Assembly Language Programming II

x86-64 Architecture

CSCI 2050U - Computer Architecture

Randy J. Fortier
@randy_fortier

OntarioTech UNIVERSITY

Outline

- System V ABI calling convention
- Basic input and output
  - Using the c library (aka libc)

System V ABI Calling Convention

CSCI 2050U - Computer Architecture

OntarioTech UNIVERSITY

Passing Arguments via Registers

- System V AMD64 ABI (application binary interface):
  - A calling convention used by Linux, MacOS, BSD
  - For integer or pointer arguments:
    - `rdi` - first integer/pointer argument
    - `rsi` - second integer/pointer argument
    - `rdx` - third integer/pointer argument
    - `rcx` - fourth integer/pointer argument
    - `r8` - fifth integer/pointer argument
    - `r9` - sixth integer/pointer argument
  - All remaining arguments are passed via the stack

https://web.archive.org/web/20160801075139/http://www.x86-64.org/documentation/abi.pdf

Passing Arguments via the Stack

- System V AMD64 ABI:
  - Recall that rsp points to the top of the stack
  - When calling a function
    - `[rsp]` contains the return address
    - `[rsp+8]` contains the first stack parameter

https://web.archive.org/web/20160801075139/http://www.x86-64.org/documentation/abi.pdf

Return Values

- System V AMD64 ABI:
  - Integers are returned in `rax` or `rdx:rax`, and floating point values are returned in `xmm0` or `xmm1:xmm0`

Stack Alignment

- System V AMD64 ABI mandates that, during function calls, the stack address must be a multiple of 16
  - This is called stack alignment
  - Many other operating systems have the same rule
  - This means that we may often have to put padding data onto the stack when we call a function to ensure that this is true
    - We'll see this in our examples, today

Basic Input and Output

CSCI 2050U - Computer Architecture

Basic Input and Output

- Every operating system has a set of system calls (aka *syscalls*) that perform input and output
  - These are different from operating system to operating system
- In order to make code that will be easier to move to another operating system, we'll use a higher-level library called libc
  - Lib C is so named because it is literally the standard library for the C language
  - It is simple enough to learn to use, yet higher level enough to abstract away the operating system-specific details

Basic Output - `printf`

- In C, the most basic way to print a string output is using `printf`
  - `printf` is short for *print formatted*
- The first argument to `printf` is a format string which contains a number of typed placeholders
  - Each remaining argument corresponds to one of these placeholders (normally, in the same order that they are in the string)
- Some common types:
  - `%d` - decimal (i.e. integer)
  - `%f` - floating point
  - `%s` - string

Basic Output - `printf`

- Print a single integer (in C):

```
int age = 19;
printf("%d", age);
```

Basic Output - `printf`

- Print a single integer (in x86-64 assembly):

```
section .text
  mov rdi, resultFormat
  mov rsi, [age]
  mov rax, 0
  push rbx        ; must align the stack to a multiple of 16 bytes
  call printf     ; call adds 8 bytes/64 bits to the stack
  pop rbx                  ; remove this unnecessary stuff off of the stack

section .data
  age dq 19
  resultFormat db "%d", 0
```

Basic Output - `printf`

- Print two integer numbers, separated by a comma, ending with a newline (in C):

```
int x = 3;
int y = 2;
printf("%d,%d\n", x, y);
```

Basic Output - `printf`

- Print two integer numbers, separated by a comma, ending with a newline (in x86-64 assembly):

```
section .text
  mov rdi, format
  mov rsi, [x]    ; note the [], which means the value at this address (x is an address)
  mov rdx, [y]
  mov rax, 0
  push rbx
  call printf
  pop rbx
section .data
  x dq 3
  y dq 2
  format db "%d,%d", 0ah, 0dh, 0
```

Basic Input - `scanf`

- In C, the most basic way to read input is using `scanf`
  - `scanf` is short for *scan formatted*
- The first argument to `scanf` is a format string which contains a number of typed placeholders
  - Each remaining argument corresponds to one of these placeholders (normally, in the same order that they are in the string), an address where to put the value
- The types are the same ones used by `printf`

Basic Input - `scanf`

- Read a single integer (in C):

```
int age = 0;
scanf("%d", &age);
```

Basic Input - `scanf`

- Read a single integer (in x86-64 assembly):

```
section .text
  mov rdi, format
  mov rsi, age   ; note the lack of [], since we want the address
  mov rax, 0
  push rbx
  call scanf
  pop rbx

section .data
  age dq 0  ; initial value
  format db "%d", 0
```

Basic Input - `scanf`

- Read two integer numbers (in C):

```
int x;
int y;
scanf("%d%d", &x, &y);
```

Basic Input - `scanf`

- Read two integer numbers (in x86-64 assembly):

```
section .text
  mov rdi, format
  mov rsi, x
  mov rdx, y
  mov rax, 0
  push rbx
  call scanf
  pop rbx
section .data
  x dq 0
  y dq 0
  format db "%d%d", 0
```

Wrap Up

- System V ABI calling convention
- Basic input and output
  - Using the c library (aka libc)

What is next?

- Moving data between registers and memory
- Arithmetic operations
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Shift
  - Rotation