Mikrocontroller Simulator

Table of Contents

Table of Contents

Software simulator for microcontrollers

uCsim can be used to simulate microcontrollers. It supports MCS51 family. AVR and Z80 support is under development.

Download

Simulator is available for two platforms: DOS and UNIX but DOS version is not supported any more. DOS version is not finished so I call it demo version and it is available in binary only. Limitations of DOS version are:

- There is no built in help available;
- Some of the utilities are not working, for example calculator, bit simulator;
- Serial line works in mode 1 independently of mode bits.

I used this simulator to verify my applications and I never used missing feautres mentioned above, so I think that this "demo" version can be usefull anyway.

Download now

How to install

DOS version

Get the ZIP file and unzip it. ZIP file contains three files: **s51.exe** the executable itself, **dpmi16bi.ovl** and **rtm.exe**. Last two files are required because **s51.exe** is a DOS protected mode program. Put these files in a directory which is in the PATH or keep them together with the **s51.exe**.

UNIX version

UNIX version is distributed in source.

1. Get archive file, uncompress and untar it. These steps will produce a directory **ucsim−X.Y.Z** where X.Y.Z is the version number.
2. Go to the directory and configure the package. Issue **`configure`** command. It will determine your system and make **Makefile**. Installation directory can be specified with **`--prefix=`*`dir`*** option to the **`configure`**. Default directory is `/usr/local`. Executable file will be placed in `bin` subdirectory.
3. Compile the package with **`make`** command.
4. Install executable **s51** to any directory you want. It can be done with **`make install`** command which will place files in installation directory specified with **`--prefix=`*`dir`*** option of **`configure`**. Note that you may have to have special privilege to do this if installation directory is not writable by you.

How to use

DOS version

S51 for DOS has been written in Borland Pascal using Turbo Vision to produce menu driven, multiwindow user interface which is very easy to use. The program can be started using following command:

```
C:\> s51 [input file]
```

Parameter is optional. If it specified it must be the name of an Intel hex file. Some screenshots of the program and short descriptions of them:

- CPU window
- Different kind of windows
- Timer/Counters
- Interrupt system
- Terminal window and CPU option dialog box

UNIX version

Invokation.
Starting the simulator program.

Features of the simulator

- Code analyzer.
  The simulator tries to figure out places of valid instructions in code area. This feature included in both DOS and UNIX versions.
- Processor types.
  The simulator can simulate different type of microcontrollers.
- Multiple consoles.
  The simulator can handle more than one command consoles and accepts command from multiple sources. It also can be driven by other programs such as debugger interfaces.
- Serial interfaces.
  The simulator can virtually connect a terminal to serial interface of the simulated CPU.
- Command syntax and command reference. *(Now updated up to 0.2.38)*
  The simulator can be controlled via a command line interface. It accepts simple commands.

---

drdani@mazsola.iit.uni−miskolc.hu

Starting the simulator

There are separate programs to simulate different microcontroller families:

MCS51 family is simulated by **s51**
AVR family is simulated by **savr**
Z80 processor is simulated by **sz80**

The simulator can be started in the following way:

```
$ s51 [-hHVvP] [-p prompt] [-t CPU] [-X freq[k|M]] [-c file] [-s file]
[-S optionlist] [-Z portnum] [files...]
```

Specified files must be names of Intel hex files. Simulator loads them in specified order into the ROM of the simulated system.

Options:

-t CPU
: Type of CPU. Recognized types are: 51, 8051, 8751, C51, 80C51, 87C51, 31, 8031, C31, 80C31, 52, 8052, 8752, C52, 80C52, 87C52, 32, 8032, C32, 80C32, 51R, 51RA, 51RB, 51RC, C51R, C51RA, C51RB, C51RC, 89C51R, 251, C251. Note that recongition of a CPU type as option does not mean that the simulator can simulate that kind of CPU. Default type is C51.
See how to select CPU type.

-X freq[k|M]
: XTAL frequency is **freq** Hertz. **k** or **M** can be used to specify frequency in kHZ or MHz. Space is not allowed between the number and the **k** or **M**. Default value is 11059200 Hz.

-c file
: Open command console on **file**. Command consoles are on standard input and output by default. Using this option the console can be opened on any file for example on the serial interface of the computer.

-Z portnum
: Listen for incoming connections on port **portnum**. Using this option *uCsim* can serve multiple consoles. You can get a console by simply telnet into machine running *uCsim* to port **portnumber**. This option is not available on platforms which doesn't support BSD networking.
See how to use multiple consoles.

-s file
: Connect serial interface of the simulated microcontroller to the **file**. Nothing is used by default which means that characters transmitted by serial interface of the simulated microcontroller go to nowhere and it will never receive anything. If you are going to communicate with serial interface interactively the best idea is to specify a teminal with −s option.

-S in=file,out=file
: Using this option you can specify different files for input and output streams that *uCsim* uses to simulate microprocessor's serial interface.
See more about serial interface simulation.

-p prompt
: Using this option you can specify any string to be the prompt of command interpreter, for example:

```
$ s51 -p "s51> "
ucsim 0.2.12, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
```

```
under certain conditions; type `show c' for details.
s51>
```

-P

Prompt will be a null ('\0') character. This feature can be useful for programs which controlls simulator through a pipe.

-V

Verbose mode. The simulator notifies you about some kind of internal actions for example interrupts. Messages are printed on command console.

-v

Print out version number and stop.

-H

Print out types of known CPUs. Names printed out by this option can be used to determine CPU type using **`-t`** option.

-h

Print out a short help about the options and stop.

Code analyzer

This feature is included into both DOS and UNIX versions. Code analyzer analyzes the code of the simulated program and tries to figure out which address contains valid instruction. Analyzer starts analyzing of the code with address 0 which always must contain an instruction because reset of the device sets zero into program counter. So, analyzer marks address 0 that it contains an instruction. Then it marks the address of the next instruction and so on. If the analyzer finds a *jump* instruction it continues with destination address of the *jump* instruction. If it finds a *call* instruction, it recursively analyzes destination address as well as next instruction follows the *call* one. Analyzator stops if it finds an address which is already marked or if it finds a *return* instruction or an indirect *jump* which is impossible to follow. All these mean that it is impossible to discover all instructions of the program.

This problem is solved in very simple way in UNIX version. If the execution reaches an umarked address, the analyzer is called to analyze the code starting at actual address pointed by the PC. This method is not included into DOS version but this re−analyzation can be requested by pressing Ctrl−F7 at any time.

Select type of CPU

The simulator supports following type of microprocessors from MCS51 family:

8051/8031
8052/8032
8051R
89C51R
80251

8051/8031

You can select this type of microcontroller using one of the following parameter for −t option:

- 51 (HMOS type)
- 8051 (HMOS type)
- 8751 (HMOS type)
- C51 (CMOS type)
- 80C51 (CMOS type)
- 87C51 (CMOS type)
- 31 (HMOS type)
- 8031 (HMOS type)
- C31 (CMOS type)
- 80C31 (CMOS type)

It includes core 51 features:

- 64k external RAM
- 64k ROM
- 128 byte internal RAM
- 2 timers (timer #0, timer #1)
- Serial interface
- 4 8−bit quazi bi−directional ports

CMOS types also include:

- Idle mode
- Power down mode

8052/8032

You can select this type of microcontroller using one of the following parameter for −t option:

- 52 (HMOS type)
- 8052 (HMOS type)
- 8752 (HMOS type)
- C52 (CMOS type)
- 80C52 (CMOS type)
- 87C52 (CMOS type)

- 32 (HMOS type)
- 8032 (HMOS type)
- C32 (CMOS type)
- 80C32 (CMOS type)

It includes same features as 8051/8031 microcontrollers and additionaly:

- 3 timers (timer #0, timer #1, timer #2)
- 256 bytes of internal RAM

8051R

You can select this type of microcontroller using one of the following parameter for −t option (all are CMOS types):

- 51R
- 51RA
- 51RB
- 51RC
- C51RA
- C51RB
- C51RC

It includes all features that 8052/8032 does and additionaly:

- Extended interrupt priority system (4 priority levels)
- Extended serial line interface (automatic address recognition)
- Watchdog timer

89C51R

You can select this type of microcontroller using `89C51R` parameter for −t option (this CPU is always CMOS). This is a Philips clone, excluding duplicated DPTR it is similar to 8051XR from Intel.

