CSE443 Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu

343 Davis Hall
Phases of a compiler

Figure 1.6, page 5 of text

Target machine code generation
getReg function

\[ x = y \text{ op } z \]

How do we do this?
getReg function

\[ x = y \ op \ z \]

"1. If \( y \) is currently in a register, pick a register already containing \( y \) as \( R_y \). Do not issue a machine instruction to load this register, as none is needed.

2. If \( y \) is not in a register, but there is a register currently empty, pick one such register as \( R_y \). [LD \( R_y, y \)]
3. The difficult case occurs when y is not in a register, and there is no register that is currently empty. We need to pick one of the allowable registers anyway, and we need to make it safe to reuse. Let R be a candidate register, and suppose v is one of the variables that the register descriptor for R says is in R. We need to make sure that v's value either is not really needed, or that there is somewhere else we can go to get the value of v. The possibilities are:
getReg function
\[ x = y \text{ op } z \]

(a) If the address descriptor for \( v \) says that \( v \) is somewhere else besides \( R \), then we are OK.
(b) If \( v \) is \( x \), the variable being computed by instruction \( I \), and \( x \) is not also one of the other operands of instruction \( I \) (\( z \) in this example), then we are OK. The reason is that in this case we know this value of \( x \) is never again going to be used, so we are free to ignore it.
getReg function

\[ x = y \text{ op } z \]

(c) Otherwise, if \( v \) is not used later (that is, after the instruction \( I \), there are no further uses of \( v \), and if \( v \) is live on exit from the block, then \( v \) is recomputed within the block), then we are OK.
getReg function

\[
x = y \text{ op } z
\]

(d) If we are not OK by one of the first three cases, then we need to generate the store instruction \( ST \ v, R \) to place a copy of \( v \) in its own memory location. This operation is called a spill." [p. 647-648]
getReg function

\[ x = \text{op} \ y \]

Repeat the above (a) - (d) steps for each variable \( v \) currently in \( R \).

Let the score of \( R \) be the number of ST instructions generated. Choose the \( R \) with lowest score to actually use.
getReg function

\[ x = y \text{ op } z \]

We also need a register for the result, Rx. "The issues and options are almost as for y, so we shall only mention the differences.

1. Since a new value of x is being computed, a register that holds only x is always an acceptable choice for Rx. This statement holds even if x is one of y and z, since our machine instructions allow two registers to be the same in one instruction.
getReg function

\[ x = y \text{ op } z \]

2. If \( y \) is not used after instruction I, in the sense described for variable \( v \) in item (3c), and \( Ry \) holds only \( y \) after being loaded, if necessary then \( Ry \) can also be used as \( Rx \). A similar option holds regarding \( z \) and \( Rz \).” [p. 548]
Phases of a compiler

Figure 1.6, page 5 of text

Optimizations
Algebraic Identities [p. 536]

\[ x + 0 = 0 + x = x \]
\[ x \times 1 = 1 \times x = x \]
\[ x - 0 = x \]
\[ x / 1 = x \]
Algebraic Identities [p. 536]

\[ x^2 = x \times x \]
\[ 2 \times x = x + x \]
\[ x / 2 = x \times 0.5 \]

Can also use left and right for integers
(But see [https://en.wikipedia.org/wiki/Arithmetic_shift](https://en.wikipedia.org/wiki/Arithmetic_shift))
Algebraic Identities [p. 536]

Constant folding

"...evaluate constant expressions at compile time and replace the constant expressions by their values."
See footnote 2:

"Arithmetic expressions should be evaluated the same way at compile time as they are at run time. K. Thompson has suggested an elegant solution to constant folding: compile the constant expression, execute the target code on the spot, and replace the expression with the result. Thus, the compiler does not need to contain an interpreter."
Peephole optimization
[p 549]

"The peephole is a small, sliding window on a program." [p. 549]

"In general, repeated passes over the target code are necessary to get the maximum benefit." [p. 550]
Peephole optimization: redundant LD/ST

LD RO, a
ST a, RO

If the ST instruction has a label, cannot remove it. (If instructions are in the same block we're OK.)
if $E=K$ goto L1
goto L2
L1: ...
... 
L2: ...
...

Peephole optimization: unreachable code

Suppose $K$ is a constant.

This case takes several slides...
Peephole optimization: unreachable code

if E=K goto L1
goto L2
L1: ...do something...
...
L2: ...do something...
...

Eliminate jumps over jumps
Peephole optimization: unreachable code

if $E=K$ goto L1
goto L2
L1: ...
...
L2: ...
...

if $E!=K$ goto L2
L1: ...
...
L2: ...
...

Eliminate jumps over jumps
Peephole optimization: unreachable code

if E=K goto L1
  goto L2
L1: ...
  ...
L2: ...
  ...

if E!=K goto L2
  ...
  ...
  L2: ...
  ...

If there are no jumps to L1, we can remove label
Peephole optimization: unreachable code

If $E$ is set to a constant value other than $K$, then...

```plaintext
if $E$=K goto L1
  goto L2
L1: ...
L2: ...
```

```plaintext
if $E$!=K goto L2
  ...
L2: ...
```
Peephole optimization: unreachable code

if \( E = K \) goto L1

goto L2

L1: ...

... conditional jump becomes unconditional ...

L2: ...

if true goto L2

...
Peephole optimization: unreachable code

```
if E=K goto L1
goto L2
L1: ...
...
L2: ...
...
goto L2
...
...
```

...and the unreachable code can be removed.
Peephole optimization: flow-of-control

```
goto L1
...
L1: goto L2
...
L2:
```
Peephole optimization: flow-of-control

goto L1
...
L1: goto L2
...
L2:

L2:
...
L1: goto L2
...
L2:
If there are no jumps to L1, and L1 is preceded by an unconditional jump...
Peephole optimization: flow-of-control

...then we can eliminate the statement labelled L1
Peephole optimization: flow-of-control

if \( a < b \) goto L1

... L1: goto L2

... L2:

if \( a < b \) goto L2

... L2:

... similar arguments can be made for conditional jumps.