



UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas
in
Computer Architecture
(a.k.a. Machine Structures)



UC Berkeley
Lecturer
Justin Yokota

RISC-V Data Transfer

IS AI-GENERATED ART FAIR?

This year, the Colorado State Fair's annual art competition gave out prizes in all the usual categories: painting, quilting, sculpture. But one entrant, Jason M. Allen of Pueblo West, Colo., didn't make his entry with a brush or a lump of clay. He created it with Midjourney, an artificial intelligence program that turns lines of text into hyper-realistic graphics. Mr. Allen's work, "Théâtre D'opéra Spatial," took home the blue ribbon in the fair's contest for emerging digital artists — making it one of the first A.I.-generated pieces to win such a prize, and setting off a fierce backlash from artists who accused him of, essentially, cheating. Reached by phone on Wednesday, Mr. Allen defended his work. He said that he had made clear that his work — which was submitted under the name "Jason M. Allen via Midjourney" — was created using A.I., and that he hadn't deceived anyone about its origins. "I'm not going to apologize for it," he said. "I won, and I didn't break any rules."



Art, AI Collide in Landmark Legal Dispute

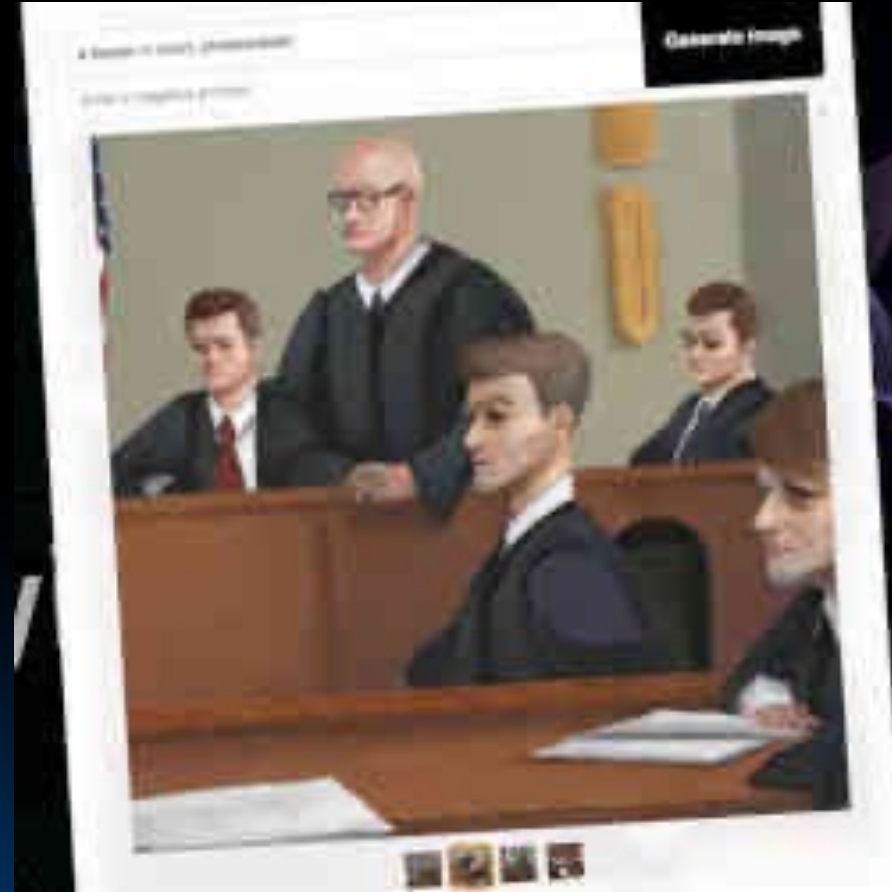
Human artists and artificial intelligence (AI) companies are disputing generative AI-intellectual property in a landmark legal case.

Visual media company Getty Images filed a copyright claim against free image-generating **Stable Diffusion** tool developer Stability AI in the U.K. High Court.

The tool was trained on 2.3 billion images harvested from the Web by a third-party website; Getty alleges Stability AI illegally copied and processed copyrighted images for its commercial benefit.

