Bluebird

64 bits:
18446744073709551616

We will assume (only for Bluebird) that the programmer gives right types for each operation.
(no "true + true")

```
expr ::= ....
       | true
       | false
       | ....
       | if <expr> then <expr> else <expr>
```

(1) Repurpose some bits:

sign bit → 0 0000000 00000000 00000000 0000000 00000000 00000000 00000000 00001000

Ashlet $2^3 2^2 2^1 2^0$ — $2^2 2^1 2^0$ ↑ tag

Not only 3 — Not exclusive

(2) Be like C —— maybe I don't differentiate
or like C++ —— use type information

(3) Be like Python —— use more space   int("5")
1+2   --plus--

Choosing binary representation

If last bit is 0 then value is a 62-bit signed integer
If last bit is 1 then (for now) value is a boolean

for lecture (not lab) {
true = 0x8000000000000001
false = 0x0 ———— 01

There is a blob of code in the writeup

AsmMov (ArgRegister RAX, ArgConstant [ ])

Bluebird 4 in binary:
0 ———— 0 1000

In Bluebird:

compile (4) → mov rax, 8

compile (true) → mov rax, 0x80 ———— 01 ✓

AsmMov (ArgRegister RAX, ArgConstant "0x8000000000000001")

Interacting with Representation

4+2

Normal Asm Rep

```
mov rax, 4
add rax, 2
```

rax = 6

Bluebird Rep

```
mov rax, 8
add rax, 4
```

rax = 12

$R(n)$ = representation of n

(n: actual number; representation: how it will appear in reg)

$$R_{Bluebird}(n) = 2n$$

$x + y = z$

$R(x) + R(y) = R(z)$

$2x + 2y = 2z$ ✓

$2(x+y) = 2z$

$x - y = z$

$R(x) - R(y) = R(z)$ ✓

$x * y = z$

$R(x) * R(y) = R(z)$ ✗

$R(x) * R(y) = 2x * 2y = 4xy = 2(R(xy))$

asm

```
mov rax, 4
imul rax, 4
```

rax = 16

bb

```
mov rax, 8
imul rax, 8
```

rax = 64 !!

Proposal: fix by dividing by 2

What if $2(R(xy))$ overflows?

$$\frac{2x * 2y}{2} = \boxed{\frac{2x}{2} * 2y}$$

63-bit signed: $-2^{62} \ldots 2^{62} - 1$

$4,611,686,018,427,387,903$

Bluebird: $2000000000 * 2000000000 \Rightarrow 4,000,000,000,000,000,000$

```
mov rax, 8 ; BB 4
sar rax, 1 ; div by 2
imul rax, 8 ; BB 4
```

rax = 32

BB 16

```
expr ::= ...
       | isbool(<expr>)
       | isint(<expr>)
       | ...
```

isbool(5) ⟹ false

```
mov rax, 10
and rax, 0x1
shl rax, 63
or  rax, 0x1
```

```
0x0 ——— 00
; 0x0 ——— 01
; 0x0 ——— 0
; 0x80 ——— 0
; 0x0 ——— 01
  0x80 ——— 01
```

< >

just use jumps