



Toy Instruction Set Architecture

CS 2130: Computer Systems and Organization 1
February 13, 2026

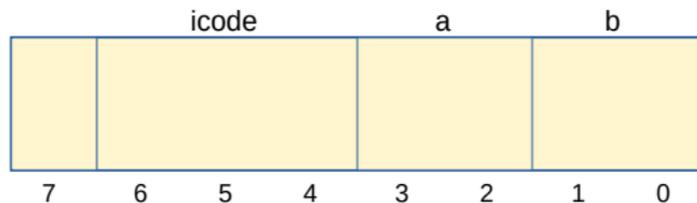
Announcements

- Homework 3 due Monday on Gradescope
- Midterm 1 next Friday (February 20, 2026) in class
 - Written, closed notes
 - If you have SDAC, please schedule ASAP
- Review session in class next Wednesday

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



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

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 $pc + 1$

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!

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

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

Function Calls

Memory

What kinds of things do we put in memory?

- Code: binary code like instructions in our example ISA
 - Intel/AMD compatible: x86_64
 - Apple Mx and Ax, ARM: ARM
 - And others!
- Variables: we may have more variables that will fit in registers
- Data Structures: organized data, collection of data
 - Arrays, lists, heaps, stacks, queues, ...

Dealing with Variables and Memory

What if we have many variables? Compute: $x += y$

Arrays

Array: a sequence of values (collection of variables)
In Java, arrays have the following properties:

- Fixed number of values
- Not resizable
- All values are the same type

Arrays

Array: a sequence of values (collection of variables)
In Java, arrays have the following properties:

- Fixed number of values
- Not resizable
- All values are the same type

How do we store them in memory?

Arrays

Storing Arrays

In memory, store array sequentially

- Pick address to store array
- Subsequent elements stored at following addresses
- Access elements with math

Example: Store array *arr* at 0x90

- Access *arr*[3] as $0x90 + 3$ assuming 1-byte values

What's Missing?

What are we missing?

- Nothing says “this is an array” in memory
- Nothing says how long the array is