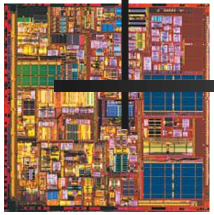
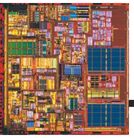


ECE 411 – Fall 2017

Lecture 18

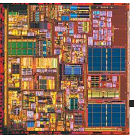
IEEE Floating-Point Standard

Objective



- To understand the fundamentals of floating-point representation
- To know the IEEE-754 Floating Point Standard
- CUDA GPU Floating-point speed, accuracy and precision
  - Cause of errors
  - Algorithm considerations
  - Deviations from IEEE-754
  - Accuracy of device runtime functions
  - -fastmath compiler option
  - Future performance considerations

What is IEEE floating-point format?



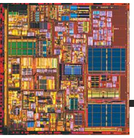
- A floating point binary number consists of three parts:
  - sign (S), exponent (E), and mantissa (M).
  - Each (S, E, M) pattern uniquely identifies a floating point number.

- For each bit pattern, its IEEE floating-point value is derived as:

  - value = $(-1)^S * M * \{2^E\}$, where $1.0 \le M < 10.0_{\mathbf{B}}$

- The interpretation of S is simple: S=0 results in a positive number and S=1 a negative number.

Normalized Representation



- Specifying that $1.0_B \leq M < 10.0_B$ makes the mantissa value for each floating point number unique.
  - For example, the only one mantissa value allowed for 0.5_D is M =1.0
    - $0.5_D = 1.0_B * 2^{-1}$
  - Neither $10.0_B * 2^{-2}$ nor $0.1_B * 2^{0}$ qualifies

- Because all mantissa values are of the form 1.XX…, one can omit the "1." part in the representation.
  - The mantissa value of 0.5_D in a 2-bit mantissa is 00, which is derived by omitting "1." from 1.00.
  - Mantissa without implied 1 is called the *fraction*

Exponent Representation

- In an n-bits exponent representation, $2^{n-1}-1$ is added to its 2's complement representation to form its excess representation.
  - See Table for a 3-bit exponent representation
- A simple unsigned integer comparator can be used to compare the magnitude of two FP numbers
- Symmetric range for +/- exponents (111 reserved)

2's complement	Actual decimal	Excess-3
000	0	011
001	1	100
010	2	101
011	3	110
100	**(reserved pattern)**	**111**
101	-3	000
110	-2	001
111	-1	010

A simple, hypothetical 5-bit FP format

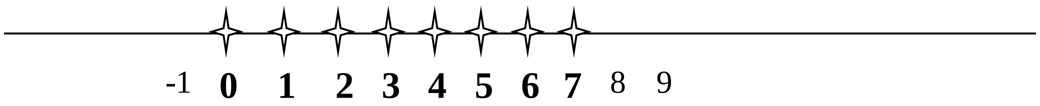
- Assume 1-bit S, 2-bit E, and 2-bit M
  - $0.5_D = 1.00_B * 2^{-1}$
  - 0.5_D = **0 00 00**, where S = 0, E = 00, and M = (1.)00

2’s complement	Actual decimal	Excess-1
00	0	01
01	1	10
10	(reserved pattern)	11
11	-1	**00**

Representable Numbers

- The representable numbers of a given format is the set of all numbers that can be exactly represented in the format.
- See Table for representable numbers of an unsigned 3-bit integer format

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7



Representable n 5-bit Hypothetical I

Cannot represent Zero!

