EXPERT LEVEL DEBUGGING

JAMES MITCHELL | SHG MELBOURNE

04/10/22

SLEDGE HAMMER GAMES

CALL STACK CORRUPTION

Call Stack

Name
00000000fa2a10cd()
00007ffb0000000b()
0000000000000110()

Path	Optimized	User Code	Symbol Status	Symbol File
C:\Users\jamitchell\source\repos...	N/A	Yes	Symbols loaded.	C:\Users\jamitchell\sourc...

DISCLAIMER

- The examples in this talk will be using
  - C++
  - Visual Studio
  - Windows 10
  - X86-64 architecture
- Other platforms, debuggers and languages will differ slightly.

AGENDA

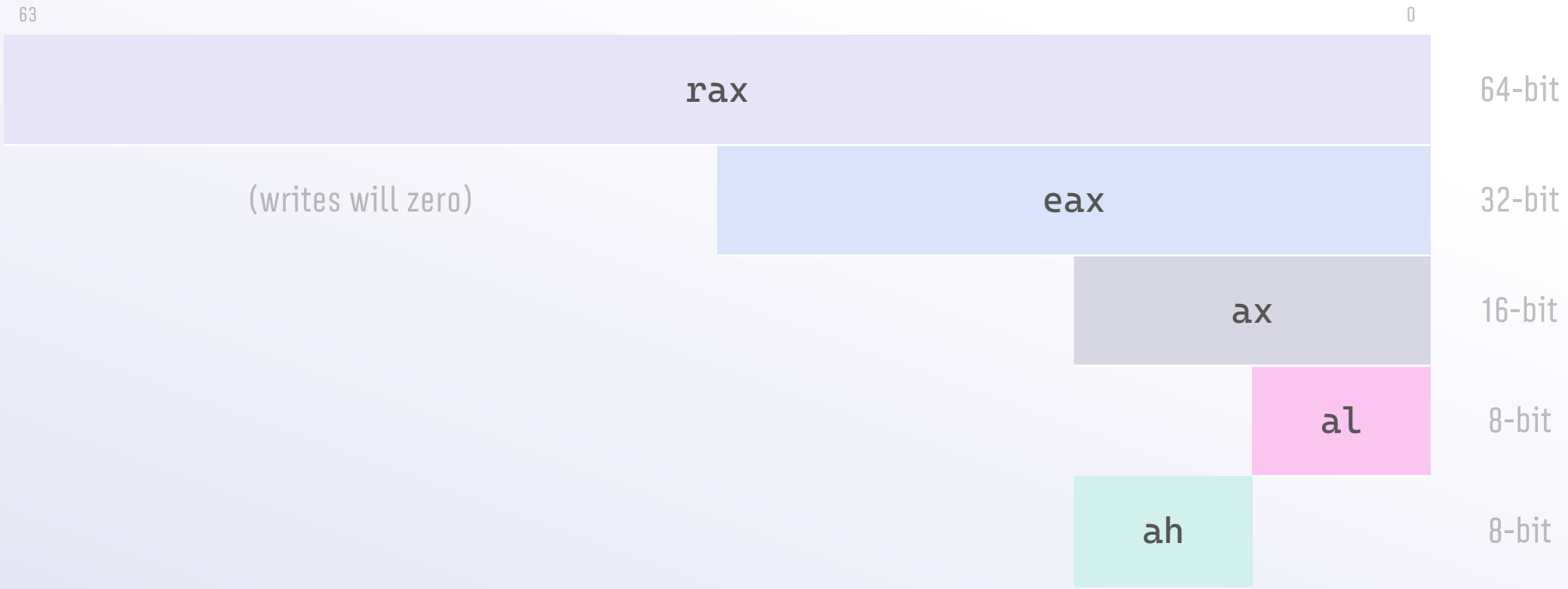
- x86-64 assembly introduction
- Demo: Call stack corruption

