# X86 Debug

Computer Systems Section 3.11

# GDB is a "Source Level" debugger

- We have learned how to debug at the C level
- Now, C has been translated to X86 assembler!
- How does GDB play the shell game?
  - Makes it seem like we execute C code
  - Actually we are executing X86 Assembler Code
- How do we debug at the X86 level?

# Simple C Function

```
int myfunc(int a,int b) {
    int c;
    c = a + 3;
    c = c + b;
    return c;
}
```

# gcc -m32 -O0 -S prog.c

**prog.c**

```
2. int myfunc(int a,int b) {
3.     int c;
4.     c = a + 3;
5.     c = c + b;
6.     return c;
7. }
```

**prog.s**

```
myfunc:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        addl    $3, %eax
        movl    %eax, -4(%ebp)
        movl    12(%ebp), %eax
        addl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        ret
```

# myfunc x86 view

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

myfunc:

```
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl    8(%ebp), %eax
addl    $3, %eax
movl    %eax, -4(%ebp)
movl    12(%ebp), %eax
addl    %eax, -4(%ebp)
movl    -4(%ebp), %eax
leave
ret
```

| Memory | | |
|---|---|---|
| | xFFFF FFF0 | |
| b | xFFFF FFEC | x0000 0004 |
| a | xFFFF FFE8 | x0000 0003 |
| | xFFFF FFE4 | |
| | xFFFF FFE0 | |
| c | xFFFF FFDC | x0000 000A |
| | xFFFF FFD8 | |
| | ... | |

| Reg | Value |
|---|---|
| ebp | xFFFF FFE0 |
| eax | x0000 000A |

# Cross Reference Information

- Function Cross Reference
  - Always keep a list of function names, and the location of the first instruction in that function in the executable command
- The –g flag to the compiler tells the compiler to generate and save more cross reference information
  - For each line of code, the address of the first instruction associated with that code
  - For each variable, the location and type of the variable stored in memory
  - For each function, the parameter list

# Example Cross Reference

| | | |
|---|---|---|
| | xFFFF FFF0 | |
| b | xFFFF FFEC | x0000 0004 |
| a | xFFFF FFE8 | x0000 0003 |
| | xFFFF FFE4 | |
| | xFFFF FFE0 | |
| c | xFFFF FFDC | x0000 000A |
| | xFFFF FFD8 | |
| | ... | |
| add $0x3,%eax | x0804 839d | 83c003 |
| mov 0x8(%ebp),%eax | x0804 839a | 8b4508 |
| sub $0x10,%esp | x08048 397 | 83ec10 |
| mov %esp,%ebp | x0804 8395 | 89e5 |
| pushl %ebp | x0804 8394 | 55 |
| | | |

| Func | Location | Parms |
|---|---|---|
| myfn | x0804 839a | a,b |
| main | ... | argc,argv |

| Fn/Line | Location |
|---|---|
| myfn.4 | x0804 839a |
| myfn.5 | ... |
| myfn.6 | |

| Variable | Location | Type |
|---|---|---|
| myfn.a | xFFFF FFE8 | int |
| myfn.b | xFFFF FFEC | int |
| myfn.c | xFFFF FFDC | int |

# Breakpoint Processing

- When you set a breakpoint…
- GDB looks up address of first instruction at that breakpoint
- Replaces the beginning of the instruction with a TRAP (e.g. 1/0)
- When the trap is executed, it causes the OS to transfer to the trap handler… (in this case, gdb has registered a trap handler)
- The trap handler invokes the GDB prompt
- On continue, replace trap with correct instruction, single step, then replace trap

# Example break 4

| | | |
|---|---|---|
| | xFFFF FFF0 | |
| b | xFFFF FFEC | x0000 0004 |
| a | xFFFF FFE8 | x0000 0003 |
| | xFFFF FFE4 | |
| | xFFFF FFE0 | |
| c | xFFFF FFDC | x0000 000A |
| | xFFFF FFD8 | |
| | ... | |
| add $0x3,%eax | x0804 839d | 83c003 |
| mov 0x8(%ebp),%eax | x0804 839a | TRAP; 8b4508 |
| sub $0x10,%esp | x08048 397 | 83ec10 |
| mov %esp,%ebp | x0804 8395 | 89e5 |
| pushl %ebp | x0804 8394 | 55 |
| | | |

| Func | Location | Parms |
|---|---|---|
| myfn | x0080 483b1 | a,b |
| main | ... | x0804 839a |

| Fn/Line | Location |
|---|---|
| myfn.4 | x0804 839a |
| myfn.5 | ... |
| myfn.6 | |
| main.11 | |
| main.12 | |

