2.4.8 Control Instructions

Instructions in this group affect the flow of control.

$$
\begin{array}{lcl}
\mathit{blocktype} & ::= & \mathit{typeidx} \mid \mathit{valtype}^{?} \\
\mathit{instr} & ::= & \dots \\
& \mid & \mathsf{nop} \\
& \mid & \mathsf{unreachable} \\
& \mid & \mathsf{block}\ \mathit{blocktype}\ \mathit{instr}^{*}\ \mathsf{end} \\
& \mid & \mathsf{loop}\ \mathit{blocktype}\ \mathit{instr}^{*}\ \mathsf{end} \\
& \mid & \mathsf{if}\ \mathit{blocktype}\ \mathit{instr}^{*}\ \mathsf{else}\ \mathit{instr}^{*}\ \mathsf{end} \\
& \mid & \mathsf{br}\ \mathit{labelidx} \\
& \mid & \mathsf{br\_if}\ \mathit{labelidx} \\
& \mid & \mathsf{br\_table}\ \mathit{vec}(\mathit{labelidx})\ \mathit{labelidx} \\
& \mid & \mathsf{return} \\
& \mid & \mathsf{call}\ \mathit{funcidx} \\
& \mid & \mathsf{call\_indirect}\ \mathit{tableidx}\ \mathit{typeidx}
\end{array}
$$

The nop instruction does nothing.

The unreachable instruction causes an unconditional trap.

The block, loop and if instructions are *structured* instructions. They bracket nested sequences of instructions, called *blocks*, terminated with, or separated by, end or else pseudo-instructions. As the grammar prescribes, they must be well-nested.

A structured instruction can consume *input* and produce *output* on the operand stack according to its annotated *block type*. It is given either as a type index that refers to a suitable function type, or as an optional value type inline, which is a shorthand for the function type $[] \to [\mathit{valtype}^{?}]$.

Each structured control instruction introduces an implicit *label*. Labels are targets for branch instructions that reference them with label indices. Unlike with other index spaces, indexing of labels is relative by nesting depth, that is, label 0 refers to the innermost structured control instruction enclosing the referring branch instruction, while increasing indices refer to those farther out. Consequently, labels can only be referenced from *within* the associated structured control instruction. This also implies that branches can only be directed outwards, "breaking" from the block of the control construct they target. The exact effect depends on that control construct. In case of block or if it is a *forward jump*, resuming execution after the matching end. In case of loop it is a *backward jump* to the beginning of the loop.

Note: This enforces *structured control flow*. Intuitively, a branch targeting a block or if behaves like a break statement in most C-like languages, while a branch targeting a loop behaves like a continue statement.

Branch instructions come in several flavors: br performs an unconditional branch, br_if performs a conditional branch, and br_table performs an indirect branch through an operand indexing into the label vector that is an immediate to the instruction, or to a default target if the operand is out of bounds. The return instruction is a shortcut for an unconditional branch to the outermost block, which implicitly is the body of the current function. Taking a branch *unwinds* the operand stack up to the height where the targeted structured control instruction was entered. However, branches may additionally consume operands themselves, which they push back on the operand stack after unwinding. Forward branches require operands according to the output of the targeted block's type, i.e., represent the values produced by the terminated block. Backward branches require operands according to the input of the targeted block's type, i.e., represent the values consumed by the restarted block.

The call instruction invokes another function, consuming the necessary arguments from the stack and returning the result values of the call. The call_indirect instruction calls a function indirectly through an operand indexing into a table that is denoted by a table index and must have type funcref. Since it may contain functions of heterogeneous type, the callee is dynamically checked against the function type indexed by the instruction's second immediate, and the call is aborted with a trap if it does not match.