It includes all features that 8051R does and additionaly:

- PCA (Programmable Counter Array): 5 16−bit counter, one of them can act as watchdog timer.
- Duplicated DPTR.

80251

You can select this type of microcontroller using one of the following parameter for −t option (all are CMOS types):

- 251
- C251

It includes all features that 89C51R does and additionaly:

- Nothing implemented yet.

Using multiple consoles

Why?

Using more than one console can be useful if you want to issue a command during the simulated program is executed.

How?

To get multiple consoles you have to execute the simulator in the *background* like daemons run in UNIX systems. The simulator then will listen and wait for network connection requests and provide console functions for network connections.

To run *uCsim* in the background you have to use −Z option for the simulator:

```
pigmy$ s51 -Z 5555 foo.hex
```

In this case s51 runs in forground in your command interpreters point of view. Of course you can run the program really in the background:

```
pigmy$ s51 -Z 5555 foo.hex &
```

The parameter of the −Z option is a port number. This can be number of any unused port of your machine. If the specified port is already occupied then following message appears:

```
pigmy$ s51 -Z 5555
bind: Address already in use
```

In this case you have to use an other number.

Let's suppose you have found a free port number and the simulator listens on it. Now go to somewhere else, at least to an other window and connect to the simulator:

```
other_machine$ telnet pigmy 5555
```

First parameter to the telnet command is the name of the machine where the simulator is running on. It can be `localhost` if you are on the same machine or the fully qualified host name if you are at the other end of the world. Second parameter is the number of the port where the simulator is listening. It must be the same number which was specified as parameter of the −Z option when the simulator was started (see above).

Connecting to the simulator you get a command console:

```
pigmy$ telnet pigmy 5555
Trying 127.0.0.1...
Connected to pigmy.talker.bt.
Escape character is '^]'.
ucsim 0.2.21, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
>
```

What to do with it?

Obviously you can telnet into the simulator as many times as many command consoles you want. You can start the execution using one console and while the program is executed you can, for example, modify ports on the other console.

Stop

Using quit (q) command you can not stop the simulator. It just stops the actual console and the simulator continues to listen for incoming network connections.

To stop the simulator completely you have to use kill command. Note that if you stop the simulator then all the active network connections (all other consoles) will stop.

Tricks

You can get a console on the terminal where you started the simulator. To do this you must explicitly ask the simulator to open a console on the standard input/output. You can do this using −c option and specify the actual terminal as parameter for it:

```
pigmy$ s51 −Z 5555 foo.hex −c /dev/tty
```

Using serial interfaces

Connecting a terminal

You can easily connect a terminal to the serial interface of the simulated microcontroller. This terminal is just a file so it can be anything which is represented as a file. It can be a real serial line of the computer:

```
$ s51 -s/dev/ttyS1
```

Of course you must use the actual device name of your operating system. Device name `ttyS1` above is used in Linux systems. Your system can use other names.

You can use a terminal of your system. It can be a virtual console if your system provides such as Linux does for example. On X Windows you can use **xterm** windows as terminals, one for running the simulator and one as a terminal on CPU's serial line. Here is a sample how to do this:

1. Prepare the terminal window which will be connected to the serial line:
   - Check the device name which represents the terminal:

     ```
     $ tty
     /dev/ttyp1
     ```
   - Disconnect the shell from the terminal. Usualy I use the **tail** command and any existing text file:

     ```
     $ tail -f $HOME/.profile
     ```
2. Run the simulator in the other window:

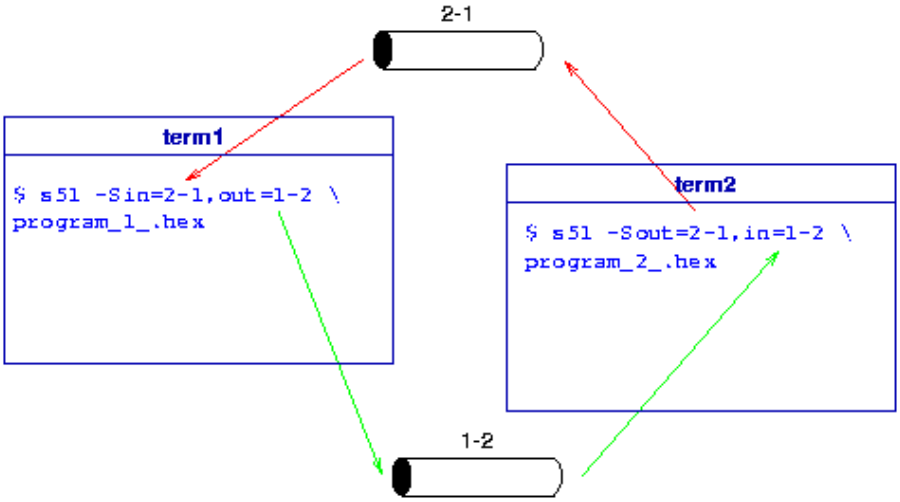
   ```
   $ s51 -s/dev/ttyp1 program.hex
   ```

   Use the output of the **tty** command above as the parameter of the `-s` option.

Every character sent out by the simulated program appears in the "terminal" window and every charater you type in there will be received by the simulated controller's serial line.

Connecting two instances of simulator

Executing two instances of the simulator, serial lines of two simulators (micros) can be connected together so they can talk to each other over their serial interface. It is because you can specify separate files for serial input and output. For example you run two simulators "1" and "2", here is the sample how to connect them:



