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[1], Krste Asanović[1,2]
[1]SiFive Inc.,
[2]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
RVWMO	2.0	**Ratified**
RV32I	**2.1**	**Ratified**
RV64I	**2.1**	**Ratified**
RV32E	*1.9*	*Draft*
RV128I	*1.7*	*Draft*
Extension	**Version**	**Status**
M	**2.0**	**Ratified**
A	**2.1**	**Ratified**
F	**2.2**	**Ratified**
D	**2.2**	**Ratified**
Q	**2.2**	**Ratified**
C	**2.0**	**Ratified**
Counters	*2.0*	*Draft*
L	*0.0*	*Draft*
B	*0.0*	*Draft*
J	*0.0*	*Draft*
T	*0.0*	*Draft*
P	*0.2*	*Draft*
V	*0.7*	*Draft*
Zicsr	**2.0**	**Ratified**
Zifencei	**2.0**	**Ratified**
Zam	*0.1*	*Draft*
Ztso	*0.1*	*Frozen*

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
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	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  00000000000000000000001010110011
2  11111111111101011000010110010011
3  00000000101000101000001010110011
4  11111111111101011000010110010011
5  11111110000001011101110011100011
6  00000000101000000000010100110011
7  00000000000000001000000001100111
```

```
1  00000000000000000000001010110011
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     x1001     9
                1000
               0000
              0000
            +1000
            01001000     72
```

- m bits x 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 | 1001010          Dividend
             -1000
                 10
                101
               1010
              -1000
                 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!


CASIO
JF-120TM
TWO WAY POWER
12 DIGITS
F CUT 5/4
4 3 2 1 0 A
TAX−
TAX RATE
TAX+
RATE SET
%
GT
MRC
M
C
ON
AC
7
8
9
4
5
6
1
2
3
0
00
=
+
−
×

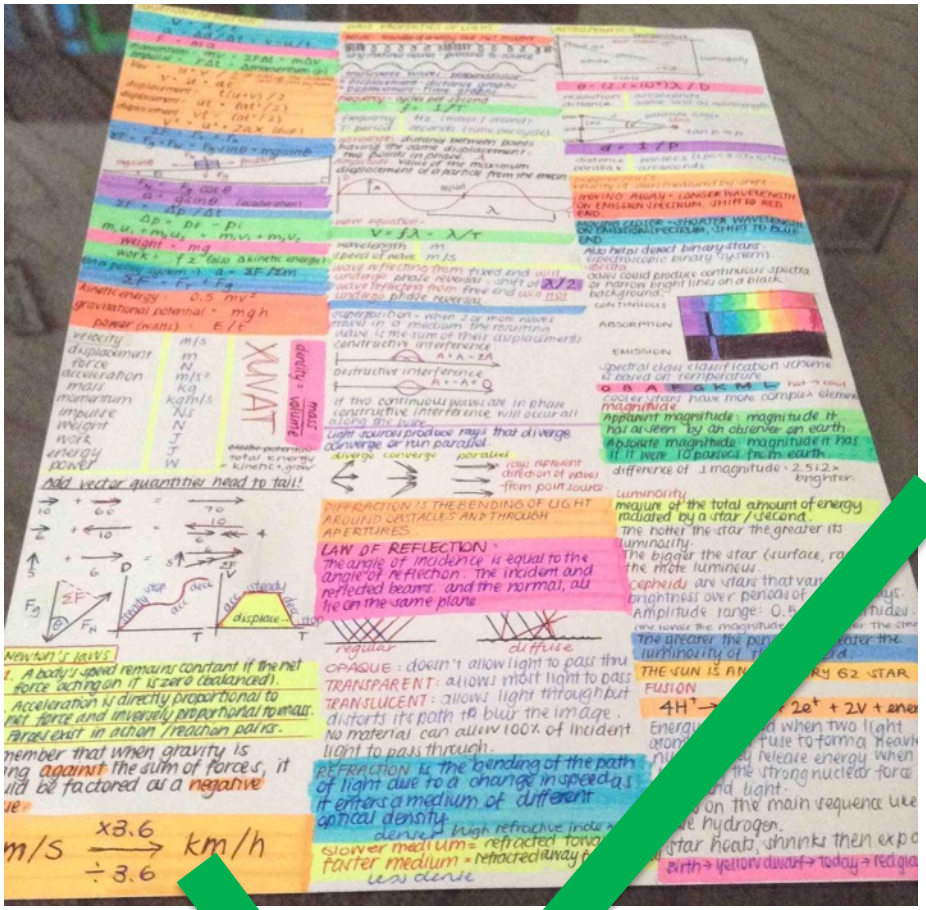

08:08
Thu, Sep 1
HUAWEI

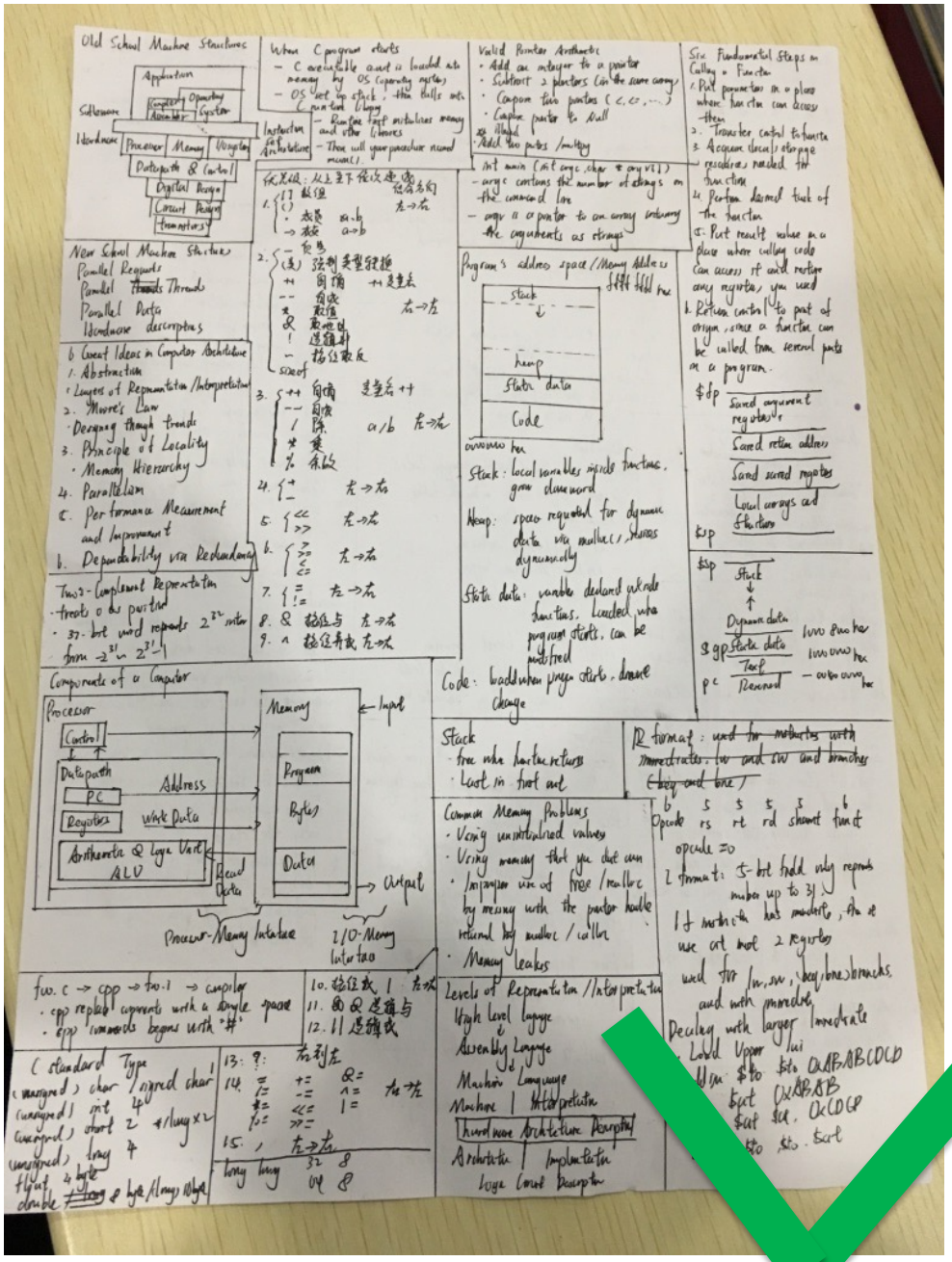

Tue
9

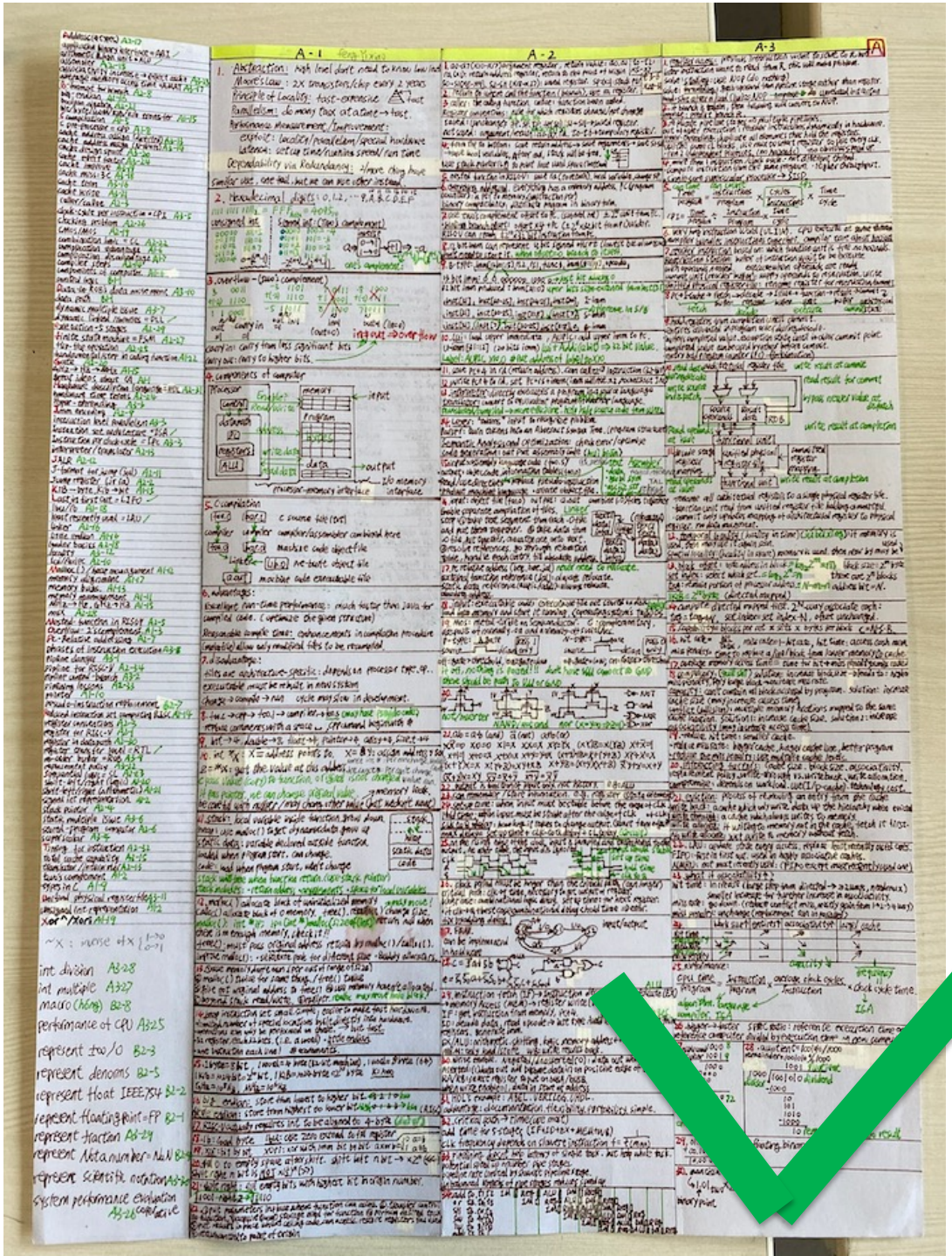

COMPUTER
ORGANIZATION
AND DESIGN
THE HA WARE/SOFTWA INTERFACE
RISC-V EDITION
MK
MORGAN KAUFMANN
DAVID A. PATTERSON
JOHN L. HENNESSY

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**!




Old School Machine Structures
Software
Hardware
Application
Compiler
Operating System
Assembler
Processor
Memory
I/O system
Datapath & Control
Digital Design
Circuit Design
Transistors
New School Machine Structures
Parallel Requests
Parallel Threads
Parallel Data
Hardware descriptions
6 Great 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
Components of a Computer
Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit
ALU
Memory
Input
Output
Program
Bytes
Data
Processor-Memory Interface
I/O-Memory Interfaces
Stack
Heap
Static data
Code
Levels of Representation / Interpretation
High Level Language
Assembly Language
Machine Language
Machine Interpretation
Hardware Architecture Description
Architecture Implementation
Logic Circuit Description
Common Memory Problems
Six Fundamental Steps in Calling a Function
Valid Pointer Arithmetic
Stack
Heap
Static data
Code
R format
I format
J format



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

SANS COMPUTER FORENSICS and INCIDENT RESPONSE

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.

