7.6. Recursion

Recursive functions are a special class of functions that call themselves (also known as **self-referential** functions) to compute a value. Like their nonrecursive counterparts, recursive functions create new stack frames for each function call. Unlike standard functions, recursive functions contain function calls to themselves.

Let's revisit the problem of summing up the set of positive integers from *1* to *n*. In previous sections, we discussed the `sumUp` function to achieve this task. Table 1 shows a related function called `sumDown` that adds the numbers in reverse (*n* to *1*), and its recursive equivalent `sumr` :

Table 1. Iterative version (sumDown) and recursive version (sumr)

Iterative	Recursive

```
int sumDown(int n) {
    int total = 0;
    int i = n;
    while (i > 0) {
        total += i;
        i--;
    }
    return total;
}
```
	```
	int sumr(int n) {
	    if (n <= 0) {
	        return 0;
	    }
	    return n + sumr(n-1);
	}
	```

The base case in the recursive function `sumr` accounts for any values of *n* that are less than one. The recursive step calls `sumr` with the value *n-1* and adds the result to *n* prior to returning. Compiling `sumr` and disassembling it with GDB yields the following assembly code:

```
Dump of assembler code for function sumr:
0x400551 <+0>:  push   %rbp                # save %rbp
0x400552 <+1>:  mov    %rsp,%rbp           # update %rbp (new stack frame)
0x400555 <+4>:  sub    $0x10,%rsp          # expand stack frame by 16 bytes
0x400559 <+8>:  mov    %edi,-0x4(%rbp)     # move first param (n) to %rbp-
0x4
0x40055c <+11>: cmp    $0x0,-0x4(%rbp)     # compare n to 0
0x400560 <+15>: jg     0x400569 <sumr+24> # if (n > 0) goto <sumr+24>
[body]
```

```
0x400562 <+17>: mov   $0x0,%eax          # copy 0 to %eax
0x400567 <+22>: jmp   0x40057d <sumr+44> # goto <sumr+44> [done]
0x400569 <+24>: mov   -0x4(%rbp),%eax    # copy n to %eax (result = n)
0x40056c <+27>: sub   $0x1,%eax          # subtract 1 from %eax (result -=
1)
0x40056f <+30>: mov   %eax,%edi          # copy %eax to %edi
0x400571 <+32>: callq 0x400551 <sumr>    # call sumr(result)
0x400576 <+37>: mov   %eax,%edx          # copy returned value to %edx
0x400578 <+39>: mov   -0x4(%rbp),%eax    # copy n to %eax
0x40057b <+42>: add   %edx,%eax          # add sumr(result) to n
0x40057d <+44>: leaveq                   # prepare to leave the function
0x40057e <+45>: retq                     # return result
```

Each line in the preceding assembly code is annotated with its English translation. Table 2 shows the corresponding `goto` form and C program without `goto` statements:

