

CSC 6580

Spring 2020

Instructor: Stacy Prowell

gdb

—

Using gdb

You can use gdb to debug your assembly programs. To debug the **binary** program you could use:

```
$ gdb binary
```

Or you can load **binary** after starting gdb with **file binary**.

Run with **run**.

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31
sprowell@sanders:~/CSC6580-2020-Spring/02-06$ gdb
GNU gdb (Ubuntu 8.2.91.20190405-0ubuntu3) 8.2.91.20190405-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file binary
Reading symbols from binary...
(No debugging symbols found in binary)
(gdb) disass main
Dump of assembler code for function main:
    0x0000000000401b80 <+0>:   movabs rdi,0xff00f043210099aa
    0x0000000000401b8a <+10>:  call 0x401b9a <write_binary_qword>
    0x0000000000401b8f <+15>:  call 0x401c1d <write_endl>
    0x0000000000401b94 <+20>:  mov     eax,0x0
    0x0000000000401b99 <+25>:  ret
End of assembler dump.
(gdb) █
```

Using gdb

Instead of referring to line numbers you use addresses or labels. For instance, to see the code starting at the label `main`, use:

`disassemble main`

You can abbreviate `disassemble` as `disass`.

The disassembly will be "around" the address you give (or if none, around `RIP`), or you can give a range, or an address. Include `/r` to see the bytes.

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31
sprowell@sanders:~/CSC6580-2020-Spring/02-06$ gdb
GNU gdb (Ubuntu 8.2.91.20190405-0ubuntu3) 8.2.91.20190405-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file binary
Reading symbols from binary...
(No debugging symbols found in binary)
(gdb) disass main
Dump of assembler code for function main:
   0x0000000000401b80 <+0>:    movabs rdi,0xff00f043210099aa
   0x0000000000401b8a <+10>:   call 0x401b9a <write_binary_qword>
   0x0000000000401b8f <+15>:   call 0x401c1d <write_endl>
   0x0000000000401b94 <+20>:   mov    eax,0x0
   0x0000000000401b99 <+25>:   ret
End of assembler dump.
(gdb) █
```

Using gdb

Set breakpoints with **break** (or just **b**) followed by a location. Use an asterisk (*) for a memory expression.

```
b main
```

```
b *main+20
```

List breakpoints with **info b**, and delete them with **d** and their number.

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from binary...
(No debugging symbols found in binary)
(gdb) b main
Breakpoint 1 at 0x401b80
(gdb) b *main+20
Breakpoint 2 at 0x401b94
(gdb) info b
Num      Type             Disp Enb Address            What
1        breakpoint       keep y   0x0000000000401b80  <main>
2        breakpoint       keep y   0x0000000000401b94  <main+20>
(gdb) run
Starting program: /home/sprowell/snap/odrive-unofficial/2/Google Drive/CSC6580-2020-Spring/02-06/binary

Breakpoint 1, 0x0000000000401b80 in main ()
(gdb) cont
Continuing.
1111 1111 0000 0000 1111 0000 0100 0011 0010 0001 0000 0000 1001 1001 1010 1010

Breakpoint 2, 0x0000000000401b94 in main ()
(gdb) d 1
(gdb) info b
Num      Type             Disp Enb Address            What
2        breakpoint       keep y   0x0000000000401b94  <main+20>
          breakpoint already hit 1 time
(gdb) █
```

Using gdb

Catch signals with `catch`. You can catch all the usual signal, or you can catch specific signals by name or number. Of particular interest:

`catch signal SIGSEGV`

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31

Register group: general
rax      0x2          2
rbx      0x7fffffffdf50 140737488346960
rcx      0x0          0
rdx      0x4059000000000000 463673729135463288
rsi      0xffffffff00000000 4503595332403200
rdi      0x4c40f0      4997360
rbp      0x7fffffffde20 0x7fffffffde20
rsp      0x7fffffffdd18 0x7fffffffdd18

0x414ceb <printf+27> mov QWORD PTR [rsp+0x48],r9
0x414cf0 <printf+32> test al,al
0x414cf2 <printf+34> je 0x414d2b <printf+91>
> 0x414cf4 <printf+36> movaps XMMWORD PTR [rsp+0x50],xmm0
0x414cf9 <printf+41> movaps XMMWORD PTR [rsp+0x60],xmm1
0x414cfe <printf+46> movaps XMMWORD PTR [rsp+0x70],xmm2
0x414d03 <printf+51> movaps XMMWORD PTR [rsp+0x80],xmm3
0x414d0b <printf+59> movaps XMMWORD PTR [rsp+0x90],xmm4
0x414d13 <printf+67> movaps XMMWORD PTR [rsp+0xa0],xmm5

native process 27528 In: printf L?? PC: 0x414cf4
(gdb) run
Starting program: /home/sprowell/snap/odrive-unofficial/2/Google Drive/CSC6580-2020-Spring/02-06/sqrt_list
[Inferior 1 (process 27523) exited normally]
(gdb) run 100
Starting program: /home/sprowell/snap/odrive-unofficial/2/Google Drive/CSC6580-2020-Spring/02-06/sqrt_list 100

Catchpoint 1 (signal SIGSEGV), 0x0000000000414cf4 in printf ()
(gdb) █
```

Using gdb

Find out what's on the stack with [where](#).

