Instruction Set Architectures, Stacks

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 4 available today due Friday at 11:59pm on Gradescope
  - Note the earlier deadline!
  - You have written most of this code already
  - Lab 6 may provide a fast way to get started

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
  - Example: How many machines implement our example ISA?
- Common in how we design hardware

Intel and AMD's CPU both using x86_64 ISA. 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

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

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

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

- Notable Exception: Apple

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

UNIVERSITY of VIRGINIA

What about real ISAs?

CISC: Lots of instructions, some very specialized.
e.g. 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)

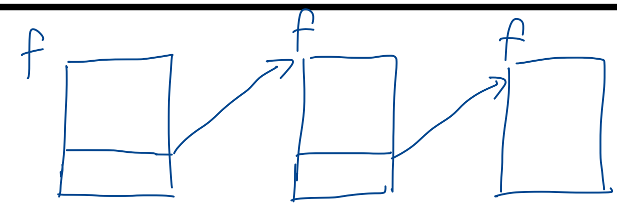
Our Instructions Set Architecture

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
```

sums up the numbers between one and x.

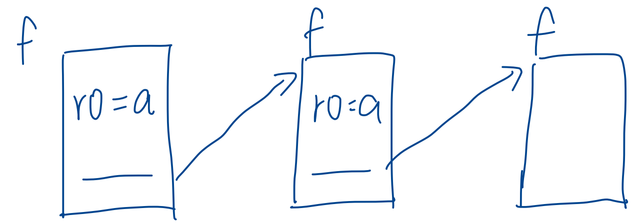
Recursion

- 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 set),

Storing Variables in Memory

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

Where do we store a?



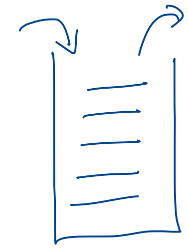
Save it in a register R0? No!

save it to memory? No!

override this value over and over

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

The Stack



stack of plates

Stack - a last-in-first-out (LIFO) data structure

- The solution for solving this problem

rsp - Special register - the *stack* pointer

- Points to a special location in memory — have the address. (the index in memory of a certain point).
- Two operations most ISAs support:
  - – push - put a new value on the stack
  - – pop - return the top value off the stack

The Stack: Push and Pop

push r0

- Put a value onto the "top" of the stack
