UNIVERSITY of VIRGINIA

Building to a Computer

CS 2130: Computer Systems and Organization 1

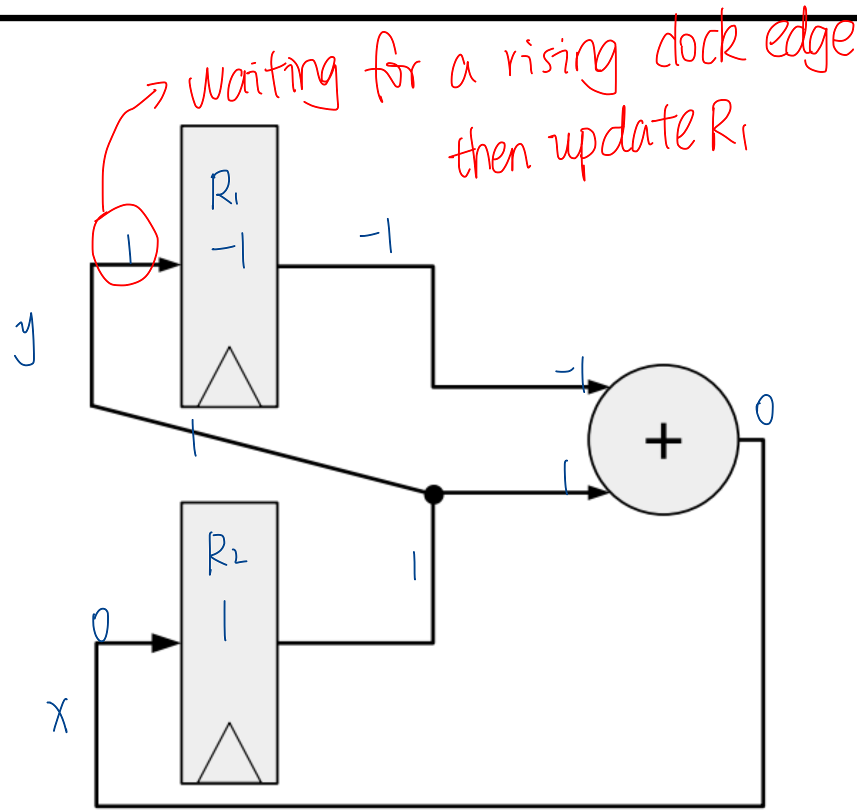
Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 2 due Monday
- Office hours most days!

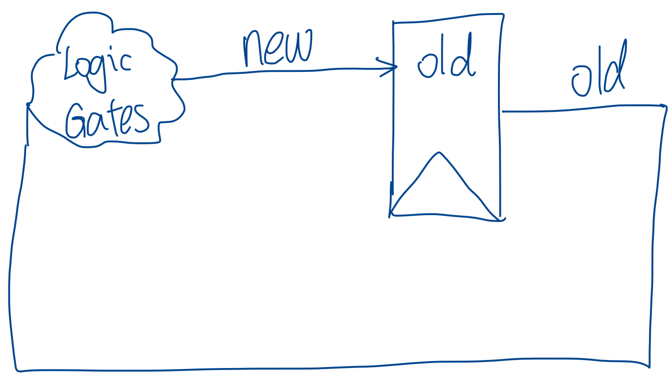
Another Counter



clock	x	y	R_1	R_2
0	0	1	-1	1
1	1	0	1	0
2	1	1	0	1
3	2	1	1	1
4	3	2	1	2
5	5	3		

x : Fibonacci sequence.

Common Model in Computers



The register ignore all the calculations/updates until rising clock edge.

