# 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





## **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.



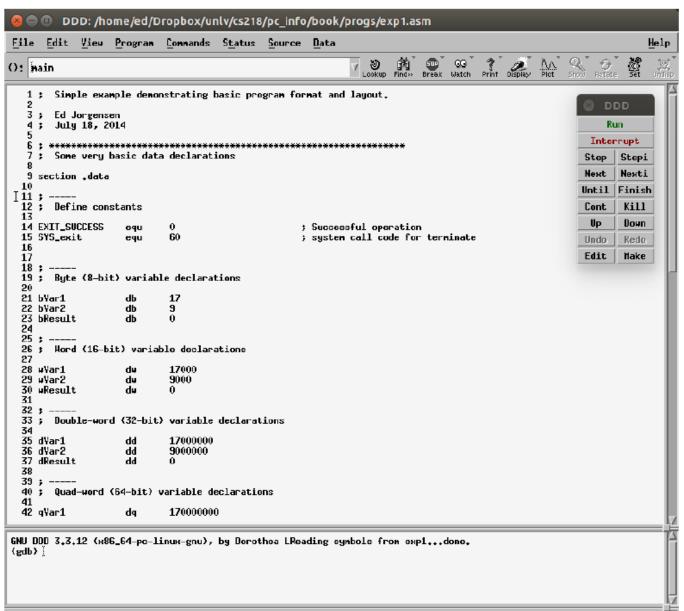
#### **Starting DDD**

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







#### **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).

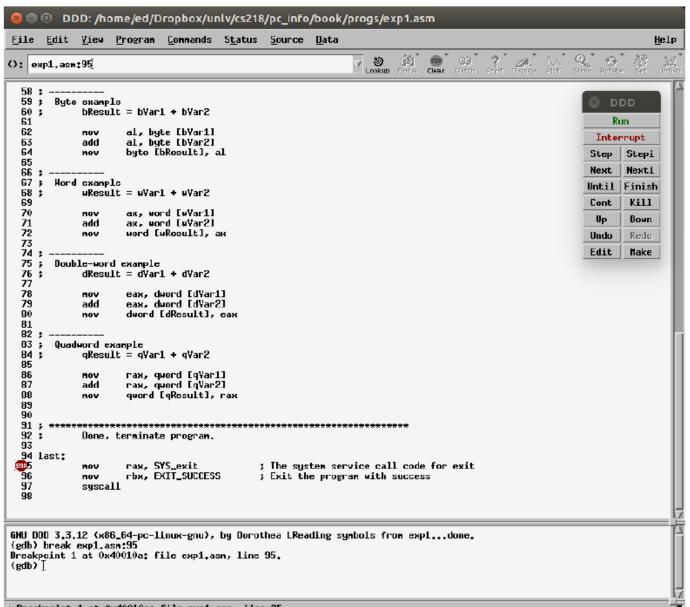
Chap 06



#### **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



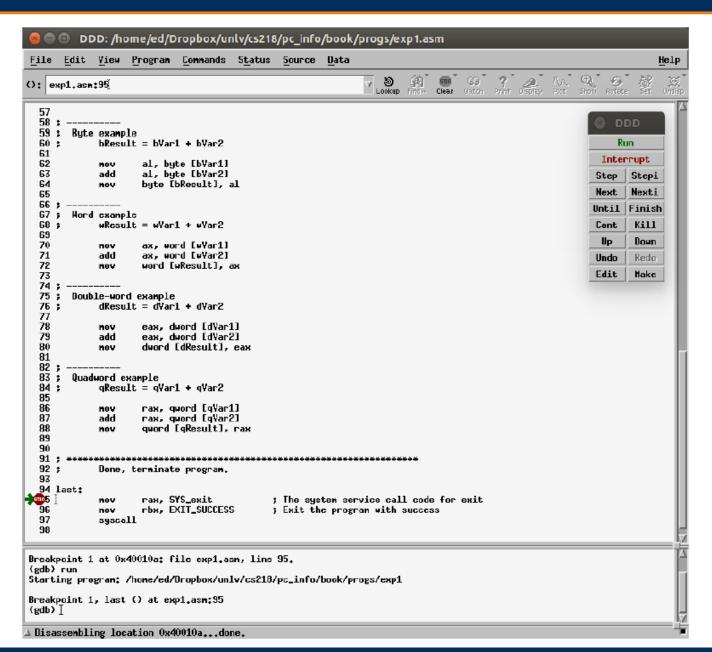




#### **Executing Programs**

- 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.







#### **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=""></label></label>	Set a break point (stop point) at <label> or <address>.</address></label>
run <args>   r <args></args></args>	Execute the program (to the first breakpoint).
continue   c	Continue execution (to the next breakpoint).
continue <n>   c <n></n></n>	Continue execution (to the next breakpoint), skipping $n$ -1 crossing of the breakpoint. This is can be used to quickly get to the $n$ <sup>th</sup> 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</u></f></n>	Examine contents of the stack.



### **DDD/GDB Commands Summary**

Command	Description
x/ <n><f><u> &amp;<variable></variable></u></f></n>	Examine memory location <variable> <n> number of locations to display, 1 is default.</n></variable>
	<f> format: d – decimal (signed)</f>
	x – hex
	u – decimal (unsigned)
	c – character
	s – string
	f – floating-point
	<u> unit size: b – byte (8-bits)</u>
	h – halfword (16-bits)
	w – word (32-bits)
	g – giant (64-bits)
source <filename></filename>	Read commands from file <filename>.</filename>
set logging file <filename></filename>	Set logging file to <filename>, default is gdb.txt.</filename>
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 <gdbln.txt prog</li>
- Which will create the output file (as specified in the gdbln.txt input file) and exit the debugger.



## **End of Chapter 6**