



# *Assembly Language Lecture Series:* **x86-64 Introduction to GDB**

Sensei RL Uy, College of Computer Studies,  
De La Salle University, Manila, Philippines

# Copyright Notice

This lecture contains copyrighted materials and is use solely for instructional purposes only, and not for redistribution.

Do not edit, alter, transform, republish or distribute the contents without obtaining express written permission from the author.

# What is **GDB**?

**01** Acronym:  
GNU Debugger

**02** Debugger for Assembly  
Language

**03** Also supports C Language

**04** See what's *inside* your  
program

<https://www.gnu.org/software/gdb/>

# Summary of GDB Command

Syntax	Description
Set disassembly-flavor intel	Intel format display
break main	Set breakpoint
Run (r)	Run program
disassemble/r main	List program; /r means opcode
display/i \$pc	Display one line instruction after every execution
next instruction (ni)	Run next instruction

# Summary of GDB Command

Syntax	Description
<code>display/fmt \$reg</code> <code>Display/nfu &lt;mem_address&gt;</code>  Example: <code>display/x \$esi</code> <code>display /4fw 0x403030</code>	Display reg or memory value after every execution (n- # of locations; f-format; u-unit)  <b>fmt:</b> o(octal), x(hex), d(decimal), u(unsigned decimal), t(binary), f(float), a(address), i(instruction), c(char) and s(string) <b>unit:</b> b(byte), h(halfword), w(word), g(giant, 8 bytes)
<code>undisplay #</code>	Cancel display request
<code>Print/fmt [\$reg   \$memvar]</code>	Print value (but preferably register)

# Summary of GDB Command

Syntax	Description
Info all	Show all register
Info address <var_name>	Obtain the address of <b>var_name</b>
x/nfu <mem_address>	Display address (n- # of locations; f-format; u-unit) unit b(byte), h(halfword), w(word), g(giant, 8 bytes)

# GDB in SASM

The screenshot displays the SASM (SAS Assembly) IDE interface. The main window shows assembly code with line numbers 1 through 13. Line 9 is highlighted in yellow. The code includes a macro definition and assembly instructions.

```
1 %include "io64.inc"
2 section .data
3 var1 dq 0x123456789ABCDEF0, 0x1122334455667788
4
5 section .text
6 global CMAIN
7 CMAIN:
8 ;write your code here
9 → mov rax, [var1]
10 mov rbx, [var1*8]
11 add rax, rbx
12 xor rax, rax
13 ret
```

Below the code editor, the disassembly for the `main` function is shown:

```
> disassemble/r main
Dump of assembler code for function main:
=> 0x00000000004014e0 <+0>: 48 8b 04 25 10 30 40 00   mov rax,QWORD PTR ds:0x403010
0x00000000004014e8 <+8>: 48 8b 1c 25 18 30 40 00   mov rbx,QWORD PTR ds:0x403018
0x00000000004014f0 <+16>: 48 01 d8      add rax,rbx
0x00000000004014f3 <+19>: 48 31 c0      xor rax,rax
0x00000000004014f6 <+22>: c3           ret
0x00000000004014f7 <+23>: 66 0f 1f 84 00 00 00 00 nop WORD PTR [rax+rax*1+0x0]
```

On the right side, the Registers window shows the current state of CPU registers:

Register	Hex	Value
rax	0x7ffd06e74d28	1407247192753
rbx	0x1	1
rcx	0x1	1
rdx	0xab3f60	11222880
rsi	0x34	52
rdi	0xab22f0	11215600
rbp	0x8	0x8
rsp	0x60fe38	0x60fe38
r8	0xab27c0	11216832
r9	0x7ffd084b6950	1407247426133
r10	0x0	0
r11	0x60fc58	6356056
r12	0xab3f60	11222880
r13	0x0	0
r14	0x0	0
r15	0x0	0
rip	0x4014e0	0x4014e0 <main>
eflags	0x202	[ IF ]
cs	0x33	51
ss	0x2b	43
ds	0x0	0
es	0x0	0
fs	0x0	0
gs	0x0	0
st0	0	(raw 0x00000000)
st1	0	(raw 0x00000000)
st2	0	(raw 0x00000000)
st3	0	(raw 0x00000000)
st4	0	(raw 0x00000000)
st5	0	(raw 0x00000000)
st6	0	(raw 0x00000000)
st7	0	(raw 0x00000000)
fcml	0x37f	895
fstat	0x0	0
ftag	0x0	0
fiseq	0x0	0
fioff	0x0	0

The bottom status bar shows the GDB command prompt and buttons for Print and Perform.

# GDB in SASM

The screenshot displays the SASM (SASM Assembly Editor) interface, which is used for editing and debugging assembly code. The main window shows the assembly code, and the right-hand side displays the memory, registers, and GDB command output.

