, etc) are located.

Example 1:

[HLLGNUUtilitiesPath](#) C:\Program Files\THRSim11\utils\

[HLLGNUPathsAreRelativePaths](#) 0

Example 2:

[HLLGNUUtilitiesPath](#)..\utils\

[HLLGNUPathsAreRelativePaths](#) 1

When [HLLGNUPathsAreRelativePaths](#) is set to 1 the path is relative to the ???\THRSim11\program directory.

Serial window options

Some serial windows options that can only be set by changing the THRSim11_options.txt file are:

- [SerialTransmitEnterCode](#) 0

This options controls which character(s) is/are send when the user types an "Enter" on the keyboard. This caused the start of a new line in the Serial Transmitter Window. The default value 0 causes a CR (Carriage Return) character to be send. When you set this option to 1 a LF (Line Feed) character is send. When you set

this option to 2 a CR character immediately followed by a LF character is send.

- [SerialReceiveEnterCode 2](#)

This options controls the start of a new line in the Serial Receiver Window. The default value 2 causes a new line to be started when a CR immediately followed by a LF is received. When you set this option to 0 a new line is started when a CR is received. When you set this option to 1 a new line is started when a LF is received.

- [SerialReceiveDisplayNUL<000>Character 0](#)

When the Serial Receiver Window receives a NUL character it is ignored when this option is 0. When this option is set to 1 a NUL character is displayed as <000>.

- [SerialReceiveDisplayBEL<007>Character 0](#)

When the Serial Receiver Window receives a BEL character the PC speaker sounds a beep when this option is 0. When this option is set to 1 a BEL character is also displayed as <007>.

- [SerialReceiveDisplayLF<010>Character 0](#)

When the Serial Receiver Window receives a LF (Line Feed) character a new line may be started depending on the [SerialReceiveEnterCode](#) option. When the [SerialReceiveDisplayLF<010>Character](#) option is set to 1 a LF character is always displayed as <010>.

- [SerialReceiveDisplayCR<013>Character 0](#)

When the Serial Receiver Window receives a CR (Carriage Return) character a new line may be started depending on the [SerialReceiveEnterCode](#) option. When the [SerialReceiveDisplayCR<013>Character](#) option is set to 1 a CR character is always displayed as <013>.

Important note: You have to **close THRSim11 before you edit** the **THRSim11_options.txt** file because this file will be overridden with the old values when THRSim11 closes.

Target connect option

When you use an USB to serial converter cable to communicate to your 68HC11 target board this option can be helpful:

- [TargetUSBToSerialInterfaceF5U109 0](#)

This option must be set to 1 when you use a Belkin USB PDA adapter cable F5U109. The serial driver which comes with this cable does not support the normal serial port API and therefore THRSim11 can only use a subset of this API to communicate with your board. If you use an other USB to serial converter and THRSim11 can not communicate with your board you can also try to set this option to 1.

Important note: You have to **close THRSim11 before you edit** the **THRSim11_options.txt** file because this file will be overridden with the old values when THRSim11 closes.

Other options

Some other options that (only) can be set by changing the THRSim11_options.txt file are:

- [InitRegisterPCwithRandomValue 0](#)

When you set this option to 1 the PC register is initialised with a random value when THRSim11 starts. Otherwise PC is initialised with the first ROM address.

- [InitRegisterSPwithRandomValue 0](#)

When you set this option to 1 the SP register is initialised with a random value when THRSim11 starts. Otherwise SP is initialised with the last RAM address.

- [InitRegisterDwithRandomValue 0](#)

When you set this option to 1 the D register is initialised with a random value when THRSim11 starts. Otherwise D is always initialised with the same value \$bdxx (xx = last two digits of the year number).

- [CommandWindowMaxNumberOfLines](#) 0x400

The number of lines in the [Command Window](#) is truncated to [CommandWindowMaxNumberOfLines](#). This value must be specified in hexadecimal notation (0x400 = 1024 decimal).

- [MaxMemoryNameLength](#) 0x28

In the [Memory List Window](#) and [Target Memory List Window](#) the names of memory locations (which includes all labels set to a specific address) are truncated to [MaxMemoryNameLength](#) characters. This value must be specified in hexadecimal notation (0x28 = 40 decimal).

Important note: You have to **close THRSim11 before you edit** the [THRSim11_options.txt](#) file because this file will be overridden with the old values when THRSim11 closes.