Code to Build Circuits from Gates

Write code to build circuits from gates

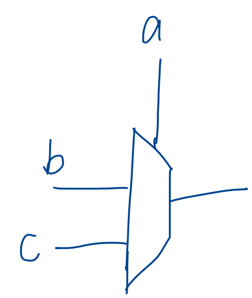
- Gates we already know: &, |, ^, ~
- Operations we can build from gates: +, -
- Others we can build:

*:
```
      2130
    × 1101
    ------
      2130
     00000
    2130
   2130
   -------
```
(left shift and add)

Code to Build Circuits from Gates

Write code to build circuits from gates

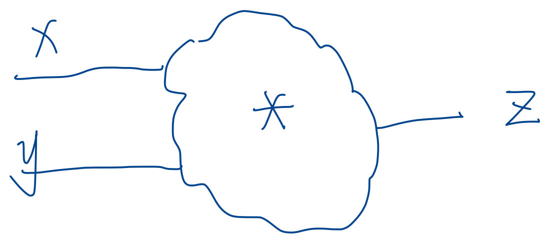
- Gates we already know: &, |, ^, ~
- Operations we can build from gates: +, -
- Others we can build:
- Ternary operator: ? : a==0 ? b : c



Equals

Equals: =

- Attach with a wire (i.e., connect things)
- Ex: z = x * y



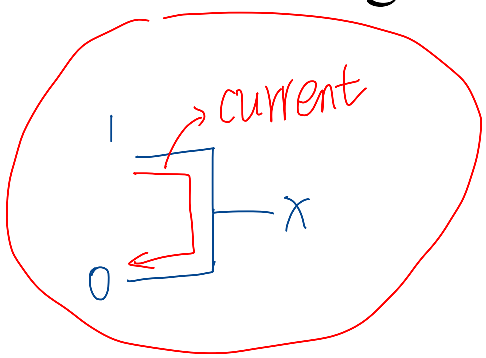
attaching the circuit to whatever I'm trying to output

Equals

Equals: =

- Attach with a wire (i.e., connect things)
- Ex: z = x * y
- What about the following?

x = 1
x = 0



doesn't work!

Equals

Equals: =

- Attach with a wire (i.e., connect things)
- Ex: z = x * y
- What about the following?

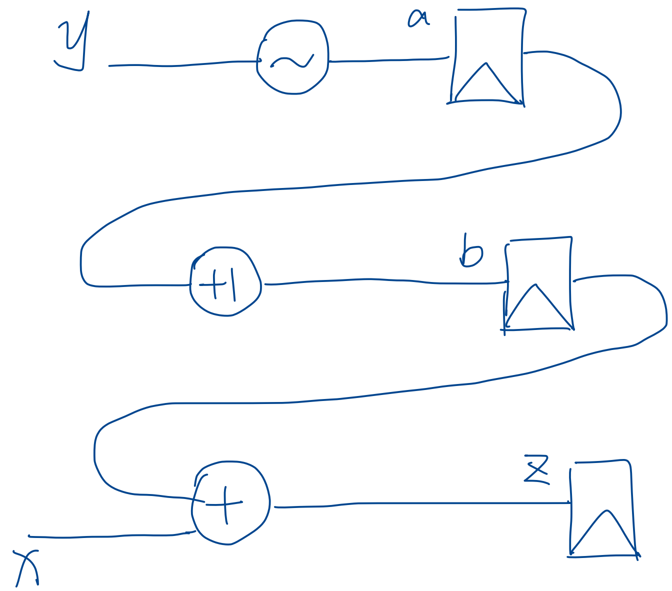
  x = 1

  x = 0
- **Single assignment:** each variable can only be assigned a value once

Subtraction

$z = x + \sim y + 1$

$a = \sim y$
$b = a + 1$
$z = x + y$



Comparisons

Each of our comparisons in code are straightforward to build:

- == - xor then nor bits of output

Comparisons

Each of our comparisons in code are straightforward to build:

- == - xor then nor bits of output
- != - same as == without not of output

Comparisons

Each of our comparisons in code are straightforward to build:

- == - xor then nor bits of output
- != - same as == without not of output
- < - consider x < 0

Comparisons

Each of our comparisons in code are straightforward to build:

- == - xor then nor bits of output → bits are equal iff the XOR is 0
- != - same as == without not of output
- < - consider x < 0
- >, <=, => are similar

Compute d = x ^ y, if all bits of d are 0 ⇒ equal

x<y : x-y<0

→ (x >> 31) & 1

if the result is 1, then negative.

