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)
GE     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)
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 Temp1
ADDI A Temp1 Temp2
```

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)

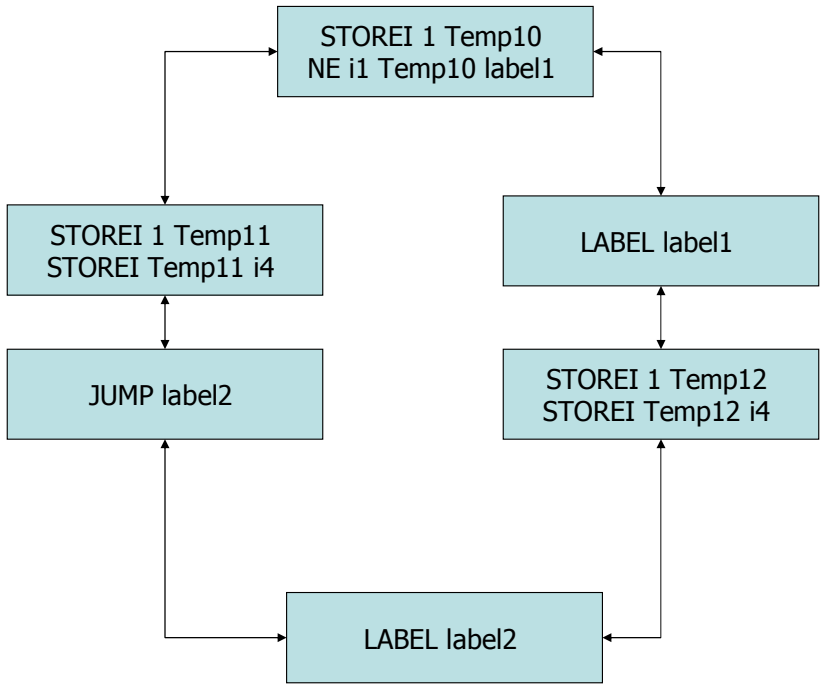
```
INT a;
FLOAT b,c;
c = a+b;
```

```
ADDF    a     b     Temp1                /* Temp1 is a “float” */
STOREF Temp1   c
```

Example: IF-ELSE

```
If (i1 = 1)
THEN      i4 := 1;
ELSE       i4 := 1;
ENDIF
```

```
STOREI 1 Temp10                 /* i1 == 1? */
NE i1 Temp10 label1
STOREI 1 Temp11                 /* i4 = 1; <THEN> part here */
STOREI Temp11 i4
JUMP label2                     /* to out label */
LABEL label1                    /* else label */
STOREI 1 Temp12                 /* i4 = 1; <ELSE> part here */
STOREI Temp12 i4
LABEL label2                    /* out label */
```



Example: REPEAT-UNTIL

```
REPEAT
        i4 := i4 *i1;
        i1 := i1 – 1;
UNTIL (i1 < 1);

LABEL label3                    /* top label */
MULTI i4 i1 Temp14              /* i4 = i4 * i1 */
STOREI Temp14 i4
STOREI 1 Temp15                 /* i1 = i1 -1 */
SUBI i1 Temp15 Temp16
STOREI Temp16 i1
STOREI 1 Temp13                 /* i1 <= 1 ? Exit */
GE i1 Temp13 label3             /* back to the loop start */
LABEL label4                    /* out label */
```

Stage 3; Tiny code generation

In this step you will traverse the IRNodes in "list" and generate tiny code.
The output of this step will be tested using "tinyR" which could use up to 1000 registers. Please download tinyR code from
http://cobweb.ecn.purdue.edu/~eigenman/ECE573/tiny/
(The executable only works on SUN workstations. Shay is a sun workstation.)

All temporary variables will be changed into registers. Write a function that returns new register names and call it when ever a new register is needed.

Sometimes normal variables will also be moved to registers due to the fact that tiny instructions have only two operands and the second has to be a register most of the time.
Example

```
        ADDI   A    B    Temp11
```

Becomes

```
        move    A    r14
        addi     r14    B
```

For "tiny" instruction set please read the "Tiny" page on the course webpage. (again, http://cobweb.ecn.purdue.edu/~eigenman/ECE573/tiny/)

```
/* Start of tiny code */

var f4
str eol "\n"
jmp main
label main
……
/* example of write */
sys writes f4
sys writes eol

/* end of tiny code */
sys halt
end
```

Example conversion (only the body of the function main is shown)

```
STOREI 1 Temp10
NE i1 Temp10 label1

STOREI 1 Temp11
STOREI Temp11 i4
JUMP label2
LABEL label1
STOREI 1 Temp12
STOREI Temp12 i4
LABEL label3
STOREI 1 Temp13
LE i1 Temp13 label4

MULTI i4 i1 Temp14
STOREI Temp14 i4

STOREI 1 Temp15
SUBI i1 Temp15 Temp16

STOREI Temp16 i1
JUMP label3
LABEL label4
LABEL label2
```

```
move 1 r7
cmpi i1 r7
jne label1
move 1 r8
move r8 i4
jmp label2
label label1
move 1 r9
move r9 i4
label label3
move 1 r10
cmpi i1 r10
jle label4
move i4 r11
muli i1 r11
move r11 i4
move 1 r12
move i1 r13
subi r12 r13
move r13 i1
jmp label3
label label4
label label2
```