Project 4 description

Stage 1: Design the IRNode

IRNode have a basic format like this:

Opcode	First operand	Second operand	Result

The IRs that you will be using in this step will look like

ADDI	OP1	OP2	RESULT (Integer Add)
ADDF	OP1	OP2	RESULT (Floating Point Add)
SUBI	OP1	OP2	RESULT (Integer Subtract)
SUBF	OP1	OP2	RESULT (Floating Point Subtract: RESULT = OP1/OP2)
MULTI	OP1	OP2	RESULT (Integer Multiplication)
MULTF	OP1	OP2	RESULT (Floating Point Multiplication)
DIVI	OP1	OP2	RESULT (Integer Division)
DIVF	OP1	OP2	RESULT (Floating Point Division: RESULT = OP1/OP2)
STOREI	OP1		RESULT (Integer Store, Store OP1 to RESULT)
STOREF	OP1		RESULT (FP Store)
GT	OP1	OP2	LABEL (If OP1 > OP2 Goto LABEL)
GE	OP1	OP2	LABEL (If OP1 >= OP2 Goto LABEL)
LT	OP1	OP2	LABEL (If OP1 < OP2 Goto LABEL)
LE	OP1	OP2	LABEL (If OP1 <= OP2 Goto LABEL)
NE	OP1	OP2	LABEL (If OP1 != OP2 Goto LABEL)
EQ	OP1	OP2	LABEL (If OP1 = OP2 Goto LABEL)
JUMP			LABEL (Direct Jump)
LABEL			STRING (set a STRING Label)
READI			RESULT
READF			RESULT
WRITEI			RESULT
WRITEF			RESULT

(Note: here STORE means the same thing as MOVE in other IRs. See the examples.)

Stage 2: Create IRNode and place it in "list" and "graph"

In this step you will implement semantic routines to generate IRNodes and then you will put the IRNodes in two different data structures

The "list" is what you will need for code (tiny code) generation and "graph" is what you need in later steps to do data flow analysis.

While writing semantic routines you would have to introduce temporary variables to store temporary results from an arithmetic expression.

Example:

```
A + (B + C)

ADDI B C $T1     /* $T1 is a temporary variable */
ADDI A $T1 $T2
```

In many languages when you have a “float” and “int” operand mixed together in an expression, the result type becomes a “float”. If you need to get a new Temp variable to store the result, it will be of type “float” in that case. HOWEVER, as a simplification we don’t mix types in our micro language.

Example:

```
INT a;
FLOAT b, c;
c := a + b;

ADDF a b $T1     /* $T1 is a “float” */
STOREF $T1 c
```

Example: IF-ELSIF

```
INT a, b;
a := 2;
IF (a = 1)
    b := 1;
ELSIF (TRUE)
    b := 2;
ENDIF

STOREI 2 $T1        /* a := 2 */
STOREI $T1 a
STOREI 1 $T2         /* a = 1? */
NE a $T2 label1
STOREI 1 $T3         /* b := 1 */
STOREI $T3 b
JUMP label2          /* to out label */
LABEL label1        /* elsif label */
STOREI 1 $T4        /* TRUE can be handled by checking 1 = 1? */
STOREI 1 $T5
NE $T4 $T5 label3   /* jump to the next elsif label */
STOREI 2 $T6        /* b := 2 */
STOREI $T6 b
JUMP label2         /* jump to out label */
LABEL label3        /* extra elsif label, you can avoid it if you wish, but it is fine */
LABEL label2        /* out label */
```

Example: DO-WHILE

```
INT a, b;
a := 1;
b := 5;
 DO
 a := a * b;
 b := b – 1;
 WHILE (b >= 1);

 STOREI 1 $T1
 STOREI $T1 a
 STOREI 5 $T2
 STOREI $T2 b
 LABEL label1          /* top label */
 MULTI a b $T3
 STOREI $T3 a
 STOREI 1 $T4
 SUBI b $T4 $T5
 STOREI $T5 b
 STOREI 1 $T6
 LT b $T6 label2       /* if the condition is false, jump to out label */
 JUMP label1           /* otherwise jump to the start of the loop */
 LABEL label2          /* out label */
```