CPSC-240 Computer Organization and Assembly Language

Chapter 6

DDD Debuger

Instructor: Yitsen Ku, Ph.D.
Department of Computer Science,
California State University, Fullerton, USA

CALIFORNIA STATE UNIVERSITY
FULLERTON™

Outline

- Starting DDD
- Program Execution with DDD
  - Setting Breakpoints
  - Displaying Register Contents
  - DDD/GDB Commands Summary
  - Displaying Stack Contents
  - Debugger Commands File

Starting DDD

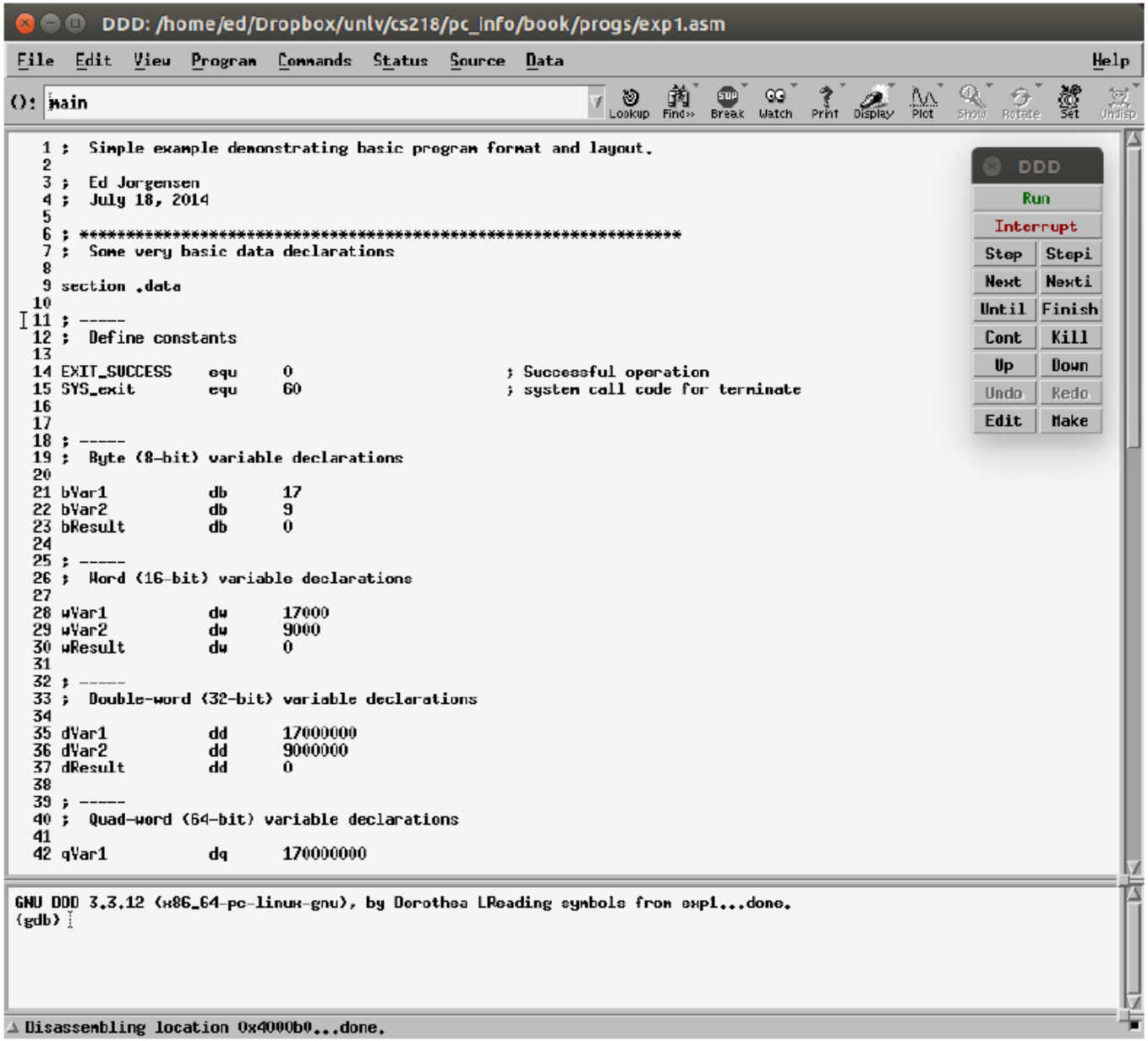
DDD Debugger

- A debugger allows the user to control execution of a program, examine variables, other memory (i.e., stack space), and display program output (if any).
- The open source GNU Data Display Debugger (DDD) is a visual front-end to the GNU Debugger (GDB) and is widely available.

Starting DDD

- The **ddd** debugger is started with the executable file. The program must be assembled and linked with the correct options (as noted in the previous chapter). For example, using the previous sample program, *example*, the command would be:

ddd example


DDD: /home/ed/Dropbox/unlv/cs218/pc_info/book/progs/exp1.asm
File Edit View Program Commands Status Source Data Help
(): main
Lookup Find>> Break Watch Print Display Plot Show Rotate Set Undisp
1 ; Simple example demonstrating basic program format and layout.
2
3 ; Ed Jorgensen
4 ; July 18, 2014
5
6 ; **
7 ; Some very basic data declarations
8
9 section .data
10
11 ; -----
12 ; Define constants
13
14 EXIT_SUCCESS equ 0 ; Successful operation