Target communications

THRSim11 can communicate with the Motorola EVM and EVB boards or with any other board running the BUFFALO monitor program. This monitor program can be downloaded (for free) from the Motorola website.

You can start communicating by using the [Connect](#) option from the [Target menu](#).

If the communication with the target fails an error message is displayed. You can monitor the communication with the target board by opening a [Target Command Window](#) from the [Target menu](#).

The connection with the target can be closed by using the [Disconnect](#) option from the [Target menu](#).

If the communication fails and you are sure the serial cable is connected you can try to change the [Target Communication Options](#) from the [Target menu](#).

Target connect

THRSim11 can communicate with the Motorola EVM and EVB boards or with any other board running the BUFFALO monitor program. This monitor program can be downloaded (for free) from the Motorola website.

You can start communicating by using the [Connect](#) option from the [Target menu](#).

If the connection is opened successfully the [Target Toolbar](#) is opened.

If the communication fails and you are sure the serial cable is connected you can try to change the [Target Communication Options](#) from the [Target menu](#).

If you use an USB to serial converter cable you may have to change an option in the [thrsim11_options.txt](#) file.

The status of the target connection is permanently shown in the [statusbar](#).

Target disconnect

THRSim11 can communicate with the Motorola EVM and EVB boards or with any other board running the BUFFALO monitor program. This monitor program can be downloaded (for free) from the Motorola website.

You can stop communicating by using the Disconnect option from the Target menu.

If the connection is closed successful the Target Toolbar is also closed.

The status of the target connection is permanently shown in the statusbar.

Target command window

You can communicate (almost) directly to your target board by using the Target command window. You can use any label set in THRSim11 in your target commands.

Note: It is almost never needed to use the target command window. It is much easier to work with the other target windows.

THRSim11 can currently work with:

- The Motorola EVM board.
- The Motorola EVB or EVBU board or any other board running the Motorola BUFFALO monitor program. This monitor program can be downloaded for free from the Motorola website.

When you press your right mouse button or when you press Alt+F10 a pop-up menu appears which allows you to select one of the most common commands.

Target command: A

A <byte>

or

A=<byte>

Note: it is easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The value <byte> is written to the target's A register.

Target command: B

B <byte>

or

B=<byte>

Note: it is easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The value <byte> is written to the target's B register.

Target command: C

C <byte>

or
C=<byte>
or
CRC <byte>
or
CRC<byte>

Note: it is easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The value <byte> is written to the target's CCR register.

Target command: D

D <word>
or
D=<word>

Note: it is easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The value <word> is written to the target's A and B registers.

Target command: P

P=<word>
or
PC <word>
or
PC=<word>

Note: it is easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The value <word> is written to the target's P register.

Target command: S

S <word>
or
S=<word>

Note: it is easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The value <word> is written to the target's S register.

Target command: X

X <word>
or
X=<word>

Note: it is easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The value <word> is written to the target's X register.

Target command: Y

Y <word>
or
Y=<word>

Note: it is easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The value <word> is written to the target's Y register.

Target command: RS

RS

The target CPU register contents are copied to the simulator's CPU registers.

Target command: RT

RT

The simulator CPU register contents are copied to the target board's CPU registers.

Target command: MS

MS <address1> <address2>

The target memory contents from <address1> upto and including <address2> are copied to the simulator's memory. If you do not specify both addresses correctly a dialog box pops up which you can use to specify the memory range to copy.

Target command: MT

MT [<address1>](#) [<address2>](#)

The simulator memory contents from <address1> upto and including <address2> are copied to the target board's memory. If you do not specify both addresses correctly a dialog box pops up which you can use to specify the memory range to copy.

Target command: ET

ET

The simulator EEPROM memory contents from \$B600 upto and including \$B7FF are copied to the target board's memory. This command is just an abbreviation for:

MT B600 B7FF

Target command: BS

BS

The target breakpoints are copied to the simulator's PC breakpoints.

Target command: BT

BT

The simulator PC breakpoints are copied to the target board's breakpoint table.

Target command: TS

TS

This command is translated into an EVM T <count> command or EVB T <count> command. The value of <count> is calculate to fill one screen full of output.

Motorola EVM commands

All EVM commands and some extra commands are supported. See for a complete description the Motorola 68HC11 EVM User's manual.