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

oad: SIFT Workstation VM Appliance

Do FT Workstation Installation

2. BOOT SIFT VM

Login: *sansforensics*
Password: *forensics*

$ sudo su

3. ELEVATE PRIVS

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):

EWF/

nt_ewf.py image.E01 /mnt/ewf

 # mage.E01 /mnt/ewf/

Not Needed For 7-A

 # mount -t ntfs -o ro,loop,show_sys_files,streams_interface=windows,offset=#### /mnt/ewf/<image> /mnt/windows_mount/

MOUNT TO MOUNT POINT

DD

SINGLE IM

 # mount -t ntfs -o ... w_sys_files,streams_interface=windows,offset=#### image.dd /m... ows_mount/

SPLIT IMAGE (2 step proc...

 # affuse image.001 /mnt/aff

 # mount –t ntfs-3g –o loop,ro,show... /mnt/aff/<image> /mnt/windows...

HOW TO CALCULATE THE OFFSET FOR MOUNTING

1. Run mmls to query partition layout
 # mmls image.E01
2. Identify partition and byte offset
3. (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_mount2/*

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

7-A & 7-B

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 pl...

XP **# log2timeline-sift –z EST5EDT -p 0 -i parti...**

WIN7 **# log2timeline-sift -win7 -z EST5EDT ...tion**

OTHER USAGE EXAMPLES:

Display list of available plu...
 # log2timeline -f list
Run log2timeline use... use only specific plugins:
 # log2timeline-sift ...refetch –z EST5EDT -i image.dd
Help (man page...
 # log2timeli...

NUAL "MICRO" TIMELINE CREATION

...TIONS] [-f FORMAT] [-z TIMEZONE] [-o OUTPUT MODULE] [-w ...G_FILE/LOG_DIR [--] [FORMAT FILE OPTIONS]

...EM METADATA (using log2timeline or fls)

...stem data w/log2timeline from mounted file system:
 # log... ...f mft -o mactime –r -z EST5EDT -w mft.bo... ...olume/
OR Extract M... ...age using Sleuthkit:
 # fls -m "" -o offs... ...dd > fls.body
Convert body file for... ...format w/ mactime:
 # mactime –b fls.body –d...

ARTIFACTS (run l2l on mounted file s... ...h plugins recursively)

Extract artifacts w/ log2timeline and ru... ...d file system:
 # log2timeline -f firefox3,chrome -o mactim... ...EDT -w web.body /mnt/volume/
Convert body file format to CSV format w/ mactime:
 # mactime –b log2timeline.body –d > log2timeline.csv

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 a ASCII output
-TLNX - Format used by some of H Carvey tools, expressed as a XML document

8. CSV F... ...PUT (/cases/timeline-output-folder)

-dat... ...e event, in the format of MM/DD/YYYY
-t... ...of day, expressed in a 24h format, HH:MM:SS
-z: the timezone that was used to call the tool with.
-m: MACB meaning of the fields, comp w/ mactime format.
-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.

9. FILTER TIMELINE

Filter timeline with date range to include only:
l2t_process -b timeline.csv MM-DD-YYYY..MM-DD-YYYY > filtered.csv
Filter timeline with keyword list (one term per line in keywords.txt):
l2t_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:"
FiindMountPoints2 entries that reference E Drive
grep"MountPoints2 key" timeline.csv} | grep"E drive"
grepUSB 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

10. CONNECT TO SIFT

✓ ... > SETTINGS -> OPTIONS -> Shared Fo...ers -> Always Enabled (Check)

✓ 2. 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

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 containg 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
tln - Body file in the TLN format
volatility - Volatility output files (psscan2, sockscan2, ...)
win_link - Windows shortcut file (or a link file)
wmiprov - wmiprov log file
xpfirewall - XP Firewall log

List plugins # log2timeline -f list
...HELP EXPAND THIS LIST. BUILD PLUGINS!!!

