Chapter 20

Optimization

Techniques

- Use `ldc` wherever possible
- Re-use temps
- Constant folding
- Register allocation
- Peephole optimization

R1a	R1 with `ldc` in place of `ld` wherever possible
R1b	R1a with temporary variable re-use
R1c	R1b with constant folding
R1d	R1c with register allocation
R1e	R1c with peephole optimization

Use ldc wherever possible

Replace

```
emitInstruction("ld", expVal);
```

with

```
emitLoad(expVal);
```

```
private void emitLoad(int opndIndex)
{
  if (st.isldcConstant(opndIndex)
    emitInstruction("ldc", st.getdwValue(opndIndex)):
  else
    emitInstruction("ld", opndIndex);
}
```

Re-using temporary variables

```
ld          b
add         c
st          @t0
ld          @t0     ; @t0 re-usable at this point
add         d
st          @t0     ; re-use @t0
ld          @t0
st          a
```

As soon as a temp is used, it can be re-used.

Handling temps

a)

```
1 public void freeTemp(int opndIndex)
2 {
3    if (st.isTemp(opndIndex))
4      tempIndex--;
5 }
```

b)

```
1 public void assign(int left, int expVal)
2 {
3    emitLoad(expVal);
4    freeTemp(expVal);
5    emitInstruction("st", left);
6 }
```

c)

```
1 public int add(int left, int right)
2 {
3    emitLoad(left);
4    emitInstruction("add", right);
5    freeTemp(left);
6    freeTemp(right);
7    int temp = getTemp();
8    emitInstruction("st", temp);
9    return temp;
10 }
```

Line 10: no constant folding

```
 1 private int termList(int left)
 2 {
 3   int right, temp, expVal;
 4
 5   switch(currentToken.kind)
 6   {
 7     case PLUS:
 8       consume(PLUS);
 9       right = term();
10       temp = cg.add(left, right);   // emits ld/add/st
11       expVal = termList(temp);
12       return expVal;
13     case RIGHTPAREN:
14     case SEMICOLON:
15       return left;     // do this at end of expression
16     default:
17       throw genEx("\"+\", \")\", or \";\"");
18   }
19 }
```

Constant folding: replace line 10 with

```
 1 if the left and right operands are both constants
 2 {
 3    set leftValue to int value of left operand
 4    set rightValue to int value of right operand
 5
 6    result = leftValue + rightValue;
 7    if (result >= 0)
 8      temp=st.enter("@" + result, "" + result, false);
 9    else
10      temp=st.enter("@_" + -result, "" + result, false);
11 }
12 else
13    temp = cg.add(left, right);
```

dw for constant only when used needed

```
1 private void emitInstruction(String op, int opndIndex)
2 {
3   if (st.isConstant(opndIndex))
4     st.setNeedsdw(opndIndex);
5   emitInstruction(op, st.getSymbol(opndIndex));
6 }
```

Register allocation

`ac` variable keeps track of what is in the `ac` register. Avoid unnecessary loads of the `ac` register.

Register allocation: assign method

```
1 public void assign(int left, int expVal)
2 {
3   if (ac != expVal)
4     emitLoad(expVal);
5   freeTemp(expVal);
6   emitInstruction("st", left);
7   ac = left;
8 }
```

Register allocation: add method

```
 1 public int add(int left, int right)
 2 {
 3   if (ac == left)
 4     emitInstruction("add", right);
 5   else
 6   if (ac == right)
 7     emitInstruction("add", left);
 8   else
 9   {
10     if (st.isTemp(ac))
11     {
12       emitInstruction("st", ac);
13       st.setNeedsdw(ac);
14     }
15     emitLoad(left);
16     emitInstruction("add", right);
17   }
18   freeTemp(left);
19   freeTemp(right);
20   int temp = getTemp();
21   ac = temp;
22   return temp;
23 }
```

Peephole optimization

Optimize based on the two instructions under the peephole.

Eliminates unnecessary `st-ld` sequences:

```
st    x        replace with    st    x
ld    x

st    @t0      omit entirely
ld    @t0
```

Emit actions go through the peephole

```
 1 public int add(int left, int right)
 2 {
 3   if (!st.isTemp(left) && st.isTemp(right))
 4   {
 5     peephole("ld", right);
 6     peephole("add", left);
 7   }
 8   else
 9   {
10     peephole("ld", left);
11     peephole("add", right);
12   }
13   freeTemp(left);
14   freeTemp(right);
15   int temp = getTemp();
16   peephole("st", temp);
17   return temp;
18 }
```

Emit actions go through the peephole

```
1 public void println(int expVal)
2 {
3   peephole("ld", expVal);
4   freeTemp(expVal);
5   peephole("dout");
6   peephole("ldc", "'\\n'");
7   peephole("aout");
8 }
```

Principal peephold method

```
 1  public void peepHole(String op, int opndIndex)
 2  {
 3     String opnd = symTab.getSymbol(opndIndex);
 4
 5     // replace ld with ldc if possible
 6     if (opnd is a constant)
 7     {
 8        if (
 9              current instruction is a ld
10                       and
11              the constant is in the range 0-4095
12           )
13        {
14           set op to ldc
15           set opnd to symTab.getdwValue(opndIndex);
16        }
17        else
18           set needsdw to true for the constant
19     }
```

Principal peephold method continued

```
21      // check if okay not to emit current instruction
22      if (
23              previous op is st and current op is ld and operands match
24                                          or
25              current instruction is a st into a temp
26          )
27          don't emit current instruction
28      else
29      {
30          // must emit previous inst if st into temp
31          if previous instruction is a st into a temp
32          {
33              emit previous instruction
34              set needsdw to true for temp in previous instruction
35          }
36          emit current instruction
37      }
38
39      // save current instruction
40      previousOp = op;
41      previousOpnd = opnd;
42      previousOpndIndex = opndIndex;
43  }
```

peephole method overloaded

```
 1    public void peephole(String op, String opnd)
 2    {
 3        if previous instruction st into a temp
 4        {
 5            emit previous instruction
 6            set needsdw to true for temp in previous instruction
 7        }
 8
 9        emit current instruction
10
11        previousOp = op;
12        previousOpnd = opnd;
13        previousOpndIndex = -1;
14    }
```

peephole method overloaded

```
1     public void peephole(String op)
2     {
3        peephole(op, "");
4     }
```