UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture

(a.k.a. Machine Structures)

UC Berkeley
Lecturer
Justin Yokota

Intro to RISC-V

The RISC-V Instruction Set Architecture

- The RISC-V Instruction Set Architecture
- Elements of Architecture: Registers
- Add/Sub Instructions
- Immediates

Great Idea #1: Abstraction (Levels of Representation/Interpretation)

High Level Language Program (e.g., C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler

Assembly Language Program (e.g., RISC-V)

```
lw   x3, 0(x10)
lw   x4, 4(x10)
sw   x4, 0(x10)
sw   x3, 4(x10)
```

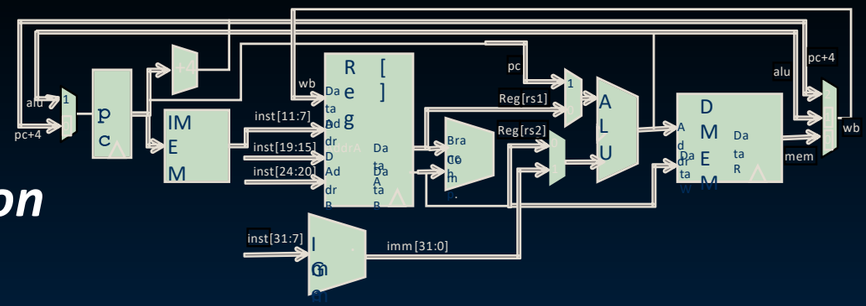
Anything can be represented as a number, i.e., data or instructions

Assembler

Machine Language Program (RISC-V)

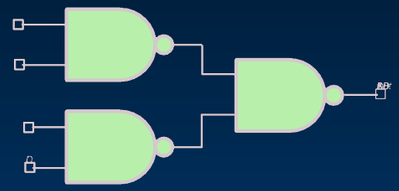
```
1000 1101 1110 0010 0000 0000 0000 0000
1000 1110 0001 0000 0000 0000 0000 0100
1010 1110 0001 0010 0000 0000 0000 0000
1010 1101 1110 0010 0000 0000 0000 0100
```

Hardware Architecture Description (e.g., block diagrams)



Architecture Implementation

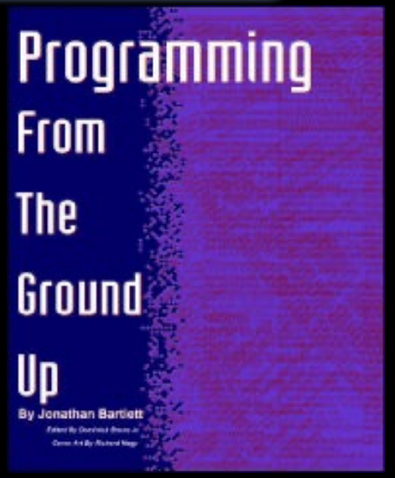
Logic Circuit Description (Circuit Schematic Diagrams)



Assembly Language

- **Basic job of a CPU: execute lots of instructions**
- **Instructions are the primitive operations that the CPU may execute**
  - Like a sentence: *operations* (verbs) applied to *operands* (objects), processed in sequence …
- **Different CPUs implement different sets of instructions. The set of instructions a particular CPU implements is an Instruction Set Architecture (ISA)**
  - Examples: ARM (cell phones), Intel x86 (i9, i7, i5, i3), IBM/Motorola PowerPC (old Macs), MIPS, *RISC-V*, ...

Book: Programming From the Ground Up



"A new book was just released which is based on a new concept – teaching computer science through assembly language (Linux x86 assembly language, to be exact). This book teaches how the machine itself operates, rather than just the language. I've found that the key difference between mediocre and excellent programmers is whether or not they know assembly language. Those that do tend to understand computers themselves at a much deeper level. Although [almost!] unheard of today, this concept isn't really all that new -- there used to not be much choice in years past. Apple computers came with only BASIC and assembly language, and there were books available on assembly language for kids. This is why the old-timers are often viewed as 'wizards': they had to know assembly language programming."