Look at that! The stack is not aligned. Use [up](#) to move up a stack frame (assuming that what's on the stack is a correct stack frame).

There's the problem! The [push rax](#) mis-aligns the stack.

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31

Register group: general
rax      0x2                2
rbx      0x7fffffffdf50     140737488346960
rcx      0x0                0
rdx      0x4059000000000000 4636737291354636288
rsi      0xffffffff00000000 4503595332403200
rdi      0x4c40f0           4997360
rbp      0x7fffffffde20     0x7fffffffde20
rsp      0x7fffffffddf8     0x7fffffffddf8

0x401c26 <loop+47>    mov     eax,0x2
0x401c2b <loop+52>    push    rax
0x401c2c <loop+53>    call   0x414cd0 <printf>
> 0x401c31 <loop+58>    add     rbx,0x8
0x401c35 <loop+62>    jmp     0x401bf7 <loop>
0x401c37 <good>       mov     eax,0x0
0x401c3c <done>       leave
0x401c3d <done+1>    ret
0x401c3e <done+2>    xchg    ax,ax

native process 27528 In: loop                                L??    PC: 0x401c31
#2  0x0000000000000002 in ?? ()
#3  0x0000000000402b80 in ?? ()
#4  0x0000000000402c10 in ?? ()
#5  0xffffdf5040590000 in ?? ()
#6  0x00000000200007ff in ?? ()
#7  0x0000000000402b80 in ?? ()
#8  0x0000000000402440 in __libc_start_main ()
--Type <RET> for more, q to quit, c to continue without paging--
#9  0x0000000000401aea in _start ()
(gdb) █
```

Using gdb

Use the Text User Interface (TUI) mode.

- `gdbtui`
- `gdb -tui`
- `gdb` and then `ctrl+x ctrl+a` (toggle)

Also:

- `tui enable`/`tui disable`
- `tui reg`

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31
[ No Source Available ]
None No process in: L?? PC: ??
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
--Type <RET> for more, q to quit, c to continue without paging--
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) |
```


Using gdb

The interface can get messed up when it scrolls.
You can refresh it with `ctrl+l` or with `refresh`.

A trick some folks use is to create a *user hook* for commands that will automatically call `refresh`.

```
define hook-nexti
  refresh
end
```

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31
B+ 0x401b80 <main> movabs rdi,0xff00f043210099aa
B+ 0x401b8a <main+10> call 0x401b9a <write_binary_qword>
> 0x401b8f <main+15> call 0x401c1d <write_endl>
> 0x401b8f <main+15> call 0x401c1d <write_endl>
> 0x401b94 <main+20> mov eax,0x0
0x401b9a <write_binary_qword> push rbp
0x401b9b <write_binary_qword+1> mov rbp,rsi
0x401b9e <write_binary_qword+4> push rdi
0x401b9f <write_binary_qword+5> mov ecx,0x8
0x401ba4 <write_binary_qword.top> mov eax,0x0
0x401ba9 <write_binary_qword.top+5> mov al,BYTE PTR [rbp+rcx*1-0x9]
0x401bad <write_binary_qword.top+9> push rcx
0x401bae <write_binary_qword.top+10> push rax
0x401baf <write_binary_qword.top+11> and rax,0xf0
0x401bb5 <write_binary_qword.top+17> shr rax,0x2
0x401bb9 <write_binary_qword.top+21> mov edi,0x1
0x401bbe <write_binary_qword.top+26> lea rsi,[rax+0x4b80f0]
0x401bc5 <write_binary_qword.top+33> mov edx,0x4

native process 23233 In: main L?? PC: 0x401b8f
94
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb) ni
0x0000000000401b8a in main ()
(gdb) ni
0x0000000000401b8f in main () 0100 0011 0010 0001 0000 0000 1001 1001 1010 1010
(gdb) ni
0x0000000000401b94 in main ()
(gdb) █
```

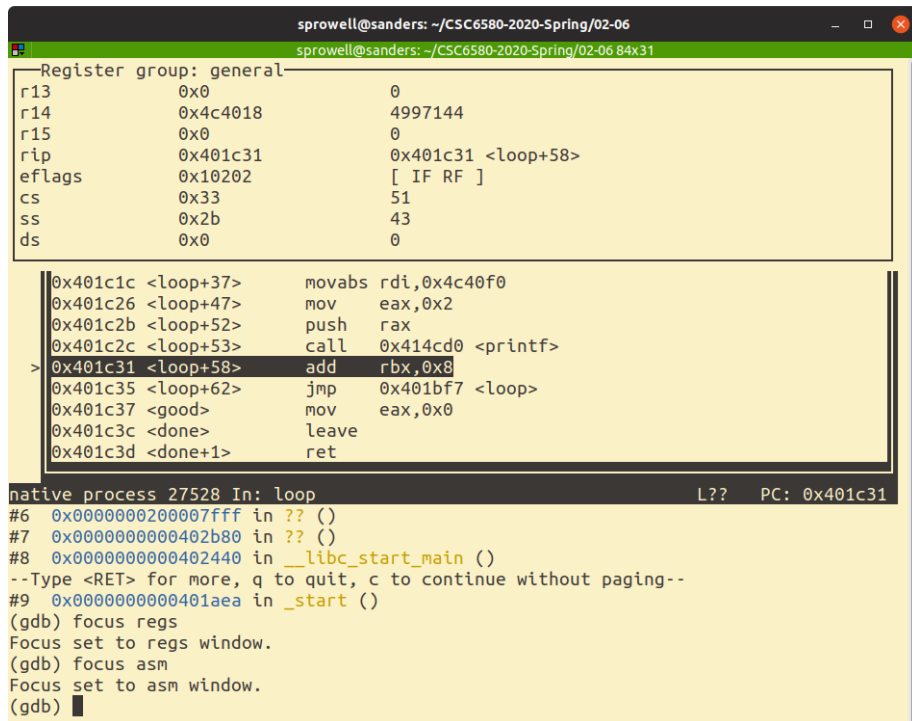
Using gdb

Switch to the assembly layout or to register layout.

layout asm

layout regs

The display follows [RIP](#)... but you can scroll around, too, with the arrow keys. You can change the focused window with [focus](#).