15 SYS_exit equ 60 ; system call code for terminate
16
17
18 ; -----
19 ; Byte (8-bit) variable declarations
20
21 bVar1 db 17
22 bVar2 db 9
23 bResult db 0
24
25 ; -----
26 ; Word (16-bit) variable declarations
27
28 wVar1 dw 17000
29 wVar2 dw 9000
30 wResult dw 0
31
32 ; -----
33 ; Double-word (32-bit) variable declarations
34
35 dVar1 dd 17000000
36 dVar2 dd 9000000
37 dResult dd 0
38
39 ; -----
40 ; Quad-word (64-bit) variable declarations
41
42 qVar1 dq 170000000
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
GNU DDD 3.3.12 (x86_64-pc-linux-gnu), by Dorothea LReading symbols from exp1...done.
(gdb)
Disassembling location 0x4000b0...done.

Starting DDD

- If the code is not displayed in a similar manner as shown above, the assemble and link steps should be verified. Specifically, the **-g** qualifier must be included in both the assemble and link steps.
- Built in help is available by clicking on the Help menu item (upper right-hand corner). The DDD and GDB manuals are available from the virtual library link on the class web page. To exit DDD/GDB, select **File** → **Exit** (from the top menu bar).

DDD Configuration Settings

- Some additional DDD/GDB configuration settings suggestions include:

  Edit → Preferences → General → Suppress X Warning

  Edit → Preferences → Source → Display Source Line Numbers

- These are not required, but can make using the debugger easier. If set, the options will be saved and remembered for successive uses of the debugger (on the same machine).

