Computer Architecture
Machine Representation of Programs: Procedures 2

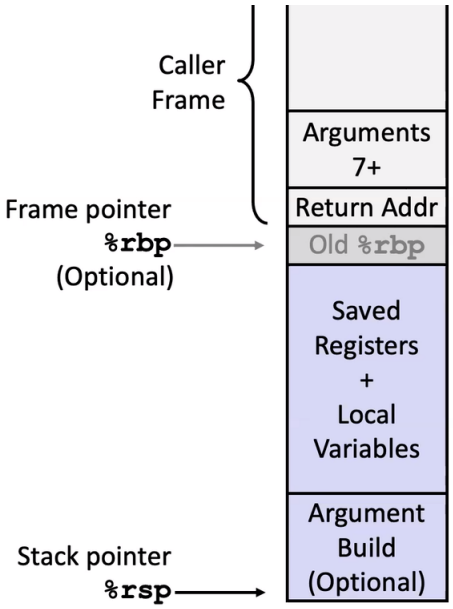
Cain Susko

Queen's University
School of Computing

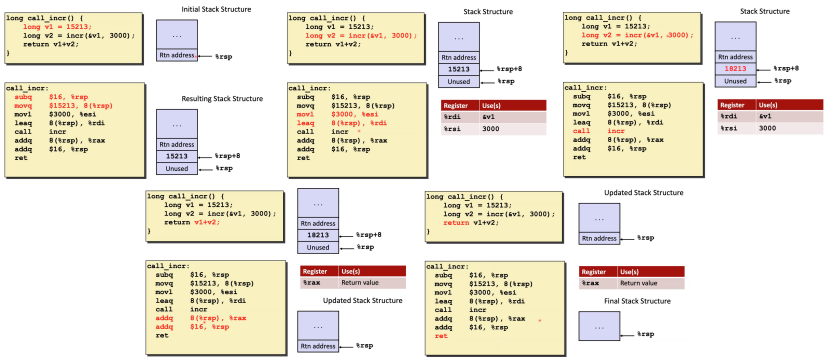
February 16, 2022

Recall

recall the stack frame for x86-64 linux systems:



Additionally, recall the example `incr`:

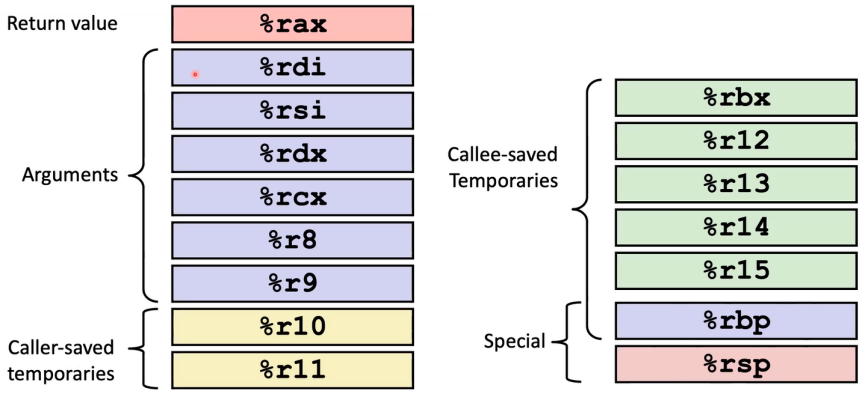


Register Saving Conventions

There are 2 main conventions when saving registers:

Caller Saved	caller saves temp vals in frame before call
Callee Saved	callee saves temp vals in frame before using them callee then restores vals before returning to caller.

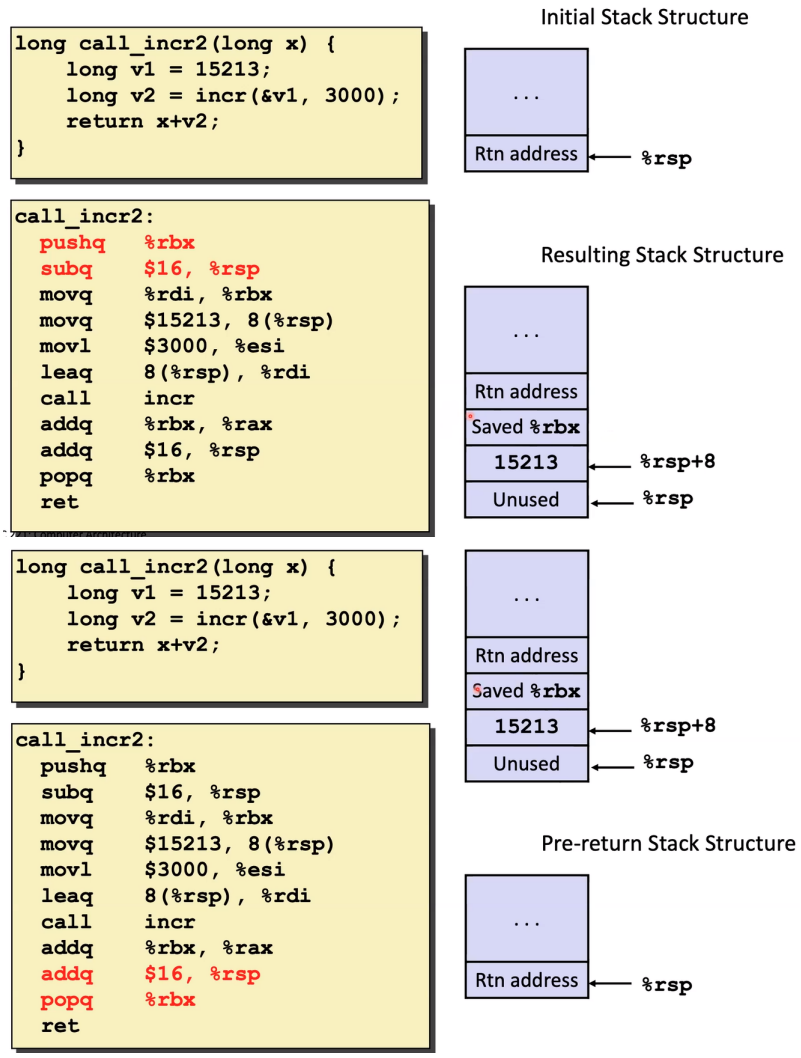
These conventions are defined within the architecture its self.
Within a Linux x86-64 system, the registers are used like so:



Where:

- `%rax` is caller saved and can be modified by procedure
- `%rdi,..., %r9` are also caller saved but can be modified by the callee procedure.
- `%r10, %r11` are caller saved and can be modified by callee procedure.
- `%rbx,...,%r15` is callee saved. The callee must save and restore the data (push and pop to stack)
- `%rbp` is callee saved and the callee must save and restore the register. Additionally, it can be used as a frame pointer.
- `%rsp` special form of callee save where it is restored to original value upon exit from procedure

For Example:


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, 8(%rsp)
movl $3000, %esi
leaq 8(%rsp), %rdi
call incr
addq %rbx, %rax
addq $16, %rsp
popq %rbx
ret
Initial Stack Structure
...
Rtn address
%rsp
Resulting Stack Structure
...
Rtn address
Saved %rbx
15213
Unused
%rsp+8
%rsp
Pre-return Stack Structure

Recursive Functions & the Stack

Finally, the behaviour that a recursive function has within this structure of registers, stacks, and frames is shown by the following example:

```
/* 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 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

Figure 1: terminal case

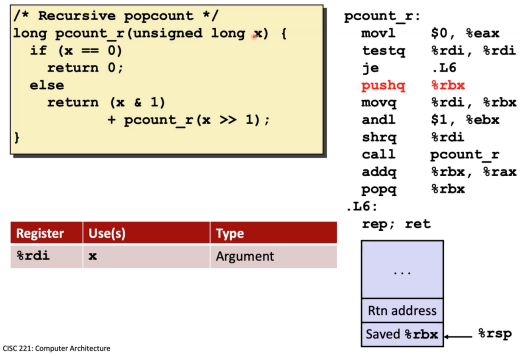


Figure 2: register save

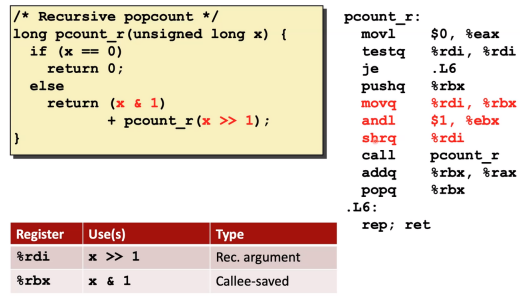


Figure 3: call setup

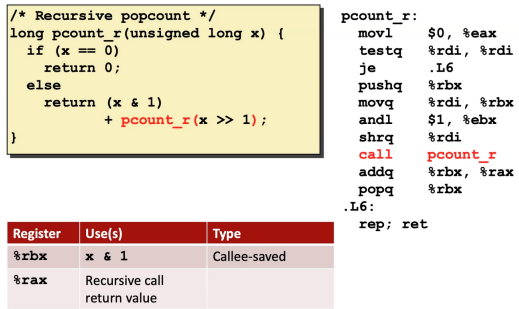


Figure 4: call

```
/* 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	

Figure 5: result

```
/* 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

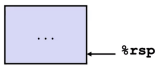


Figure 6: complete

Observations about Recursion

- Handled without special consideration
  - stack frames mean that each function call has private storage
    - saved registers and local variables
    - saved return pointer
  - register saving conventions prevent one function call from corrupting another's data
    - unless the C code explicitly does so (ie. programmer error)
  - stack discipline follows call / return pattern
    - if P calls Q, then Q returns before P
    - Last In First Out
  - also works with mutual recursion (ie. P calls Q; Q calls P)

x86 Procedure Summary

- The stack is the right data structure for procedure call and return as it has all the safety needed to ensure consistent computations.
- recursion and mutual recursion are handled with normal calling conventions.
- `%rax` always returns result
- pointers are addresses of values on the stack or globally.