Instruction Set Architectures, Stacks

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 3 due today
- Homework 4 available soon due Monday after Spring Break on Gradescope
  – You have written most of this code already

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 (6B: 1011)

26 (0110)

r0 = 0x80 → 60 80

M[r0] = r1 → 54

r0 = 0x81 → 60 81

M[r0] = r2 → 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:

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

they do have order index by integers.

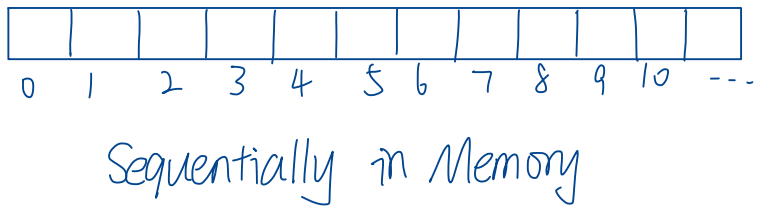
Arrays

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

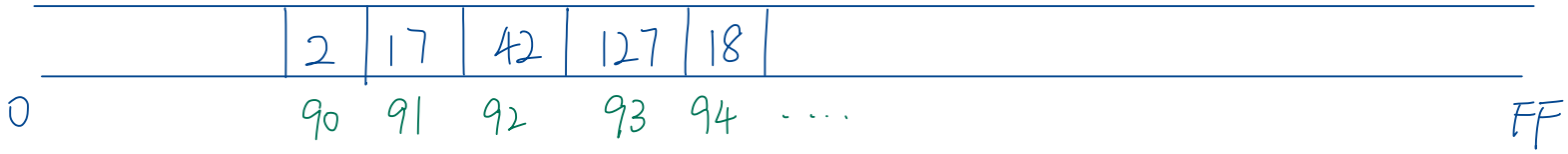
How do we store them in memory?



Arrays

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

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



arr[3] = 0x90 + 3 = 0x93

3 → index

0x90 → 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 = ~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

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 both using x86_64 ISA. So they can run the same program.
  – Example: How many machines implement our example ISA?
- Common in how we design hardware

Instructions Set Architecture

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

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.

- 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

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

- 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

0
reserved

I can add more things.

My new machine still can run the old code.
Old instructions still there.

Most manufacturers choose an ISA and stick with it

- Notable Exception: Apple

My new machine is much faster
There may be some issues, but that is another story

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

① IBM's PowerPC (1994 - 2005) MacOS 10.
(PowerBook, PowerMac → PowerPC too hot/not scaling)

② ⇒ Intel x86 (2006 - 2020)

RISC → CISC → RISC

① All programs had to be recompiled.
② Hardware completely changed.
③ Backward compatibility became an issue.

Apple didn't want old programs to break. So they created: Rosetta
Rosetta is a translation layer that dynamically converted PowerPC instruction into x86 instructions.
It essentially simulating the old ISA. Eventually, Apple "sunset" it and dropped support.

③ ⇒ Apple Silicon (2020 - now) M1 chip (their own ARM chip)

This is another ISA migration.
They resurrected Rosetta (Rosetta 2). It translates x86 → ARM on the fly.

① Changing ISA is a massive ecosystem shift.
② Most companies avoid changing ISA.
③ ISA stability enables software longevity.
④ Backward compatibility matters!
⑤ Hardware / software boundaries matter!
⑥ Binary translation preserves ecosystems.
⑦ ISA decisions affect decades of software.

What about real ISAs?

CISC: Lots of instructions, some very specialized.

eg. x86-64: thousands of instructions, very complex.

RISC: fewer instructions. but each executes very efficiently.

e.g.: RISC-V: 47 basic instructions. small and clean.

extensions: grow to hundreds of instructions.

ARM: 500 ~ 1000 instructions.

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)

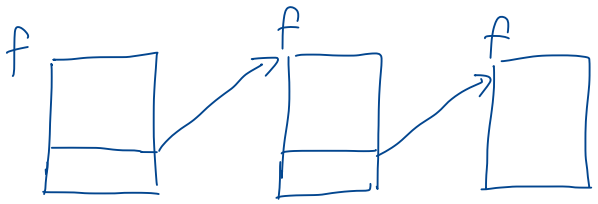
Our Instructions Set Architecture

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*

Storing Variables in Memory

So far... we/compiler chose location for variable



Consider the following example:

```
f(x):
    a=x
    if (x <= 0) return 0
    else return f(x-1) + a
```

x=5

a=5

f(4)+5 ⇒ Sums up the numbers between one and x

Recursion

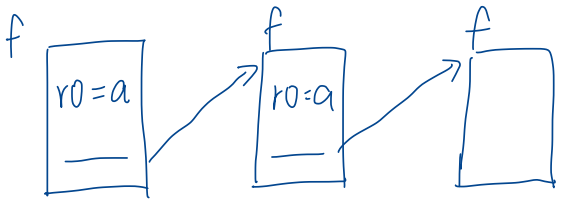
- The formal study of a function that calls itself

while it computing the solution or the result for F, it actually calls itself in a smaller input.

Storing Variables in Memory

```
f(x):
    a=x
    if (x <= 0) return 0
    else return f(x-1) + a
```

Where do we store a?



Save it in a register R0 ? No !

save it to memory ? M[0x80] No!

override this value over and over

When I call the same code again. jump to the same code again. I accidently overwrite the value I really want.

We need something that will help us to organize memory so that we keep track of these variables.