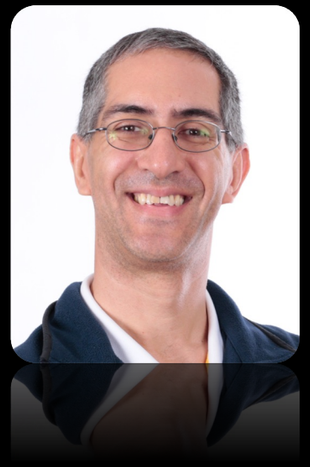


UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture

(a.k.a. Machine Structures)

RISC-V Data Transfer

Storing Data in Memory

RV32 So Far…

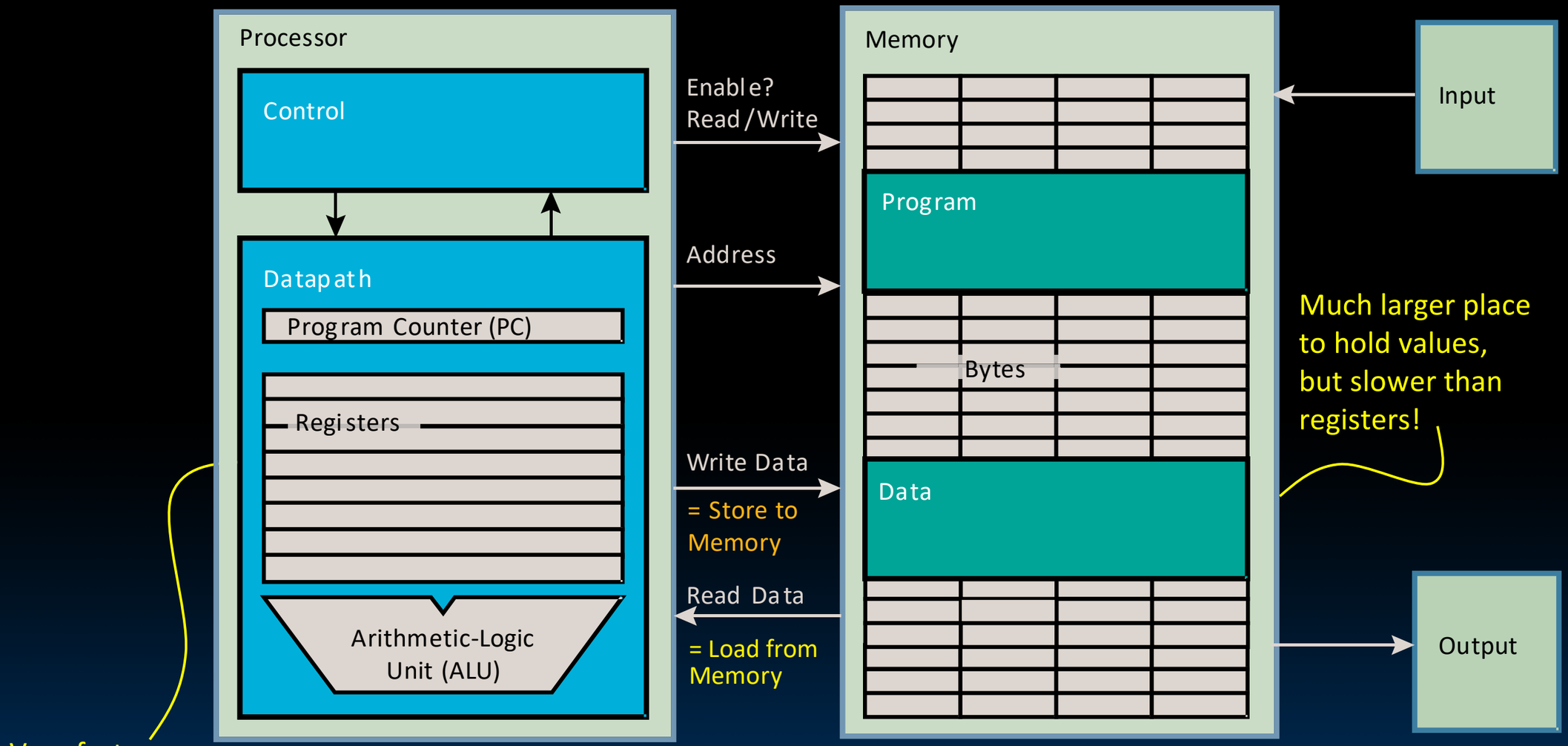
Addition/subtraction

```
add rd, rs1, rs2
    R[rd] = R[rs1] + R[rs2]
sub rd, rs1, rs2
    R[rd] = R[rs1] - R[rs2]
```

