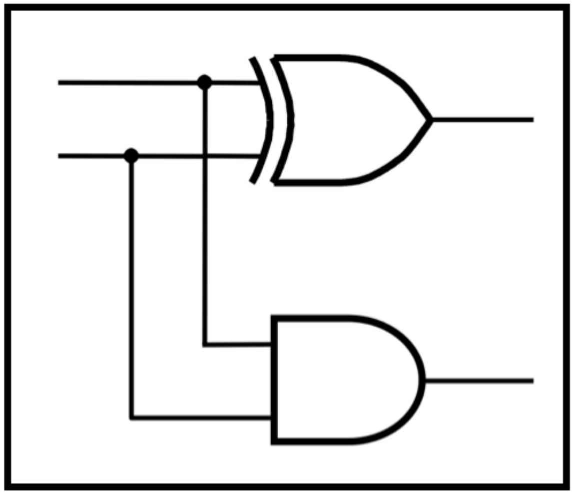


CprE 281: Digital Logic

Instructor: Alexander Stoytchev

http://www.ece.iastate.edu/~alexs/classes/

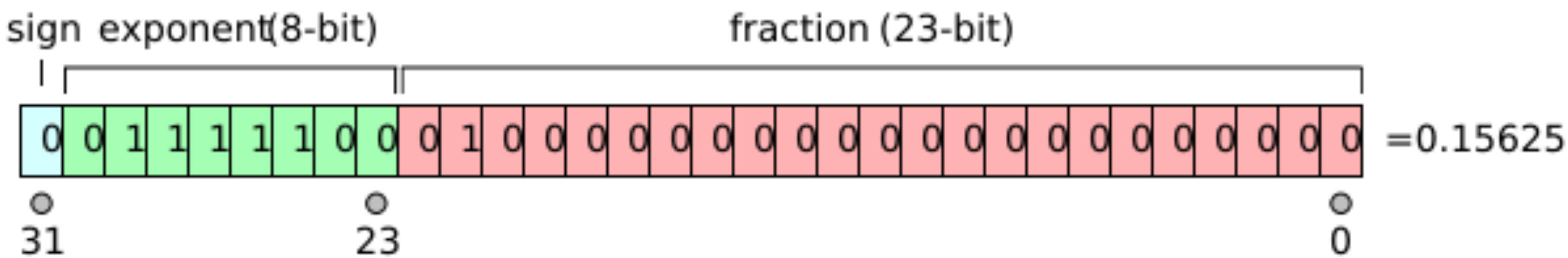
Floating Point Numbers

CprE 281: Digital Logic
Iowa State University, Ames, IA

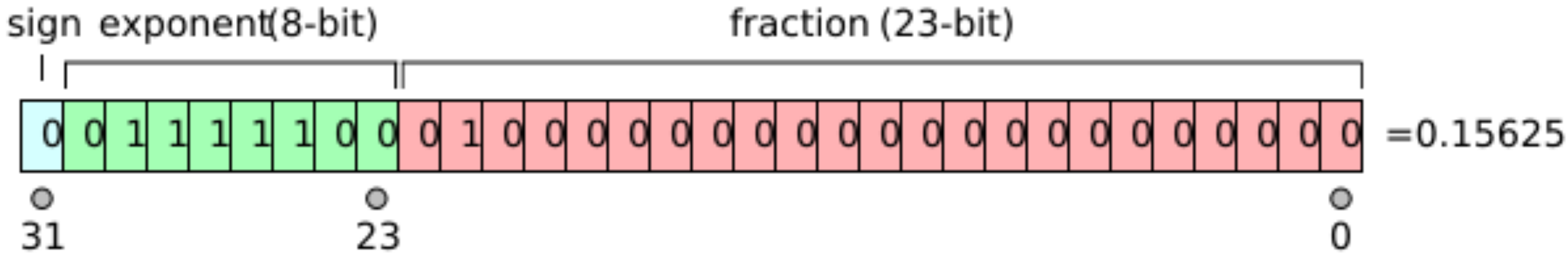
Administrative Stuff

- HW 6 is out
- It is due on Monday Oct 12 @ 4pm

The story with floats is more complicated IEEE 754-1985 Standard



[http://en.wikipedia.org/wiki/IEEE_754]



$$v = (-1)^{sign} \times 2^{exponent-exponent\ bias} \times 1.fraction$$

s = +1 (positive numbers and +0) when the sign bit is 0

s = −1 (negative numbers and −0) when the sign bit is 1

e = *exponent* − 127 (in other words the exponent is stored with 127 added to it, also called "biased with 127")

In the example shown above, the *sign* is zero so s is +1, the *exponent* is 124 so e is −3, and the significand m is 1.01 (in binary, which is 1.25 in decimal). The represented number is therefore +1.25 × 2^{-3}, which is +0.15625.