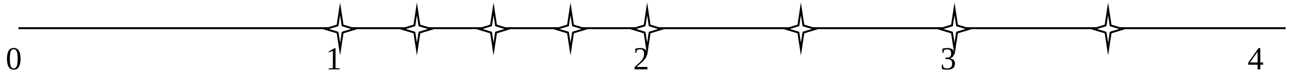
0

		No-zero		Abrupt underflow		Gradual underflow	
E	M	**S=0**	**S=1**	S=0	S=1	S=0	S=1
00	00	2^{-1}	$-(2^{-1})$	0	0	0	0
	01	$2^{-1}+1*2^{-3}$	$-(2^{-1}+1*2^{-3})$	0	0	$1*2^{-2}$	$-1*2^{-2}$
	10	$2^{-1}+2*2^{-3}$	$-(2^{-1}+2*2^{-3})$	0	0	$2*2^{-2}$	$-2*2^{-2}$
	11	$2^{-1}+3*2^{-3}$	$-(2^{-1}+3*2^{-3})$	0	0	$3*2^{-2}$	$-3*2^{-2}$
01	00	2^{0}	$-(2^{0})$	2^{0}	$-(2^{0})$	2^{0}	$-(2^{0})$
	01	$2^{0}+1*2^{-2}$	$-(2^{0}+1*2^{-2})$	$2^{0}+1*2^{-2}$	$-(2^{0}+1*2^{-2})$	$2^{0}+1*2^{-2}$	$-(2^{0}+1*2^{-2})$
	10	$2^{0}+2*2^{-2}$	$-(2^{0}+2*2^{-2})$	$2^{0}+2*2^{-2}$	$-(2^{0}+2*2^{-2})$	$2^{0}+2*2^{-2}$	$-(2^{0}+2*2^{-2})$
	11	$2^{0}+3*2^{-2}$	$-(2^{0}+3*2^{-2})$	$2^{0}+3*2^{-2}$	$-(2^{0}+3*2^{-2})$	$2^{0}+3*2^{-2}$	$-(2^{0}+3*2^{-2})$
10	00	2^{1}	$-(2^{1})$	2^{1}	$-(2^{1})$	2^{1}	$-(2^{1})$
	01	$2^{1}+1*2^{-1}$	$-(2^{1}+1*2^{-1})$	$2^{1}+1*2^{-1}$	$-(2^{1}+1*2^{-1})$	$2^{1}+1*2^{-1}$	$-(2^{1}+1*2^{-1})$
	10	$2^{1}+2*2^{-1}$	$-(2^{1}+2*2^{-1})$	$2^{1}+2*2^{-1}$	$-(2^{1}+2*2^{-1})$	$2^{1}+2*2^{-1}$	$-(2^{1}+2*2^{-1})$
	11	$2^{1}+3*2^{-1}$	$-(2^{1}+3*2^{-1})$	$2^{1}+3*2^{-1}$	$-(2^{1}+3*2^{-1})$	$2^{1}+3*2^{-1}$	$-(2^{1}+3*2^{-1})$
11	Reserved pattern						

Flush to Zero

- Treat all bit patterns with E=0 as 0.0
  - This takes away several representable numbers near zero and lump them all into 0.0
  - For a representation with large M, a large number of representable numbers will be removed.

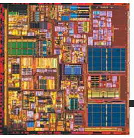
0



Flush to Zero

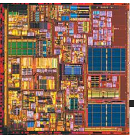
0

		No-zero		**Flush to Zero**		Denormalized	
E	M	S=0	S=1	**S=0**	**S=1**	S=0	S=1
00	00	2^{-1}	$-(2^{-1})$	**0**	**0**	0	0
	01	$2^{-1}+1*2^{-3}$	$-(2^{-1}+1*2^{-3})$	**0**	**0**	$1*2^{-2}$	$-1*2^{-2}$
	10	$2^{-1}+2*2^{-3}$	$-(2^{-1}+2*2^{-3})$	**0**	**0**	$2*2^{-2}$	$-2*2^{-2}$
	11	$2^{-1}+3*2^{-3}$	$-(2^{-1}+3*2^{-3})$	**0**	**0**	$3*2^{-2}$	$-3*2^{-2}$
01	00	2^{0}	$-(2^{0})$	**2^{0}**	**$-(2^{0})$**	2^{0}	$-(2^{0})$
	01	$2^{0}+1*2^{-2}$	$-(2^{0}+1*2^{-2})$	**$2^{0}+1*2^{-2}$**	**$-(2^{0}+1*2^{-2})$**	$2^{0}+1*2^{-2}$	$-(2^{0}+1*2^{-2})$
	10	$2^{0}+2*2^{-2}$	$-(2^{0}+2*2^{-2})$	**$2^{0}+2*2^{-2}$**	**$-(2^{0}+2*2^{-2})$**	$2^{0}+2*2^{-2}$	$-(2^{0}+2*2^{-2})$
	11	$2^{0}+3*2^{-2}$	$-(2^{0}+3*2^{-2})$	**$2^{0}+3*2^{-2}$**	**$-(2^{0}+3*2^{-2})$**	$2^{0}+3*2^{-2}$	$-(2^{0}+3*2^{-2})$
10	00	2^{1}	$-(2^{1})$	**2^{1}**	**$-(2^{1})$**	2^{1}	$-(2^{1})$
	01	$2^{1}+1*2^{-1}$	$-(2^{1}+1*2^{-1})$	**$2^{1}+1*2^{-1}$**	**$-(2^{1}+1*2^{-1})$**	$2^{1}+1*2^{-1}$	$-(2^{1}+1*2^{-1})$
	10	$2^{1}+2*2^{-1}$	$-(2^{1}+2*2^{-1})$	**$2^{1}+2*2^{-1}$**	**$-(2^{1}+2*2^{-1})$**	$2^{1}+2*2^{-1}$	$-(2^{1}+2*2^{-1})$
	11	$2^{1}+3*2^{-1}$	$-(2^{1}+3*2^{-1})$	**$2^{1}+3*2^{-1}$**	**$-(2^{1}+3*2^{-1})$**	$2^{1}+3*2^{-1}$	$-(2^{1}+3*2^{-1})$
11	Reserved pattern						



