CPSC 323 - Fall 2017
Programming Project #4 - Optimization
due Friday, 12/1 by 11:55pm on Titanium

This programming assignment is an extension of Program #3. The goal is to optimize the assembly code developed in that earlier project so that the final version can run faster.

The optimization will be accomplished in two stages:

Stage 1. Use registers instead of variables

Where: In the parser code.

What to do: Get rid of unnecessary lw and sw operations.

How:

Assign a designated register to each variable as soon as that variable is declared. Any changes that were originally meant for that variable will be made to this register. Therefore, there is no need to load a value from RAM to the register each time the variable is used. Similarly, there is no need to store the value from the register to RAM.

Instructions that just moved data between RAM and those registers (lw,sw) can be eliminated (just don't generate the assembly code that occurred in program #3).

Math operations whose results were originally sent to variables, can now be sent to the specific registers representing those variables.

Stage 2. Optimize the assembly code by removing unneeded move instructions.

Where: In new functions that look at the assembly code from the parser

What to do: Inside basic blocks, remove 'move' statements that are not needed.

How:

Remove move instructions that do self-assignment (i.e. move $t1, $t1)

Modify instructions where data is stored in one register and then moved to another so that it will store the data into the second register originally and then remove the move instruction.

```
ex.     add  $t3, $t1, $t2
        move  $t0, $t3
```

could be changed to just

```
        add $t0, $t1, $t2     without the move instruction
```

In addition, the program should now identify an error if a variable value is being used in an expression before it has been given a value.

The program should run as before, with the name of the input file being passed in as a command line argument and output of the final assembly code being sent to the console window.

Sample results for input file three:

Original output: 26 instr.	After stage 1: 21 instr. t0-myint, t1-foo, t2-bar	After stage 2: 17 instr.
li $t0, 1	li $t3, 1	li $t1, 1
sw $t0, foo	move $t1, $t3	
li $t0, 2	li $t3, 2	li $t2, 2
sw $t0, bar	move $t2, $t3	
---	---	---
while:	while:	while:
lw $t0, foo		
li $t1, 10	li $t3, 10	li $t3, 10
beq $t0, $t1, endWh	beq $t1, $t3, endWh	beq $t1, $t3, endWh
---	---	---
lw $t0, foo		
li $t1, 1	li $t3, 1	li $t3, 1
add $t0, $t0, $t1	add $t4, $t1, $t3	add $t1, $t1, $t3
sw $t0, foo	move $t1, $t4	
lw $t0, bar		
lw $t1, foo		
mult $t0, $t1	mult $t2, $t1	mult $t2, $t1
mflo $t0	mflo $t3	mflo $t3
li $t1, 2	li $t4, 2	li $t4, 2
div $t0, $t1	div $t3, $t4	div $t3, $t4
mflo $t0	mflo $t3	mflo $t2
sw $t0, bar	move $t2, $t3	
b while	b while	b while
---	---	---
endWh:	endWh:	endWh:
lw $t0, bar		
li $v0, 1	li $v0, 1	li $v0, 1
move $a0, $t0	move $a0, $t2	move $a0, $t2
syscall	syscall	syscall
li $v0, 10	li $v0, 10	li $v0, 10
syscall	syscall	syscall