Target code generation for comparison operators

What code to generate for `arg1 <.int arg2`?

- produce zero or non-zero int value into some result register

MIPS: use an `slt` instruction to compute boolean-valued int result into a register

x86 (and most other machines): no direct instruction

Have comparison instructions, which set condition codes

- e.g. `cmpl %arg2, %arg1`

Later conditional branch instructions can test condition codes

- e.g. `jl`, `jle`, `jge`, `jg`, `je`, `jne` *label*

What code to generate?

Target code generation for comparison operators

```
Location emitIntLessThanValue(ILExpr arg1,
                              ILExpr arg2) {
  Location arg1_location = arg1.codegen(this);
  Location arg2_location = arg2.codegen(this);
  emitOp("cmpl",
         regOperand(arg2_location),
         regOperand(arg1_location));
  deallocateReg(arg1_location);
  deallocateReg(arg2_location);
  Location result_location =
    allocateReg(ILType.intILType());
  String true_label = getNewLabel();
  emitOp("jl", true_label);
  emitOp("movl", intOperand(0),
         regOperand(result_location));
  String done_label = getNewLabel();
  emitOp("jmp", done_label);
  emitLabel(true_label);
  emitOp("movl", intOperand(1),
         regOperand(result_location));
  emitLabel(done_label);
  return result_location;
}
```

Target code generation for branch statements

What code to generate for `iftrue` *test* `goto` *label*?

Target code generation for branch statements

```
void emitConditionalBranchTrue(ILExpr test,
                               ILLabel target){
  Location test_location = test.codegen(this);
  emitOp("cmpl", intOperand(0),
         regOperand(test_location));
  emitOp("jne", target.getName());
}
```

Target code generation for branch statements

What is generated for
  `iftrue arg1 <.int arg2 goto label`?

```
   <emit arg1 into %arg1>
   <emit arg2 into %arg2>
   cmpl %arg2, %arg1
   jl true_label
   movl $0, %res
   jmp done_label
 true_label:
   movl $1, %res
 done_label:

   cmpl $0, %res
   jne label
```

Can we do better?

Optimized target code generation for branches

Idea: boolean-valued IL expressions can be generated two ways, depending on their consuming context

- for their value
- for their "condition code"

Existing `codegen` operation on IL expression produces its value

New `codegenTest` operation on IL expression produces its condition code

- `X86ComparisonResultLocation` represents this result

Now conditional branches evaluate their test expression in the "for condition code" style

```
void emitConditionalBranchTrue(ILExpr test,
                               ILLabel target){
   Location test_location = test.codegen(this);
   X86ComparisonResultLoc cc =
     (X86ComparisonResultLoc) test_location;
   emitOp("j" + cc.branchTrueOp(),
          target.getName());
}
```

IL `codegenTest` default behavior

```
class ILExpr extends ILExpr {
   ...
   Location codegenTest(Target target) {
     return target.emitTest(this);
   }
}
```

In `X86Target` class:

```
Location emitTest(ILExpr arg) {
   Location arg_location = arg.codegen(this);
   emitOp("cmpl", intOperand(0),
          regOperand(arg_location));
   deallocateReg(arg_location);
   return new X86ComparisonResultLoc("ne");
}
```

IL `codegenTest` specialized behavior

```
class ILIntLessThanExpr extends ILExpr {
   ...
   Location codegenTest(Target target) {
     return target.emitIntLessThanTest(arg1,
                                       arg2);
   }
}
```

In `X86Target` class:

```
Location emitIntLessThanTest(ILExpr arg1,
                             ILExpr arg2) {
   Location arg1_location = arg1.codegen(this);
   Location arg2_location = arg2.codegen(this);
   emitOp("cmpl",
          regOperand(arg2_location),
          regOperand(arg1_location));
   deallocateReg(arg1_location);
   deallocateReg(arg2_location);
   return new X86ComparisonResultLoc("l");
}
```