  - rsp -= 1
  - M[rsp] = r0

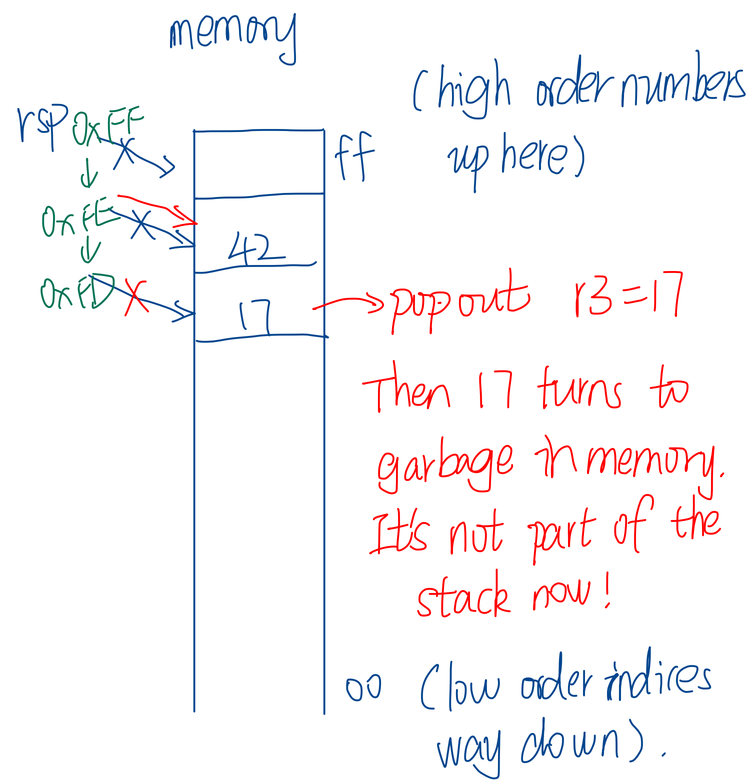
pop r2

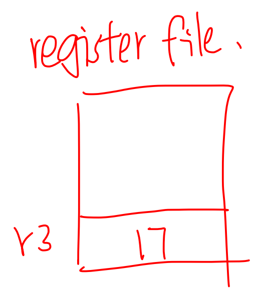
- Read value from "top", save to register
  - r2 = M[rsp]
  - rsp += 1

push 42

push 17

$\boxed{\text{pop r3}}$





The Stack: Push and Pop

push(17)

push(23)

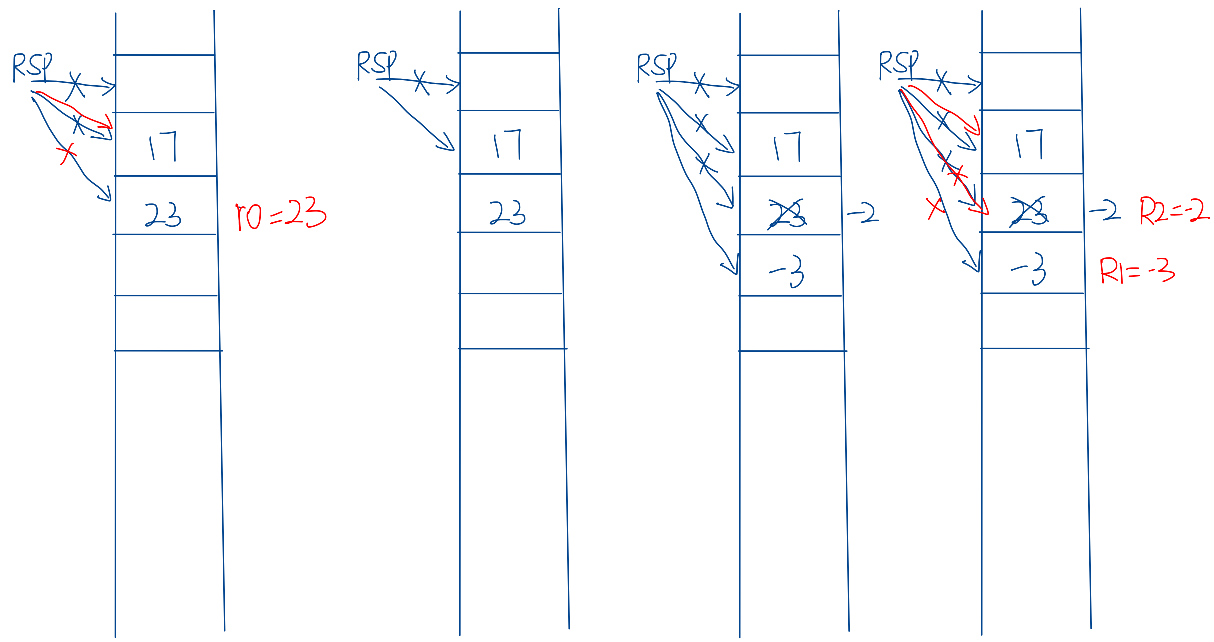
x = pop (R0)

push(-2)

push(-3)

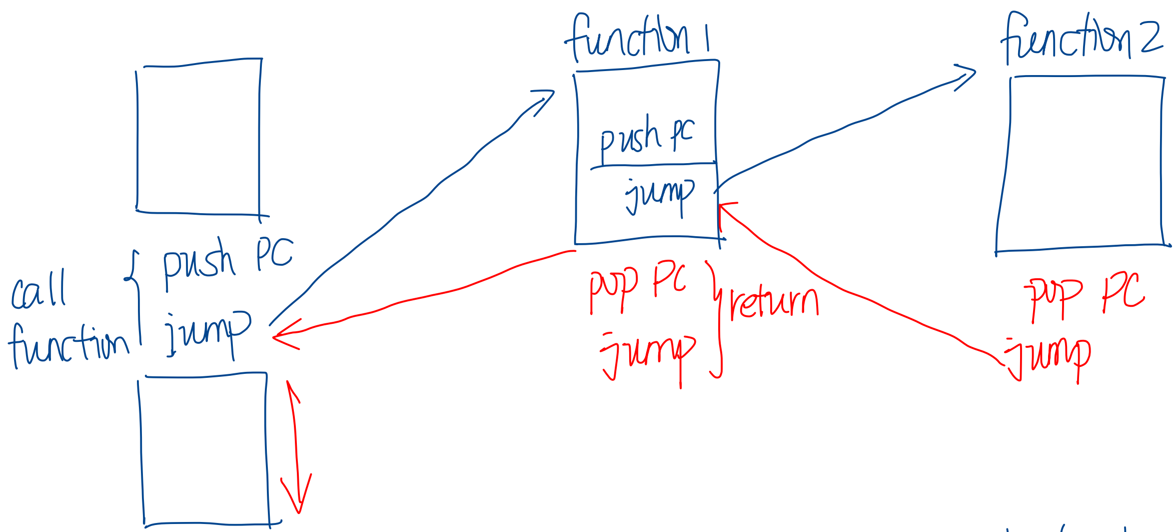
y = pop (R1)

z = pop (R2)



The Stack: Push and Pop

Function Calls



What about parameters ?

return value ?

calling conventions
- r2, r3 have operands
- r0 has return value.

A short aside…

Time to take over the world!

Backdoors

→ allow an attacker/developer to take complete control of your system — often without you knowing it.

Backdoor: secret way in to do new unexpected things

- Get around the normal barriers of behavior
- Ex: a way in to allow me to take complete control of your computer

operating system has security checks: passwords, permissions, firewalls ⇒ all meant to control "who can do what"

Someone installs a backdoor ⇒ sneak past those protections and gain control without going through the normal process.

Backdoors

maybe a program ? scripts ?

Exploit - a way to use a vulnerability or backdoor that has been created

- Our exploit today: a **malicious payload**
  - – A passcode and program
  - – If it ever gets in memory, run my program regardless of what you want to do

backdoor : hidden entrance

exploit : how you find it, open it and use it to get inside.

Our Hardware Backdoor

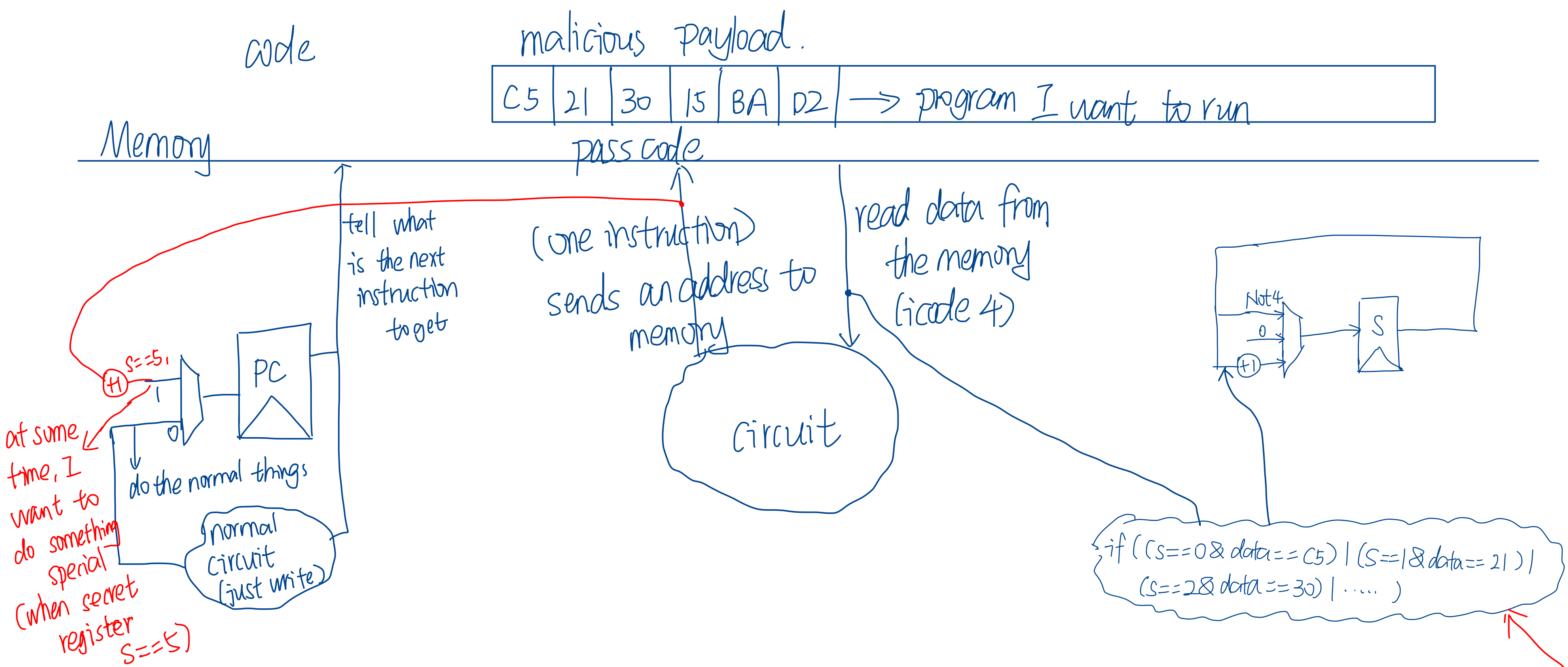
Our backdoor will have 2 components

- Passcode: need to recognize when we see the passcode
- Program: do something bad when I see the passcode

The hardware needs to recognize it when it appears.
(Like secret knock on a door, if you hear that pattern, you know someone special wants in.

Once the hardware recognizes the passcode, it executes some hidden or harmful behavior.

Our Hardware Backdoor



Something not icode 4: keep the original value