**Memory:**

Variable or expression	Value	Type
var1	(0x123456789abcdef0, 0x1122334455667788)	Hex -q -2
Add variable...		Smart -d -Array size

**Assembly Code:**

```
1 %include "io64.inc"
2 section .data
3 var1 dq 0x123456789ABCDEF0, 0x1122334455667788
4
5 section .text
6 global CMAIN
7 CMAIN:
8 ;write your code here
9 → mov rax, [var1]
10 mov rbx, [var1+8]
11 add rax, rbx
12 xor rax, rax
13 ret
```

**Registers:**

Register	Hex
rax	0x7ffd06e74d28 1407247192753
rbx	0x1 1
rcx	0x1 1
rdx	0xab3f60 11222880
rsi	0x34 52
rdi	0xab22f0 11215600
rbp	0x8 0x8
rsp	0x60fe38 0x60fe38
r8	0xab27c0 11216832
r9	0x7ffd084b6950 1407247426133
r10	0x0 0
r11	0x60fec58 6356056
r12	0xab3f60 11222880
r13	0x0 0
r14	0x0 0
r15	0x0 0
rip	0x4014e0 0x4014e0 <mai
eflags	0x202 [ IF ]
cs	0x33 51
ss	0x2b 43
ds	0x0 0
es	0x0 0
fs	0x0 0
gs	0x0 0
st0	0 (raw 0x000000)
st1	0 (raw 0x000000)
st2	0 (raw 0x000000)
st3	0 (raw 0x000000)
st4	0 (raw 0x000000)
st5	0 (raw 0x000000)
st6	0 (raw 0x000000)
st7	0 (raw 0x000000)
fctrl	0x37f 895
fstat	0x0 0
ftag	0x0 0
fiseg	0x0 0
fioff	0x0 0

**GDB Command Output:**

```
> info address var1
Symbol "var1" is at 0x403010 in a file compiled without debugging.
```



# GDB in SASM

The screenshot displays the SASM (SAS Assembly) IDE interface. The main window shows assembly code with the following content:

```
1 %include "io64.inc"
2 section .data
3 var1 dq 0x123456789ABCDEF0, 0x1122334455667788
4
5 section .text
6 global CMAIN
7 CMAIN:
8 ;write your code here
9 → mov rax, [var1]
10 mov rbx, [var1+8]
11 add rax, rbx
12 xor rax, rax
13 ret
```

Below the code, the GDB output is shown:

```
> x/2hg 0x403010
0x403010 <var1>: 0x123456789abcdef0 0x1122334455667788
> x/2dg 0x403010
0x403010 <var1>: 1311768467463790320 1234605616436508552
```

The right sidebar contains several panels:

- Memory:** A table with columns for Variable or expression, Value, Type, and Address. It shows 'var1' with a value of '0x123456789abcdef0, 0x1122334455667788' and a type of 'dq' (double quadword).
- Registers:** A table with columns for Register, Hex, and Value. It lists various registers (rax, rbx, rcx, rdx, rsi, rdi, rbp, rsp, r8, r9, r10, r11, r12, r13, r14, r15, rip, cs, ss, ds, es, fs, gs, st0, st1, st2, st3, st4, st5, st6, st7, fctrl, fstat, ftag, fiseq, fioff) and their corresponding values.
- Input/Output:** Two empty text boxes for input and output.

The bottom status bar shows the GDB command line and buttons for 'Print' and 'Perform'.

# GDB in SASM

The screenshot displays the SASM (SASM Assembly Studio) interface. The main window shows assembly code with line numbers 1 through 13. Line 11, `add rax, rbx`, is highlighted in yellow. The code includes a macro `CMMAIN` and a `ret` instruction. Below the code, the GDB command prompt shows the results of `p/x $rax` and `p/x $rbx`, both displaying hexadecimal values.

**Memory Table:**

Variable or expression	Value	Type
var1	0x123456789abcdef0, 0x1122334455667788	Hex -q -2
Add variable...		Smart -d - Array size

**Registers Table:**

Register	Hex
rax	0x123456789abcdef0
rbx	0x1122334455667788
rcx	0x1
rdx	0xab3f60
rsi	0x34
rdi	0xab22f0
rbp	0x8
rsp	0x60fe38
r8	0xab27c0
r9	0x7fcd084b6950
r10	0x0
r11	0x60fc58
r12	0xab3f60
r13	0x0
r14	0x0
r15	0x0
rip	0x4014f0
eflags	0x202
cs	0x33
ss	0x2b
ds	0x0
es	0x0
fs	0x0
gs	0x0
st0	0
st1	0
st2	0
st3	0
st4	0
st5	0
st6	0
st7	0
fctrl	0x37f
fstat	0x0
ftag	0x0
fiseg	0x0
fioff	0x0

**GDB Command Prompt:**

```
> p/x $rax
$16 = 0x123456789abcdef0
> p/x $rbx
$17 = 0x1122334455667788
```

# GDB in SASM

The screenshot displays the SASM IDE interface. The main window shows assembly code with line numbers 1 through 13. Line 3 is highlighted in yellow. The code includes a macro definition, a global symbol, and assembly instructions. The GDB console at the bottom shows the execution of commands to display the value of the \$rax register and the contents of memory at address 0x403010. The registers window on the right shows the current state of the CPU registers, with the \$rax register containing the value 0x123456789abcdef0. The output window shows the message "[New Thread 12592.0x6348]".

