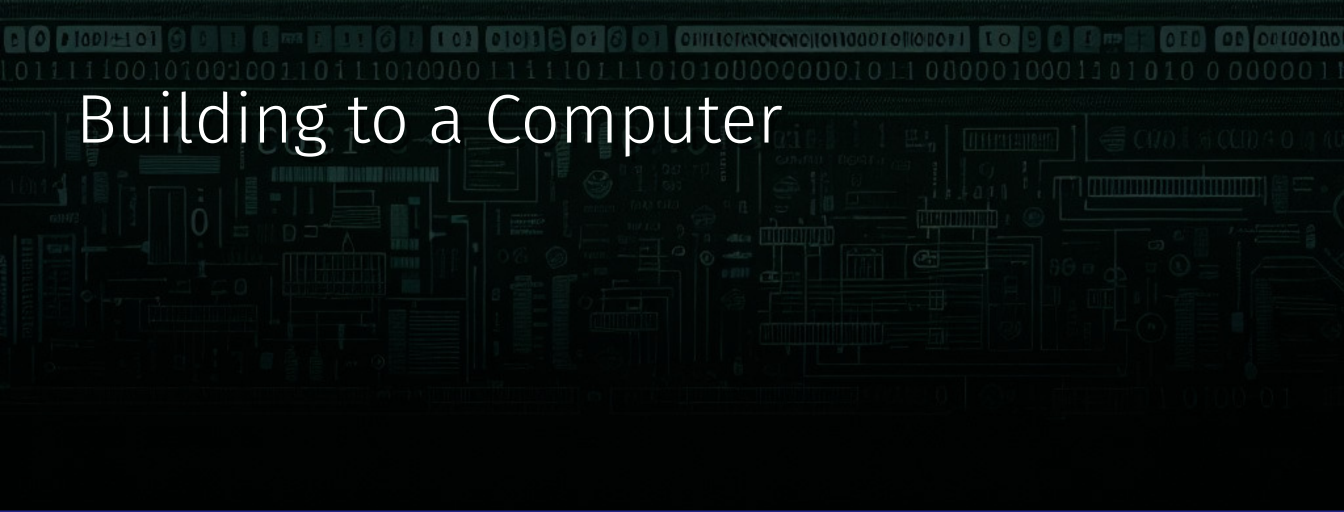
Building to a Computer



CS 2130: Computer Systems and Organization 1
February 4, 2026

Announcements

- Homework 2 due Monday

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:

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: `?   :`

Equals

Equals: =

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

Equals

Equals: =

- Attach with a wire (i.e., connect things)
- Ex: `z = x * y`
- What about the following?
  ```
  x = 1
  x = 0
  ```

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 + ~y + 1
```

```
a = ~y
b = a + 1
z = x + b
```

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
- `!=` - same as `==` without not of output
- `<` - consider `x < 0`
- `>`, `<=`, `=>` are similar

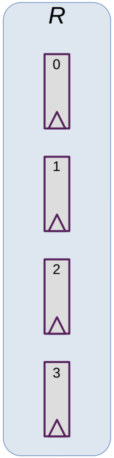
Indexing

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

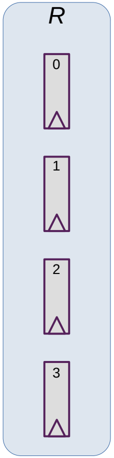


Aside: 4-input Mux

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 $j == w$?

Need one more thing to build computers

Memory and Storage

Registers $\approx$ KiB

- 6 gates each, ≈ 24 transistors
- Efficient, fast
- Expensive!
- Ex: local variables

These do not persist between power cycles

Memory and Storage

Memory $\approx$ GiB

- Two main types: SRAM, DRAM
- 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 $\approx$ 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

Our story so far

- 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
...
```

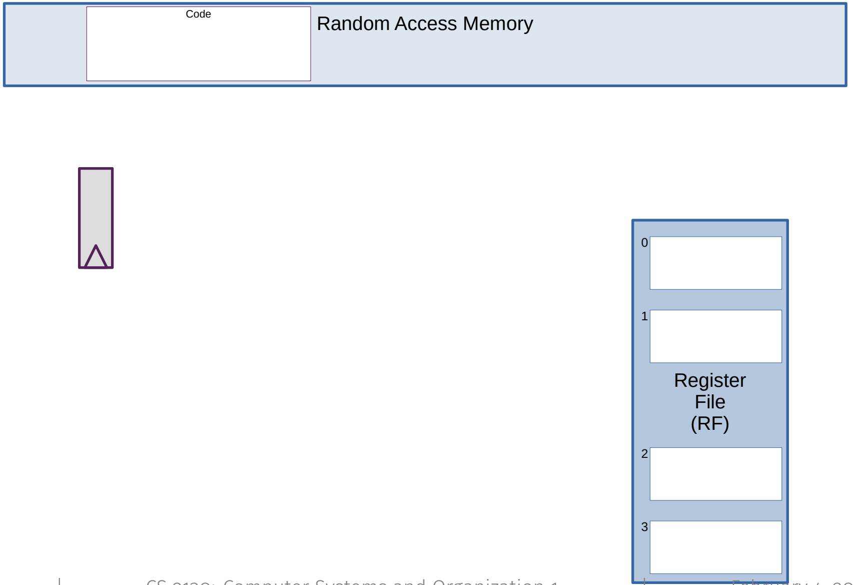
What is the value of *x* after line 10?

