Stack and Function Calls

Kumaravel Jagasivamani
April 15, 2024

Table of Contents

Stack

- Types of storage:
  - Registers: Fast access, but limited storage
  - Stack: Temporary memory space that can be re-used for different purposes
  - Memory: Pre-defined allocation of variables in memory
- Stack: Last-In, First-Out
- Writing: push
- Reading: pop
- rsq: contains the address with the most recent stored value

Memory and Stack Example

```
Memory example:
    temp_var resq 1
    temp_var2 resq 1
…
    mov rax, 53
    mov rbx, 38
    mov [temp_var], rax
    mov [temp_var2], rbx
    mov rax, 12
    mov rbx, 183
    mov rax, [temp_var]
    mov rbx, [temp_var2]

Stack example:
    mov rax, 53
    mov rbx, 38
    push rax
    push rbx
    mov rax, 12
    mov rbx, 183
    pop rbx
    pop rax
```

Tutorial commands

- set disassembly-flavor intel
- break main
- run
- disassemble main
- break *0x(addr)
- continue
- stepi
- print/d $rax
- info registers
- x/1dg $rsp
- x/1dg $rsp+8

Function Calls

- Terminology:
  - Caller: code that is calling the function
  - Callee: function that is being called
- CALL statement
  - Pushes the return address of instruction memory into stack
- RET statement
  - Pops the return address of instruction memory from stack

Passing arguments

- Arguments through registers:
  - RDI, RSI, RDX, RCX, R8, R9
- Additional arguments through stack
- Return values through registers:
  - RAX
- Additional return values through stack

Passing function parameters example

- Function parameters:

```
uint64_t load_input_buffer(uint64_t *output_base_addr, uint64_t v1, uint64_t v2, uint64_t v3, uint64_t v4, uint64_t v5, uint64_t v6, uint64_t v7, uint64_t v8) {
```

- Passing function inputs:

```
    mov rdi, input_buffer
    mov rsi, 2
    mov rdx, 3
    mov rcx, 4
    mov r8, 5
    mov r9, 6
    mov rax, 9
    push rax
    mov rax, 8
    push rax
    mov rax, 7
    push rax
    call load_input_buffer
```

.

Using registers in function

- Callee save list:
  - RBX, RBP, R12, R13, R14, R15
- Caller save list:
  - RDI, RSI, RDX, RCX, R8, R9, RAX, R10, R11

Storing registers example

```
Caller:
    push rax
    call print_number
    pop rax

•...

print_number:
    push rbp
    mov rbp, rsp
    push rbx

•…

    pop rbx
    pop rbp
    ret
```

Temporary values in function

- Use rbp register to determine base of temporary variables
- Pre-define amount of storage needed for temporary variables
- Example:

```
    sub rsp, 16
    mov qword [rbp-16], 5
    mov qword rbx, [rbp-16]
    add rbx, 3
    mov qword [rbp-24], rbx
...
    add rsp, 16
```

-

Stack Example

Code

```
    push rax ; push RAX value into stack
    mov rdi, input_buffer
    mov rsi, 2
    mov rdx, 3
    mov rcx, 4
    mov r8, 5
    mov r9, 6
    mov rax, 9
    push rax
    mov rax, 8
    push rax
    mov rax, 7
    push rax
    call load_buffer
    mov rbx, rax ; RIP value of this instruction is stored in stack by call
    pop rax ; pop RAX value from stack


load_buffer:
    push rbp ; Store RBP value into stack
    mov rbp, rsp ; Use current value of stack pointer for rbp
    push rbx ; Store RBX value into stack
    sub rsp, 16 ; Store two temporary values (8 bytes each) onto stack
    mov qword [rbp-16], 5 ; Store 5 into the first temporary value
    mov qword rbx, [rbp-16]
    add rbx, 3
    mov qword [rbp-24], rbx ; Store 8 into the second temporary value
...
    add rsp, 16 ; Don't need temporary values in stack anymore
    pop rbx ; Restore RBX value
    pop rbp ; Restore RBP value
    ret ; Restore RIP value from stack (inst. memory addr of 'mov rbx, rax')
```

