Memory and Floating Point (part I)

CSCI 237: Computer Organization
8th Lecture, Monday, Sept. 23

Kelly Shaw

1

1

Administrative Details

- Lab #2 checkpoint Tuesday at 11pm
  - What questions do you have?
- Read CSAPP 2.4-2.5

2

2

Last Time: Arithmetic Operations

- Integers (Ch 2.2-2.3)
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, multiplication, shifting (Ch 2.3)

3

3

Today: Floating Point (part I)

- Integers (Ch 2.2-2.3)
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, multiplication, shifting (Ch 2.3)
- Byte-oriented Memory Organization
  - Byte-ordering
- Background: Fractional binary numbers
- IEEE FP standard (normalized and denormalized values)
- Examples

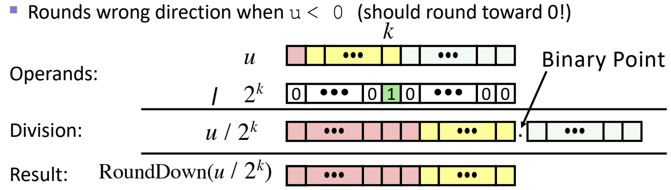
4

4

Signed Power-of-2 Divide with Shift

- Operation
  - `u >> k` gives ⌊ `u / 2`^{k} ⌋
  - Uses arithmetic shift
  - Rounds wrong direction when `u < 0` (should round toward 0!)

We want::
⌊ u / 2^k ⌋ if u > 0, but
⌈ u / 2^k ⌉ if u < 0



	Division	Computed	Hex	Binary
`y`	-15213	-15213	`C4 93`	`11000100 10010011`
`y >> 1`	-7606.5	-7607	`E2 49`	`11100010 01001001`
`y >> 4`	-950.8125	-951	`FC 49`	`11111100 01001001`
`y >> 8`	-59.4257813	-60	`FF C4`	`11111111 11000100`

5

5

Divide by Power of 2 (k = 2, divide by 4)

	-16/4 = -4	-17/4 = -4.25	-18/4 = -4.5	-19/4 = -4.75
	1100 00	1011 11	1011 10	1011 01
Divide by Shifting 2	111100 ✓	111011 ✗	111011 ✗	111011 ✗
Add bias	1100 00 + 0000 11 ----------------- 1100 11	1011 11 + 0000 11 ----------------- 1100 10	1011 10 + 0000 11 ----------------- 1100 01	1011 01 + 0000 11 ----------------- 1100 00
Divide by Shifting 2	111100 ✓ -4	111100 ✓ -4	111100 ✓ -4	111100 ✓ -4

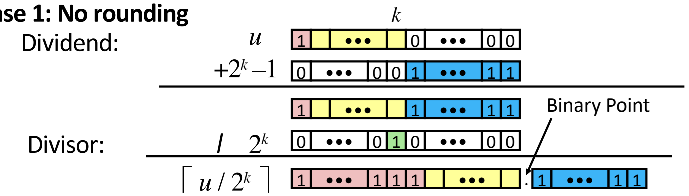
6

6

Signed Power-of-2 Divide

- Operation behavior?
  - Want ⌊ `u / 2`^{k} ⌋ for positive quotients (Round Toward 0)
  - Want ⌈ `u / 2`^{k} ⌉ for negative quotients (Round Toward 0)
  - Compute as ⌊ `(u+2`^{k}`-1)/ 2`^{k} ⌋
    - In C: `(u + (1<<k)-1) >> k`
    - 2^k-1 Biases dividend toward 0

Case 1: No rounding



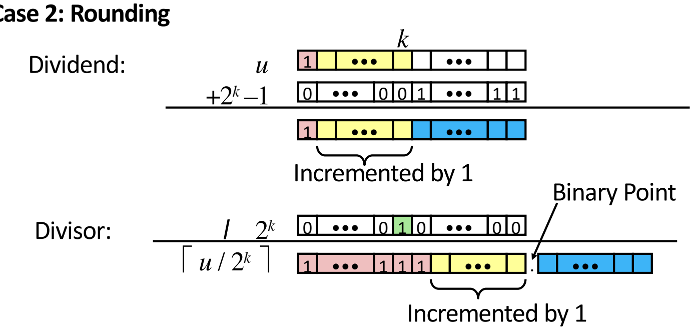
Biasing has no effect

7

7

Correct Signed Power-of-2 Divide (Cont.)

