Experiment #4 Generating Target Code for Assignment, Conditional Statements and functions

Assignment Due: 04/04, 11:59pm

- **To do this assignment, you *may* work in teams of two. Team members must be from the same lab group.**
- **Submit the assignment in piazza as a private message:**
    - **Title: Lab Experiment 03_roll01_roll02_(Odd/EVEN)**

Assignment Details:
In this assignment we build up on the solution to assignment 2 where you generated 3AC for assignment statements. Here we will look into conditional statements and function or subprogram calls for language X. You can find the informal reference manual of the programming language X with some example programs **here.**Please read the instructions carefully.

A. Grammar of Language X

You can find the CFG of the programming language X **here**. In this assignment you are to generate target code following MIPS syntax of every type of statement supported by **X** as an output of your compiler. Several example of MIPS language structure can be found in here, also here.

- Caller's local variable and values should be stored in the stack before calling.
- The parameters should be passes through stack (using stack pointer).
- Return address should be maintained in a return address variable (*ra*).
- Callee's local data should be maintained in its own activation record.
- Callee's return values should be stored in the accumulator variable (there can be more than one accumulator.
- You have to maintain a proper data structure for the activation record and effectively print it in a output file.

C. Implementation:

You will have to write a LEX/FLEX and YACC/Bison specifications in this assignment. In YACC or Bison you have to put down the corresponding CFG productions of each line of the syntax description. You will have to decide which symbols/variables in the language specification you want to define in the lexical analyzer and return to the parser, and which ones you want to be matched in YACC/Bison. In order to generate the codes.

(i) You will need to store the code generated at different grammar variables. The types of different grammar symbols must be appropriately declared. Try to adopt the syntax directed code generation schemes discussed in class.

(ii) For error in a specific line, it's not acceptable to have the program terminated. You have to implement simple error recovery mechanism, some related discussions can be found **here** and also **here**. (Initially you can skip this part)

Sample Input # 1
 	
if x < y ? x := y-1
:: x > y ? x := y+1
fi
	li \$a0 y
	move \$t1 \$a0
	li \$a0 x
	slt \$a0 \$a0 \$t1
	li \$t1 0
	beq \$a0 \$t1 L0
	li \$a0 1
	sw \$a0 0(\$sp)
	addiu \$sp \$sp-4
	li \$a0 y
	lw \$t1 4(\$sp)
	sub \$a0 \$a0 \$t1
	addiu \$sp \$sp 4
	b L2
	L0 :
	li \$a0 y
	move \$t1 \$a0
	li \$a0 x
	sgt \$a0 \$a0 \$t1
	li \$t1 0
	beq \$a0 \$t1 L1
	li \$a0 1
	sw \$a0 0(\$sp)
	addiu \$sp \$sp-4
	li \$a0 y
	lw \$t1 4(\$sp)
	add \$a0 \$a0 \$t1
	addiu \$sp \$sp 4
	b L2
	L1 :
	L2 :

Sample Input # 2	L1 : li \$a0 y

do x < y ? x := x+7.111 od	move $t1 $a0 li $a0 x slt $a0 $a0 $t1 li $t1 0 beq $a0 $t1 L0 li $a0 7.111 sw $a0 0($sp) addiu $sp $sp-4 li $a0 x lw $t1 4($sp) add $a0 $a0 $t1 addiu $sp $sp 4 b L1 L0 :

Sample Input # 5
 	
x:=true & false;
y:=myfunc:=1,4,3;
myfunc (x, y, z){
  if x>=0 ? m:=x*y*z fi;
  return m
}
	li $a0 True
	move $t1 $a0
	li $a0 False
	and $a0 $t1 $a0
	
	sw $fp 0($sp)
	addiu $sp $sp -4
	li $a0 3
	sw $a0 0($sp)
	addiu $sp $sp -4
	li $a0 4
	sw $a0 0($sp)
	addiu $sp $sp -4
	li $a0 1
	sw $a0 0($sp)
	addiu $sp $sp -4
	jal myfunc
	
	myfunc:
	move $fp $sp
	sw $ra 0($sp)
	addiu $sp $sp -4
	li $a0 0
	move $t1 $a0
	li $a0 x
	sge $a0 $a0 $t1
	li $t1 0
	beq $a0 $t1 L0

	li $a0 z sw $a0 0($sp) addiu $sp $sp-4 li $a0 y sw $a0 0($sp) addiu $sp $sp-4 li $a0 x lw $t1 4($sp) multu $a0 $t1 move $a0 $lo addiu $sp $sp 4 lw $t1 4($sp) multu $a0 $t1 move $a0 $lo addiu $sp $sp 4 b L1 L0 : L1 : lw $ra 4($sp) addiu $sp $sp 20 lw $fp 0($sp) jr $ra

Stack Printing:

Before Function Call:

$SP->

On Entry:

16($SP)->old fp

12($SP)->3

8($SP)->4

4($SP)->1

$SP->

Before Function Return:

20($SP)->old fp

16($SP)->3

12($SP)->4

8($SP)->1

4($SP)->fp->return address

$SP->

After Function Return:

$SP->