```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31

Register group: general
r13      0x0          0
r14      0x4c4018     4997144
r15      0x0          0
rip      0x401c31     0x401c31 <loop+58>
eflags   0x10202     [ IF RF ]
cs       0x33         51
ss       0x2b         43
ds       0x0          0

0x401c1c <loop+37>    movabs rdi,0x4c40f0
0x401c26 <loop+47>    mov  eax,0x2
0x401c2b <loop+52>    push rax
0x401c2c <loop+53>    call 0x414cd0 <printf>
> 0x401c31 <loop+58>  add  rbx,0x8
0x401c35 <loop+62>    jmp  0x401bf7 <loop>
0x401c37 <good>      mov  eax,0x0
0x401c3c <done>      leave
0x401c3d <done+1>    ret

native process 27528 In: loop                                L??  PC: 0x401c31
#6  0x00000000200007fff in ?? ()
#7  0x0000000000402b80 in ?? ()
#8  0x0000000000402440 in __libc_start_main ()
--Type <RET> for more, q to quit, c to continue without paging--
#9  0x0000000000401aea in _start ()
(gdb) focus regs
Focus set to regs window.
(gdb) focus asm
Focus set to asm window.
(gdb) █
```

Using gdb

Use Intel syntax with
`set disassembly-flavor intel.`

Put this (and other commands) in your
`.gdbinit`.

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31

Register group: general
rax      0x0      0
rbx      0x0      0
rcx      0x0      0
rdx      0x0      0
rsi      0x0      0
rdi      0x0      0
rbp      0x0      0x0
rsp      0x7fffffffdf40 0x7fffffffdf40

> 0x401a60 <_start> xor    ebp,ebp
0x401a62 <_start+2> mov    r9,rdx
0x401a65 <_start+5> pop    rsi
0x401a66 <_start+6> mov    rdx,rsi
0x401a69 <_start+9> and    rsp,0xfffffffffffffff0
0x401a6d <_start+13> push   rax
0x401a6e <_start+14> push   rsp
0x401a6f <_start+15> mov    r8,0x402be0
0x401a76 <_start+22> mov    rcx,0x402b50

native process 24211 In: _start L?? PC: 0x401a60
(gdb) starti
Starting program: /home/sprowell/snap/odrive-unofficial/2/Google Drive/CSC6580-2020-Spring/02-06/binary

Program stopped.
0x00000000401a60 in _start ()
(gdb) █
```

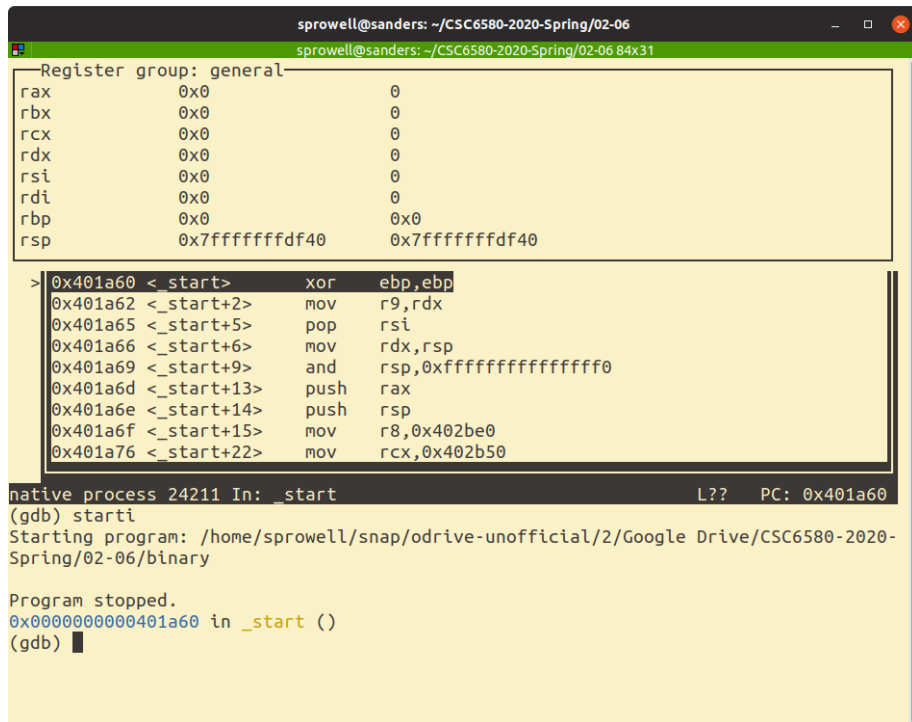
Using gdb

See what's in a register with `print` (or `p`). There are many optional modifiers you can add after `print`; for instance, `/x` prints hexadecimal.

```
print $rsp
print /x $rsp
```