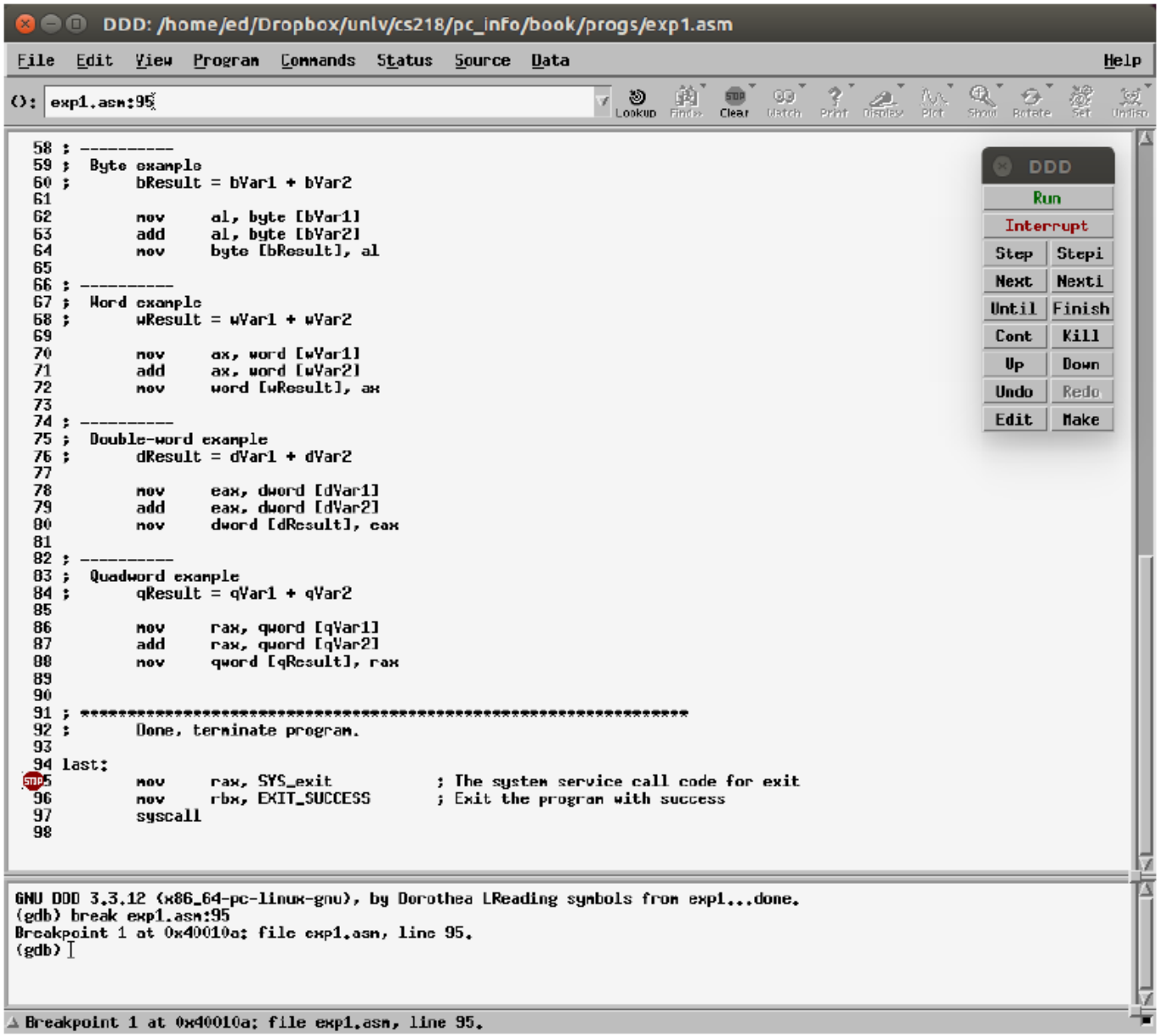
Program Execution with DDD

Program Execution with DDD

- To execute the program, click on the **Run** button from the command tool menu (shown below). Alternately, you can type **run** at the (gdb) prompt (bottom GDB console window). However, this will execute the program entirely and, when done, the results will be reset (and lost).