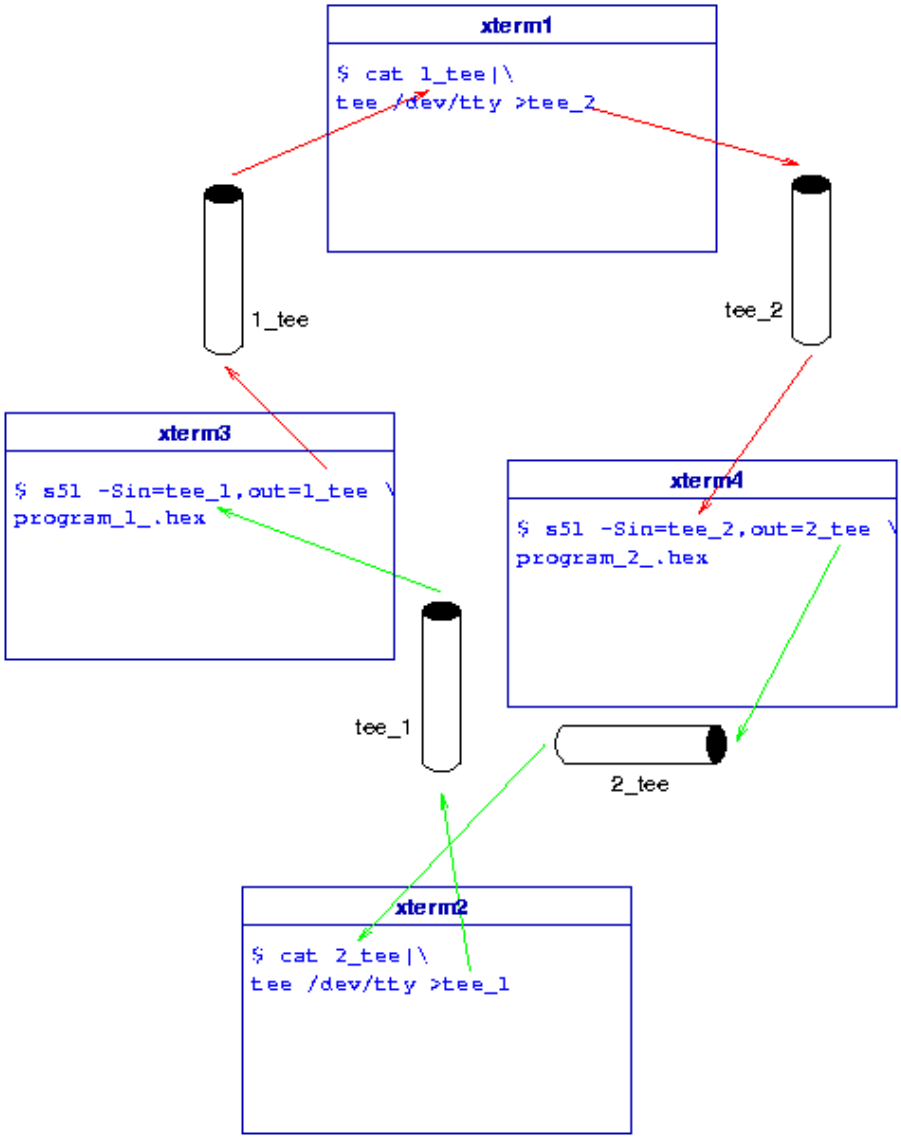
1. Make two FIFOs to represent physical wires in serial cable connecting two micros:

   ```
   $ mkfifo 1-2 2-1 # 1-2: 1->2  and 2-1: 2->1
   ```
2. Start two simulators and specify the FIFOs as input and output of serial interface:

   ```
   term1 $ # start sim "1"
   term1 $ s51 -Sin=2-1,out=1-2 program_1_.hex
   term2 $ # start sim "2"
   term2 $ s51 -Sout=2-1,in=1-2 program_2_.hex
   ```

   Because opening a pipe blocks the program until other direction is opened, the order of arguments above is **important**!
3. Debug programs as usual.

Using the most usefull unix commands **cat** and **tee** and just some more FIFOs you can monitor serial communication, here is a sample:



1. Make some FIFOs to use between simulators and tee "monitors":

```
$ mkfifo 1_tee tee_2 2_tee tee_2
```

2. Run monitoring programs (in two xterms for example):

```
xterm1 $ cat 1_tee|tee /dev/tty >tee_2 # monitor 1->2
xterm2 $ cat 2_tee|tee /dev/tty >tee_1 # monitor 2->1
```

3. Now you can start simulators (on two other terminals:)

```
xterm3 $ s51 -Sin=tee_1,out=1_tee program_1_.hex
xterm4 $ s51 -Sin=tee_2,out=2_tee program_2_.hex
```

4. Start your apps and listen what they are talking about.

Command syntax of uCsim

Command names

Type of parameters

String

Bit

Array

Number

Some commands accept parameters which in most cases can be numbers. Numbers can be entered in C−style form. If the number begins whith **`0x`** or **`0X`** it is interpreted as a hexadecimal number. If it begins with **`0`** followed by digits it is interpreted as octal number. In other cases it is interpreted as decimal number.

Symbol

Interpretation of parameters

Address

Number

Data

String

Data list

Memory

Hardware element

Bit

Command reference of ucsim

General commands and information

- **help,?** Help about commands (changed 0.2.38)
- **show** Show different kind of information (since 0.1.3) (changed 0.2.38)
- **file,load** Load FILE into ROM
- **download,dl** Download (intel hex) data
- **quit** Quit (changed 0.2.12)
- **kill** Shut down the simulator (since 0.2.12)
- **reset** Reset
- **where,Where** Search for data in memory (since 0.2.38)
- **conf** Configuration
- **state** State of simulator
- **info** Information (changed 0.2.38)
- **timer** Managing timers to measure execution time and profiling
- **set** Set various things (since 0.2.38)
- **get** Get various things (since 0.2.38)
- **fill** Fill memory region with data (since 0.2.38)

Execution

- **stop** Stop (since 0.2.12)
- **s,step** Step (renamed from **s** 0.2.38)
- **n,next** Next
- **pc** Set/get PC (since 0.1.5)
- **r,run,go** Go (since 0.2.20) (changed 0.2.38)

Breakpoints

- **break** Set fix breakpoint (changed 0.2.38)
- **tbreak** Set temporary breakpoint (changed 0.2.38)
- **clear** Clear fix breakpoint
- **delete** Delete breakpoint(s)

Dump commands

- **dump** Dump memory or bits (since 0.2.38)
- **disassemble** Disassemble
- **dc** Dump code in disass format
- **dch** Dump code in hex format
- **di** Dump Internal RAM
- **dx** Dump External RAM
- **ds** Dump SFR

Removed obsolet commands

- wi,Wi Search for string in Internal RAM (removed 0.2.38, use where,Where instead)
- wx,Wx Search for string in External RAM (removed 0.2.38, use where,Where instead)
- wc,Wc Search for string in ROM (removed 0.2.38, use where,Where instead)
- gopt Get value of option(s) (removed 0.2.38, use get option instead)

- sopt Set value of option (removed 0.2.38, use set option instead)
- g Go (changed 0.2.12) (removed 0.2.38, use run instead)
- bs Set FETCH Breakpoint (removed 0.2.20, use break or tbreak instead)
- bse Set EVENT Breakpoint (removed 0.2.38, use break or tbreak instead)
- bd Delete FETCH Breakpoint (removed 0.2.20, use clear instead)
- bde Delete EVENT Breakpoint (removed 0.2.38, use delete instead)
- ba Delete all breakpoints (removed 0.2.38, use delete instead)
- bl List Breakpoints (removed 0.2.20, use info breakpoints instead)
- dr Dump registers (removed 0.2.37, use info registers instead)
- dp Dump ports (removed 0.2.38, use info hardware instead)
- si Set Internal RAM (removed 0.2.38, use set memory instead)
- sx Set External RAM (removed 0.2.38, use set memory instead)
- sc Set code (ROM) (removed 0.2.38, use set memory instead)
- ss Set SFR area (removed 0.2.38, use set memory instead)
- sb Set bit (removed 0.2.38, use set bit instead)
- fi Fill IRAM area with data (removed 0.2.38, use fill instead)
- fx Fill XRAM area with data (removed 0.2.38, use fill instead)
- fs Fill SFR area with data (removed 0.2.38, use fill instead)
- fc Fill ROM area with data (removed 0.2.38, use fill instead)
- db Dump bit (removed 0.2.38, use dump instead)
- sp Set port pins (removed 0.2.38, use set port instead)

General commands of uCsim

Every command which changes content of ROM area such as **dl** or **set memory** deletes result of code analyzer and causes to re−analyze the code.