EVM commands concerning CPU registers:

- Register display (RD)
- Register modify (RM)

Extra commands concerning CPU registers:

- change A register (A)
- change B register (B)
- change CC register (C)
- change D register (D)
- change P register (PC)
- change S register (S)
- change X register (X)
- change Y register (Y)
- Copy CPU registers from simulator to target (RT)
- Copy CPU registers from target to simulator (RS)

EVM commands concerning memory:

- Memory display (MD)
- Memory modify (MM)
- Assemble/disassemble (ASM)
- Block fill (BF)

Extra commands concerning memory:

- Copy memory from target to simulator (MS)
- Copy memory from simulator to target (MT)
- Copy EEPROM memory from simulator to target (ET)

EVM commands concerning breakpoints:

- Set breakpoint (BR)
- Remove breakpoint (NOBR)
- Proceed after breakpoint (P)

Extra commands concerning breakpoints:

- Copy PC breakpoints from simulator to target (BT)
- Copy breakpoints from target to simulator (BS)

EVM commands concerning the execution of a program

- Load S-records from file (LOAD)
- Trace program (T)
- Execute program (G)

Extra command concerning the execution of a program

- Trace Screen full (TS)

General EVM command

- Commands quick overview (HELP)

EVM commands concerning EEPROM

- Erase individual EEPROM bytes (ERASE)
- Bulk erase EEPROM or CONFIG register (BULK)
- Program EEPROM via pseudo EEPROM (PROG)
- Check EEPROM (CHCK)
- Copy EEPROM to user map (COPY)
- Verify EEPROM against user map (VERF)

EVM commands concerning the HOST connection.

- Baud rate select for host I/O port (SPEED)
- Transparent mode (TM)

EVM command: **ASM**

ASM <address>

Note: it is much easier to use the editor and assembler build into THRSim11 and download the S records produced by this assembler. If you use the ASM command your program is not saved in a file.

The assembler/disassembler is an interactive assembler/editor. Each source line is converted into the proper machine language code and is stored in memory starting at <address> on a line-by-line basis at the time of entry. In order to display an instruction, the machine code is disassembled and the instruction mnemonic and operands are displayed. All valid opcodes are converted to assembly language mnemonic. All invalid opcodes return a Form Constant Byte (FCB) conversion.

Assembler input must have exactly one space between the mnemonic and the operand. There must be no space between the operand and the index specification (,X) except in the case of indexed no offset. Assembler input must be terminated by a carriage return. No comments, etc., are allowed after the instruction input.

After each new assembler input line, the new line is disassembled for the user before stepping to the new instruction. The new line may assemble to a different number of bytes than the previous one. Branch address offsets are automatically calculated by the assembler, thus the address (or label) is inputted as the operand rather than an offset value.

The assembler is terminated by pushing the ESC key or by entering a period (.) followed by a carriage return as the only entry on the command input line. Entering a carriage return alone on an input line steps to the next instruction.

EVM command: BF

BF <address1> <address2> <data>

or

FILL <address1> <address2> <data>

The BF command allows the user to repeat a specific pattern throughout a determined user memory range. The <memory from <address1> to <address2> is filled with <data>. If an invalid address is specified, the invalid address and the message "BAD MEMORY" is displayed.

EVM command: BR

BR [<address>]...

or

SB [<address>]...

Note: it is much easier to use the Target Set or Target List menu item from the Breakpoint menu.

The BR or SB command sets the <address> into the breakpoint address table. During execution of the user program, a debug halt occurs immediately preceding the execution of any instruction address in the breakpoint table.

The user should not place a breakpoint on a software interrupt SWI instruction because this is the instruction that the monitor uses to breakpoint/single step a user program. The user may use the SWI instruction in the user program.

A maximum of five breakpoints may be set. After setting the breakpoint, the current breakpoint addresses, if any, are displayed.

EVM command: BULK

BULK [[<address>](#)]

Note: prior to entering this command, the user must follow the EEPROM erasing procedure as described in paragraph 3.7.2. of the EVM user's manual.

The BULK command can be invoked with or without an [<address>](#). If the command is invoked without an [address](#), a default to location \$B600 occurs and 512 bytes of EEPROM are bulk erased. If an [address](#) is specified with the command, only the CONFIG register [address](#) (\$103F) can be used. Specifying the \$103F [address](#) with the BULK command will erase the CONFIG register and the EEPROM array.

EVM command: CHCK

CHCK [<address1>](#) [[<address2>](#)]

Note: prior to entering this command, the user must follow the EEPROM content checking procedure as described in paragraph 3.7.1. of the EVM user's manual.