-- slashdot.org comment, 2004-02-05

Instruction Set Architectures

- **Early trend was to add more and more instructions to new CPUs to do elaborate operations**
  - VAX architecture had an instruction to multiply polynomials!
- Reduced Instruction Set Computer (RISC) **philosophy:**
  - Cocke IBM, Patterson, Hennessy, 1980s.
  - Keep the instruction set small and simple, makes it easier to build *fast* hardware
  - Let software do complicated operations by composing simpler ones
  - This went against the conventional wisdom of the time (he who laughs last, laughs best)

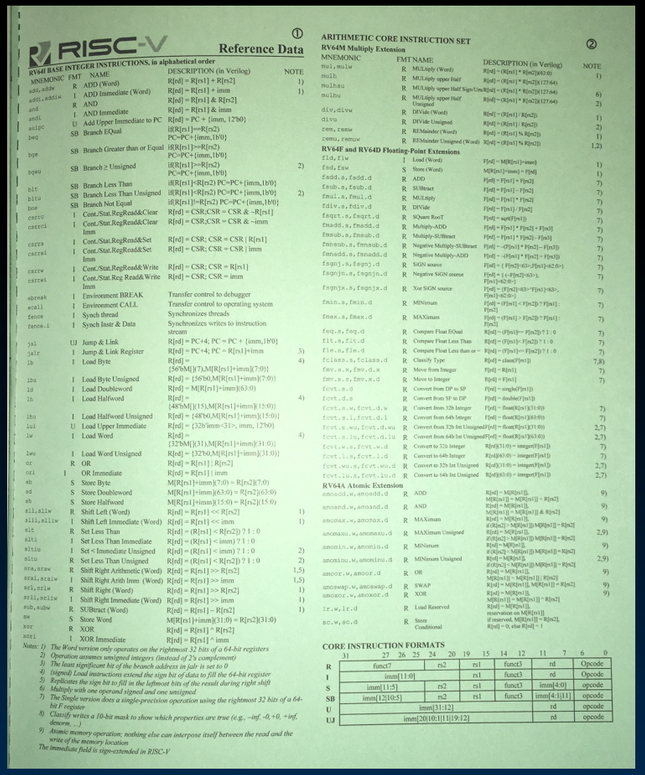
Patterson and Hennessy win Turing!

https://engineering.berkeley.edu/news/2018/06/patterson-wins-turing-award/

RISC-V Architecture

IBM 360 Green Card

- **New open-source, license-free ISA spec**
  - Supported by growing shared software ecosystem
  - Appropriate for all levels of computing system, from microcontrollers to supercomputers
  - 32-bit, 64-bit, and 128-bit variants (class/textbook uses 32-bit)
- **Why RISC-V instead of Intel x86-64?**
  - RISC-V is simple and elegant. We don't need to get bogged down in gritty details
  - RISC-V has exponential adoption, from microcontrollers to CPUs to warehouse-scale computers

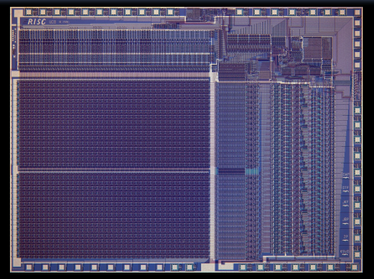


RISC-V Green Card

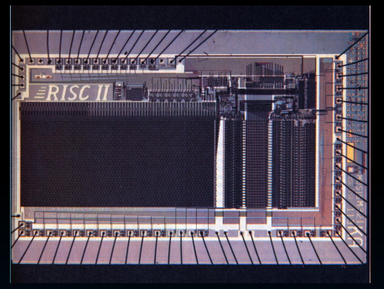
https://cs61c.org/ → Resources

RISC-V Origins

- **Started in Summer 2010 to support open research and teaching at UC Berkeley**
  - Lineage can be traced to RISC-I/II projects (1980s)
- **As the project matured, it migrated to RISC-V foundation (www.risc-v.org)**
- **Many commercial and research projects based on RISC-V**
  - Open-source and proprietary
  - Widely used in education
- **Read more:**
  - https://riscv.org/risc-v-history/
  - https://riscv.org/risc-v-genealogy/



RISC-I



RISC-II

Elements of Architecture: Registers

- The RISC-V Instruction Set Architecture
- Elements of Architecture: Registers
- Add/Sub Instructions
- Immediates

Instruction Set

Preliminary discussion of the logical design of an electronic computing instrument[1]

Arthur W. Burks / Herman H. Goldstine / John von Neumann

"instruction sets"

3.1. It is easy to see by formal-logical methods that there exist codes that are *in abstracto* adequate to control and cause the execution of any sequence of operations which are individually available in the machine and which are, in their entirety, conceivable by the problem planner. The really decisive considerations from the present point of view, in selecting a code, are more of a practical nature: simplicity of the equipment demanded by the code, and the clarity of its application to the actually important problems together with the speed of its handling of those problems. It would take us much too far afield to discuss these questions at all generally or from first principles. We will therefore restrict ourselves to analyzing only the type of code which we now envisage for our machine.

