Machine-Level Programming IV: Procedures - recursion

These slides adapted from materials provided by the textbook authors.

Recursive Function

```c
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
            + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl      $0, %eax
  testq     %rdi, %rdi
  je        .L6
  pushq     %rbx
  movq      %rdi, %rbx
  andl      $1, %ebx
  shrq      %rdi
  call      pcount_r
  addq      %rbx, %rax
  popq      %rbx
.L6:
  rep; ret
```

Recursive Function Terminal Case

```c
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl      $0, %eax
  testq     %rdi, %rdi
  je        .L6
  pushq     %rbx
  movq      %rdi, %rbx
  andl      $1, %ebx
  shrq      %rdi
  call      pcount_r
  addq      %rbx, %rax
  popq      %rbx
.L6:
  rep; ret
```

Register	Use(s)	Type
`%rdi`	`x`	Argument
`%rax`	Return value	Return value

Why rep;ret? http://repzret.org/p/repzret/

Recursive Function Register Save

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
          + pcount_r(x >> 1);
}
```

```
pcount_r:
    movl      $0, %eax
    testq     %rdi, %rdi
    je        .L6
    pushq     %rbx
    movq      %rdi, %rbx
    andl      $1, %ebx
    shrq      %rdi
    call      pcount_r
    addq      %rbx, %rax
    popq      %rbx
.L6:
    rep; ret
```
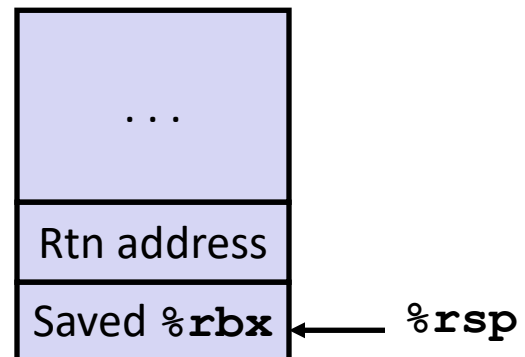
Register	Use(s)	Type
%rdi	x	Argument

```
   . . .
Rtn address
Saved %rbx    ◄—— %rsp
```

Recursive Function Call Setup

```c
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl      $0, %eax
  testq     %rdi, %rdi
  je        .L6
  pushq     %rbx
  movq      %rdi, %rbx
  andl      $1, %ebx
  shrq      %rdi
  call      pcount_r
  addq      %rbx, %rax
  popq      %rbx
.L6:
  rep; ret
```

Register	Use(s)	Type
%rdi	x >> 1	Rec. argument
%rbx	x & 1	Callee-saved

Recursive Function Call

```c
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl    $0, %eax
  testq   %rdi, %rdi
  je      .L6
  pushq   %rbx
  movq    %rdi, %rbx
  andl    $1, %ebx
  shrq    %rdi
  call    pcount_r
  addq    %rbx, %rax
  popq    %rbx
.L6:
  rep; ret
```

Register	Use(s)	Type
`%rbx`	`x & 1`	Callee-saved
`%rax`	Recursive call return value	

Recursive Function Result

```c
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
          + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl     $0, %eax
  testq    %rdi, %rdi
  je       .L6
  pushq    %rbx
  movq     %rdi, %rbx
  andl     $1, %ebx
  shrq     %rdi
  call     pcount_r
  addq     %rbx, %rax
  popq     %rbx
.L6:
  rep; ret
```

Register	Use(s)	Type
%rbx	x & 1	Callee-saved
%rax	Return value	

Recursive Function Completion

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
          + pcount_r(x >> 1);
}
```

```
pcount_r:
    movl     $0, %eax
    testq    %rdi, %rdi
    je       .L6
    pushq    %rbx
    movq     %rdi, %rbx
    andl     $1, %ebx
    shrq     %rdi
    call     pcount_r
    addq     %rbx, %rax
    popq     %rbx
.L6:
    rep; ret
```
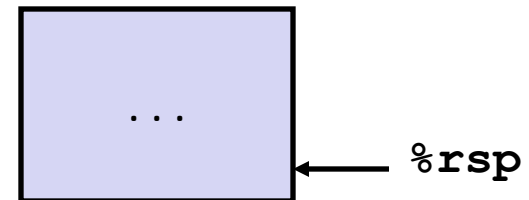
Register	Use(s)	Type
%rax	Return value	Return value

...

%rsp

Observations About Recursion

- **Handled Without Special Consideration**
    - Stack frames mean that each function call has private storage
        - Saved registers & local variables
        - Saved return pointer
    - Register saving conventions prevent one function call from corrupting another's data
        - Unless the C code explicitly does so (e.g., buffer overflow)
    - Stack discipline follows call / return pattern
        - If P calls Q, then Q returns before P
        - Last-In, First-Out
- **Also works for mutual recursion**
    - P calls Q; Q calls P

x86-64 Procedure Summary

- **Important Points**
    - Stack is the right data structure for procedure call / return
        - If P calls Q, then Q returns before P
- **Recursion (& mutual recursion) handled by normal calling conventions**
    - Can safely store values in local stack frame and in callee-saved registers
    - Put function arguments at top of stack
    - Result return in `%rax`
- Pointers are addresses of values
    - On stack or global

Caller Frame
Arguments 7+
Return Addr

`%rbp` → Old %rbp
(Optional)

Saved Registers + Local Variables
Argument Build

`%rsp` →