

**CS 110**  
**Computer Architecture**  
**Lecture 8:**  
***Multiplication & Floats***

Instructors:  
**Sören Schwertfeger & Chundong Wang**

**School of Information Science and Technology SIST**

**ShanghaiTech University**

**Slides based on UC Berkley's CS61C**

# RISC-V ISA Specification

- Different modules
- Class covers RV32I Base Integer Instruction Set
  - RV64I (used in textbook) and RV128I also available
  - RV32E: Embedded Systems (only 16 registers)
- Various Extensions, named with letters
- The RISC-V Instruction Set Manual; Volume II: Privileged Architecture
  - For Operating System

# RISC-V Specifications

- <https://riscv.org/technical/specifications/>
  - ISA Specification
  - Debug Specification
  - Trace Specification
  - Compliance Framework
- <https://five-embeddev.com/riscv-isa-manual/latest/intro.html>
  - Manual
- <https://github.com/riscv/riscv-isa-manual/releases/latest>
  - Latest draft document

Editors: Andrew Waterman<sup>1</sup>, Krste Asanović<sup>1,2</sup>  
<sup>1</sup>SiFive Inc.,

<sup>2</sup>CS Division, EECS Department, University of California, Berkeley  
andrew@sifive.com, krste@berkeley.edu

March 5, 2021

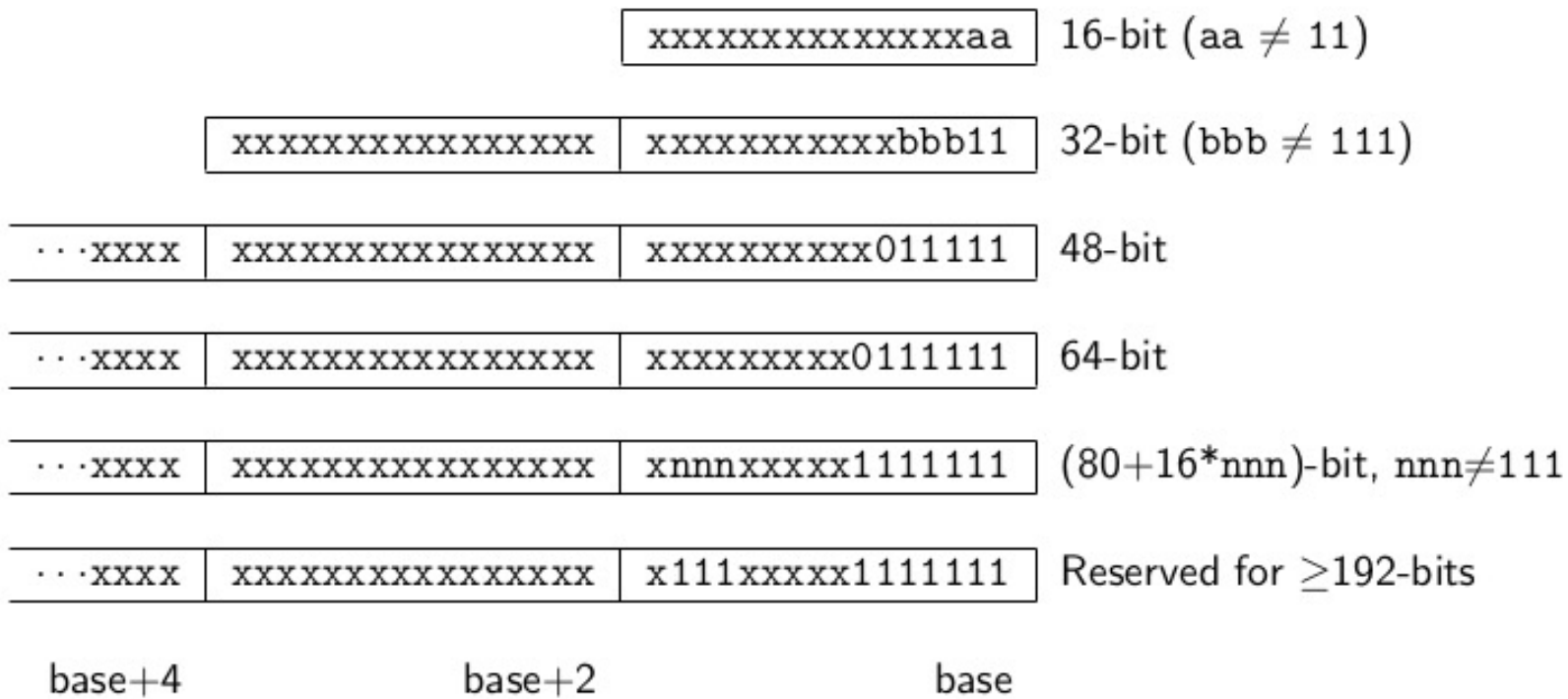
Subset	Name	Implies
Base ISA		
Integer	I	
Reduced Integer	E	
Standard Unprivileged Extensions		
Integer Multiplication and Division	M	
Atomics	A	
Single-Precision Floating-Point	F	Zicsr
Double-Precision Floating-Point	D	F
General	G	IMADZifencei
Quad-Precision Floating-Point	Q	D
Decimal Floating-Point	L	
16-bit Compressed Instructions	C	
Bit Manipulation	B	
Dynamic Languages	J	
Transactional Memory	T	
Packed-SIMD Extensions	P	
Vector Extensions	V	
User-Level Interrupts	N	
Control and Status Register Access	Zicsr	
Instruction-Fetch Fence	Zifencei	
Misaligned Atomics	Zam	A
Total Store Ordering	Ztso	
Standard Supervisor-Level Extensions		
Supervisor-level extension "def"	Sdef	
Standard Hypervisor-Level Extensions		
Hypervisor-level extension "ghi"	Hghi	
Standard Machine-Level Extensions		
Machine-level extension "jkl"	Zxmjkl	
Non-Standard Extensions		
Non-standard extension "mno"	Xmno	

Base	Version	Status
<b>RVWMO</b>	<b>2.0</b>	<b>Ratified</b>
<b>RV32I</b>	<b>2.1</b>	<b>Ratified</b>
<b>RV64I</b>	<b>2.1</b>	<b>Ratified</b>
<i>RV32E</i>	<i>1.9</i>	<i>Draft</i>
<i>RV128I</i>	<i>1.7</i>	<i>Draft</i>
Extension	Version	Status
<b>M</b>	<b>2.0</b>	<b>Ratified</b>
<b>A</b>	<b>2.1</b>	<b>Ratified</b>
<b>F</b>	<b>2.2</b>	<b>Ratified</b>
<b>D</b>	<b>2.2</b>	<b>Ratified</b>
<b>Q</b>	<b>2.2</b>	<b>Ratified</b>
<b>C</b>	<b>2.0</b>	<b>Ratified</b>
<i>Counters</i>	<i>2.0</i>	<i>Draft</i>
<i>L</i>	<i>0.0</i>	<i>Draft</i>
<i>B</i>	<i>0.0</i>	<i>Draft</i>
<i>J</i>	<i>0.0</i>	<i>Draft</i>
<i>T</i>	<i>0.0</i>	<i>Draft</i>
<i>P</i>	<i>0.2</i>	<i>Draft</i>
<i>V</i>	<i>0.7</i>	<i>Draft</i>
<b>Zicsr</b>	<b>2.0</b>	<b>Ratified</b>
<b>Zifencei</b>	<b>2.0</b>	<b>Ratified</b>
<i>Zam</i>	<i>0.1</i>	<i>Draft</i>
<i>Ztso</i>	<i>0.1</i>	<i>Frozen</i>

# Clarifications

- RISC-V ISA Spec: Does NOT define Assembly Syntax
  - Defines Binary Machine Instructions and their behavior
  - Different Assemblers could have different syntax (i.e. allow commas or not)
- Project 1 RISC-V emulator: behave exactly like Venus!
- ALL I-Type instructions (including sltiu):
  - do sign-extension
  - (in Venus): input number is signed, even if hex

# RISC-V instruction sizes



# Compressed Instruction Set “C”

- Use 16 bit instead of 32 bit instructions =>
  - Save space => faster
- E.g.:
  - Use only 8 “popular” registers -> only 3 bits needed
  - immediates have only 6 bits
  - Stack-pointer based loads and stores
  - R & I -type instructions: rs1 and rd the same
  - etc.
- Can mix 32bit and 16bit instructions!