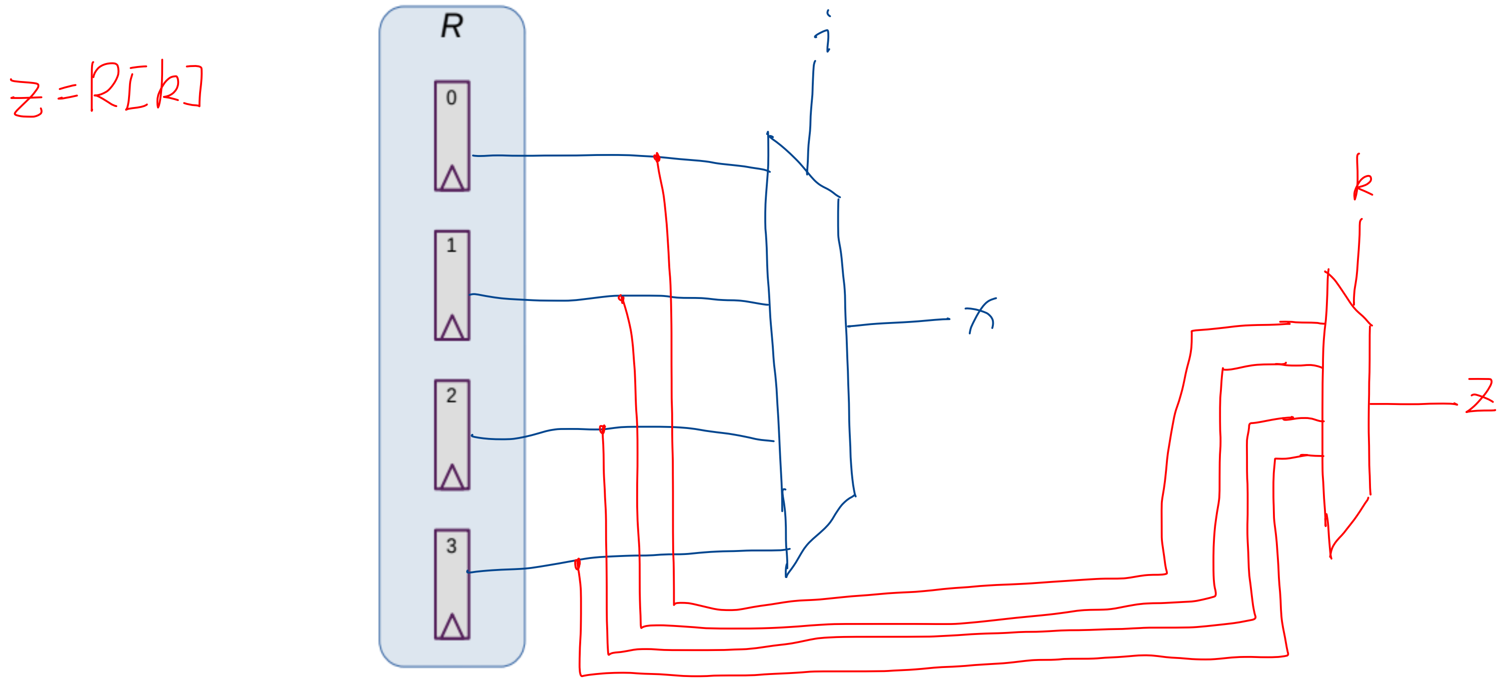
Indexing

Indexing with square brackets: [ ]

- **Register bank** (or **register file**) - an array of registers
  - Can programmatically pick one based on index
  - I.e., can determine which register while running
- Two important operations:
  - x = R[i] - Read from a register
  - R[j] = y - Write to a register

