Floating Point (part III) and Machine Level Programming: Basics

CSCI 237: Computer Organization
10th Lecture, Friday, Sept. 27

Kelly Shaw

1

1

Administrative Details

- Lab #2 due Tuesday at 11pm
  - Any questions?
- Quiz on Glow due today at 2:30pm
- Read CSAPP 3.1-3.4
- Colloquium talk on Friday at 2:35pm in Wege
  - Sam Thomas, Brown University
  - Towards a Practical Secure Memory for Modern Deployments

2

2

Last Time: Floating Point (part II)

- IEEE FP standard (normalized and denormalized values)
- Tiny Floating Point Example
- Floating point in C
- Summary

3

3

Today: Floating Point and Machine-Level Programming: Basics

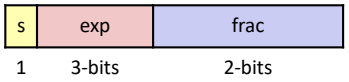
- Floating point in C
- Summary
- History of Intel processors and architectures

4

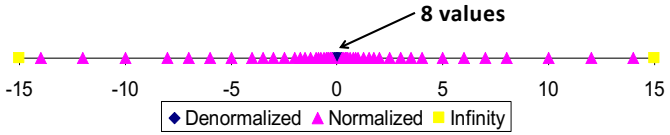
4

Distribution of Values

- 6-bit IEEE-like format
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is $2^{3-1}-1 = 3$



- Notice how the distribution gets denser toward zero.

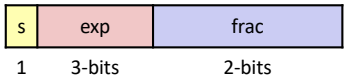


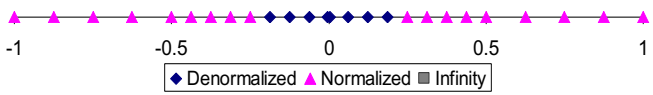
5

5

Distribution of Values (close-up view)

- 6-bit IEEE-like format
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is 3

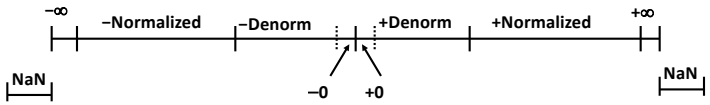




6

6

Visualization: Floating Point Encodings



7

7

Special Properties of the IEEE Encoding

- FP Zero Same as Integer Zero
  - All bits = 0
- Can (Almost) Use Unsigned Integer Comparison
  - Must first compare sign bits
  - Must consider −0 = 0
  - NaNs problematic
    - Will be greater than any other values
    - What should comparison yield?
  - Otherwise OK
    - Denorm vs. normalized
    - Normalized vs. infinity

8

8

Today: Floating Point and Machine-Level Programming: Basics

- Floating point in C
- Summary
- History of Intel processors and architectures

9

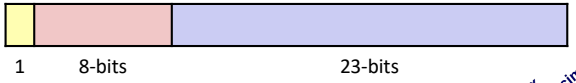
9

C float Decoding Example

$v = (-1)^s M 2^E$
$E = exp - bias$

float: 0xC0A00000

binary:



Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

E =

S =

M =

$v = (-1)^s M 2^E$

10

10

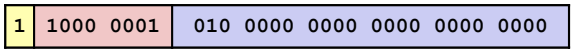
C float Decoding Example

$v = (-1)^s M 2^E$
$E = exp - bias$

float: 0xC0A00000

Bias = $2^{k-1} - 1 = 127$

binary: 1100 0000 1010 0000 0000 0000 0000 0000



Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

E =

S =

M =

$v = (-1)^s M 2^E$

11

11

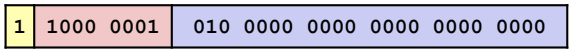
C float Decoding Example

$v = (-1)^s M 2^E$
$E = exp - bias$

float: 0xC0A00000

Bias = $2^{k-1} - 1 = 127$

binary: 1100 0000 1010 0000 0000 0000 0000 0000



Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

E = exp – bias = 129 – 127 = 2

S = 1 (negative number)