# Format Comparison

## CORE INSTRUCTION FORMATS

	31	27	26	25	24	20	19	15	14	12	11	7	6	0
<b>R</b>	funct7				rs2	rs1	funct3			rd	Opcode			
<b>I</b>	imm[11:0]					rs1	funct3			rd	Opcode			
<b>S</b>	imm[11:5]				rs2	rs1	funct3			imm[4:0]		opcode		
<b>SB</b>	imm[12 10:5]				rs2	rs1	funct3			imm[4:1 11]		opcode		
<b>U</b>	imm[31:12]									rd	opcode			
<b>UJ</b>	imm[20 10:1 11 19:12]									rd	opcode			

Format	Meaning	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CR	Register	funct4				rd/rs1				rs2				op			
CI	Immediate	funct3	imm		rd/rs1				imm				op				
CSS	Stack-relative Store	funct3	imm					rs2				op					
CIW	Wide Immediate	funct3	imm							rd'		op					
CL	Load	funct3	imm		rs1'		imm		rd'		op						
CS	Store	funct3	imm		rs1'		imm		rs2'		op						
CA	Arithmetic	funct6					rd'/rs1'		funct2		rs2'		op				
CB	Branch/Arithmetic	funct3	offset			rd'/rs1'		offset				op					
CJ	Jump	funct3	jump target							op							

*Compressed 16-bit RVC instruction formats.*



# Example

```
1 000000000000000000000000000000001010110011
2 111111111111101011000010110010011
3 00000000101000101000001010110011
4 11111111111101011000010110010011
5 11111110000001011101110011100011
6 000000000101000000000010100110011
7 0000000000000000001000000001100111
```

```
1 000000000000000000000000000000001010110011
2 0001010111111101
3 1001001010101010
4 0001010111111101
5 11111110000001011101111011100011
6 1000010100010110
7 1000000010000010
```



```
1 multiply:
2   add  t0, zero, zero
3   addi a1, a1, -1
4 accumulate:
5   add  t0, t0, a0
6   addi a1, a1, -1
7   bge  a1, zero, accumulate
8   add  a0, zero, t0
9   ret
```

# Project 1

- Project 1.1:
  - In C89, write a program to compress RV32 to RV32C (on Assembler level)
    - Carefully read webpage and documentation
  - Of course: Best results when compiler is aware of compression (use according instructions/ registers)
- Project 1.2:
  - In RISC-V, write a program to de-compress RV32C to RV32

# **MULTIPLICATION AND DIVISION FOR RV32I: EXTENSION M**

# Integer Multiplication (1/3)

- Paper and pencil example (unsigned):

Multiplicand	1000	8
Multiplier	<u>x1001</u>	9
	1000	
	0000	
	0000	
	+1000	
	<u>01001000</u>	72

- $m$  bits  $\times$   $n$  bits =  $m + n$  bit product

# Integer Multiplication (2/3)

- In RISC-V, we multiply registers, so:
  - 32-bit value x 32-bit value = 64-bit value
- Multiplication is *not* part of standard RISC-V because:
  - It requires a more complicated ALU
  - The compiler can use a series of shifts and adds if the multiplier isn't present
- Syntax of Multiplication (signed):
  - **mul rd, rs1, rs2**
  - **mulh rd, rs1, rs2**
  - Multiplies 32-bit values in those registers and returns either the lower or upper 32b result
    - If you do mulh/mul back to back, the architecture can fuse them
  - Also unsigned versions of the above

# Integer Multiplication (3/3)

- Example:
  - in C: **a = b \* c;**
    - **int64\_t a; int32\_t b, c;**
    - These types are defined in C99, in `stdint.h`
- in RISC-V:
  - let **b** be **s2**; let **c** be **s3**; and let **a** be **s0** and **s1**  
(since it may be up to 64 bits)
  - **mulh s1, s2, s3**  
**mul s0, s2, s3**

# Integer Division (1/2)

- Paper and pencil example (unsigned):

– Quotient =  $1001010 / 1000$

– Remainder =  $1001010 \% 1000$

		1001	Quotient
Divisor	1000	<u>1001010</u>	Dividend
		-1000	
		10	
		101	
		1010	
		<u>-1000</u>	
		10	Remainder
			(or Modulo result)

Dividend = Quotient x Divisor + Remainder

# Integer Division (2/2)

- Syntax of Division (signed):
  - **div rd, rs1, rs2**  
**rem rd, rs1, rs2**
  - Divides 32-bit rs1 by 32-bit rs2, returns the quotient (/) for div, remainder (%) for rem
  - Again, can fuse two adjacent instructions
- Example in C: `a = c / d; b = c % d;`
- RISC-V:
  - `a↔s0; b↔s1; c↔s2; d↔s3`
  - **div s0, s2, s3**  
**rem s1, s2, s3**



# Note Optimization...

- A recommended convention
  - `mulh s1 s2 s3`  
`mul s0 s2 s3`
  - `div s0 s2 s3`  
`rem s1 s2 s3`
- Not a *requirement but...*
  - RISC-V says "if you do it this way, *and* the microarchitecture supports it, it can fuse the two operations into one"
  - Same logic behind much of the 16b ISA design: If you follow the convention you can get significant optimizations

# “And in Conclusion...”

- Simplification works for RISC-V: Instructions are same size as data word (one word) so that they can use the same memory.
- Computer actually stores programs as a series of these 32-bit numbers.
- We have covered all RISC-V instructions and registers
  - R-type, I-type, S-type, B-type, U-type and J-type instructions
  - Practice assembling and disassembling
- Introduced Compressed Instructions for Project 1
- RISC-V Multiplication and Division

# Admin

- Midterm I
  - March 29 during lecture hours
    - We start sharp at 8:15!
    - We expect you to sit in your seat at 8:05 – so we can distribute the exams!
    - Be there at 8:00!
- Contents:
  - Everything till (including) Datapath (March 24 lecture)

# Midterm I

- Switch cell phones **off!**  
(not silent mode – off!)
  - Put them in your bags.
- Bags in the front. On the table: nothing but:  
pen, 1 drink, 1 snack, your student ID card and your  
cheat sheet!
- The RISC V green card will be provided
- No other electronic devices are allowed!
  - No ear plugs, music, smartwatch...
- Anybody touching any electronic device will **FAIL** the  
course!
- Anybody found cheating (copy your neighbors answers,  
additional material, ...) will **FAIL** the course!







# COMPUTER ORGANIZATION AND DESIGN

THE HARDWARE/SOFTWARE INTERFACE

 RISC-V EDITION



**MK**  
MORGAN KAUFMANN

DAVID A. PATTERSON  
JOHN L. HENNESSY



# Cheat Sheet

- 1 A4 Cheat Sheet allowed (double sided)
  - Midterm II: 2 pages
  - Final: 3 pages
- Rules:
  - Hand-written – **not printed!**
  - Your **name** in pinyin on the top!
  - Cheat Sheets not complying to this rule will be **confiscated!**

