Fetch, Decode, Execute

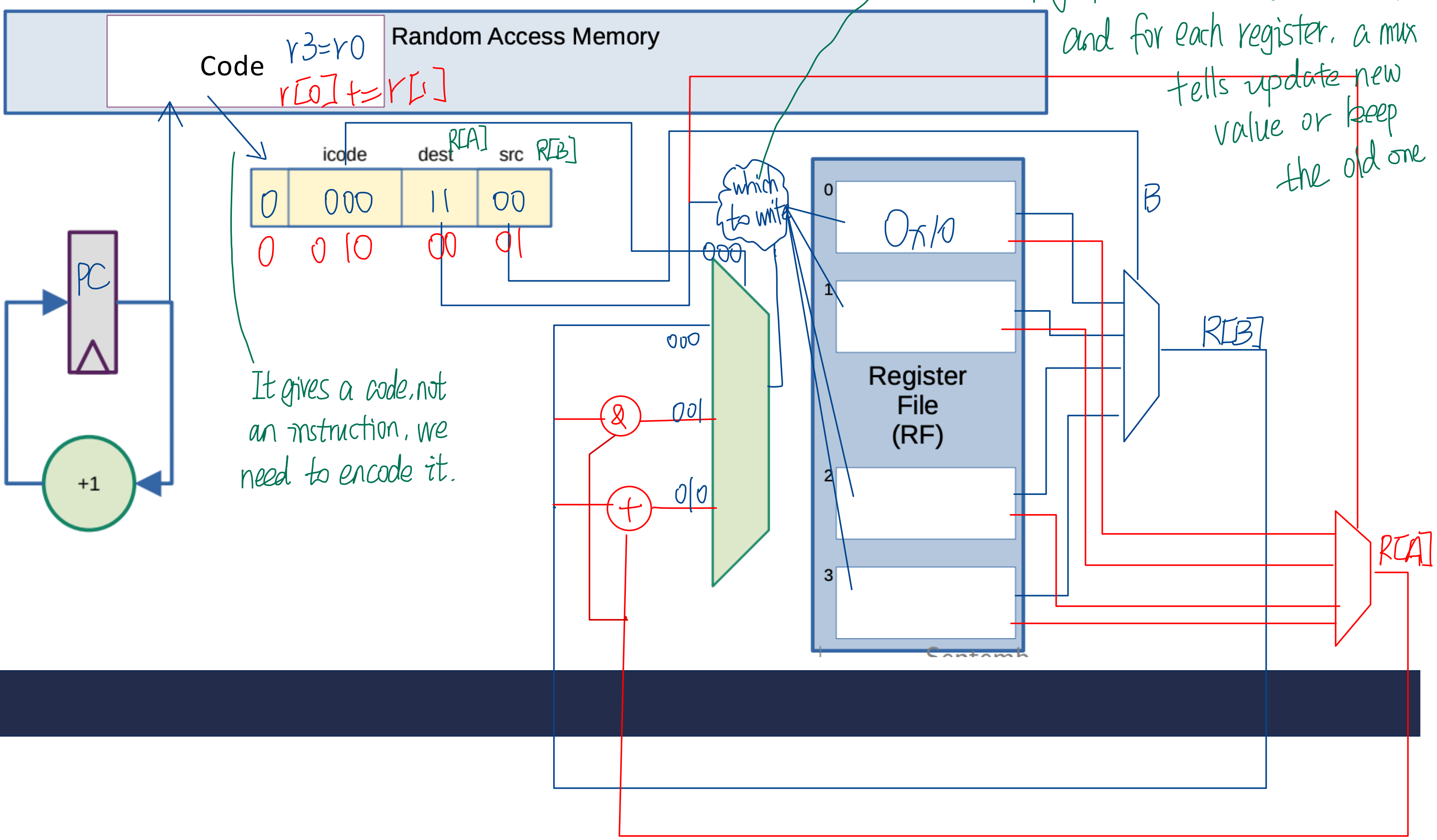
CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Building a Computer

x : R[0]
y : R[3]

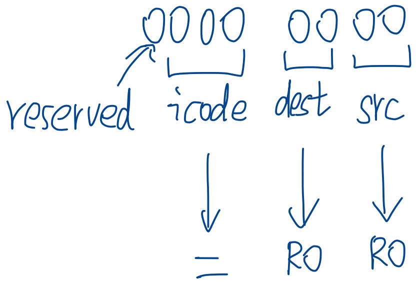


This is the circuit tells you which register to write (==0, ==1), and for each register, a mux tells update new value or keep the old one

It gives a code, not an instruction, we need to encode it.