**Memory**

Variable or expression	Value	Type
var1	(0x123456789abcdef0, 0x1122334455667788)	Hex -q -2 Address
Add variable...		Smart -d -Array size Address

**\*New**

```
1 %include "io64.inc"
2 section .data
3 var1 dq 0x123456789ABCDEF0, 0x1122334455667788
4
5 section .text
6 global CMAIN
7 CMAIN:
8 ;write your code here
9     mov rax, [var1]
10    mov rbx, [var1+8]
11    add rax, rbx
12    xor rax, rax
13    ret
```

**Input**

**Output**

[New Thread 12592.0x6348]

**Registers**

Register	Hex
rax	0x123456789abcdef0
rbx	0x1122334455667788
rcx	0x1
rdx	0xab3f60
rsi	0x34
rdi	0xab22f0
rbp	0x8
rsp	0x60fe38
r8	0xab27c0
r9	0x7ffd084b6950
r10	0x0
r11	0x60fc58
r12	0xab3f60
r13	0x0
r14	0x0
r15	0x0
rip	0x4014f0
eflags	0x202
cs	0x33
ss	0x2b
ds	0x0
es	0x0
fs	0x0
gs	0x0
st0	0
st1	0
st2	0
st3	0
st4	0
st5	0
st6	0
st7	0
fctrl	0x37f
fstat	0x0
ftag	0x0
fiseg	0x0
fioff	0x0

**GDB command:**

```
> display/x $rax
5: /x $rax = 0x123456789abcdef0
> display /2hg 0x403010
6: x/2xg 0x403010
0x403010 <var1>: 0x123456789abcdef0 0x1122334455667788
```

# GDB in SASM

The screenshot displays the SASM (Simple Assembler Simulator) interface, which is used for writing and executing assembly code. The main window is divided into several panes:

- Memory Pane:** Shows a table with columns for Variable or expression, Value, and Type. It includes options for Hex, Smart, and Array size, and checkboxes for Address.
- Code Pane:** Contains assembly code with line numbers. The code includes a preprocessor directive, a section declaration, a variable definition, and a main routine with assembly instructions. Line 11 is highlighted in yellow.
- Registers Pane:** Displays the current state of CPU registers, including their names, hexadecimal values, and decimal values. The registers are organized into two columns: General Purpose Registers (rax, rbx, rcx, rdx, rsi, rdi, rbp, rsp, r8-r15, rip) and Segment Registers (eflags, cs, ss, ds, es, fs, gs).
- Input/Output Pane:** Shows the input and output of the program. The output pane displays the message "[New Thread 12592.0x6348]".
- Command Line:** At the bottom, there is a command line where the user can enter GDB commands. The command "> info registers" has been entered, and the output is displayed in the registers pane.

The assembly code in the Code Pane is as follows:

```
1 %include "io64.inc"
2 section .data
3 var1 dq 0x123456789ABCDEF0, 0x1122334455667788
4
5 section .text
6 global CMAIN
7 CMAIN:
8 ;write your code here
9     mov rax, [var1]
10    mov rbx, [var1+8]
11    add rax, rbx
12    xor rax, rax
13    ret
```

The output of the "info registers" command is shown in the Registers pane:

Register	Hex	Dec
rax	0x123456789ABCDEF0	131176846
rbx	0x1122334455667788	123460561
rcx	0x1	1
rdx	0xab3f60	11222880
rsi	0x34	52
rdi	0xab22f0	11215600
rbp	0x8	8
rsp	0x60fe38	0x60fe38
r8	0xab27c0	11216832
r9	0x7ffd084b6950	140724742
r10	0x0	0
r11	0x60fc58	6356056
r12	0xab3f60	11222880
r13	0x0	0
r14	0x0	0
r15	0x0	0
rip	0x4014f0	0x4014f0
eflags	0x202	[ IF ]
cs	0x33	51
ss	0x2b	43
ds	0x0	0
es	0x0	0
fs	0x0	0
gs	0x0	0
st0	0	(raw 0x00)
st1	0	(raw 0x00)
st2	0	(raw 0x00)
st3	0	(raw 0x00)
st4	0	(raw 0x00)
st5	0	(raw 0x00)
st6	0	(raw 0x00)
st7	0	(raw 0x00)
fctrl	0x37f	895
fstat	0x0	0
ftag	0x0	0
fiseg	0x0	0
fioff	0x0	0