FUNCTIONS OF SEVERAL VARIABLES		$z = f(x, y)$ $w = f(x, y, z)$	DOMAIN: Allowed $(x, y)$ , $(x, y, z)$ RANGES: $z$ max's
<b>LEVEL CURVES</b> 2-D $z = f(x, y) = k = \text{const.}$ CONTOUR MAPS (2-D) $w = f(x, y, z) = k = \text{const.}$ SURFACE LAYERS (3-D)	<b>FUNCTION OF <math>n</math> VARIABLES</b> $\mathbb{R}^n \rightarrow \mathbb{R}$ 1. As a function of $n$ real variables $x_1, x_2, \dots, x_n$ 2. As a function of a single real variable $x$ 3. As a function of a single real var. $x$ and $n-1$ constants	<b><math>\epsilon</math>-<math>\delta</math> DEFINITION OF CONTINUITY</b> LET $f$ BE A FUNCTION OF 2 VARIABLES DEFINED ON A DISK $w$ CENTERED $(a, b)$ , EXCEPT POSSIBLY $(a, b)$ . THEN $\lim_{(x, y) \rightarrow (a, b)} f(x, y) = L$ IF FOR EVERY $\epsilon > 0$ , THERE IS A CORRESPONDING $\delta > 0$ ST. IF $(x, y) \in D_f$ AND $\sqrt{(x-a)^2 + (y-b)^2} < \delta$	
<b>PARTIAL DERIVATIVES</b> $z = f(x, y)$ $f_x(x, y) = \frac{\partial z}{\partial x}$ $f_y(x, y) = \frac{\partial z}{\partial y}$	<b>NOTATIONS</b> SAME METHODS FOR FUNCTIONS OF MORE THAN TWO VARIABLES	IF THE LIMIT AS A FUNCTION APPROACHES A POINT $(a, b)$ ALONG TWO DIFFERENT PATHS IS NOT THE SAME, THE LIMIT DOES NOT EXIST $\emptyset$ $f(x, y)$ IS CONTINUOUS AT $(a, b)$ IF THE LIMIT OF $(x, y)$ AS $(x, y) \rightarrow (a, b)$ EXISTS.	
<b>SECOND PARTIAL DERIVATIVES</b> $f_{xx} = \frac{\partial^2 z}{\partial x^2}$ $f_{xy} = \frac{\partial^2 z}{\partial y \partial x}$ $f_{yx} = \frac{\partial^2 z}{\partial x \partial y}$ $f_{yy} = \frac{\partial^2 z}{\partial y^2}$	<b>CLAIRAUT'S THEOREM</b> IF $f_{xy}$ AND $f_{yx}$ ARE BOTH CONTINUOUS $f_{xy}(a, b) = f_{yx}(a, b)$ <b>PARTIAL DIFF. EQS</b> LAPLACE'S EQUATION $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$ etc... HEAT EQUATION $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = c$ etc...	COMPOSITE FUNCTIONS OF CONTINUOUS FUNCTIONS ARE CONTINUOUS, AS ARE SUMS AND PRODUCTS <b>EQUATIONS OF TANGENT PLANES TO SURFACES</b> $z = f(x, y)$ @ $(x_0, y_0, z_0)$ EVALUATED AT A POINT $z - z_0 = f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0)$ <b>TOTAL DIFFERENTIAL</b> ( $dz = f'(x, y) \cdot \text{single variable}$ ) $dz = f_x(x, y)dx + f_y(x, y)dy = \frac{\partial z}{\partial x} dx + \frac{\partial z}{\partial y} dy$ <b>INCREMENTS</b> $dx, dy, dz$ DIFFERENTIALS, $dx, dy, dz$ OF $R$ SMALL $dx, dy$ $dz = dx, dy = dy$ ( $z$ CHANGE IN HEIGHT OF SURFACE ( $\Delta z$ ) CHANGE IN HEIGHT OF THE TANGENT PLANE ( $dz$ ))	
<b>THE CHAIN RULE</b> SINGLE VARIABLE $y = f(x)$ , $w = g(x)$ , IF $z = f(g(x))$ $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$ IF $z = f(x, y)$ , $x = g(t)$ , $y = h(t)$ $\frac{dz}{dt} = \frac{dz}{dx} \frac{dx}{dt} + \frac{dz}{dy} \frac{dy}{dt}$ IF $w = f(x, y, z)$ , $x = g(t)$ , $y = h(t)$ , $z = k(t)$ $\frac{dw}{dt} = \frac{dw}{dx} \frac{dx}{dt} + \frac{dw}{dy} \frac{dy}{dt} + \frac{dw}{dz} \frac{dz}{dt}$	<b>CHAIN RULE: GENERAL VERSION</b> IF $z = f(x_1, \dots, x_n)$ , $x_i = g_i(t)$ , $i = 1, \dots, n$ $\frac{dz}{dt} = \frac{dz}{dx_1} \frac{dx_1}{dt} + \dots + \frac{dz}{dx_n} \frac{dx_n}{dt}$ FOR EACH $i = 1, 2, \dots, n$	IF $f_x$ AND $f_y$ ARE CONTINUOUS $\Delta z \approx dz$ ( $z$ CHANGE IN HEIGHT OF SURFACE ( $\Delta z$ ) CHANGE IN HEIGHT OF THE TANGENT PLANE ( $dz$ )) <b>THEOREM</b> $\Delta z = f_x(a, b)\Delta x + f_y(a, b)\Delta y - f(a, b)$ $\Delta z = f_x(a, b)\Delta x + f_y(a, b)\Delta y + \epsilon_1 \Delta x + \epsilon_2 \Delta y$ where $\epsilon_1$ and $\epsilon_2$ are functions of $\Delta x$ and $\Delta y$ that approach 0 as $(\Delta x, \Delta y) \rightarrow (0, 0)$ . DEF. $f$ IS DIFFERENTIABLE @ $(a, b)$ FOR $z = f(x, y)$	
<b>IMPLICIT DIFFERENTIATION</b> $\frac{dz}{dx} = -\frac{F_x}{F_z}$ $\frac{dz}{dy} = -\frac{F_y}{F_z}$	<b>DEPENDENCY DIAGRAMS</b> (CASE 1 & 2) YOU CAN FIND DERIVATIVES FOR ALL THE TEMPERATURES		
<b>TANGENT PLANE TO A LEVEL SURFACE</b> $F(x, y, z) = 0$ $\nabla F(x_0, y_0, z_0) \cdot \nabla F(x, y, z) = 0$	<b>THE GRADIENT VECTOR</b> $\nabla f(x, y, z)$ $\nabla f(x, y, z) = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$	<b>DIRECTIONAL DERIVATIVES</b> $\text{Diff } \vec{u} = (a, b)$ $\text{Diff } f(x, y) = f_x(x, y)a + f_y(x, y)b$ SAME FOR $\text{Diff } f(x, y, z) = \nabla f(x, y, z) \cdot \vec{u}$ 3 VARIABLES $\text{Diff } f$ max occurs when $\vec{u}$ is in the same dir. as $\nabla f$ $\text{Diff } f = \nabla f \cdot \vec{u} =  \nabla f   \vec{u}  \cos \theta =  \nabla f   \vec{u}  \cos \theta$	
<b>SPECIAL CASE</b> $f_x(x, y, z) = 0$ , $f_y(x, y, z) = 0$ , $f_z(x, y, z) = 0$ THEN $F_x = -1$ , $F_y = f_x(x, y, z)$ and TANGENT PLANE $z = z_0 + f_x(x - x_0) + f_y(y - y_0)$	<b>MAXIMUM AND MINIMUM VALUES</b> $z = f(x, y)$ $f_x(a, b) = 0$ , $f_y(a, b) = 0$ NECESSARY BUT NOT SUFFICIENT TO GUARANTEE A MAX. OR MIN. IF $f_{xx} f_{yy} > 0$ SADDLE POINT IF $f_{xx} f_{yy} < 0$ SADDLE POINT IF $f_{xx} f_{yy} > 0$ LOCAL MAX. OR MIN. IF $f_{xx} f_{yy} < 0$ SADDLE POINT	<b>THE GRADIENT VECTOR IS ORTHOGONAL TO THE LEVEL CURVES OF A SURFACE</b> <b>IF <math>\nabla f</math> HAS MANY COMPONENTS, AS <math>f</math> HAS INDEPENDENT VARIABLES, <math>\nabla f(x, y, z)</math> POINTS TO THE HIGHER <math>\Delta z</math> FUNCTION IS <math>\perp</math> TO YOUR SURFACE</b> TO FIND THE NORMAL (AND LATER TANGENT) PLANE TO A SURFACE, LET THAT SURFACE BE THE LEVEL SET OF SOME HIGHER DIMENSIONAL FUNCTION. THEN THE GRADIENT OF THE HIGHER $\Delta z$ FUNCTION IS $\perp$ TO YOUR SURFACE IF $\vec{u} = (a, b, c)$ IS THE NORMAL VECTOR TO THE PLANE $ax + by + cz = d$	
<b>FINDING ABSOLUTE MAX. AND MINS.</b> FOR A CLOSED BOUNDARY 1. Find values of $f$ at the critical points of $f$ in $D$ 2. Find the endpoint values of $f$ on the boundary of $D$ 3. The largest value from 1, 2 is the ABS. MAX, the smallest is the ABS. MIN	<b>MAXIMIZING AND MINIMIZING</b> SET OF A FUNCTION OF TWO VARIABLES OF THE FORM $z = f(x, y)$ AND SUCH DOMAIN USUAL CONSTANT		