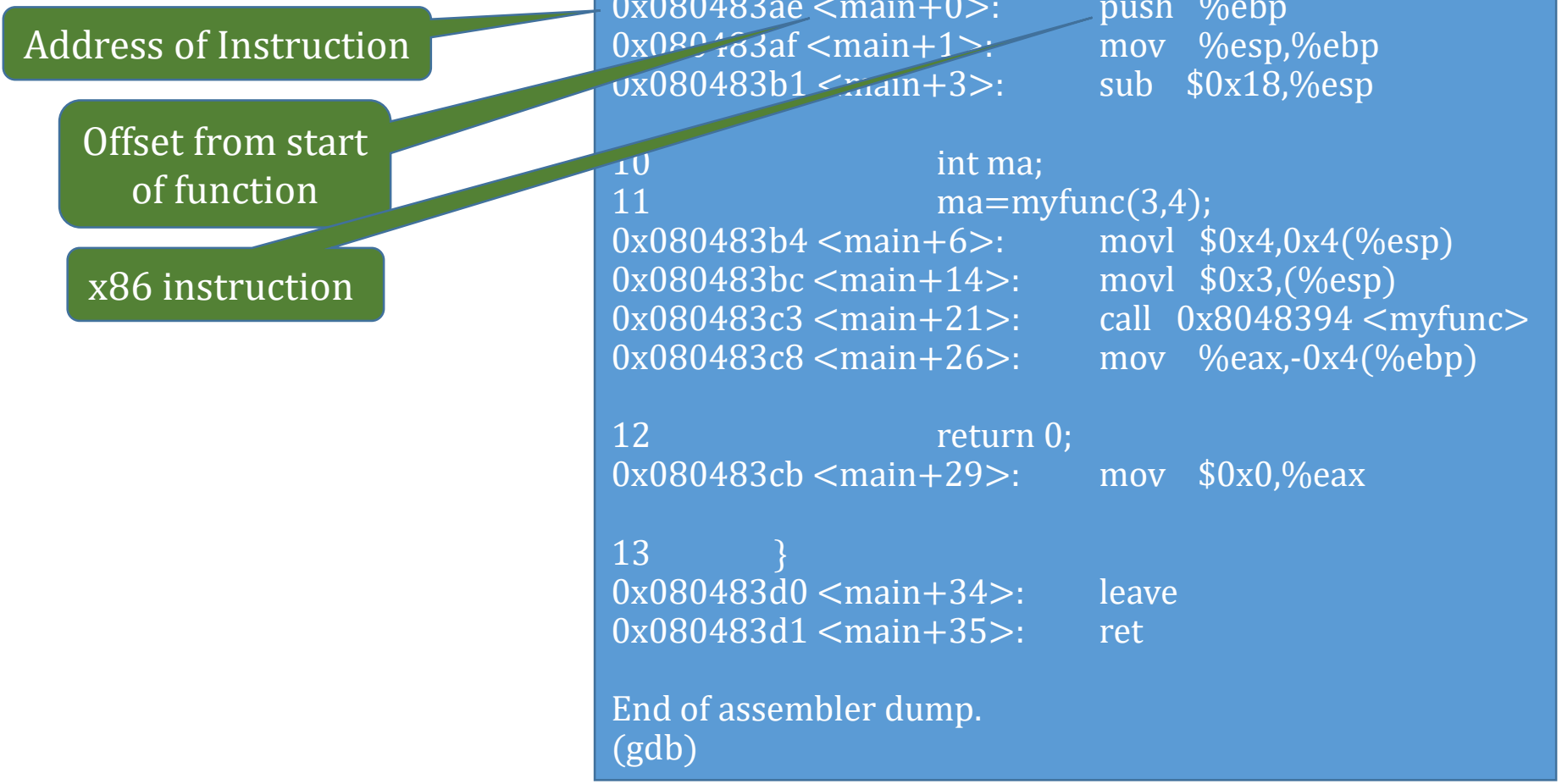
| Variable | Location | Type |
|---|---|---|
| myfn.a | xFFFF FFE8 | int |
| myfn.b | xFFFF FFEC | int |
| myfn.c | xFFFF FFDC | int |

# Displaying x86 instructions

## (gdb) disas[semble] [/m]

- prints object and x86 assembler version of current function
- /m – include C symbols/instructions when available

