Machine Level Programming: Procedures

Computer Systems Organization (Spring 2016)
CSCI-UA 201, Section 2

Instructor: Joanna Klukowska

Slides adapted from
Randal E. Bryant and David R. O'Hallaron (CMU)
Mohamed Zahran (NYU)

Procedures
- Passing control
  - To beginning of procedure code
  - Back to return point
- Passing data
  - Procedure arguments
  - Return value
- Memory management
  - Allocate during procedure execution
  - Deallocate upon return
- Mechanisms all implemented with machine instructions
- x86-64 implementation of a procedure uses only those mechanisms required

x86-64 Stack
- Region of memory managed with stack discipline
- Grows toward lower addresses
- Register %rsp contains lowest stack address (i.e., the address of “top” element)

x86-64: push
- pushq Src
  - Fetch operand at Src
  - Decrement %rsp by 8
  - Write operand at address given by %rsp

x86-64: pop
- popq Dest
  - Read value at address given by %rsp
  - Increment %rsp by 8
  - Fetch operand at Dest (must be register)
Passing Control

Procedure Control Flow

- Use stack to support procedure call and return
- **Procedure call**: call label
  - Push return address on stack
  - Jump to label
- **Return address**:
  - Address of the next instruction right after call
  - Example from disassembly
- **Procedure return**: ret
  - Pop address from stack
  - Jump to address

Control Flow Example

Control Flow Example

Control Flow Example
Control Flow Example

Passing Data

Passing arguments and returning values

Procedure arguments:

- **Registers**
  - First six integer-pointer arguments are placed in registers: rdi, rsi, rdx, rbx, rcx, r8, r9
  - Note: you have to remember the order because that's how the arguments are mapped

- **Stack**
  - 7+ arguments (integer and pointer) saved on the stack
  - (In IA-32 all arguments were saved on the stack - accessing stack is slower than accessing the registers)

Return value:

- Register rax is used to transfer a

Example: Passing Data

Stack-Based Languages

- Languages that support recursion
  - e.g., C, Pascal, Java
  - Code must be "Reentrant"
  - Multiple simultaneous instantiations of single procedure
  - Need some place to store state of each instantiation
    - Arguments
    - Local variables
    - Return pointer

- Stack discipline
  - State for given procedure needed for limited time
    - From when called to when return
    - Call stack returns before caller does

- Stack allocated in Frames
  - State for single procedure instantiation

Local Data
Example: Function Call Chain

Procedure `aml()` is recursive

Stack Frames

- **Contents**
  - Return information
  - Local storage (if needed)
  - Temporary space (if needed)

- **Management**
  - Space allocated when enter procedure
    - “Set-up” code
    - Includes push by call instruction
  - Duplicated when return
    - “Finess” code
    - Includes pop by ret instruction

Stack

Previous Frame
Frame Pointer: `%bp`
(Optional)
Frame for procedure
Stack Pointer: `%rsp`
Stack “Top”
X86-64 Stack Frame

- Current Stack Frame ("Top" to Bottom)
  - "Argument build."
    - Parameters for function about to call
    - Local variables
    - If can’t keep in registers
    - Saved register context
    - Old frame pointer (optional)

- Caller Stack Frame
  - Return address
    - Pushed by call instruction
  - Arguments for this call

Examples
**incr function**

```c
long incr(long *p, long val) {
    long x = *p;
    long y = x + val;
    *p = y;
    return x;
}
```

**Calling incr function**

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(v1, 3000);
    return v1+v2;
}
```

**Stack Structure**

```
...  Rtn address  %gps + 8
15213 Unused
```

**Registe

<table>
<thead>
<tr>
<th>Register</th>
<th>Use(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>4v1</td>
</tr>
<tr>
<td>%rsi</td>
<td>3000</td>
</tr>
</tbody>
</table>

**Resulting Stack Structure**

```
...  Rtn address  %gps + 8
15213 Unused
```

**Updated Stack Structure**

```
...  Rtn address  %gps + 8
18213 Unused
```

**Calling incr function**

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(v1, 3000);
    return v1+v2;
}
```

**Registe

<table>
<thead>
<tr>
<th>Register</th>
<th>Use(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>4v1</td>
</tr>
<tr>
<td>%rsi</td>
<td>3000</td>
</tr>
</tbody>
</table>

**Final Stack Structure**

```
...  Rtn address  %gps + 8
```

**Calling incr function**

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(v1, 3000);
    return v1+v2;
}
```

**Registe

<table>
<thead>
<tr>
<th>Register</th>
<th>Use(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>4v1</td>
</tr>
<tr>
<td>%rsi</td>
<td>3000</td>
</tr>
</tbody>
</table>

**Final Stack Structure**

```
...  Rtn address  %gps + 8
```
Register Saving Conventions

- When procedure `you` calls `who`:
  - `you` is the caller
  - `who` is the callee
- Can register be used for temporary storage?
- Conventions
  - “Caller Saved” - Caller saves temporary values in its frame before the call
  - “Called Saved” - Caller saves temporary values in its frame before using (Caller restores temp before returning to caller)

```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

call_incr2:
```
pushq %rbx
subq $16, %rsp
movq %rdi, %rbx
movq $15213, %rax
movl $3000, %esi
leaq @(%rsp), %rdi
call incr
addq %rax, %rax
addq $16, %rsp
popq %rbx
ret
```

Initial Stack Structure

```
<table>
<thead>
<tr>
<th>...</th>
<th>Rtn address</th>
<th>%rsp</th>
</tr>
</thead>
</table>
```

Resulting Stack Structure

```
| ... | Rtn address | %rbp |
```

```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

call_incr2:
```
pushq %rbx
subq $16, %rsp
movq %rdi, %rbx
movq $15213, %rax
movl $3000, %esi
leaq @(%rsp), %rdi
call incr
addq %rbx, %rax
addq $16, %rsp
popq %rbx
ret
```

```
<table>
<thead>
<tr>
<th>...</th>
<th>Rtn address</th>
<th>%rsp</th>
</tr>
</thead>
</table>
```

```
| 15213 | Saved %rax |
```

```
| ... | Rtn address | %rsp |
```

```
| 15213 | Used        |
```

```
| ... | Rtn address | %rsp |
```

```
| ... | Rtn address | %rsp |
```

```
| ... | Saved %rbx |
```

```
| ... | Rtn address | %rbp |
```