Add immediate

```
addi rd, rs1, imm
    R[rd] = R[rs1] + imm
```

Data Transfer: Load from and Store to memory



Memory Addresses are in Bytes

Data typically smaller than 32 bits, but rarely smaller than 8 bits (e.g., char type)–works fine if everything is a multiple of 8 bits

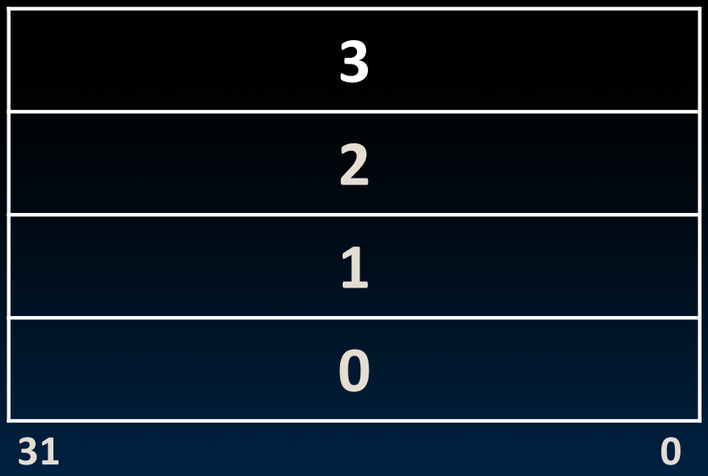
8 bit chunk is called a *byte* (1 word = 4 bytes)

Memory addresses are

really in *bytes*, not words

Word addresses are
4 bytes apart

- Word address is same as address of rightmost byte – least-significant byte (i.e. Little-endian convention)



Memory Addresses are in Bytes

Data typically smaller than 32 bits, but rarely smaller than 8 bits (e.g., char type)–works fine if everything is a multiple of 8 bits

8 bit chunk is called a *byte* (1 word = 4 bytes)

Memory addresses are

really in *bytes*, not words

Word addresses are
4 bytes apart

- Word address is same as address of rightmost byte – least-significant byte (i.e. Little-endian convention)

Least-significant byte in a word ↓

15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

31 24 23 16 15 8 7 0

Least-significant byte gets the smallest address

Big Endian vs. Little Endian

en.wikipedia.org/wiki/endianness

The adjective endian has its origin in the writings of 18th century writer Jonathan Swift. In the 1726 novel Gulliver's Travels, he portrays the conflict between sects of Lilliputians divided into those breaking the shell of a boiled egg from the big end or from the little end. He called them the "Big-Endians" and the "Little-Endians".

- The order in which BYTES are stored in memory
- Bits always stored as usual within a byte (E.g., `0xC2=0b 1100 0010`)

Consider the number 1025 as we typically write it:

BYTE3	BYTE2	BYTE1	BYTE0
00000000	00000000	00000100	00000001

Big Endian

ADDR3	ADDR2	ADDR1	ADDR0
BYTE0	BYTE1	BYTE2	BYTE3
00000001	00000100	00000000	00000000

Examples

Names in China or Hungary (e.g., Garcia Dan)

Java Packages: (e.g., org.mypackage.HelloWorld)

Dates in ISO 8601 YYYY-MM-DD (e.g., 2020-09-07)

Eating Pizza crust first

Little Endian

ADDR3	ADDR2	ADDR1	ADDR0
BYTE3	BYTE2	BYTE1	BYTE0
00000000	00000000	00000100	00000001

Examples

Names in the US (e.g., Dan Garcia)