The CHCK command allows the user to be sure that the MCU internal EEPROM installed in the programming socket is properly erased.

EVM command: COPY

COPY [<address1>](#) [[<address2>](#)] [[<address3>](#)]

Note: prior to entering this command, the user must follow the EEPROM copying procedure as described in paragraph 3.7.4. of the EVM user's manual.

The COPY command allows the user to copy the contents of the programmed MCU internal EEPROM into the EVM user map pseudo EEPROM/ROM, or user supplied memory (expanded/multiplexed mode only).

EVM command: ERASE

ERASE [<address1>](#) [[<address2>](#)]

Note: prior to entering this command, the user must follow the EEPROM erasing procedure as described in paragraph 3.7.2. of the EVM user's manual.

The ERASE command allows the user to erase individual bytes of the programmed MCU internal EEPROM.

EVM command: G

G [[<address>](#)]

The G command you initiate user program execution (free run in real time). The user may optionally specify a starting [<address>](#) where execution is to begin. Execution starts at the current Program Counter

(PC) address location, unless a starting <address> is specified. Program execution continues until a breakpoint is encountered, or the EVM ABORT switch is activated (pressed), or the MASTER RESET switch is pressed.

Before executing a G command, the user must insure that the Stack Pointer (SP) is preset to a valid RAM address.

EVM command: HELP

HELP

The HELP command enables the user available EVM command information to be displayed.

EVM command: LOAD

LOAD [<filename>]

The LOAD command downloads machine code in S-record file format from the PC to the EVM user pseudo ROM. If no filename is provided a dialogbox pops up.

EVM command: MD

MD <address1> [<address2>]

or

DUMP <address1> [<address2>]

Note: it is much easier to use the Target Memory Dump or Target Memory List menu item from the Target Menu (Target Memory submenu).

The MD or DUMP command is used to display a section of user memory beginning at address1 and continuing to address2. If address2 is not entered, 16 bytes are displayed beginning at address1. If address1 is greater than address2, the display will wrap around the 64k user memory map.

EVM command: MM

MM <address>

Note: it is much easier to use the Target Memory Dump or Target Memory List menu item from the Target Menu (Target Memory submenu).

The MM command allows the user to examine/modify contents in user memory at specified locations in an interactive manner. Once entered, the MM command has several submodes of operation that allow modification and verification of data. The following terminators are recognized.

[<data>]<Enter>	Update location and sequence forward.
[<data>]^<Enter>	Update location and sequence backward.
[<data>]=<Enter>	Update location and reopen same location.

[<data>].<Enter>	Update location and terminate.
------------------	--------------------------------

An entry of only "<Enter>" terminates the memory modify interactive operation.

EVM command: NOBR

NOBR [[<address>](#)]...

or

DB [[<address>](#)]...

Note: it is much easier to use the Target Remove or Target Remove All menu item from the Breakpoint menu.

The NOBR or DB command is used to remove one or more breakpoints from the internal breakpoint table. This command functions oppositely of the BR command. After removing the breakpoint, the current breakpoint address, if any, are displayed.

EVM command: P

P [[<count>](#)]

<count> is the number (in hexadecimal, \$FFFF max.) of times the current breakpoint location is to be passed before the breakpoint returns control to the monitor. All other breakpoints are ignored during this command. This command is ideal for applications where registers must be examined after a given number of passes within a software loop.

EVM command: PROG

PROG < [address1](#) > [[<address2>](#)] <data>

Note: prior to entering this command, the user must follow the EEPROM MCU programming procedure as described in paragraphs 3.7 and 3.7.3. of the EVM user's manual must be followed.

The PROG command is used to program the MCU internal EEPROM. Two token PROG command is used when programming data is downloaded from the user pseudo EEPROM. Three token PROG command is used to block fill specific memory locations with an assigned hexadecimal data value. This command is also used to program the MCU configuration register located at \$103F.

EVM command: RD

RD

Note: it is much easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

Contents of the following registers are displayed:

P = Program counter

Y = Y index register

X = X index register
A = A accumulator
B = B accumulator
C = Condition codes
S = Stack pointer

EVM command: RM

RM

Note: it is much easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The RM command is used to modify the MCU registers contents. The RM command takes no parameters and starts by displaying the S (stack pointer) register contents and allowing changes to be made. The order of the registers displayed are:

S (stack pointer)
P (program counter)
Y (index register Y)
X (index register X)
A (accumulator A)
B (accumulator B)
C (condition code)

Once entered, the RM command has several submodes of operation that allow modification and verification of data. The following terminators are recognized.

