Review for Midterm

This review outlines the key concepts that you *must* understand before taking the midterm exam. Also, I will highlight the topics that many students got wrong in the midterm. However, The midterm exam is *not limited* to the topics that appear in this review since many knowledge we covered in this course are often interconnected with one another.

1 Bits, Bytes and Integers

Basic Math (for computer scientists). Make sure you can comfortably perform basic Base10(Decimal), Base2(Binary), Base16(Hexadecimal) number conversions and arithmetics

The sizes of C primitive data types in x86-64. I'm not a big fan of forcing students to memorize things for the exam. But knowing the widths of the data types and the range of numbers they can represent is essential for any programmers.

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
`char`	1	1	1
`short`	2	2	2
`int`	4	4	4
`long`	4	8	8
`float`	4	4	4
`double`	8	8	8
`long double`	–	–	10/16
pointer	4	8	8

Bit-wise operators and logical operators in C.

- Know the difference between bitwise and logical operaters.
- Regarding shift, some students were confused about loigcal shift vs arithmetic right shift. Arithmetic is the one that replicates the MSB on the left.
- Know that many processor architectures use shifting to perform power-of-2 mult/div of a number and a multiples of 2 (e.g., `shl`).

Unsigned and two's complement.

- Converting between unsigned and two's complement. You should be able to do this by now
- Be able to calculate the numeric range of a number with bit width of w.
- When does overflow happen when you are adding numbers in a computer?

Representing data. Understand the difference between little-endian and big-endian. Also understand that a string is simply a series of one byte that is terminated by a `NULL`. Therefore, a string is stored the same way in little/big endian.

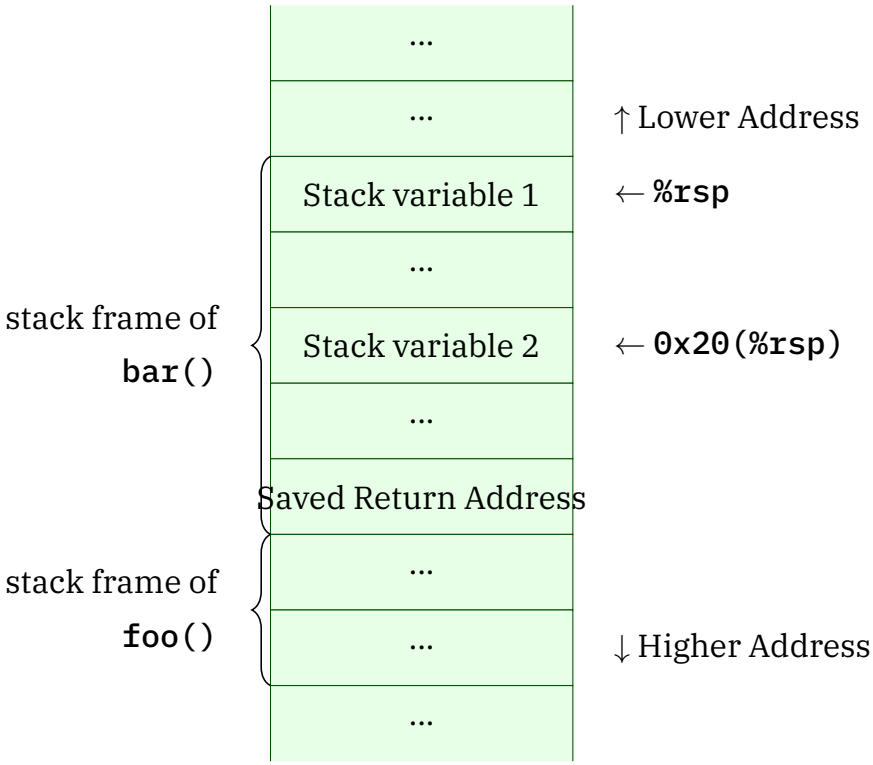
2 Machine-level Representations of Programs

Calling convention. Calling conventions allow exchange of data between functions and corruption of registers and memory. Understand data passing (arguments and return value) between the functions (now you know why C only supports one return value). Also, understand how the callee function can return to caller.

System V AMD64 ABI (64-bit Linux on x86)	
Argument Passing Registers	`RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9`
Return Value Register	`RAX`
Callee-saved Registers	`RBX`, `RSP`, `RBP`, and `R12`–`R15`

Loops. Loops in machine-level are composed of condition checks and jumps. Take a look at the lecture slide example of transforming C code loops such that they only have `if`s and `goto`s. It will help you understanding loops at the machine-level.

Stack layouts. You should be able to draw stack layouts for functions in assembly language. As you probably noticed by now, human conventions that express data (e.g., `char array[10];` or `int a;`) do not exist at the machine level. Stack pointer register is adjusted during function execution, and the code that access the stack space defines the location of the local variables. Make sure that you can picture the locations of the data in your head by reading the assembly code (if you have not acquired that skill through AttackLab).



Accessing Arrays. Be able to understand the assembly code that accesses array. Can you guess the length, size of array elements, and array dimensions?

Nested Arrays. What do nested arrays look like in memory? Given an {`Datatype`} `A[R][C]`, how is accessing an element `A[M][N]` achieved in assembly?