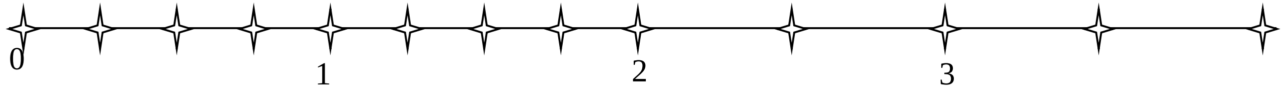
Why is Flush to Zero problematic?

- Many physical model calculations work on values that are very close to zero
  - Dark (but not totally black) sky in movie rendering
  - Small distance fields in electrostatic potential calculation
  - …
- With Flush to Zero, these calculations tend to create artifacts that compromise the integrity of the models



Denormalized Numbers

- The actual method adopted by the IEEE standard is called denromalized numbers or gradual underflow.
  - The method relaxes the normalization requirement for numbers very close to 0.
  - whenever E=0, the mantissa is no longer assumed to be of the form 1.XX. Rather, it is assumed to be 0.XX. In general, if the n-bit exponent is 0, the value is
    - $0.M * 2^{- 2\wedge(n-1) + 2}$



Denormalization

0

		No-zero		Flush to Zero		**Denormalized**	
E	M	S=0	S=1	S=0	S=1	**S=0**	**S=1**
00	00	2^{-1}	$-(2^{-1})$	0	0	**0**	**0**
	01	$2^{-1}+1*2^{-3}$	$-(2^{-1}+1*2^{-3})$	0	0	**$1*2^{-2}$**	**$-1*2^{-2}$**
	10	$2^{-1}+2*2^{-3}$	$-(2^{-1}+2*2^{-3})$	0	0	**$2*2^{-2}$**	**$-2*2^{-2}$**
	11	$2^{-1}+3*2^{-3}$	$-(2^{-1}+3*2^{-3})$	0	0	**$3*2^{-2}$**	**$-3*2^{-2}$**
01	00	2^{0}	$-(2^{0})$	2^{0}	$-(2^{0})$	2^{0}	$-(2^{0})$
	01	$2^{0}+1*2^{-2}$	$-(2^{0}+1*2^{-2})$	$2^{0}+1*2^{-2}$	$-(2^{0}+1*2^{-2})$	$2^{0}+1*2^{-2}$	$-(2^{0}+1*2^{-2})$
	10	$2^{0}+2*2^{-2}$	$-(2^{0}+2*2^{-2})$	$2^{0}+2*2^{-2}$	$-(2^{0}+2*2^{-2})$	$2^{0}+2*2^{-2}$	$-(2^{0}+2*2^{-2})$
	11	$2^{0}+3*2^{-2}$	$-(2^{0}+3*2^{-2})$	$2^{0}+3*2^{-2}$	$-(2^{0}+3*2^{-2})$	$2^{0}+3*2^{-2}$	$-(2^{0}+3*2^{-2})$
10	00	2^{1}	$-(2^{1})$	2^{1}	$-(2^{1})$	2^{1}	$-(2^{1})$
	01	$2^{1}+1*2^{-1}$	$-(2^{1}+1*2^{-1})$	$2^{1}+1*2^{-1}$	$-(2^{1}+1*2^{-1})$	$2^{1}+1*2^{-1}$	$-(2^{1}+1*2^{-1})$
	10	$2^{1}+2*2^{-1}$	$-(2^{1}+2*2^{-1})$	$2^{1}+2*2^{-1}$	$-(2^{1}+2*2^{-1})$	$2^{1}+2*2^{-1}$	$-(2^{1}+2*2^{-1})$
	11	$2^{1}+3*2^{-1}$	$-(2^{1}+3*2^{-1})$	$2^{1}+3*2^{-1}$	$-(2^{1}+3*2^{-1})$	$2^{1}+3*2^{-1}$	$-(2^{1}+3*2^{-1})$
11		Reserved pattern					