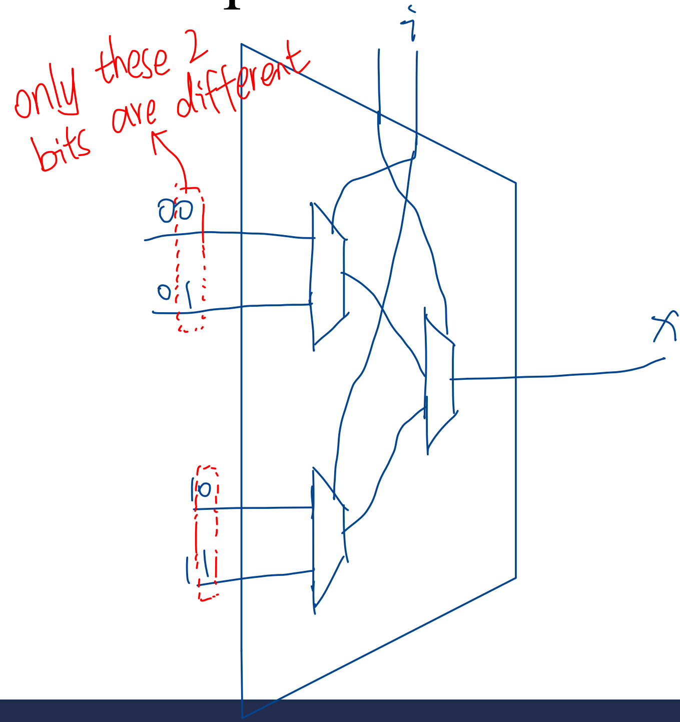
Reading

x = R[i] - connect output of registers to *x* based on index *i*



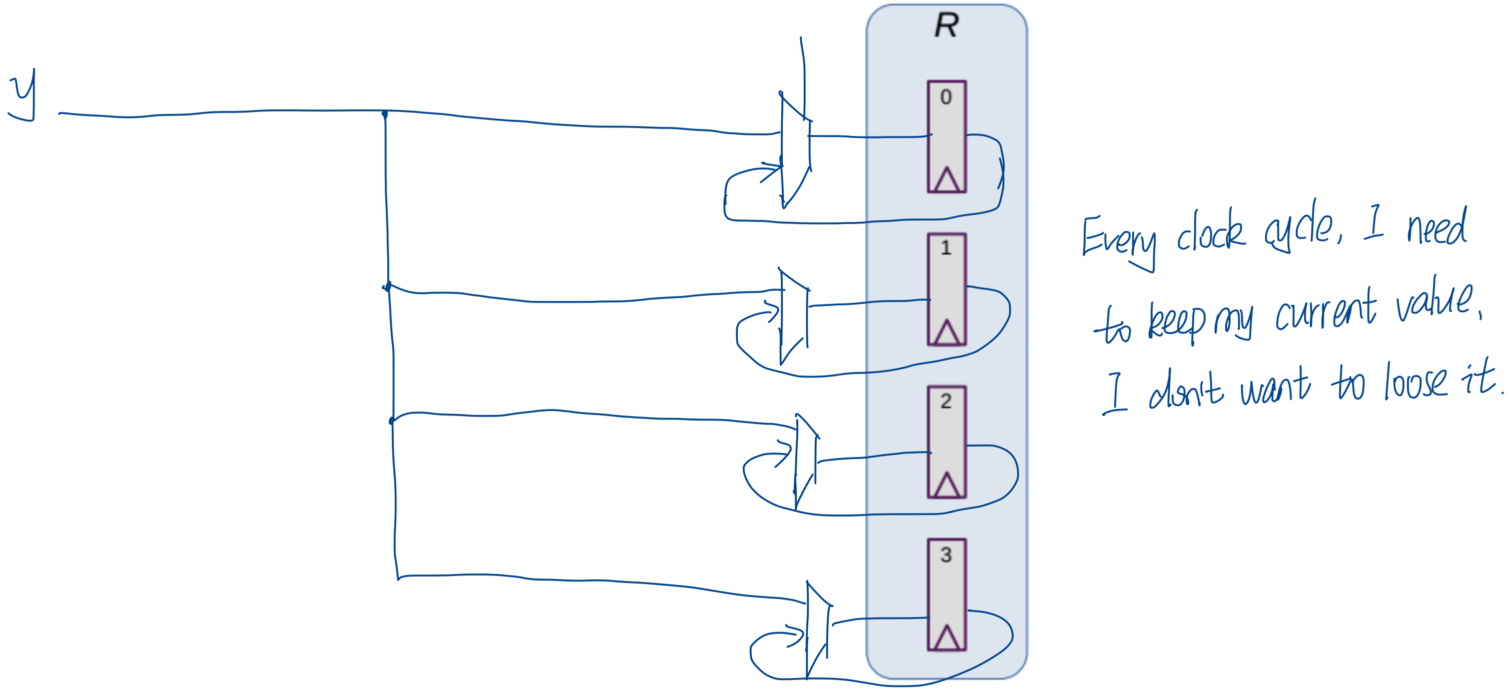
Aside: 4-input Mux

How do we build a 4-input mux? How many wires should *i* be?



Writing

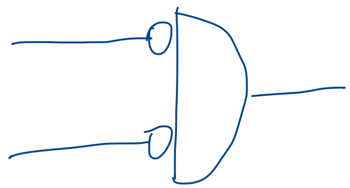
R[j] = y - connect *y* to input of registers based on index *j*



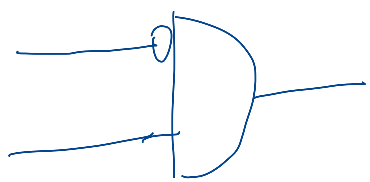
Aside: Creating == 0 gates

How do we build gates that check for $j == w$?

j		== 0
0	0	1
0	1	0
1	0	0
1	1	0



j		== 1
0	0	0
0	1	1
1	0	0
1	1	0



Need one more thing to build computers

Memory and Storage

1 byte = 8 bits

Registers $\approx$ KiB

1024 Bytes

- 6 gates each, $\approx$ 24 transistors
- Efficient, fast
- Expensive!
- Ex: local variables

These do not persist between power cycles

Memory and Storage

Memory ≈ GiB

Random access memory

- Two main types: SRAM, DRAM

SRAM faster than DRAM → catch

- DRAM: 1 transistor, 1 capacitor per bit
- DRAM is cheaper, simpler to build
- Ex: data structures, local variables

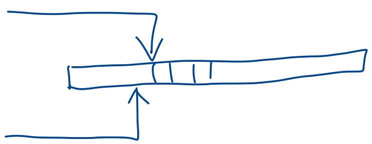
These do not persist between power cycles

Memory and Storage

Disk ≈ GiB-TiB

- Two main types: flash (solid state), magnetic disk
- Magnetic drive
  - Platter with physical arm above and below
  - Cheap to build
  - Very slow! Physically move arm while disk spins
- Ex: files

Data on disk does persist between power cycles



Putting it all together

- Information modeled by voltage through wires (1 vs 0)
- Transistors
- Gates: & | ~ ^
- Multi-bit values: representing integers, floating point numbers
- Multi-bit operations using circuits
- Storing results using registers, clocks
- Memory

Code

How do we run code? What do we need?