[http://en.wikipedia.org/wiki/IEEE_754]

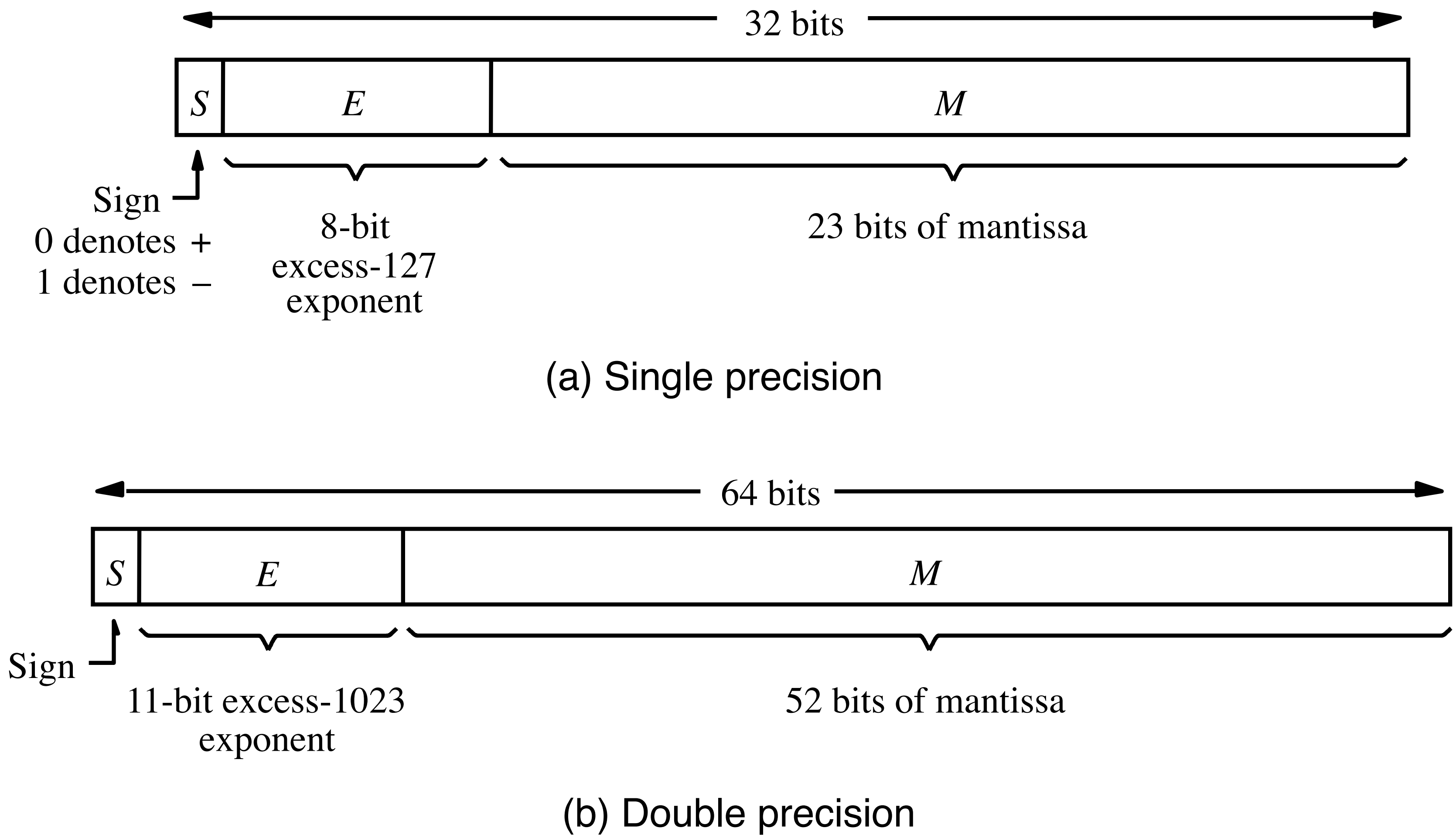


Figure 3.37. IEEE Standard floating-point formats.

