Common subexpression elimination
: Finding instructions that perform the same operation on the same operands and removing the latter with a copy of the former's result.

Dead code elimination
: If a register is defined and then redefined without an intervening use, then the instruction performing the original definition is dead code and can be removed.

Branch optimizations consider improving the control-flow graph so that the most likely paths are laid out in a manner that is optimal for the target of the compiler. Other branch optimizations look to remove redundant tests and unroll loops.

Escape analysis looks at whether objects can be accessed within just the context of the code being compiled and also whether an object that isn't just accessed in the local context can be shared among threads. If an object is just accessed locally, the requirement to allocate memory for it can be removed and the object's fields moved into registers. If an object is accessed in just one thread, then synchronization operations upon it can be removed.

A number of optimizations that use extra information added to the intermediate form are described later in "Scalar and extended array SSA forms."

LIR

The Low-level Intermediate Representation (LIR) converts the previous bytecode-like operations into more machine-like operations. Operations upon fields are changed into load and store operations; operators such as `new` and `checkcast` are expanded into calls into runtime services that provide those operations and are possibly inlined. Most of the optimizations applicable to HIR are also applicable to LIR, but optimizations such as escape analysis are not. Due to the kind of operations supported, there are some small differences between LIR for different architectures. As with HIR instructions, an unbounded number of symbolic registers can be used in LIR instructions.

MIR

Creating the final Machine-level Intermediate Representation (MIR) involves the three interdependent yet competing transformations of a compiler backend. The transformations are said to compete because their order can determine the performance of the generated machine code. In Jikes RVM, the first transformation is instruction selection. Instruction selection is the process whereby the RISC-like LIR instructions are converted to instructions that exist on the actual machine. Sometimes more than one instruction is required to perform an LIR instruction; for example, an LIR long addition requires two instructions on 32-bit architectures. At other times, the tree pattern-matching instruction selector, known as a *Bottom Up Rewrite System* (BURS), merges several instructions into one instruction. Figure 10-6 shows an example of pattern matching. Two patterns are found for Intel's IA32 architecture: one that