CSI 465 Compiler Design

LAB 2: Expressions

Objectives:

a) Understand expression evaluation order.
b) Understand memory allocation and register use in the evaluation of expressions.
c) Compare and contrast a zero operand instruction set with a two operand instruction set.
d) Critically think about code optimization for simple expressions.

1) Background: In our course you are creating a compiler that utilizes a stack to evaluate expressions. The assembly language associated with Frances uses two-operand syntax. In this lab you will examine the use of memory to evaluate expressions for this two operand instruction set.

2) Exercises:
   a) Type the following code in the Frances code window.

```
int main(){
   int x, y, z;
   x = 2;
   z = 5;
   y = (x + z) * 12;
 }
```

      i) Write the assembly code that performs the assignments and the expression.
      ii) Explain exactly what is occurring in this assembly code.
      iii) What changes in the assembly code if you change the expression to `y = 12*(x + z);`
      iv) How does this compare to what we discussed in class in regards to expression evaluation with a stack architecture?

   b) Next change the code to the following.

```
int main(){
  int x, y, z;
  x = 2;
  z = 5;
  y = (x + z);
  y = y * 12;
}
```

      i) What is the difference in the assembly code?
      ii) Is this more efficient?
      iii) Is there a better way to generate assembly code to evaluate the expression `y = (x + z)*12`

c) Next enter the following code.

```
int main(){
  int x, y, z, w;
  x = 2;
  y = 5;
  z = 9;
  w = x - y – z;
}
```

i) Explain the resulting code in terms of the variables x, y, z, w.