IEEE 754 Format and Precision

- Single Precision
  - 1 bit sign, 8 bit exponent (bias-127 excess), 23 bit fraction
- Double Precision
  - 1 bit sign, 11 bit exponent (1023-bias excess), 52 bit fraction
  - The largest error for representing a number is reduced to $1/2^{29}$ of single precision representation
- Half Precision
  - Newly proposed standard for image and graphics processing

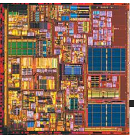
Special Bit Patterns

exponent	mantissa	meaning
11…1	≠ 0	NaN
11…1	=0	$(-1)^S * \infty$
00…0	≠0	denormalized
00…0	=0	0

- An ∞ can be created by overflow, e.g., divided by zero. Any representable number divided by +∞ or -∞ results in 0.
- NaN (Not a Number) is generated by operations whose input values do not make sense, for example, 0/0, 0*∞, ∞/∞, ∞-∞.
  - also used to for data that have not been properly initialized in a program.
  - Signaling NaN’s (SNaNs) are represented with most significant mantissa bit cleared whereas
  - Quiet NaN’s are represented with most significant mantissa bit set.

Floating Point Accuracy and Rounding

- The accuracy of a floating point arithmetic operation is measured by the maximal error introduced by the operation.
- The most common source of error in floating point arithmetic is when the operation generates a result that cannot be exactly represented and thus requires rounding.
- Rounding occurs if the mantissa of the result value needs too many bits to be represented exactly.



Rounding and Error

- Assume our 5-bit representation, consider

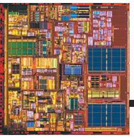
$1.0*2^{-2}$ (0, 00, 01) + $1.00*2^{1}$ (0, 10, 00)

denorm

- The hardware needs to shift the mantissa bits in order to align the correct bits with equal place value with each other

$0.001*2^{1}$ (0, 00, 0001) + $1.00*2^{1}$ (0, 10, 00)

The ideal result would be $1.001 * 2^{1}$ (0, 10, 001) but this would require 3 mantissa bits!



Rounding and Error

- In some cases, the hardware may only perform the operation on a limited number of bits for speed and area cost reasons
  - An adder may only have 3 bit positions in our example so the first operand would be treated as a 0.00
  - Additional bit positions in adders are needed to maintain the accuracy for final rounding

$0.001 * 2^1$ (0, 00, 0001) + $1.00 * 2^1$ (0, 10, 00)

Example

- Assume our 5-bit representation, consider

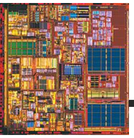
$1.00*2^1$ (0, 10, 00) $-1.0*2^{-2}$ (0, 00, 01)

$= 1.000 * 2^1 – 0.001 * 2^1$

$= 0.111 * 2^1$

$= 1.11 * 2^0$ (0, 01 11)

This is a perfectly representable result! It needs 4 bit positions in the adder.



Error Measure

- If an hardware adder has at least two more bit positions than the total (both implicit and explicit) number of mantissa bits, the rounding error for addition and subtraction would never be more than half of the place value of the mantissa
  - 0.001 in our 5-bit format

- We refer to this as 0.5 ULP (Units in the Last Place).
  - If the hardware is designed to perform arithmetic and rounding operations perfectly, the most error that one should introduce should be no more than 0.5 ULP.
  - The error is actually limited by the precision of the format for this case.

Order of operations matters.

- Floating Point operations are not strictly associative
- The root cause is that some times a very small number can disappear when added to or subtracted from a very large number.
  - (Large + Small) + Small ≠ Large + (Small + Small)

Algorithm Considerations

- Sequential sum

$$1.00*2^0 + 1.00*2^0 + 1.00*2^{-2} + 1.00*2^{-2}$$
$$= 1.00*2^1 + 1.00*2^{-2} + 1.00*2^{-2}$$
$$= 1.00*2^1 + 1.00*2^{-2}$$
$$= 1.00*2^1$$

- Parallel reduction

$$(1.00*2^0 + 1.00*2^0) + (1.00*2^{-2} + 1.00*2^{-2})$$
$$= 1.00*2^1 + 1.00*2^{-1}$$
$$= 1.0\underline{1}*2^1$$

READ THE SUPPLEMENT READING