- Instruction set **for a particular architecture (e.g. RISC-V) is represented by the** assembly language
- **Each line of assembly code represents one instruction for the computer**

```
add    x1, x2, x3
```

operation name

registers

Higher-Level Language vs. Assembly Language (1/2)

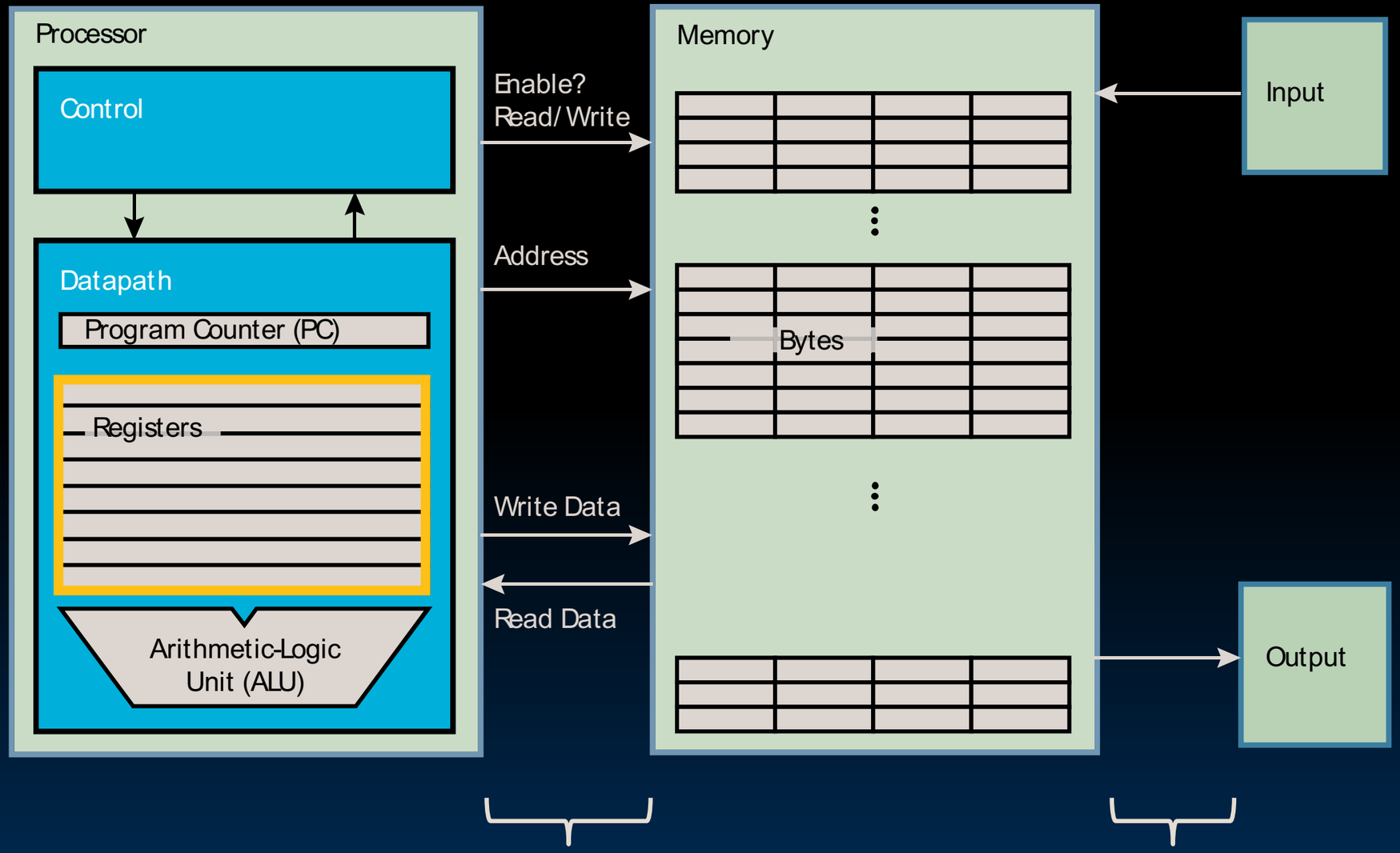
C, Java

- **In C (and most high-level languages), variables declared first and given a type**
  - `int fahr, celsius;`
    `char a, b, c, d, e;`
- **Each variable can ONLY represent a value of the type it was declared as (cannot mix/match `int`/`char` variables)**
- **In high-level languages,** variable types determine operation
  - `int *p = …;`
    `p = p + 2;`
    `int x = 42;`
    `x = 3 * x;`