X86-64 ASSEMBLY INTRODUCTION

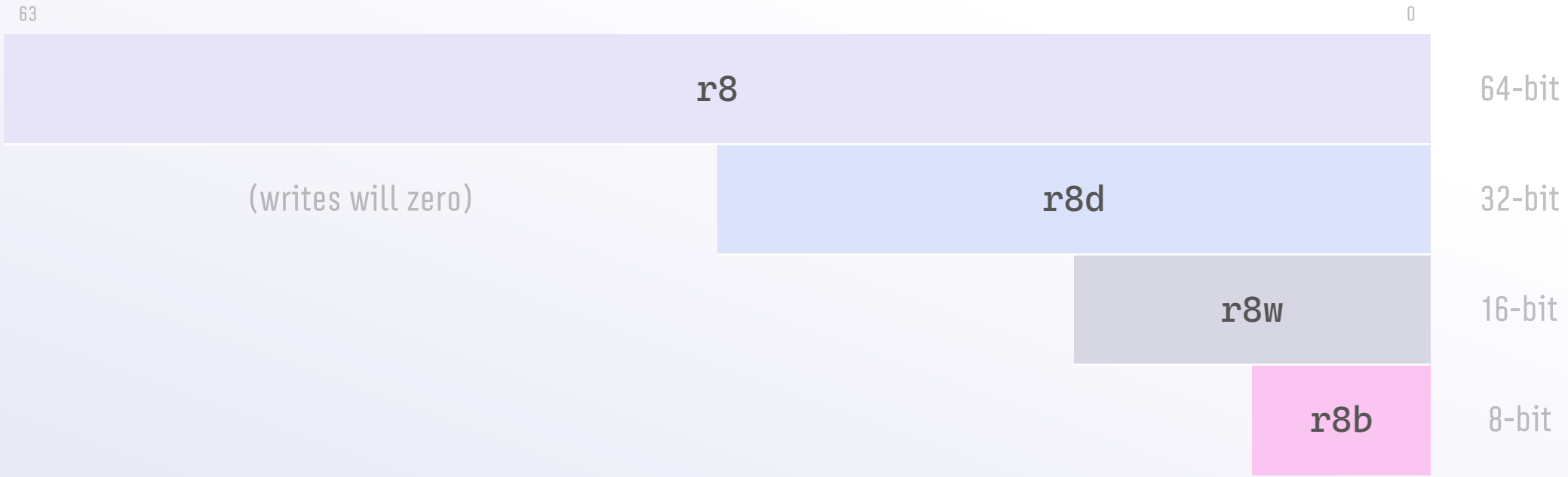
X86-64 ASSEMBLY : REGISTERS

- `rax, rbx, rcx, rdx, rsp, rbp, rsi, rdi, r8-r15` are general purpose registers
- `xmm0-xmm15` are SSE registers (used for floating point operating also)
- `rip` is the instruction pointer
- Windows calling conventions
  - `rcx, rdx, r8, r9` are arguments the rest go on the stack
  - `rax` is the return value
  - `rax, rcx, rdx, r8-r11, xmm0-xmm5` are volatile (Can be changed by the called function)
  - `rbx, rbp, rdi, rsi, rsp, r12-r15, xmm6-xmm15` are non-volatile (Cannot be changed by the called function)

X86-64 ASSEMBLY : REGISTERS



X86-64 ASSEMBLY : REGISTERS



X86-64 ASSEMBLY : INSTRUCTIONS

```
op
op dest
op dest, src
op dest, src1, src2
```