[<data><Enter>	Update register and sequence forward.
[<data>]^<Enter>	Update register and sequence backward.
[<data>]=<Enter>	Update register and reopen same location.
[<data>].<Enter>	Update register and terminate.

An entry of only "<Enter>" terminates the register modify interactive mode.

EVM command: SPEED

SPEED <baud rate>

Note: this command is useless in THRSim11 because you don't use the host connection (only the terminal connection) to communicate with your EVM target.

The SPEED command only allows the user to reassign or modify the baud rate operating speed of the EVM host I/O port for specific applications.

EVM command: T

T [<count>]

Note: it is much easier to use the Target Step menu item from the Execute menu.

The T command allows the user to monitor program execution on an instruction-by-instruction basis. The user may optionally execute several instructions at a time by entering a <count> value (up to \$FFFF). Execution

starts at the current PC. The PC displayed with the event message is of the next instruction to be executed. During the tracing operation, breakpoints are active and the user program execution stops upon the PC encountering a breakpoint address.

The user should not try to trace an instruction that branches to itself (e.g., BRA). Because the monitor places an SWI instruction on the object of the branch, the instruction would never be executed. However, it would look to the user as if the instruction executed. The user may enter a G command while the PC points to this type of instruction as long as the instruction is not a breakpoint address. The user should also not trace an RTI instruction with the interrupt enabled and pending.

The TS command calculates the value of <count> to fill just one screen full of output.

EVM command: TM

TM [<exit character>]

or

HOST [<exit character>]

Note: this command is useless in THRSim11 because you don't use the host connection (only the terminal connection) to communicate with your EVM target.

The TM or HOST command connects the EVM host port to the terminal port, which allows direct communication between the terminal and the host computer.

EVM command: VERF

VERF <address1> [<address2>] [<address3>]

Note: prior to entering this command, the user must follow the EEPROM MCU verification procedure as described in paragraph 3.7.5. of the EVM user's manual must be followed.

The VERF command allows the user to verify the contents of the programmed MCU internal EEPROM against the EVM user map pseudo EEPROM/ROM, or user supplied memory (expanded multiplexed mode only).

Motorola EVB commands

All EVB commands are supported. See for a complete description the Motorola 68HC11 EVB User's manual.

EVB commands concerning CPU registers:

- Register modify (RM)

Extra commands concerning CPU registers:

- change A register (A)
- change B register (B)
- change CC register (C)
- change D register (D)
- change P register (PC)
- change S register (S)
- change X register (X)
- change Y register (Y)
- Copy CPU registers from simulator to target (RT)

- Copy CPU registers from target to simulator (RS)

EVB commands concerning memory:

- Memory display (MD)
- Memory modify (MM)
- Assemble/disassemble (ASM)
- Block fill (BF)
- Memory move/copy(MOVE)

Extra commands concerning memory:

- Copy memory from target to simulator (MS)
- Copy memory from simulator to target (MT)
- Copy EEPROM memory from simulator to target (ET)

EVB commands concerning breakpoints:

- Set or remove breakpoint (BR)
- Proceed after breakpoint (P)

Extra commands concerning breakpoints:

- Copy PC breakpoints from simulator to target (BT)
- Copy breakpoints from target to simulator (BS)

EVB commands concerning the execution of a program

- Load S-records from file (LOAD)
- Verify S-records from file (VERIFY)
- Trace program (T)
- Execute program (G)
- Execute subroutine (CALL)
- Stop at address (STOPAT)

Extra command concerning the execution of a program

- Trace Screen full (TS)

General EVB command

- Commands quick overview (HELP)

EVB commands concerning EEPROM

- Bulk erase EEPROM (BULK)
- Bulk erase EEPROM and CONFIG register (BULKALL)
- Modify EEPROM mapping(EEMOD)

Miscellaneous EVB commands.

- Transparent mode (TM)
- Send program to another M68HC11 via bootstrap mode (XBOOT)

EVB command: **ASM**

ASM [<address>]

Note: it is much easier to use the editor and assembler build into THRSim11 and download the S records produced by this assembler. If you use the ASM command your program is not saved in a file.

The assembler/disassembler is an interactive assembler/editor. Each source line is converted into the proper machine language code and is stored in memory overwriting previous data on a line-by- line basis at the time of entry. In order to display an instruction, the machine code is disassembled and the instruction mnemonic and operands are displayed. All valid opcodes are converted to assembly language mnemonics. All invalid opcodes are displayed as "ILLOP".