Case 2: Rounding



Biasing adds 1 to final result

8

8

Practice On Your Own

- What would be be printed?

```
char c = 128;
char d = 3;
char e = c * d;
printf(”e: %d \n”, e);

char s = 42;
char r1 = s / 8;
char r2 = s >> 3;
printf(“r1: %d r2: %d \n”, r1, r2);
```

9

9

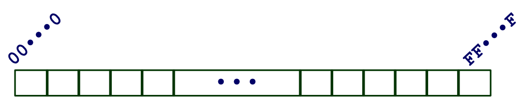
Today: Floating Point (part I)

- Integers (Ch 2.2-2.3)
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, multiplication, shifting (Ch 2.3)
- Byte-oriented Memory Organization
  - Byte-ordering
- Background: Fractional binary numbers
- IEEE FP standard (normalized and denormalized values)
- Examples

10

10

Byte-Oriented Memory Organization



- Programs refer to data by address
  - Conceptually, envision memory as a very large array of bytes
    - (In reality, it's not, but we can think of it that way…)
  - An address is like an index into that array
    - Each variable is assigned an address
    - “Type” of a variable tells us how to interpret bytes at the variable's associated address
- Note: system provides private “address spaces” to each “process”
  - Think of a process as a program being executed
  - So, a program can clobber its own data, but not that of others

11

11

Machine Words

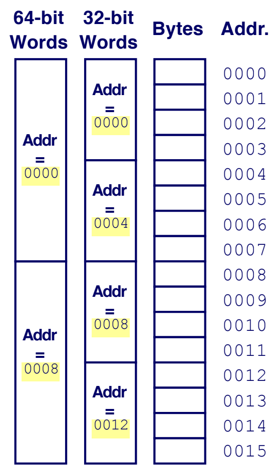
- Any given computer has a “Word Size”
  - Word size is the size of an address (pointer)
- Until recently*, most machines used 32 bit words (4 bytes)
  - This limits addresses (memory size) to 4GiB (2^{32} bytes)
- Now, most* machines have 64-bit word size
  - Potentially, could have 18 EB (exabytes) of addressable memory
  - That's 18.4 X 10^{18}
- Machines still support multiple data formats
  - Primitive types are always an integral number of bytes
  - Fractions (e.g., `char`, `short`) or multiples (e.g., `long int`) of word size

12

12

Word-Oriented Memory Organization

- Addresses Specify Byte Locations
  - Address "points to" first byte in word
  - Addresses of successive words always differ by 4 (32-bit) or 8 (64-bit)



13

13

Aside: Byte Ordering

- So, how are the bytes within a multi-byte word ordered in memory?
- Conventions
  - Big Endian: Sun, PPC Mac, Internet
    - Least significant byte has highest address
  - Little Endian: x86, ARM processors running Android, iOS, and Windows
    - Least significant byte has lowest address

14

14

Aside: Byte Ordering Example

- Example
  - Let variable `x` have 4-byte value of 0x01234567
  - The address given by `&x` is 0x100

Big Endian

		0x100	0x101	0x102	0x103		
		01	23	45	67		

Little Endian

		0x100	0x101	0x102	0x103		
		67	45	23	01		

Big Endian: Least significant byte has highest address
Little Endian: Least significant byte has lowest address

15

15

Aside: Byte ordering Takeaway

- Endianness matters…
- But you really just need to be aware of it
- Consider endianness any time you store data for interpretation by another program in the future
- Thankfully, most network and I/O libraries contain "serialization" and "deserialization" functions that take care of translating endianness for you

16

16

Today: Floating Point (part I)

- Integers (Ch 2.2-2.3)
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, multiplication, shifting (Ch 2.3)
- Byte-oriented Memory Organization
  - Byte-ordering
- Background: Fractional binary numbers
- IEEE FP standard (normalized and denormalized values)
- Examples

17

17

How Can We Represent Numbers with Fractional Components in Hardware?

- How do we express decimal numbers in binary?
- Examples:
  - 3.14
  - 2/3
  - $1.3*10^{-15}$
  - $-7.2*10^{20}$

18

18

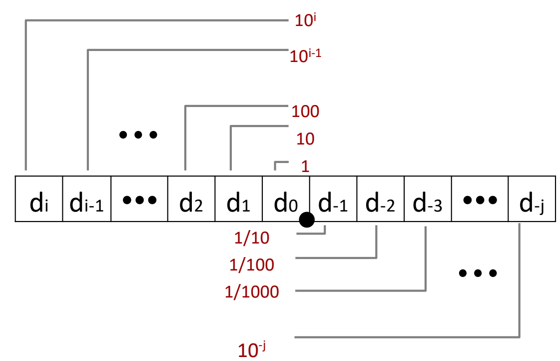
Fractional binary numbers

- What is decimal representation of $\texttt{1011.101}_2$?

19

19

Fractional Decimal Numbers

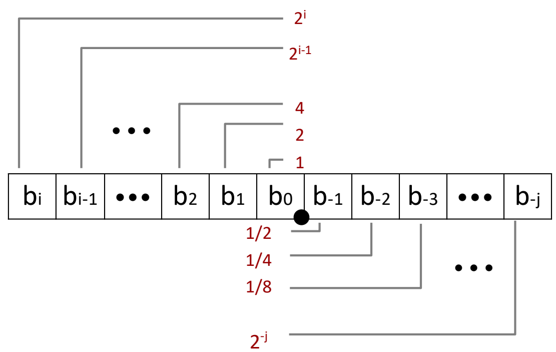


- Digits to right of “decimal point” represent fractional powers of 10
- Represents rational number: $\sum_{k=-j}^{i} d_k \times 10^k$

