AVR Simulator Manual

Introduction

Welcome to the ATMEL *AVR* Simulator. This manual describes the usage of the Simulator. The Simulator covers the whole range of microcontrollers in the AT90S family.

The Simulator executes object code generated for the AT90S microcontrollers. In addition to being an instruction set Simulator, it supports simulation of various I/O functions.

The Simulator can be controlled through a Command window, through Menus, or by clicking on Toolbar buttons.

The Simulator reads a self defined object file format which is described in this document. This format is supported by the ATMEL *AVR* Assembler. The object format allows assembly source level simulation.

The Simulator runs under Microsoft Windows 3.11, Windows95 and Windows NT. Most of this document is available through the on-line help function.

To get quickly started, the Quick-Start Tutorial is an easy way to get familiar with the ATMEL *AVR* Simulator.

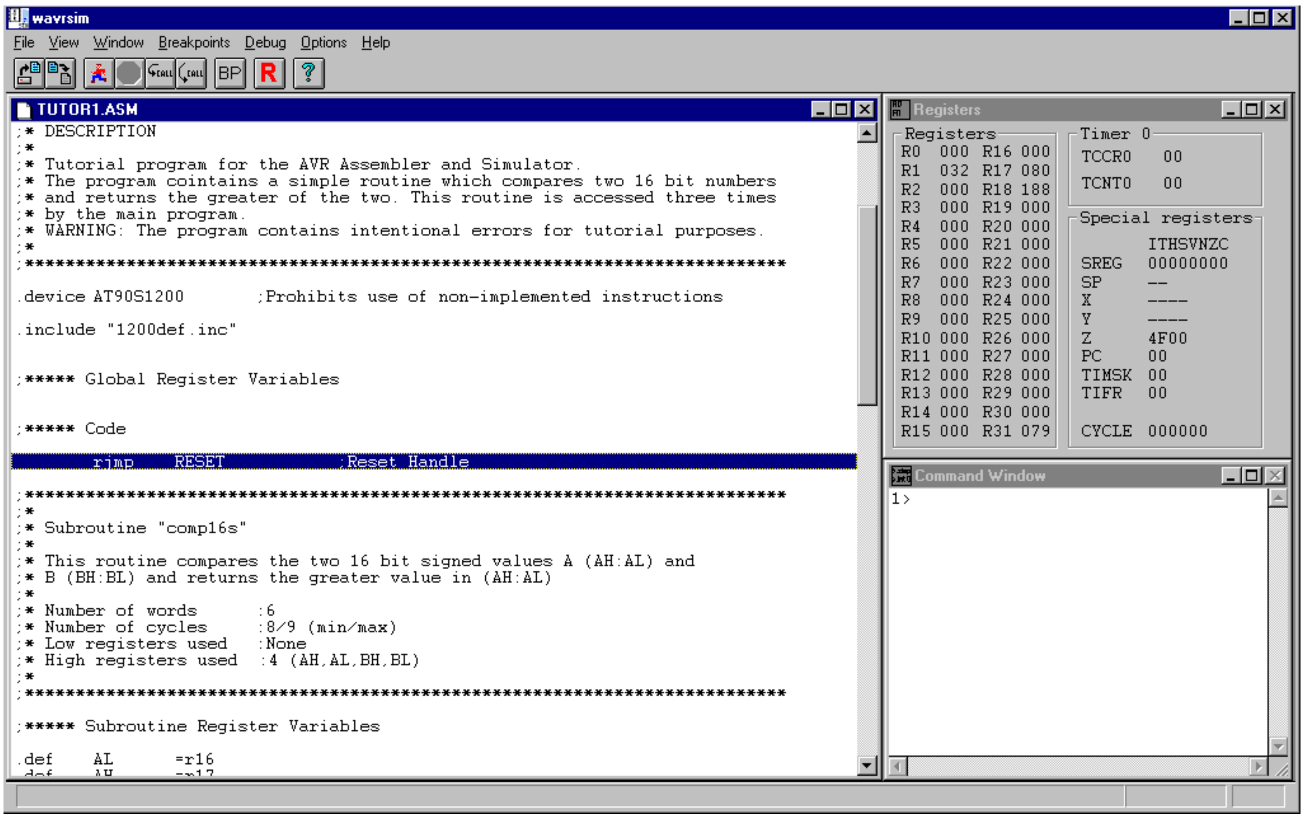
8-Bit AVR Simulator Manual

Simulator Quick-Start Tutorial

This tutorial assumes that the *AVR* Assembler and all program files that come with it are properly installed on your computer. In addition, the user must have completed the Assembler Tutorial. Please refer to the Installation Instructions and the Assembler Tutorial.

Getting Started

Start the *AVR* Simulator. By selecting “File>>Open” or by clicking , open the file “tutor1.obj” which was generated when running the Assembler Tutorial. Select “View>>Register” from the top Menu. Since this program is written for the AT90S1200, the Simulator must be set up to simulate this device. Select “Options>>AVR options” and tick AT90S1200 in the appearing dialog box. Then resize your windows until the screen looks something like the following figure:



The window setup will be restored the next time you start the Simulator.