show

Show command can be used to display different kind of information. It must be followed by a subcommand. Subcommands are:

show copying
show warranty

show copying

This command can be used to list licensing information. It is first 10 point of the version 2 of GNU Genral Public License. If you do not accept GPL simply do not use the program.

show warranty

This command prints out last 2 point of the license ("NO WARRANTY" message).

download,dl

Download command. It is same as load above but it reads information from command console which is standard input by default. This command stops read records when it detects an "END" record which is normaly the last record. This command has two equivalent forms **download** and **dl**.

```
$ s51 −V
```

```
ucsim 0.2.24, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> do
:03000000020003F8
:1000030075812078207600D8FC900000AE83AF8203
:1000130090000001200416005E4F0A380F690009F79
:1000230012004A9000A312004A9000A712006890A1
:1000330000AD12006875D0001200B50200B3EF6581
:10004300827003EE658322E493F8740193F97402DA
:1000530093FE740393F5828E83E869700122E4931F
:10006300F6A30880F4E493FC740193FD740293FEF9
:10007300740393FF740493F8740593F582888312D1
:1000830000041700122E493A3A883A9828C838D820B
:10009300F0A3AC83AD828883898280E3212100B5FC
:1000A300212100B5000000B500B5000000B500B582
:0200B30080FECD
:1000B5007520117501AA85012075010285012022 8F
:00000001FF
197 bytes loaded
>
```

quit

Quit. This command terminates actual console, it does not ask you to confirm your intention. Simulator always reads commands from a file so end of file condition finishes too. If command console is on standard input/output, pressing the CTRL−D will quit just like the quit command.

Note that if −Z option was used at invokation then the quit command does not terminate the simulator program. In this case kill command can be used to terminate the simulator. See for more information about using multiple consoles.

```
$ s51
ucsim 0.2.24, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> q
$
```

kill

This kommand terminates the simulator. It does not ask for confirmation. It doesn't matter how many consoles are used and what commands are running on them.

?,help [command]

Help command. It prints out short description of the commands.

If a command name is given as parameter then uCsim prints out all command that has the same name.

If parameter is unique name of a command then long description of the command is printed out.

reset

Reset command. It resets the microcontroller. It has same effect as active signal on the RST pin.

```
$ s51 -V remoansi.hex
ucsim 0.2.24, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> dr
000000 00 00 00 00 00 00 00 00 ........
000000 00 .  ACC= 0x00   0 .  B= 0x00   DPTR= 0x0000 @DPTR= 0x00   0 .
000000 00 .  PSW= 0x00 CY=0 AC=0 OV=0 P=0
   000000 02 01 60 LJMP  0160
> s 2
000000 00 00 00 00 00 00 00 00 ........
000000 00 .  ACC= 0x00   0 .  B= 0x00   DPTR= 0x0000 @DPTR= 0x00   0 .
000000 00 .  PSW= 0x00 CY=0 AC=0 OV=0 P=0
   000160 c2 90    CLR   P1.0
000000 00 00 00 00 00 00 00 00 ........
000000 00 .  ACC= 0x00   0 .  B= 0x00   DPTR= 0x0000 @DPTR= 0x00   0 .
000000 00 .  PSW= 0x00 CY=0 AC=0 OV=0 P=0
   000162 c2 97    CLR   P1.7
> res
> dr
000000 00 00 00 00 00 00 00 00 ........
000000 00 .  ACC= 0x00   0 .  B= 0x00   DPTR= 0x0000 @DPTR= 0x00   0 .
000000 00 .  PSW= 0x00 CY=0 AC=0 OV=0 P=0
   000000 02 01 60 LJMP  0160
>
```

where,Where memory_type data...

Searching for some data in memory. First parameter specifies memory. Class name of the memory must be used, it can be checked using conf command which lists size and class name of all available memories.

Other parameters can be mixed list of strings (characters between " and ") and numbers. Strings can contain escape sequencies. Ucsim merges all parameters together and will search for megred list of values in specified memory.

where command do case unsensitive search while **Where** command is for case sensitive search.

Search is done in whole memory and all matches are dumped out.

```
$ /s51
ucsim 0.2.38-pre2, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> set mem x 20 "Dani d ani D ani dani Dani"
0x0014 44 61 6e 69 20 64 20 61 Dani d a
0x001c 6e 69 20 44 20 61 6e 69 ni D ani
0x0024 20 64 61 6e 69 20 44 61  dani Da
0x002c 6e 69                   ni
> where x "dani"
0x0014 44 61 6e 69             Dani
```

```
0x0025 64 61 6e 69             dani
0x002a 44 61 6e 69             Dani
> Where x "d ani"
0x0019 64 20 61 6e 69          d ani
>
```

conf

This command prints out configuration of the simulator:

```
> conf
ucsim version 0.2.37
Type of microcontroller: 51 CMOS
Controller has 8 hardware element(s).
  timer0[0]
  timer1[1]
  uart[0]
  port[0]
  port[1]
  port[2]
  port[3]
  irq[0]
Memories:
  ROM  size= 0x010000  65536 width=  8 class= "rom"
  XRAM size= 0x010000  65536 width=  8 class= "xram"
  IRAM size= 0x000100    256 width=  8 class= "iram"
  SFR  size= 0x000100    256 width=  8 class= "sfr"
>
```

First line contains version number of the program. Second line informs about type of the simulated microcontroller. Third line prints out how many hardware elements are simulated. Following lines contain information about memories. Note that size of the simulated memory segment can be bigger than size of addressable part of the memory as you see above SFR type of the memory has bigger simulated than addressable size.

state

State of the simulator and the simulated microcontroller:

```
> state
CPU state= OK PC= 0x004349 XTAL= 1.10592e+07
Total time since last reset= 0.614873 sec (6800004 clks)
Time in isr = 0.0144227 sec (159504 clks) 2.3%
Time in idle= 0 sec (0 clks)   0%
Max value of stack pointer= 0x000049, avg= 0x000026
>
```

The "CPU state" in the first line is an internal information. PC is value of the program counter. First line shows XTAL frequency too.

Following lines contain information about simulated time. First, full simulated time (elapsed from last reset) is printed out in seconds and number of clock periods then same data is printed out about time spent in interrupt service routines as well as in idle mode. Last data in lines of ISR and IDLE time shows ratio of ISRs, Idle times and main program.

Last line infroms about maximum value of the stack pointer and a "not very well" calculated average value of it.

file,load "FILE"

Loads file named FILE into the simulated code memory. File must contain data in Intel HEX format.

```
> file "../../remo.hex"
55470 words read from ../../remo.hex
>
```

Don't forget to enclose file name in quotes to make the parameter to be a string.

info

This command prints out information about different things which must be specified as paramater to the command. Following subcommands are known:

info breakpoints
info registers
info hardware

info breakpoints

This subcommand prints out information about breakpoints:

```
> b 12
Breakpoint 1 at 0x00000c: MOV   R7,A
> tb 43
Breakpoint 2 at 0x00002b: MOV   R7,A
> bse ws f 0x80
> i b
Num Type       Disp Hit   Cnt   Address  What
1   fetch      keep 1     1     0x00000c MOV   R7,A
2   fetch      del  1     1     0x00002b MOV   R7,A
1   event      keep 1     1     0x000080 ws
>
```

As you see above, the command can be shortened to "i b". The list of breakpoints contains 7 columns:

Num
: Number of the breakpoint. Normal and event breakpoints are numbered separately.