Question

What happens if we get the 0-byte instruction? 00



r0=r0

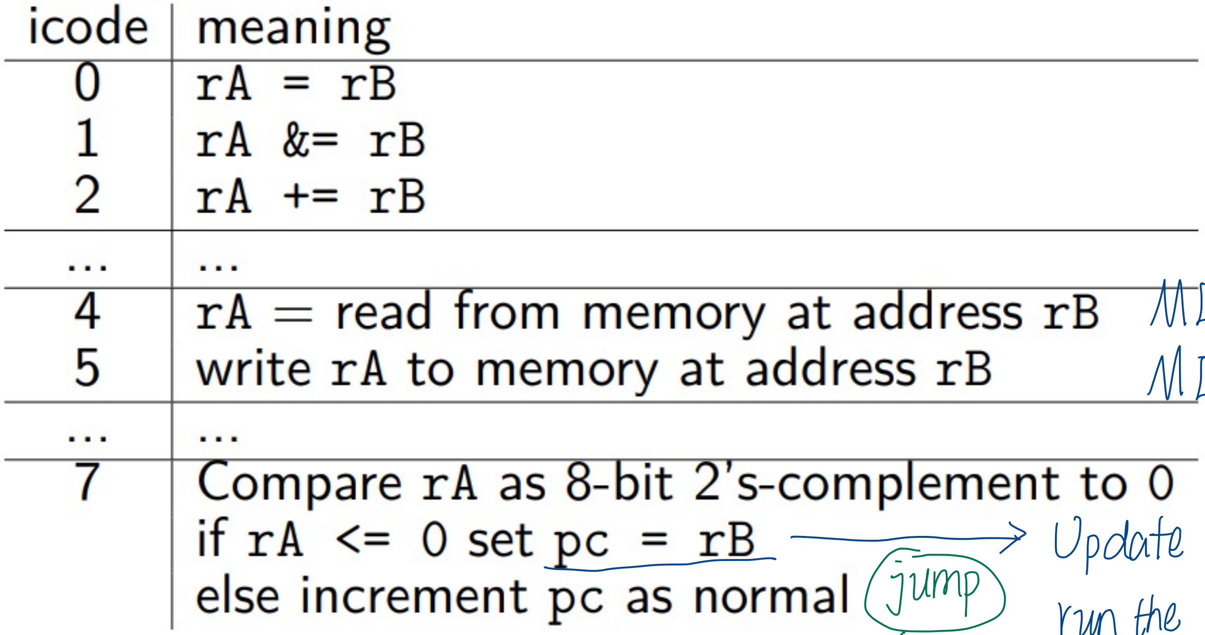
This is a no operation (noop)

Doesn't do anything. ⇒ but useful

wait for cycles to pass while doing sth. else.

Our Computer’s Instructions

Toy ISA 3-bit icode



icode	meaning
0	rA = rB
1	rA &= rB
2	rA += rB
…	…
4	rA = read from memory at address rB
5	write rA to memory at address rB
…	…
7	Compare rA as 8-bit 2’s-complement to 0 if rA <= 0 set pc = rB else increment pc as normal

M[r[B]]

M[r[B]] = rA

Update PC and say, I want to run the code at a different place.

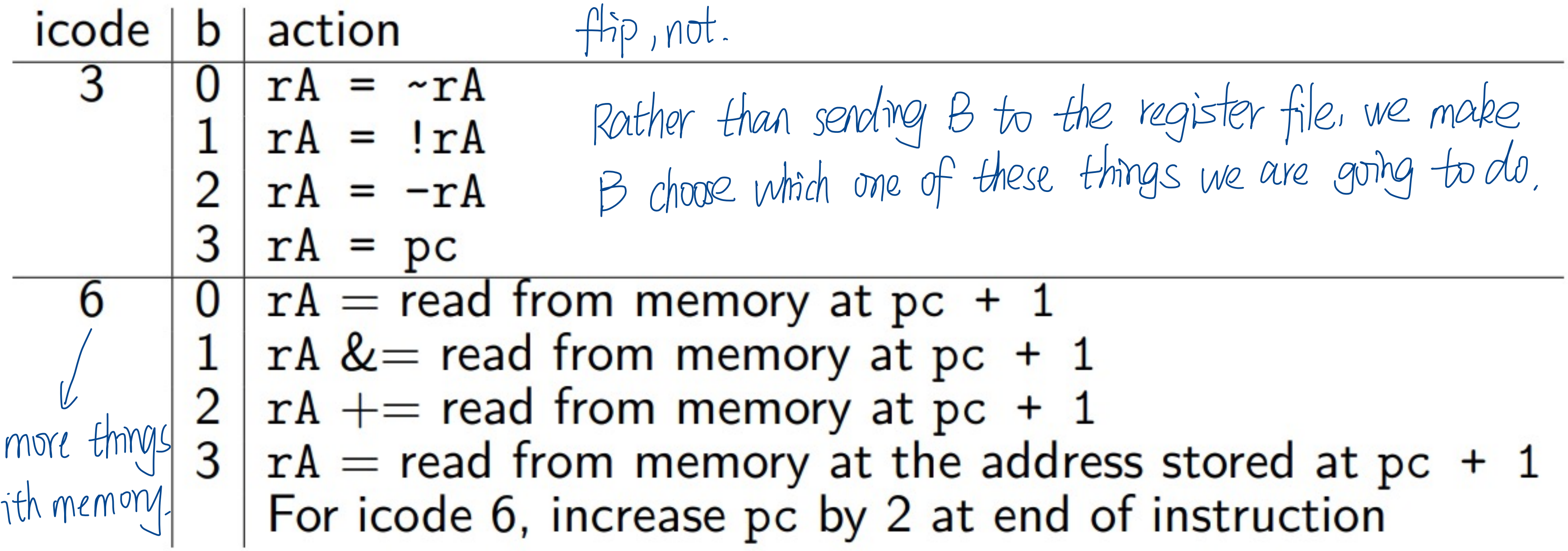
jump

loop/if statement

Our Computer's Instructions

Toy ISA 3-bit icode

Some operations that only need one register like flip, not.