Use `*` to try to dereference pointers. Use parens to cast.

```
print *(int)$rip
```



The screenshot shows a GDB terminal window with the following content:

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31

Register group: general
rax      0x0      0
rbx      0x0      0
rcx      0x0      0
rdx      0x0      0
rsi      0x0      0
rdi      0x0      0
rbp      0x0      0x0
rsp      0x7fffffffdf40 0x7fffffffdf40

> 0x401a60 <_start> xor    ebp,ebp
0x401a62 <_start+2> mov    r9,rdx
0x401a65 <_start+5> pop    rsi
0x401a66 <_start+6> mov    rdx,rsi
0x401a69 <_start+9> and    rsp,0xfffffffffffffff0
0x401a6d <_start+13> push   rax
0x401a6e <_start+14> push   rsp
0x401a6f <_start+15> mov    r8,0x402be0
0x401a76 <_start+22> mov    rcx,0x402b50

native process 24211 In: _start L?? PC: 0x401a60
(gdb) starti
Starting program: /home/sprowell/snap/odrive-unofficial/2/Google Drive/CSC6580-2020-Spring/02-06/binary

Program stopped.
0x0000000000401a60 in _start ()
(gdb) █
```

Using gdb

Watch a value with `display`.

```
display $rsp
```

```
display /s $rdi
```

```
display $rsp
```

Remove a display with `undisplay` and the display number (or remove them all).

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31

0x401bd8 <main+8>      je      0x401bf3 <main.okay>
> 0x401bda <main+10>   movabs  rdi,0x4b8132
0x401be4 <main+20>     call    0x411960 <puts>
0x401be9 <main+25>     mov     eax,0x1
0x401bee <main+30>     jmp     0x401c95 <main.done>
0x401bf3 <main.okay>   sub     rsp,0x10
0x401bf7 <main.okay+4> push    rsi
0x401bf8 <main.okay+5> mov     rdi,rsi
0x401bf9 <main.okay+8> mov     rdi,QWORD PTR [rdi+0x8]
0x401bff <main.okay+12> call    0x40f1f0 <atoi>
0x401c04 <main.okay+17> mov     QWORD PTR [rbp-0x8],rax
0x401c08 <main.okay+21> pop     rdi
0x401c09 <main.okay+22> mov     rdi,QWORD PTR [rdi+0x10]
0x401c0d <main.okay+26> call    0x40f1f0 <atoi>
0x401c12 <main.okay+31> mov     QWORD PTR [rbp-0x10],rax
0x401c16 <main.okay+35> movabs  rdi,0x4b815a
0x401c20 <main.okay+45> call    0x411960 <puts>
0x401c25 <main.okay+50> mov     rdi,QWORD PTR [rbp-0x8]

native process 31339 In: main                                L??    PC: 0x401bda
3: $rdi = 1
4: $rsi = 140737488346952
(gdb) layout asm
(gdb) nt
0x0000000000401bda in main ()
1: $rip = (void (*)(void)) 0x401bda <main+10>
2: $rsp = (void *) 0x7ffffffde20
3: $rdi = 1
4: $rsi = 140737488346952
(gdb) █
```

Using gdb

Similarly, see what's in memory with `x`. Expect strings with `/s`.

```
x $rip
```

```
x /s $rdi
```

Several other options. See `help x` and `help print` for more.

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31

Register group: general
r13      0x0          0
r14      0x4c4018     4997144
r15      0x0          0
rip      0x401c31     0x401c31 <loop+58>
eflags   0x10202     [ IF RF ]
cs       0x33        51
ss       0x2b        43
ds       0x0          0

0x401c0e <loop+23>   call    0x401c40 <sqrtf64>
0x401c13 <loop+28>   movsd   xmm1,xmm0
0x401c17 <loop+32>   movsd   xmm0,QWORD PTR [rbp-0x14]
0x401c1c <loop+37>   movabs  rdi,0x4c40f0
0x401c26 <loop+47>   mov     eax,0x2
0x401c2b <loop+52>   push    rax
0x401c2c <loop+53>   call    0x414cd0 <printf>
> 0x401c31 <loop+58>   add     rbx,0x8
0x401c35 <loop+62>   jmp     0x401bf7 <loop>

native process 27528 In: loop                                L??   PC: 0x401c31
(gdb) refresh
(gdb) print $rip
$11 = (void (*)(())) 0x401c31 <loop+58>
(gdb) x $rip
0x401c31 <loop+58>:      "H\203\303\b\353\300\270"
(gdb) x /x $rip
0x401c31 <loop+58>:      0x48
(gdb) x /s $rdi
0x4c40f0:      "sqrt(%f) = %f\n"
(gdb) █
```

Using gdb

Modify the content of registers with set.

```
set $edi = 0x21aae436
set $rsi = $rdi + 100
```

Yes, this includes **RIP** and **RSP**.

