Princeton University
COS 217: Introduction to Programming Systems
GDB Tutorial and Reference
for ARMv8 Assembly Language

Part 1: Tutorial

Motivation

Suppose you're composing the `power.s` program. Further suppose that the program assembles and links cleanly, but is producing incorrect results at runtime. What can you do to debug the program?

One approach is temporarily to insert calls of `printf(...)` throughout the code to get a sense of the flow of control and the values of variables at critical points. That's fine, but often is inconvenient. It is especially inconvenient in assembly language: the calls of `printf()` will change the values of registers, and thus may corrupt the very data that you wish to view.

An alternative is to use `gdb`. `gdb` allows you to set breakpoints in your code, step through your executing program one line at a time, examine the contents of registers and memory at breakpoints, etc.

Editing for gdb

To prepare your assembly language code to use `gdb`, make sure that the definition of each function ends with a `.size` directive indicating the size of that function. For example, in `power.s` the `main()` function should end with this `.size` directive:

```
.size main, (. - main)
```

Building for `gdb`

To prepare to use `gdb`, build the program with `gcc217` using the `-g` option:

```
$ gcc217 -g power.s -o power
```

Running GDB

The next step is to run `gdb`. You can run `gdb` directly from the shell. But it's much handier to run it from within `emacs`. So launch `emacs`, with no command-line arguments:

```
$ emacs
```

Now call the `emacs gdb` function via these keystrokes:

```
<Esc key> x gdb <Enter Key> power <Enter key>
```

At this point you are executing `gdb` from within `emacs`. `gdb` is displaying its `(gdb)` prompt.

Running Your Program

Issue the `run` command to run the program:

```
(gdb) run
```

`gdb` runs the program to completion, indicating that the process "exited normally."

`gdb` also displays the cryptic message "Missing separate debuginfos..." That message is innocuous; ignore it.

Command-line arguments and file redirection can be specified as part of the `run` command. For example the command `run 1 2 3` runs the program with command-line arguments 1, 2, and 3, and the command `run < myfile` runs the program with its `stdin` redirected to `myfile`.

Using Breakpoints

Set a breakpoint near the beginning of the `main()` function using the `break` command:

```
(gdb) break main
```

Run the program:

```
(gdb) run
```

`gdb` pauses execution at the beginning of the `main()` function. It opens a second window in which it displays your source code, with the about-to-be-executed line of code highlighted.

Issue the `continue` command to tell command `gdb` to continue execution past the breakpoint:

```
(gdb) continue
```

`gdb` continues past the breakpoint at the beginning of `main()`, and executes the program to completion.

Stepping Through the Program

Run the program again:

```
(gdb) run
```

Execution pauses at the beginning of the `main()` function. Issue the `next` command to execute the next instruction of your program:

```
(gdb) next
```

Continue issuing the `next` command repeatedly until the next instruction to be executed is the `bl printf` that appears near the end of the program.

Characters that are written to `stdout` do not necessarily appear in your terminal window immediately. As described in the *Debugging: Part 1* lecture, for efficiency characters written to `stdout` often are buffered; the characters are flushed from the buffer to your terminal window at some later time.

The `step` command is the same as the `next` command, except that it commands `gdb` to step into a called function which you have defined.

The `step` command does not cause `gdb` to step into a standard C function. The `stepi` ("step instruction") command causes `gdb` to step into any function, including a standard C function.

Examining Registers

Issue the `info registers` command to examine the contents of the registers:

```
(gdb) info registers
```

Issue the `print` command to examine the contents of any given register. Some examples:

```
(gdb) print/d $x1      Print as a decimal integer the 8 bytes
                       which are the contents of register X1
(gdb) print/a $x0      Print as a hexadecimal address the 8 bytes
                       which are the contents of register X0
```

Note that you must precede the name of the register with `$`.