BY DAVID NIDES (12/16/2011)
TWITTER: @DAVNADS
BLOG: DAVNADS.BLOGSPOT.COM
EMAIL: DNIDES@KPMG.COM

CREDITS TO: ED GOINGS, ROB LE... KRISTINN GUDJONSSON, KPMG &...!
QUESTIONS/FEEDBACK–CONTACT US!

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

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)
   => $31{,}556{,}926{,}000_{10}$ ($3.1556926_{10} \times 10^{10}$)
2. Very small numbers? (Bohr radius)
   => 0.0000000000529177_{10}m ($5.29177_{10} \times 10^{-11}$)
3. Numbers with <u>both</u> integer & fractional parts?
   => 1.5

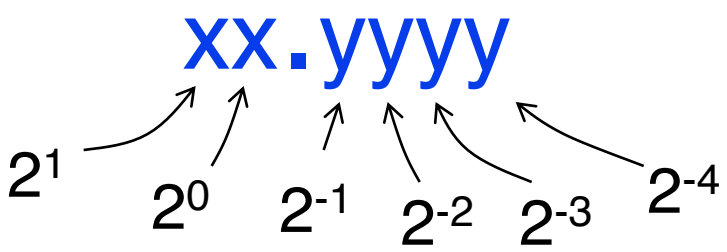
First consider #3.

…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:



$10.1010_{two} = 1x2^1 + 1x2^{-1} + 1x2^{-3} = 2.625_{ten}$

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

0 to 3.9375 (almost 4)

Fractional Powers of 2

i	2^{-i}	
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:

```
  01.100   1.5ten
+ 00.100   0.5ten
  10.000   2.0ten
```

Multiplication a bit more complex:

```
  01.100   1.5ten
  00.100   0.5ten
  -------
  00 000
 000 00
0110 0
00000
00000
----------
0000110000
```

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_{ten} into binary. Represent with 5-bits choosing where to put the binary point.

… 000000.001010100000…

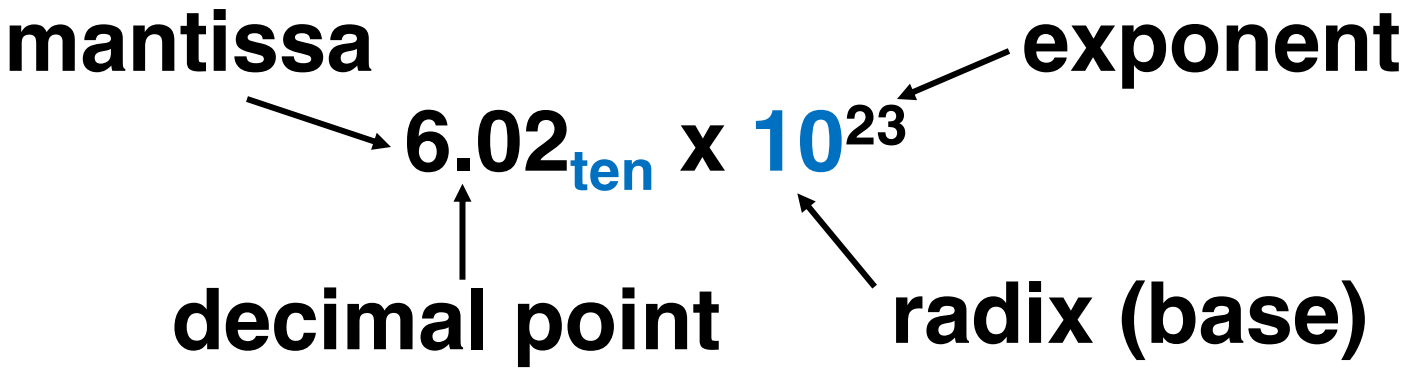
Store these bits and keep track of the binary point 2 places to the left of the MSB

Any other solution would lose accuracy!

