Functions

Functions and procedures help break-up a program into smaller parts making it easier to code, debug, and maintain. Function calls involve two main actions:

- **Linkage**
  Functions must be able to return to the correct place in which it was originally called.
- **Argument Transmission**
  Functions must be able to access parameters to operate on or to return results

Functions

Linkage

The linkage is about getting to and returning from a function call correctly. Two instructions handle the linkage:

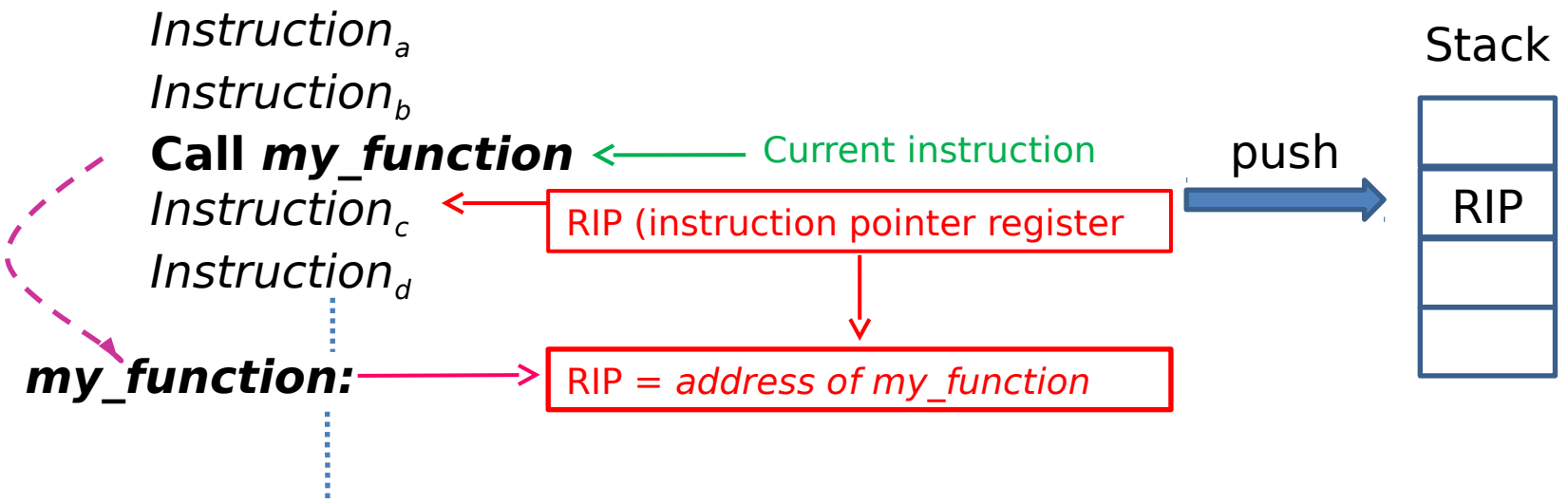
```
call <funcName>

ret
```

call transfers control to the named function.

ret returns control back to the calling routine.