Type
: This column shows type of the breakpoint. It can be *fetch* for normal breakpoints or *event* for event breakpoints. First the normal breakpoints are listed and then the event breakpoints.

Disp
: This shows if the breakpoint is temporary (*del*) or not (*keep*).

Hit
: How many times the breakpoint must be hit before it really stops the program.

Cnt
: Counter of breakpint hits. This counter decrements and the breakpoint is activated if it reaches zero.

Address
: Address where the breakpoint is set.

What

For normal breakpoints this field contains disassembled instruction where the breakpoint is set. For event breakpoints it contains type of event.

info registers

This subcommand prints out full register set of the CPU. Output of this command depends of type of CPU.

Registers of MCS51 family

```
$ s51 remoansi.hex
ucsim 0.2.12, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> sopt stopit 1
> g
5
 * 000023 02 01 1c LJMP  011c
> i r
000000 18 02 16 ba 00 02 00 0a ........
000018 4a J  ACC= 0x0a  10 .  B= 0x00   DPTR= 0x16ba @DPTR= 0x00   0 .
000002 16 .  PSW= 0x00 CY=0 AC=0 OV=0 P=0
 * 000023 02 01 1c LJMP  011c
>
```

In first line the actual register bank is dumped out. Register banks reside in internal RAM, address at the begining of the line shows start address of actual register bank selected by RS0 and RS1 bits of PSW register.

Next two lines begin with value of indirectly addressed internal RAM cells. Second line shows IRAM cell pointed by R0 while third line shows IRAM addressed by R1.

Second line displays some important registers. First one is the accumulator. Its value dumped out in hexadecimal, decimal form and then the ASCII character of its value. It is followed by value of the B register which is dumped out in hexadecimal form only. Next is DPTR register in hexadecimal and then external RAM cell which is addressed by DPTR. This is dumped out in hexadecimal, decimal and ASCII too.

In third line you find program status word in hexadecimal and then some flag bits of PSW register. Last line is disassembled instruction at PC.

Registers of AVR family

```
$ savr test_arith.hex
ucsim 0.2.37, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> 83 words read from test_arith.hex
83 words read from test_arith.hex
> run

Stop at 0x000047: (105) User stopped
F 0x000047
> i r
000000 00 2c 21 23 20 35 19 14 03 00 00 00 00 00 00 00 .,!# 5.........
```

```
000010 00 01 1e 89 01 00 10 e2 14 00 01 10 00 00 00 00 ................
ITHSVNZC  SREG= 0x03   3 .
00000011  SP  = 0x000000
X= 0x1001 [X]= 0x00   0 .  Y= 0x0000 [Y]= 0x00   0 .  Z= 0x0000 [Z]= 0x00   0 .
 * 000047 940c 0047 jmp   0x000047
>
```

First two lines show first 32 bytes of internal RAM which is the register set of AVR controllers.

At the beginning of next two lines bits of status register are printed. These lines present hexadecimal, decimal and ASCII values of the status register too, and value of the stack pointer.

Following line shows indirect addressing registers X, Y, and Z as well as pointed memory values.

Last line is disassembled instruction ap PC.

info hardware|hw identifier

This subcommand prints out information about a unit of the controller. **identifier** specifies hardware element. One form of it is a name. Names of hardware elements can be checked by conf command. This form is accepted only when name is unique. If more than one element exists with the same name then name must be followed by id number is squere brackets in same form as it is listed by conf command.

Ouput format of this command depends on CPU family and type of the CPU too because requested unit can be different in different type of controller even in the same CPU family.

```
$ s51 -t 51
ucsim 0.2.38-pre2, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> conf
ucsim version 0.2.38-pre2
Type of microcontroller: 51 CMOS
Controller has 8 hardware element(s).
  timer0[0]
  timer1[1]
  uart[0]
  port[0]
  port[1]
  port[2]
  port[3]
  irq[0]
Memories:
  ROM  size= 0x010000  65536 width=  8 class= "rom"
  XRAM size= 0x010000  65536 width=  8 class= "xram"
  IRAM size= 0x000100    256 width=  8 class= "iram"
  SFR  size= 0x000100    256 width=  8 class= "sfr"
> i h port[2]
port[2]
P2    11111111 0xff 255 . (Value in SFR register)
Pin2  11111111 0xff 255 . (Output of outside circuits)
Port2 11111111 0xff 255 . (Value on the port pins)
> i h t[0]
timer0[0] 0x0000 13 bit timer OFF irq=0 dis prio=0
> i h u
uart[0] Shift, fixed clock MultiProc=none irq=dis prio=0
Receiver OFF RB8=0 irq=0
```

```
Transmitter TB8=0 irq=0
>
```

Timer #2 differs a little bit from other timers of MCS51:

```
$ s51 -t 52
ucsim 0.2.38-pre2, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> i h timer0
timer0[0] 0x0000 13 bit timer OFF irq=0 dis prio=0
> i h t[2]
timer2[2] 0x0000 reload 0x0000 timer OFF irq=0 dis prio=0
>
```

timer a|d|g|r|s|v id [value]

Handling of timers. First parameter of timer command determines the operation which can be

add (or simply a)
: to create a new timer. New timers are turned ON by default and initialized to value 0.

del (or simply d)
: to remove a timer if you don't need it any more.

get (or simply g)
: to get value of timers. See comment below.

run (or simply r)
: to turn a timer ON.

stop (or simply s)
: to turn a timer OFF. It still exist but doesn't count xtal periods.

value (or simply v)
: to set value of the timer (number of xtal periods). `param' is the new value.

id can be a number or a string. Timers are numbered from 1. You can use any number greater than 0 to identify a timer. Or you can use a simbolic name, in this case simulator uses the first unused number to allocate a new timer:

```
> tim a 3
> tim g 0
timer #0("time") ON: 0.463255 sec (5123232 clks)
timer #0("isr") ON: 0.0051888 sec (57384 clks)
timer #3("unnamed") ON: 0 sec (0 clks)
> tim a "a"
> tim g 0
timer #0("time") ON: 0.463255 sec (5123232 clks)
timer #0("isr") ON: 0.0051888 sec (57384 clks)
timer #1("a") ON: 0 sec (0 clks)
timer #3("unnamed") ON: 0 sec (0 clks)
>
```

If you use 0 as timer id in "get" operation, simulator prints out value of all timers including predefined ones.

set

This command can be used to set various kind of things. It requires a subcommand to specify what is going to be set. Known subcommands are:

set memory
set bit
set port
set option

set memory memory_type address data...

This command can be used to modify content of any simulated memory. First parameter must be a class name to specify type of memory. Class names can de checked by conf command.

Second parameter specifies start address of the modification.

Remaining parameters will be written into the memory starting at address specified by second parameter. Data list can include numbers as well as strings. See syntax for more details. q

Modified memory locations are dumped out.

```
$ s51
ucsim 0.2.38-pre2, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> set mem x 1 "ab\tcd\0ef\012ghq" 2 "ABC"
0x0001 61 62 09 63 64 00 65 66 ab.cd.ef
0x0009 0a 67 68 71 02 41 42 43 .ghq.ABC
> set mem sfr pcon 0x34
0x87 34                     4
>
```

set bit address 0|1

Set one bit to 0 or 1. First parameter specifies the bit. It can be the address of the bit (number or syqmbolic name) or it can be specified in *address.bitnumber* format where *address* addresses SFR area and *bitnumber* is number of bit in specified SFR location. Using this syntax any SFR (and 8051's internal RAM) location can be modified it need not be really bit addressable.