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

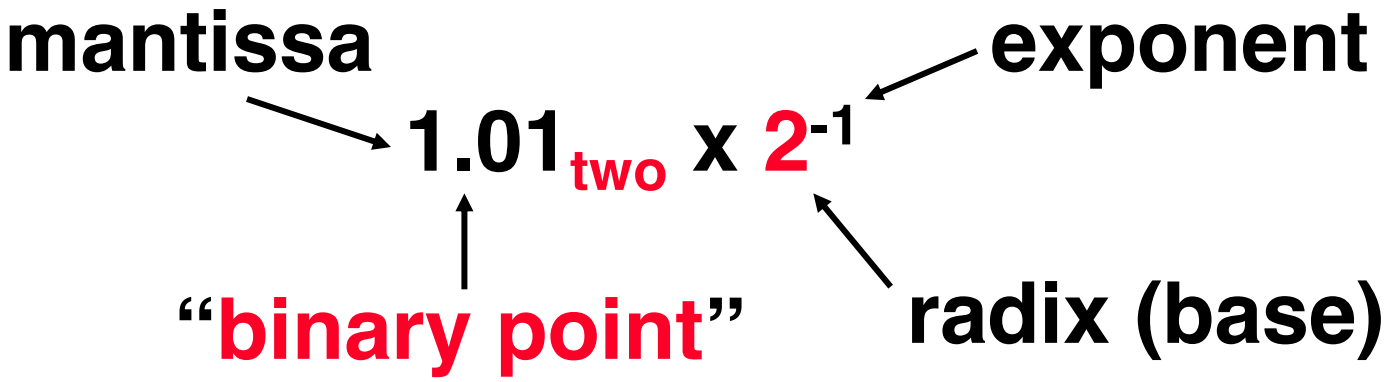
The binary point can be outside the stored bits, so very large and small numbers can be represented.

Scientific Notation (in Decimal)



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

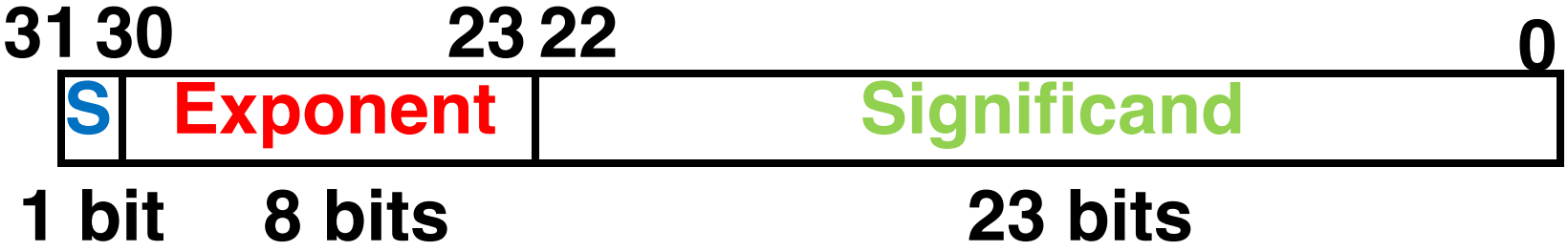
Scientific Notation (in Binary)



- 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: $+1.xxx\ldots x_{two} * 2^{yyy\ldots y_{two}}$
- 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 $2.0_{ten} \times 2^{127}$
- $2^{126} = 8.507059173023462\ e37 \approx 10^{38}$

Floating-Point Representation (2/2)

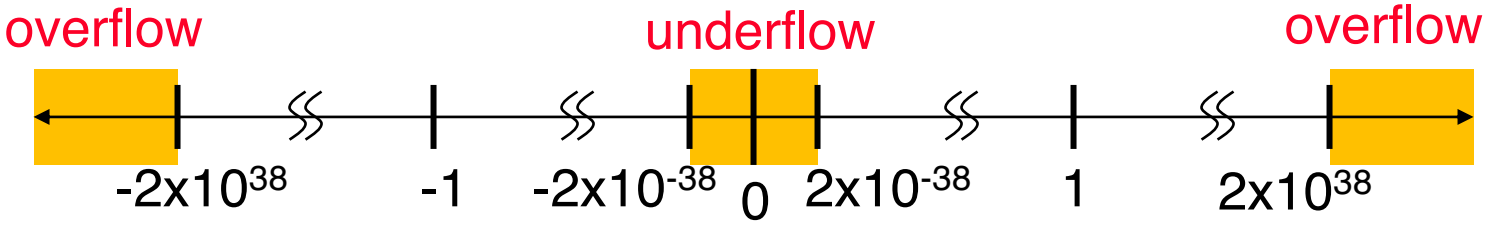
- What if result too large?

  (> 2.0×10^{38} , < -2.0×10^{38})

  - **Overflow!** => Exponent larger than represented in 8-bit Exponent field
- What if result too small?

