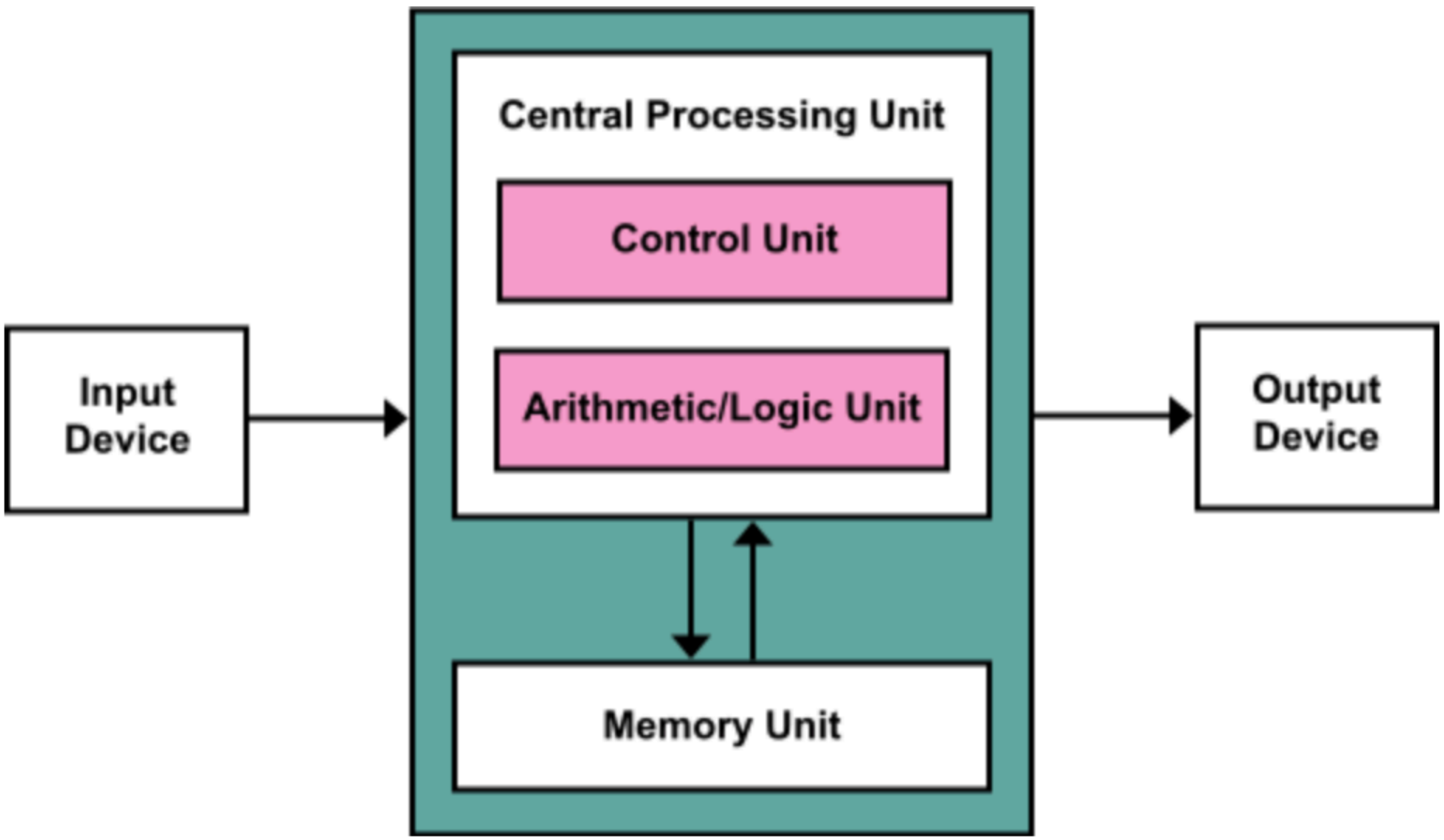
<u>Part 8</u>

Register Machines

von Neumann Machines

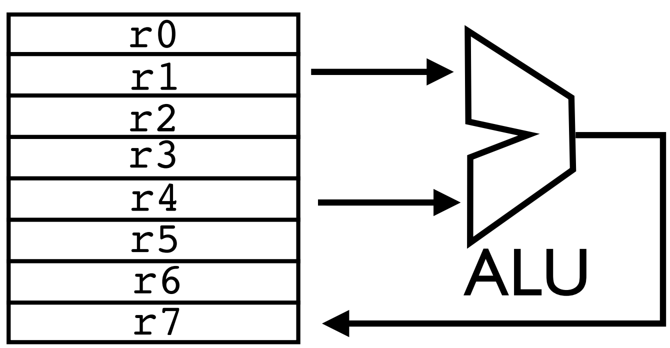


Arithmetic/Logic

- CPUs have instructions that perform <u>single</u> arithmetic operations

```
add, sub, mul, div, and, or, xor, not, eq, lt, ...
```

- These operations are applied to values, typically supplied from "registers" on the CPU



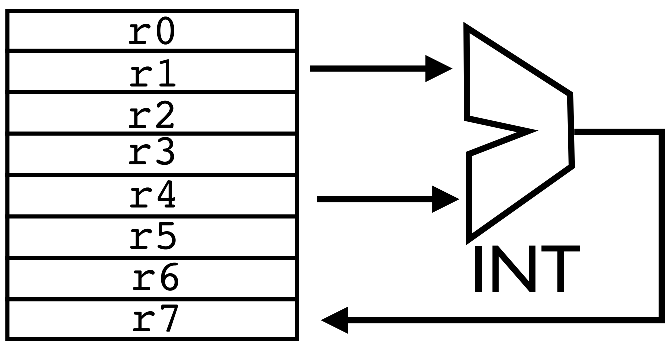

```
add r1, r4, r7
```

Data Types

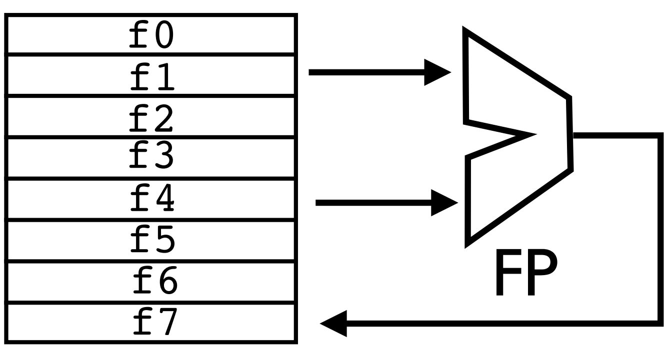
- There are two main datatypes (of varying sizes)

```
Integers (i8, i16, i32, i64, i128, etc.)
Floats (f32, f64, f128, etc.)
```

- Often handled by different ALUs & instructions




```
i32.add r1, r4, r7
```




```
f64.add f1, f4, f7
```

The Reality

- CPUs have limited resources
- Limited registers
- Limited memory
- Limited instructions
- Code needs to map to this constrained setting

Expression Evaluation

- Evaluate and show your work

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

- Use registers

Expression Evaluation

- Evaluate and show your work

```
2 + 3 * (10 - 2) + 5        r1 = 2

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

- Use registers

Expression Evaluation