Examining Memory

Issue the `x` command to examine the contents of memory at any given address. Some examples:

```
(gdb) x/gd &lBase            Examine as a "giant" decimal integer
                             the 8 bytes of memory at lBase
(gdb) x/gd 0x420035          Examine as a "giant" decimal integer
                             the 8 bytes of memory at 0x420035
(gdb) x/c &printfFormatStr   Examine as a char the 1 byte of
                             memory at printfFormatStr
(gdb) x/30c &printfFormatStr Examine as 30 chars the bytes of
                             memory beginning at printfFormatStr
(gdb) x/s &printfFormatStr   Examine as a string the bytes of
                             memory beginning at printfFormatStr
(gdb) x/s $x0                Examine as a string the bytes of
                             memory beginning at the address
                             contained in register X0
```

Quitting GDB

As usual, type:

```
<Ctrl-x> <Ctrl-c>
```

to exit `emacs`.

Command Abbreviations

The most commonly used `gdb` commands have one-letter abbreviations (`r`, `b`, `c`, `n`, `s`, `p`). Also, pressing the Enter key without typing a command tells `gdb` to reissue the previous command.

Part 2: Reference

gcc217 -g ... -o *program*	Assemble and link with debugging information
gdb [-d *sourcefiledir*] [-d *sourcefiledir*] ... *program* [*corefile*]	Run `gdb` from a shell
ESC x gdb [-d *sourcefiledir*] [-d *sourcefiledir*] ... *program* [*corefile*]	Run `gdb` from Emacs

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

Running the Program	
run [*arg1*],[*arg2*] …	Run the program with command-line arguments *arg1*, *arg2*, ...
set args *arg1 arg2* ...	Set program's the command-line arguments to *arg1*, *arg2*, ...
show args	Print the program's command-line arguments.

Using Breakpoints	
info breakpoints	Print a list of all breakpoints.
break *addr*	Set a breakpoint at memory address *addr*. The address can be denoted by a label.
condition *bpnum expr*	Add a condition to breakpoint *bpnum* such that the break occurs if and only if expression *expr* is non-zero (TRUE).
commands [*bpnum*] *cmd1 cmd2* ...	Execute commands *cmd1*, *cmd2*, ... whenever breakpoint *bpnum* (or the current breakpoint) is hit.
continue	Continue executing the program.
kill	Stop executing the program.
delete [*bpnum1*][,*bpnum2*]...	Delete breakpoints *bpnum1*, *bpnum2*, ..., or all breakpoints.
clear [*addr*]	Clear the breakpoint at memory address *addr*. The address can be denoted by a label. Or clear the current breakpoint.
disable [*bpnum1*][,*bpnum2*]...	Disable breakpoints *bpnum1*, *bpnum2*, ..., or all breakpoints.
enable [*bpnum1*][,*bpnum2*]...	Enable breakpoints *bpnum1*, *bpnum2*, ..., or all breakpoints.

Stepping through the Program	
next	"Step over" the next instruction.
step	"Step into" the next instruction.
finish	"Step out" of the current function.

Examining Registers and Memory	
info registers	Print the contents of all registers.
print/*f* $*reg*	Print the contents of register *reg* using format *f*. The format is typically 'd' (decimal), 'a' (address), 'x' (hexadecimal), 'c' (character), or 'i' (instruction); it defaults to 'd'.
x/*rsf addr*	Examine the contents of memory at address *addr*. The repeat count *r* is optional; it defaults to 1. The size *s* is typically 'h' (two bytes), 'w' (four bytes), or 'g' (eight bytes); its default varies based upon format *f*.
x/*rsf* &*label*	Examine the contents of memory at the address denoted by *label*.
x/*rsf* $*reg*	Examine the contents of memory at the address contained in register *reg*.
info display	Print the display list.
display/*f* $*reg*	Add an entry to the display list; at each break, print the contents of register *reg*.
display/*rsf addr*	Add an entry to the display list; at each break, print the contents of memory at address *addr*.
display/*rsf* &*label*	Add an entry to the display list; at each break, print the contents of memory at the address denoted by *label*.
undisplay *displaynum*	Remove entry with number *displaynum* from the display list.

Examining the Call Stack	
where	Print the call stack.
frame	Print the top of the call stack.
up	Move the context toward the bottom of the call stack.
down	Move the context toward the top of the call stack