Sandra Wachter at the U.K.'s Oxford Internet Institute said the case will decide whether companies can use such data for their own purposes.

Said Estelle Derclaye at the U.K.'s University of Nottingham, "Ultimately, [AI companies] are copying the entire work in order to do something else with it — the work may not be recognizable in the output, but it's still required in its entirety."



Storing Data in Memory

RV32 So Far...

- Addition/subtraction

add rd, rs1, rs2

$R[rd] = R[rs1] + R[rs2]$

sub rd, rs1, rs2

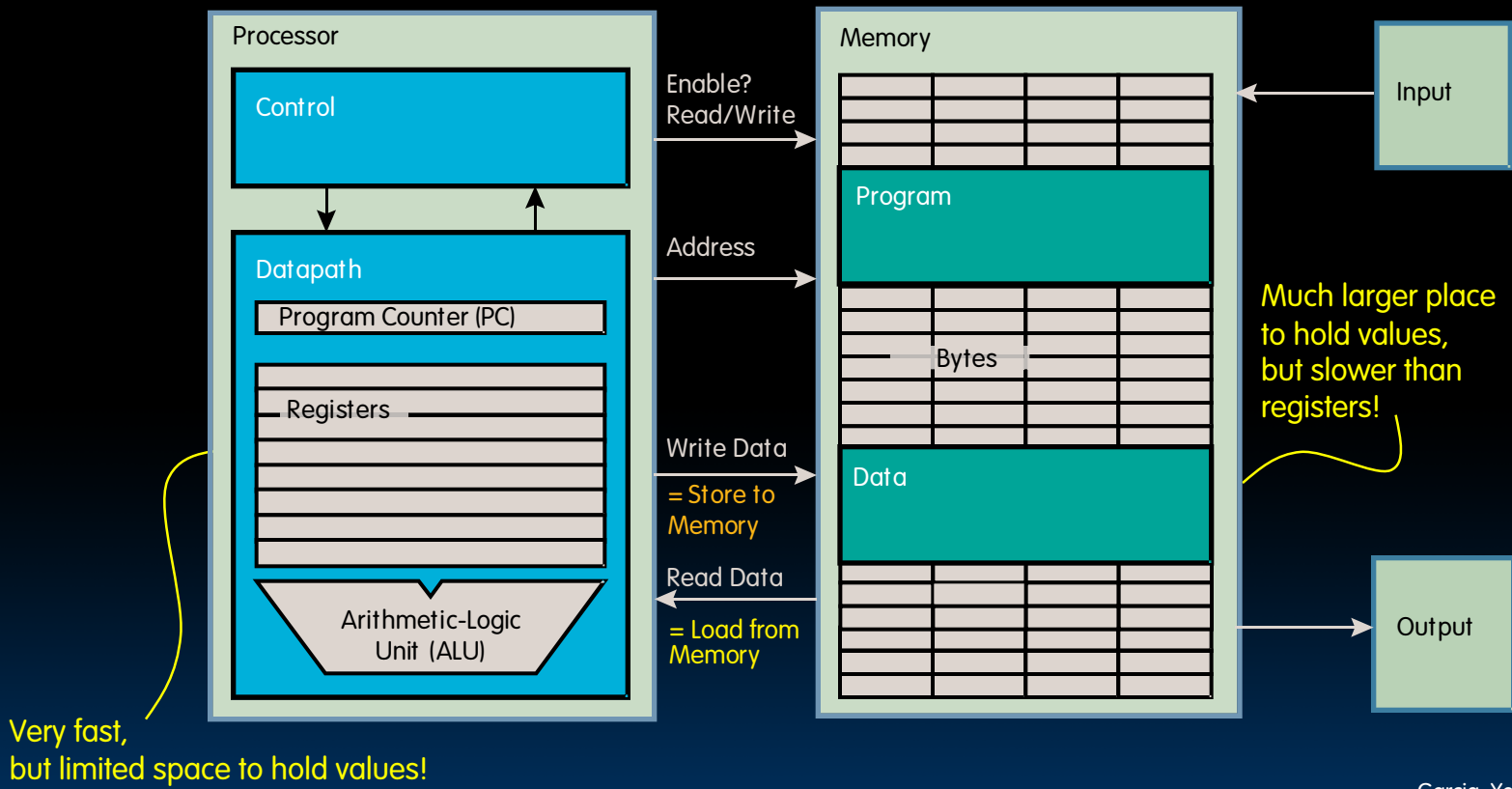
$R[rd] = R[rs1] - R[rs2]$

- Add immediate

addi rd, rs1, imm

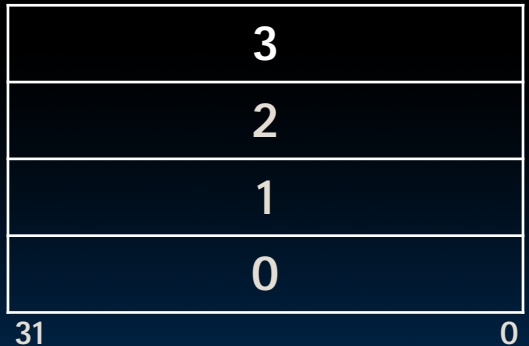
$R[rd] = R[rs1] + imm$

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 ↓

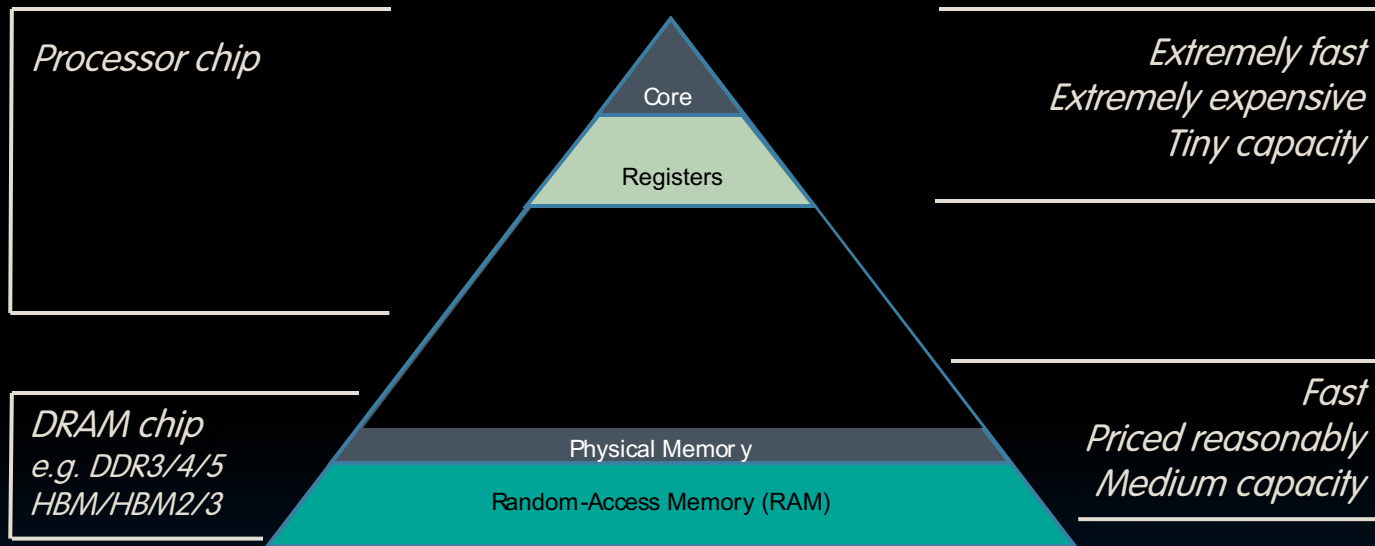
15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

