Ashley Nguyen - apn2my

InLab08 - inlab8.pdf

3/29/16

For passing by value for ints, floats and chars, main starts by loading the stack pointer's register above an int bits (4) of initialized local variables in main into ECX. My program prompted the user to enter in integers as the parameters passed. Once "cout" was pushed onto the stack, the caller called basic ostream to use the functions within that package. The callee (basic ostream package) performed add and sub operations on the stack pointer to update the stack on the values that are to be inputted (user input). A simple average was calculated from 3 user inputs, which in assembly, the "cin" values were pushed onto the stack and then the code jumps to prompt the user for another input. Passing in values manipulates the stack directly with EAX, EBX, ECX etc. However this is not the case when passing by reference.

For passing by reference, retrieving the input into the assembler with cout and cin are generally the same. BUT when performing the calculation by performing the operations on the stack, the DWORD PTR[ebp] is used as a holder for the reference values. Whatever the input retrieved by reference, eax is moved onto the DWORD PTR which is then used to reference the calculation operations (multiplication in this case). The DWORD PTR values are moved onto the actual stack registers (EAX, EDX) to perform the imul operation on. In addition there are more lea instructions to copy addresses of the parameters to the registers. The values in these registers are then pushed onto the stack. Passing by pointers operates similarly to passing by values as described above.

For an array of ints, my program looped through the array and placed the value of the average from the previous program added to the index of the array in an array of 10 elements. When the callee is called, the parameters were accessed by first creating the whole array on the stack. To begin accessing the elements, the EBP pointer begins at the first element in the array. Then, lea is called to load a pointer to the array which is then saved in the register which holds the array. The callee accesses the parameters inside the function since the assembler iteratively moves each subsequent value ([EBX + 4], [EBX + 8] for integers) onto EAX in backwards order to be called by LEA (load effective address) to load the array values. LEA creates a pointer to the array that is saved in the array variable (another register). Thus, the array values are pushed onto the stack which the callee can access.