Consider the following code:

```
...
8:    x = 16
9:    y = x
10:   x += y
...
```

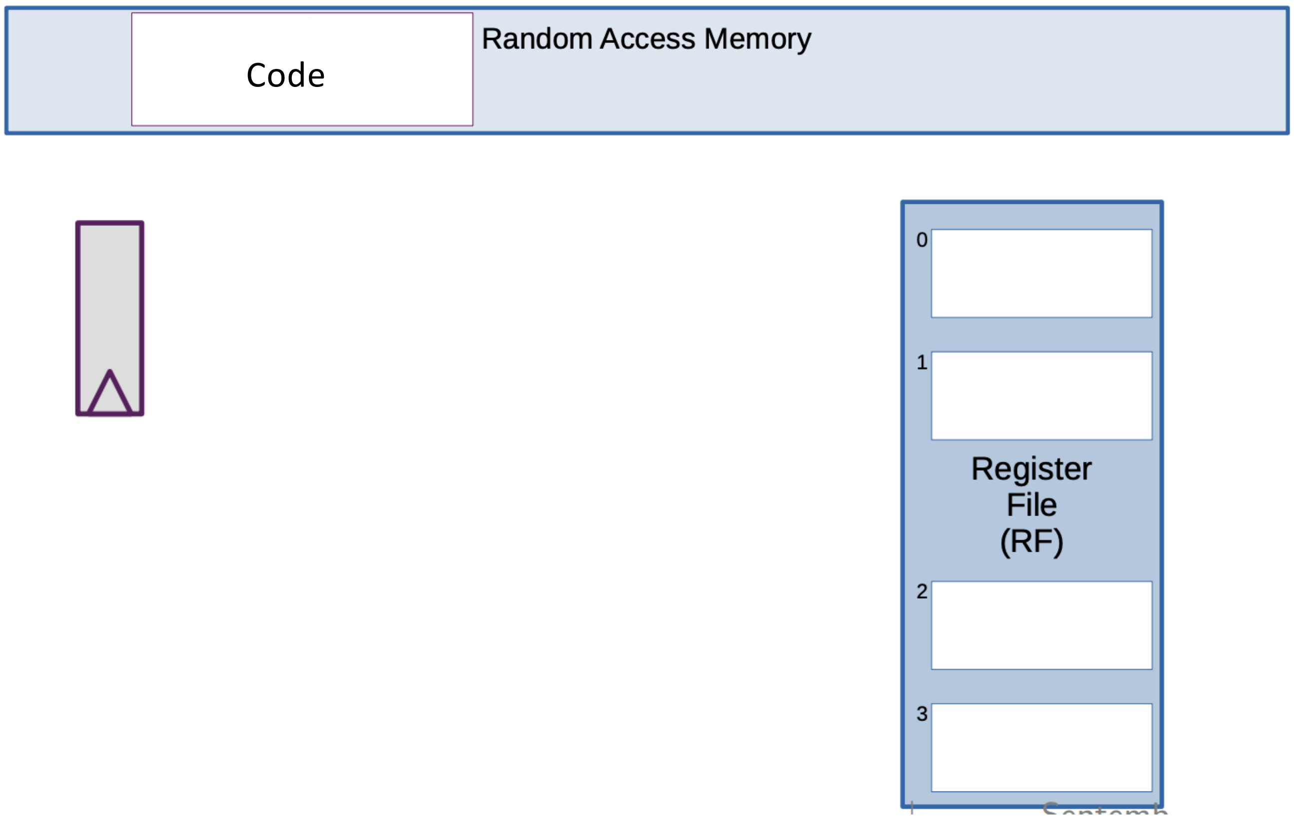
What is the value of x after line 10? 32

Bookkeeping

What do we need to keep track of?

- **Code** - the program we are running
  - RAM (Random Access Memory)
- **State** - things that may change value (i.e., variables)
  - Register file - can read and write values each cycle
- **Program Counter (PC)** - where we are in our code
  - Single register - byte number in memory for next instruction