31 24 23 16 15 8 7 0

Least-significant byte gets the smallest address

Data Transfer Instructions

Great Idea #3: Principle of Locality / Memory Hierarchy



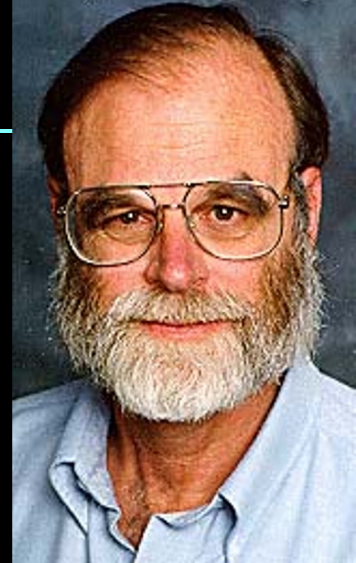


Speed of Registers vs. Memory

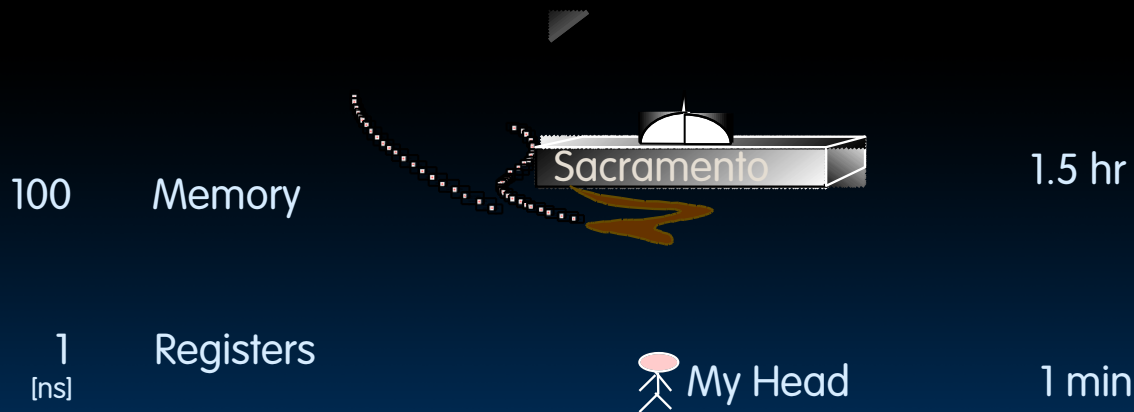
- Given that
 - Registers: 32 words (128 Bytes)
 - Memory (DRAM): Billions of bytes (2 GB to 96 GB on laptop)
- and physics dictates...
 - Smaller is faster
- How much faster are registers than DRAM??
 - About 50-500 times faster!
(in terms of latency of one access - tens of ns)
 - But subsequent words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



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]
```



Data flow

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**.

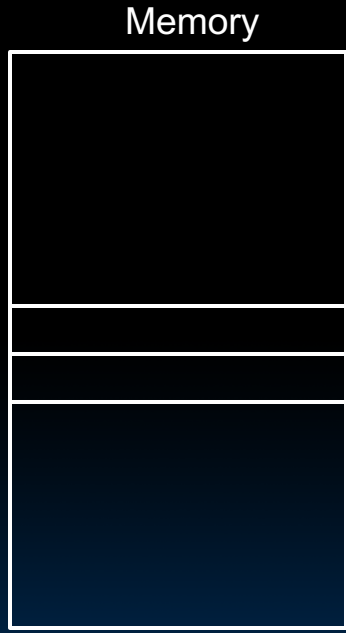
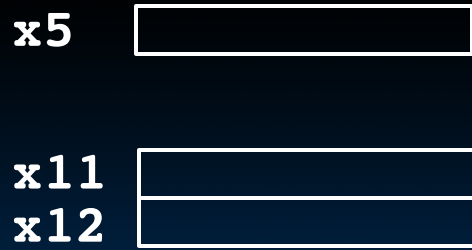
RISC-V also has "unsigned byte" loads (**lbu**) which zero extends to fill register. Why no unsigned store byte '**sbu**'?





Example: What is in x12 ?

```
addi x11,x0,0x93F5
sw x11,0(x5)
lb x12,1(x5)
```



L08a What is in x12?

What is in x12 ?

```
addi x11,x0,0x93F5
```

```
sw x11,0(x5)
```

```
lb x12,1(x5)
```

0x0
0x1
0x12
0xF5
0x93
0x9300
0x93F5
0x99999993
0x99999345
0xFFFF9345
0xFFFFF93
0xFFFFFFFF

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

L08b 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