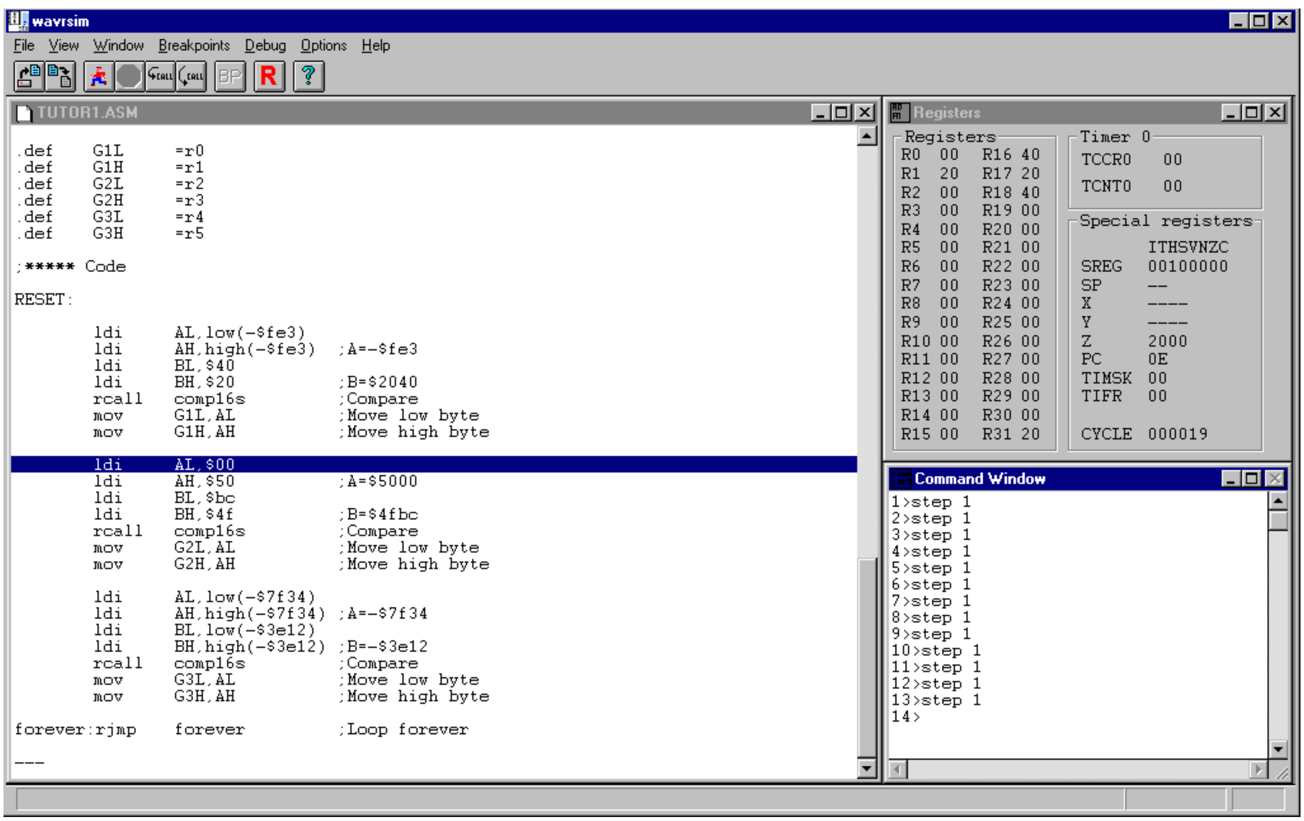
For a program like this, it might be useful to view the registers as hexadecimal values instead of decimal. Select “Options>>Simulator Options” and deselect “Show registers in decimal” in the dialog box. Observe that all register values changes from “000” to “00”.

Simulating

With the Simulator set up and ready to go, it is time for some action. There are several ways to run the program, however, when running a program for the first time, it is often useful to run it line by line. This can be done by using two different step/trace features:

- Trace Into: The program is executed one line at a time. When running a subroutine call, the Simulator jumps to the subroutine and runs it line by line. The Trace Into function is executed by clicking , selecting "Debug>>Trace Into", or by pressing F7.
- Step Over: The program is executed one line at a time. When running a subroutine call, the Simulator executes the subroutine at full speed, and halts at the instruction following the subroutine call. The Step Over function is executed by clicking , selecting "Debug>>Step Over", or by pressing F8.

Now run the program by pressing F7 until the cursor in the Code window is positioned as shown in the following figure:

