Computer Architecture

Travis Thiel

Programming Assignment 2: Computer Arithmetic (Calc)

This project accepts from the user four arguments through the terminal and saves them as the parameters op, num1, num2, and base. It then declares a character negfin that will hold the final negative value and two integers, neg1 and neg2, that will hold 1 if negative. Memory is then dynamically located for a pointer to a structure called BinaryArr which holds a string binRep which will be the binary representation of the number and a pointer to another BinaryArr. This structure acts as a Linked List data structure, which I chose to use to deal with arbitrarily long inputs. If any of the arguments accepted from the user are null then an error is printed to stderr and the program exits. Through a series of if statements the sign and bases of the two strings are found and depending on the base the string is passed to a function, either binToBin(), decToBin(), hexToBin(), or octToBin(),to be converted into their binary representations. After the now binary numbers are either added or, well, added. If the op is '+' then based on the signs of each number and which one is bigger decides the final sign of the sum. If the op is '-'then the sign of the second number is simply switched to the opposite and they are added as in the previous case. This is analogous to x-y=x+(-1*y). The solution is then outputted in the user determined base represented in a string.

Since this program needs to be able to handle arbitrary size numbers in different bases I choice to represent the numbers in a linked list with binary values. Though not the most efficient method, it was the simplest within the time given. Binary numbers are easy to add as the carry will either be 0 or 1 and that can easily be transferred to the next node. The linked list allows me to continue to accept characters even after the 32 bit limit of integers. In that case the head would hold the first 32 bits and next node will contain the next 32 bits, and so on. When dealing with negative numbers all I have to do is take the two's complement as the strings represent unsigned numbers. The final point that makes binary linked lists effective for this program is the simplicity of converting them to the different output bases. As of right now I am having issues with both addition between nodes of number 1 and number 2, but any 32 bit numbers do add together.

My biggest challenge that I have come across is inputting the solution in octal. With my implementation of the linked lists, octal numbers do not fit nicely into the 32 bit strings. With the first node, when placing the binary representation there are 2 indices left but octal numbers come in groups of three binary digits. I have it implemented where it will move the left most digits over to the next node, but then that node gets shifted and it gets very messy.

The biggest strain on this program efficiency is the fact that for reading, converting to binary, and converting to the output base, all have to iterate through a string 32 bits long for each node in the linked list and each linked list as there are 2. Fixing the number of nodes to 'n', the worst case and best case

efficiency analysis is (32*3*2n). The Big O is O(n) time. As the number of nodes increases the time and amount of space this program requires increases linearly.