



# Toy Instruction Set Architecture

CS 2130: Computer Systems and Organization 1  
September 24, 2025

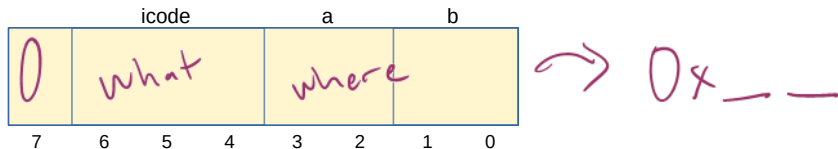
# Announcements

- Homework 3 due Monday at 11:59pm on Gradescope
- Midterm 1 next Friday (October 3, 2025) in class
  - Written, closed notes
  - If you have SDAC, please schedule ASAP
- No lab check-off on Mondays

# 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



# Toy ISA 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		$rA = \text{read from memory at address } rB$
5		write $rA$ to memory at address $rB$
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
7		Compare $rA$ as 8-bit 2's-complement to 0 if $rA \leq 0$ set $pc = rB$ else increment $pc$ as normal

# 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		$rA = \text{read from memory at address } rB$
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

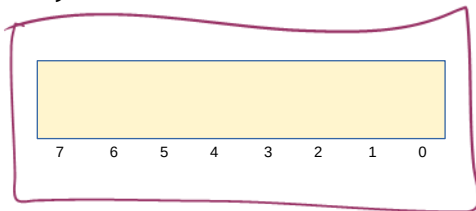
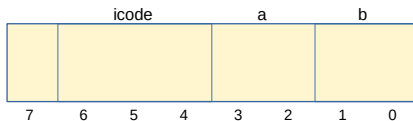
icode	b	meaning
1		$rA \&= rB$
2		$rA += rB$
3	0	$rA = \sim rA$
	1	$rA = !rA$
	2	$rA = -rA$
6	1	$rA \&= \text{read from memory at } pc + 1$
	2	$rA += \text{read from memory at } pc + 1$

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

# Immediate values

icode 6 provides literals, **immediate** values

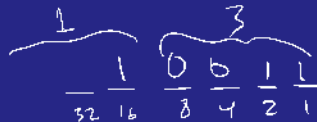
icode	b	action
6	0	$rA = \text{read from memory at } pc + 1$
	1	$rA \&= \text{read from memory at } pc + 1$
	2	$rA += \text{read from memory at } pc + 1$
	3	$rA = \text{read from memory at the address stored at } pc + 1$
		For icode 6, increase $pc$ by 2 at end of instruction



# 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		$rA = \text{read from memory at address } rB$
5		write $rA$ to memory at address $rB$
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	$rA = \text{read from memory at the address stored at } pc + 1$
7		For icode 6, increase $pc$ by 2 at end of instruction
		Compare $rA$ as 8-bit 2's-complement to 0
		if $rA \leq 0$ set $pc = rB$
		else increment $pc$ as normal

$r1 += 19$   $0 \times 13$

↓ 0 6 1 2  
operation rA b  
icode  
#13  
immediat

0110 0110  
6 6

pc pc+1  
66 13

# 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		$rA = \text{read from memory at address } rB$ ←
5		write $rA$ to memory at address $rB$
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
7		Compare $rA$ as 8-bit 2's-complement to 0 if $rA \leq 0$ set $pc = rB$ else increment $pc$ as normal

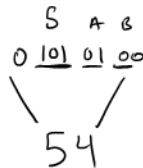
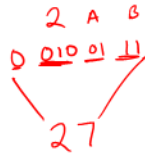
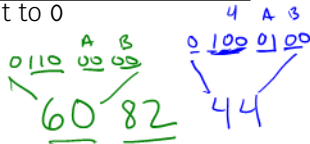
$M[0x82] += r3$

→  $r0 = 0x82$


→  $r1 = M[r0]$

→  $r1 += r3$

→  $M[r0] = r1$

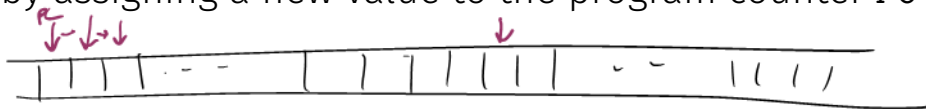


# Writing Code: Homework Hints

1. Write pseudocode that does the desired task
- 2-3 ... deal with control flow
4. Split multi-operation lines into series of single-operation lines  
 $x = y - z$ ; becomes  $x = y$ ;  $x -= z$ ;
5. Convert operations to those in our instruction set  
 $x -= z$ ; becomes  $w = z$ ;  $w = -w$ ;  $x += w$ ;   $x += -w$
6. ... deal with loops
7. Assign variables to our four registers, ex:  $r0=x$ ,  $r1=y$ ,  $r2=z$ ,  $r3=w$   
 $r0 = r1$ ;  $r3 = r2$ ;  $r3 = -r3$ ;  $r0 += r3$
- 10- Write those instructions into triples, then hex

# 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

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

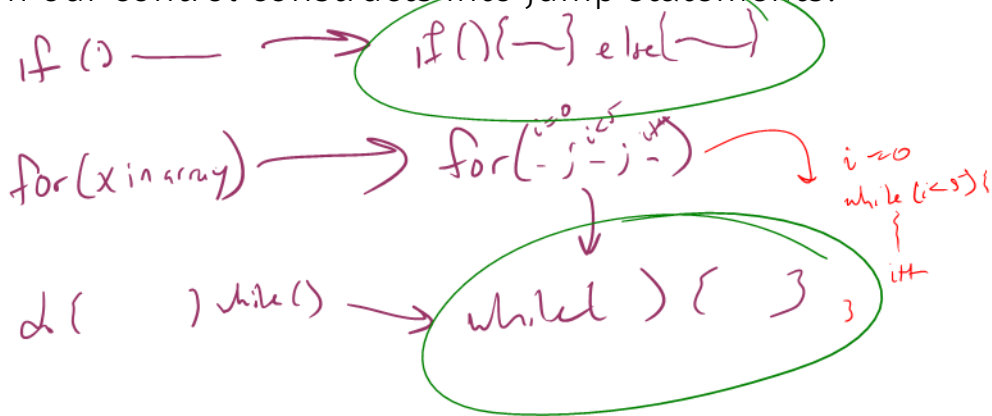
We can now write any\* program!

- 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

## Pseudocode using if/else

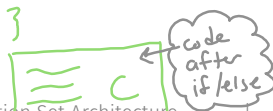
```
if ( D ) {
```



```
} else {
```



```
}
```



### Notes!

if D is true:  
run code in A, then C  
→ skip B

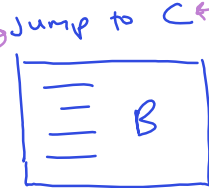
if D is false:  
run code in B, then C  
→ skip A

## Using Jumps

```
if ( !D ) jump to B
```



↑  
address  
of first  
instruction  
of B



↑  
address  
of first  
instruction  
of C



← no jump.  
After  
B, do  
C.

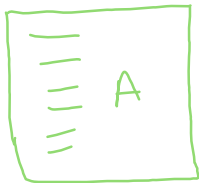
Code is in memory (think array)



# while to jump

Pseudocode of while loop:

```
while ( C ) {
```



```
}
```



Code after loop

Notes:

→ if C is true, run code in A, then go back and check C again

if C is false, skip A, go to B

We have two options!

- jump to the check of C

- check at end of A before jumping back

option 1

address of first instruction of ...  
↓

option 2

D → if (!C) jump to B



jump to D unconditionally



if (!C) jump to B



if (C) jump to A