- Evaluate and <u>show your work</u>

```
2 + 3 * (10 - 2) + 5        r1 = 2
                            r2 = 3
2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

- Use registers

Expression Evaluation

- Evaluate and show your work

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 2
r2 = 3
r3 = 10
```

- Use registers

Expression Evaluation

- Evaluate and show your work

```
2 + 3 * (10 - 2) + 5          r1 = 2
                              r2 = 3
2 + 3 * 8 + 5                 r3 = 10
                              r4 = 2
2 + 24 + 5

26 + 5

31
```

- Use registers

Expression Evaluation

- Evaluate and show your work

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 2
r2 = 3
r3 = 10
r4 = 2
r3 = sub(r3, r4)  ; r3 = 8
```

- Use registers

Expression Evaluation

- Evaluate and show your work

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 2
r2 = 3
r3 = 10
r4 = 2
r3 = sub(r3, r4)  ; r3 = 8
r2 = mul(r2, r3)  ; r2 = 24
```

- Use registers

Expression Evaluation

- Evaluate and <u>show your work</u>

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 2
r2 = 3
r3 = 10
r4 = 2
r3 = sub(r3, r4)  ; r3 = 8
r2 = mul(r2, r3)  ; r2 = 24
r1 = add(r1, r2)  ; r1 = 26
```