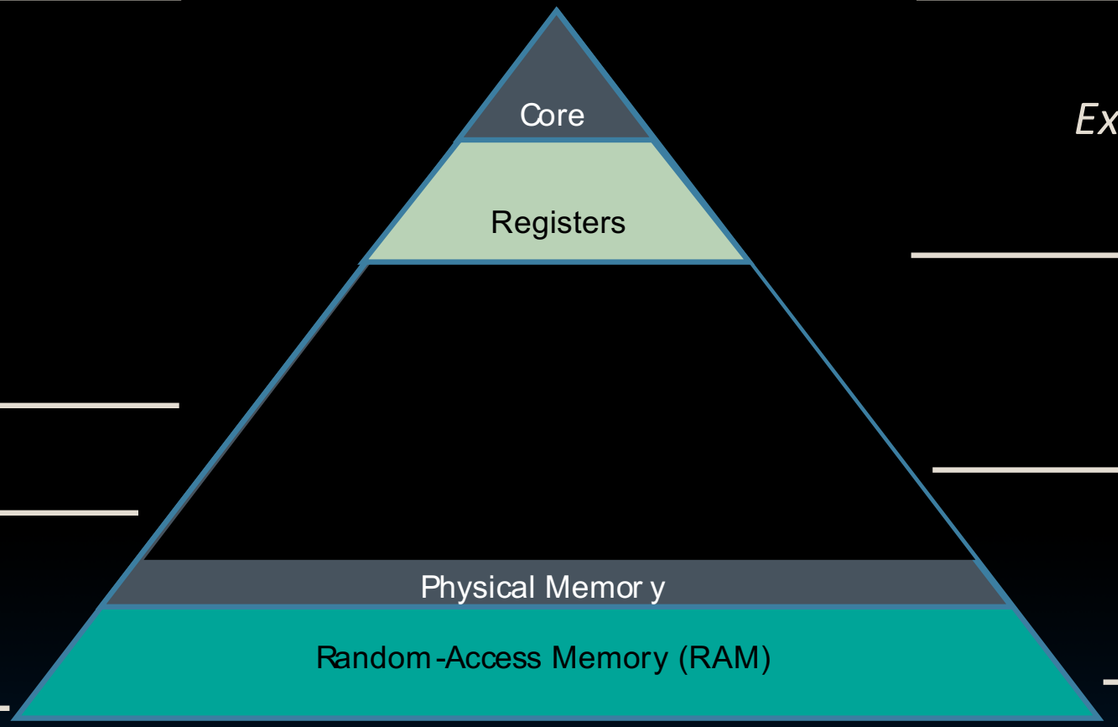
Internet names (e.g., cs.berkeley.edu)

Dates written in Europe DD/MM/YYYY (e.g., 07/09/2020)

Eating Pizza skinny part first

Great Idea #3: Principle of Locality / Memory Hierarchy

Processor chip



Extremely fast
Extremely expensive
Tiny capacity

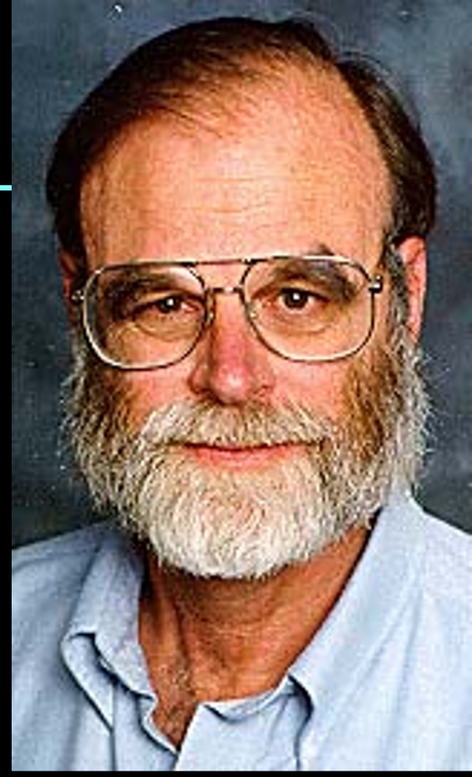
DRAM chip
e.g. DDR3/4/5
HBM/HBM2/3

Fast
Priced reasonably
Medium capacity

Speed of Registers vs. Memory

- Given that
  - Registers: 32 words (128 Bytes)
  - Memory (DRAM): Billions of bytes (2 GB to 96 GB on laptop)
- and physics dictates…
  - Smaller is faster
- How much faster are registers than DRAM??
  - About 50-500 times faster! (in terms of latency of one access - tens of ns)
    - But subsequent words come every few ns

Jim Gray's Storage Latency Analogy: How Far Away is the Data?



Jim Gray
Turing Award
B.S. Cal
1966
Ph.D. Cal
1969

10	Memory		1.5 hr
0			
1	Registers	My Head	1 min
[ns]			

Berkeley
UNIVERSITY OF CALIFORNIA

Load from Memory to Register

C code

```
int  A[100];
g = h + A[3];
```

Using Load Word (`lw`) in RISC-V:

```
lw  x10,12(x15) # Reg x10 gets A[3]
add x11,x12,x10 # g = h + A[3]
```

Note: **x15** – base register (pointer to A[0])

