

Arrays

CS 2130: Computer Systems and Organization 1

September ~~23~~, 2022

26

Announcements

- Homework 3 due tonight at 11pm on Gradescope
 - Please remember that homeworks are **individual** assignments if not stated otherwise on the assignment
 - Your code should be space-separated bytes as hex values
- Exam 1 Friday (in class)
 - Bring questions for Wednesday (review!)
 - For SDAC accommodations, please schedule a time with their testing center

Quiz Review

$$\rightarrow r_0 + = -r_2$$

$$\begin{matrix} 1 & 2 \\ 0 & 1 & 1 & 0 \end{matrix}$$

$$06$$

$$r_1 = r_2$$

$$\begin{matrix} 1 & 1 \\ 0 & 1 & 0 & 1 \end{matrix}$$

$$55$$

$$r_1 = -r_1$$

$$\begin{matrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{matrix}$$

$$r_0 + = r_1$$

What kinds of things do we put in memory?

- Variables: we may have more variables that will fit in registers
- Data Structures: organized data, collection of data
 - Arrays, lists, heaps, stacks, queues, ...
- Code: binary code like instructions in our example machine
 - Intel/AMD compatible: x86_64
 - Apple Mx and Ax, ARM: ARM
 - And others!

Quiz Question 8

x = 5

y = 2

i = -5

do {

 x += y

 i++

} while (i <= 0)

Quiz Question 8



-6 FA -5 FB -4 FC -3 FD -2 FE -1 FF 00

$r0 = i$ $r1 = x$
 $r3 = 0x24$ $r2 = 4$

```
x = 5
y = 2
i = -5
do {
    x += y
    i++
} while (i <= 0)
```

x = 5

y = 2

i = -5

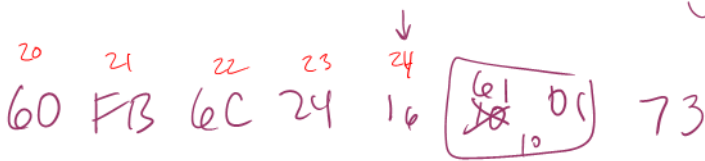
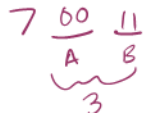
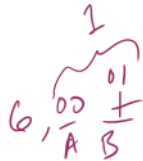
~~tmp = pc~~

x += y

~~i++~~ $r0 + 1$

if (i <= 0) goto tmp

⁰⁰⁰⁰60 ⁻⁵FB
¹¹⁰⁰6C 24
 16
⁰⁰⁰¹61 01



Dealing with Variables and Memory

What if we have many variables?

Quiz Question 8

$r0 = \text{mem addr}$
 $r1, r2 = \text{variables}$

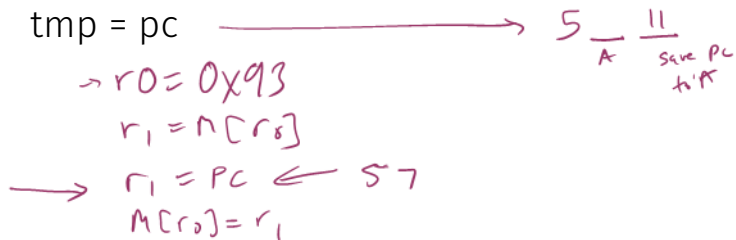
Var	Mem Addr
x	0x90 ←
y	0x91
i	0x92
tmp	0x93
z	0x94
	⋮

x = 5

$r0 = 0x90$
 $r1 = M[r0]$
 $r1 = 5$
 $M[r0] = r1$

...

tmp = pc



Quiz Question 8

Var	Mem Addr
x	0x90
y	0x91
i	0x92
tmp	0x93

...

x += y

$r_0 = 0x90$

$r_1 = M[r_0]$

$r_0 = 0x91$

$r_2 = M[r_0]$

$r_1 += r_2$

$M[r_0] = r_2$

$r_0 = 0x90$

$M[r_0] = r_1$

...

if (i <= 0) goto tmp ←

r_0

r_1

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: [42, 3, 15, 5, 7] at 6x A0

Arr[3]
↓
A0 + 3

Addr.	A0	A1	A2	A3	A4
Values	42	3	15	<u>5</u>	7

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

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		$rA =$ read from memory at address rB
4		write rA to memory at address rB ←
5	0	$rA = \sim rA$
	1	$rA = -rA$
	2	$rA = !rA$
	3	$rA = pc$
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 θ if $rA \leq \theta$ 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

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)

Our Instruction Set Architecture

icode	b	meaning
0		$rA = rB$
1		$rA += rB$
2		$rA \&= rB$
3		$rA =$ read from memory at address rB
4		write rA to memory at address rB
5	0	$rA = \sim rA$
	1	$rA = -rA$
	2	$rA = !rA$
	3	$rA = pc$
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 θ if $rA \leq \theta$ set $pc = rB$ else increment pc as normal

What about real ISAs?

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)
- 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`

Recursion

- The formal study of a function that calls itself

Storing Variables in Memory

```
f(x):  
  a = x  
  if (x <= 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`

- Add a value onto the stack

`M[rsp] = r0`

`rsp += 1`

`pop r2`

- Read top value, save to register

`rsp -= 1`

`r2 = M[rsp]`

The Stack: Push and Pop

The Stack: Push and Pop