M = 1.010 0000 0000 0000 0000 0000

= 1 + 1/4 = 1.25

$v = (-1)^s M 2^E = (-1)^1 * 1.25 * 2^2 = -5$

12

12

Floating Point in C

- C Guarantees Two Levels
  - **`float`** single precision
  - **`double`** double precision
- Conversions/Casting
  - Casting between **`int`**, **`float`**, and **`double`** changes bit representation
  - **`double`**/**`float`** → **`int`**
    - Truncates fractional part
    - Like rounding toward zero
    - Not defined when out of range or NaN: Generally sets to TMin
  - **`int`** → **`double`**
    - Exact conversion, as long as **`int`** has ≤ 53 bit word size
  - **`int`** → **`float`**
    - Will round according to rounding mode

13

13

Floating Point Puzzles

- For each of the following C expressions, either:
  - Argue that it is true for all argument values
  - Explain why not true

```
int x = …;
float f = …;
double d = …;
```

Assume neither `d` nor `f` is NaN

- `x == (int)(float) x` ✗
- `x == (int)(double) x` ✓
- `f == (float)(double) f` ✓
- `d == (double)(float) d` ✗
- `f == -(-f);` ✓
- `2/3 == 2/3.0` ✗
- `d * d >= 0.0` ✓
- `(d+f)-d == f` ✗

14

14

Practice on Your Own

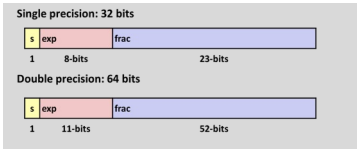
- What is the IEEE 754 single precision encoding for the decimal number -28.6125?
- What is the decimal value if the hexadecimal number 0xFA400000 is interpreted as 32 floating point encoding?
- If these hexadecimal number are interpreted as a 32 bit floating point numbers, are they denormalized, normalized, or special?
  - 0x80AF4203
  - 0x8102C00D
  - 0xFFAF0F10
  - 0xFACE4789

15

15

Summary

- IEEE Floating Point has clear mathematical properties
- Represents numbers of form M x 2^E
- One can reason about operations independent of implementation
  - As if computed with perfect precision and then rounded
- Not the same as real arithmetic
  - Violates associativity/distributivity
  - Makes life difficult for compilers & serious numerical applications programmers



16

16

Today: Floating Point and Machine-Level Programming: Basics

- Floating point in C
- Summary
- History of Intel processors and architectures

Major shift:
Data representation (Ch. 2)
to
Machine instructions and assembly programming (Ch. 3)

17

17

Intel x86 Processors

- Dominate laptop/desktop/server market (for now)
- Evolutionary design
  - Backwards compatible up until 8086, introduced in 1978
  - Added more features as time goes on
- Complex instruction set computer (CISC)
  - Many different instructions with many different formats
    - But, only small subset encountered with Linux programs
  - Hard to match performance of Reduced Instruction Set Computers (RISC)
  - But, Intel has done just that!
    - In terms of speed. Less so for low power.

18

18

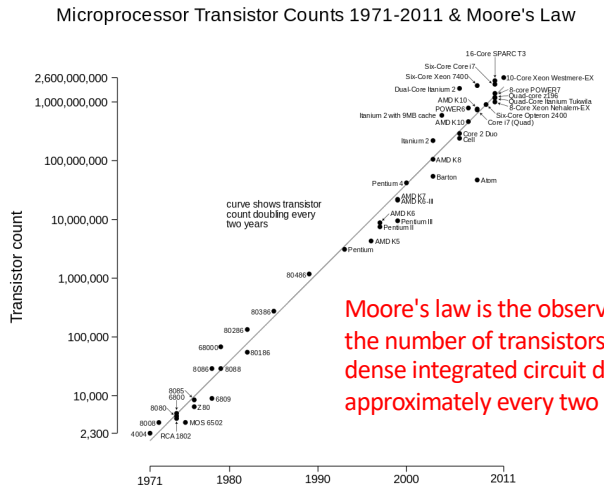
Intel x86 Evolution: Milestones

Name	*Date*	*Transistors*	*MHz*
8086	1978	29K	5-10
386	1985	275K	16-33
Pentium 4E	2004	125M	2800-3800
Core 2	2006	291M	1060-3500
Core i7	2008	731M	1700-3900

- First 16-bit Intel processor. Basis for IBM PC & DOS
- 1MB address space
- First 32 bit Intel processor , referred to as IA32
- Added "flat addressing", capable of running Unix
- First 64-bit Intel x86 processor, referred to as x86-64
- First multi-core Intel processor (core = CPU)
- Four cores

19

19



20

20

Intel x86 Processors, cont.

