64-bit and Endianness

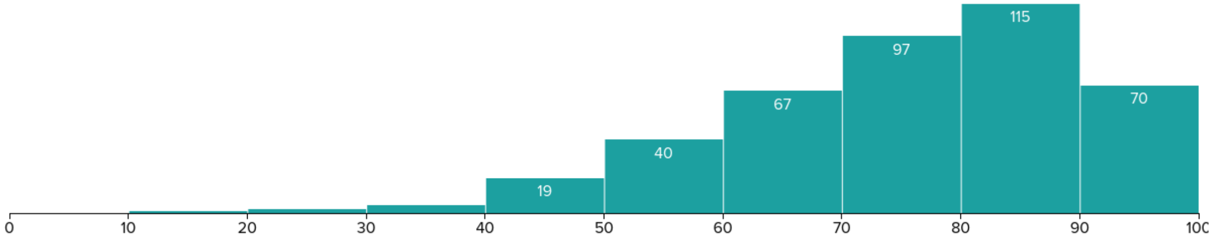
CS 2130: Computer Systems and Organization 1

October 7, 2022

Announcements

- Homework 4 due Monday, 11pm on Gradescope
- Quiz 5 opens at 5pm, due Monday by 8am
- Exam 1 scores released after class

Exam 1



Statistics

Mean	75.4
Median	77.0
Std. Dev.	15.4

Exam 1: Most Missed Questions

r0 = r2 & 80

Suppose we extended the ISA simulator you wrote in Lab 4 with the following code:

1100 00 10
C A B

```
if (reserved == 1 && icode == 4) {
    R[a] = R[b] & M[oldPC + 1];
    return oldPC + ___;
}
```

Using the new instruction above at least once, write a program that determines if the contents of register 2 is a negative number in two's complement and stores the result in register 0. That is, if the high order bit of register 2's value is a 1, your program should store a `0x01` in register 0. Answer in hexadecimal bytes, separated by spaces. *Hint: you may need to write additional instructions.*

C2 80 52 52

Exam 1: Most Missed Questions

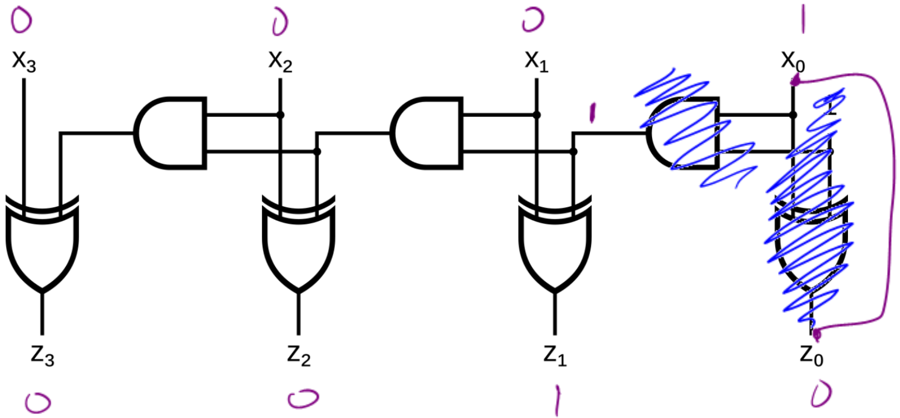
icode	b	meaning
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		`rA` = read from memory at address `rB`
4		write `rA` to memory at address `rB`
5	0	`rA = ~rA`
	1	`rA = -rA`
	2	`rA = !rA`
	3	`rA = pc`
6	0	`rA` = read from memory at `pc + 1`
	1	`rA` += read from memory at `pc + 1`
	2	`rA` &= read from memory 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 `0`
		if `rA <= 0` set `pc = rB`
		else increment `pc` as normal

! (!(80) = 0) = 01

!(!(00) = 1) = 00

Exam 1: Most Missed Questions

In class, we discussed a 4-bit increment circuit below that added 1 to the input.



How can we change this circuit to instead increment by 2, i.e., `x += 2`? Draw the new circuit below.

Note: you should not use more gates than the original circuit.

Patents and Copyright

Can we patent our ISA? Should we?

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 = ~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 `0`
		if `rA <= 0` set `pc = rB`
		else increment `pc` as normal

Patents and Copyright

Copyright

- “Everyone is a copyright owner. Once you create an original work and fix it, like taking a photograph, writing a poem or blog, or recording a new song, you are the author and the owner.”

from https://www.copyright.gov/what-is-copyright/

Patent

- “Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.”

from 35 U.S.C. 101

Patents

In software and hardware, patents become messy

- Code is a description of a process we want the computer to do
- Do not have to implement the process to patent it

Question: Should we patent something like our ISA?

Patents

In software and hardware, patents become messy

- Code is a description of a process we want the computer to do
- Do not have to implement the process to patent it

Question: Should we patent something like our ISA?

What is the current state of the art?

Common Approaches to Software

How can we get value from what we create?

- Copyright - distribute closed source software
- License Agreements (in contract law)
- Always innovate

Moving On

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 = ~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 `0`
		if `rA <= 0` set `pc = rB`
		else increment `pc` as normal

So far, we've been dealing with an 8-bit machine!

64-bit Machines

64-bit machine: The **registers** are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

64-bit Machines

64-bit machine: The **registers** are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access?