RISC-V

- **Assembly operands are** registers
  - Registers are limited number of special locations, built directly into the hardware
  - RISC-V: Operations can only be performed on register operands
- **In assembly language, the registers have no type**
  - Register contents are just bits
- Operation determines "type," **i.e., how register contents are treated**
  - E.g., as value, memory address, etc.

Registers are Inside the Processor

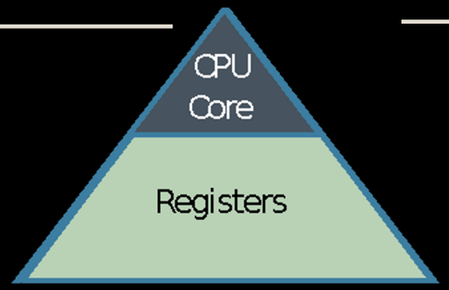


Benefit:
Since registers are directly in hardware, they're very fast (faster than 0.25ns)

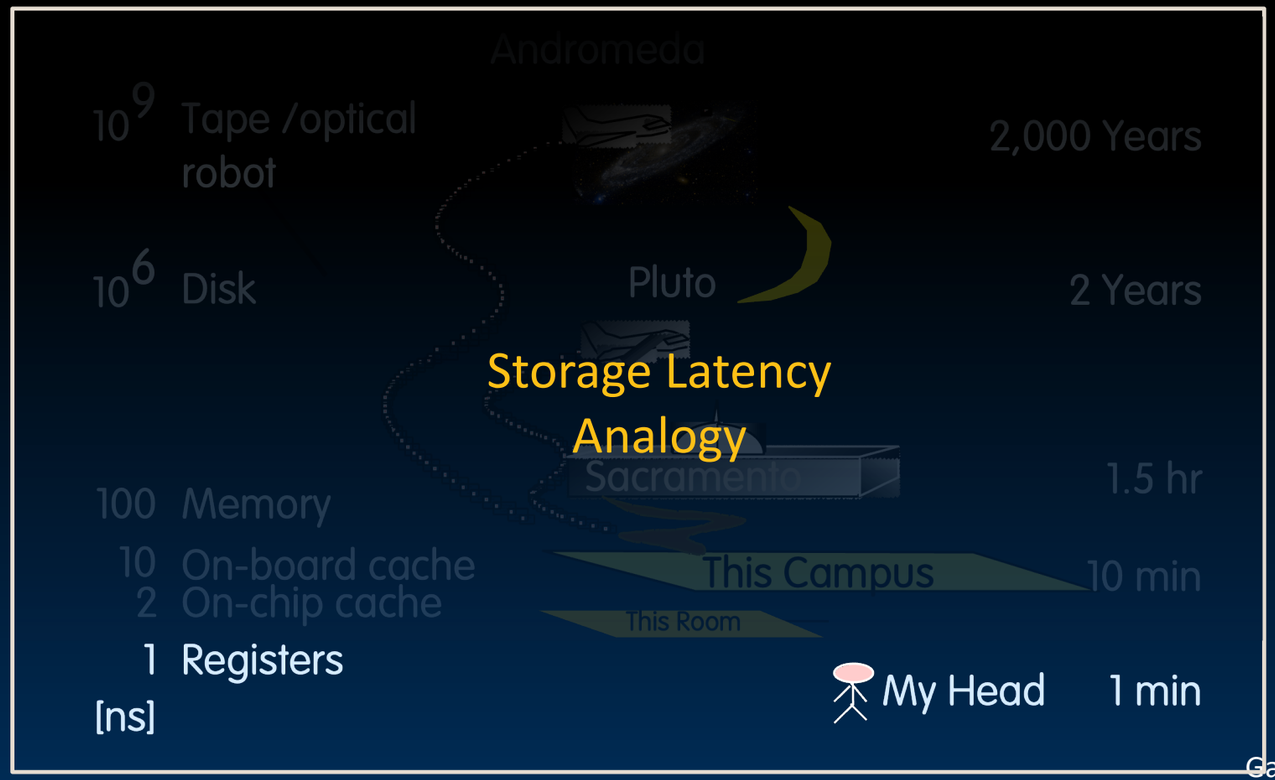
- Speed of light: $3x10^8$m/s = 0.3m/ns = 30cm/ns
  = 10cm/0.3ns!!!
- 0.3ns is the clock period of a 3.33GHz computer