On-line IEEE 754 Converter

- http://www.h-schmidt.net/FloatApplet/IEEE754.html

Conversion of fixed point numbers from decimal to binary

Convert $(214.45)_{10}$

$\frac{214}{2} = 107 + \frac{0}{2}$ → 0 LSB

$\frac{107}{2} = 53 + \frac{1}{2}$ → 1

$\frac{53}{2} = 26 + \frac{1}{2}$ → 1

$\frac{26}{2} = 13 + \frac{0}{2}$ → 0

$\frac{13}{2} = 6 + \frac{1}{2}$ → 1

$\frac{6}{2} = 3 + \frac{0}{2}$ → 0

$\frac{3}{2} = 1 + \frac{1}{2}$ → 1

$\frac{1}{2} = 0 + \frac{1}{2}$ → 1 MSB

$0.45 \times 2 = 0.90$ → 0 MSB

$0.90 \times 2 = 1.80$ → 1

$0.80 \times 2 = 1.60$ → 1

$0.60 \times 2 = 1.20$ → 1

$0.20 \times 2 = 0.40$ → 0

$0.40 \times 2 = 0.80$ → 0

$0.80 \times 2 = 1.60$ → 1 LSB

$(214.45)_{10} = (11010110.0111001\ldots)_2$

[Figure 3.44 from the textbook]

Memory Analogy

Address 0

Address 1

Address 2

Address 3

Address 4

Address 5

Address 6

Memory Analogy (32 bit architecture)

Address 0

Address 4

Address 8

Address 12

Address 16

Address 20

Address 24

Memory Analogy (32 bit architecture)

Address 0x00

Address 0x04

Address 0x08

Address 0x0C

Address 0x10

Address 0x14

Address 0x18

Hexadecimal

Address 0x0A

Address 0x0D

Storing a Double

Address 0x08

Address 0x0C

Storing 3.14

- **3.14 in binary IEEE-754 double precision (64 bits)**

sign	exponent	mantissa
0	10000000000	1001000111101011100001010001111010111000010100011111

- **In hexadecimal this is (hint: groups of four):**

0100 0000 0000 1001 0001 1110 1011 1000 0101 0001 1110 1011 1000 0101 0001 1111

4 0 0 9 1 E B 8 5 1 E B 8 5 1 F

Storing 3.14

- **So 3.14 in hexadecimal IEEE-754 is 40091EB851EB851F**
- **This is 64 bits.**
- **On a 32 bit architecture there are 2 ways to store this**

Small address:	**40091EB8**	**51EB851F**
Large address:	**51EB851F**	**40091EB8**
	Big-Endian	Little-Endian
Example CPUs:	Motorola 6800	Intel x86

Storing 3.14

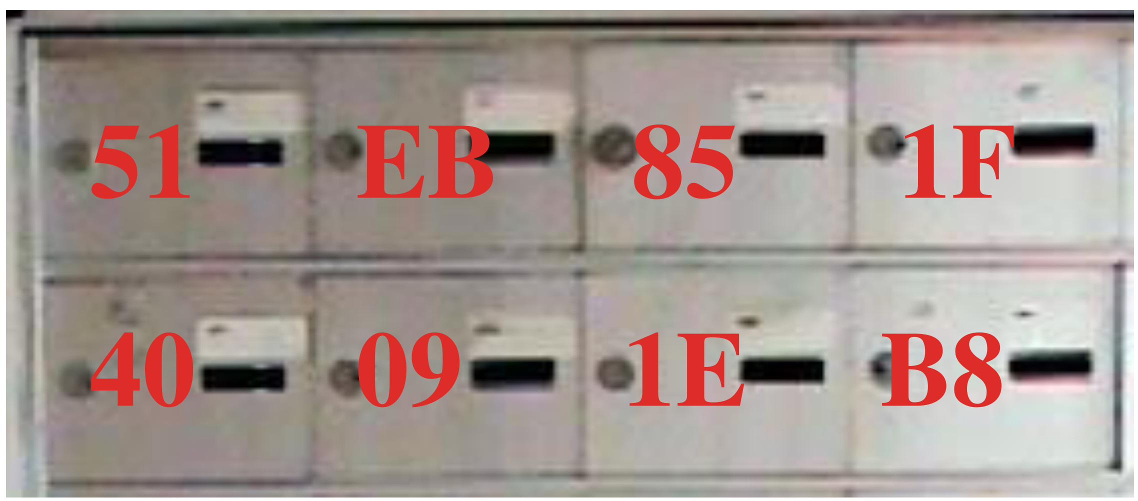
Address 0x08

Address 0x0C

Big-Endian

Address 0x08

Address 0x0C



Little-Endian

Storing 3.14 on a Little-Endian Machine
(these are the actual bits that are stored)