Second parameter interpreted as 1 if it is not zero.

Modified memory location is dumped out.

```
$ s51
ucsim 0.2.38-pre2, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> set bit tf1 1
0x88 80 .
> set bit 130 0
0x80 fb .
```

```
> set bit pcon.2 1
0x87 04 .
> set bit 10.7 1
0x0a 80 .
>
```

set port hardware_id data

This command can be used to set value of external circuits which connected to ports of simulated controller. First parameter specifies port number. It can be an array which specifies a hardware element or simply a number.

```
> set port 0 12
> i h po[0]
port[0]
P0    11111111 0xff 255 . (Value in SFR register)
Pin0  00001100 0x0c  12 . (Output of outside circuits)
Port0 00001100 0x0c  12 . (Value on the port pins)
> set port port[0] 23
> i h po[0]
port[0]
P0    11111111 0xff 255 . (Value in SFR register)
Pin0  00010111 0x17  23 . (Output of outside circuits)
Port0 00010111 0x17  23 . (Value on the port pins)
>
```

set option name value

Set option value. Options described at (**get option**) command can be set using this command. First parameter must be an option name and second the new value. Interpretation of the value depends on type of the option. Value for a boolean type of option interpreted as follows: if first character of the value is one of `1`, `t`, `T`, `y`, `Y` then value will be TRUE otherwise it will be FALSE.

```
$ s51 -V
ucsim 0.2.38, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> get opt
verbose TRUE Verbose flag.
stopit FALSE Stop if interrupt accepted.
debug FALSE Debug messages appears on this console.
> set opt debug t
> get opt
verbose TRUE Verbose flag.
stopit FALSE Stop if interrupt accepted.
debug TRUE Debug messages appears on this console.
>
```

get

This command can be used to get value of various kind of things. It requires a subcommand to specify what is going to be set. Known subcommands are:

get sfr
get option

get sfr address...

This command can be used to check values of SFR location(s) if SFR exists in simulated memory. Note that dump memory or ds can be used as well.

Parameters are interpreted as SFR names or addresses and values of addressed locations are dumped out.

```
$ s51
ucsim 0.2.38-pre2, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> get sfr pcon p1 0 0x80 kahd scon
0x87 00 .
0x90 ff .
0x00 00 .
0x80 ff .
Warning: Invalid address kahd
0x98 00 .
>
```

get option name

Get actual value of option(s). Some options can be set by **set option** to modify behavior of the simulator. Using **get option** you can get actual value of these options. If you use this command without parameter you get list of all options known by the program. In this way you can figure out which options can be used.

```
$ s51 -V
ucsim 0.2.38, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> get opt
verbose TRUE Verbose flag.
stopit FALSE Stop if interrupt accepted.
debug FALSE Debug flag.
> get opt stopit
stopit FALSE Stop if interrupt accepted.
>
```

This version of the simulator knows about three options. First element of the list is name of the option (**verbose**, **debug** and **stopit**). This name can be used to identify the option in **gopt** and **set option** command. Second element of the list is the value of the option. In this case type of both options is boolean, their value can be TRUE or FALSE. Third element is a short description of the option.

verbose
: ?

debug
: Debug flag can be set by **-V** option of the program to TRUE otherwise its default value is FALSE. If it is TRUE, the simulator prints out short messages about important events.

stopit

Setting this option to TRUE (default value is FALSE) forces execution to stop every time when CPU accepts an interrupt. You do not have to use breakpoints to inspect interrupts.

fill memory_type start end data

Fill memory region with specified data. First parameter specifies memory. Class name of the memory must be used, it can be checked using conf command which lists size and class name of all available memories.

start and **end** parameters specify first and last address of the memory region to be filled.

data parameter specifies the data which is used to fill the memory region.

```
$ s51
ucsim 0.2.38-pre2, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> fill x 12 0x12 0x56
> dump x 0 0x20
0x0000 00 00 00 00 00 00 00 00 ........
0x0008 00 00 00 00 56 56 56 56 ....VVVV
0x0010 56 56 56 00 00 00 00 00 VVV.....
0x0018 00 00 00 00 00 00 00 00 ........
0x0020 00                      .
>
```

Execution commands of uCsim

stop

This command stops the simulation, it freezes the CPU and all the peripherials.

```
$ s51 remoansi.hex
ucsim 0.2.24, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> g
Simulation started
> dr
000000 00 01 00 14 00 00 5a 47 ......ZG
000000 00 .  ACC= 0x47  71 G  B= 0x01   DPTR= 0x001c @DPTR= 0x47  71 G
000001 01 .  PSW= 0x00 CY=0 AC=0 OV=0 P=0
   006f02 90 00 1e MOV   DPTR,#001e
> stop
   006ac5 24 a5    ADD   A,#a5
>
```

Simulated program is executed in background and the simulator accepts commands. If it stopped by the stop command the instruction pointed by PC is dissassembled, see the **dis** command for description of disassembled form.

s,step

Step command. It executes one instruction without effect of breakpoints.

```
$ s51 remoansi.hex
ucsim 0.2.12, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> s
000000 00 00 00 00 00 00 00 00 ........
000000 00 .  ACC= 0x00   0 .  B= 0x00   DPTR= 0x0000 @DPTR= 0x00   0 .
000000 00 .  PSW= 0x00 CY=0 AC=0 OV=0 P=0
   000160 c2 90    CLR   P1.0
>
```

After execution of actual instruction pointed by PC the **step** command dumps out the register set.

n,next

Execute until next instruction is reached. This command is similar to **step** command described above but if actual instruction to execute is a subroutine call the **next** command places a dynamic breakpoint after the call instruction and starts to execute the subroutine. If the subroutine is infinite the breakpoint set by **next** will never reached. It can be a dangerous situation because the executeion started by the **next** command can not be stopped interactively. But it can be stopped by other breakpoints.

```
$ s51 remoansi.hex
ucsim 0.2.12, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> run 0 0x172
   000172 79 04    MOV   R1,#04
> dis 0x172 0 5
   000170 7f 00    MOV   R7,#00
   000172 79 04    MOV   R1,#04
   000174 12 0d b8 LCALL 0db8
   000177 0f       INC   R7
   000178 d9 fa    DJNZ  R1,0174
   00017a 75 0b 00 MOV   0b,#00
> n
000000 00 04 00 00 00 00 00 00 ........
000000 00 .  ACC= 0x00   0 .  B= 0x00   DPTR= 0x0000 @DPTR= 0x00   0 .
000004 00 .  PSW= 0x00 CY=0 AC=0 OV=0 P=0
   000174 12 0d b8 LCALL 0db8
> n
000000 00 04 00 00 00 00 00 00 ........
000000 00 .  ACC= 0x00   0 .  B= 0x00   DPTR= 0x0167 @DPTR= 0x00   0 .
000004 00 .  PSW= 0x00 CY=0 AC=0 OV=0 P=0
   000177 0f       INC   R7
>
```

pc [address]

Using this command without any parameter it simply dumps out instruction pointed by the PC. Specify address if you want to set the PC.

```
$ s51
S51 0.1.5, Copyright (C) 1997 Daniel Drotos, Talker Bt.
S51 comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> pc
 * 000000 ff       MOV   R7,A
> pc 123
 * 00007b ff       MOV   R7,A
> sopt debug t
> pc 32
Warning: maybe not instruction at 000020
 * 000020 ff       MOV   R7,A
>
```

r,run,go [start [stop]]

This command starts the execution of the simulated program.