- Use registers

Expression Evaluation

- Evaluate and show your work

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 2
r2 = 3
r3 = 10
r4 = 2
r3 = sub(r3, r4)  ; r3 = 8
r2 = mul(r2, r3)  ; r2 = 24
r1 = add(r1, r2)  ; r1 = 26
r2 = 5
```

- Use registers

Expression Evaluation

- Evaluate and <u>show your work</u>

```
2 + 3 * (10 - 2) + 5        r1 = 2
                            r2 = 3
2 + 3 * 8 + 5               r3 = 10
                            r4 = 2
2 + 24 + 5                  r3 = sub(r3, r4)  ; r3 = 8
                            r2 = mul(r2, r3)  ; r2 = 24
26 + 5                      r1 = add(r1, r2)  ; r1 = 26
                            r2 = 5
31                          r1 = add(r1, r2)  ; r1 = 31
```

- Use registers (4 required)

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

- Use registers

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5        r1 = 10

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

- Use registers

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5        r1 = 10
                            r2 = 2
2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

- Use registers

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5        r1 = 10
                            r2 = 2
2 + 3 * 8 + 5               r1 = sub(r1, r2)   ; r1 = 8

2 + 24 + 5

26 + 5

31
```

- Use registers

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 10
r2 = 2
r1 = sub(r1, r2)   ; r1 = 8
r2 = 3
```

- Use registers

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5         r1 = 10
                             r2 = 2
2 + 3 * 8 + 5                r1 = sub(r1, r2)   ; r1 = 8
                             r2 = 3
2 + 24 + 5                   r1 = mul(r2, r1)   ; r1 = 24

26 + 5

31
```

- Use registers

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 10
r2 = 2
r1 = sub(r1, r2)    ; r1 = 8
r2 = 3
r1 = mul(r2, r1)    ; r1 = 24
r2 = 2
```

- Use registers

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5          r1 = 10
                              r2 = 2
2 + 3 * 8 + 5                 r1 = sub(r1, r2)    ; r1 = 8
                              r2 = 3
2 + 24 + 5                    r1 = mul(r2, r1)    ; r1 = 24
                              r2 = 2
26 + 5                        r1 = add(r2, r1)    ; r1 = 26

31
```

- Use registers

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 10
r2 = 2
r1 = sub(r1, r2)    ; r1 = 8
r2 = 3
r1 = mul(r2, r1)    ; r1 = 24
r2 = 2
r1 = add(r2, r1)    ; r1 = 26
r2 = 5
```

- Use registers

Expression Evaluation

- Evaluate more efficiently

```
2 + 3 * (10 - 2) + 5        r1 = 10
                            r2 = 2
2 + 3 * 8 + 5               r1 = sub(r1, r2)   ; r1 = 8
                            r2 = 3
2 + 24 + 5                  r1 = mul(r2, r1)   ; r1 = 24
                            r2 = 2
26 + 5                      r1 = add(r2, r1)   ; r1 = 26
                            r2 = 5
31                          r1 = add(r1, r2)   ; r1 = 31
```

- Use registers (2 required!)

Expression Evaluation

- Evaluate with fewer instructions

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

- Previously took 9 instructions

Expression Evaluation

- Evaluate with fewer instructions

```
2 + 3 * (10 - 2) + 5        r1 = 2

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

- Previously took 9 instructions

Expression Evaluation

- Evaluate with fewer instructions

```
2 + 3 * (10 - 2) + 5            r1 = 2
                                r2 = 10
2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

- Previously took 9 instructions

Expression Evaluation

- Evaluate with fewer instructions

```
2 + 3 * (10 - 2) + 5          r1 = 2
                              r2 = 10
2 + 3 * 8 + 5                 r2 = sub(r2, r1)   ; r2 = 8

2 + 24 + 5

26 + 5

31
```

- Previously took 9 instructions

Expression Evaluation

- Evaluate with fewer instructions

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 2
r2 = 10
r2 = sub(r2, r1)    ; r2 = 8
r3 = 3
```