```
sprowell@sanders: ~/CSC6580-2020-Spring/02-06
sprowell@sanders: ~/CSC6580-2020-Spring/02-06 84x31
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./sqrt_list...
(No debugging symbols found in ./sqrt_list)
(gdb) catch signal SIGSEGV
Catchpoint 1 (signal SIGSEGV)
(gdb) layout regs
(gdb) disass
Dump of assembler code for function main:
   0x0000000000401be0 <+0>:  push    rbp
   0x0000000000401be1 <+1>:  mov     rbp, rsp
   0x0000000000401be4 <+4>:  sub     rsp, 0x20
=>  0x0000000000401be8 <+8>:  mov     DWORD PTR [rbp-0x4], edi
   0x0000000000401beb <+11>: add     rsi, 0x8
   0x0000000000401bef <+15>: mov     QWORD PTR [rbp-0xc], rsi
   0x0000000000401bf3 <+19>: mov     rbx, QWORD PTR [rbp-0xc]
End of assembler dump.
(gdb) set $edi = 10
(gdb) print $edi
$1 = 10
(gdb) █
```



Using gdb

Running the program:

- `starti`
- `stepi`
- `nexti (ni)`
- `down / up`
- `b / info b / delete`

Use a leading asterisk with an address.

Getting information:

- `info registers / info files`
- `print $rip`
- `print/d $rip`
- `print/x $rip`
- `x/16xg $rip`

`o`(octal), `x`(hex), `d`(decimal), `u`(unsigned decimal),
`t`(binary), `f`(float), `a`(address), *`i`*(instruction), `c`(char),
`s`(string) and `z`(hex, zero padded on the left)

`b`(byte), `h`(halfword), `w`(word), `g`(giant, 8 bytes)



Using gdb

You can script gdb in several ways. One of the easiest is to use the `-ex` switch to run a command.

```
gdb -batch -ex "file addsub" -ex "disass main"  
gdb -batch -ex "disass/r main" addsub
```

(The `-batch` causes gdb to exit after it runs the commands.)

```
function main() { gdb -batch -ex "file $1" -ex "disass main" ; }
```

Anti-Disassembly



Basic Blocks

A **basic block** is a sequence of code that has a single entry, single exit, and exactly one path from beginning to end.



Basic Blocks

```
section .data
    msg db "Hello, world!", 0x0a
    len equ $ - msg
    stdout equ 1
    sys_exit equ 60
    sys_write equ 1

section .text
global _start
_start:
    mov rbp, rsp           ; Helps the debugger.

    ; Print the string.
    mov rdi, stdout
    mov rsi, msg
    mov rdx, len
    mov rax, sys_write
    syscall

