a. Pipelining - when seperate phases of code are executing simulatenously to improve performance
b. %rdi - used for integer and memory address arguments
c. %esp - register that will always point to the top of the stack
d. Stack frame - space allocated on the stack, holds neccesary information to save and restore the state of a procedure
e. Address alignment - keeps processor from grabbing only half of a word so not to than wasting its clock cycle
f. Caller save / callee save - caller save means the current stack frame will save register values, callee means the stack frame for the function being called will save the registers. callee is generally faster because it will only save the registers it uses rather then all of the registers.

2. Why use the lea instruction for mathematical operations when it is designed for calculating memory addresses? What limitations does it have?: ability to perform addition with either 2 or 3 operands and ability to store the result in any register.
 - doesnt affect condition flags

3. How are 1D & 2D arrays represented in memory? : stored linearly

4. How is nested array element access performed?
5. How is multi-level array element access performed?

6. What is assert() and why do we use it? : procedure that decalres 'this(w/e your looking for)' has to be true or it will crash.

7. What happens in a procedure call? : old register values are stored in memory, procedure stored at top stack and executed, when procudure finished old register values restored and poped off stack

8. How does the stack make recursion work? : the stack adds recursive calls to top of stack and pops off when no longer needed

9. Describe what 8(%ebp) is using normal compiler conventions in IA32. : holds local variables

10. Why is x86-64’s passing of arguments to procedures significantly faster than IA32? : 64 has 2x amount of register can hold more values