

The background of the slide features a dark, textured pattern. At the top, there are horizontal bands of binary code (0s and 1s). Below this, there are faint, light-colored circuit diagrams and technical drawings, including what appears to be a keyboard layout and various electronic components connected by lines, suggesting a computer system or hardware architecture.

# Toy Instruction Set Architecture

CS 2130: Computer Systems and Organization 1  
February 9, 2026

# Announcements

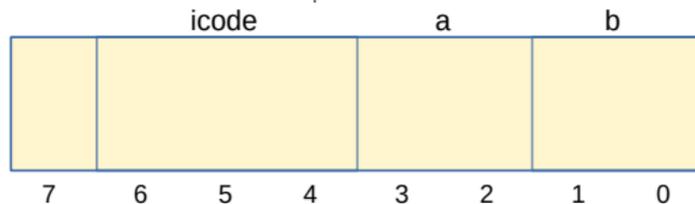
- Homework 2 due tonight at 11:59pm on Gradescope
- Homework 3 out tomorrow, due next Monday at 11:59pm on Gradescope

# Encoding Instructions

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

- Numeric mapping from icode to operation

icode	meaning
0	$rA = rB$
1	$rA \&= rB$
2	$rA += rB$
...	...

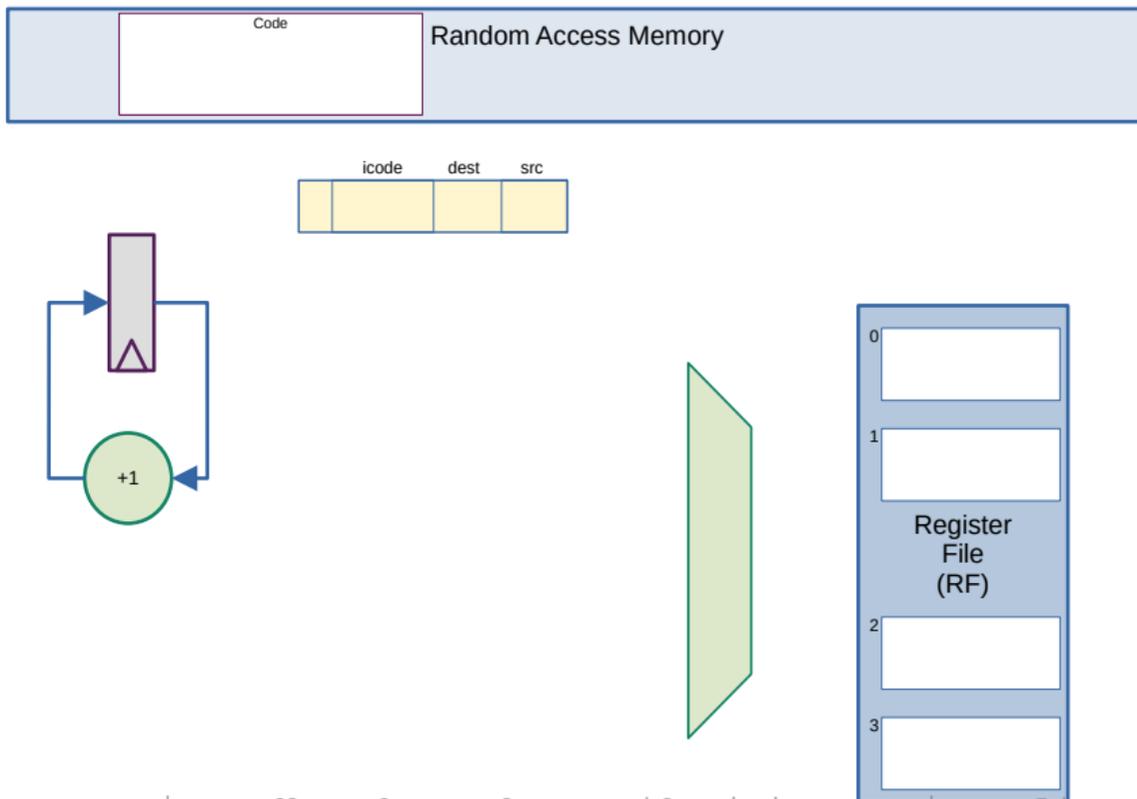


# Encoding Instructions

## Encoding of Instructions

- 3-bit icode (which operation to perform)
  - Numeric mapping from icode to operation
- Which registers to use (2 bits each)
- Reserved bit for future expansion

# Building a Computer





# Question

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

# 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 $rA \leq 0$ set $pc = rB$ else increment $pc$ as normal

# Our Computer's Instructions

## Toy ISA 3-bit icode

icode	b	action
3	0	$rA = \sim rA$
	1	$rA = !rA$
	2	$rA = -rA$
	3	$rA = pc$
6	0	$rA = \text{read from memory at } pc + 1$
	1	$rA \&= \text{read from memory at } pc + 1$
	2	$rA += \text{read from memory at } pc + 1$
	3	$rA = \text{read from memory at the address stored at } pc + 1$
		For icode 6, increase $pc$ by 2 at end of instruction

# 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

Few forms

- Register to register (icode 0),  $x = y$
- Register to/from memory (icodes 4-5),  $x = M[b], M[b] = x$

Memory

- **Address:** an index into memory.
  - Addresses are just (large) numbers
  - Usually we will not look at the number and trust it exists and is stored in a register