# Example of disassemble with debug




```
(gdb) disassemble /m
Dump of assembler code for function main:
9               int main() {
0x080483ae <main+0>:        push   %ebp
0x080483af <main+1>:        mov    %esp,%ebp
0x080483b1 <main+3>:        sub    $0x18,%esp

10                          int ma;
11                          ma=myfunc(3,4);
0x080483b4 <main+6>:        movl   $0x4,0x4(%esp)
0x080483bc <main+14>:       movl   $0x3,(%esp)
0x080483c3 <main+21>:       call   0x8048394 <myfunc>
0x080483c8 <main+26>:       mov    %eax,-0x4(%ebp)

12                          return 0;
0x080483cb <main+29>:       mov    $0x0,%eax

13              }
0x080483d0 <main+34>:       leave
0x080483d1 <main+35>:       ret

End of assembler dump.
(gdb)
```

# Example of disassemble without debug

```
(gdb) b main
Breakpoint 1 at 0x80483b4
(gdb) run
Starting program: /import/linux/home/tbarten1/CS220/examples/xmp_x86/prog

Breakpoint 1, 0x080483b4 in main ()
(gdb) disassemble
Dump of assembler code for function main:
0x080483ae <main+0>:     push   %ebp
0x080483af <main+1>:     mov    %esp,%ebp
0x080483b1 <main+3>:     sub    $0x18,%esp
0x080483b4 <main+6>:     movl   $0x4,0x4(%esp)
0x080483bc <main+14>:    movl   $0x3,(%esp)
0x080483c3 <main+21>:    call   0x8048394 <myfunc>
0x080483c8 <main+26>:    mov    %eax,-0x4(%ebp)
0x080483cb <main+29>:    mov    $0x0,%eax
0x080483d0 <main+34>:    leave
0x080483d1 <main+35>:    ret
End of assembler dump.
(gdb)
```

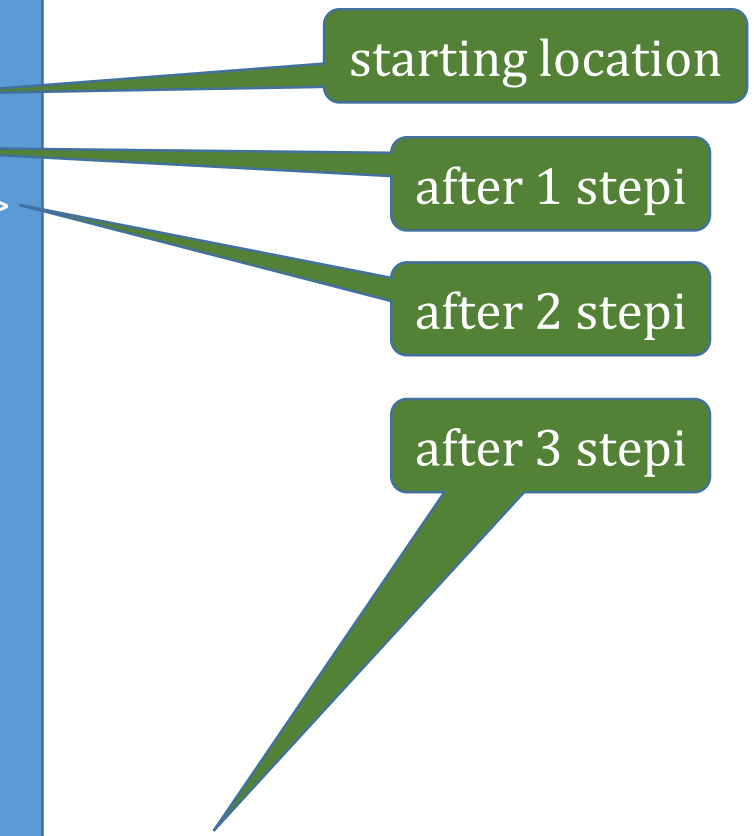
Notice… break in main AFTER function entry!

# Stepping through C/x86 code

- With debug, step executes to next (debugged) C instruction
  - If you invoke a function that was compiled without debug, skips that function!
- Without debug, step moves to next (debugged) C instruction
  - Often, OS function that invokes main!
  - Practically Useless!
- Alternatives : nexti and stepi
  - Executes to the next x86 instruction
  - nexti skips function calls
  - stepi can step into protected (invisible) code!