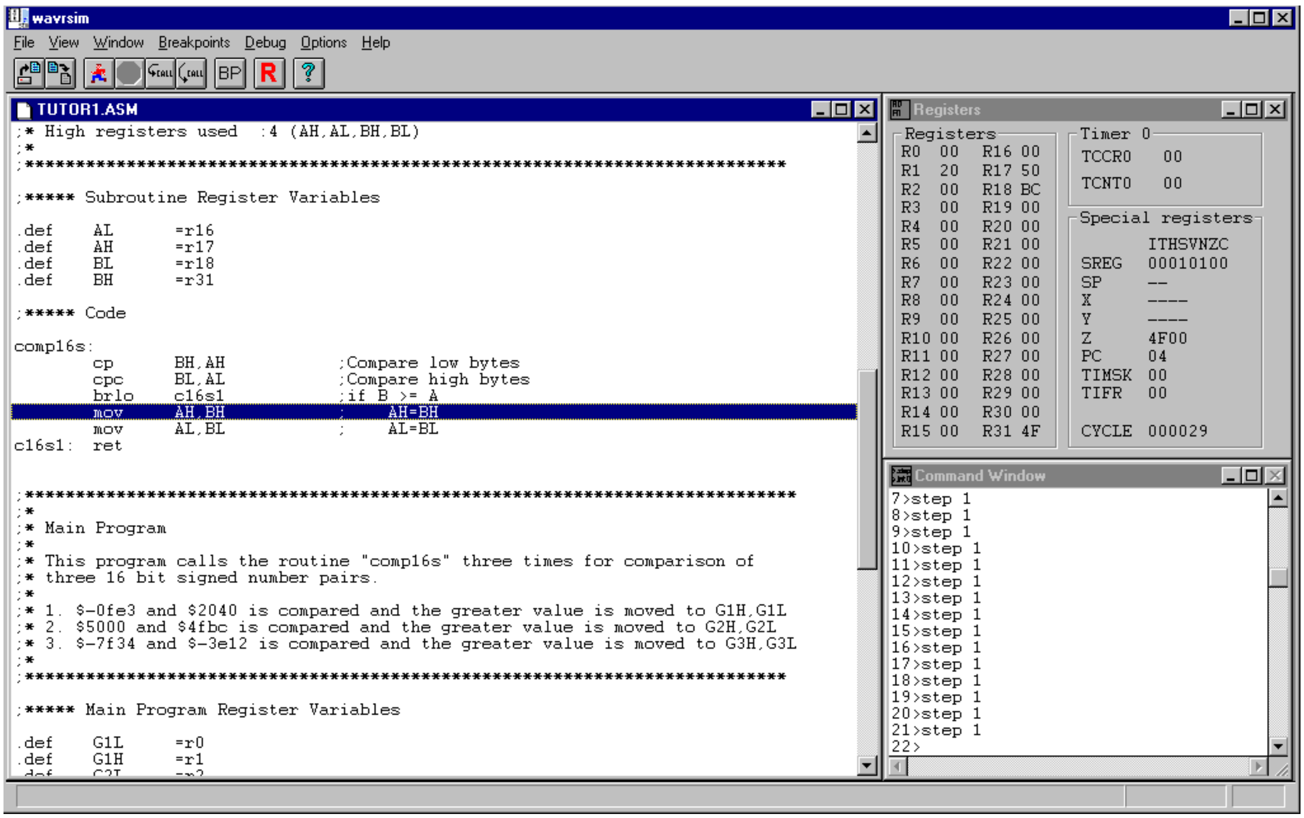


Observe how registers are loaded and moved while stepping through the code.

Next, run the program by pressing F8 until the program counter (PC) in the Register window shows 15. Observe the difference in running the program this way, as the subroutine call is executed, but stepped over by the cursor.

Usually, during debugging of a program, specific parts of the code are constantly revised while other remains the same. To get quickly to the interesting parts while skipping code which is known to work, the Goto Cursor-function is useful. Use the scroll bars in the Code window to scroll downwards until the subroutine labeled “comp16s” becomes visible. Position the cursor at the first line by clicking on it. Then press F4 to go to this line. The Simulator runs at full speed until the first line of the subroutine is reached, whereupon it halts and waits for further actions from the user. Single step through the rest of the code until the eternal loop at the end is reached.

Even more advanced debugging is achieved by the use of Breakpoints. Breakpoints are user defined locations in the code where the program halts when running at full speed. Scroll the program to show the subroutine and position the cursor at the line “`mov AH,BH`” as shown in the following figure:



Press F5 (or click BP) to insert a breakpoint here. F5 toggles breakpoints, i.e. inserts a breakpoint if there is none on the line, or removes an existing breakpoint. Observe that the presence of a breakpoint is indicated by displaying the code line in red.

Reset the program by clicking R on the Toolbar, and start program execution by clicking . Observe that the program runs and halts at the breakpoint after 12 cycles. This means that it is the first time the subroutine is called. Run the program once more and observe that the program halts after 47 cycles this time. This is during the third and last time the routine is executed, because in the second call variable A was greater than B, hence the code line containing the breakpoint was never executed. Click  once more and observe that the program runs forever because the breakpoint is never reached.

Modifying the Program

Suppose that debugging revealed that the comparison of the two 16-bit variables in the subroutine should have been inverted. This part of the tutorial shows how to modify the code, re-assemble and re-simulate the program.

Start the Assembler, or switch to it if already open (Windows' ALT-TAB is useful here). Modify line 61 to "`brge c16s1 ;if B < A`". Assemble the new file, and make sure that the message "Assembly complete with no errors" appears before switching back to the Simulator.
Once in the Simulator, click  to reload the file. Step through the program and verify that the comparison now is inverted.

The Simulator Menu

Like most MS-Windows programs WAVRSIM have a “drop-down” Menu a the top of the main window. From this Menu you can set program options and execute Simulator commands. The “Window” and “Help” menu items are the standard MS-Windows Menus and are not discussed here. All the commands can be executed while a program is simulated.

