Plan for today

Finish possible improvements over spill all
- register allocation for Tree.Exp

Review predictive parsing
- first and follow sets
- predictive parsing table
- predictive parsing code
- error recovery for predictive parsing

Possible improvements over spill all

Linear passes over the Assem.Instrs
- after spill all, remove loads from sw-lw pairs where the temp and frame location are the same
  eg.  
  sw $t2, -16($fp)  
  lw $t2, -16($fp)

- before spill all, assign each Temp to a callee-saved register until run out
  eg.  
  t34 => frame.RA  
  t35 => frame.S0

... calleeSaves = { RA, S0, S1, S2, S3, S4, S5, S6, S7}

- after spill all, assign each frame location to a callee-saved register until run out
  eg.  
  $fp-12 => frame.RA  
  $fp-16 => frame.S0

... calleeSaves = { RA, S0, S1, S2, S3, S4, S5, S6, S7}

Possible improvements over spill all cont...

Register allocation for expression Trees
- assign Temps associated with machine registers to intermediate results within an expression tree
- indicate to spillAll that those Temps should not be spilled
- here can use caller-saved registers as well as callee-saved registers since Exp.CALL won’t be an intermediate expression tree

Reviewing predictive parsing

Write a predictive parser using panic mode error recovery for the grammar shown below.

\[
(0) S \rightarrow E \$
(1) E \rightarrow B E'
(2) E' \rightarrow t \text{ or } B E'
(3) E' \rightarrow \epsilon
(4) B \rightarrow t
(5) B \rightarrow f
\]

<table>
<thead>
<tr>
<th></th>
<th>FIRST</th>
<th>nullable</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>t</td>
<td>f</td>
<td>no</td>
</tr>
<tr>
<td>E'</td>
<td>or</td>
<td>yes</td>
<td>$</td>
</tr>
<tr>
<td>E</td>
<td>t</td>
<td>f</td>
<td>no</td>
</tr>
<tr>
<td>S</td>
<td>t</td>
<td>f</td>
<td>no</td>
</tr>
</tbody>
</table>
Predictive parsing table

Building a predictive parsing table

- one row for each nonterminal, one column for each terminal
- production “X -> gamma” should be in row X, column T for each T in
  first(gamma)
- if gamma is nullable, put “X -> gamma” in row X, column T for each T in
  follow(X)

Predictive parser implementation

Implementing a predictive parser

- write a function for each nonterminal X with a switch on the current input token
- switch statements will have a case for each terminal T, where row X and column T
  in the parsing table contains a production rule
- in function X, case T, nonterminal functions and eat(token) calls will parse the
  right-hand-side of the production in row X and column T of the parsing table

```c
void eat(t) { if (tok==t) advance(); else error(); }
void advance() { tok=getToken(); }
void S() { switch(tok) {
  case t: case f: E(); eat($); break;
  default: error();
}
```