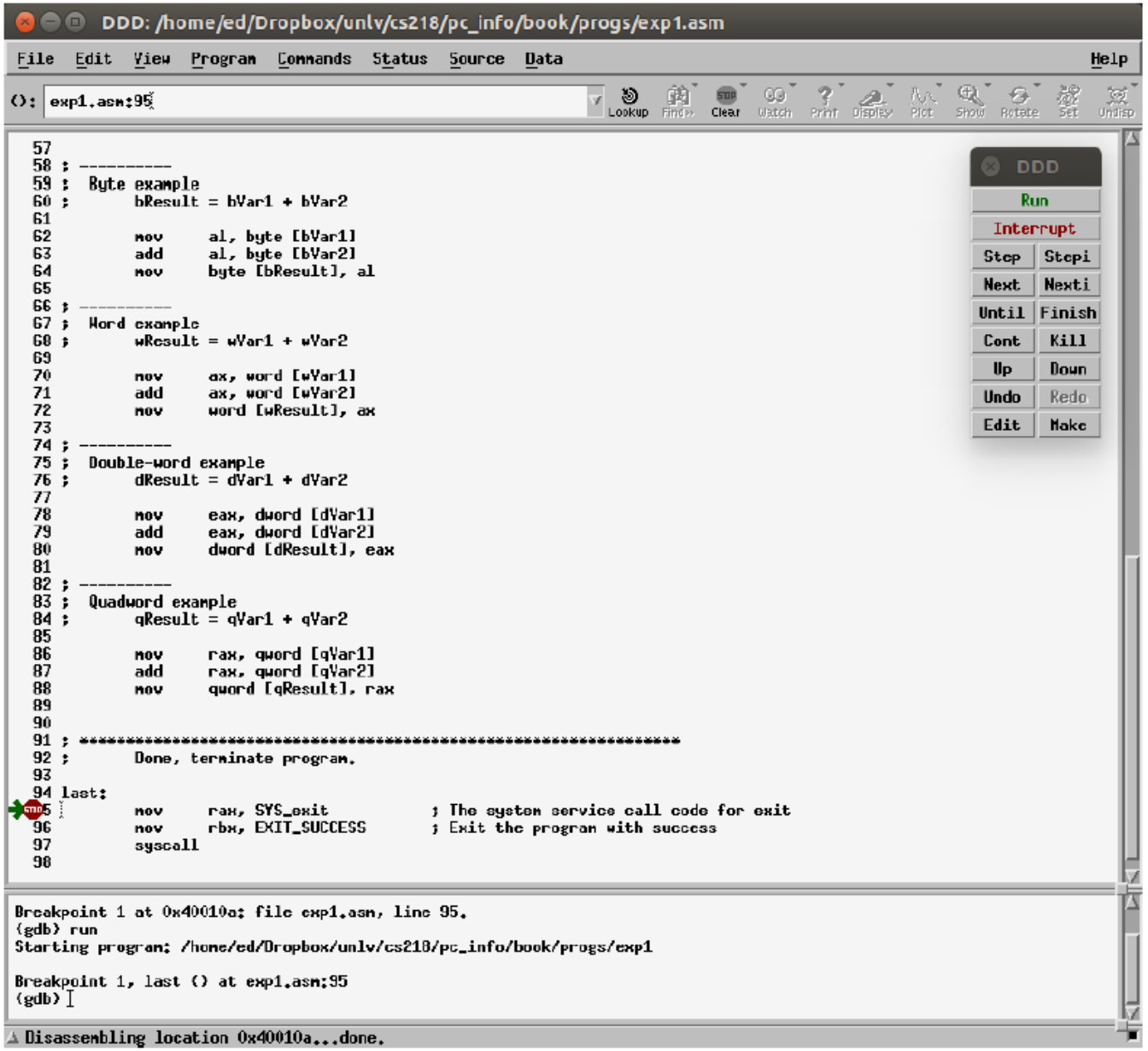
Setting Breakpoints

- In order to control program execution, it will be necessary to set a breakpoint (execution pause location) to pause the program at a user selected location. This can be done by selecting the source location (line to stop at). For this example, we will stop at line 95.
- The breakpoint can be done one of three ways:
  - Right click on the line number and select: *Set Breakpoint*
  - In the GDB Command Console, at the (gdb) prompt, type: **break last**
  - In the GDB Command Console, at the (gdb) prompt, type: **break 95**


