

Instruction Set Architectures, Stacks

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

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

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

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

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

Instructions 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

What about real ISAs?

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 $(x \leq 0)$ return 0

else return $f(x-1) + a$

Recursion

- The formal study of a function that calls itself

Storing Variables in Memory

$f(x)$:

$a = x$

if $(x \leq 0)$ return 0

else return $f(x-1) + a$

Where do we store a ?

The Stack

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
- 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
 - $\text{rsp} -= 1$
 - $\text{M}[\text{rsp}] = \text{r0}$

pop r2

- Read value from “top”, save to register
 - $\text{r2} = \text{M}[\text{rsp}]$
 - $\text{rsp} += 1$

The Stack: Push and Pop

The Stack: Push and Pop

Function Calls

A short aside...

Time to take over the world!

Backdoors

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

Backdoors

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

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

Our Hardware Backdoor

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!

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

Signed integer

Floating point w/ 4-bit exponent

ASCII

Bitvector sets

Our example ISA

eighty-four

positive eighty-four

twelve

capital letter T: T

The set {2, 3, 5}

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
- Adjacentlly stored instructions (PC+1)
- Pointers of where to jump/goto (addresses in memory)