64-bit Machines

64-bit machine: The **registers** are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? 256 bytes

64-bit Machines

64-bit machine: The **registers** are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? 256 bytes
- Late 70s - 16 bits:

64-bit Machines

64-bit machine: The **registers** are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? 256 bytes
- Late 70s - 16 bits: 65,536 bytes

64-bit Machines

64-bit machine: The **registers** are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? 256 bytes
- Late 70s - 16 bits: 65,536 bytes
- 80s - 32 bits:

64-bit Machines

64-bit machine: The **registers** are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? 256 bytes
- Late 70s - 16 bits: 65,536 bytes
- 80s - 32 bits: ≈ 4 billion bytes   4 GiB

64-bit Machines

64-bit machine: The **registers** are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? 256 bytes
- Late 70s - 16 bits: 65,536 bytes
- 80s - 32 bits: $\approx$ 4 billion bytes
- Today's processors - 64 bits:

64-bit Machines

64-bit machine: The **registers** are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? 256 bytes
- Late 70s - 16 bits: 65,536 bytes
- 80s - 32 bits: $\approx$ 4 billion bytes 4 GiB
- Today's processors - 64 bits: 2^{64} addresses ?

Aside: Powers of Two

Powers of Two

Value	base-10	Short form	Pronounced
2^{10}	1024	Ki	Kilo
2^{20}	1,048,576	Mi	Mega
2^{30}	1,073,741,824	Gi	Giga
2^{40}	1,099,511,627,776	Ti	Tera
2^{50}	1,125,899,906,842,624	Pi	Peta
2^{60}	1,152,921,504,606,846,976	Ei	Exa

Example: 2^{27} bytes

Aside: Powers of Two

Powers of Two

Value	base-10	Short form	Pronounced
2^{10}	1024	Ki	Kilo
2^{20}	1,048,576	Mi	Mega
2^{30}	1,073,741,824	Gi	Giga
2^{40}	1,099,511,627,776	Ti	Tera
2^{50}	1,125,899,906,842,624	Pi	Peta
2^{60}	1,152,921,504,606,846,976	Ei	Exa

Example: 2^{27} bytes $= 2^{7} \times 2^{20}$ bytes

Aside: Powers of Two

Powers of Two

Value	base-10	Short form	Pronounced
2^{10}	1024	Ki	Kilo
2^{20}	1,048,576	Mi	Mega
2^{30}	1,073,741,824	Gi	Giga
2^{40}	1,099,511,627,776	Ti	Tera
2^{50}	1,125,899,906,842,624	Pi	Peta
2^{60}	1,152,921,504,606,846,976	Ei	Exa

Example: 2^{27} bytes $= 2^7 \times 2^{20}$ bytes $= 2^7$ MiB $= 128$ MiB

64-bit Machines

How much can we address with 64-bits?

64-bit Machines

How much can we address with 64-bits?

- 16 EiB (2^{64} addresses = $2^4 \times 2^{60}$)

64-bit Machines

How much can we address with 64-bits?

- 16 EiB (2^{64} addresses = $2^4 \times 2^{60}$)
- But I only have 8 GiB of RAM

A Challenge

There is a disconnect:

- Registers: 64-bits values
- Memory: 8-bit values (i.e., **1 byte** values)
    - Each address addresses an 8-bit value in memory
    - Each address points to a 1-byte slot in memory

A Challenge

There is a disconnect:

- Registers: 64-bits values
- Memory: 8-bit values (i.e., **1 byte** values)
  - Each address addresses an 8-bit value in memory
  - Each address points to a 1-byte slot in memory
- How do we store a 64-bit value in an 8-bit spot?

Rules

0x 00|AB|CD|EF|

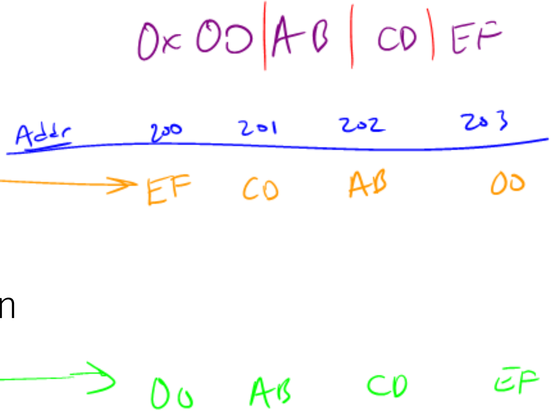
Rules to break "big values" into bytes (memory)

1. Break it into bytes
2. Store them adjacently

0x200 0x201 0x202 0x203

3. Address of the overall value = smallest address of its bytes
4. Order the bytes
    - If parts are ordered (i.e., array), first goes in smallest address
    - Else, hardware implementation gets to pick (!!)
        - Little-endian
        - Big-endian

Ordering Values



Little-endian

- Store the low order part/byte first
- Most hardware today is little-endian

Big-endian

- Store the high order part/byte first

Example

Store [0x1234, 0x5678] at address 0xF00

	Address	Little endian	Big-endian
0x1234	0xF00	34	12
	0xF01	12	34
0x5678	0xF02	78	56
	0xF03	56	78

Endianness

Why do we study endianness?

- It is **everywhere**
- It is a source of weird bugs
- Ex: It's likely your computer uses:
  - Little-endian from CPU to memory
  - Big-endian from CPU to network
  - File formats are roughly half and half

Time to take over the world!