The View Menu

From this Menu it can be selected which windows to display in addition to the command window (which is always displayed) and the source window. The following windows can be selected:

- Register Window: This window displays the contents of all the internal registers, some of the I/O registers, the Program Counter and the Cycle Counter. It is possible to change the radix between decimal and hexadecimal (see the description of the Options Menu item). Keyboard shortcut: ALT+V R.
- SRAM Window: This window displays the contents of the SRAM. The size of this SRAM is given in the “Options>>AVR Options” Menu. Keyboard shortcut: ALT+V S.
- EEPROM Window: This window displays the contents of the EEPROM memory. The size of this EEPROM is given in the “Options>>AVR Options” Menu. Keyboard shortcut: ALT+V E.

The Breakpoints Menu

The Simulator supports setting and clearing of individual breakpoints, clearing all breakpoints and listing all breakpoints. The following commands are available through the Breakpoints Menu:

- Toggle breakpoint. This command should be used to set or clear a breakpoint directly in the source. Usage: Press the left mouse button on the source line where you want the breakpoint set and select this command. Use the same procedure to turn the breakpoint off. Keyboard shortcut: F5. Using this command issues a BS or a BC command in the command window depending on whether the breakpoint was already set or cleared. This command is also available from the Toolbar.
- Clear all breakpoints. This command clears all breakpoints. Keyboard shortcut: Ctrl+F5. Using this command issues a BD command in the command window.
- List all breakpoints. This command lists the Program memory address of all defined breakpoints in the command window. Keyboard shortcut: Shift+F5. Using this command issues a BL command in the command window.

The Debug Menu

The following commands are available through the Debug Menu:

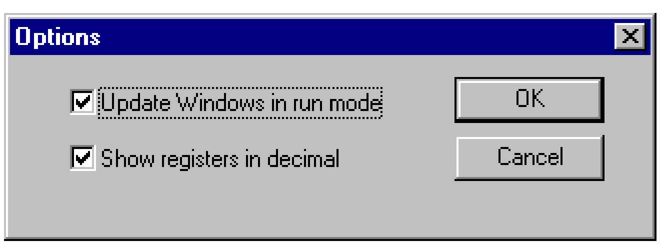
- Run. Start execution of the program from the current position of the program counter. Keyboard shortcut: Ctrl+F9. Using this command issues a GO command in the command line window. This command is also available from the Toolbar.
- Step into: Execute one instruction. Keyboard shortcut: F7. Using this command issues a STEP 1 command in the command window. This command is also available from the Toolbar.
- Step over: Much like step into except when you come to a subroutine call, all the instructions in the subroutine are executed, and the cursor is placed on the instruction following the subroutine call. Keyboard shortcut: F8. Using this command issues a STEP 1 or GOTO command in the command window. This command is also available from the Toolbar.
- Goto cursor: Runs the program until the location corresponding to the source line with the vertical bar is encountered. Usage: press the left mouse button on the line you want to step to and then select this command. Keyboard shortcut F4. Using this command issues a GOTO command in the command window.
- Reset: This command resets the Simulator. Keyboard shortcut: Ctrl+F2. Using this command issues a RESET command in the command window. This command is also available from the Toolbar.

The Options Menu

There are two choices in the Options Menu - the Simulator Options and the AVR options.

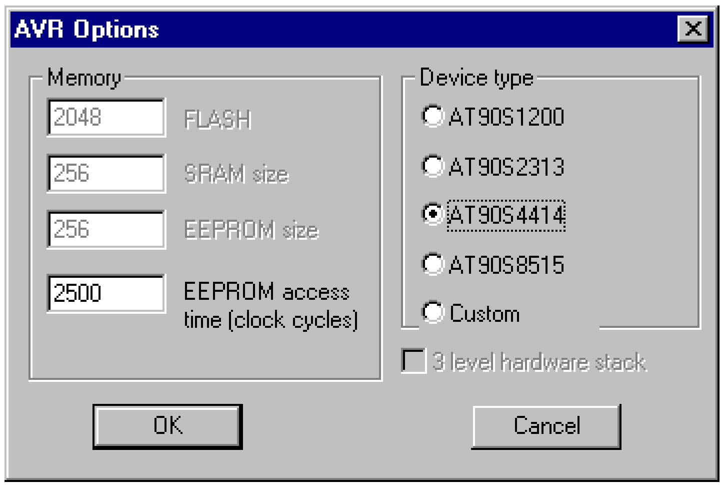
Simulator options

When the Simulator options is selected, a dialog like the one presented below appears on the screen. The “Update Windows in run mode” check box causes the contents of all windows to be updated after every instruction. This causes the program to run slower (depends on the number of displayed windows) . The “Show registers in decimal” checkbox causes all the registers to be displayed in decimal when checked and hexadecimal when not checked. These options are remembered in subsequent runs of the Simulator.



AVR options

When the *AVR* options is selected, a dialog like the one in next figure appears on the screen. The ROM, RAM and EEPROM sizes tells the Simulator how much program memory, data memory and EEPROM memory to support. In addition, the Simulator supports a 4 level hardware stack available in some base line microcontrollers. These options are remembered in subsequent runs of the Simulator.



If a predefined device is selected, ROM size, RAM size, EEPROM size and Hardware stack is set as specified for this device. In order to set the Memory sizes freely, first click the Custom Device type, and then change the ROM size, RAM size, EEPROM size and whether or not the Custom device should have a Hardware stack.