if it is icode4:
① reset counter if that's not part of my passcode
② increasmentor: under special condition

Our Hardware Backdoor

Will you notice this on your chip?

Our Hardware Backdoor

Will you notice this on your chip?

- Modern chips have **billions** of transistors
- We're talking adding a few hundred transistors

Our Hardware Backdoor

Will you notice this on your chip?

- Modern chips have **billions** of transistors
- We're talking adding a few hundred transistors
- Maybe with a microscope? But you'd need to know where to look!

If I had a microscope and I knew exactly where to look, I probably can find it.

Or, maybe, I'm very, very lucky.

But - Most exploits are going to be found after somebody tries to use them.

Our Hardware Backdoor

Have you heard about something like this before?

Our Hardware Backdoor

Have you heard about something like this before?

- Sounds like something from the movies
- People claim this might be happening

Our Hardware Backdoor

Have you heard about something like this before?

- Sounds like something from the movies
- People claim this might be happening
- To the best of my knowledge, no one has ever admitted to falling in this trap

Ethics, Business, Tech

Are there reasons to do this? Not to do this?

- No technical reason not to, it’s easy to do!

Ethics, Business, Tech

Are there reasons to do this? Not to do this?

- No technical reason not to, it's easy to do!
- Ethical implications
- Business implications (lawsuits, PR, etc)

Ethics, Business, Tech

Are there reasons to do this? Not to do this?

- No technical reason not to, it’s easy to do!
- Ethical implications
- Business implications (lawsuits, PR, etc)

Can we make a system where one bad actor can’t break it?

Ethics, Business, Tech

Are there reasons to do this? Not to do this?

- No technical reason not to, it's easy to do!
- Ethical implications
- Business implications (lawsuits, PR, etc)

Can we make a system where one bad actor can't break it?