- Previously took 9 instructions

Expression Evaluation

- Evaluate with fewer instructions

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 2
r2 = 10
r2 = sub(r2, r1)    ; r2 = 8
r3 = 3
r2 = mul(r3, r2)    ; r2 = 24
```

- Previously took 9 instructions

Expression Evaluation

- Evaluate with fewer instructions

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 2
r2 = 10
r2 = sub(r2, r1)    ; r2 = 8
r3 = 3
r2 = mul(r3, r2)    ; r2 = 24
r1 = add(r2, r1)    ; r1 = 26
```

- Previously took 9 instructions

Expression Evaluation

- Evaluate with fewer instructions

```
2 + 3 * (10 - 2) + 5          r1 = 2
                              r2 = 10
2 + 3 * 8 + 5                 r2 = sub(r2, r1)   ; r2 = 8
                              r3 = 3
2 + 24 + 5                    r2 = mul(r3, r2)   ; r2 = 24
                              r1 = add(r2, r1)   ; r1 = 26
26 + 5                        r2 = 5

31
```

- Previously took 9 instructions

Expression Evaluation

- Evaluate with fewer instructions

```
2 + 3 * (10 - 2) + 5

2 + 3 * 8 + 5

2 + 24 + 5

26 + 5

31
```

```
r1 = 2
r2 = 10
r2 = sub(r2, r1)    ; r2 = 8
r3 = 3
r2 = mul(r3, r2)    ; r2 = 24
r1 = add(r2, r1)    ; r1 = 26
r2 = 5
r1 = add(r1, r2)    ; r1 = 31
```

- Now takes 8 instructions! (but 3 registers)

New Questions!

- What happens if you run out of registers?
- What is the most efficient use of registers?
- The register scheduling problem
- It is non-trivial

Register Spilling

- If registers are exhausted, must "spill" an existing register to memory and read it back later

```
r1 = 10
...
...
...
store(r1, temp1)    ; spill r1 -> temp
r1 = 123           ; Load a new value into r1
...
r1 = load(temp1)    ; load temp -> r1
...
```

- Memory is slow. You want to avoid spilling

Register Scheduling

- To efficiently use registers, must perform advanced "analysis" on the code to know how values can be reused
- Example:

```
(x + y)/a + (x + y)/b
```

- Maybe could rewrite

```
t = x + y
t/a + t/b
```

- "Common subexpression elimination"

Example

```
(x+y)/2 + (x+y)/4
```

```
r1 = load(x)
r2 = load(y)
r3 = add(r1, r2)
r4 = 2
r5 = div(r3, r4)
r6 = load(x)
r7 = load(y)
r8 = add(r6, r7)
r9 = 4
r10 = div(r8,r9)
r11 = add(r5,r10)
```

```
r1 = load(x)
r2 = load(y)
r3 = add(r1, r2)
r4 = 2
r5 = div(r3, r4)
     ; already in r1
     ; already in r2
     ; already in r3
r9 = 4
r10 = div(r3, r4)
r11 = add(r5, r10)
```

Register Lifetimes

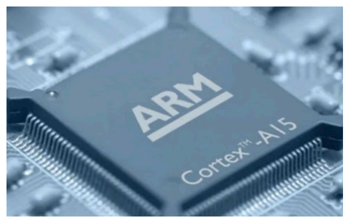
```
r1 = load(x)
r2 = load(y)
r3 = add(r1, r2)    ; r1, r2 not referenced again
r4 = 2
r5 = div(r3, r4)    ; r4 not referenced again
r9 = 4
r10 = div(r3, r4)  ; r3, r4 not referenced again
r11 = add(r5, r10) ; r5, r10 not reference again
```

- Rewrite with register overwriting

```
r1 = load(x)
r2 = load(y)
r1 = add(r1, r2)
r2 = 2
r2 = div(r1, r2)
r3 = 4
r3 = div(r1, r3)
r1 = add(r2, r3)
```

A Moment of Reflection

- All CPUs are similar



- But the low-level details vary. For example, number of registers, variety of instructions, etc.