icode	b	action
3	0	`rA = ~rA`
	1	`rA = !rA`
	2	`rA = -rA`
	3	`rA = pc`
6	0	rA = read from memory at pc + 1
	1	rA &= read from memory at pc + 1
	2	rA += read from memory at pc + 1
	3	rA = read from memory at the address stored at pc + 1
		For icode 6, increase pc by 2 at end of instruction

Rather than sending B to the register file, we make B choose which one of these things we are going to do.

Do more things with memory.

High-level Instructions

In general, 3 kinds of instructions

- **moves** - move values around without doing “work”
- **math** - broadly doing “work”
- **jumps** - jump to a new place in the code

Moves (move a copy of value from one place to another)

Few forms

(primitive variables)

- Register to register (icode 0), x = y → direct register moves like moving the values from r3 to r2 (r2=r3).
- Register to/from memory (icodes 4-5), x = M[b], M[b] = x or, more details: R0=M[R[2]]

(objects or arrays)

① go to memory, look at the value of b

② Use that value as the index of my big array in memory. (memory is a big array of bytes)

Memory

- Address: an index into memory.
  - maybe 256 bytes?
  - Addresses are just (large) numbers
  - “dot” operation/square brackets ⇒ read the values from somewhere
  - Usually we will not look at the number and trust it exists and is stored in a register
  - just indexes.

Moves

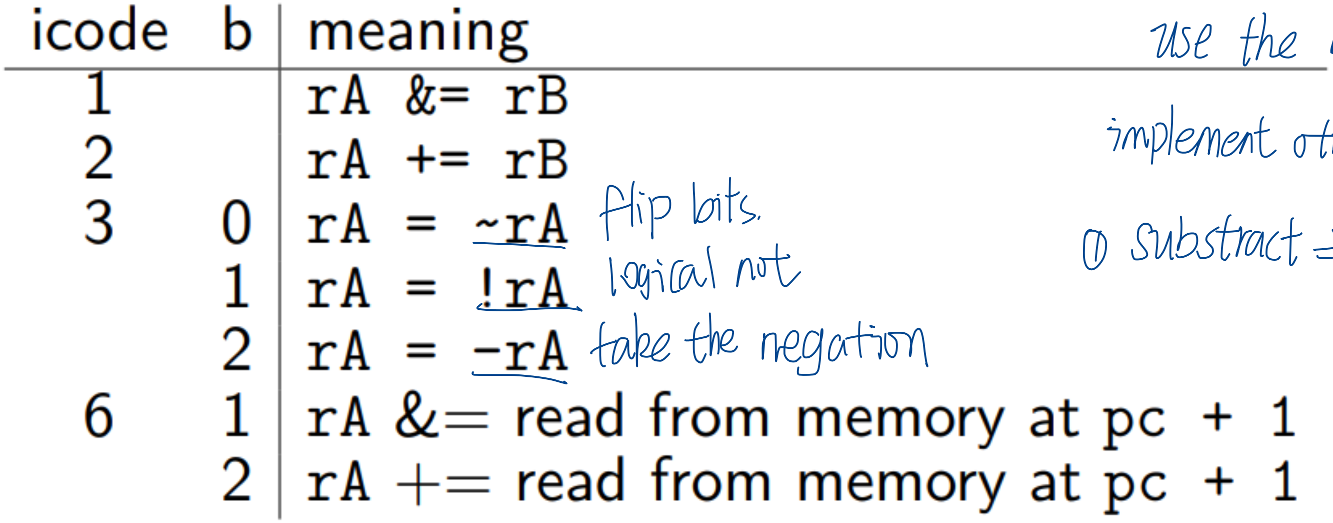
icode	b	action
0		`rA = rB`
3	3	`rA = pc`
4		`rA` = read from memory at address `rB`
5		write `rA` to memory at address `rB`
6	0	`rA` = read from memory at `pc + 1`
	3	`rA` = read from memory at the address stored at `pc + 1`

next instruction

do things with function calls. (learn it later).

Math

Broadly doing work



icode	b	meaning
1		rA &= rB
2		rA += rB
3	0	rA = ~rA flip bits.
	1	rA = !rA logical not
	2	rA = -rA take the negation
6	1	rA &= read from memory at pc + 1
	2	rA += read from memory at pc + 1

use the basic operations to implement other operations.

① substract ⇒ taking one value, negating it and adding it to the other.

② multiply → repeated addition

Note: We can implement other operations using these things!