Brian Nguyen

Assignment 9

* Briefly describe the concept of "Stack Frame" (1 through 5 points depending on quality of the answer)

A stack Frame is a portion of the stack. When a function is called a stack frame is created which consists of the parameters of the routine or procedure called, the local variables of the routine and the memory address of the last stack frame. Stack frames are necessary in order to keep track of the running of the program so that variables required for the operations of a routine or provided when necessary and so the program can execute in the right order which routine is called and properly return to the place in the code where it was called last and continue running.

Hint: <https://en.wikipedia.org/wiki/X86_calling_conventions>

*  Briefly describe the concept of a "Calling Convention" (1 through 5 points depending on quality of the answer)

“Calling Conventions” refer to the rules or procedure that is followed by parts of the registers and stack when a function is called. Some of these rules include the order in which parameters are pushed on to the stack upon a procedure call, which type of registers may be assigned which types of values or whether registers may or may not keep their values they had before the call.

*  Why do the book examples always start functions with "push ebp" and "mov ebp,esp" (2 points)

Pushing ebp is “saving” the value of the base pointer onto the stack which is the memory address of the calling function. Then moving esp into ebp is moving the value of the stack pointer into the base pointer. So, the value of the stack pointer which would be the memory address of the current function’s stack frame would be held by ebp. This is done in order to be able to “move” back to the original calling function once the called function is done executing.

*  What is the x64 equivalent to "push ebp" and "mov ebp,esp)" (2 points)

Push rbp

Mov rbp,rsp

*  What are the differences between "Microsoft x64 calling convention" and "System V AMD64 ABI calling convention" (2 points)

Hint: <https://en.wikipedia.org/wiki/X86_calling_conventions>

For the Microsoft x64 calling convention rcx, rdx, r8,r9 are used for the for first integer pointer arguments—xmm0 to xmm3 for floating point arguments—and additional arguments are pushed on the stack and the rest of the arguments are pushed on to the stack in a region called the “shadow space” containing 32 bytes. For the System V AMD64 calling convention rdi, rsi, rdx, rcx, r8, r9 are used for arguments – xmm0 to xmm7 used for floating point arguments—and additional arguments are pushed on to the stack. For x64 the registers rbx, rbp, rdi, rsi, rsp, r12 ,r13, r14 and r15 must be saved by the calling function while for V AMD64 ABI rbx, rbp, and r12 to r15 must be saved by the calling function.

void myFunction(int a, int b, int c, int d, int e, int f) {

//some code here

}

*  Upon entry to myFunction (above), what does the stack look like when using Microsoft x64 calling convention? (3 points)

The first four arguments from left to right are assigned to registers so the remaining arguments for int e and int f are pushed on to the stack.

So…

e pushed on stack.

f pushed on stack.

Hint: <https://docs.microsoft.com/en-us/cpp/build/parameter-passing>

*  Upon entry to myFunction (above), what does the stack look like when using stdcall convention (3 points)

Hint: <https://en.wikipedia.org/wiki/X86_calling_conventions>

In stdcall argument passing order is right to left so the arguments are pushed to the stack from left to right.

So…

f pushed on stack.

e pushed on stack.

d pushed on stack.

c pushed on stack.

b pushed on stack.

a pushed on stack.

*  Upload your assignment to GitHub
*  Email [james.wilson@rccd.edu](mailto:james.wilson@rccd.edu) with the url to the source code on GitHub

\*\*\* 22 points \*\*\*