Function Calls, Memory Instruction Set Architectures

CS 2130: Computer Systems and Organization 1
September 29, 2025

Announcements

- Homework 3 due Wednesday at 11:59pm on Gradescope
- Midterm 1 Friday (October 3, 2025) in class
  - Written, closed notes
  - If you have SDAC, please schedule ASAP

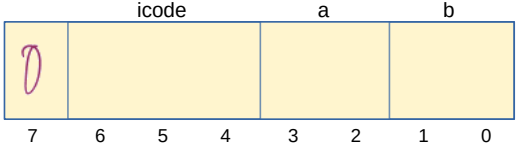
Instructions

icode	b	meaning
0		`rA = rB`
1		`rA &= rB`
2		`rA += rB`
3	0	`rA = ~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 <= 0` set `pc = rB` else increment `pc` as normal

Encoding Instructions

Encoding of Instructions

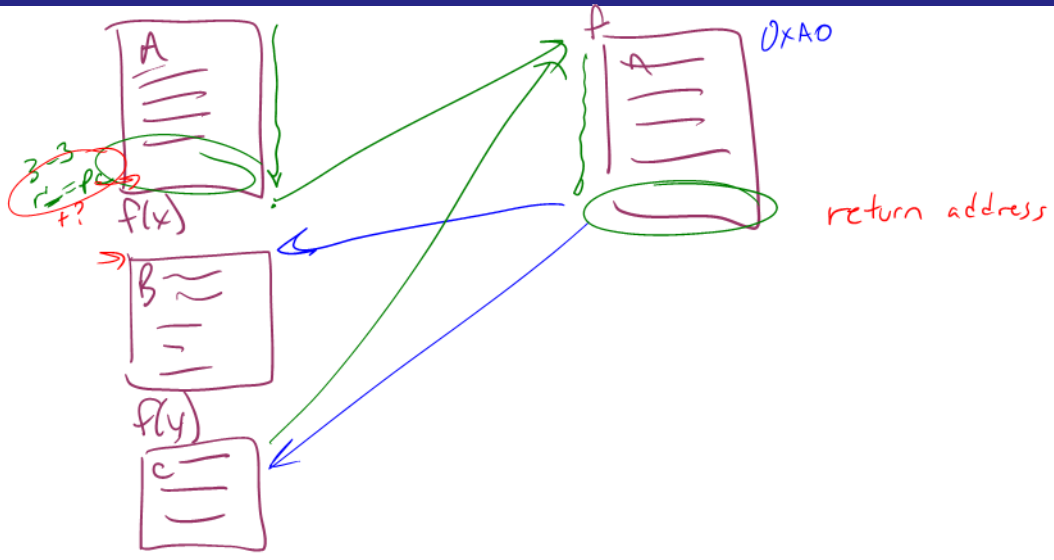
- 3-bit icode (which operation to perform)
  - Numeric mapping from icode to operation
- Which registers to use (2 bits each)
- Reserved bit for future expansion