- Past Generations

		Process technology
1st Pentium Pro	1995	600 nm
1st Pentium III	1999	250 nm
1st Pentium 4	2000	180 nm
1st Core 2 Duo	2006	65 nm

- Recent Generations

1.	Nehalem	2008	45 nm
2.	Sandy Bridge	2011	32 nm
3.	Ivy Bridge	2012	22 nm
4.	Haswell	2013	22 nm
5.	Broadwell	2014	14 nm
6.	Skylake	2015	14 nm
7.	Kaby Lake	2016/7	14 nm
8.	Coffee Lake	2017	14nm
9.	C.L. Refreshed	2018	14nm
10.	Ice Lake	2019	10nm

Process technology dimension = width of narrowest wires (10 nm ≈ 100 atoms wide)

21

21

Practice Together

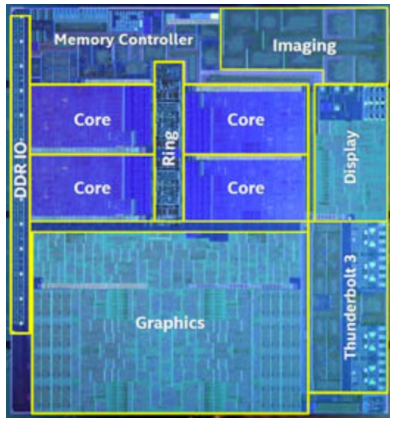
- Consider the hexadecimal value 0xC295D000. Assume this 32b binary number is a IEEE single precision floating point number. What is its decimal floating point value?

22

22

2019 State of the Art: Ice Lake

- Mobile Device: Core i7
  - 1-2.3 GHz
    - Turbo (3.8-4.1 GHz)
  - 9-28 W
  - Integrated Intel Gen 11 GPU
  - 2-4 CPUs



https://www.servethehome.com/intel-ice-lake-era-with-microarchitecture-and-gen11-gpu-improvements/

23

23

Our Coverage

- IA32
  - The traditional x86
- x86-64
  - The standard
  - `> gcc hello.c`
  - `> gcc -m64 hello.c`
- Presentation
  - Book covers x86-64
  - Web aside on IA32
  - We will only cover x86-64

24

24

Today: Floating Point and Machine-Level Programming: Basics

- Floating point in C
- Summary
- History of Intel processors and architectures
- Instruction Set Architecture (ISA)

25

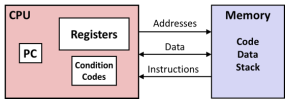
25

Levels of Abstraction

C programmer

C code

Assembly programmer



Computer Designer

Caches, clock freq, layout, …

Computer scientists love layers…

26

26

Variables: Source Code → Compiler Internal Representation → Executable

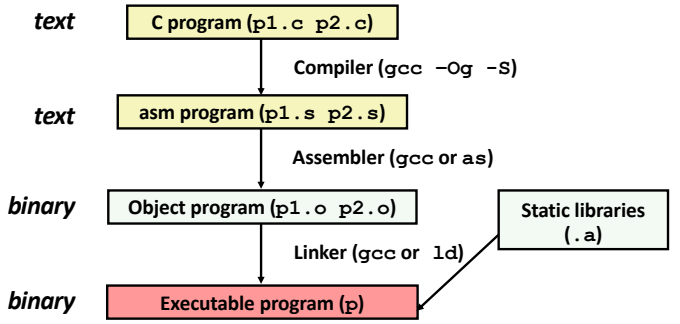
- Source code variable type
  - Determines how many bytes needed to represent variables
  - Determines how bits in bytes should be interpreted
- Compiler's internal representation
  - For each variable, keeps track of variable type and name
  - For each variable, determines # of bytes needed and determines memory locations
- Compiler generates executable that is all memory locations/registers and machine language instructions
  - Names and types are lost. Everything is just memory addresses and registers.
  - Information about size and types is implied by the size of bytes and what versions of instructions are used (different machine instructions for different variable types) by compiler in generating code

27

27

Turning C into Machine Code

- Code in files `p1.c p2.c`
- Compile with command: `gcc –Og p1.c p2.c -o p`
  - Use basic optimizations (`-Og`)
  - Put resulting binary in file `p`



28

28