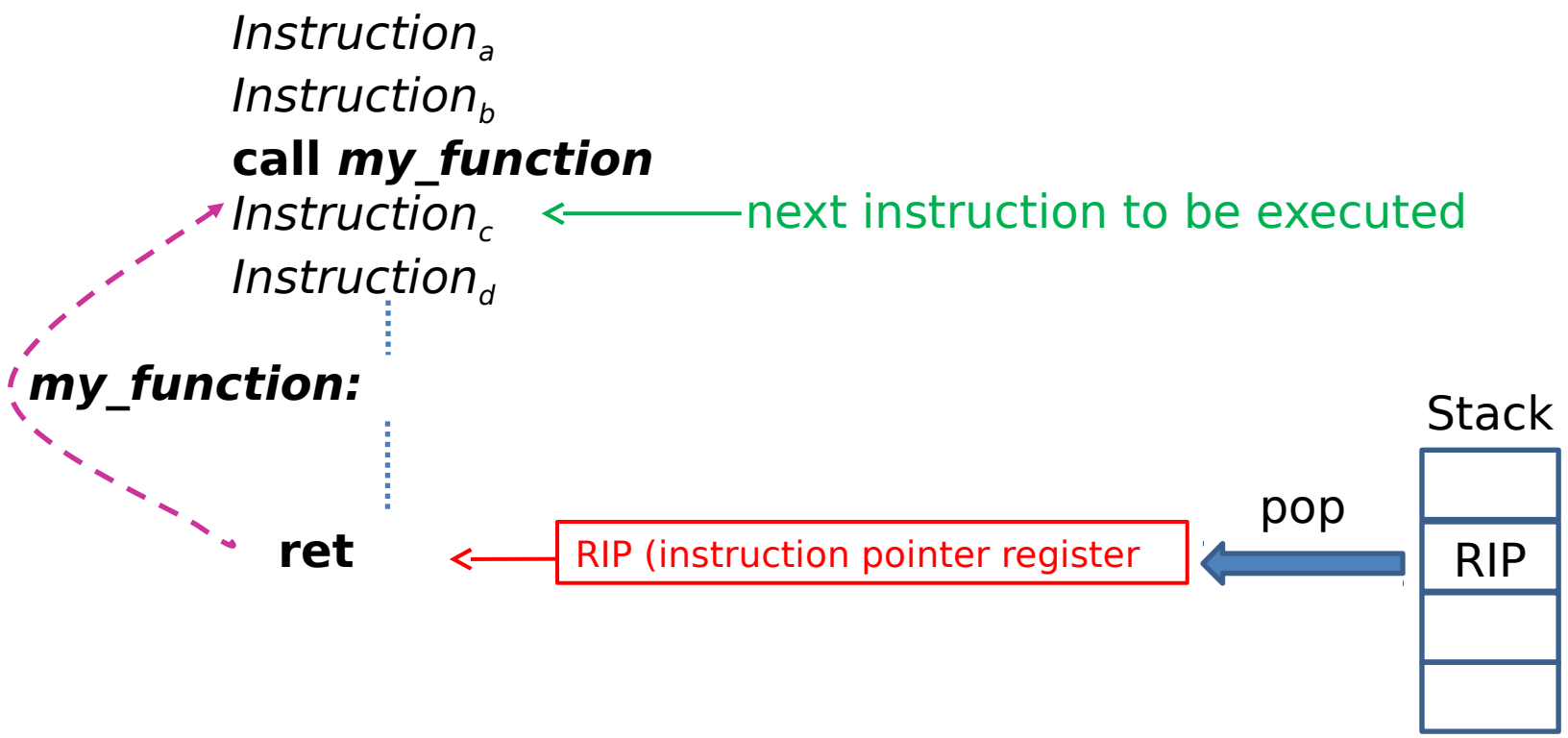
Functions

• The call works by saving *return address* by placing contents of the RIP register on the stack. The RIP register points to the next instruction to be executed (which is the instruction immediately after the call).



Functions

• The ret instruction is used in a procedure to return. The ret instruction pops the current top of the stack (**RSP**) into the **RIP** register.



Functions

Argument Transmission

There are various ways to pass arguments to and/or from a function.

- Placing values in register
  - Easiest, but has limitations (i.e., the number of registers).
  - Used for first six integer arguments.
  - Used for system calls.
- Globally defined variables
  - Generally poor practice, potentially confusing, and will not work in many cases.
  - Occasionally useful in limited circumstances.
- Putting values and/or addresses on stack
  - No specific limit to count of arguments that can be passed.
  - Incurs higher run-time overhead.

Functions

Parameter Passing

Register	Usage
rax	Return Value
rbx	Callee Saved
rcx	4th Argument
rdx	3rd Argument
rsi	2nd Argument
rdi	1st Argument
rbp	Callee Saved
rsp	Stack Pointer
r8	5th Argument
r9	6th Argument
r10	Temporary
r11	Temporary
r12	Callee Saved
r13	Callee Saved
r14	Callee Saved
r15	Callee Saved

Functions

Example: Passing arguments via registers

```
; stats1(arr, len, sum, ave);
mov rcx, ave  ; 4th arg, addr of ave
mov rdx, sum ; 3rd arg, addr of sum
mov esi, dword [len] ; 2nd arg, value of len
mov rdi, arr    ; 1st arg, addr of arr
call stats1
```

```
global stats1
stats1:
        push r12 ; prologue
        mov r12, 0 ; counter/index
        mov rax, 0 ; running sum
sumLoop:
        add eax, dword [rdi+r12*4] ; sum += arr[i]
        inc r12
        cmp r12, rsi
        jl sumLoop
        mov dword [rdx], eax ; return sum
        cdq
        idiv esi            ; compute average
        mov dword [rcx], eax ; return ave
        pop r12             ; epilogue
        ret
```

