CMSC430: Introduction to Compilers

Michael Li

Various notes for the latter part of CMSC430

`rsp` grows downward (`Sub` to push and `Add` to pop)

C functions calls are 16-bytes aligned

Callee-Saved Register: Caller doesn't have to worry about registers being clobbered since the called function is responsible for saving and restoring the value before returning

- If the entry code wants to use Callee Registers, it needs to save and restore the registers before exiting

Caller-Saved Register: Called function can clobber the data so we need to save and restore the value (Push/Pop)

- If the entry code wants to use external functions, it is a caller and has to main caller registers before passing control to the external function

Registers

- `rdi` Caller argument 1
- `rsp` Stack pointer
- `rbx` Pointer to the next free memory location
- `eax` 32 bit register used to read and write characters

Hoax Vector Pointer contains the size of the vectors and a list of words (content of the vector)

- For an empty vector, we use a single representation (symbol `'()`) rather than pushing the length `0` onto the heap

String Pointer: contains the length of the string and a list of **32-bit** characters to save space (don't need all 64 bits)

- If string length is odd, we need add another 32 bits to realign into 64 bit words
- Need to use `eax` register (32 bits), instead of a typical 64 bit register to read/write

Conventions on Calling:

Note that `Call` pushes the return address to the stack, so we need to offset access to arguments by 8 in the callee function

This is annoying to deal with since we need to access past the return address in the stack

Instead in **Iniquity**, we use `Lea (label r)` first to push the return address, then push any arguments so we don't have to offset `rsp` in the callee function, and finally **Jump** to the callee function label

As a design decision, we let the function callee pop arguments before returning (it will know how many arguments there are)

Function Caller Conventions

- Push return address

- Push arguments
- Jump to function label

Function Callee Conventions

- Access arguments on stack starting at offset 0
- Pop arguments before returning
- Return to the pushed return address created by caller

Tail Functions: Call that occurs in the **tail position** (last subexpression to be computed)

- For the interpreter, this was already handled by Racket (uses tail calls)

To implement tail calls in the compiler, we need to override the passed arguments by offsetting `rsp`

- We evaluate `e0` then place it at the location of `a0`
- We evaluate `e1` then place it at the location of `a1`

To determine if an expression is a tail call, we add a boolean parameter to the compiler expression to signify if it is a tail call or not

- **Note**: Top level expression and the body of a function is ALWAYS a tail call