**WAVELENGTH**  
 $v = \lambda \cdot f$   
SPEED OF LIGHT  $c = 3 \times 10^8$  m/s  
WAVELENGTH  $\lambda$   
FREQUENCY  $f$   
PERIOD  $T = 1/f$

**DIFFRACTION**  
DIFFRACTION IS THE BENDING OF LIGHT AROUND OBSTACLES AND THROUGH APERTURES.  
CONSTRUCTIVE INTERFERENCE: when 2 or more waves travel in  $\phi$  phase the resulting wave is the sum of their displacements  
DESTRUCTIVE INTERFERENCE: when 2 or more waves travel in  $\phi$  phase the resulting wave is the sum of their displacements

**LAW OF REFLECTION**  
The angle of incidence is equal to the angle of reflection. The incident and reflected beams, and the normal, all lie on the same plane.

**REFRACTION**  
REFRACTION IS THE BENDING OF THE PATH OF LIGHT DUE TO A CHANGE IN SPEED AS IT ENTERS A MEDIUM OF DIFFERENT OPTICAL DENSITY.  
DENSE  $\rightarrow$  HIGH REFRACTIVE INDEX  
SLOWER MEDIUM  $\rightarrow$  REFRACTED TOWARD THE NORMAL  
LESS DENSE  $\rightarrow$  REFRACTED AWAY FROM THE NORMAL

**INTERFERENCE**  
DIFFRACTION: doesn't allow light to pass thru TRANSPARENT; allows light to pass TRANSLUCENT; allows light throughout; distorts its path  $\rightarrow$  blur the image. No material can allow 100% of incident light to pass through.

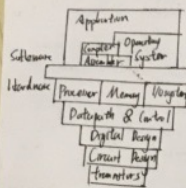
**NEWTON'S RINGS**  
A body's speed remains constant if the net force acting on it is zero (balanced). Acceleration is directly proportional to net force and inversely proportional to mass. Forces exist in action/reaction pairs. member that when gravity is acting against the sum of forces, it will be factored as a negative.

**GRAVITATIONAL POTENTIAL**  
Gravitational potential:  $U = mgh$   
Kinetic energy:  $K = \frac{1}{2}mv^2$   
Work:  $W = F \cdot d$   
Power:  $P = \frac{W}{t}$

**OPTICS**  
EMISSION: spectral clay, clear liquid, white light, color, stars have more complex elements  
ABSORPTION: spectral clay, clear liquid, white light, color, stars have more complex elements  
APPARENT MAGNITUDE: magnitude that is observed by an observer on earth  
ABSOLUTE MAGNITUDE: magnitude that it is 10 parsecs from earth  
DIFFERENCE OF 1 magnitude = 2.512x brighter  
LUMINOSITY: measure of the total amount of energy radiated by a star/second  
THE BRIGHTER THE STAR, THE GREATER ITS LUMINOSITY.  
THE BIGGER THE STAR (SURFACE, RATE OF THE NUCLEUS)  
CEPHEIDS ARE STARS THAT VARY IN BRIGHTNESS OVER PERIODS OF DAYS.  
AMPLITUDE RANGE: 0.5 to 1000 days  
THE GREATER THE PERIOD, THE GREATER THE LUMINOSITY OF THE STAR.  
THE SUN IS AN ORANGE-YELLOW STAR.  
FUSION:  $4H^1 \rightarrow He^4 + 2e^+ + 2\nu_e + \text{energy}$   
Energy is released when two light atoms fuse to form a heavier nucleus. They release energy when the strong nuclear force binds the nucleus together.  
The main sequence like the sun.  
The star heats, shrinks then expands  $\rightarrow$  yellow dwarf  $\rightarrow$  red giant  $\rightarrow$  red giant.

**UNIT CONVERSION**  
m/s  $\times 3.6 \rightarrow$  km/h  
km/h  $\div 3.6 \rightarrow$  m/s

Old School Machine Structure

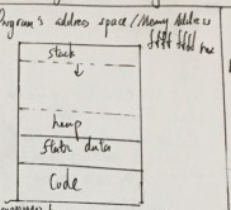


When C program starts  
 - C executable and is loaded into memory by OS (copying bytes)  
 - OS puts it on stack, then calls into main function  
 - Run-time info: variables memory and the library  
 - That will give processor some manual

- 优先级: 从上到下依次递增  
 1. 门地址  
 2. 左移 右移  
 3. 强制类型转换  
 4. 自增 自减  
 5. 取地址  
 6. 乘除  
 7. 加减  
 8. 移位  
 9. 关系运算符

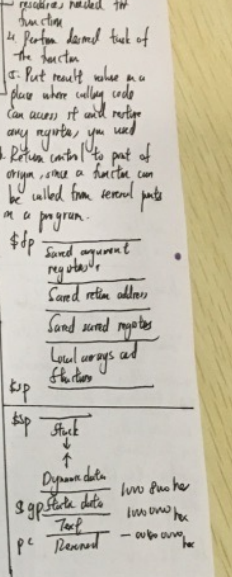
Valid Pointer Arithmetic  
 - Add an integer to a pointer  
 - Subtract 2 pointers for the same array  
 - Compare two pointers (C, C++, ...)  
 - Cannot pointer to null  
 - Add two pointers / multiply

int main (int argc, char \* argv[])  
 - argc contains the number of things in the command line  
 - argv is a pointer to an array containing the arguments as strings



Stack: local variables inside functions, grow downwards  
 Heap: space requested for dynamic data via malloc(), grows dynamically  
 Static data: variable declared outside functions, loaded when program starts, can be modified  
 Code: loaded when program starts, doesn't change

Five Fundamental Steps in Calling a Function  
 1. Put parameters in a place where function can access them  
 2. Transfer control to function  
 3. Access local storage  
 4. Perform desired task of the function  
 5. Put result value in a place where calling code can access it and restore any registers you used



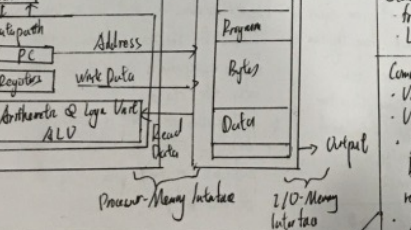
New School Machine Structure

- Parallel Requests  
 Parallel Threads  
 Parallel Data  
 Hardware descriptors

Guest Ideas in Computer Architecture  
 1. Abstraction  
 2. Moore's Law  
 3. Principle of Locality  
 4. Parallelism  
 5. Performance Measurement and Improvement  
 6. Dependability via Redundancy

Two's Complement Representation  
 - treats 0 as positive  
 - 32-bit word represents  $2^{31}$  into form  $-2^{31}$  to  $2^{31}-1$

Components of a Computer



Processor-Memory Interface I/O-Memory Interface

- fun. C → C++ → Fortran → compiler  
 - C++ replaces comments with a double quote  
 - C++ comments begins with #  
 (standard type)  
 char signed char  
 int 4  
 long 8  
 float 4 byte  
 double 8 byte / long double 10 byte

Stack  
 - free when function returns  
 - last in first out

Common Memory Problems  
 - Very uninitialized values  
 - Very memory that you don't own  
 - Important use of malloc / free by memory with the pointer handle  
 - Memory leaks

Levels of Representation / Interpretation  
 High level language  
 Assembly language  
 Machine language  
 Machine / Interpretation  
 Hardware Architecture Description  
 Architecture / Implementation  
 Logic Level Description

R format: used for instructions with immediate, low and high and branches (key and base)