NOTE:
Intel Assembly Syntax

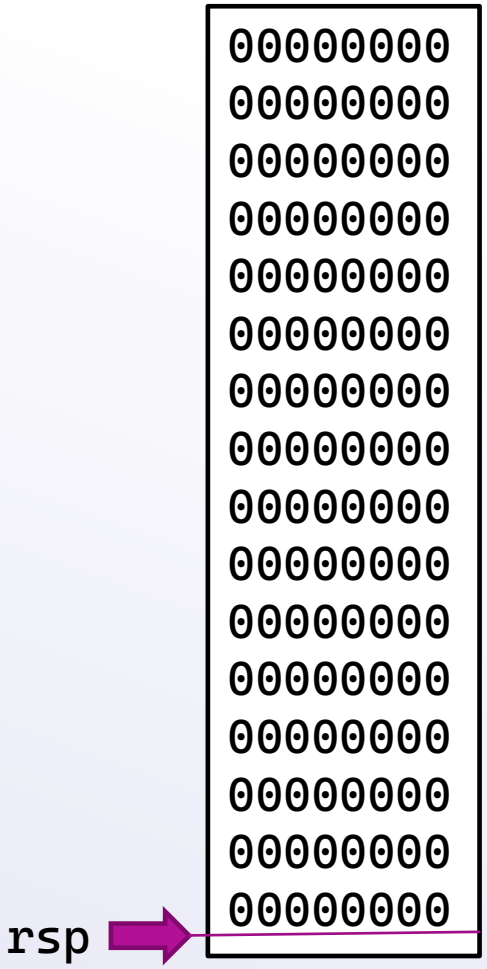
- op is an operation (e.g. call, ret, add, sub, etc)
- dest and src are operands which can be
  - Registers (e.g. rax)
  - Immediate values (e.g. 30h)
  - Memory addresses (e.g. [myvar], [rsp + 30h], [rdx + rbx * 8])
    - May be prefixed with data type sizes (e.g. DWORD PTR, QWORD PTR)

X86-64 ASSEMBLY : THE STACK

NOTE:
Using 32-bits in this example for ease of reading, 64-bit acts the same.

```
push 10h
pop rax
…
```

rax 00000000



Lower Addresses
Stack Growth

High Addresses

X86-64 ASSEMBLY : THE STACK

NOTE:
Using 32-bits in this example for ease of reading, 64-bit acts the same.

```
push 10h
pop rax
…
```

rax 00000000

00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000010

rsp

Lower Addresses
Stack Growth

High Addresses

X86-64 ASSEMBLY : THE STACK

NOTE:
Using 32-bits in this example for ease of reading, 64-bit acts the same.

```
push 10h
pop rax
…
```

rax 00000010

00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000010

rsp

Lower Addresses
Stack Growth

High Addresses

X86-64 ASSEMBLY : THE STACK

NOTE:
Using 32-bits in this example for ease of reading, 64-bit acts the same.

```
example():
; prolog - Setup stack-frame
push rbx
sub rsp, 24h
…
; epilog - Exit stack-frame
add rsp, 24h
pop rbx
ret
```

rax 00000000

00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000

rsp

Lower Addresses
Stack Growth

High Addresses

X86-64 ASSEMBLY : THE STACK

NOTE:
Using 32-bits in this example for ease of reading, 64-bit acts the same.

```
example():
…
; Write stack variables
mov dword ptr [rsp+4h], 10h

; Read stack variables
mov rax, dword ptr [rsp+4h]
```

rax 00000000

	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
rsp	00000000
	00000000
	00000000
	00000000

Lower Addresses
Stack Growth

High Addresses

X86-64 ASSEMBLY : THE STACK

NOTE:
Using 32-bits in this example for ease of reading, 64-bit acts the same.

```
example():
…
; Write stack variables
mov dword ptr [myvar], 10h

; Read stack variables
mov rax, dword ptr [myvar]
```

rax 00000000

00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000

rsp

Lower Addresses
Stack Growth

High Addresses

X86-64 ASSEMBLY : THE STACK

NOTE:
Using 32-bits in this example for ease of reading, 64-bit acts the same.

```
example():
; Call:
; 1. Pushes an address to return to
; 2. Jumps to function
7F000000 call example2 (7F000010h)
7F000004 mov rax, 10h
…

example2():
; Ret:
; 1. Pops a return address
; 2. Jumps to that function
7F000010 ret
```

rax 00000000

rip 7F000000

rsp

00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000

Lower Addresses
Stack Growth

High Addresses

X86-64 ASSEMBLY : THE STACK

NOTE:
Using 32-bits in this example for ease of reading, 64-bit acts the same.

```
example():
; Call:
; 1. Pushes an address to return to
; 2. Jumps to function
7F000000 call example2 (7F000010h)
7F000004 mov rax, 10h
…

example2():
; Ret:
; 1. Pops a return address
; 2. Jumps to that function
7F000010 ret
```

rax 00000000

rip 7F000010

rsp

00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
7F000004
00000000
00000000
00000000
00000000