DDD: /home/ed/Dropbox/unlv/cs218/pc_info/book/progs/exp1.asm
File Edit View Program Commands Status Source Data Help
(): exp1.asm:95
Lookup Find... Clear Watch Print Display Plot Show Rotate Set Unset
58 ; ----------
59 ; Byte example
60 ; bResult = bVar1 + bVar2
61
62 mov al, byte [bVar1]
63 add al, byte [bVar2]
64 mov byte [bResult], al
65
66 ; ----------
67 ; Word example
68 ; wResult = wVar1 + wVar2
69
70 mov ax, word [wVar1]
71 add ax, word [wVar2]
72 mov word [wResult], ax
73
74 ; ----------
75 ; Double-word example
76 ; dResult = dVar1 + dVar2
77
78 mov eax, dword [dVar1]
79 add eax, dword [dVar2]
80 mov dword [dResult], eax
81
82 ; ----------
83 ; Quadword example
84 ; qResult = qVar1 + qVar2
85
86 mov rax, qword [qVar1]
87 add rax, qword [qVar2]
88 mov qword [qResult], rax
89
90
91 ; ***
92 ; Done, terminate program.
93
94 last:
95 mov rax, SYS_exit ; The system service call code for exit
96 mov rbx, EXIT_SUCCESS ; Exit the program with success
97 syscall
98
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
GNU DDD 3.3.12 (x86_64-pc-linux-gnu), by Dorothea LReading symbols from exp1...done.
(gdb) break exp1.asm:95
Breakpoint 1 at 0x40010a: file exp1.asm, line 95.
(gdb)
Breakpoint 1 at 0x40010a: file exp1.asm, line 95.

Executing Programs

- Once the debugger is started, in order to effectively use the debugger, an initial breakpoint must be set.
- Once the breakpoint is set, the run command can be performed via clicking **Run** menu window or typing **run** at the (gdb) prompt. The program will execute up to, *but not including* the statement with the green arrow.


DDD: /home/ed/Dropbox/unlv/cs218/pc_info/book/progs/exp1.asm
File Edit View Program Commands Status Source Data Help
exp1.asm:95
Lookup Find Clear Watch Print Display Plot Show Rotate Set Undisp
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
57
58 ; ----------
59 ; Byte example
60 ; bResult = bVar1 + bVar2
61
62 mov al, byte [bVar1]
63 add al, byte [bVar2]
64 mov byte [bResult], al
65
66 ; ----------
67 ; Word example
68 ; wResult = wVar1 + wVar2
69
70 mov ax, word [wVar1]
71 add ax, word [wVar2]
72 mov word [wResult], ax
73
74 ; ----------
75 ; Double-word example
76 ; dResult = dVar1 + dVar2
77
78 mov eax, dword [dVar1]
79 add eax, dword [dVar2]
80 mov dword [dResult], eax
81
82 ; ----------
83 ; Quadword example
84 ; qResult = qVar1 + qVar2
85
86 mov rax, qword [qVar1]
87 add rax, qword [qVar2]
88 mov qword [qResult], rax
89
90
91 ; **
92 ; Done, terminate program.
93
94 last:
95 mov rax, SYS_exit ; The system service call code for exit
96 mov rbx, EXIT_SUCCESS ; Exit the program with success
97 syscall
98
Breakpoint 1 at 0x40010a: file exp1.asm, line 95.
(gdb) run
Starting program: /home/ed/Dropbox/unlv/cs218/pc_info/book/progs/exp1
Breakpoint 1, last () at exp1.asm:95
(gdb)
Disassembling location 0x40010a...done.

Run/Continue

- As needed, additional breakpoints can be set. However, click the **Run** command will re-start execution from the beginning and stop at the initial breakpoint.

Next/Step

- The **next** command will execute to the next instruction. This includes executing an entire function if necessary. The **step** command will execute one step, stepping into functions if necessary. For a single, non-function instruction, there is no difference between the **next** and **step** commands.