Parameters are optional. First parameter specifies start address of the execution. By default execution starts at address specified by actual value of the PC. This can be overridden by first parameter.

If second parameter is specified it places a dynamic breakpoint at specified address which stops the execution. If stop address is not given the simulator stops if it reaches a breakpoint, or the CPU accepts an interrupt and **stopit** option is TRUE, or fatal error occures or stop command is used on an other console, or ENTER key is pressed on the console where the run command was issued.

If program execution is started the console is *frozen* it is not possible to give commands on this console to the simulator while execution is running. If it is needed to control the simulator during program execution then more consoles can be used. See using multiple consoles for more information.

Note that first instruction is executed without effect of breakpoints and simulation will be started afterwards. It means that if there is a breakpoint at start address then it will not be hit. See following example:

```
$ ./s51
ucsim 0.2.38-pre1, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> b 0x10
Breakpoint 1 at 0x000010: MOV   R7,A
> b 0x20
Breakpoint 2 at 0x000020: MOV   R7,A
> run 0x10
Warning: maybe not instruction at 0x000010
Simulation started, PC=0x000010
Stop at 0x000020: (104) Breakpoint
F 0x000020
>
```

Commands of uCsim to manage breakpoints

Two kind of breakpoints can be used: fetch and event breakpoint. **Fetch** breakpoints are classical breakpoints. They can be placed at any instruction in the code memory. Breakpoint will be hit if CPU fetches instruction code from the memory location specified by the breakpoint. Only fetching of first byte of the instruction hits the breakpoint. If the execution reaches a breakpoint it stops *before* the instruction at location specified by the breakpoint would be executed.

Event breakpoints are special onces. They cause stop of execution if event specified by the breakpoint occures. Events are:

wi
: writing into internal RAM at specified location;

ri
: reading from internal RAM at specified address;

wx
: writing into external RAM at specified location (MOVX instruction);

rx
: reading from external RAM at specified address (MOVX instruction);

ws
: writing into SFR area at specified location;

rs
: reading from SFR area at specified address;

rc
: reading from code memory at specified location (MOVC instruction).

Event breakpoint stops execution *after* specified event occured.

Every breakpoint can be **fix** (permanent) or **dynamic** (temporary). Dynamic breakpoints are automatically removed when they reached. Some commands place dynamic fetch breakpoints into the code, for example **go** or **next**.

A **hit number** can be associated to any breakpoint. This hit number specifies how many times the breakpoint must be hit before it causes the execution to stop. This hit number is 1 by default.

break addr [hit]
break memory_type r|w addr [hit]

tbreak addr [hit]
tbreak memory_type r|w addr [hit]

Set fetch or event breakpoint. The command specifies if the breakpoint will be fix (**break**) or dynamic (temporary) (**tbreak**).

Fetch or event breakpoint can be defined. First form defines fetch while second form defines event breakpoint.

Fetch breakpoint
: First parameter specifies address where the breakpoint must be placed to. It should be address of an

instruction.

Second parameter is optional and it specifies the hit number. It is 1 by default.

Event breakpoint

First parameter specifies class name of memory where we are going to watch for en event. Class names of memories can be checked by qconf command.

Second parameter specifies the event. It can be **r** to specify **read** operation or **w** which means **write** operation.

Remaining parameters are address of watched memory location and an optional hit number (1 by default).

```
$ s51 remoansi.hex
ucsim 0.2.24, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> dc 0 0x10
   000000 02 01 60 LJMP  0160
 * 000003 02 00 3c LJMP  003c
 * 000006 ff       MOV   R7,A
 * 000007 ff       MOV   R7,A
 * 000008 ff       MOV   R7,A
 * 000009 ff       MOV   R7,A
 * 00000a ff       MOV   R7,A
 * 00000b 02 3b e0 LJMP  3be0
 * 00000e ff       MOV   R7,A
 * 00000f ff       MOV   R7,A
 * 000010 ff       MOV   R7,A
> break 0x160
Breakpoint 1 at 0x000160: CLR   P1.0
> tbreak 8 2
Breakpoint 2 at 0x000008: MOV   R7,A
> g
Simulation started, PC=0x000000
Stop at 000160: (4) Breakpoint
F 000160
>
```

clear [addr...]

Delete fetch breakpoint. Parameter specifies address of breakpoint. If there is no breakpoint specified at given address this command prints out a warning message.

If parameter is not given then breakpoint at current PC will be deleted if it exists. If more than one address is specified then all breakpoints at specified addresses will be deleted.

```
> i b
Num Type       Disp Hit   Cnt   Address  What
1   fetch      keep 1     1     0x000160 CLR   P1.0
2   fetch      del  1     1     0x000180 LJMP  022a
1   event      keep 1     1     0x000006 wi
> clear 160
No breakpoint at 0000a0
> clear 0x160
> i b
Num Type       Disp Hit   Cnt   Address  What
2   fetch      del  1     1     0x000180 LJMP  022a
```

```
1   event      keep 1     1     0x000006 wi
>
```

delete [number...]

Delete breakpoint(s) by its number.

If parameter is not used then all breakpoints are deleted.

Dump commands of uCsim

dump memory_type [start [end [bytes_per_line]]]
dump bit_name...

First form can be used get content of memory while second form can be used to check value of bit or bits.

dump memory_type [start [end [bytes_per_line]]]

Hexadecimal dump of a memory region. First parameter specifies memory. Class name of the memory must be used, it can be checked using conf command which lists size and class name of all available memories.

start and **end** parameters can be used to specify the first and last address of the region. If **end** is omitted then 64 memory location is dumped. If both **start** and **end** are omitted then next 64 memory location will be dumped out. Every time when dump command is used the address of last dumped memory location is stored and next dump command automatically continues at next address.

Last parameter can be used to specify how many memory locations should be dumped out in one line. It is 8 by default.

```
$ s51 remo.hex
ucsim 0.2.38-pre2, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
> 55470 words read from remo.hex
55470 words read from remo.hex
> run
Simulation started, PC=0x000000

Stop at 0x000515: (105) User stopped
F 0x000515
> du rom 20
0x0014 00 ac ff ff ff ff ff ff ........
0x001c ff ff ff ff ff ff ff 02 ........
0x0024 01 1c ff ff ff ff ff 32 .......2
0x002c ff ff ff ff 02 0f a7 02 ........
0x0034 0e 9c 02 0d d2 02 08 41 .......A
0x003c c0 82 c0 83 c0 d0 c0 e0 ........
0x0044 c0 00 a2 90 c0 d0 c2 90 ........
0x004c 78 18 06 30 03 4b 20 92 x..0.K .
0x0054 48 30 07 05 c2 07 02 00 H0......
0x005c 9d 30 08 05 20 93 3a c2 .0.. .:.
> du x 10 20 10
```

```
0x000a ff 01 00 fa 01 01 40 01 01 44 ......@..D
0x0014 01                            .
> du x
0x0015 00 00 00 00 00 00 00 00 ........
0x001d 00 10 01 00 ae 01 00 ae ........
0x0025 02 12 e1 00 5a 85 00 01 ....Z...
0x002d 00 3b 00 00 5a 85 00 ab .;..Z...
0x0035 1f 80 00 00 01 00 01 00 ........
0x003d fa 0c 02 01 00 fa 00 02 ........
0x0045 00 01 00 ab 00 00 00 00 ........
0x004d 00 00 5a 85 ff 00 01 00 ..Z.....
0x0055 00 00 00 94 a7 01 0c a6 ........
0x005d 00 6f ff 00 00 00 00 00 .o......
>
```

dump bit_name...

disassemble [start [offset [lines]]]

Disassemble code. This command can be used to list disassembled instructions which discovered by the code analyzer. First two parameters specify the address where the list starts. First parameter is address where the command starts to search an instruction. This search goes forward. When the first instruction marked by code analyzer found the simulator skips as many instructions as you specify in second parameter. If **offset** is negative the simulator goes backward and skips specified number of instructions. Default value of **start** parameter is last address which was listed by previous **dis** command and default value of **offset** is −1. It means you can make continuous list repeating parameterless **dis** command.