The syntax rules for the assembler are as follows:

- All numerical values are assumed to be hexadecimal. Therefore no base designators (e.g., \$ = hex, % = binary, etc.) are allowed.
- Operands must be separated by one or more space or tab characters.
- Any characters after a valid mnemonic and associated operands are assumed to be comments and are ignored.

Addressing modes are designated as follows:

- Immediate addressing is designated by preceding the address with a # sign.
- Indexed addressing is designated by a comma. The comma must be preceded a one byte relative offset (even if the offset is 00), and the comma must be followed by an X or Y designating which index register to use (e.g., LDAA 00,X).
- Direct and extended addressing is specified by the length of the address operand (1 or 2 digits specifies direct, 3 or 4 digits specifies extended). Extended addressing can be forced by padding the address operand with leading zeros.
- Relative offsets for branch instructions are computed by the assembler. Therefore the valid operand for any branch instruction is the branch-if-true address (or label), not the relative offset.

When a new source line is assembled, the assembler overwrites what was previously in memory. If no new source line is submitted, or if there is an error in the source line, then the contents of memory remain unchanged.

Assembler/disassembler subcommands are as follows.

- / Assemble the current line and then disassemble the same address location.
- ^ Assemble the current line and then disassemble the previous sequential address location.
- Enter Assemble the current line and then disassemble the next opcode address.
- <Ctrl>+J Assemble the current line. If there isn't a new line to assemble, then disassemble the next sequential address location. Otherwise, disassemble the next opcode address.
- <Ctrl>+A or <Esc> Exit the assembler mode of operation.

EVB command: BF

BF <address1> <address2> <data>

or

FILL <address1> <address2> <data>

The BF or FILL command allows the user to repeat a specific pattern throughout a determined user memory range. The <memory from <address1> to <address2> is filled with <data>. If an invalid address is specified, the invalid address and the message "rom-xxxx" is displayed.

EVB command: BR

BR [-] [<address>]...

Note: it is much easier to use the Target Set, Target Remove, Target Remove All or Target List menu item from the Breakpoint menu.

The BR command sets the <address> into the breakpoint address table. The BR -command removes the <address> from the breakpoint address table. During program execution, a halt occurs to the program execution immediately preceding the execution of any instruction address in the breakpoint table. A maximum of four breakpoints may be set. After setting the breakpoint, the current breakpoint addresses, if any, are displayed.

Whenever the G, CALL, or P commands are invoked, the monitor program inserts breakpoints into the user

code at the address specified in the breakpoint table. Breakpoints are accomplished by the placement of a software interrupt (SWI) at each address specified in the breakpoint address table. The SWI service routine saves and displays the internal machine state, then restores the original opcodes at the breakpoint location before returning control back to the monitor program. SWI opcode cannot be executed or breakpointed in user code because the monitor program uses the SWI vector. Only RAM locations can be breakpointed. Branch on self instructions cannot be breakpointed.

Examples:

BR	Display all current breakpoints.
BR < <u>address</u> >	Set breakpoint.
BR < <u>addr1</u> > < <u>addr2</u> > ...	Set several breakpoints.
BR -	Remove all breakpoints.
BR -< <u>addr1</u> > <>...	Remove < <u>addr1</u> > and add < <u>addr2</u> >.
BR < <u>addr1</u> > - < <u>addr2</u> >	Add < <u>addr1</u> >, clear all entries, then add < <u>addr2</u> >.
BR < <u>addr1</u> > -< <u>addr2</u> >	Add < <u>addr1</u> >, then remove < <u>addr2</u> >.

EVB command: BULK

BULK
or
ERASE

The BULK or ERASE command allows the user to erase all MCU EEPROM locations (\$B600-\$B7FF).

EVB command: BULKALL

BULKALL

The BULKALL command allows the user to erase all MCU EEPROM locations (\$B600-\$B7FF) including the configuration (CONFIG) register location (\$103F).

EVB command: CALL

CALL [<address>]

The CALL command allows the user to execute a user program subroutine. Execution starts at the current program counter (PC) address location, unless a starting <address> is specified. The first unmatched return from subroutine (RTS) encountered will return control back to the monitor program. Thus any user program subroutine can be called and executed via the monitor program. Program execution continues until a breakpoint is encountered, or the EVB reset switch is activated (pressed).