Displaying Register Contents

- The registers window is not displayed by default, but can be viewed by selecting **Status** → **Registers** (from the top menu bar).

DDD/GDB Commands Summary

Command	Description
quit | q	Quit the debugger.
break <label/addr> **| b <label/addr>**	Set a break point (stop point) at <label> or <address>.
run <args> | r <args>	Execute the program (to the first breakpoint).
continue | c	Continue execution (to the next breakpoint).
continue <n> | c <n>	Continue execution (to the next breakpoint), skipping ***n***-1 crossing of the breakpoint. This is can be used to quickly get to the n^{th} iteration of a loop.
step | s	Step into next instruction (i.e., steps into function/procedure calls).
next | n	Next instruction (steps through function/procedure calls).
F3	Re-start program (and stop at first breakpoint).
where	Current activation (call depth).
x/<n><f><u> $rsp	Examine contents of the stack.

DDD/GDB Commands Summary

Command	Description
x/<n><f><u> &<variable>	Examine memory location <variable> <n> number of locations to display, 1 is default. <f> format: d – decimal (signed) x – hex u – decimal (unsigned) c – character s – string f – floating-point <u> unit size: b – byte (8-bits) h – halfword (16-bits) w – word (32-bits) g – giant (64-bits)
source <filename>	Read commands from file <filename>.
set logging file <filename>	Set logging file to <filename>, default is *gdb.txt*.
set logging on	Turn logging (to a file) on.
set logging off	Turn logging (to a file) off.
set logging overwrite	When logging (to a file) is turned on, overwrite previous log file (if any).

DDD/GDB Commands, Examples

- For example, given the below data declarations:

```
bnum1 db 5
wnum2 dw -2000
dnum3 dd 100000
qnum dq 1234567890
class db "Assembly", 0
twopi dd 6.28
```

- Assuming ***signed data***, the commands to examine memory commands would be as follows:

```
x/db &bnum1
x/dh &wnum2
x/dw &dnum3
x/dg &qnum
x/s &class
x/f &twopi
```

DDD/GDB Commands, Examples

- To display an array in DDD, the basic examine memory command is used.

  x/<n><f><u> &<variable>

- For example, assuming the declaration of:

  list1 dd 100001, -100002, 100003, 100004, 100005

- The examine memory commands would be as follows:

  x/5dw &list1

- The basic examine memory command can be used with a memory address directly (as opposed to a variable name). For example:

  x/dw 0x600d44

Displaying Stack Contents

Displaying Stack Contents

- The stack is normally comprised of 64-bit, unsigned elements. The examine memory command is used, however the address is in the **rsp** register (not a variable name). The examine memory command to display the current top of the stack would be as follows:

  x/ug $rsp

- The examine memory command to display the top 6 items on the stack would be as follows:

  x/6ug $rsp

Debugger Commands File

Debugger Commands File

```
#---------------------------------
# Debugger Input Script
#---------------------------------
echo \n\n
break last
run
set pagination off
set logging file out.txt
set logging overwrite
set logging on
set prompt
echo ---------------------------------- \n
echo display variables \n
echo \n
x/100dw &list
x/dw &length
echo \n
x/dw &listMin
x/dw &listMid
x/dw &listMax
x/dw &listSum
x/dw &listAve
echo \n \n
set logging off
quit
```

Debugger Commands File (non-interactive)

- The debugger command to read a file is ''source <filename>''. For example, if the command file is named *gdbIn.txt*,

  (gdb) source gdbIn.txt

- Based on the above commands, the output will be placed in the file *out.txt*. The output file name can be changed as desired.

Debugger Commands File (non-interactive)

- It is possible to obtain the output file directly without an interactive DDD session. The following command, entered at the command line, will execute the command in the input file on the given program, create the output file, and exit the program.

  gbd <gdbIn.txt prog

- Which will create the output file (as specified in the *gdbIn.txt* input file) and exit the debugger.

Thanks