A4Q3

a) Constant folding is an optimization in which constant expressions are are recognized and evaluated during compile time, instead of during runtime. For example, for the expression "j = 1 + 2 + 3", the expression would be evaluated during compile time, and the value would be stored as a constant (in this case, 6). The evaluation occurs via the recognition of various structures present in the expression, such as the addition sign's, and the integer literals (1, 2, 3). This means that this expression (j = 1 + 2 + 3) is effectively "evaluated" during compile time, instead of having to evaluate it during runtime.

Constant propagation is an optimization in which the known constant values are substituted into expressions during compile time. For example, for "i = 2" where i is a known constant value, it will be substituted into the expression " i + 3 " during compile time, turning the expression "i + 3" into " 2 + 3".

(Stage 1)

```
1       a = 7
2       b = a * 52 * 10
3       x = 0
4       for c = 1 to 100
5               if (c + a * 100) < b
6                       x = x + c
7       return x
```

Constant propagation would propagate "a" from line 1 into line 2 and 5, yielding:

(Stage 2)

```
1       b = 7 * 52 * 10
2       x = 0
3       for c = 1 to 100
4               if (c + 7 * 100) < b
5                       x = x + c
6       return x
```

Continuing the propagation of "b" into line 4, and expressions in line 1 and 4, we have :

(Stage 3)

```
1       b = 3640
2       x = 0
3       for c = 1 to 100
4               if (c + 700) < 3640
5                       x = x + c
6       return x
```

Through the use of constant folding, we have computed b and a during compile time and are ready to use the above simplified code in runtime.

In MIPS, the optimization in going from stage 2 to stage 3 would be the evaluation of the expression b = 7 * 52 * 10 through the use of the "mult" instruction two times to store into the respective register that b represents. The for loop containing the variable c would have to be computed during runtime as it is dependant on the current iteration – it would simply be a register that increments its value by 1 for

every iteration using the instruction “addi” - there is no optimization here.