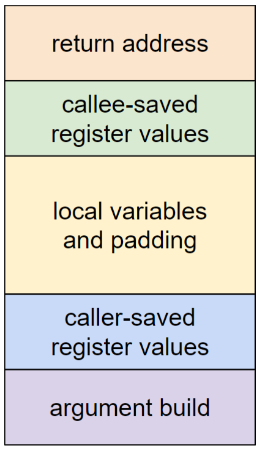
CSE 351 Section 5 – Calling Convention & Stack Discipline

Welcome back to section. We're happy that you're here ☺

Procedures and the Stack

The Stack is a region in memory which starts from the highest memory address and grows downwards when necessary as programs execute. We consider the region with the highest address the Stack "bottom" and the region with the lowest address the Stack "top". **%rsp** is a dedicated special register which points to the current Stack top.

In x86-64, the Stack can be divided into individual **stack frames** for each function, which may contain the following sections (in order):


return address
callee-saved register values
local variables and padding
caller-saved register values
argument build

- **Return address**
  - Pushed by callq; address of instruction after callq
- **Callee-saved registers**
  - Only if function modifies/uses them
- **Local variables**
  - Variables that fit in a register may not be allocated on the Stack
  - Unavoidable if variable is too big for a register (*e.g.*, array)
  - Unavoidable if variable needs an address (*i.e.*, uses &var)
- **Caller-saved registers**
  - Only if values are needed across a function call
- **Argument build**
  - Only if function calls a function with more than six arguments

Example: consider the following lines of code and draw out the stack frames for main and foo right before foo returns (*i.e.*, before any deallocation):

```
int main(int argc, char* argv[]) {
  int x = 351;
  int a[] = {1, 2, 3};
  int y = foo(&x, 2, 3, 4, 5, 6, 7);
  return y + argc;
}

int foo(int* arg1, int arg2, ..., int arg7) {
  return *arg1 + arg7;
}
```

Consider the following x86-64 assembly and C code for the recursive function `rfun`.

```
// Recursive function rfun
long rfun(char *s) {
  if (*s) {
    long temp = (long)*s;
    s++;
    return temp + rfun(s);
  }
  return 0;
}

// Main Function - program entry
int main(int argc, char **argv) {
  char *s = "CSE351";
  long r = rfun(s);
  printf("r: %ld\n", r);
}
```

```
00000000004005e6 <rfun>:
  4005e6: 0f b6 07                movzbl (%rdi),%eax
  4005e9: 84 c0                   test %al,%al
  4005eb: 74 13                   je 400600 <rfun+0x1a>
  4005ed: 53                      push %rbx
  4005ee: 48 0f be d8             movsbq %al,%rbx
  4005f2: 48 83 c7 01             add $0x1,%rdi
  4005f6: e8 eb ff ff ff          callq 4005e6 <rfun>
  4005fb: 48 01 d8                add %rbx,%rax
  4005fe: eb 06                   jmp 400606 <rfun+0x20>
  400600: b8 00 00 00 00          mov $0x0,%eax
  400605: c3                      retq
  400606: 5b                      pop %rbx
  400607: c3                      retq
```

a) In terms of the C function, what value is being saved on the stack?

b) What is the return address to `rfun` that gets stored on the stack during the recursive calls (in hex)?

c) Assume `main` calls `rfun` with `char *s = "CSE351"` and then prints the result using the `printf` function, as shown in the C code above. Assume `printf` does not call any other procedure. Starting with (and including) `main`, how many total stack frames are created, and what is the maximum depth of the stack?

d) Assume `main` calls `rfun` with `char *s = "CSE351"`, as shown in the C code. After `main` calls `rfun`, we find that the return address to `main` is stored on the stack at address `0x7fffffffdb38`. On the first call to `rfun`, the register `%rdi` holds the address `0x4006d0`, which is the address of the input string "CSE351" (i.e. `char *s = 0x4006d0`) during the **fourth** call to `rfun`.

For each address in the stack diagram below, fill in both the ***value*** *and a* ***description*** *of the entry.*

Memory Address	Value	Description
0x7fffffffdb48	Unknown	%rsp when main is entered
0x7fffffffdb38	0x400616	Return address to main
0x7fffffffdb30	Unknown	Original %rbx
0x7fffffffdb28		
0x7fffffffdb20		
0x7fffffffdb18		
0x7fffffffdb10		
0x7fffffffdb08		
0x7fffffffdb00		