Lower Addresses
Stack Growth

High Addresses

X86-64 ASSEMBLY : THE STACK

NOTE:
Using 32-bits in this example for ease of reading, 64-bit acts the same.

```
example():
; Call:
; 1. Pushes an address to return to
; 2. Jumps to function
7F000000 call example2 (7F000010h)
7F000004 mov rax, 10h
…

example2():
; Ret:
; 1. Pops a return address
; 2. Jumps to that function
7F000010 ret
```

rax `00000000`

rip `7F000004`

	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	00000000
	7F000004
rsp	00000000
	00000000
	00000000
	00000000

Lower Addresses
Stack Growth

High Addresses

X86-64 ASSEMBLY : THE STACK

- For Windows x86-64 ABI
  - The stack is always aligned at 16-byte boundaries (except inside the prolog)
  - 32-bytes reserved parameter stack or home area, after the return address
    - Callee controlled
    - Typically used to store non-volatile registers

DEMO: CALL STACK CORRUPTION

CALL STACK CORRUPTION

```
void zero_number(int index)
{
    int numbers[] = { 1,2,3,4 };

    numbers[index] = 0;

    for (int num : numbers)
    {
        std::cout << num << '\n';
    }
}
```

CALL STACK CORRUPTION

```
void zero_number(int index)
{
push        rbx
sub         rsp,40h
mov         rax,qword ptr [00007FF65E2F5000h]
xor         rax,rsp
mov         qword ptr [rsp+30h],rax
    int numbers[] = { 1,2,3,4 };
movdqa      xmm0,xmmword ptr [00007FF65E2F32E0h]

    numbers[index] = 0;

    for (int num : numbers)
lea         rbx,[rsp+20h]
movsxd      rax,ecx
movdqu      xmmword ptr [rsp+20h],xmm0
mov         dword ptr [rsp+rax*4+20h],0
nop         dword ptr [rax]
nop         word ptr [rax+rax+0000000000000000h]
```

```
    {
        std::cout << num << '\n';
mov         edx,dword ptr [rbx]
mov         rcx,qword ptr [00007FF65E2F3080h]
call        qword ptr [00007FF65E2F3090h]
mov         rcx,rax
call        00007FF65E2F1200
lea         rax,[rsp+30h]
add         rbx,4
cmp         rbx,rax
jne         00007FF65E2F1040
    }
}
mov         rcx,qword ptr [rsp+30h]
xor         rcx,rsp
call        00007FF65E2F1590
add         rsp,40h
pop         rbx
ret
```

CALL STACK CORRUPTION

```
void zero_number(int index)
{
    int numbers[] = { 1,2,3,4 };

    numbers[index] = 0;

    for (int num : numbers)
    {
        std::cout << num << '\n';
    }
}
```

Stack frame

Stack offset	Purpose	
rsp+00h	home	
rsp+08h	home	
rsp+10h	home	
rsp+18h	home	
rsp+20h	numbers[0]	numbers[1]
rsp+28h	numbers[2]	numbers[3]
rsp+30h	numbers[4]	numbers[5]
rsp+38h	numbers[6]	numbers[7]
rsp+40h	numbers[8]	numbers[9]
rsp+48h	numbers[10]	numbers[11]

CALL STACK CORRUPTION

LIVE DEMO TIME

Let's pray to the demo god's that it crashes

QUESTIONS?

THANKS

JAMES MITCHELL | SHG MELBOURNE

04/10/22

SLEDGE HAMMER GAMES