Writing to EEPROM takes a certain time. This is a self time feature where the Hardware clears a specified bit in the I/O area when the EEPROM write is completed (see the *AVR* Data Book for more information on this issue). The number of clock cycles needed to complete an EEPROM write depends on the external clock used in the system, hence the user can select how many clock cycles it takes in order to complete the EEPROM write operation.

The Toolbar

The most commonly used functions are placed on the Toolbar on top of the screen. Being placed on the Toolbar, the user does not have to go through the Menus, but only has to click one button always present on the Toolbar.

The Toolbar contains the following buttons:

Open object file. Pressing this button opens a file select dialog for selecting the object file to simulate

Reopen object file. Pressing this button reloads the object file already loaded. This is a useful feature when a file needs to be reassembled.

Run. Pressing this button makes the Simulator start/resume simulation of the program.

Stop. Pressing this button makes the Simulator stop executing a running program.

Trace into. Pressing this button makes the Simulator execute one instruction and then stop. If the instruction is a subroutine call, the Simulator will stop on the first instruction within the subroutine.

Step over. Pressing this button makes the Simulator execute one instruction and then stop. If the instruction to execute is a subroutine call, the Simulator will execute the whole subroutine, and stop on the instruction following the subroutine call instruction.

Toggle breakpoint. Pressing this button will set a breakpoint on the line where the cursor is. If there was already a breakpoint on this address, the breakpoint will be removed instead.

Reset. Pressing this button will make the Simulator reset.

Help. Pressing this button will invoke the Help system.

Commands

The Simulator has a command window in which all the different commands can be given. As described, the Simulator also has Menus and a Toolbar for issuing commands. Issuing commands from the Menus or the Toolbar will automatically insert the corresponding command in the command window.

An overview of all commands recognized by the Simulator is given in the following table:

Command	Description
BC	Clear breakpoint
BD	Delete all breakpoints
BL	List all breakpoints
BS	Set breakpoint
ENDINS	Close instruction log file
ENDREG	Close register log file
FILE	Run from file
GO	Run program
GOTO	Run program to a specified location
INSLOG	Log instructions on file
LOAD	Load an Object file to simulate
PC	Set program counter
RAM	Change SRAM
RAMDUMP	Dump SRAM contents on file
REG	Set register
REGLOG	Log registers on file
RESET	Reset the Simulator
SAVE	Save session
STEP	Step through the code
STOP	Stop the Simulator

BC - Clear Breakpoint

The BC command clears a breakpoint. The Simulator can have 256 breakpoints which are numbered from 0 to 255. The breakpoint to be cleared must be numbered.

Clearing a breakpoint which is not set does not have any effect. To clear all defined breakpoints, use the BD command

Syntax:

```
BC breakpointnumber
```

Example:

```
1>load “\user\asm\addsub.asm”      ; Load a file
2>bs 0,13                          ; Set breakpoint 0 at PC=13
3>bl                               ; List all breakpoints
Breakpoint 0 = 0000d
4>bc 0                             ; Clear breakpoint 0
```

BD - Delete all Breakpoints

The BD command clears all breakpoints defined. In order to clear individual breakpoints, use the BC command.

Syntax:

```
BD
```

Example:

```
1>load “\user\asm\addsub.obj”      ; Load a file
2>bs 0,13                          ; Set breakpoint 0 at PC=13
3>bs 1,18                          ; Set breakpoint 1 at PC=18
4>bl                               ; List all breakpoints
Breakpoint 0 = 0000d
Breakpoint 1 = 00012
5>bd                               ; Clear all breakpoints
6>bl                               ; List all breakpoints

No breakpoints defined
```

BL - List all Breakpoints

The BL command lists all the breakpoints which have been set. One output line is produced for each defined breakpoint, where the breakpoint number and the address it is set on are listed. If no breakpoints are defined, a message indicating this is issued.

Syntax:

```
BL
```

Example:

```
1>load "\user\asm\addsub.obj"      ; Load a file
2>bs 0,13                          ; Set breakpoint 0 at PC=13
3>bs 1,18                          ; Set breakpoint 1 at PC=18
4>bl                               ; List all breakpoints
Breakpoint 0 = 0000d
Breakpoint 1 = 00012
5>bd                               ; Clear all breakpoints
6>bl                               ; List all breakpoints

No breakpoints defined
```

BS - Set Breakpoint

The BS command sets a breakpoint on a given location in program memory. The Simulator can have 256 breakpoints which are numbered from 0 to 255. The breakpoint to be set must be numbered, and the program memory address must be known.

Setting a breakpoint which is in use overwrites this previously set breakpoint.

Syntax:

BS *breakpointnumber,address*

Example:

```
1>load "\user\asm\addsub.obj"      ; Load a file
2>bs 0,13                          ; Set breakpoint 0 at PC=13
3>bl                               ; List all breakpoints
Breakpoint 0 = 0000d
4>bc 0                             ; Clear breakpoint 0
```

ENDINS - Close the Instruction log file

The ENDINS command closes the Instruction log file and stops logging of the instructions that are executed. The Simulator has the ability to log all instructions that are executed to a file. To start logging, an INSLOG command must be issued, and the ENDINS command stops the logging.