Address 0x08

Address 0x0C

Once again, 3.14 in IEEE-754 double precision is:

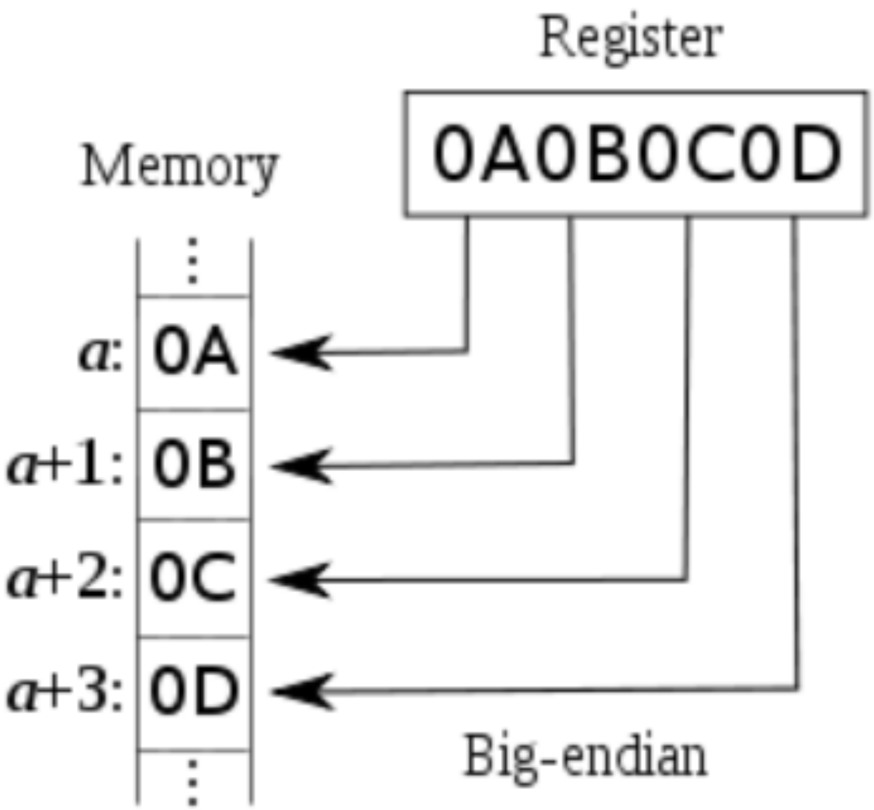
sign	exponent	mantissa
0	10000000000	1001000111101011100001010001111010111000010100011111

They are stored in binary
the hexadecimals are just for visualization

Address 0x08

Address 0x0C

Big-Endian



increasing addresses →

...	0x0A	0x0B	0x0C	0x0D	...

(8-bit architecture)