- Question: Do you write a compiler for a single model of a specific CPU? <u>No.</u>

Abstract Machines

- Compilers often target an "abstract machine"
- A generic "CPU"
- With a standard set of basic "instructions"

Intermediate Representation

- The abstract machine is programmed using "intermediate representation" or IR Code
- It's like a generic machine code
  - Mimics architecture of actual CPUs
  - "Easy" to translate to actual machine code

Big Picture

Source

```
print 2 + 3 * 4;
```

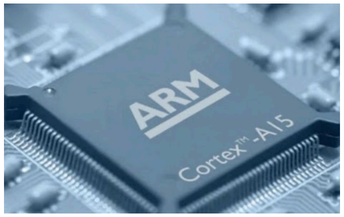
AST

```
PrintStatement(
    BinOp('+',
          Integer(2),
          BinOp('*',
                Integer(3),
                Integer(4)
               )
         )
)
```

IR

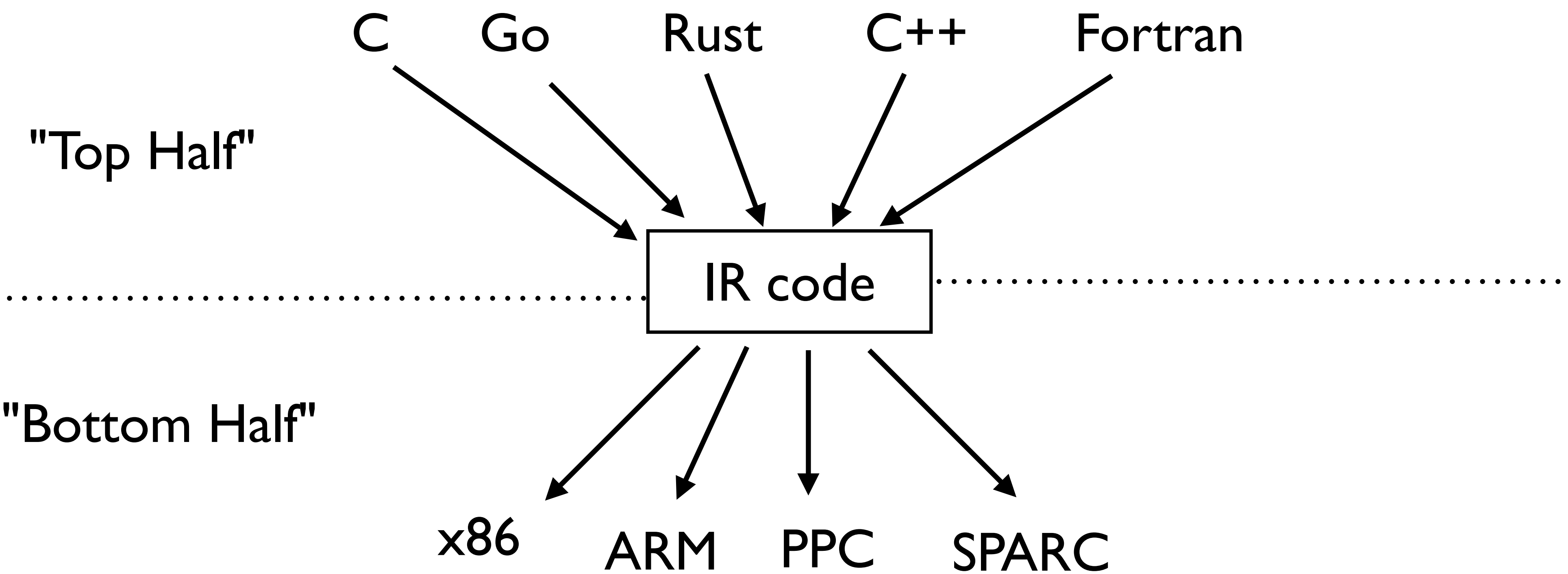
```
('i32.const', 2, 'r1'),
('i32.const', 3, 'r2'),
('i32.const', 4, 'r3'),
('i32.mul','r2','r3','r4')
('i32.add','r1','r4','r5')
...
```

translate



Metal

Compiler Design



IR Code

- What does IR code look like?
- It embodies a few essential concepts
    - A Model of Computation
    - Control Flow
    - Memory
    - Modules
- Examples: LLVM, WebAssembly

Project

- Compile Wabbit to IR
  - See `wabbit/llvm.py`
  - See `wabbit/wasm.py`