Opcode rs rt rd shamt funct  
 1 format: 5-bit field only represent number up to 31  
 If instruction has immediate, then use at most 2 registers  
 used for imm, src, reg, (one) branches and with immediate  
 Delay with larger immediate  
 Label Upper lui  
 Imm \$to \$fs OUI, AUI, BUI, LUI  
 Shift \$to \$fs OUI, AUI, BUI, LUI  
 \$to \$fs OUI, AUI, BUI, LUI  
 \$to \$fs OUI, AUI, BUI, LUI



# SIFT REFERENCE GUIDE (V.1.1) – CREATING TIMELINES WITH THE SIFT WORKSTATION



## 1. VISIT: <http://computer-forensics11.sans.org/community/downloads>

## 2. BOOT SIFT VM

Load: SIFT Workstation VM Appliance  
 Download SIFT Workstation Installation

## 3. ELEVATE PRIVS

\$ sudo su  
 Login: `sansforensics`  
 Password: `forensics`

## 4. CONNECT IMAGE TO SIFT

Plug hard drive to physical host and attach to SIFT VM

## 5. HARD DRIVE MOUNTING (if you are using log2timeline-sift and Single DD you can skip to 7-A)

### SINGLE OR SPLIT IMAGE (2 options):

`# mount -t ntfs -o ro,loop,show_sys_files,streams_interface=windows image.E01 /mnt/ewf/`  
`# mount -t ntfs -o ro,loop,show_sys_files,streams_interface=windows image.E01 /mnt/ewf/`

`# mount -t ntfs -o ro,loop,show_sys_files,streams_interface=windows image.E01 /mnt/windows_mount/`

### MOUNT TO MOUNT POINT

### SINGLE IMAGE

`# mount -t ntfs -o ro,loop,show_sys_files,streams_interface=windows,offset=#### image.dd /mnt/windows_mount/`

### SPLIT IMAGE (2 step process)

`# affuse image.001 /mnt/aff`  
`# mount -t ntfs-3g -o loop,ro,show_sys_files,streams_interface=windows /mnt/aff/<image> /mnt/windows`

6. log2timeline default timezone is set to examiner local host. To change use `-z [TIMEZONE]` option. To list all available timezones: `# log2timeline -z list`

- log2timeline PARSING PLUGINS**
- apache2\_error - Apache2 error log file
  - chrome - Chrome history file
  - encase\_dirlisting - CSV file that is exported from encase
  - evt - Windows 2k/XP/2k3 Event Log
  - evtx - Windows Event Log File (EVTX)
  - exif - Metadata information from files using ExifTool
  - ff\_bookmark - Firefox bookmark file
  - firefox2 - Firefox 2 browser history
  - firefox3 - Firefox 3 history file
  - ftk\_dirlisting - CSV file that is exported from FTK Imager (dirlisting)
  - generic\_linux - Generic Linux logs that start with MMM DD HH:MM:SS
  - iehistory - index.dat file containing IE history
  - iis - IIS W3C log file
  - isatxt - ISA text export log file
  - jp\_ntfs\_change - CSV output file from JP (NTFS Change log)
  - mactime - Body file in the mactime format
  - mcafee - Log file
  - mft - NTFS MFT file
  - mssql\_errlog - ERRORLOG file produced by MS SQL server
  - ntuser - NTUSER.DAT registry file
  - opera - Opera's global history file
  - oxml - OpenXML document pcap
  - pcap - PCAP file
  - pdf - Available PDF document metadata
  - prefetch - Prefetch directory
  - recycler - Recycle bin directory
  - restore 0.9 - Restore point directory
  - safari - Safari History.plist file
  - sam - SAM registry file
  - security - SECURITY registry file
  - setupapi - SetupAPI log file in Windows XP
  - skype\_sql - Skype database
  - software - SOFTWARE registry file
  - sol - .sol (LSO) or a Flash cookie file
  - squid - Squid access log (http\_emulate off)
  - syslog - Linux Syslog log file
  - system - SYSTEM registry file
  - tlf - Body file in the TLF format
  - volatility - Volatility output files (psscan2, socksca2, ...)
  - win\_link - Windows shortcut file (or a link file)
  - wmipro - WMI log file
  - xpfirewall - XP Firewall log

BY DAVID NIDES (12/16/2011)  
 TWITTER: @DAVNADS  
 BLOG: DAVNADS.BLOGSPOT.COM  
 EMAIL: DNIDES@KPMG.COM  
 CREDITS TO: ED GOINGS, ROB LEWIS, KRISTINN GUDJONSSON, KPMG & PWC  
 QUESTIONS/FEEDBACK-CONTACT US!

**KEY**  
 Red text – image/source  
 Blue text – mount point  
 Purple text – output file  
 Green text – log2timeline plugins  
 Brown text – TimeZone

## 7-A: AUTOMATED SUPER TIMELINE CREATION

`log2timeline-sift -o -z [TIMEZONE] -p [PARTITION #] -i [IMAGE FILE]`

### DISK IMAGE (prompt for partition, mount, and run):

XP `# log2timeline-sift -z EST5EDT -i image`

WIN7 `# log2timeline-sift -win7 -z EST5EDT -i image`

### FOR PARTITION (mount and run using all applicable plugins)

XP `# log2timeline-sift -z EST5EDT -p 0 -i partition`

WIN7 `# log2timeline-sift -win7 -z EST5EDT -i partition`

### OTHER USAGE EXAMPLES:

Display list of available plugins: `# log2timeline -f list`  
 Run log2timeline using only specific plugins: `# log2timeline-sift -p prefetch -z EST5EDT -i image.dd`  
 Help (man pages): `# log2timeline -h`

## 8. CSV FILE OUTPUT (/cases/timeline-output-folder)

`-date`: Time of the event, in the format of MM/DD/YYYY  
`-time`: Time of day, expressed in a 24h format, HH:MM:SS  
`-timezone`: The timezone that was used to call the tool with.  
`-source`: Source short name (i.e. registry entries are REG)  
`-sourcetype`: Desc of the source ("Internet Explorer" instead of WEBHIST)  
`-type`: Timestamp type (i.e. "Last Accessed", "Last Written")  
`-user`: Username associated with the entry, if one is available.  
`-host`: Hostname associated with the entry, if one is available.  
`-short`: Contains less text than the full description field.  
`-desc`: where majority info is stored, the actual parsed desc of the entry.  
`-version`: Version number of the timestamp object.  
`-filename`: Filename with the full path that contained the entry  
`-inode`: inode number of the file being parsed.  
`-notes`: Some input modules insert additional information in the form of a note, which comes here. Or it can be used during the review.  
`-format`: Input module name used to parse the file.  
`-extra`: Additional information parsed is joined together and put here.

## 7-B: MANUAL "MICRO" TIMELINE CREATION

`log2timeline-sift -o [OPTIONS] [-f FORMAT] [-z TIMEZONE] [-o OUTPUT MODULE] [-w LOG_FILE/LOG_DIR [-] [FORMAT FILE OPTIONS]]`

### EXTRACT METADATA (using log2timeline or fls)

Extract metadata w/ log2timeline from mounted file system:  
`# log2timeline -f mft -o mactime -r -z EST5EDT -w mft.body /mnt/volume/`  
 OR Extract metadata using Sleuthkit:  
`# fls -m "" -o offset -d dd > fls.body`  
 Convert body file format to CSV format w/ mactime:  
`# mactime -b fls.body -d > log2timeline.csv`

### ARTIFACTS (run I2I on mounted file system with plugins recursively)

Extract artifacts w/ log2timeline and run mactime on mounted file system:  
`# log2timeline -f firefox3,chrome -o mactime -r -z EST5EDT -w web.body /mnt/volume/`  
 Convert body file format to CSV format w/ mactime:  
`# mactime -b log2timeline.body -d > log2timeline.csv`

## 9. FILTER TIMELINE

Filter timeline with date range to include only:  
`I2t_process -b timeline.csv MM-DD-YYYY..MM-DD-YYYY > filtered.csv`  
 Filter timeline with keyword list (one term per line in keywords.txt):  
`I2t_process -b timeline.csv -k keywords.txt > filtered.csv`  
 What sources are in your timeline?  
`awk -F, '{print $6;}' timeline.csv | grep -v sourcetype | sort | uniq`  
 Find all LNK files that reference E Drive  
`grep "Shortcut LNK" timeline.csv | grep "E:"`  
 Find MountPoints2 entries that reference E Drive  
`grep "MountPoints2 key" timeline.csv | grep "E drive"`  
`grep "USB timeline.csv" | grep "SetupAPILog"`

File System	M	A	C	B
Ext2/3	Modified	Accessed	Changed	N/A
FAT	Written	Accessed	N/A	Created
NTFS	File Modified	Accessed	MFT Modified	Created
UFS	Modified	Accessed	Changed	N/A

THE PURPOSE OF THIS REFERENCE GUIDE IS TO WALK THROUGH THE PROCESS OF BOOTING THE SIFT WORKSTATION, CREATING A TIMELINE ("SUPER" OR "MICRO") AND REVIEWING IT.