The Instruction log contains a header, and one line for each cycle elapsed during the logging. These lines contains the cycle counter, an 8 bits interrupt vector indicating whether an interrupt request is pending, interrupt acknowledge, the program counter and the disassembled instruction. In some cases, the instruction field will be empty. This is the case when an instruction needs more than one cycle to complete. The instruction is placed on the first cycle of its execution.

Issuing an ENDINS command when no logging is done does not give any warnings or errors.

Syntax:

```
ENDINS
```

Example:

```
1>load “\user\asm\addsub.obj”      ; Load a file
2>step 417                         ; Execute some instr.
3>inslog “addsub.log”              ; Start logging
4>step 12                          ; Execute some instr.
5>endins                           ; End logging
```

Example of Instruction log file contents:

```
Instruction log file. Generated by wavrsim.
Format: cycle: intvec intack address instruction.
000001a2: 00000000 0 000008 st       Z+,r16
000001a3: 00000000 0 000009
000001a4: 00000000 0 000009 dec      r16
000001a5: 00000000 0 00000a cpse     r16,r17
000001a6: 00000000 0 00000b rjmp     -3
000001a7: 00000000 0 00000c
000001a8: 00000000 0 000008 st       Z+,r16
000001a9: 00000000 0 000009
000001aa: 00000000 0 000009 dec      r16
000001ab: 00000000 0 00000a cpse     r16,r17
000001ac: 00000000 0 00000b rjmp     -3
000001ad: 00000000 0 00000c
000001ae: 00000000 0 000008 st       Z+,r16
000001af: 00000000 0 000009
000001b0: 00000000 0 000009 dec      r16
000001b1: 00000000 0 00000a cpse     r16,r17
000001b2: 00000000 0 00000b rjmp     -3
000001b3: 00000000 0 00000c
```

ENDREG - Close the Register log file

The ENDREG command closes the Register log file and stops logging of the register contents. The Simulator has the ability to log the contents of all registers plus the status register for each cycle to a file. To start logging, a REGLOG command must be issued, and the ENDREG command stops the logging.

The Register log contains a header, and one line for each cycle elapsed during the logging. These lines contains the cycle counter, the status register bit by bit, and the contents of all the 32 register in hexadecimal format.

Issuing an ENDREG command when no logging is done does not give any warnings or errors.

Syntax:

```
ENDREG
```

Example:

```
1>load “\user\asm\addsub.obj”      ; Load a file
2>step 385                         ; Execute som instr.
3>reglog “addsub.log”              ; Start logging
4>step 12                          ; Execute some instr.
5>endreg                           ; End logging
```

Example of Register log file contents:

```
Register log file. Generated by wavrsim.
   cycle: ITOSVNZC 31302928272625242322212019181716 151413121110 9 8 7 6 5 4 3 2 1 0
00000241: 01000000 010101003f000000000000000202003f 00000000000000000000000000383938
00000242: 01000000 010101003f000000000000000202003f 00000000000000000000000000383938
00000243: 01000000 010101003f000000000000000202003f 00000000000000000000000000383938
00000244: 01000000 010201003f000000000000000202003f 00000000000000000000000000383938
00000245: 01000000 010201003f000000000000000202003f 00000000000000000000000000383938
00000246: 01000000 010201003f000000000000000202003e 00000000000000000000000000383938
00000247: 01000000 010201003f000000000000000202003e 00000000000000000000000000383938
00000248: 01000000 010201003f000000000000000202003e 00000000000000000000000000383938
00000249: 01000000 010201003f000000000000000202003e 00000000000000000000000000383938
0000024a: 01000000 010301003f000000000000000202003e 00000000000000000000000000383938
0000024b: 01000000 010301003f000000000000000202003e 00000000000000000000000000383938
0000024c: 01000000 010301003f000000000000000202003d 00000000000000000000000000383938
0000024d: 01000000 010301003f000000000000000202003d 00000000000000000000000000383938
0000024e: 01000000 010301003f000000000000000202003d 00000000000000000000000000383938
0000024f: 01000000 010301003f000000000000000202003d 00000000000000000000000000383938
00000250: 01000000 010401003f000000000000000202003d 00000000000000000000000000383938
00000251: 01000000 010401003f000000000000000202003d 00000000000000000000000000383938
00000252: 01000000 010401003f000000000000000202003c 00000000000000000000000000383938
```

FILE - Run from file

The FILE command enables batch execution of commands. If there is a sequence of commands that are given, these commands can be written in a file, and can be executed from this file by issuing a FILE command. The commands used in a session are logged and can be saved to file by using the SAVE command. To repeat a simulation session, the file saved with the SAVE command can be executed with the FILE command.

Syntax:

```
FILE "filename"
```

Example:

```
FILE “IOPROG.CMD”                  ; Execute command file
```

GO - Run program

The GO command starts execution of the program loaded into the Simulator. If the program has already been executed and then stopped, the GO command resumes execution from where it was stopped. When a GO command is issued, the Simulator runs until a STOP command is issued, or a breakpoint is encountered.

Issuing a GO command to a Simulator already running will produce an error message, but will not stop the Simulator. When the Simulator is running, the text “RUNNING” appears in the bottom line of the Simulator window.