  (>0 & < 2.0×10^{-38} , <0 & > -2.0×10^{-38})

  - **Underflow!** => 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 < 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):**

- **$(-1)^S$ x (1 + Significand) x $2^{(Exponent-127)}$**
  - 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: -1×2^{128}

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

C: -1×2^{1}

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

E: -1.5×2^{1}

Representation for ± ∞

- In FP, divide by 0 should produce ± ∞, not overflow.
- Why?
  - OK to do further computations with ∞
    E.g., X/0 > Y may be a valid comparison

- IEEE 754 represents ± ∞
  - Most positive exponent reserved for ∞
  - Significands all zeroes

Representation for 0

- Represent 0?
  - exponent all zeroes
  - significand all zeroes
  - What about sign? Both cases valid

```
+0: 0 00000000 00000000000000000000000
-0: 1 00000000 00000000000000000000000
```

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	+/- ∞
255	nonzero	**???**

Clever idea:

– Use exp=0,255 & Sig!=0

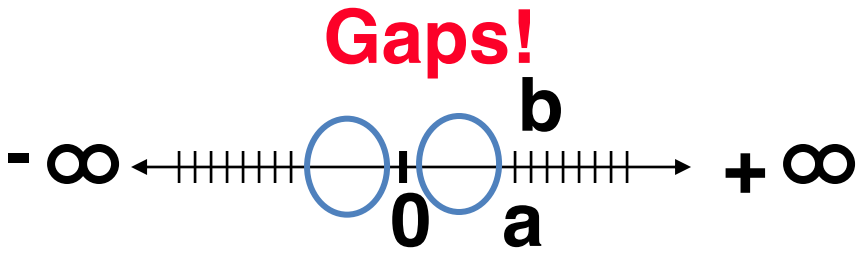
Representation for Not a Number

- What do I get if I calculate
  `sqrt(-4.0)` or `0/0`?
  - If ∞ 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: op(NaN, X) = 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!

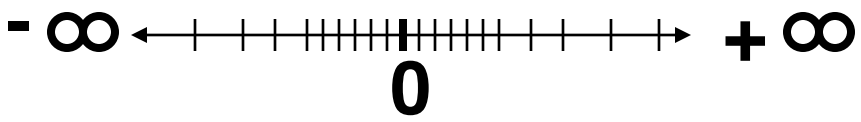


Representation for Denorms (2/2)

- **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	+/- ∞
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, ± ∞

- **Summary (single precision):**

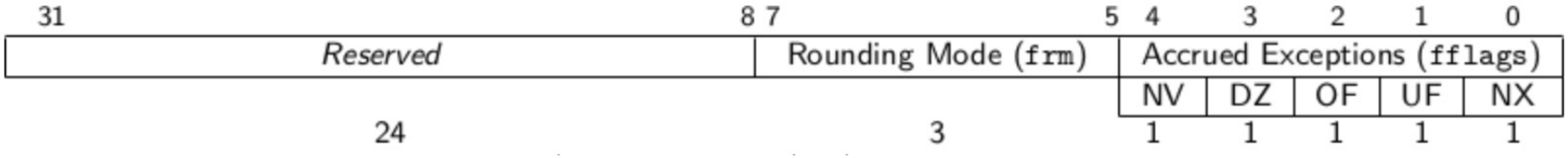
- **$(-1)^S$ x (1 + Significand) x $2^{(Exponent-127)}$**

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

Play with: https://www.h-schmidt.net/FloatConverter/IEEE754.html

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		*Reserved for future use.*
110		*Reserved for future use.*
111	DYN	In instruction's *rm* field, selects dynamic rounding mode; In Rounding Mode register, *reserved.*

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`

Category	Name	Fmt	RV32{F|D|Q} (HP/SP,DP,QP Fl Pt)	
Move	Move from Integer	R	FMV.{H|S}.X	rd,rs1
	Move to Integer	R	FMV.X.{H|S}	rd,rs1
Convert	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
Load	Load	I	FL{W,D,Q}	rd,rs1,imm
Store	Store	S	FS{W,D,Q}	rs1,rs2,imm
Arithmetic	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
Mul-Add	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
Sign Inject	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
Min/Max	MINimum	R	FMIN.{S|D|Q}	rd,rs1,rs2
	MAXimum	R	FMAX.{S|D|Q}	rd,rs1,rs2
Compare	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
Categorization	Classify Type	R	FCLASS.{S|D|Q}	rd,rs1
Configuration	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