Code Generation for Control Structures

Sequential Composition

```
How to compile statement sequence?
s1: s2: ...; sN
```

Concatenate byte codes for each statement!
 def compileStmt(e: Stmt): List[Bytecode] = e match {

Compiling Control: Example

```
(func $func0
                                           (param $var0 i32) (param $var1 i32)
int count(int counter,
                                           (param $var2 i32) (result i32)
                                           (local Svar3 i32)
           int to,
                                           i32.const 0
           int step) {
                                           set local $var3
 int sum = 0:
                                           loop $label0
                                            get local $var3
 do {
                                            get_local $var0
   counter = counter + step;
                                            get local $var2
   sum = sum + counter:
                                            i32.add
                                            tee local $var0
 } while (counter < to):
                                            i32.add
 return sum; }
                                            set local $var3
                                            get_local $var0
We need to see how to:
                                            get local Svar1
                                            i32.lt s

    translate boolean expressions

                                            br if $label0

    generate jumps for control

                                           end Slabel0
```

get local \$var3)

Representing Booleans

"All comparison operators yield 32-bit integer results with 1 representing true and 0 representing false." – WebAssembly spec

Our generated code uses 32 bit int to represent boolean values in: **local variables**, **parameters**, and intermediate **stack values**.

- 1, representing true
- 0, representing false

i32.eq: sign-agnostic compare equal

i32.ne: sign-agnostic compare unequal

i32.lt s: signed less than

i32.le_s: signed less than or equal

i32.gt_s: signed greater than

i32.ge_s: signed greater than or equal

i32.eqz: compare equal to zero (return 1 if operand is zero, 0 otherwise) // not

Truth Values for Relations: Example

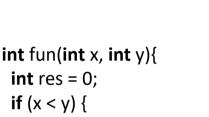
```
(func $func0
 (param $var0 i32)
```

(param \$var1 i32) int test(int x, int y){ (result i32) return (x < y); get local \$var0 get local \$var1 i32.lt s

Comparisons, Conditionals, (local \$var2 i32) block \$label1 block \$label0

```
Scoped Labels
 int res = 0:
```

```
res = (v / x):
} else res = (x / y);
return res+x+v;
```





i32.div s

end Slabel1

get local \$var1 get local Svar0 i32.add get local \$var2 i32.add

get local \$var0 get local \$var1

// done with if get local \$var0 get local \$var1 set local Svar2

// else branch

// end of if

Main Instructions for Labels

- block: the beginning of a block construct, a sequence of instructions with a label at the end
- loop: a block with a label at the beginning which may be used to form loops
- **br**: branch to a given label in an enclosing construct
- br_if: conditionally branch to a given label in an enclosing construct
 - return: return zero or more values from this function
 - **end**: an instruction that marks the end of a block, loop, if, or function

Compiling If Statement

```
block $label1 block $label0
                                             (negated condition code)
                                            br if $label0
                                                            // to else branch
Notation for compilation:
                                               (true case code)
[ if (cond) tStmt else eStmt ] =
                                            br $label1
                                                             // done with if
                                            end $label0
                                                             // else branch
        block SnAfter block SnElse
                                               (false case code)
        [!cond]
                                            end $label1
                                                             // end of if
        bf if $nElse
        [tStmt]
        br SnAfter
end SnElse:
       [eStmt]
end $nAfter:
```

Is there alternative without negating condition?

How to introduce labels

For forward jumps to \$label: use
 block \$label

...

end \$label

For backward jumps to \$label: use loop \$label

•••

end \$label