# Example of stepi without debug

```
Breakpoint 1, 0x080483b4 in main ()
(gdb) disas
Dump of assembler code for function main:
0x080483ae <main+0>:    push   %ebp
0x080483af <main+1>:    mov    %esp,%ebp
0x080483b1 <main+3>:    sub    $0x18,%esp
0x080483b4 <main+6>:    movl   $0x4,0x4(%esp)
0x080483bc <main+14>:   movl   $0x3,(%esp)
0x080483c3 <main+21>:   call   0x8048394 <myfunc>
0x080483c8 <main+26>:   mov    %eax,-0x4(%ebp)
0x080483cb <main+29>:   mov    $0x0,%eax
0x080483d0 <main+34>:   leave
0x080483d1 <main+35>:   ret
End of assembler dump.
(gdb) stepi
0x080483bc in main ()
(gdb) stepi
0x080483c3 in main ()
(gdb) stepi
0x08048394 in myfunc ()
(gdb)
```



# Avoid Stepping Into Protected Code

- If you do, stepi continues to work…
- But you can't see where you are
- You can't see the instructions you are executing
- You may use the "finish" command to continue this function until it returns to its caller

# Continuous x86 Assembler Print

- When I debug at the C level, gdb prints out the C instruction it is about to execute
- When I do “stepi” or “nexti”, all I get is an address…
- Until I execute:

(gdb) set disassemble-next-line on

# stepi with disassemble-next-line

```
(gdb) set disassemble-next-line on
(gdb) b main
Breakpoint 1 at 0x80483b4
(gdb) run
Starting program: /import/linux/home/tbarten1/CS220/examples/xmp_x86/prog

Breakpoint 1, 0x080483b4 in main ()
0x080483b4 <main+6>:        c7 44 24 04 04 00 00 00  movl   $0x4,0x4(%esp)
(gdb) stepi
0x080483bc in main ()
0x080483bc <main+14>:       c7 04 24 03 00 00 00     movl   $0x3,(%esp)
(gdb) stepi
0x080483c3 in main ()
0x080483c3 <main+21>:       e8 cc ff ff ff           call   0x8048394 <myfunc>
(gdb) stepi
0x08048394 in myfunc ()
0x08048394 <myfunc+0>:      55                       push   %ebp
(gdb)
```

# Breakpoints in X86

- Its no fun to step through an entire program.
- I want to set a breakpoint… but there is no line number
  - Especially if there is no debug  turned on!

- (gdb) break *<address>

- Sets a breakpoint at a specific instruction address
  - To specify a hexadecimal address, use “0x” prefix!

# break at addr

```
Dump of assembler code for function main:
0x080483ae <main+0>:    push   %ebp
0x080483af <main+1>:    mov    %esp,%ebp
0x080483b1 <main+3>:    sub    $0x18,%esp
0x080483b4 <main+6>:    movl   $0x4,0x4(%esp)
0x080483bc <main+14>:   movl   $0x3,(%esp)
0x080483c3 <main+21>:   call   0x8048394 <myfunc>
0x080483c8 <main+26>:   mov    %eax,-0x4(%ebp)
0x080483cb <main+29>:   mov    $0x0,%eax
0x080483d0 <main+34>:   leave
0x080483d1 <main+35>:   ret
End of assembler dump.
(gdb) b *080483c3
Invalid number "080483c3".
(gdb) b *x080483c3
No symbol table is loaded.  Use the "file" command.
(gdb) b *0x080483c3
Breakpoint 2 at 0x80483c3
(gdb) c
Continuing.

Breakpoint 2, 0x080483c3 in main ()
0x080483c3 <main+21>:    e8 cc ff ff ff          call   0x8048394 <myfunc>
(gdb)
```

# Register Information

- (gdb) info reg
- Displays x86 regs and values

```
(gdb) info reg
eax            0xffffdab4       -9548
ecx            0x6a0e39fb       1779317243
edx            0x1      1
ebx            0xf7fafff4       -134545420
esp            0xffffd9f0       0xffffd9f0
ebp            0xffffda08       0xffffda08
esi            0x0      0
edi            0x0      0
eip            0x80483c8        0x80483c8 <main+26>
eflags         0x286    [ PF SF IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x63     99
```

# Memory

- (gdb) x /<nfu> <address_expression>
  - eXamines memory starting at <address_expression> using format <nfu>
  - <n> - Number of values to print
  - <f> - Format:
  - <u> - Unit size
- Address expression can be
  - Constant, e.g. (gdb) x /4i 0x1004011b5
  - Register, e.g. (gdb) x /4i $eip
  - Pointer variable, e.g. (gdb) x /8cb argv[0]

| f | |
|---|---|
| x | hexadecimal |
| d | decimal |
| u | unsigned dec |
| f | floating point |
| a | address |
| c | character |
| s | string |
| i | instruction |

| u | |
|---|---|
| b | 1 |
| h | 2 |
| w | 4 |
| q | 8 |

# Don’t Forget Other Cool GDB stuff

```
(gdb)break *0x080483c3 if $eax > 13
(gdb) commands
x /4dw $esp
stepi
end
(gdb)
```