Debugging using gdb

gdb is the GNU debugger on our CS machines.
gdb is most effective when it is debugging a program that has debugging symbols linked in to it. With gcc and g++, this is accomplished using the -g option, for example, to compile sample_prog.s and have the executable stored in sample, use the following:

`arm-linux-gnueabi-gcc-5 -Wall -gstabs -o sample sample_prog.s --static`

Once you have an executable file (in this case sample) you can use gdb to debug it. First you must launch the debugger. To launch the debugger using sample from above, do the following:

// In a new terminal, type
`qemu-arm-static -g 10101 sample`

// nothing will happen; keep this terminal window open

// In a different terminal window (or the one where you compiled the program), type the following to start gdb, load symbols and connect gdb to qemu
`gdb-multiarch sample`

// to check that gdb is ready, you can type list at the command. This will list 10 lines of your program

(gdb) list
1 /* This program executes a WHILE loop for values of a less than or equal to 17
2 *
3 * The C code looks like this:
4 *
5 * main(){
6 *   int a,b,c; Declare the variables
7 *   a = 0; Initialize the variables
8 *   b = 1; "
9 *   c = 0; "
10 *

// SET ANY BREAKPOINTS
(gdb) b 11
Breakpoint 1 at 0x8928: file sample_prog.s, line 11.

// AT THE gdb PROMPT, TYPE THE TEXT THAT IS IN BOLD
(gdb) target remote :10101
Remote debugging using :10101
[New Remote target]
[Switching to Remote target]
0x00008880 in _start ()

// AT THE gdb PROMPT, TYPE c TO CONTINUE. DO NOT TYPE r.
(gdb) c
Continuing.
Breakpoint 1, at sample_prog.s:22
22 sub sp, sp, #16
(gdb)

// type n to move to the next instruction.

Quick Reference Guide to gdb
Common gdb commands:

- `run` -- run the program
- `run args` -- run program with command line args.
- `break function` -- set breakpoint at function entry
- `break linenumber` -- set breakpoint at line
- `break *addr` -- set breakpoint at address
- `break ... if cond` -- set breakpoint; break if condition
- `clear function` -- remove breakpoint at function entry
- `delete bnum` -- delete breakpoint bnum
- `disable bnum` -- disable breakpoint bnum
- `enable bnum` -- enable breakpoint bnum
- `condition bnum` -- set conditions for breakpoint bnum
- `commands bnum` -- set commands for breakpoint bnum
- `cont` -- continue execution to next break point
- `next` -- step next source level statement or function
- `nexti` -- step next machine instruction or function
- `step` -- step next source level statement
- `stepi` -- step next machine instruction
- `print expr` -- print value of expression including $n for machine registers
- `print/f expr` -- print value of expression according to format specified by f: x hexadecimal, d decimal, u unsigned decimal, o octal, a address, c character, f single precision floating point.
- `x/sf addr` -- Examine memory of size s bytes in format f: s = b one byte, s = h halfword, s = w word, s = g double word; x hexadecimal, d decimal, u unsigned decimal, o octal, a address, c character, f single precision floating point, s ascii string, I machine instruction
- `display/f expr` -- p/sf, print every gdb command
- `display/sf expr` -- x/sf, examine every gdb command
- `undisplay n` -- remove item n from display list.
- `jump *addr` -- execute next instruction at address addr.
- `printf string, expr` -- formatted output, similar to printf in C but without the parentheses surrounding the arguments.
- `info data` -- information about break, display, registers, functions, variables
- `list` -- list ten source lines
- `where` -- show call stack
- `q` -- exit gdb
- `disassemble` -- dump the assembly code
- `display` -- done at each prompt
- `commands` -- done at specific breakpoint

Note: pressing Enter repeats the last command.