- Code reviews, double checks, verification systems, automated verification systems, ...

Why does this work?

Why?

Why does this work?

- **It's all bytes!**
- Everything we store in computers are bytes
- We store code and data in the same place: memory

It's all bytes

Memory, Code, Data... It's all bytes!

- **Enumerate** - pick the meaning for each possible byte
- **Adjacency** - store bigger values together (sequentially)
- **Pointers** - a value treated as address of thing we are interested in

Enumerate

Enumerate - pick the meaning for each possible byte

What is 8-bit 0x54?

Unsigned integer	eighty-four
Signed integer	positive eighty-four
Floating point w/ 4-bit exponent	twelve
ASCII	capital letter T: `T`
Bitvector sets	The set $\{2, 3, 5\}$
Our example ISA	Flip all bits of value in r1

Adjacency

Adjacency - store bigger values together (sequentially)

- An array: build bigger values out of many copies of the same type of small values
  – Store them next to each other in memory
  – Arithmetic to find any given value based on index

Adjacency

Adjacency - store bigger values together (sequentially)

- Records, structures, classes
  - Classes have fields! Store them adjacently
  - Know how to access (add offsets from base address)
  - If you tell me where object is, I can find fields

Pointers

Pointers - a value treated as address of thing we are interested in

- A value that really points to another value
- Easy to describe, hard to use properly
- We'll be talking about these a lot in this class!

Pointers

Pointers - a value treated as address of thing we are interested in

- Give us strange new powers (represent more complicated things), e.g.,
  - Variable-sized lists
  - Values that we don’t know their type without looking
  - Dictionaries, maps

Programs Use These!

How do our programs use these?

- Enumerated icodes, numbers
- Ajacently stored instructions (PC+1)
- Pointers of where to jump/goto (addresses in memory)