HW4

- Plan to drop, unless there's an objection.

- Distribute points evenly to other 3 HWs.
Phases of a compiler

Figure 1.6, page 5 of text
getReg function

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

How do we do this?
getReg function

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

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 \( R_y \). \([LD \ R_y, y]\)
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.
getReg function
\[ x = y \text{ op } z \]

(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
\[ x = y \ op \ z \]

(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. 547-548]
getReg function

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

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, \( R_x \).

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

Figure 1.6, page 5 of text
gcc -fno-asynchronous-unwind-tables -O0 -S foo.c