20

20

Fractional Binary Numbers



- Bits to right of "binary point" represent fractional powers of 2
- Represents rational number: $\sum_{k=-j}^{i} b_k \times 2^k$

Recap: Fractional binary numbers

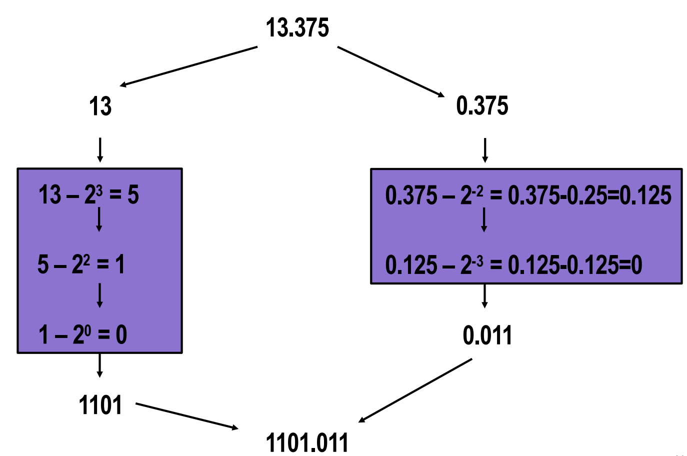
- What is decimal representation of 1011.101_2?
  $2^3 2^2 2^1 2^0 . 2^{-1} 2^{-2} 2^{-3}$
  - $(2^3 * 1) + (2^2 * 0) + (2^1 * 1) + (2^0 * 1) + (2^{-1} * 1) + (2^{-2} * 0) + (2^{-3} * 1) = 11.625$

Fractional Binary Numbers: Examples

Value	Representation
$23/4 = 5\ 3/4 = 4 + 1 + \frac{1}{2} + \frac{1}{4}$	101.11_2
$23/8 = 2\ 7/8 = 2 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}$	10.111_2
$23/16 = 1\ 7/16 = 1 + \frac{1}{4} + \frac{1}{8} + \frac{1}{16}$	1.0111_2

- Observations
  - Divide by 2 by shifting right (unsigned)
  - Multiply by 2 by shifting left
  - Numbers of form $0.111111\ldots_2$ are just below 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - If we use notation $1.0 - \varepsilon$, then adding more bits brings ε closer and closer to 0

Decimal to Binary Example



Practice on Your Own

- Convert the decimal number 27.3125 to its binary representation

- Convert the binary number 1101.1001 to its decimal representation

25

25

Fractional Binary Number Rep. is Limited

- Limitation #1
  - Can only **exactly** represent fractional numbers of the form $x/2^k$
    - Other rational numbers have repeating bit representations
  - Value / Representation
    - 1/3 `0.0101010101[01]`…$_2$
    - 1/5 `0.001100110011[0011]`…$_2$
    - 1/10 `0.0001100110011[0011]`…$_2$

- Limitation #2
  - If we standardize representation, we would have one fixed location for binary point within the *w* bits
    - Limited range of numbers (very small values? very large?)

26

26

IEEE Floating Point

- IEEE Standard 754
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - IEEE 754 is supported by all major CPUs

- Driven by numerical concerns (aforementioned limitations)
  - Nice standards for rounding, overflow, underflow
  - Problem: Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard

- Result: expressive but complex format for expressing fractional numbers.

27

27

What to Take Away From 2 FP Lectures

- You should *understand* the spec.
  - In other words, you should be able to convert from a bitwise representation to a numerical value, and from a numerical value to a bitwise representation
- We will go over examples, but please complete the practice problems
- It is a mechanical process, so practice will solidify the concepts

28

28

Floating Point Representation

Example:
$15213_{10} = (-1)^0 \times 1.1101101101101_2 \times 2^{13}$

- Numerical Form:

$$(-1)^s * M * 2^E$$

  - Sign bit s determines whether number is negative or positive
  - Significand (mantissa) M normally a fractional value in range [1.0,2.0).
  - Exponent E weights value by power of two

- Encoding
  - MSB is sign bit s
  - exp field *encodes* E (but is not equal to E)
  - frac field *encodes* M (but is not equal to M)

29

29

Precision options

- Single precision: 32 bits
  ≈ 7 decimal digits, $10^{\pm 38}$

- Double precision: 64 bits
  ≈ 16 decimal digits, $10^{\pm 308}$

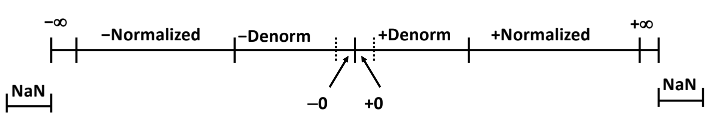
- Other formats: half precision, quad precision

30

30

3 "Cases" in Floating Point Format

- Special values: infinity, negative infinity, and NaN
- So-called "normalized" form
- So-called "denormalized" form



- We will go over each case individually, and revisit this number line at the end

31

31