EVB command: EEMOD

EEMOD [<address1>] [<address2>]

The EEMOD command is only used when the EVB resident MCU is changed from XC68HC11A1FN

device to an XC68HC11A2FN, XC68HC11E2FN, or C68HC811E2FN device. The EEMOD command informs the monitor where the EEPROM resident MCU is mapped.

If the starting address is specified and the ending address is not specified, the re-mapped address location will end at <address1> + 2K bytes.

EVB command: G

G [<address>]

The G command allows the user to initiate user program execution (free run in real time). The user may optionally specify a starting <address> where execution is to begin. Execution starts at the current program counter (PC) address location, unless a starting <address> is specified. Program execution continues until a breakpoint is encountered, or the EVB reset switch is activated (pressed).

EVB command: HELP

HELP

The HELP command enables the user available EVB command information to be displayed.

EVB command: LOAD

LOAD [<filename>]

The LOAD command downloads machine code in S-record file format from the PC to the EVM user pseudo ROM. If no filename is provided a dialogbox pops up.

EVB command: MD

MD <address1> [<address2>]

or

DUMP <address1> [<address2>]

Note: it is much easier to use the Target Memory Dump or Target Memory List menu item from the Target Menu (Target Memory submenu).

The MD or DUMP command allows the user to display a block of user memory beginning at <address1> and continuing to <address2>. If <address2> is not entered, 9 lines of 16 bytes are displayed beginning at <address1>. If <address1> is greater than <address2>, the display will default to the first address. If no addresses are specified, 9 lines of 16 bytes are displayed near the last memory location accessed.

EVB command: MM

MM [<address>]

or
MEMORY [[<address>](#)]

Note: it is much easier to use the Target Memory Dump or Target Memory List menu item from the Target Menu (Target Memory submenu).

The MM or MEMORY command allows the user to examine/modify contents in user memory at specified locations in an interactive manner. The MM command will also erase any EEPROM location, and will reprogram the location with the corresponding value (EEPROM locations treated as if RAM). Once entered, the MM command has several sub-modes of operation that allow modification and verification of data. The following subcommands are recognized.

[<data>]<Enter>	Update location and sequence forward.
[<data>]^<Enter>	Update location and sequence backward.
[<data>]/<Enter>	Update location and reopen same location.
<Ctrl>+J	Open next location.
<Esc> or <Ctrl>+A	Terminate the memory modify interactive operation.

EVB command: **MOVE**

MOVE [<address1>](#) [<address2>](#) [[<address3>](#)]

or

COPY [<address1>](#) [<address2>](#) [[<address3>](#)]

or

READ [<address1>](#) [<address2>](#) [[<address3>](#)]

The MOVE, COPY, or READ command allows the user to copy/move memory to new memory location. If the destination [<address3>](#) is not specified, the block of data residing from [<address1>](#) to [<address2>](#) will be moved up one byte. Using the MOVE command on EEPROM locations will program EEPROM cells. The MOVE command is useful when programming EEPROM. As an example, a program is created in user RAM using the assembler, debugged using the monitor, and then programmed into EEPROM with the MOVE command.

EVB command: **P**

P

This command is used to proceed or continue program execution without having to remove assigned breakpoints. This command is used to bypass assigned breakpoints in a program executed by the G command.

EVB command: **RM**

RM [p|y|x|a|b|c|s]

or

RD [p|y|x|a|b|c|s]

or

REGISTER [p|y|x|a|b|c|s]

Note: it is much easier to use the Target CPU Register menu item from the Target menu (Target Registers submenu) or the statusbar to inspect or change the target CPU registers.

The RM command is used to modify the MCU registers contents. The RM command takes one parameter and starts by displaying the specified register contents and allowing changes to be made. The order of the registers displayed are:

P (program counter)
Y (index register Y)
X (index register X)
A (accumulator A)
B (accumulator B)
C (condition code)
S (stack pointer)

Once entered, the RM command has several submodes of operation that allow modification and verification of data. The following terminators are recognized.

[<data>]<Enter>	Update register and sequence forward.
<Enter>	Open next register.
<Esc> or <Ctrl>+A	Terminate the register modify interactive operation.

EVB command: STOPAT

STOPAT <address>

Note: This command is not supported in EVB version 2.5.

Note: it is much easier to use the Target Go Until menu item from the Execute menu or the popup menu from the list window.