Building a Computer

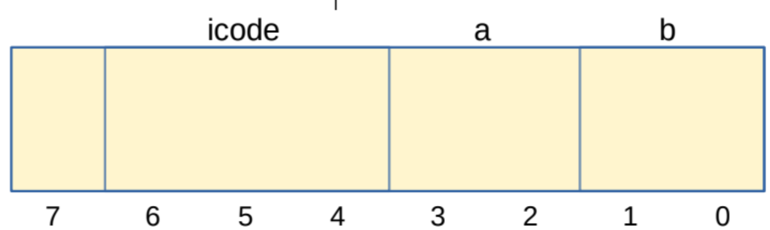


Encoding Instructions

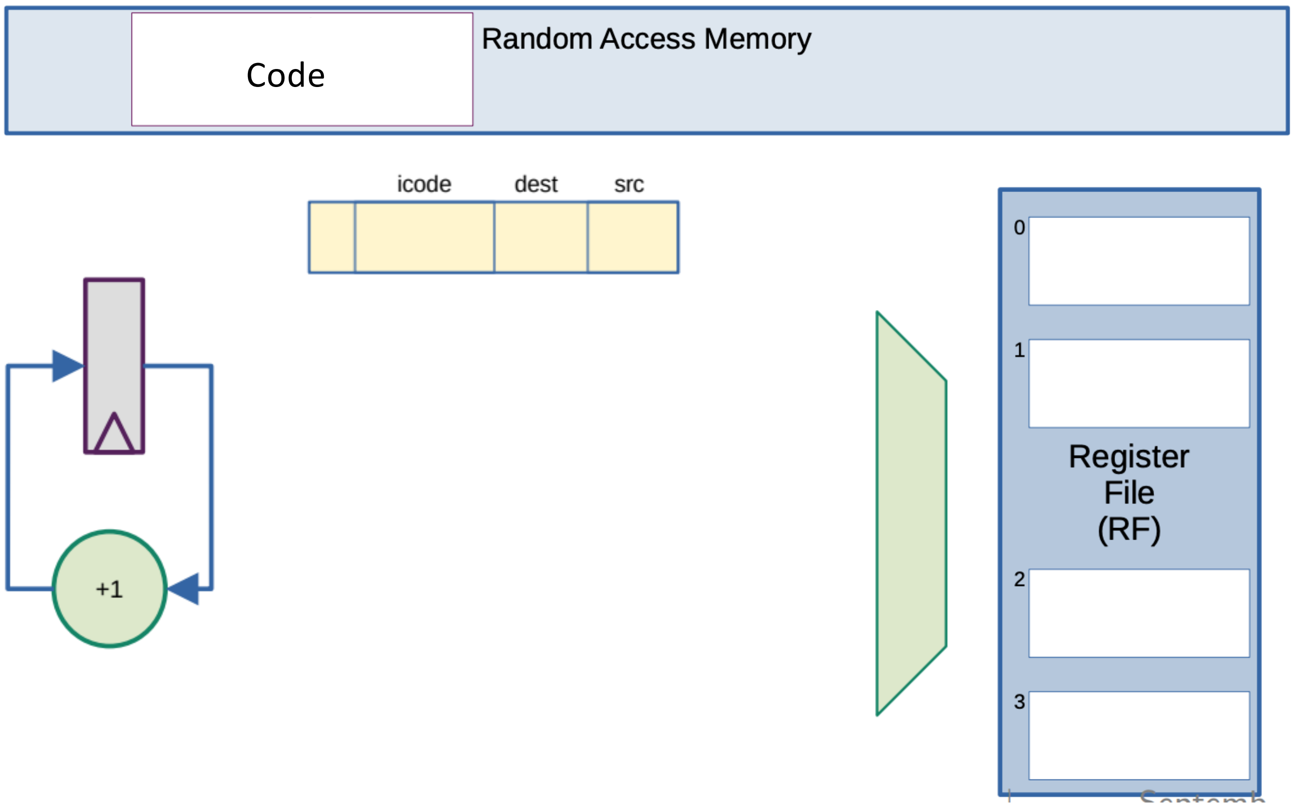
Encoding of Instructions (**icode** or **opcode**)

- Numeric mapping from icode to operation

icode	meaning
0	`rA = rB`
1	`rA &= rB`
2	`rA += rB`
…	…



Building a Computer



Building a Computer