Table 2. C goto form and translation of sumr assembly code

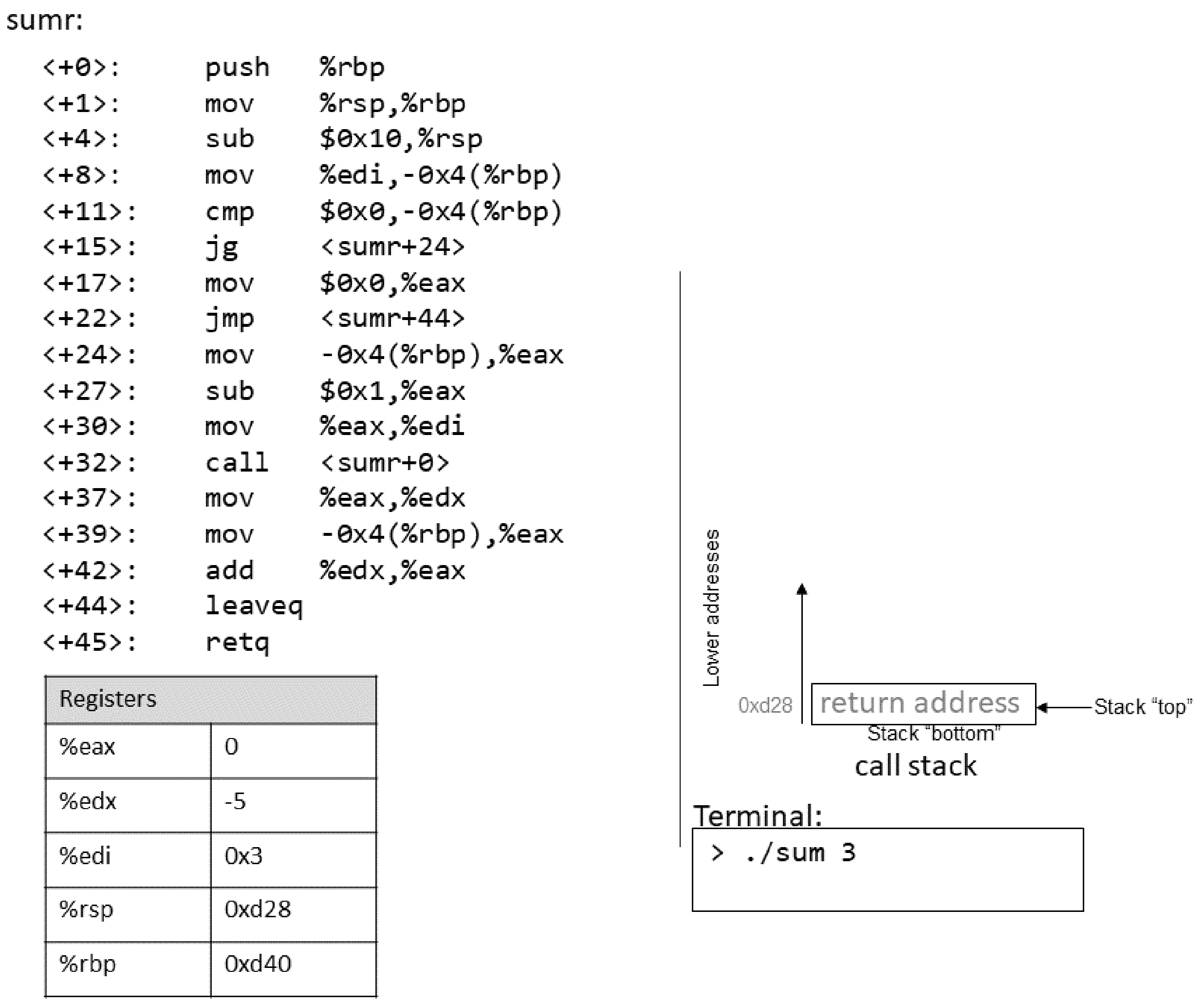
C goto form	C version without goto statements

```
int sumr(int n) {
    int result;
    if (n > 0) {
        goto body;
    }
    result = 0;
    goto done;
body:
    result = n;
    result -= 1;
    result = sumr(result);
    result += n;
done:
    return result;
}
```
	```
	int sumr(int n) {
	    int result;
	    if (n <= 0) {
	        return 0;
	    }
	    result = sumr(n-1);
	    result += n;
	    return result;
	}
	```

Although this translation may not initially appear to be identical to the original `sumr` function, close inspection reveals that the two functions are indeed equivalent.

7.6.1. Animation: Observing How the Call Stack Changes

As an exercise, we encourage you to draw out the stack and see how the values change. The animation below depicts how the stack is updated when we run this function with the value 3.



```
sumr:
   <+0>:     push   %rbp
   <+1>:     mov    %rsp,%rbp
   <+4>:     sub    $0x10,%rsp
   <+8>:     mov    %edi,-0x4(%rbp)
   <+11>:    cmp    $0x0,-0x4(%rbp)
   <+15>:    jg     <sumr+24>
   <+17>:    mov    $0x0,%eax
   <+22>:    jmp    <sumr+44>
   <+24>:    mov    -0x4(%rbp),%eax
   <+27>:    sub    $0x1,%eax
   <+30>:    mov    %eax,%edi
   <+32>:    call   <sumr+0>
   <+37>:    mov    %eax,%edx
   <+39>:    mov    -0x4(%rbp),%eax
   <+42>:    add    %edx,%eax
   <+44>:    leaveq
   <+45>:    retq
```

Registers	
%eax	0
%edx	-5
%edi	0x3
%rsp	0xd28
%rbp	0xd40

```
> ./sum 3
```

Contents