What would we need to add to introduce *loops* into our language?

(let (result (loop ...))
  ... use result ...)

Concrete Syntax – what new syntactic forms are in the example below?

```
expr := ...
      | (loop <expr>)          means run subexpression repeatedly
      | (break <expr>)         means exit the current loop with the given value
      | (block <expr> ...)     means eval exprs in order, answer is last expr
      | (set! <id> <expr>)     means change the id on stack to expr (ans is expr)
```

Examples (can you think of any without `set!` or `block`?)

```
(let (sum 0)
  (let (i 10)
    (loop
      (if (= i 0) (break sum)
          (block
            (set! sum (+ sum i))
            (set! i (+ i -1)))))))
```

Answer: 55
(10+9+...)

Generated code for examples

Abstract Syntax

Sketch of compiler code changes

compile_expr(..., cur_brk: &str) (String)

Break(expr) ⇒

jmp {cur_brk}

If we're OK with 63-bit numbers, can use LSB for *tag*

```
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1010  = 5
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001  = false
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011  = true
```

```
cmp <reg>, <val>          compute <reg> - <val> and set condition codes (value in <reg> does not change)

test <reg>, <val>         perform bitwise and on the two values, but don't change <reg>, and set condition
                          codes as appropriate. Useful for mask checking. test rax, 1 will set Z to true
                          if and only if the LSB is 0

<label>:                  set this line as a label for jumping to later

jne <label>               jump to <label> if Zero is not set (last cmped values not equal)
je <label>                jump to <label> if Zero is set (last cmped values are equal)
jge <label>               jump to <label> if Overflow is the same as Sign (which corresponds to >= for last cmp)
jle <label>               jump to <label> if Zero set or Overflow != Sign (which corresponds to <= for last cmp)
```

```
mov [rsp-8], 0          sum:[8]
mov [rsp-16], 20        i:[16]
loopstart:
    cmp [rsp-16], 0
    jne else1
        mov rax, [rsp-8]
        jmp loopend          (break sum)
    jmp ifend
else1:
        mov rax, [rsp-8]
        mov [rsp-24], rax
        mov rax, [rsp-16]
        add rax, [rsp-24]
        mov [rsp-8], rax     (set! ...)
        mov rax, [rsp-16]
        mov [rsp-24], rax
        mov rax, -2
        add rax, [rsp-24]
        mov [rsp-16], rax    (set! ...)
ifend:
    jmp loopstart
loopend:
```

(if ...)