Great Idea #3: Principle of Locality / Memory Hierarchy

Processor chip



Extremely fast
Extremely expensive
Tiny capacity



32 Registers in RISC-V

- **Drawback: Each ISA a predetermined number of registers**
  - Why? Registers are built into hardware
  - Solution: RISC-V code must be very carefully put together to efficiently use registers
    - Nowadays: compiler maps C variables to RISC-V registers across declarations, function calls, etc.
- 32 registers **in RISC-V**
  - Why 32? Smaller is faster, but too small is bad
  - Goldilocks principle ("This porridge is too hot; This porridge is too cold; this porridge is just right")
- **Each RISC-V register is 32 bits wide (in RV32 variant)**
  - Groups of 32 bits called a word in RV32

Register Names and Numbers

- **Registers are numbered from 0 to 31**
  - Referred to by number `x0` – `x31`
- **`x0` is special, always holds value zero**
  - So only 31 registers able to hold variable values
- **Each register can be referred to by number or name**
  - We'll explain names next week
- **Ok, enough already…gimme my RV32!!!**

Add/Sub Instructions

- The RISC-V Instruction Set Architecture
- Elements of Architecture: Registers
- Add/Sub Instructions
- Immediates

Higher-Level Language vs. Assembly Language (2/2)

C, Java

- **Each line can contain multiple operations**

```
a = b * 2 - (arr[2] + *p);
```

- **Common operations are:**
  - =, +, –, *, /
  - Here, * includes both multiply and dereference

RISC-V

- **Each statement is an instruction**
  - An instruction executes exactly one short list of simple commands
- **Each line of assembly code contains at most 1 instruction**

```
add     x1, x2, x3
sub     x1, x1, x4
```

The compiler translates from higher-level language down to assembly…
…so RISC-V instructions are often closely related to common C/Java operations.

RISC-V Arithmetic Instruction Syntax

```
opname  rd, rs1, rs2
```

opname	rd	rs1	rs2
operation, by name	”destination” operand getting result	”source1” 1st operand for operation	”source2” 2nd operand for operation

- **Syntax is rigid:**
  - 1 operator, 3 operands
  - Why? Keep hardware simple via regularity (more in a few weeks)

RISC-V Addition and Subtraction

```
opname  rd, rs1, rs2
```

- opname: operation, by name
- rd: "destination" operand getting result
- rs1: "source1" 1st operand for operation
- rs2: "source2" 2nd operand for operation

- **Addition**

RISC-V `add x1,x2,x3`

C `a = b + c;`

- **Subtraction**

RISC-V `sub x4,x5,x6`

C `d = e - f;`

(remember! order of operands matter in `sub`)

RISC-V Arithmetic, Example 1

- **How could we translate this C statement?**

```
a = b + c + d - e;
```

```
x10   x1   x2   x3   x4
```

  - Assume the above mapping for C variables ↔ RISC-V registers.

- **Solution: Break into multiple instructions!**

```
add x10, x1, x2      # a_temp = b + c
add x10, x10, x3     # a_temp = a_temp + d
sub x10, x10, x4     # a = a_temp - e
```

In-line comments prefixed by #, work like C99's //.
Unlike C, no multi-line comment (/* */) support.

A single line of C may break up into several lines of RISC-V.

RISC-V Arithmetic, Example 2

- **How do we do this?**

```
f = (g + h) - (i + j);
```

```
x19   x20   x21    x22   x23   x5   x6
```

  - Assume the above mapping for C variables ↔ RISC-V registers.

- **Solution: Use intermediate temporary registers!**

```
add x5, x20, x21          # a_temp = g + h
add x6, x22, x23          # b_temp = i + j
sub x19, x5, x6           # f = (g + h)- (i + j)
```

- **Note: A good compiler may actually do the following. Why?**

```
f = g + h - i - j;
```

Can use zero temp registers!

