Lab 3: A Calculator Compiler Middle End - Transformation and Evaluation

Lab 3 is due Wednesday 11:59PM May 15th. The late policy applies to this lab project.

This lab is an individual project. Get started early! The required format for lab reports can be found on the resource page.

Objective: The objective of this project is to implement a calculator compiler middle end transformation and evaluate the benefit/cost of the transformation.

Task 1: Generate Basic Blocks

Implement the basic block finding algorithm in your compiler, and print out all basic blocks in a program. The end of the basic block must be “goto”(s) to other basic blocks. For the above example program, an example output can be:

```
BB1:
        Tmp1 = C*1;
        A = B+Tmp1;
        B = D/2;
        C = B;
        D=4;
        If(D){
                goto BB2;
        } else {
                goto BB3;
        }

BB2:
        D=5;
        E=5;
        goto BB4;

BB3:
        E=0;
        goto BB4;

BB4:
        F=D+E;
```

Task 2: Performance Modeling

Calculate and report the number of cycles needed to run three-address code. Assume that independent statements can always be issued in the next cycle. For dependent statements, use the following latency table of operations. Latency is defined as the number of cycles that the operation can generate results. In other words, the dependent statements need to wait "latency-1" cycles before being issued.

Operations	**Latency (cycles)**
+, -	1
?	2
=	3
*, /	4
**	8

For example, the following code segment needs in total 5 cycles.

```
A = B*1
C = D+E
F = A+C
```

Task 3: CSE Optimization

(1) Implement the common subexpression elimination.
(2) Also write a process to iteratively apply either optimization, and be able to tell whether a fix point has been reached, i.e., not more optimization can be done.
(3) CSE is not always beneficial. That is, in some cases, it helps performance, while in others, may hurt performance. In this task, please develop and implement a heuristic of when to use CSE. And also develop a code sample whose performance may be degraded by CSE, while your heuristic can avoid such situation.
(4) Use the performance model from the previous task to report before/after performance in cycles.

For example, for the code segment:

```
S1: A = C*3;
S2: X = B+A;
S3: D = C*3;
S4: C = D/2;
```

An example output after the common subexpression elimination is:

```
S1: Tmp1 = C*3;
S2: A = Tmp1;
S3: X = B+A;
S4: D = Tmp1;
S5: C = D/2;
```

```
While(not fixed point){
        Do common subexpression elimination
}
```

What to Turn In

All project source files, **your test files** and report should be submitted to CANVAS by the deadline.