### HOW TO CALCULATE THE OFFSET FOR MOUNTING

- Run `mmls` to query partition layout  
`# mmls image.E01`
- Identify partition and byte offset
- (Partition byte offset) x (bytes per sector) = `offset ####` to use!  
 Example: 63 X 512 = 32256

Note: If needed, repeat for each partition. Make new mount point:  
`# mkdir /mnt/windows_mount/2/`

### HELP? OPTIONS? USAGE?

`log2timeline -help`  
`Log2timeline-sift -help`  
`L2t_process -help`

### OTHER log2timeline OUTPUT FORMATS

Note: CSV is Default Output

- BeeDocs - Mac OS X visualization tool
- CEF - Common Event Format - ArcSight
- CFTL - XML file- CyberForensics TimeLab visualization tool
- CSV - comma separated value file
- Mactime - Both older and newer version of the format supported for use by TSK's mactime
- SIMILE - XML file - SIMILE timeline visualization widget
- SQLite - SQLite database
- TLN - Tab Delimited File
- TLN - Format used by some of H Carvey tools, expressed as an ASCII output
- TLNX - Format used by some of H Carvey tools, expressed as a XML document

## 10. CONNECT TO SIFT

- Settings -> Shared Folders -> Always Enabled (Check)
- SIFT Desktop -> VMware-Shared-Drive
- Access from a Win Machine \\SIFTWORKSTATION

## 11. REVIEW TIMELINE

- Review timelines using:
- Open, Soft, Filter with Excel
  - Import into SPLUNK
  - SIMILE
  - Tapestry

# Review of Integer Numbers

- Computers are made to deal with numbers
- What can we represent in N bits?
  - $2^N$  things, and no more! They could be...
  - Unsigned integers:
    - $0$  to  $2^N - 1$
    - (for  $N=32$ ,  $2^N - 1 = 4,294,967,295$ )
  - Signed Integers (Two's Complement)
    - $-2^{(N-1)}$  to  $2^{(N-1)} - 1$
    - (for  $N=32$ ,  $2^{(N-1)} = 2,147,483,648$ )

# What about other numbers?

1. Very large numbers? (seconds/millennium)  
 $\Rightarrow 31,556,926,000_{10}$  ( $3.1556926_{10} \times 10^{10}$ )
2. Very small numbers? (Bohr radius)  
 $\Rightarrow 0.0000000000529177_{10}\text{m}$  ( $5.29177_{10} \times 10^{-11}$ )
3. Numbers with both integer & fractional parts?  
 $\Rightarrow 1.5$

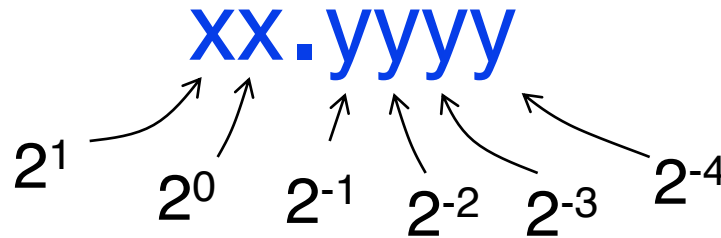
*First consider #3.*

*...our solution will also help with #1 and #2.*

# Representation of Fractions

“Binary Point” like decimal point signifies boundary between integer and fractional parts:

Example 6-bit representation:



$$10.1010_{\text{two}} = 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 2.625_{\text{ten}}$$

If we assume “fixed binary point”, range of 6-bit representations with this format:

0 to 3.9375 (almost 4)



# Fractional Powers of 2

<b>i</b>	<b><math>2^{-i}</math></b>	
0	1.0	1
1	0.5	1/2
2	0.25	1/4
3	0.125	1/8
4	0.0625	1/16
5	0.03125	1/32
6	0.015625	
7	0.0078125	
8	0.00390625	
9	0.001953125	
10	0.0009765625	
11	0.00048828125	
12	0.000244140625	
13	0.0001220703125	
14	0.00006103515625	
15	0.000030517578125	

# Representation of Fractions with Fixed Pt.

## What about addition and multiplication?

Addition is straightforward:

$$\begin{array}{r} 01.100 \\ + 00.100 \\ \hline 10.000 \end{array} \quad \begin{array}{r} 1.5_{\text{ten}} \\ 0.5_{\text{ten}} \\ \hline 2.0_{\text{ten}} \end{array} \quad \begin{array}{r} 01.100 \\ 00.100 \\ \hline 00.000 \\ 000.00 \\ 0110.0 \\ 00000 \\ 00000 \\ \hline 0000110000 \end{array} \quad \begin{array}{r} 1.5_{\text{ten}} \\ 0.5_{\text{ten}} \end{array}$$

Multiplication a bit more complex:

$$\begin{array}{r} 00.000 \\ 000.00 \\ 0110.0 \\ 00000 \\ 00000 \\ \hline 0000110000 \end{array}$$

Where's the answer, 0.11? (need to remember where point is)

# Representation of Fractions

So far, in our examples we used a “fixed” binary point. What we really want is to “float” the binary point. Why?

Floating binary point most effective use of our limited bits (and thus more accuracy in our number representation):

**example:** put  $0.1640625_{\text{ten}}$  into binary. Represent with 5-bits choosing where to put the binary point.

... 000000.001010100000...



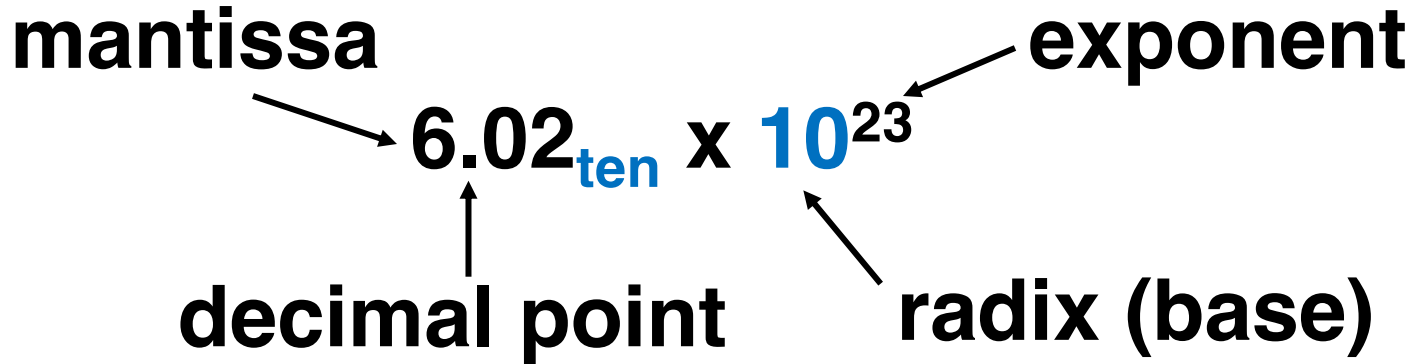
Store these bits and keep track of the binary point 2 places to the left of the MSB

Any other solution would lose accuracy!

With floating-point rep., each numeral carries an exponent field recording the whereabouts of its binary point.

