

The gdb Debugger for Assembly Language Programs

`gdb [-d sourcefiledir] [-d sourcefiledir] ... program [corefile]`

Run gdb from a shell

Miscellaneous	
<code>quit</code>	Exit gdb.
<code>directory [dir1] [dir2] ...</code>	Add directories <i>dir1</i> , <i>dir2</i> , ... to the list of directories searched for source files, or clear the directory list.
<code>help [cmd]</code>	Print a description command <i>cmd</i>

Running the Program	
<code>run [arg1],[arg2] ...</code>	Run the program with command-line arguments <i>arg1</i> , <i>arg2</i> , ...
<code>set args arg1 arg2 ...</code>	Set program's the command-line arguments to <i>arg1</i> , <i>arg2</i> , ...
<code>show args</code>	Print the program's command-line arguments.

Using Breakpoints	
<code>info breakpoints</code>	Print a list of all breakpoints.
<code>break *addr</code>	Set a breakpoint at memory address <i>addr</i> .
<code>break fn</code>	Set a breakpoint at the beginning of function <i>fn</i> .
<code>condition bnum expr</code>	Break at breakpoint <i>bnum</i> only if expression <i>expr</i> is non-zero (TRUE).
<code>commands [bnum] cmd1 cmd2 ...</code>	Execute commands <i>cmd1</i> , <i>cmd2</i> , ... whenever breakpoint <i>bnum</i> (or the current breakpoint) is hit.
<code>continue</code>	Continue executing the program.
<code>kill</code>	Stop executing the program.
<code>delete [bnum1][,bnum2]...</code>	Delete breakpoints <i>bnum1</i> , <i>bnum2</i> , ..., or all breakpoints.
<code>clear [*addr]</code>	Clear the breakpoint at memory address <i>addr</i> , or the current breakpoint.
<code>clear [fn]</code>	Clear the breakpoint at function <i>fn</i> , or the current breakpoint.
<code>disable [bnum1][,bnum2]...</code>	Disable breakpoints <i>bnum1</i> , <i>bnum2</i> , ..., or all breakpoints.
<code>enable [bnum1][,bnum2]...</code>	Enable breakpoints <i>bnum1</i> , <i>bnum2</i> , ..., or all breakpoints.

Stepping through the Program	
<code>nexti</code>	"Step over" the next instruction.
<code>stepi</code>	"Step into" the next instruction.
<code>finish</code>	"Step out" of the current function.

Examining Registers and Memory	
<code>info registers</code>	Print the contents of all registers.
<code>print/f \$reg</code>	Print the contents of register <i>reg</i> using format <i>f</i> . The format can be x (hexadecimal), d (decimal), u (unsigned decimal), o (octal), a (address), c (character), or f (floating point).
<code>x/rsf addr</code>	Print the contents of memory at address <i>addr</i> using repeat count <i>r</i> , size <i>s</i> , and format <i>f</i> . The repeat count is optional; it defaults to 1. The size is optional; it can be b (byte), h (halfword), w (word), or g (double word). The format can be x (hexadecimal), d (decimal), u (unsigned decimal), o (octal), a (address), c (character), f (floating point), s (string), or i (instruction).
<code>info display</code>	Print the display list.
<code>display/f \$reg</code>	At each break, print the contents of register <i>reg</i> using format <i>f</i> (as with a print command). Common: display/i \$pc
<code>display/si addr</code>	At each break, print the contents of memory at address <i>addr</i> using size <i>s</i> (as with an x command).
<code>display/ss addr</code>	At each break, print the string of size <i>s</i> that begins in memory at address <i>addr</i> (as with an x command).
<code>undisplay displaynum</code>	Remove <i>displaynum</i> from the display list

Examining the Call Stack	
<code>where</code>	Print the call stack.
<code>backtrace</code>	Print the call stack.
<code>frame</code>	Print the top of the call stack.
<code>up</code>	Move the context toward the bottom of the call stack.
<code>down</code>	Move the context toward the top of the call stack.