Call and Return Sequence
Calling Convention

- A standardized method for a program to pass parameters to a function and receive a result value back from it
- Where they place parameters and return values
  - In registers
  - On the call stack
  - A mix of both
- The order in which parameters are passed
- How responsibility for setting up and cleaning up a function call is distributed between the caller and callee
SNUVM#1 Calling Convention

- **r0-r3**
  - Argument, scratch
  - `r0` contains the result of callee
  - The calling routine saves their contents if it needs them again
- **r4-r12**
  - Variable register
  - The callee saves their contents and restore them before returning to the caller
SNUVM#1 Calling Convention (contd.)

- r13
  - Stack pointer (sp)
  - The value of sp at the entry of the callee must be equal to the value at the exit of the callee

- r14
  - Link register (lr)
  - If the return address is saved in the memory, it can be used freely

- r15
  - Program counter (pc)
  - Cannot be used for other purposes
SNUVM#1 Calling Convention (contd.)

- Call stack
  - system stack, run-time stack
  - The place where registers are saved at the entry of the callee and restored at the exit of the callee
  - The place where registers are saved by the caller
- Full descending stack
  - A full stack is where the stack pointer points to the last data item written (pushed) on the stack
  - A descending stack grows downwards in memory (i.e., grows to the direction of decreasing addresses)
SNUVM#1 Calling Convention (contd.)

- Parameter passing
  - through r0, r1, r3, r4, and the call stack

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>first</td>
<td></td>
</tr>
<tr>
<td>r1</td>
<td>second</td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td>third</td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td>4th</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sp-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sp-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sp-12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sp-16</td>
</tr>
</tbody>
</table>

...
Stack Frame

- The block of information stored on the stack to effect a subroutine call and return
  - Parameters to the subroutine
  - Saved registers
  - Local variables (in C)
  - Return address
- Allocated on the stack when a subroutine is called
- Removed upon return from the subroutine
- In general, the stack frame for a subroutine contains all necessary information to save and restore the state of a subroutine
foo gets two arguments in r0 and r1 from the caller and returns the result in r0
   • foo(a, b) = bar(a^b)

bar gets one argument in r0 from the caller and returns the result in r0
   • bar(a) = inc(a \times a)

inc gets one argument in r0 from the caller and returns the result in r0
   • inc(a) = a + 1

...
Stack Frame (contd.)

- foo’s stack frame
- bar’s stack frame

```
call foo
   sp  lr
   r6  r5

sp  lr
r6  r5
```

```
call bar
   sp  lr
   r6  r5

sp  lr
r6  r5
```

```
call inc
   sp  lr
   r6  r5
```

foo’s stack frame
bar’s stack frame
Stack Frame (contd.)

- foo’s stack frame
- bar’s stack frame

Diagram showing stack frames for return from foo, return from bar, and return from inc.
Factorial

- **Recursive definition**
  - Defines a function in terms of itself
- **Divide a problem into sub-problems of the same type**
  - Divide and conquer
- **Recursive subroutine**
  - When a subroutine calls itself, the subroutine is called recursive
- **Iteration**
  - Repetition of a sequence of instructions (using a loop)

\[
\text{fact}(n) = \begin{cases} 
1 & \text{if } n = 0 \\
 n \times \text{fact}(n - 1) & \text{if } n > 0
\end{cases}
\]
Factorial (contd.)

```
fact:
    mov r1, #1
L1:
    cmp r0, #0
    beq DONE
    mul r2, r1, r0
    mov r1, r2
    sub r0, r0, #1
    b L1
DONE:
    mov r0, r1
    mov pc, lr
```

Iteration based

```
fact:
    sub sp, sp, #4
    str lr, [sp]
    cmp r0, #0
    beq BASE
    mov r1, r0
    sub r0, r0, #1
    bl fact
    mul r2, r1, r0
    mov r0, r2
    b DONE
BASE:
    mov r0, #1
DONE:
    ldr lr, [sp]
    add sp, sp, #4
    mov pc, lr
```

Recursion based
Factorial (contd.)

call fact(3)
call fact(2)
call fact(1)
call fact(0)
Recursion vs. Iteration

- Any function that can be evaluated by a computer can be expressed in terms of recursive functions without the use of iteration.
- Any recursive function can be expressed in terms of iteration.
  - Iteration + stack