Syntax:

```
GO
```

Example:

```
1>load “addsub.obj”; Load a file
2>go                               ; Execute loaded code
```

GOTO - Run program to a specified location

The GOTO command starts execution of the program loaded into the Simulator and executes until the program counter equals the specified location. If the program have already been ran and stopped, the GOTO command resumes execution from where it was stopped. If a breakpoint is encountered before the program counter equals the specified location, the Simulator will stop.

Issuing a GOTO command to a Simulator which is already running towards an unreached GOTO destination will not alter the first destination, hence, the command is ignored.

Syntax:

```
GOTO progadr
```

Example:

```
1>load “addsub.obj”; Load a file
2>goto 100                      ; Execute until PC=100
```

INSLOG - Log instructions on file

The INSLOG command starts Instruction logging to file. The Simulator has the ability to log all instructions that are executed to a file. To start logging, the INSLOG command must be issued. The ENDINS command stops the logging.

The Instruction log contains a header, and one line for each cycle elapsed during the logging. These lines contains the cycle counter, an 8 bits interrupt vector indicating whether an interrupt request is pending, interrupt acknowledge, the program counter and the disassembled instruction. In some cases, the instruction field will be empty. This is the case when an instruction needs more than one cycle to complete. The instruction is placed on the first cycle of its execution.

Issuing an INSLOG command when logging is already enabled discards the currently active log file and starts logging in the new log file. If the file already existed, it is overwritten.

Syntax:

```
INSLOG “filename”
```

Example:

```
1>load “\user\asm\addsub.obj”     ; Load a file
2>step 417                      ; Execute some instr.
3>inslog “addsub.log”           ; Start logging
4>step 12                       ; Execute some instr.
5>endins                        ; End logging
```

Example of Instruction log file contents:

```
Instruction log file. Generated by wavrsim.
Format: cycle: intvec intack address instruction.
000001a2: 00000000 0 000008 st       Z+,r16
000001a3: 00000000 0 000009
000001a4: 00000000 0 000009 dec      r16
000001a5: 00000000 0 00000a cpse     r16,r17
000001a6: 00000000 0 00000b rjmp     -3
000001a7: 00000000 0 00000c
000001a8: 00000000 0 000008 st       Z+,r16
000001a9: 00000000 0 000009
000001aa: 00000000 0 000009 dec      r16
000001ab: 00000000 0 00000a cpse     r16,r17
000001ac: 00000000 0 00000b rjmp     -3
000001ad: 00000000 0 00000c
000001ae: 00000000 0 000008 st       Z+,r16
000001af: 00000000 0 000009
000001b0: 00000000 0 000009 dec      r16
000001b1: 00000000 0 00000a cpse     r16,r17
000001b2: 00000000 0 00000b rjmp     -3
000001b3: 00000000 0 00000c
```

LOAD - Load an Object file to simulate

The LOAD command loads an object file generated by for instance the ATMEL *AVR* Assembler. If the object file is not in a recognizable format, the Simulator stops loading of the file and produces an error message.

When a LOAD command is issued, the Simulator is reset. A new window containing the source file appears. If the object file consists of several source files, the first source file will appear.

The object file format is described later in this document.

Syntax:

```
LOAD "objfile"
```

Example:

```
LOAD “test.obj”
```

PC - Set program counter

The PC command sets the value of the program counter. This command can also be used when the Simulator is running.

Syntax:

PC *progadr*

Example:

```
1>load “\user\asm\pctest.obj”      ; Load a file
2>pc 20                            ; Set PC=20 (decimal)
3>go                               ; Start execution
```

RAM - Change SRAM

The RAM command enables the user to put a specified value into an SRAM location. The RAM command accesses one byte of memory at a time. The address must be given as a constant. The size of the memory is given in the Options Menu. The addresses 0-31 are reserved for the registers. The RAM command can not be used for setting values in the registers.

Syntax:

RAM *addr,val*

Example:

```
1>load “\user\asm\iotest.obj”      ; Load a file
2>ram 32,11                        ; Put 11 into SRAM(32)
3>go                               ; Start execution
```

RAMDUMP - Dump SRAM contents to file

The RAMDUMP command dumps the contents of the SRAM to a user specified file. The RAMDUMP can also be given when the Simulator is running. The size of the SRAM is given in the Options Menu, and the complete SRAM is dumped each time this command is issued.

The files containing the SRAM contents does not have any header. They contain one line for each SRAM location and has the following format: `address:contents`. The address is 2 bytes and the contents is one byte, both represented in hexadecimal form.

Syntax:

RAMDUMP “filename”

Example:

```
1>load “\user\asm\iotest.obj”      ; Load a file
2>ram 32,11                        ; Put 11 into SRAM(32)
3>ramdump “dump.ram”               ; Dump SRAM to file
```

Example on file:

```
0020:01
0021:00
0022:a7
0023:1a
0024:03
0025:00
0026:9c
0027:33
0028:17
0029:00
002a:36
002b:25
...
```

REG - Set register

The REG command enables the user to put a specified value into a register. The REG command accesses one register at a time. The register is addressed by its number.