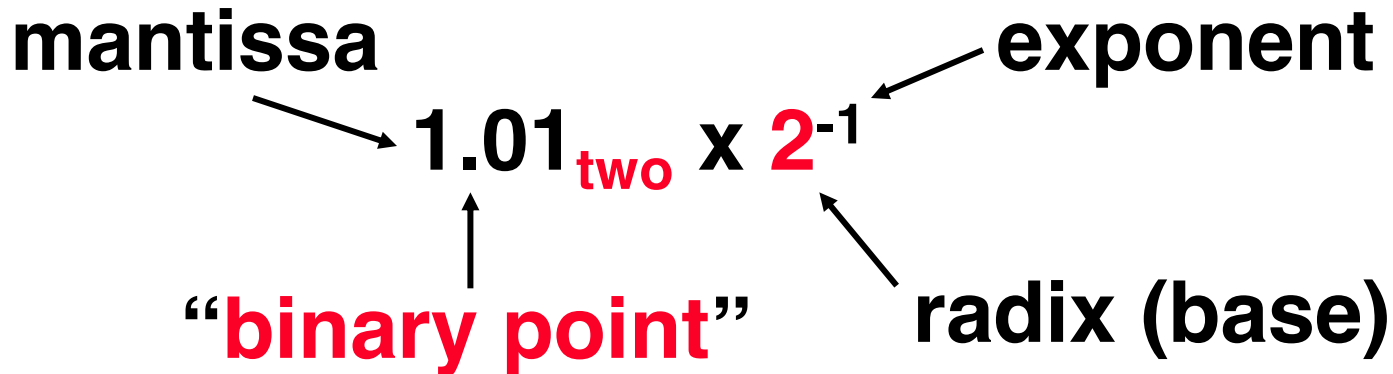
The binary point **can be outside** the stored bits, so very large and small numbers can be represented.

# Scientific Notation (in Decimal)



- Normalized form: no leading 0s (exactly one digit to left of decimal point)
- Alternatives to representing  $1/1,000,000,000$ 
  - Normalized:  $1.0 \times 10^{-9}$
  - Not normalized:  $0.1 \times 10^{-8}, 10.0 \times 10^{-10}$

# Scientific Notation (in Binary)



- Computer arithmetic that supports it called floating point, because it represents numbers where the binary point is not fixed, as it is for integers
  - Declare such variable in C as `float`
    - `double` for double precision.

# Floating-Point Representation (1/2)

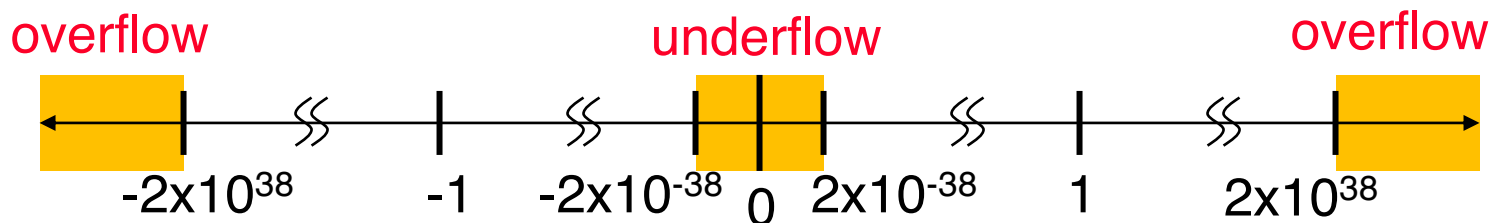
- Normal format:  $+1.x_{two} \dots x_{two} * 2^{y_{two} \dots y_{two}}$
- Multiple of Word Size (32 bits)



- **S** represents **Sign**  
**Exponent** represents **y**'s  
**Significand** represents **x**'s
- Represent numbers as small as  $2.0_{ten} \times 2^{-126}$  to as large as  $2.0_{ten} \times 2^{127}$
- $2^{126} = 8.507059173023462 \text{ e}37 \approx 10^{38}$

# Floating-Point Representation (2/2)

- What if result too large?  
( $> 2.0 \times 10^{38}$  ,  $< -2.0 \times 10^{38}$  )
  - **Overflow!**  $\Rightarrow$  Exponent larger than represented in 8-bit Exponent field
- What if result too small?  
( $>0$  &  $< 2.0 \times 10^{-38}$  ,  $<0$  &  $> -2.0 \times 10^{-38}$  )
  - **Underflow!**  $\Rightarrow$  Negative **exponent** larger than represented in 8-bit Exponent field



- What would help reduce chances of overflow and/or underflow?

# IEEE 754 Floating Point Standard (1/3)

Single Precision (Double Precision similar):



- **Sign** bit: 1 means negative 0 means positive
- **Significand** in *sign-magnitude* format (not 2's complement)
  - To pack more bits, leading 1 implicit for normalized numbers
  - 1 + 23 bits single, 1 + 52 bits double
  - always true:  $0 < \text{Significand} < 1$  (for normalized numbers)
- Note: 0 has no leading 1, so reserve exponent value 0 just for number 0



# IEEE 754 Floating Point Standard (2/3)

- IEEE 754 uses “biased exponent” representation
  - Designers wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares
  - Wanted bigger (integer) exponent field to represent bigger numbers
  - 2’s complement poses a problem (because negative numbers look bigger)
    - Use just magnitude and offset by half the range

# IEEE 754 Floating Point Standard (3/3)

- Called Biased Notation, where bias is number subtracted to get final number
  - IEEE 754 uses bias of 127 for single prec.
  - Subtract 127 from Exponent field to get actual value for exponent

- **Summary (single precision):**



- $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

- Double precision identical, except with exponent bias of 1023 (half, quad similar)

# Question

- Guess this Floating Point number:

1 1000 0000 1000 0000 0000 0000 0000 000

A:  $-1 \times 2^{128}$

B:  $+1 \times 2^{-128}$

C:  $-1 \times 2^1$

D:  $+1.5 \times 2^{-1}$

E:  $-1.5 \times 2^1$

# Representation for $\pm \infty$

- In FP, divide by 0 should produce  $\pm \infty$ , not overflow.
- Why?
  - OK to do further computations with  $\infty$   
E.g.,  $X/0 > Y$  may be a valid comparison
- IEEE 754 represents  $\pm \infty$ 
  - Most positive exponent reserved for  $\infty$
  - Significands all zeroes

# Representation for 0

- Represent 0?

- exponent all zeroes

- significand all zeroes

- What about sign? Both cases valid

- +0: 0 00000000 000000000000000000000000

- 0: 1 00000000 000000000000000000000000

# Special Numbers

- What have we defined so far? (Single Precision)

Exponent	Significand	Object
0	0	0
0	nonzero	???
1-254	anything	+/- fl. pt. #
255	0	+/- $\infty$
255	nonzero	???

Clever idea:

- Use  $\text{exp}=0,255$  &  $\text{Sig}\neq 0$

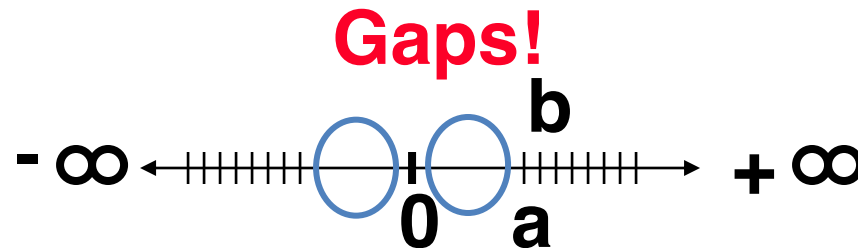
# Representation for Not a Number

- What do I get if I calculate  $\text{sqrt}(-4.0)$  or  $0/0$ ?
  - If  $\infty$  not an error, these shouldn't be either
  - Called Not a Number (NaN)
  - Exponent = 255, Significand nonzero
- Why is this useful?
  - Hope NaNs help with debugging?
  - They contaminate:  $\text{op}(\text{NaN}, X) = \text{NaN}$
  - Can use the significand to identify which!