Functions

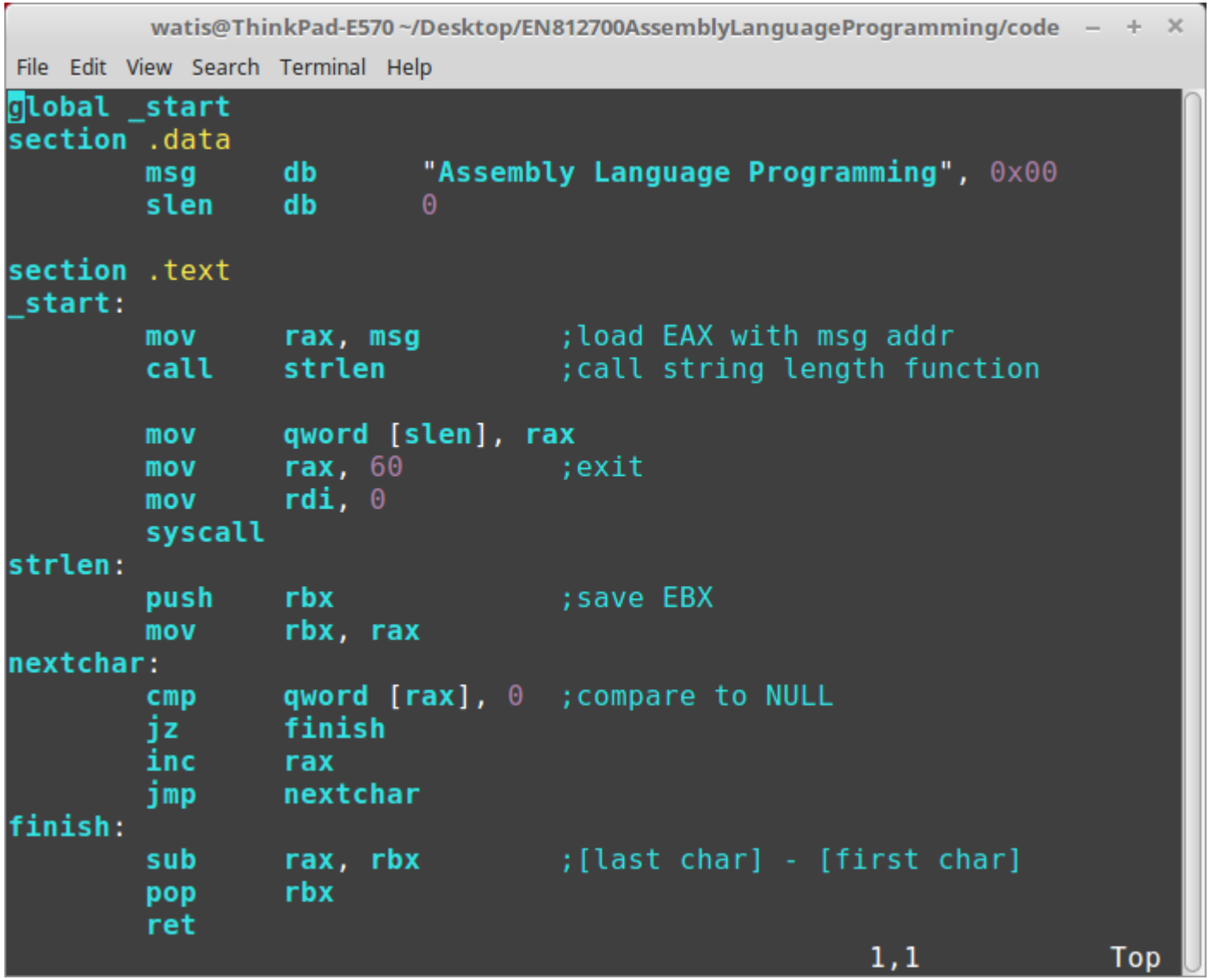
Example: Passing arguments via stack

```
Instruction_a
Instruction_b
push reg1    ; send argument 1
push reg2    ; send argument 2
call my_function
pop reg7     ; retrieve return value 2
pop reg8     ; retrieve return value 1
Instruction_c
Instruction_d
```

```
global my_function
my_function:
        pop reg3  ; retrieve argument 1
        pop reg4  ; retrieve argument 2
                  .
                  .
                  .
        push reg5 ; return value 1
        push reg6 ; return value 2
        ret
```

Functions

Example: Function



Functions

Example: Passing arguments