In third parameter you can specify how many instructions you want to list. Default value is 20.

```
$ s51 remoansi.hex
ucsim 0.2.12, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> dis
   00d47e 22       RET
   000000 02 01 60 LJMP  0160
   000160 c2 90    CLR   P1.0
   000162 c2 97    CLR   P1.7
   000164 d2 b5    SETB  P3.5
   000166 d2 b4    SETB  P3.4
   000168 75 81 22 MOV   SP,#22
   00016b 75 d0 00 MOV   PSW,#00
   00016e 7e 00    MOV   R6,#00
   000170 7f 00    MOV   R7,#00
   000172 79 04    MOV   R1,#04
   000174 12 0d b8 LCALL 0db8
   000177 0f       INC   R7
   000178 d9 fa    DJNZ  R1,0174
   00017a 75 0b 00 MOV   0b,#00
   00017d 75 0c 00 MOV   0c,#00
   000180 02 02 2a LJMP  022a
   000183 78 22    MOV   R0,#22
   000185 76 00    MOV   @R0,#00
   000187 d8 fc    DJNZ  R0,0185
```

```
> bs f 0x180
> bs d 0x189
> dis 0x180 -3 10
   000178 d9 fa    DJNZ  R1,0174
   00017a 75 0b 00 MOV   0b,#00
   00017d 75 0c 00 MOV   0c,#00
F  000180 02 02 2a LJMP  022a
   000183 78 22    MOV   R0,#22
   000185 76 00    MOV   @R0,#00
   000187 d8 fc    DJNZ  R0,0185
D  000189 22       RET
   00018a 90 09 ec MOV   DPTR,#09ec
   00018d ae 83    MOV   R6,DPH
>
```

If there is an **F** or **D** character at the beginning of the line, it means that there is a fix or dynamic fetch breakpoint at listed address. Next element on the list can be an asterix (*****) which means that the listed address is not marked by the code analyzer. **dis** lists marked instructions only so asterix never appears in the list. Next element of the list is address displayed as six digit hexadecimal number. Address is followed by hexadecimal dump of instruction's code. Last element of the list is disassembled instruction. Every number appeared on the list is hexadecimal number.

dc [start [stop]]

Disassembled dump of code memory area. This command simply produces disassambled list of memory area specified by the parameters. Default value of **start** parameter is last address listed by previous **dc** command. If **stop** parameter is not given 20 lines are listed.

```
$ s51 remoansi.hex
ucsim 0.2.12, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> dc
   000000 02 01 60 LJMP  0160
 * 000003 02 00 3c LJMP  003c
 * 000006 ff       MOV   R7,A
 * 000007 ff       MOV   R7,A
 * 000008 ff       MOV   R7,A
 * 000009 ff       MOV   R7,A
 * 00000a ff       MOV   R7,A
 * 00000b 02 3b e0 LJMP  3be0
 * 00000e ff       MOV   R7,A
 * 00000f ff       MOV   R7,A
 * 000010 ff       MOV   R7,A
 * 000011 ff       MOV   R7,A
 * 000012 ff       MOV   R7,A
 * 000013 02 00 ac LJMP  00ac
 * 000016 ff       MOV   R7,A
 * 000017 ff       MOV   R7,A
 * 000018 ff       MOV   R7,A
 * 000019 ff       MOV   R7,A
 * 00001a ff       MOV   R7,A
 * 00001b ff       MOV   R7,A
 * 00001c ff       MOV   R7,A
>
```

dch [start [stop]]

Hexadecimal dump of code memory area from address **start** to address **stop**. Default value of start address is address of following memory cell which was dumped by previous **dch** command. If **stop** parameter is not given **dch** command lists 10 lines 8 bytes per line.

```
$ s51 remo.hex
ucsim 0.2.24, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> wc remo
0013da 52 65 6d 6f             Remo
> dch 0x13da
0013da 52 65 6d 6f 20 00 56 65 Remo .Ve
0013e2 72 73 69 6f 6e 20 31 2e rsion 1.
0013ea 30 20 00 43 6f 70 79 72 0 .Copyr
0013f2 69 67 68 74 20 28 63 29 ight (c)
0013fa 20 00 31 39 39 34 2c 39  .1994,9
001402 35 20 00 54 61 6c 6b 65 5 .Talke
00140a 72 20 42 74 2e 00 53 75 r Bt..Su
001412 6e 64 61 79 2e 00 4d 6f nday..Mo
00141a 6e 64 61 79 2e 00 54 68 nday..Th
001422 75 65 73 64 61 79 2e 00 uesday..
>
```

First element in every lines is address of first byte dumped out in the line. Next elements are hexadecimal values of bytes followed by ASCII charactes of bytes dumped out in the line. If value of the memory cell is not printable than a dot is dumped out.

di [start [stop]]

Hexadecimal dump of internal RAM area from address **start** to address **stop**. Default value of start address is address of following memory cell which was dumped by previous **di** command. If **stop** parameter is not given **di** command lists 10 lines 8 bytes per line.

```
$ s51 remoansi.hex
ucsim 0.2.24, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> sopt stopit 1
> g
Simulation started, PC=0x000000
Stop at 000023: (5) Interrupt
F 000023
> di
000000 18 02 16 ba 00 02 00 0a ........
000008 00 00 00 00 00 00 00 00 ........
000010 00 00 00 00 00 00 00 00 ........
000018 4a 00 00 00 00 00 00 00 J.......
000020 bc 27 06 2d 02 ee 35 8f .'.-..5.
000028 31 e7 42 01 0e 01 0b 00 1.B.....
000030 ec 0b 7f 10 7f a9 7e 08 ......~.
000038 fe 03 09 00 00 00 af 08 ........
000040 af 08 00 00 00 00 00 00 ........
```

```
000048 00 00 00 00 00 00 00 00 ........
>
```

dx [start [stop]]

Hexadecimal dump of external RAM area from address **start** to address **stop**. Default value of start address is address of following memory cell which was dumped by previous **dx** command. If **stop** parameter is not given **dx** command lists 10 lines 8 bytes per line.

```
$ s51 remoansi.hex
ucsim 0.2.24, Copyright (C) 1997 Daniel Drotos, Talker Bt.
ucsim comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
58659 bytes read from remoansi.hex
> sopt stopit 1
> g
Simulation started, PC=0x000000
Stop at 000023: (5) Interrupt
F 000023
> dx 0x100
000100 00 00 00 00 00 00 00 00 ........
000108 00 00 00 00 00 00 00 00 ........
000110 00 00 00 00 00 00 00 00 ........
000118 00 00 00 00 00 00 00 00 ........
000120 00 00 00 00 00 00 00 00 ........
000128 00 00 00 00 00 00 00 44 .......D
000130 61 6e 69 00 00 00 02 02 ani.....
000138 07 00 00 ff 00 00 07 cb ........
000140 08 0c 32 00 07 cb 06 05 ..2.....
000148 02 00 24 00 00 00 00 00 ..$.....
>
```