Syntax:

```
REG regnum,value
```

Example:

```
1>load "\user\asm\register.obj"    ; Load a file
2>reg 2,5                          ; Set r2=5
3>go                               ; Start execution
```

REGLOG - Log the registers on file

The REGLOG command generates a Register log file and starts logging of the register contents into this file. The Simulator has the ability to log the contents of all registers plus the status register for each cycle to a file. To start logging, a REGLOG command must be issued. The ENDREG command stops the logging.

The Register log contains a header, and one line for each cycle elapsed during the logging. These lines contains the cycle counter, the status register bit by bit, and the contents of all the 32 register in hexadecimal format.

Issuing a REGLOG command when logging is already enabled discards the current log file and starts logging in the new file. If the specified file already exists, it is overwritten.

Syntax:

```
REGLOG "filename"
```

Example:

```
1>load "\user\asm\addsub.obj"      ; Load a file
2>step 385                         ; Execute som instr.
3>reglog "addsub.log"              ; Start logging
4>step 12                          ; Execute some instr.
5>endreg                           ; End logging
```

Example of Register log file contents:

```
Register log file. Generated by wavrsim.
   cycle: ITOSVNZC 31302928272625242322212019181716 151413121110 9 8 7 6 5 4 3 2 1 0
00000241: 01000000 010101003f000000000000000202003f 00000000000000000000000000383938
00000242: 01000000 010101003f000000000000000202003f 00000000000000000000000000383938
00000243: 01000000 010101003f000000000000000202003f 00000000000000000000000000383938
00000244: 01000000 010201003f000000000000000202003f 00000000000000000000000000383938
00000245: 01000000 010201003f000000000000000202003f 00000000000000000000000000383938
00000246: 01000000 010201003f000000000000000202003e 00000000000000000000000000383938
00000247: 01000000 010201003f000000000000000202003e 00000000000000000000000000383938
00000248: 01000000 010201003f000000000000000202003e 00000000000000000000000000383938
00000249: 01000000 010201003f000000000000000202003e 00000000000000000000000000383938
0000024a: 01000000 010301003f000000000000000202003e 00000000000000000000000000383938
0000024b: 01000000 010301003f000000000000000202003e 00000000000000000000000000383938
0000024c: 01000000 010301003f000000000000000202003d 00000000000000000000000000383938
0000024d: 01000000 010301003f000000000000000202003d 00000000000000000000000000383938
0000024e: 01000000 010301003f000000000000000202003d 00000000000000000000000000383938
0000024f: 01000000 010301003f000000000000000202003d 00000000000000000000000000383938
00000250: 01000000 010401003f000000000000000202003d 00000000000000000000000000383938
00000251: 01000000 010401003f000000000000000202003d 00000000000000000000000000383938
00000252: 01000000 010401003f000000000000000202003c 00000000000000000000000000383938
```

RESET - Reset the Simulator

The RESET command resets the Simulator. When the Simulator is given a RESET command, it simulates a hardware reset, but with one exception: if the Simulator is running when the command is issued, it stops running. A RESET command can also be given when the Simulator is stopped.

Syntax:

```
RESET
```

Example:

```
1>load “\user\asm\addsub.obj”      ; Load a file
2>step 385                         ; Execute som instr.
3>reset                            ; Reset the Simulator
```

SAVE - Save session

The SAVE command saves all the valid commands given to the Simulator since it was started or since it was last reset. The SAVE command will also incorporate all commands which are issued from the Menus and the Toolbar. The saved file is in text format and can be changed by using a text editor. Saved commands can be reexecuted by using the FILE command.

Syntax:

```
SAVE “filename”
```

Example:

```
1>load “\user\asm\addsub.obj”      ; Load a file
2>step 385                         ; Execute 385 instr.
3>ram 1,23                         ; Set an SRAM loc.
4>
...
40>save “session.log”              ; Save commands given
```

STEP - Step through the code

The STEP command executes a specified number of instructions. Execution of the specified number of instructions is done even if a breakpoint is encountered. Included files are automatically opened during single stepping.

Syntax:

STEP *expression*

Example:

```
1>load “\user\asm\stepex.obj”      ; Load a file
2>step 10                          ; Execute 10 instr.
```

STOP - Stop the Simulator

The STOP command halts the execution of a running program. Execution can be resumed by a GO command.

Syntax:

```
STOP
```

Example:

```
1>load “demo.obj”                  ; Load a file
2>go                               ; Start execution
3>stop                             ; Halt execution
4>go                               ; Resume execution
5>reset                            ; Reset Simulator
```

Device Support

The *AVR* AT90S family of microcontrollers consists of an increasing number of derivatives. The different derivatives have different memory sizes, different instruction sets, and different peripherals in the I/O space, see the *AVR* Data Book for details.

In the Simulator Option Menu, it is possible to select a device. Selecting a device has the following consequences:

- The size of the SRAM, Program memory and EEPROM memory is set correctly
- Hardware stack is set or unset, depending on the device selected

There is no checking on if the derivative has the instructions used in the object file. In order to confirm that the code can be executed on the derivative in question, use the DEVICE directive in the Assembler.

The memory sizes and the hardware stack option can be overridden by the user in order to simulate Custom derivatives.

Some of the peripherals are also supported by the Simulator. This includes:

- EEPROM memory
- Status register
- Stack Pointer
- Timer 0

Timer 0 interrupt vectors are as for AT90S1200.