# Moves

icode	b	action
0		$rA = rB$
3	3	$rA = pc$
4		$rA = \text{read from memory at address } rB$
5		$\text{write } rA \text{ to memory at address } rB$
6	0	$rA = \text{read from memory at } pc + 1$
	3	$rA = \text{read from memory at the address stored at } pc + 1$

# Math

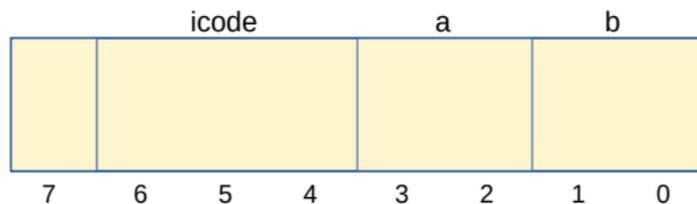
Broadly doing work

icode	b	meaning
1		$rA \&= rB$
2		$rA += rB$
3	0	$rA = \sim rA$
	1	$rA = !rA$
	2	$rA = -rA$
6	1	$rA \&= \text{read from memory at } pc + 1$
	2	$rA += \text{read from memory at } pc + 1$

*Note: We can implement other operations using these things!*

# icodes 3 and 6

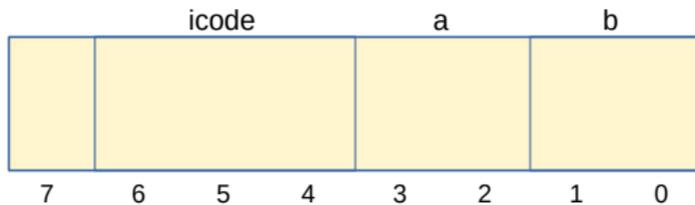
Special property of icodes 3 & 6: only one register used



icode	b	action
3	0	$rA = \sim rA$
	1	$rA = !rA$
	2	$rA = -rA$
	3	$rA = pc$

# icodes 3 and 6

Special property of 3 & 6: only one register used



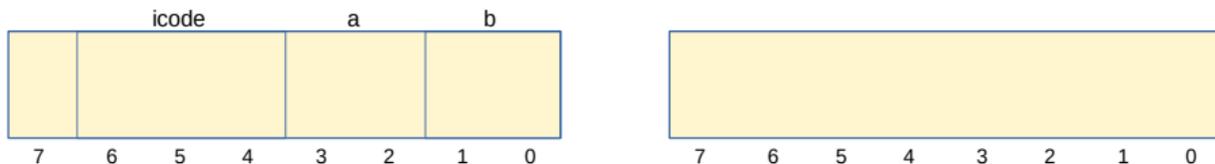
- Side effect: all bytes between 0 and 127 are valid instructions!
- As long as high-order bit is 0
- No syntax errors, any instruction given is valid

# Immediate values

icode 6 provides literals, **immediate** values

icode	b	action
6	0	$rA = \text{read from memory at } pc + 1$
	1	$rA \&= \text{read from memory at } pc + 1$
	2	$rA += \text{read from memory at } pc + 1$
	3	$rA = \text{read from memory at the address stored at } pc + 1$

For icode 6, increase  $pc$  by 2 at end of instruction



# Encoding Instructions

Example 1: `r1 += 19`

# Instructions

icode	b	meaning
0		$rA = rB$
1		$rA \&= rB$
2		$rA += rB$
3	0	$rA = \sim rA$
	1	$rA = !rA$
	2	$rA = -rA$
	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 $pc + 1$
	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
7		Compare $rA$ as 8-bit 2's-complement to 0 if $rA \leq 0$ set $pc = rB$ else increment $pc$ as normal

# Encoding Instructions

Example 2: `M[0x82] += r3`

Read memory at address `0x82`, add `r3`, write back to memory at same address

# Instructions

icode	b	meaning
0		$rA = rB$
1		$rA \&= rB$
2		$rA += rB$
3	0	$rA = \sim rA$
	1	$rA = !rA$
	2	$rA = -rA$
	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 $pc + 1$
	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
7		Compare $rA$ as 8-bit 2's-complement to 0 if $rA \leq 0$ set $pc = rB$ else increment $pc$ as normal

# Jumps

- Moves and math are large portion of our code
- We also need **control constructs**
  - Change what we are going to do next
  - `if`, `while`, `for`, functions, ...
- Jumps provide mechanism to perform these control constructs
- We jump by assigning a new value to the program counter PC

# Jumps

For example, consider an `if`

# Jumps

icode	meaning
7	Compare $rA$ as 8-bit 2's-complement to 0 if $rA \leq 0$ set $pc = rB$ else increment $pc$ as normal

Instruction icode 7 provides a **conditional** jump

- Real code will also provide an **unconditional** jump, but a conditional jump is sufficient

# Writing Code

We can now write any\* program!

- When you run code, it is being turned into instructions like ours
- Modern computers use a larger pool of instructions than we have (we will get there)

\*we do have some limitations, since we can only represent 8-bit values and some operations may be tedious.

# Our code to this machine code

How do we turn our control constructs into jump statements?

# if/else to jump

# while to jump

# Function Calls

# Encoding Instructions

Example 3: `if r0 < 9 jump to 0x42`

# Instructions

icode	b	meaning
0		$rA = rB$
1		$rA \&= rB$
2		$rA += rB$
3	0	$rA = \sim rA$
	1	$rA = !rA$
	2	$rA = -rA$
	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 $pc + 1$
	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
7		Compare $rA$ as 8-bit 2's-complement to 0 if $rA \leq 0$ set $pc = rB$ else increment $pc$ as normal