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

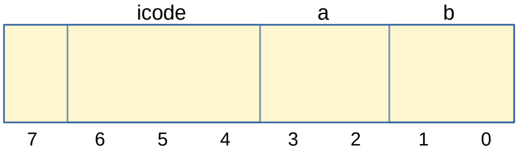


Encoding Instructions

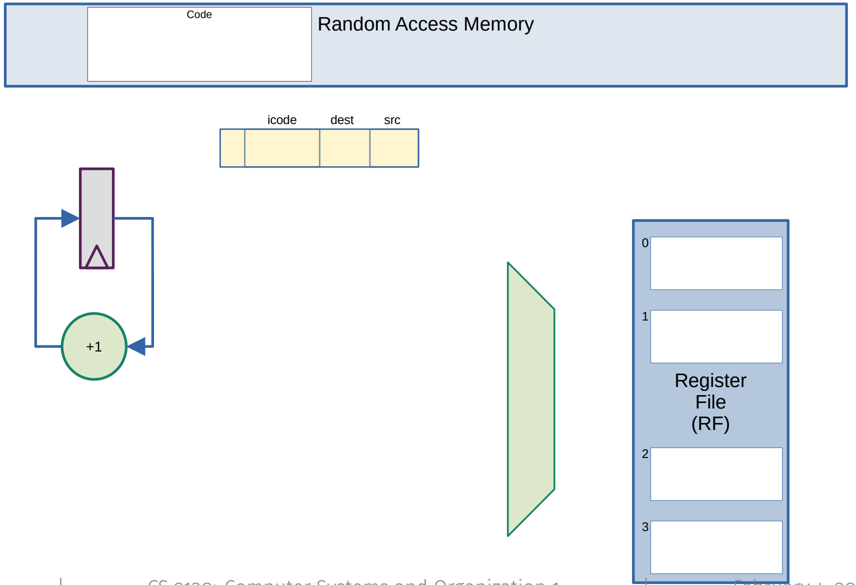
Encoding of Instructions (**icode** or **opcode**)

- Numeric mapping from icode to operation

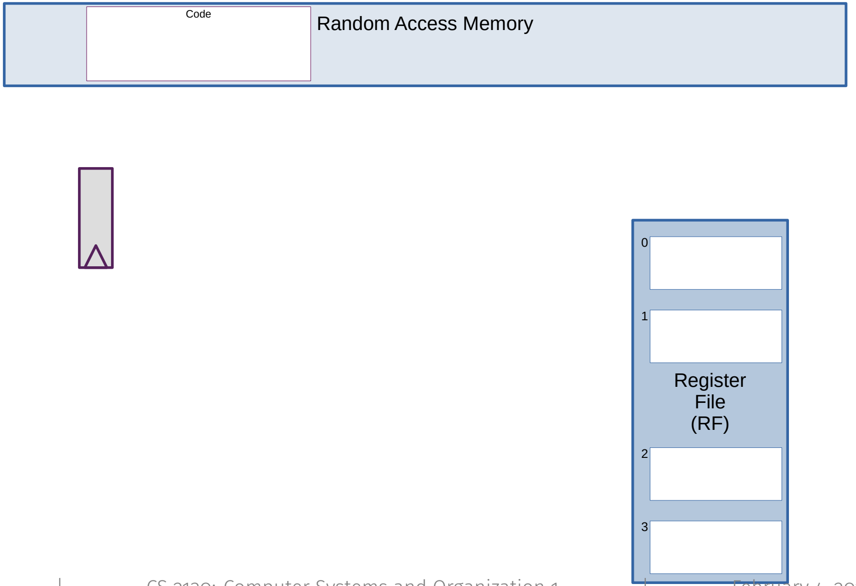
icode	meaning
0	`rA = rB`
1	`rA &= rB`
2	`rA += rB`
...	...



Building a Computer



Building a Computer


Code
Random Access Memory
0
1
Register File (RF)
2
3