Project 3
Implementation of a Code Generator
Due Sunday, January 13, 2019

1. Problem:

In this assignment you are requested to implement a code generator for MIPS processors. Your code generator should generate code only if the program is free of lexical and syntactical errors. You can use the SPIM simulator, which simulates a MIPS processor, to verify the correctness of your code generator. You can download the SPIM simulator and related information from the web site http://www.cs.wisc.edu/~larus/spim.html. You can apply the techniques for intermediate code generation in this assignment.

In this assignment, we will use a word (4 bytes) for each variable. To simplify register allocation, you can maintain a counter of 10 general registers consisting of $t0~$t9 for expression evaluation. You may assume that 10 registers are enough to evaluate every expression in the program. You can return all the registers you use to evaluate an expression back after evaluating that expression.

The Read, Write, and Exit statements will be implemented by the system calls read_int, print_int, and exit of the SPIM simulator. The code generator reads input from stdin, writes output to stdout, and writes errors to stderr.

A sample **Sunflower** program test1 is given as follows:

```
// A program to sum 1 to n
Program sum Begin
    Var  n
    Var  s

    Read n
    If n < 0 Then
        Write -1
        Exit
    Else
```

```
            Set s = 0
        EndIf
        While n > 0 Do
            Set s = s + n
            Set n = n – 1
        EndWhile
        Write s
    End
```

The stdout output for test1 is as follows:

```
        .data
n:                      # Var n
        .word 0
s:                      # Var s
        .word 0
        .text
main:
        li $v0, 5       # Read n
        syscall
        la $t0, n
        sw $v0, 0($t0)
        la $t0, n       # If n < 0
        lw $t0, 0($t0)
        li $t1, 0
        blt $t0, $t1, L2
        b L3
L2:                     # Then
        li $t0, 1       # Write -1
        neg $t0, $t0
        move $a0, $t0
        li $v0, 1
        syscall
        li $v0, 10      # Exit
        syscall
        b L1
L3:                     # Else
        li $t0, 0       # Set s = 0
        la $t1, s
        sw $t0, 0($t1)
L1:                         # EndIf
L5:                         # While n > 0
        la $t0, n
        lw $t0, 0($t0)
        li $t1, 0
        bgt $t0, $t1, L6
        b L4
L6:                         # Do
        la $t0, s           # Set s = s + n
        lw $t0, 0($t0)
        la $t1, n
        lw $t1, 0($t1)
        add $t0, $t0, $t1
        la $t1, s
        sw $t0, 0($t1)
        la $t0, n           # Set n = n - 1
        lw $t0, 0($t0)
        li $t1, 1
        sub $t0, $t0, $t1
        la $t1, n
        sw $t0, 0($t1)
        b L5
L4:                         # EndWhile
        la $t0, s           # Write s
        lw $t0, 0($t0)
        move $a0, $t0
        li $v0, 1
        syscall
```

2. Handing in your program

You should provide a make file named "makefile" to build this assignment. See online manual make(1) for more information. The executable file should be named "sunflower" namely,

```
    gcc -o sunflower $(OBJ) -lfl
```

To turn in the assignment, upload a compressed file containing makefile, *.l, *.y *.h, and *.c to Ecourse site.

3. Grading

The grading is based on the correctness of your program. The correctness will be tested by a number of test cases.

To facilitate the grading of teaching assistants, you should test your program on the machine csie1.

It is best to incrementally build your program so that you always have a partially-correct working program. It is also best to construct a shell script to systematically test your program.