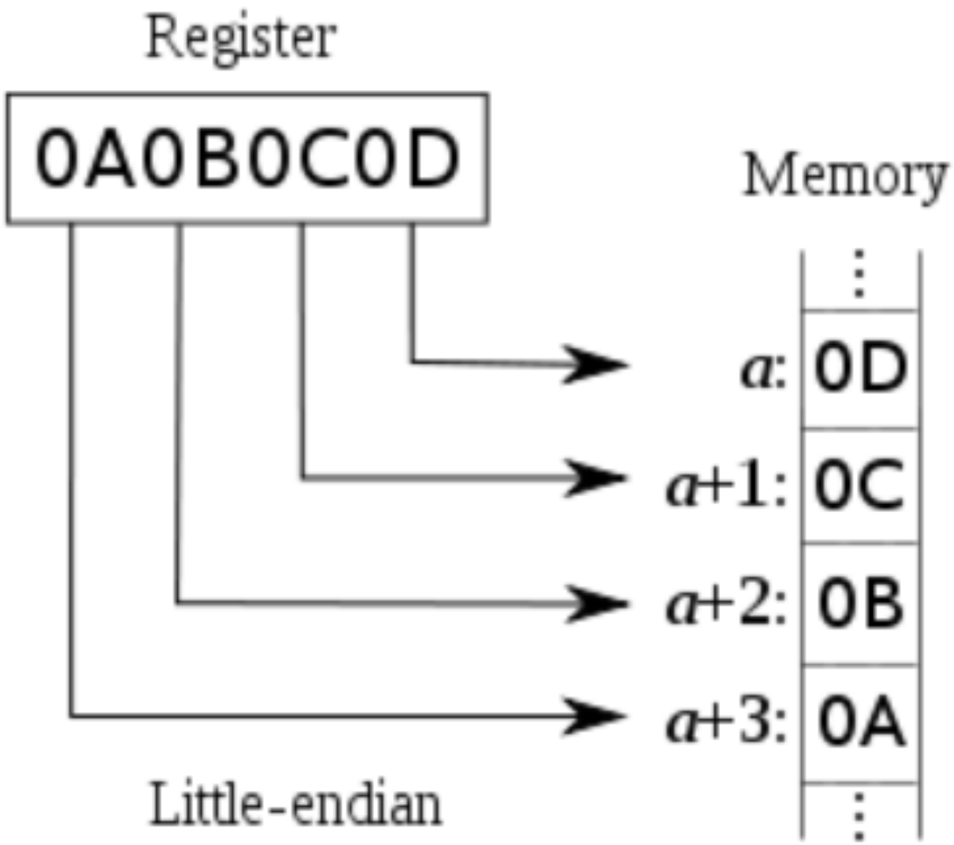
increasing addresses →

...	0x0A0B	0x0C0D	...

(16-bit architecture)

http://en.wikipedia.org/wiki/Endianness

Little Endian



increasing addresses →

...	0x0D	0x0C	0x0B	0x0A	...

(8-bit architecture)

increasing addresses →

...	0x0C0D	0x0A0B	...

(16-bit architecture)

http://en.wikipedia.org/wiki/Endianness

Big-Endian/Little-Endian analogy

[image fom http://www.simplylockers.co.uk/images/PLowLocker.gif]

Big-Endian/Little-Endian analogy

[image fom http://www.simplylockers.co.uk/images/PLowLocker.gif]

Big-Endian/Little-Endian analogy

[image fom http://www.simplylockers.co.uk/images/PLowLocker.gif]

What would be printed? (don't try this at home)

```
double pi = 3.14;
printf("%d",pi);
```

- **Result: 1374389535**

Why?

- **3.14 = 40091EB851EB851F (in double format)**
- **Stored on a little-endian 32-bit architecture**
  - **51EB851F (1374389535 in decimal)**
  - **40091EB8 (1074339512 in decimal)**

What would be printed? (don't try this at home)

```
double pi = 3.14;
printf("%d %d", pi);
```

- **Result: 1374389535 1074339512**

Why?

- **3.14 = 40091EB851EB851F (in double format)**
- **Stored on a little-endian 32-bit architecture**
  - **51EB851F (1374389535 in decimal)**
  - **40091EB8 (1074339512 in decimal)**

  - **The second %d uses the extra bytes of pi that were not printed by the first %d**

What would be printed?
(don't try this at home)

```
double a = 2.0;
printf("%d",a);
```

- **Result: 0**

Why?

- **2.0 = 40000000 00000000 (in hex IEEE double format)**
- **Stored on a little-endian 32-bit architecture**
  - **00000000 (0 in decimal)**
  - **40000000 (1073741824 in decimal)**

What would be printed?
(an even more advanced example)

```
int a[2];                   // defines an int array
a[0]=0;
a[1]=0;
scanf("%lf", &a[0]);    // read 64 bits into 32 bits
// The user enters 3.14
printf("%d %d", a[0], a[1]);
```

- **Result: 1374389535 1074339512**

Why?

- **3.14 = 40091EB851EB851F (in double format)**
- **Stored on a little-endian 32-bit architecture**
  - **51EB851F (1374389535 in decimal)**
  - **40091EB8 (1074339512 in decimal)**
- **The double 3.14 requires 64 bits which are stored in the two consecutive 32-bit integers named a[0] and a[1]**

Questions?

THE END