

Function Calls, Memory Instruction Set Architectures

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

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

Quiz Questions – Quiz 1

Q5.3 XOR

1 Point

Suppose we then shift it back and xor it with the original, like

```
((0xCA >> 3) << 3) ^ 0xCA.
```

The result is:

- ☒ the same for both signed and unsigned integers
- ☐ larger for signed than unsigned integers
- ☐ larger for unsigned than signed integers
- ☐ there is no way to know

Quiz Questions – Quiz 3

Q8 Counter

1 Point

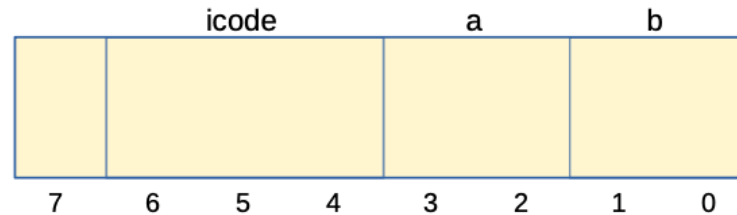
To build a 4-bit counter circuit, we could directly connect the output of the increment circuit back to the input.

- ☐ True
- ☒ False

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

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!

*Different CPUs have different ISAs.
ISA describes how to understand
the binary.*

- 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: $x += y$

$x = 0x80$

$y = 0x81$

read from memory	$r1 = M[0x80]$ $r2 = M[0x80]$
execute	$r1 += r2$
write to memory	$M[0x80] = r1$ $M[0x80] = r2$

0111

✓

67

80

6B

81

1011

26

0110

0110

60

$r0 = 0x80 \rightarrow 6080$

54

$M[r0] = r1 \rightarrow 54$

60

$r0 = 0x81 \rightarrow 6081$

58

$M[r0] = r2 \rightarrow 58$

58

67 80 6B 81 26 60 80 54 60 81 58

Arrays

Array: a sequence of values (collection of variables)

In Java, arrays have the following properties:

*they do have order
index by integers.*

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

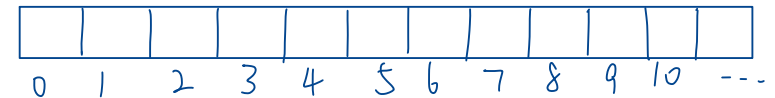
Arrays

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?

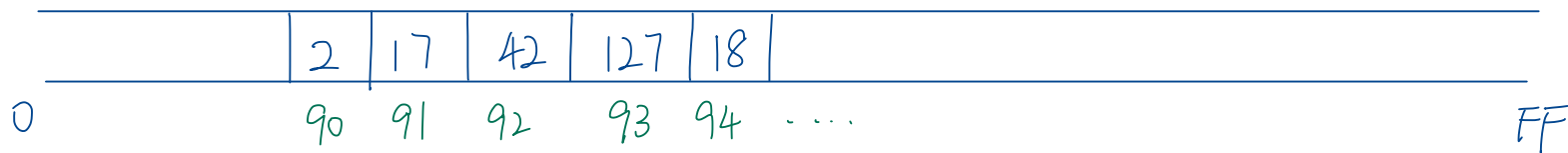


Sequentially in Memory

Arrays

$\text{arr} = \{2, 17, 42, 127, 18\} @ 0x90$

I assume all these are one byte so that each will fit in one of these slots.



$$\text{arr}[3] = \underbrace{0x90}_{\text{position of where our array start}} + \underbrace{3}_{\text{index}} = 0x93$$

→ position of where our array start.

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

arr[5] = 0x27
M[90+5] = 0x27

What's Missing?

What are we missing?

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

Memory just stores a bunch of bytes.

They might be part of the array? Part of a large number? Part of an instruction? I don't know what they are. They're just values.

Some of our coding languages like Java, stores additional information about your arrays, like how long they are, what type they are.

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

Instructions Set Architecture

We've designed an 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 *Intel and AMD's CPU*
 - Example: How many machines implement our example ISA? *both using x86-64 ISA.*
- Common in how we design hardware *So they can run the same program.*

Instructions 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

Lots of flexibility and freedom to build things that would be faster, like hyperthreading. I don't worry about on the software side. → Just make sure the code can be compiled to ISA. I can run it on hardware.

Instructions 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

We're in general at this point, going to start staying just above this barrier and to the software side.

Instructions Set Architecture

It's easy to handy
Backwards compatibility

0
reserved

- Include flexibility to add additional instructions later *I can add more things.*
- Original instructions will still work *My new machine still can run the old code.
old instructions still there.*
- 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

*One manufacturer that has enough
following to be able to make these changes and
still come out ahead.*

My new machine is much faster.

There may be some issues, but that is another story

Our Instructions 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 Instructions 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)

Our Instructions 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)
- Missing something important: *Help to put variables in memory*