    ; Exit with the length as the exit value.
    mov rdi, len
    mov rax, sys_exit
    syscall
    ret
```



Confuse

- The **machine state** is the truth. Everything else is (potentially) a lie. Or at least, misleading.
- Assembly (and disassembly) is treated as if it were execution... but it is more akin to compilation (or decompilation).
- Almost all disassemblers **assume** that an address should appear only once in a listing. This is not true. It isn't even true that an address always corresponds to a single instruction!



Linear disassembly

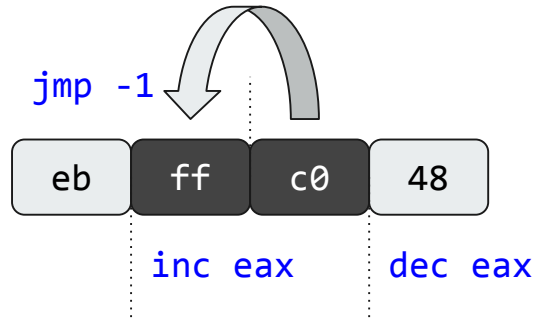
jmp -1



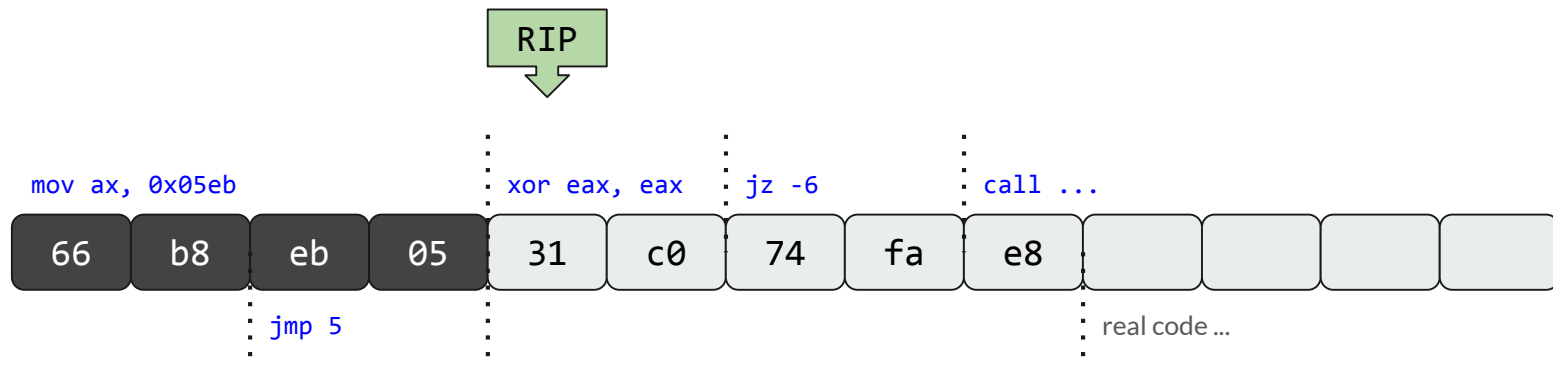
inc eax