# Representation for Denorms (1/2)

- Problem: There's a gap among representable FP numbers around 0
  - Smallest representable pos num:
    - $a = 1.0... 2 * 2^{-126} = 2^{-126}$
  - Second smallest representable pos num:
    - $b = 1.000.....1 2 * 2^{-126}$   
 $= (1 + 0.00...12) * 2^{-126}$   
 $= (1 + 2^{-23}) * 2^{-126}$   
 $= 2^{-126} + 2^{-149}$
  - $a - 0 = 2^{-126}$
  - $b - a = 2^{-149}$

**Normalization  
and implicit 1  
is to blame!**





# Representation for Denorms (2/2)

- **Solution:**

- We still haven't used Exponent = 0, Significand nonzero

- DEnormalized number: no (implied) leading 1, **implicit exponent = -126.**

- Smallest representable pos num:

$$a = 2^{-149}$$

- Second smallest representable pos num:

$$b = 2^{-148}$$



# Special Numbers

Exponent	Significand	Object
0	0	0
0	nonzero	Denorm
1-254	anything	+/- fl. pt. #
255	0	+/- $\infty$
255	nonzero	NAN

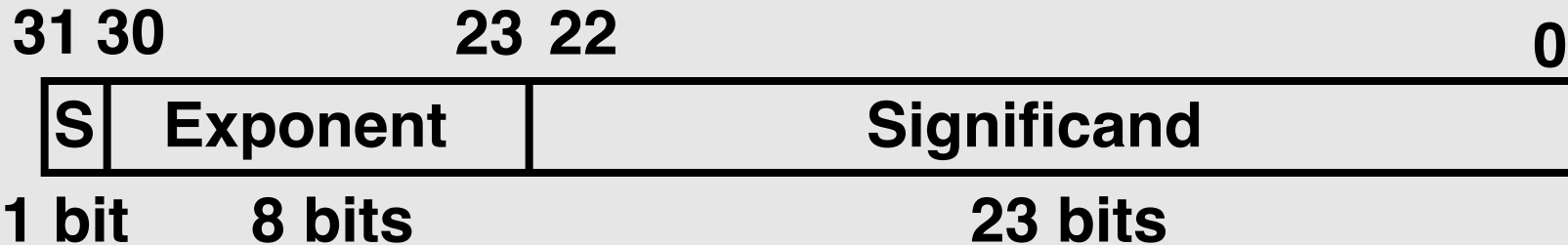
# Summary

Exponent tells Significand how much ( $2^i$ ) to count by (... , 1/4, 1/2, 1, 2, ...)

- Floating Point lets us:
  - Represent numbers containing both integer and fractional parts; makes efficient use of available bits.
  - Store **approximate** values for very large and very small #s.
- **IEEE 754 Floating-Point Standard** is most widely accepted attempt to standardize interpretation of such numbers (Every desktop or server computer sold since ~1997 follows these conventions)

Can store NaN,  $\pm \infty$

## • Summary (single precision):

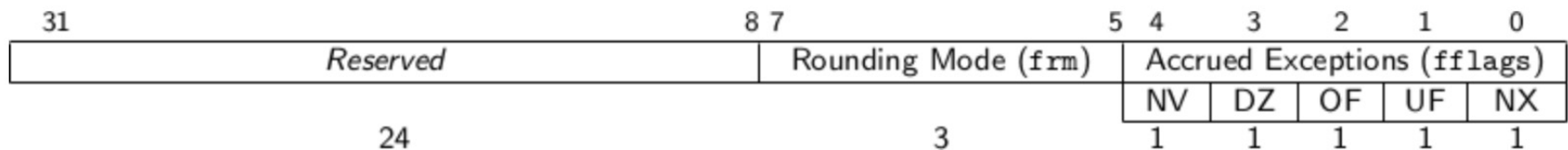


•  $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

- Double precision identical, except with exponent bias of 1023 (half, quad similar)

# RISC-V Single-Precision Floating-Point: F Extension

- 32 new registers f0 – f32 – each 32bit
  - Named registers: temporary, saved, argument
- Floating-point control and status register fcsr
  - Operating mode and exception status



Rounding Mode	Mnemonic	Meaning
000	RNE	Round to Nearest, ties to Even
001	RTZ	Round towards Zero
010	RDN	Round Down (towards $-\infty$ )
011	RUP	Round Up (towards $+\infty$ )
100	RMM	Round to Nearest, ties to Max Magnitude
101		<i>Reserved for future use.</i>
110		<i>Reserved for future use.</i>
111	DYN	In instruction's <i>rm</i> field, selects dynamic rounding mode; In Rounding Mode register, <i>reserved</i> .

Flag Mnemonic	Flag Meaning
NV	Invalid Operation
DZ	Divide by Zero
OF	Overflow
UF	Underflow
NX	Inexact

# Instruction Examples

- Load/ store – similar to int – e.g.:
  - `flw f1, 0(s1)`  
# load from address s1 to float reg 1
- Arithmetic: append `.s` for “single precision”
  - `fsub.s f2, f3, f1`
- Fused Multiply Add:
  - `Fmadd.s rd, rs1, rs2, rs3`  
#  $[rd] = [rs1] * [rs2] + [rs3]$
- Int / float conversions:
  - `fcvt.w.s f4, s4`  
# convert int in s4 to float in f4

<b>Category</b>	<b>Name</b>	<b>Fmt</b>	<b>RV32{F D Q} (HP/SP,DP,QP FI Pt)</b>
<b>Move</b>	Move from Integer	R	FMV.{H S}.X rd,rs1
	Move to Integer	R	FMV.X.{H S} rd,rs1
<b>Convert</b>	Convert from Int	R	FCVT.{H S D Q}.W rd,rs1
	Convert from Int Unsigned	R	FCVT.{H S D Q}.WU rd,rs1
	Convert to Int	R	FCVT.W.{H S D Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{H S D Q} rd,rs1
<b>Load</b>	Load	I	FL{W,D,Q} rd,rs1,imm
<b>Store</b>	Store	S	FS{W,D,Q} rs1,rs2,imm
<b>Arithmetic</b>	ADD	R	FADD.{S D Q} rd,rs1,rs2
	SUBtract	R	FSUB.{S D Q} rd,rs1,rs2
	MULTIply	R	FMUL.{S D Q} rd,rs1,rs2
	DIVide	R	FDIV.{S D Q} rd,rs1,rs2
	Square Root	R	FSQRT.{S D Q} rd,rs1
<b>Mul-Add</b>	Multiply-ADD	R	FMADD.{S D Q} rd,rs1,rs2,rs3
	Multiply-SUBtract	R	FMSUB.{S D Q} rd,rs1,rs2,rs3
	Negative Multiply-SUBtract	R	FNMSUB.{S D Q} rd,rs1,rs2,rs3
	Negative Multiply-ADD	R	FNMADD.{S D Q} rd,rs1,rs2,rs3
<b>Sign Inject</b>	SiGN source	R	FSGNJ.{S D Q} rd,rs1,rs2
	Negative SiGN source	R	FSGNJN.{S D Q} rd,rs1,rs2
	Xor SiGN source	R	FSGNJX.{S D Q} rd,rs1,rs2
<b>Min/Max</b>	MINimum	R	FMIN.{S D Q} rd,rs1,rs2
	MAXimum	R	FMAX.{S D Q} rd,rs1,rs2
<b>Compare</b>	Compare Float =	R	FEQ.{S D Q} rd,rs1,rs2
	Compare Float <	R	FLT.{S D Q} rd,rs1,rs2
	Compare Float ≤	R	FLE.{S D Q} rd,rs1,rs2
<b>Categorization</b>	Classify Type	R	FCLASS.{S D Q} rd,rs1
<b>Configuration</b>	Read Status	R	FRCSR rd
	Read Rounding Mode	R	FRRM rd
	Read Flags	R	FRFLAGS rd
	Swap Status Reg	R	FSCSR rd,rs1
	Swap Rounding Mode	R	FSRM rd,rs1
	Swap Flags	R	FSFLAGS rd,rs1
	Swap Rounding Mode Imm	I	FSRMI rd,imm
	Swap Flags Imm	I	FSFLAGSI rd,imm