==========

Sequencing

execute one statement after another

very straightforward; very imperative

Selection

sequential if statements

```
if ... then ... else

if ... then ... elsif ... else

(cond
   (C1) (E1)
   (C2) (E2)
   ...
   (Cn) (En)
   (T) (Et)
)

match e with
| pat1 when cond1 –>
| pat2 when cond2 –>
| ...
| patN when condN –>
| _            –>
```

value of explicit terminators or `begin`/`end` (or {}) brackets

need for `elsif` (`elif`)

jump code

When translating

```
if A < B then … else … fi
```

one might evaluate the condition to get a Boolean value in a register, then branch depending on its value.

That's often more instructions than needed:

```
   r1 := A
   r2 := B
   r1 := r1 < r2
   if !r1 goto L1
      <then clause>
      goto L2
   L1:
      <else clause>
   L2:
```

v.

```
   r1 := A
   r2 := B
   if r1 >= r2 goto L1
      <then clause>
      goto L2
   L1:
      <else clause>
   L2:
```

For expressions with short-circuiting, the difference is more compelling (Example 6.49 in the text):

```
   if ((A > B) and (C > D)) or (E <> F) then
       then_clause
   else
      else_clause
```

w/out short-circuiting (as in, e.g., Pascal):

```
      r1 := A            -- load
      r2 := B
      r1 := r1 > r2
      r2 := C
      r3 := D
      r2 := r2 > r3
      r1 := r1 & r2
      r2 := E
      r3 := F
      r2 := r2 <> r3
      r1 := r1 | r2
      if r1 = 0 goto L2
   L1: <then clause>     -- label not actually used
      goto L3
   L2: <else clause>
   L3:
```

with short-circuiting (as in, e.g., C):

```
      r1 := A
      r2 := B
      if r1 <= r2 goto L4
      r1 := C
      r2 := D
      if r1 > r2 goto L1
   L4: r1 := E
      r2 := F
      if r1 = r2 goto L2
   L1: then_clause
      goto L3
   L2: else_clause
   L3:
```

Note that this not only avoids performing unnecessary comparisons; it also avoids the ***and*** and ***or*** instructions.

guarded commands

example of non-determinacy

```
if
   cond1 -> stmt1
[] cond2 -> stmt2
   ...
[] condN -> stmtN
fi
```

do cond → stmt

od

similar version for loops

Fortran computed `gotos`

`case`/`switch` (introduced in Algol-W)

labels required to be disjoint

what should happen if there isn't a matching label for value?

Ada: forbid at compile time

C: no-op

Pascal: dynamic semantic error

`case` implementation

sequential testing

small number of choices, non-dense range

characteristic array (jump table)

dense range

hashing

non-dense range w/out range labels

binary search

large range, range labels

(probably don't need search tree, except perhaps if the key distribution is highly nonuniform and we want better pivots than we get with mean)

Should ranges be allowed in the label list?

they make it easy to state things for which a jump table or hash table is awful: can be done efficiently (O(log n)) with binary search

examples:

3:	1:	1:	1..48:
5:	2:	59:	97..283:
7:	3:	187:	900..1024:
9:	...	...	...
	100:	1000000:	12345..67890: