3. Flow of Control

In this chapter we temporarily leave behind the "operating system" of the lisp machine and examine aspects of the lisp language itself. In particular, we'll look at the various constructs for determining the flow of control. The notes are a little sketchier than usual, because this material is covered reasonably well in the Symbolics documentation.

3.1 Conditionals

All non-trivial lisp programs execute only those "statements," or forms, which are appropriate to their circumstances. They specify which forms to execute using conditional special operators. To test whether a condition is true, a predicate function is used.

Virtually all conditionals test the truth or falsehood of the conditions by comparing the result of the predicate with nil: if the predicate returns nil, the condition it is testing is false; any-
thing else is considered to be true.¹

Let's start with a few simple conditional operators. The when special operator tests the result of evaluating first “argument,” which is usually a predicate form; if it is nil, the when form returns nil without evaluating any of the rest of its subforms. If it's not nil, the rest of the forms are evaluated, and the value of the last one is returned.² For example:

```
(when (< n 0)
  (format t "N is negative")
  n)
```

will print “N is negative” when it is, and return the value of n. If n is zero or positive, this form will return nil. The special operator unless is just like when, except it inverts the

---

¹When there is no other, more obvious non-nil value to return, most predicates return the special symbol t. However, any value other than nil means “true.”

²I have quoted the word “argument” to show the distinction between normal functions and special operators. Functions are called with arguments, which have already been passed through eval. Special operators, on the other hand, aren't called with all their subforms already evaluated. Rather, special operators get to decide which, if any, of their subforms get evaluated. when, for example, always evaluates its first subform; only if it returns some non-nil value do the rest of the subforms get evaluated.