The STOPAT command causes a user program to be executed one instruction at time until the specified <address> is encountered. Execution begins with the current user PC address and stop just before execution of the instruction at the specified stop address. The STOPAT command should only be used when the current value of the user PC register is known. For example after a breakpoint is reached or after an RD command is used to set the user PC. The STOPAT command has an advantage over breakpoints in that a stop address can be a ROM location while breakpoints only operate in RAM or EEPROM locations. Since the STOPAT command traces one instruction at a time with a hidden return to the monitor after each user instruction, some user programs will appear to execute slowly. The stop address specified in the STOPAT command must be the address of an opcode just as breakpoints can only be set at opcode addresses.

EVB command: T

T [<count>]

Note: it is much easier to use the Target Step menu item from the Execute menu.

The T command allows the user to monitor program execution on an instruction-by-instruction basis. The user may optionally execute several instructions at a time by entering a count value (up to \$FF). Execution starts at the current program counter (PC). The PC displayed with the event message is of the next instruction to be executed. The trace command operates by setting the OC5 interrupt to time out after the first cycle of the first opcode fetched.

The TS command calculates the value of <count> to fill just one screen full of output.

EVB command: TM

TM
or
HOST

Note: this command is useless in THRSim11 because you don't use the host connection (only the terminal connection) to communicate with your EVB target.

The TM command connects the EVB host port to the terminal port, which allows direct communication between the terminal and the host computer. All I/O between the ports are ignored by the EVB until the exit character is entered from the terminal. Enter <Ctrl>+A to exit from transparent mode.

EVB command: VERIFY

VERIFY [[filename](#)]

The VERIFY command is similar to the LOAD command except that the VERIFY command instructs the EVB to compare the downloaded S-record data to the data stored in memory.

EVB command: XBOOT

XBOOT [[address1](#)] [[address2](#)]

The XBOOT command loads/transfers a block of data from [address1](#) through [address2](#) via the serial communications interface (SCI) to another MC68HC11 MCU device which has been reset in the bootstrap mode. A leading control character of \$FF is sent prior to sending the data block. This control character is part of the bootstrap mode protocol and establishes the baud rate for the rest of the transfer. If only one [address](#) is provided, the [address](#) will be used as the starting [address](#) and the block size will default to 256 bytes. If no addresses are provided, the block of addresses from \$C000 through \$C0FF is assumed by the BUFFALO monitor program.

The XBOOT command generates SCI transmitter output signals at 7812.5 baud which are intended for another MC68HC11 MCU device operating in the bootstrap mode. These signals appear as nonsense data to the terminal display used for normal communication with the EVB. After using the XBOOT command the EVB must be reset by pressing the reset switch S1 before normal communications can resume.

Debugging C/C++ programs

THRSim11 can be used to debug C/C++ programs which are compiled with the GNU Development Chain for 68HC11.

[Help about this topic is available here \(in HTML\).](#)

[An example is available here \(in HTML\).](#)

[Help for the GNU gcc compiler and other GNU tools can be found here \(in HTML\).](#)

Help for the GNU make utility can be found on the Internet: http://www.gnu.org/software/make/manual/html_chapter/make.html.

Debugging GNU as assembler programs

THRSim11 can be used to debug GNU as assembler programs which are assembled with the GNU Development Chain for 68HC11.

[Help about this topic is available here \(in HTML\).](#)

[An example is available here \(in HTML\).](#)

[Help for the GNU as assembler and other GNU tools can be found here \(in HTML\).](#)

Help for the GNU make utility can be found on the Internet: http://www.gnu.org/software/make/manual/html_chapter/make.html.

External Editor.

You can use an external editor in combination with THRSim11. If you installed THRSim11 with C support the external editor SciTE will be installed on default. This editor will be used on default to edit .C, .CPP, .H, .S, .LD, and makefile files. All other files will be automatically opened with the internal editor. If you want to change this default behaviour or if you want to use an other editor you have to change the [editor options](#) in the `THRSim11_options.txt` file. [More information about the external editor can be found here \(in HTML\).](#)

Menu: **V**iew, High Level Language Variables.

[Help about this topic is available here \(in HTML\).](#)

Menu: **T**arget, High Level Language Variables.

[Help about this topic is available here \(in HTML\).](#)

C/C++ List Window.

[Help about this topic is available here \(in HTML\).](#)

C/C++ Variables Window.

[Help about this topic is available here \(in HTML\).](#)

Target C/C++ List Window.

[Help about this topic is available here \(in HTML\).](#)

Target C/C++ Variables Window.

[Help about this topic is available here \(in HTML\).](#)