Aside: Apollo Guidance Computer

- **Margaret Hamilton**
  Director of Software Engineering
  Division of MIT Instrumentation Laboratory
  - Built Apollo Guidance Computer
    Command Module (navigation, lunar landing)
    - 72KB of memory on board *Eagle* (Apollo 11, 1969)

Margaret Hamilton and Apollo code
Wikimedia Commons

```
179                     TC      BANKCALL        # TEMPORARY, I HOPE HOPE HOPE
180                     CADR    STOPRATE        # TEMPORARY, I HOPE HOPE HOPE
181                     TC      DOWNFLAG        # PERMIT X-AXIS OVERRIDE
```

```
245                     CAF     CODE500         # ASTRONAUT:    PLEASE CRANK THE
246                     TC      BANKCALL        #               SILLY THING AROUND
247                     CADR    GOPERF1
248                     TCF     GOTOP00H        # TERMINATE
249                     TCF     P63SPOT3        # PROCEED       SEE IF HE'S LYING
250
251     P63SPOT4        TC      BANKCALL        # ENTER         INITIALIZE LANDING RADAR
252                     CADR    SETPOS1
253
254                     TC      POSTJUMP        # OFF TO SEE THE WIZARD ...
255                     CADR    BURNBABY
```

```
39  ## It traces back to 1965 and the Los Angeles riots, and was inspired
40  ## by disc jockey extraordinaire and radio station owner Magnificent Montague.
41  ## Magnificent Montague used the phrase "Burn, baby! BURN!" when spinning the
42  ## hottest new records. Magnificent Montague was the charismatic voice of
43  ## soul music in Chicago, New York, and Los Angeles from the mid-1950s to
44  ## the mid-1960s.
```

Presidential Medal of Freedom (2016)

https://abcnews.go.com/Technology/apollo-11s-source-code-tons-easter-eggs-including/story?id=40515222

Immediates

- The RISC-V Instruction Set Architecture
- Elements of Architecture: Registers
- Add/Sub Instructions
- Immediates

Immediates

- Immediates **are numerical constants.**
  - They appear often in code; hence, they have separate instructions.
- Add Immediate **instruction:**

```
RISC-V   addi x3,x4,10
```

```
    C    f = g + 10;
```

Syntax similar to `add` instruction. But the *last operand* must be *a number*, not a register.

Immediates

- Immediates **are numerical constants.**
  - They appear often in code; hence, they have separate instructions.
- Add Immediate **instruction:**

```
RISC-V   addi x3,x4,10
```

```
    C    f = g + 10;
```

Syntax similar to `add` instruction. But the *last operand* must be *a number*, not a register.

- **There is no Subtract Immediate instruction in RISC-V. Why?**
  - While there are `add` and `sub` instructions, to subtract an immediate, just use `addi`:

```
RISC-V   addi x3,x4,-10
```

```
    C    f = g - 10;
```

RISC Philosophy: **Reduce** the possible types of operations to an absolute minimum.
If an operation can be decomposed into a simpler operation, don't include it in the ISA.

Register Zero

- One particular immediate, the number zero (0), appears very often in code.
- RISC-V hardwires the register zero (x0) to value 0:

```
RISC-V   add  x3,x4,x0
```

```
    C    f = g;
```

```
RISC-V   addi x3,x0,0xff
```

```
    C    f = 0xff;
```

- Defined in hardware, so an instruction `add x0,x3,x4` will not do anything!

Concept Check: True or False?

1. **Types are associated with declaration in C (normally), but are associated with instructions (operators) in RISC-V.**
2. **Since there are only 32 registers, we can't write RISC-V for C expressions that contain > 32 variables.**
3. **If `p` (stored in `x9`) were a pointer to an array of `int`s, then `p++;` would be**

```
addi x9 x9 1
```

	123
A.	FFF
B.	FFT
C.	FTF
D.	FTT
E.	TFF
F.	TFT
G.	TTF
H.	TTT

Concept Check: True or False?

1. **Types are associated with declaration in C (normally), but are associated with instructions (operators) in RISC-V.**
2. **Since there are only 32 registers, we can't write RISC-V for C expressions that contain > 32 variables.**
3. **If `p` (stored in `x9`) were a pointer to an array of `int`s, then `p++;` would be**

```
addi x9 x9 1
```

	123
A.	FFF
B.	FFT
C.	FTF
D.	FTT
E.	TFF
F.	TFT
G.	TTF
H.	TTT

1. True, we saw that on an earlier slide.
2. False. we saw how to break up longer equations to smaller ones already.
3. False. Don't forget that ints are (usually) 4 bytes wide, so instruction would be `addi x9, x9, 4`.