The Lazy Code Motion Algorithm should have the following desirable properties.

1. All redundant computations of expressions that can be eliminated without code duplication, are eliminated

2. The optimized does not perform any computation that is not in the original program.

   (anticipated expressions)

3. Expressions are computed at the latest possible time.

   This is important since the values of expressions are held in registers until they are used. Computing a value at the latest possible time minimizes its lifetime, which in turn minimizes register usage

   → less spillovers.

Partial Redundancy Elimination

Full Redundancy: Expression e is redundant at block B if along ALL paths reaching B, e has been evaluated and the operands of e have not been redefined subsequently

Partial Redundancy: Expression e is partially redundant in block B if along some path, e has been evaluated and the operands of e have not been redefined subsequently.

The Lazy Code Motion Algorithm attempts to render all partially redundant expressions, e, fully redundant by placing additional copies of e along paths leading to e.

We must ensure that for newly inserted expressions, they do not add any new operations.

i.e With or without the inserted expression, the computation would have been carried out "eventually"

We say that an expression e is anticipated at point p, if along all paths leading from p, e would have been computed with operand values at point p.

Algorithm Overview:

1. Find all expressions anticipated at each program point.

2. The second step places the computation where the values of the expressions are first anticipated along some path.

If we wish to place expressions at the earliest possible position, simply find program points where the expression is anticipated but not yet available.

→ earliest [B] = anticipated [B].in − available [B].in.

3. To minimize value lifetimes, we wish to delay the computation until the last possible moment.

An expression is postponable at a program point, if the expression has been anticipated and has yet to be used.

We place expressions at those program points where they can no longer be postponed.

4. Eliminate assignment to temporary variables only used once in the program.