CMPT 295
Machine-Level Programming
Lecture 18 – Recursion
Managing local data in registers and on stack

x86-64 “register saving” convention:

- **callee saved** registers: `%rbx`, `%rsp`, `%rbp`, `%r12` to `%r15`
- **caller saved** registers: `%r10`, `%r11`, `%rax` and all 6 registers used for arguments (data passing)
Today’s Menu

- Introduction
  - C program -> assembly code -> machine level code
  - Assembly language basics: data, move operation
    - Memory addressing modes
  - Operation leaq and Arithmetic & logical operations
  - Conditional Statement – Condition Code + cmovX
- Loops
- Function call – Stack – Recursion
  - Overview of Function Call
  - Memory Layout and Stack - x86-64 instructions and registers
  - Passing control
  - Passing data – Calling Conventions
  - Managing local data
  - Recursion
- Array
- Floating-point operations
Recursion in x86-64

- Handled without special consideration
  - Stack discipline (LIFO) follows call / return pattern
    - If P (first invocation) calls itself (second invocation of P), then the second invocation of P must terminate before the first invocation of P can do so
  - Stack frame mean that each function call has private storage
    - Saved registers & local variables
    - Argument 7..n (if any)
    - Saved return address
  - Function call conventions
  - Register saving conventions -> prevent one function from corrupting another’s data (stored in registers)

- Also works for mutual recursion
  - P calls Q then Q calls P
Recursive Function – \texttt{countOnesR( \ldots )}

\begin{verbatim}
/* Recursive counter of 1's */
long countOnesR(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1) + countOnesR(x >> 1);
}
\end{verbatim}

What does this function do?
Recursive Function – Example – Base Case

/* Recursive counter of 1’s */
long countOnesR(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1) + countOnesR(x >> 1);
}

countOnesR:
    xorl %eax, %eax
    testq %rdi, %rdi
    je done
    pushq %rbx
    movq %rdi, %rbx
    andl $1, %ebx
    shrq %rdi
    call countOnesR
    addq %rbx, %rax
    popq %rbx
done:
    ret
Recursive Function – Example - Saving registers

/* Recursive counter of 1’s */
long countOnesR(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1) + countOnesR(x >> 1);
}

countOnesR:
    xorl %eax, %eax
    testq %rdi, %rdi
    je done
    pushq %rbx
    movq %rdi, %rbx
    andl $1, %ebx
    shrq %rdi
    call countOnesR
    addq %rbx, %rax
    popq %rbx
done:
    ret
/* Recursive counter of 1’s */
long countOnesR(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1) + countOnesR(x >> 1);
}
/* Recursive counter of 1's */
long countOnesR(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1) + countOnesR(x >> 1);
}
/* Recursive counter of 1’s */
long countOnesR(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1) + countOnesR(x >> 1);
}
Recursive Function – Example – Clean-up and return

```c
/* Recursive counter of 1's */
long countOnesR(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1) + countOnesR(x >> 1);
}
```

countOnesR:
- xorl %eax, %eax
- testq %rdi, %rdi
- je done
- pushq %rbx
- movq %rdi, %rbx
- andl $1, %ebx
- shrq %rdi
- call countOnesR
- addq %rbx, %rax
- popq %rbx

```assembly
done:
    ret
```
Recursive Function – Example – Test Cases

- Test Case 1
  - Input: $x = 0$
  - Expected result: 0

- Test Case 2
  - Input: $x = 7$
  - Expected result: 3
<table>
<thead>
<tr>
<th>base + displacement</th>
<th>Stack Variables</th>
<th>Purpose</th>
</tr>
</thead>
</table>

Register Table:
Summary

- Recursion
  - Handled without special consideration using ...
    - Stack frames
    - x86-64 Function call and Register saving conventions
Next lecture

- Introduction - C program -> assembly code -> machine level code
- Assembly language basics: data, move operation, Memory addressing modes
- Operations -> Arithmetic & logical operations
- Continue with Conditional Statement
- Loop -> demo + MT #1 Review
- Function call
  - Overview of Function Call
  - Memory Layout and Stack - x86-64 instructions and registers
  - Passing control
  - Passing data – Calling Conventions
  - Managing local data
  - Recursion
- Array
- Floating-point operations