CMPSC 210
Honor Code: This is my work.
8 December 2015

Integer and Variable Expressions Translated to MIPS Assembly Language

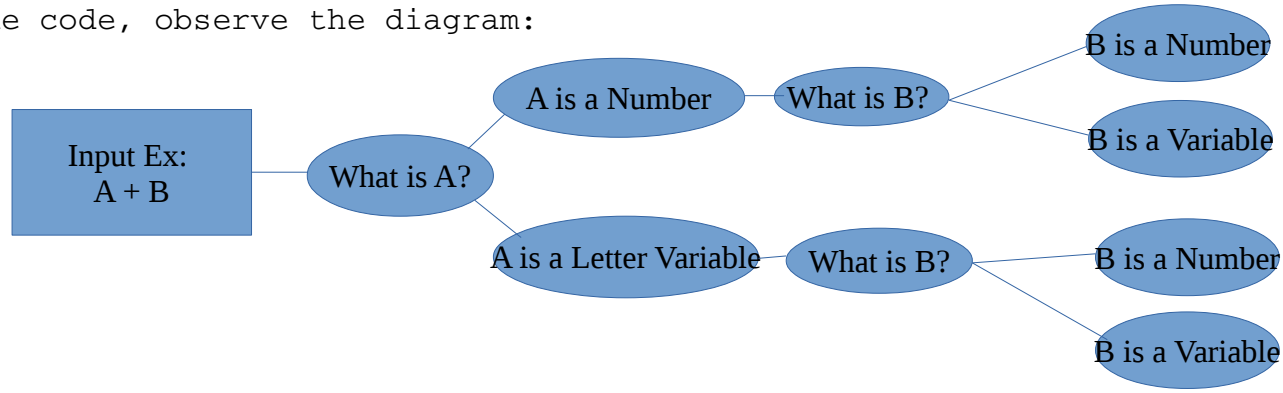
SJ Guillaume

Project Description:

The goal of this project was to use knowledge of the C programming language, to translate arithmetic expressions into MIPS assembly code. This demonstrates proficiency in both C and MIPS, and its goal is to be used as a tool to students learning MIPS. This is accomplished through extensive commenting in the C program itself, and also extensive commenting in the MIPS code. It will demonstrate to students the difference between la, li, and lw commands in MIPS as very different ways to load values into registers. It demonstrates how to use MIPS to print integer and strings using syscalls. It also demonstrates how to end a program using syscalls. The addition and subtraction show how to use registers efficiently by using only the $s0 and $s1 registers, as well as how to format the add and sub lines in order to output the correct answer. The multiplication section shows how to create loops in MIPS. The C part of this project is useful as a learning tool as well, it demonstrates effective usage of comments, as well as control flow structures. It also effectively shows how to use the strsep function and how to do character arrays. Printf statements also demonstrate very well how to print program variable values.

Explanation of Code:

The code works by first prompting the user for an expression and using fgets to read in the line. We then use strsep to get the first letter or number, we check which one it is and enter into the appropriate if statement. After this we use strsep to get the next part of the input, and we assume this is the operator since this program does not handle compound expressions. We save the operator information for use shortly after the next step is performed. We use strset to find the last part of the expression, and we find out if it is a letter or a number. Then we funnel to the appropriate part of the code, observe the diagram:



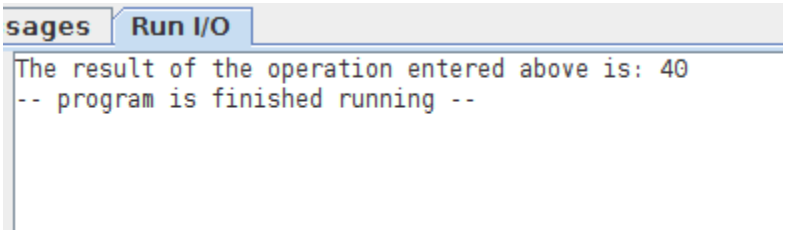
After sorting through the variable/number status of each piece of the input, we have further if statements for each of the four categories, we have if statements for what to do if the operator is for addition, subtraction, and multiplication. Once this is executed, the program goes to the end and prints out the lines for what MIPS needs to close the program, and the process is complete so the C program ends.

How to Run the Project:

This program works by the user compiling and running the program finalProj2.c . The program will prompt the user for an integer, variable, or combination expression to convert into the MIPS assembly language. The program is able to handle expressions that perform addition, subtraction, and multiplication. The program will automatically assign 10 to the variable in the first number position, called x, and 30 to the variable in the second number position, called y. The program will then provide the MIPS operations to print out the statement entered translated into valid MIPS code.
The image below is the output provided for the expression x + y.

```
guillaumes@aldenv132:~/repos/210/cs210-guillaumes/Final_Project$ ./finalProj2
Enter an integer, variable, or combination expression:
x + y
#Paste the following code into MARS:
    .data
    .align 2
message:    .asciiz "The result of the operation entered above is: "
    .align 2
x:  .word   10
    .align 2
y:  .word 30
    .text
#load the contents of x and y into $s0 and $s1 respectively
lw    $s0, x
lw    $s1, y
#add x + y and store result in $s0
add   $s0, $s0, $s1
#print result
li    $v0, 4
la    $a0, message
syscall
li    $v0, 1
add   $a0, $s0, $zero
syscall
    #close program
    li  $v0, 10
    syscall
```

By copying and pasting the output into a MIPS program and running it, we will get the following output in the terminal:



Thus we have a program which will print the output of any expression of two variables and close the program.

Lessons Learned:

There were many learning challenges in this program. Going through all the logic and correctly using the strsep function was a challenging part. Also getting all the print statements produced from C to work correctly was the most challenging part, partially because it was time consuming. I wish I had checked implemented parts before going back to check the entire program. The most valuable lesson from this was how complex of a job it is for compilers to convert information in C into assembly language. I am very glad this was the culminating project I chose because I have reflected on how many things languages such as C allow us to do more simply.