Assignment Revisited

```
assignmentOperator(struct item_t* leftItem,
                   struct item_t* rightItem) {
 if (leftItem->type != rightItem->type)
  warning("type mismatch in assignment");

  if (rightItem->type == BOOL_TYPE)
   unloadBool(rightItem);

  load(rightItem);

 // leftItem must be in VAR_MODE or REF_MODE
 // rightItem must be in REG_MODE
 put(STW, rightItem->reg, leftItem->reg, leftItem->offset);

 if (leftItem->mode == REF_MODE)
  releaseRegister(leftItem->reg);

 releaseRegister(rightItem->reg);
}

unloadBool(struct item_t* item) {
 if (item->mode == COND_MODE) {
  cJump(item);

  fixLink(item->tru);

  item->mode = REG_MODE;

  // assumes: reg[0]==0 for MOVI semantics
  put(ADDI, item->reg, 0, 1);
  put(BR, 0, 0, 2);

  fixLink(item->fls);

  // assumes: reg[0]==0 for MOVI semantics
  put(ADDI, item->reg, 0, 0);
 }
```

explained on the next page

jump here

true

false

jump here if condition evaluates to false!

OR

```
loadBool(struct item_t* item) {
 if (item->mode != COND_MODE) {
  load(item);

  item->mode = COND_MODE;
  item->operator = NEQ;
  item->fls = 0;
  item->tru = 0;
 }
}
```

branch if loaded variable is true!

called on left operand of ||

```
simpleExpressionOR(struct item_t* item) {
 if (item->type == BOOL_TYPE) {
  loadBool(item);

  put(branch(item->operator), item->reg, 0, item->tru);
  releaseRegister(item->reg);
  item->tru = PC - 1;
  fixLink(item->fls);
  item->fls = 0;
 } else error("boolean expression expected");
}
```

remember for fixup

jump here if expression so far evaluates to false!

called on both operands of || (and +,-)

```
simpleExpressionBinaryOperator(struct item_t* leftItem,
                               struct item_t* rightItem,
                               int operatorSymbol) {

 if (operatorSymbol == OR) {
  if ((leftItem->type == BOOL_TYPE) &&
      (rightItem->type == BOOL_TYPE)) {
   loadBool(rightItem);

   leftItem->reg = rightItem->reg;
   leftItem->fls = rightItem->fls;
   leftItem->tru = concatenate(rightItem->tru, leftItem->tru);
   leftItem->operator = rightItem->operator;
  } else error("boolean expressions expected");
 } else if ((leftItem->type == INT_TYPE) &&
            (rightItem->type == INT_TYPE)) {
... as before
```

leftItem->fls is already fixed

collect for fixup

AND NOT

called on both operands of && (and *, /, %)

```
termOperator(struct item_t* leftItem,
             struct item_t* rightItem,
             int operatorSymbol) {
 if (operatorSymbol == AND) {
  if ((leftItem->type == BOOL_TYPE) &&
      (rightItem->type == BOOL_TYPE)) {
   loadBool(rightItem);

   leftItem->reg = rightItem->reg;
   leftItem->fls = concatenate(rightItem->fls, leftItem->fls);
   leftItem->tru = rightItem->tru;
   leftItem->operator = rightItem->operator;
  } else error("boolean expressions expected");
 } else if ((leftItem->type == INT_TYPE) &&
            (rightItem->type == INT_TYPE)) {
...
```

dual to ||

called on negation of factor

```
factorOperator(struct item_t* item) {
 int tmp;

 if (item->type == BOOL_TYPE) {
  loadBool(item);

  tmp = item->fls;
  item->fls = item->tru;
  item->tru = tmp;
  item->operator = negate(item->operator);
 } else error("boolean expression expected");
}
```

no code generation for negation!