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`

Function Calls



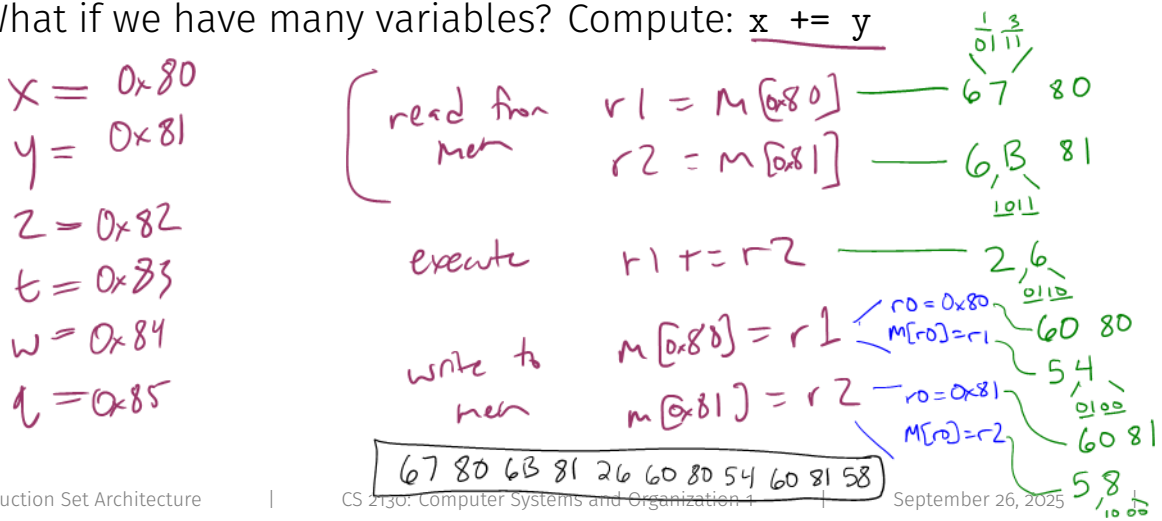
Memory

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

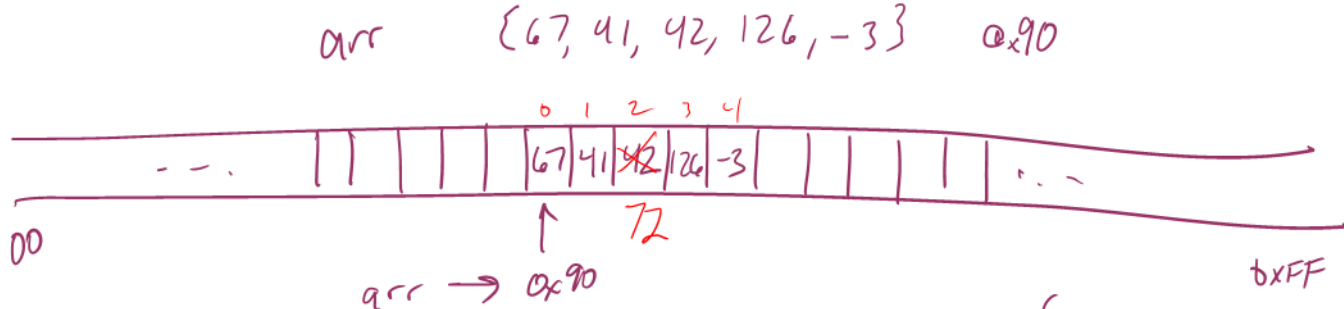
Arrays

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



arr[0] → M[0x90]

arr[3] → M[93] → M[0x90 + 3]

arr

arr[2] = 0x72

↳ M[0x93] = 0x72

Storing Arrays

In memory, store array sequentially

- Pick address to store array
- Subsequent elements stored at following addresses
- Access elements with math – 32-64 bit

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

Instructions

icode	b	meaning
0		`rA = rB`
1		`rA &= rB`
2		`rA += rB`
3	0	`rA = ~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 <= 0` set `pc = rB` else increment `pc` as normal

Instruction Set Architecture

Instruction Set Architecture (ISA) is an abstract model of a computer defining how the CPU is controlled by software

- Conceptually, set of instructions that are possible and how they should be encoded
- Results in many *different* machines to implement same ISA
  - Example: How many machines implement our example ISA?
- Common in how we design hardware

Instruction Set Architecture

Instruction Set Architecture (ISA) is an abstract model of a computer defining how the CPU is controlled by software

- Provides an abstraction layer between:
  - Everything computer is really doing (hardware)
  - What programmer using the computer needs to know (software)
- Hardware and Software engineers have freedom of design, if conforming to ISA
- Can change the machine without breaking any programs

Instruction Set Architecture

Instruction Set Architecture (ISA) is an abstract model of a computer defining how the CPU is controlled by software

- Provides an abstraction layer between:
  - Everything computer is really doing (hardware)
  - What programmer using the computer needs to know (software)

CSO: covering many of the times we'll need to think across this barrier

Instruction Set Architecture

Backwards compatibility

- Include flexibility to add additional instructions later
- Original instructions will still work
- Same program can be run on PC from 10+ years ago and new PC today

Most manufacturers choose an ISA and stick with it

- Notable Exception: Apple

Our Instruction Set Architecture

What about our ISA?

- Enough instructions to compute what we need
- As is, lot of things that are painful to do
  - This was on purpose! So we can see limitations of ISAs early

Our Instruction Set Architecture

What about our ISA?

- Enough instructions to compute what we need
- As is, lot of things that are painful to do
  - This was on purpose! So we can see limitations of ISAs early
- Add any number of new instructions using the reserved bit (7)