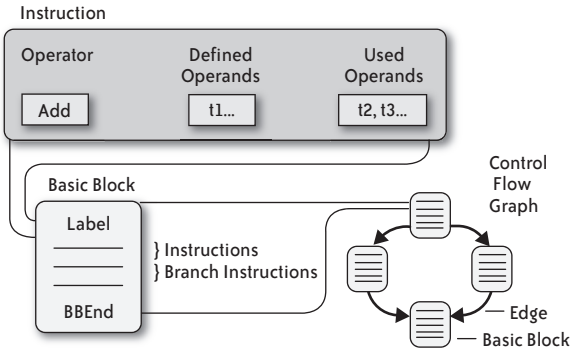

FIGURE 10-5. Overview of the optimizing compiler's intermediate representation

HIR

The High-level Intermediate Representation (HIR) is generated by a compiler phase, called BC2IR, that takes bytecode as input. This initial phase performs propagation based on the bytecode operand stack. Rather than generate separate operations, the stack is simulated and bytecodes are folded together to produce HIR instructions. The operators at the HIR level are equivalent to operations performed in bytecode but using an unbounded number of symbolic registers instead of an expression stack.

Once instructions are formed, they are reduced to simpler operations (if simpler operations exist). Following this, remaining call instructions are considered for inlining. The main part of the BC2IR phase is written recursively, so inlining performs a recursive use of the BC2IR phase.

Local optimizations are performed in the HIR phase. A local optimization is one that considers reducing the complexity of a basic block. As just a basic block is considered, the dependencies within the basic block are considered. This simplifies the compiler phase from having to consider the effect of branches and loops, which can introduce different kinds of data dependencies. Local optimizations include:

Constant propagation
: Propagating constant values avoids their placement in registers.

Copy propagation
: Replacing copies of registers with uses of the original register.

Simplification
: Reducing operations to less complex operations based on their operands.

Expression folding
: Folding trees of operations together. For example, "x=y+1; z=x+1" can have the expression "y+1" folded into "z=x+1", producing "x=y+1; z = y+2".