15-411: Compiler Design — Spring 2024

Recitation 4: Calling Conventions Solutions — 16 February

The L3 language adds support for function calls, type definitions, and header files with C interoperability. In this recitation, we'll discuss some of the implications of adding these features and how your compiler should deal with them.

Caller- and Callee-Saved Registers

In Lab 3, your compiler's code-generation and register allocation phases will need to distinguish between *callee-saved* and *caller-saved* registers:

- The values stored in **callee-saved registers** must be preserved across function calls. This means that your function must save and restore any callee-saved registers that it modifies.
- The values stored in **caller-saved registers** may be modified by any function call, so your compiler cannot assume that they will retain their values after calling a function. If you need those values to be preserved, you must save and restore them before and after the function call.

To avoid having callee-saved registers occupy a very long live range during register allocation, we can handle them separately. Prioritize allocating caller-saved registers; if they are insufficient, we assign assign callee-save registers before we resort to spilling, but we make sure to save them to the stack at the beginning of a function and restore them at the end. This is more efficient than always saving and restoring all callee-saved registers.

Function	64-bit	32-bit	16-bit	8-bit
Return Value	**%rax**	**%eax**	%ax	%al
Callee saved	**%rbx**	**%ebx**	%bx	%bl
4th Argument	**%rcx**	**%ecx**	%cx	%cl
3rd Argument	**%rdx**	**%edx**	%dx	%dl
2nd Argument	**%rsi**	**%esi**	%si	%sil
1st Argument	**%rdi**	**%edi**	%di	%dil
Callee saved	**%rbp**	**%ebp**	%bp	%bpl
Stack Pointer	**%rsp**	**%esp**	%sp	%spl
5th Argument	**%r8**	**%r8d**	%r8w	%r8b
6th Argument	**%r9**	**%r9d**	%r9w	%r9b
Caller saved	**%r10**	**%r10d**	%r10w	%r10b
Caller saved	**%r11**	**%r11d**	%r11w	%r11b
Callee saved	**%r12**	**%r12d**	%r12w	%r12b
Callee saved	**%r13**	**%r13d**	%r13w	%r13b
Callee saved	**%r14**	**%r14d**	%r14w	%r14b
Callee saved	**%r15**	**%r15d**	%r15w	%r15b

Tracing Function Calls in x86-64

In Lab 3, your compiler must conform to the standard C calling conventions for x86-64. As a reminder, this means that:

- The first six arguments to a function should be stored in `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, and `%r9` (respectively).
- The remaining arguments should be placed on the stack. The seventh argument should be stored at the address `%rsp`, the eighth at `%rsp + 8`, etc.
- The return value of a function should be stored in `%rax`.
- The use of `%rbp` as a base pointer is not required (but you may find that using it simplifies your compiler's logic significantly). LLVM uses the base pointer, but GCC does not.

Another interesting observation: unlike in C, every function in C0 (and thus in L3) has a fixed stack size that can be computed at compile time. This observation allows you to make your compiler's stack-handling much simpler than if you were unable to determine the stack size beforehand.

Checkpoint 0

Draw a stack diagram for the following L3 program at the point when execution reaches line 4. Assume that %rbp is being used as a base pointer.

```
1 int f(int we, int dont, int care, int about, int these, int args, int a, int b) {
2   // assume that x is spilled on the stack
3   int x = a + b;
4   return 2 * x;
5 }
6
7 int main() {
8   return f(0,0,0,0,0,0,3,5);
9 }
```

Solution:

Value	Pointers
Return address of _main()	
Previous %rbp	
b; Arg. 8 of f()	
a; Arg. 7 of f()	
Return address of f()	
main's %rbp	← %rbp
x	← %rsp

Checkpoint 1

Using your stack diagram, convert the program to x86-64 assembly following the standard calling conventions. Remember to use the 64-bit and 32-bit versions of the registers appropriately and that stack grows downward!

Solution:

```
_c0_f:
  push %rbp
  movq %rsp, %rbp
  subq $8, %rsp
  movl 24(%rbp), %eax
  addl 16(%rbp), %eax
  movl %eax, (%rsp)
  movl (%rsp), %eax
  imull $2, %eax
  addq $8, %rsp
  pop %rbp
  ret
_c0_main:
  push %rbp
  movq %rsp, %rbp
  subq $16, %rsp
  movl $0, %edi
  movl $0, %esi
  movl $0, %edx
  movl $0, %ecx
```

```
movl $0, %r8d
movl $0, %r9d
movl $3, (%rsp)
movl $5, 8(%rsp)
call _c0_f
addq $16, %rsp
pop %rbp
ret
```

Tips and Hints for Lab3

- **Header Files in L3:** Unlike in C, header files in L3 (and above) are only used to declare types and external functions. If a function is declared in a header file, then it may not be defined in the program – it is declared as *external*. External functions are defined in C source files, which are linked together with the assembly produced by your compiler.
- **RBP:** You are not required to use %rbp as a base pointer, so you are allowed to treat it like a normal callee-saved register in your compiler.
- **Code Review:** Code Review happens one week after Lab3 is due. So if you haven't polished the style of your compiler and added a README describing the design of various passes of your compiler, now would be a good time to start. We are looking for good coding style and comments, modular design, and that both of you are familiar with all components of the implementation.