12 – offset in bytes

Offset must be a constant known at assembly time

Store from Register to Memory

C code

```
int  A[100];
A[10] = h + A[3];
```

Using Store Word (**sw**) in RISC-V:

```
lw  x10,12(x15)   # Temp reg x10 gets A[3]
add x10,x12,x10   # Temp reg x10 gets h + A[3]
sw  x10,40(x15)   # A[10] = h + A[3]
```

Note: **x15** – base register (pointer)

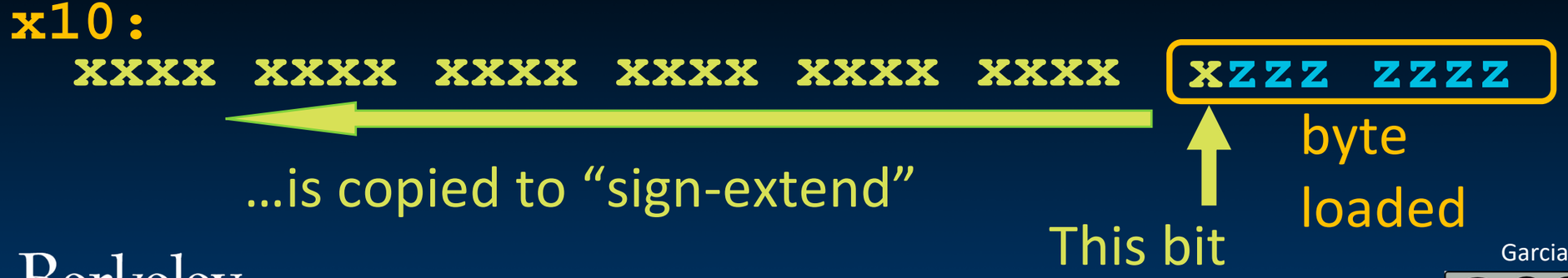
12,40 – offsets in bytes

x15+12 and **x15+40** must be multiples of 4

Loading and Storing Bytes

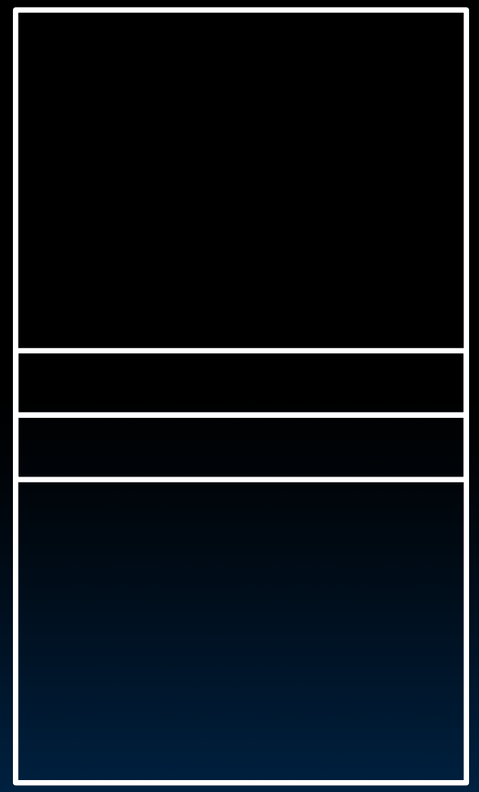
- In addition to word data transfers (lw, sw), RISC-V has byte data transfers:
  - load byte: **`lb`**
  - store byte: **`sb`**
- Same format as **`lw, sw`**
- E.g., **`lb x10,3(x11)`**
  - contents of memory location with address = sum of "3" + contents of register **`x11`** is copied to the low byte position of register **`x10`**.

RISC-V also has "unsigned byte" loads (lbu) which zero extends to fill register. Why no unsigned store byte 'sbu'?



What ends up in x12 ?

```
addi x11,x0,0x39C
sw x11,0(x5)
lb x12,0(x5)
```



Example: Translate *x = *y

We want to translate *x = *y into RISC-V

x, y ptrs stored in: x3 x5

```
1: add x3,    x5, zero
2: add x5,    x3, zero
3: lw  x3, 0(x5)
4: lw  x5, 0(x3)
5: lw  x8, 0(x5)
6: sw  x8, 0(x3)
7: lw  x5, 0(x8)
8: sw  x3, 0(x8)
```

```
1
2
3
4
5→6
6→5
7→8
```

And in Conclusion…

- Memory is byte-addressable, but **`lw`** and **`sw`** access one word at a time.
- A pointer (used by **`lw`** and **`sw`**) is just a memory address, we can add to it or subtract from it (using offset).
- Big- vs Little Endian
  - Tip: draw lowest byte on the right
- New Instructions:

`lw, sw, lb, sb, lbu`