dec eax

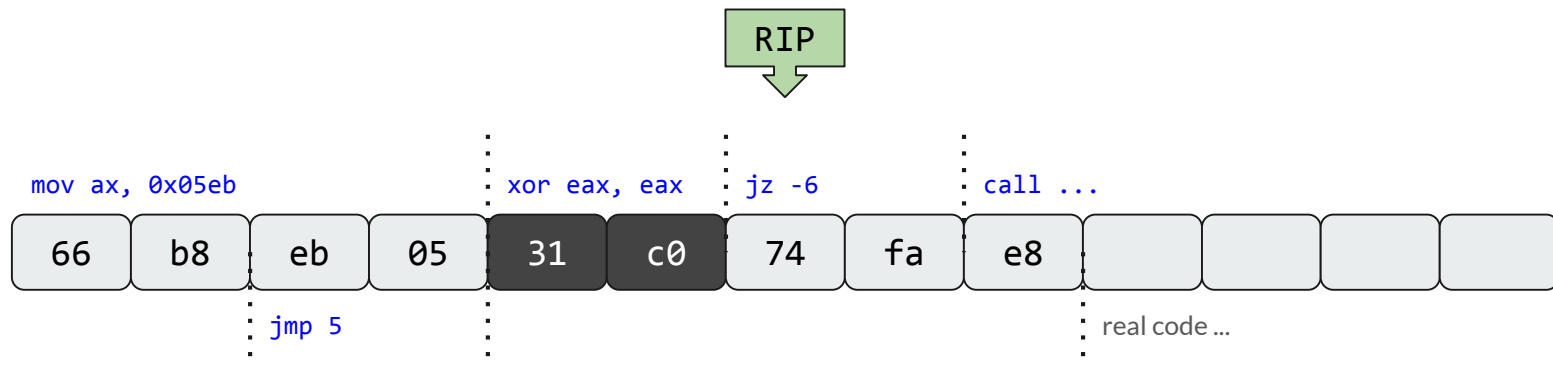
Linear disassembly



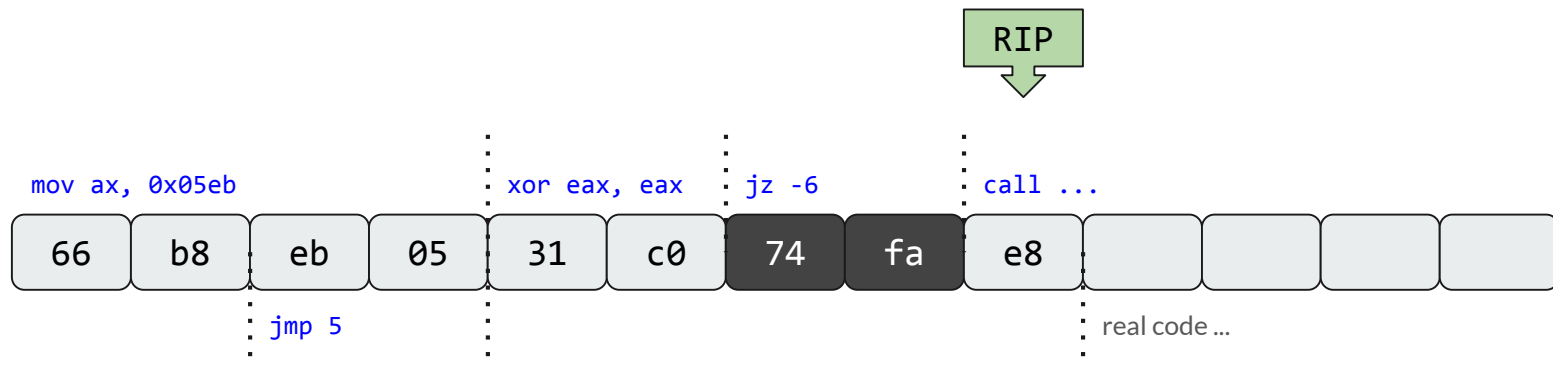
Things can get arbitrarily complex



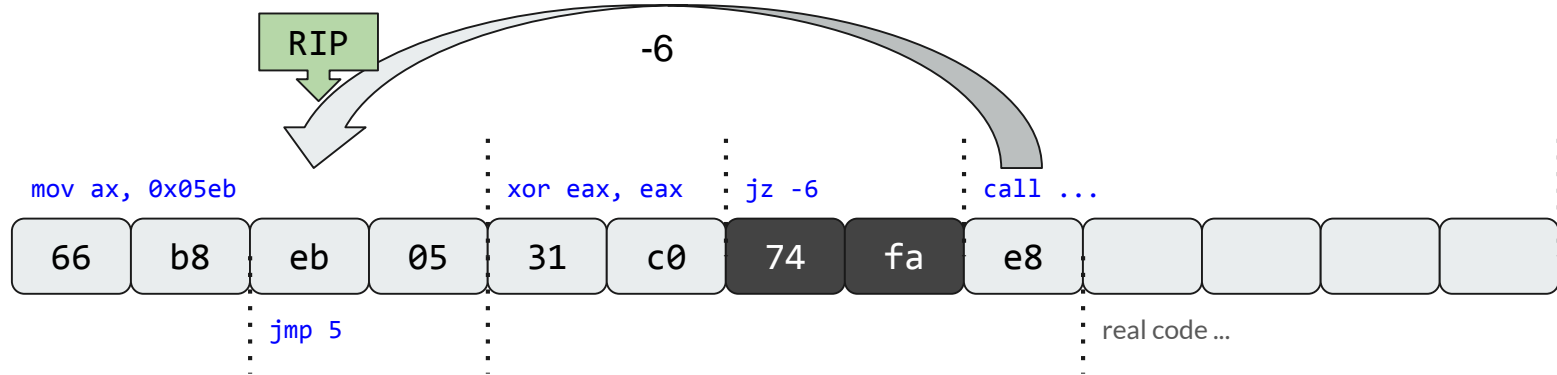
Things can get arbitrarily complex



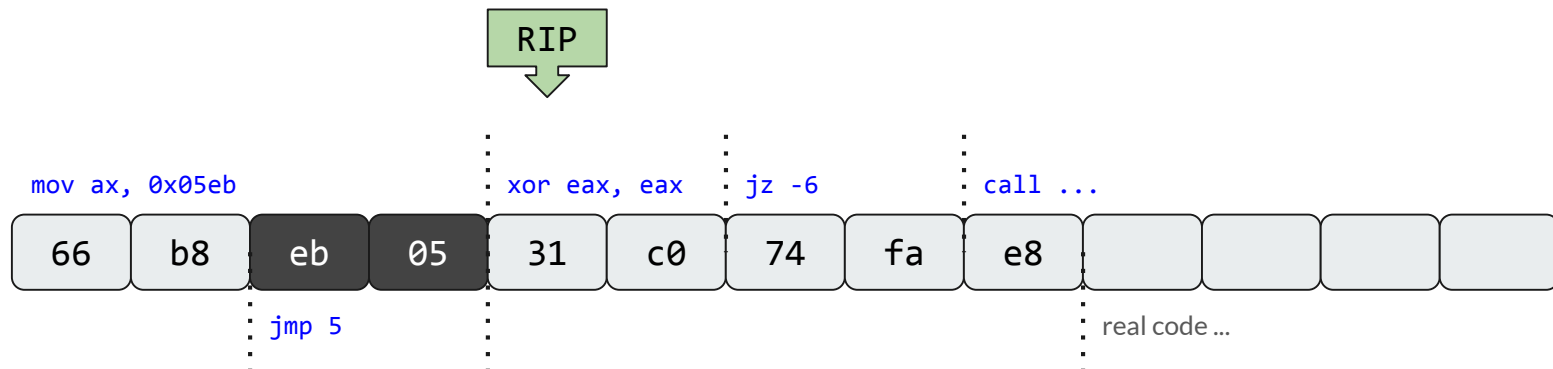
Things can get arbitrarily complex



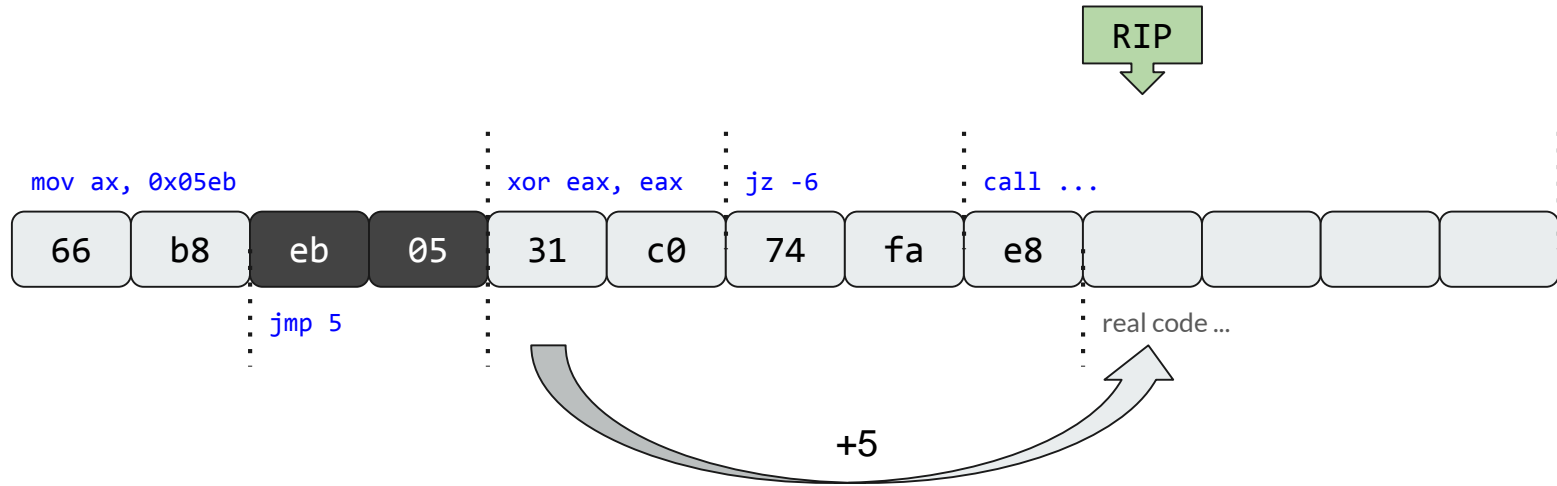
Things can get arbitrarily complex



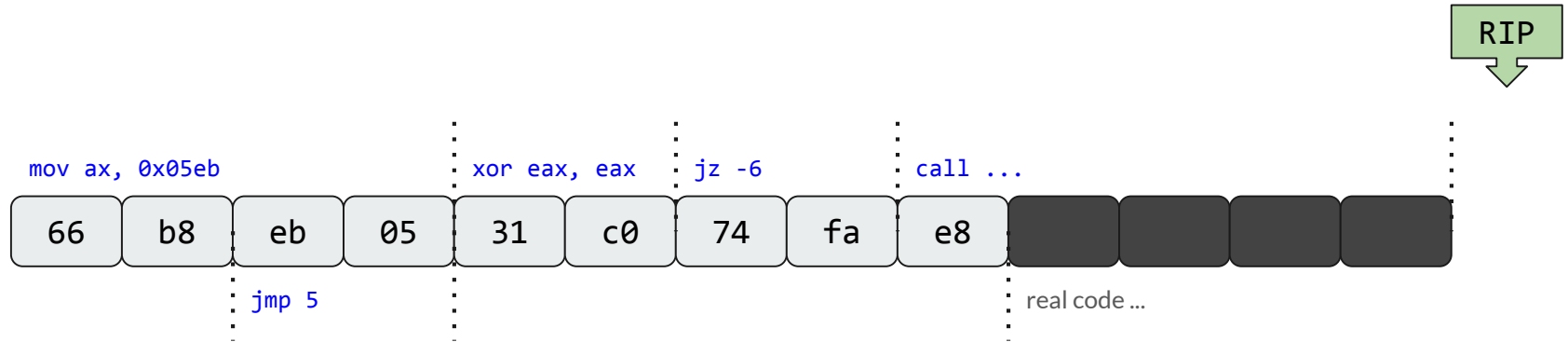
Things can get arbitrarily complex



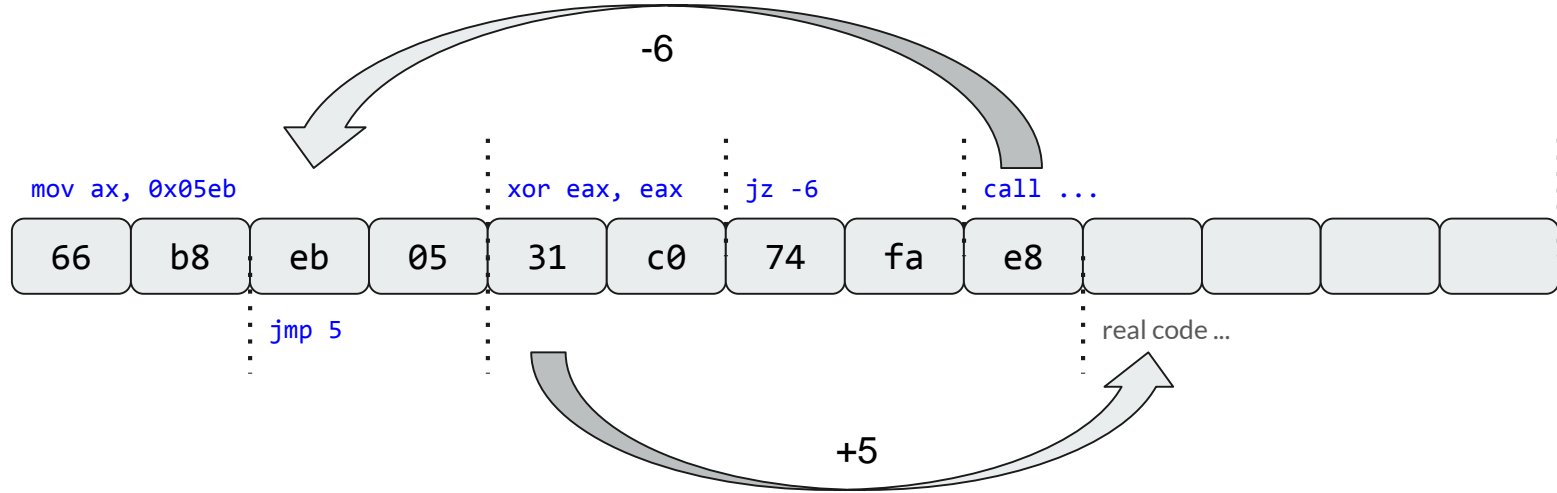
Things can get arbitrarily complex



Things can get arbitrarily complex



Things can get arbitrarily complex



It's pretty easy to code

```
global _start, start
```

```
start:  
_start:
```

```
mov ax, 0x05eb  
xor eax, eax  
db 0x74, 0xfa, 0xe8  
; hidden code begins
```

here

```
mov rdi, 22  
mov rax, 60  
syscall  
hlt
```

mov ax, 0x05eb

xor eax, eax

jz -6

call ...

jmp 5

real code ...

66

b8

eb

05

31

c0

74

fa

e8



Things can get pretty nasty

- You don't actually *have* to honor the calling conventions. You can just pretend to.
- The stack is just unprotected memory. You can write to it.
- The instruction pointer is just another register. You can modify it directly.
- Push and jump! Push and return!
- ... and lots of other stuff.

Next time:
Threaded disassembly