Stack example

Stack

Table 1

Physical Addr	Value	Pointers	Comments
FFE0	Don’t Modify		Value already in stack before function call
FFD8	Don’t Modify		Already in stack
FFD0	Don’t Modify		Already in stack
FFC8	Don’t Modify		Already in stack
FFC0	RAX		pushed by caller (part of caller save list)
FFB8	9		Function’s 9th Input parameter (read by callee using offset to RBP: [RBP+0x20])
FFB0	8		Function’s 8th Input parameter
FFA8	7		Function’s 7th Input parameter
FFA0	RIP		RIP value of the instruction after CALL (pushed automatically by CALL inst.)
FF98	FFE0	rbp	Store Old RBP value. New RBP updated to RSP through mov.
FF90	RBX		Pushed by Callee (part of callee save list)
FF88	5		Callee temporary value (written/read using offset to RBP: [RBP-0x10])
FF80	8	rsp	Callee 2nd temporary value. RSP manually updated by callee.

Memory Access Timing

- Timestamp before and after code sections
  - Accessing memory slower than accessing register

Fibonacci

Pseudo-code

- Function Inputs:
    - Number of fibonacci values to be calculated: length
    - Base of Memory location of where the values should be stored: base_ptr
- Function outputs:
    - 1,1,2,3,5,8,13,…
    - Initialize Rdi to base_ptr
    - Write 1 to [rdi]. Increment rdi
    - Write 1 to [rdi]. Increment rdi
    - Initialize loop_index to 2
    - Loop:
        - Previous value at rdi-8
        - 2nd previous value at rdi-16
        - Mov rax, [rdi-8]
        - Add rax, [rdi-16]
        - Mov [rdi], rax
        - Increment rdi… Loop_index inc, compare with length and jump if not equal

Fibonacci

Assembly code

```
    mov qword [samples_cnt], 10
    mov rdi, o
    mov qword rsi, [samples_cnt]
    call fibonnaci
    mov rdi, o
    call print_values
    jmp end_program

fibonnaci:
    mov qword [rdi], 1
    add qword rdi, 8
    mov qword [rdi], 1
    add qword rdi, 8

    mov qword [loop_index], 2
fib_loop:
    mov rax, [rdi-8]
    add rax, [rdi-16]
     mov [rdi], rax
    add qword rdi, 8

    inc qword [loop_index]
    cmp qword rsi, [loop_index]
    jne fib_loop

    ret
```

Fibonacci

Questions

- What is the value of rdi equal to when RIP is pointing at fibonacci label? Provide either numerical value or register name or offset to a register.
- What is the value of rsi equal to when RIP is pointing at fibonacci label? Provide either numerical value or register name or offset to a register.
- What is the value of rdi equal to when RIP is pointing at fib_loop label for the 1st iteration? Provide either numerical value or register name or offset to a register.
- How many times is the fib_loop executed?
- How many values are written into rdi inside the fibonacci function?
- How is the value written to [rdi] inside the fib_loop code section (purple colored line) related to values already in memory?
- How are values passed into the fibonacci function?
- Does the fibonacci function update any output registers? If so, what register is updated?
- What are the contents of stack as the function is called and returned?

Fibonacci

Output

```
[kumaran@linux2 ~/cmsc313] cmsc313_fib
1
1
2
3
5
8
13
21
34
55
•
```

Fibonacci

Answers

- What is the value of rdi equal to when RIP is pointing at fibonacci label? Provide either numerical value or register name or offset to a register. rdi is equal to o.
- What is the value of rsi equal to when RIP is pointing at fibonacci label? Provide either numerical value or register name or offset to a register. rsi is equal to 10.
- What is the value of rdi equal to when RIP is pointing at fib_loop label for the 1st iteration? Provide either numerical value or register name or offset to a register. rdi is equal to o+16.
- How many times is the fib_loop executed? 8
- How many values are written into rdi inside the fibonacci function? 10
- How is the value written to [rdi] inside the fib_loop code section (purple colored line) related to values already in memory? The value written in a loop iteration is the sum of the previous 2 values in memory.
- How are values passed into the fibonacci function? rsi and rdi
- Does the fibonacci function update any output registers? If so, what register is updated? No

Fibonacci
gdb Questions

- **What are the values printed out?**
- b fib_loop
- r
- c
- c
- c
- print (uint64_t)loop_index
- stepi
- stepi
- stepi
- print (uint64_t)*$rdi

Fibonacci

gdb Answers

- loop_index: 5
- *rdi: 8