COMP 3221

Microprocessors and Embedded Systems

Lectures 20 : Floating Point Number Representation – II

http://www.cse.unsw.edu.au/~cs3221

September, 2003

Saeid Nooshabadi

Saeid@unsw.edu.au

Overview

° **IEEE – 754 Standard**
- **Implied Hidden 1**
- **Representation for 0**

° **Decimal to Floating Point conversion, and vice versa**

° **Big Idea: Type is not associated with data**

° **ARM floating point instructions, registers**

Review: IEEE 754 Fl. Pt. Standard (#1/3)

° **Summary (single precision):**

31	30 – 23	22 – 0
S	Exponent	Significand
1 bit	8 bits	23 bits

° $(-1)^{S} \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

° **Double precision identical, except with exponent bias of 1023**

° **Hidden 1 (implied)**

Review: IEEE 754 Fl. Pt. Standard (#2/3)

° **Scientific notation in binary!**

$\pm 1.F \times 2^{\pm e}$

° **Sign Magnitude for the fixed part**

$(-1)^{s} \times 1.F \times 2^{\pm e}$

° **Hidden 1 of the significand**

$(-1)^{s} \times 1.\,fffffffffffffffffffffff \times 2^{\pm e}$

° **Excess notation for the exponent**

$(-1)^{s} \times 1.\mathrm{fffffffffffffffffffffff} \times 2^{exp - 127}$

° **Ordering the fields so integer compare works on FP**

Representing the Significand Fraction

- In normalized form, fraction is either:
  1.xxx xxxx xxxx xxxx xxxx xxx
  or
  0.000 0000 0000 0000 0000 000 (for Zero)

- Trick: If hardware automatically places 1 in front of binary point of normalized numbers, then get 1 more bit for the fraction, increasing accuracy "for free"

Hidden Bit 1.xxx xxxx xxxx xxxx xxxx xxx
becomes
(1).xxx xxxx xxxx xxxx xxxx xxx<u>x</u>

  - Comparison OK; "subtracting" 1 from both

How differentiate from Zero in Trick Format?

- .0000 ... 000 => . 0000 ... 000<u>0</u>
  .0000 ... 000 => . 0000 ... 000<u>0</u>

- Solution: Reserve most negative (value 0) exponent to be only used for Zero; rest are normalized so prepend an implied 1

- Convention is

0	–Big	00000

➔ 0.00000
and not 1.00000 X2^{-127}

0	>–Big	00000

➔ 1.00000 x 2^{Exp}
(– Big = 127 ➔ Sig=00000000 in biased notation

31	30 … 23	22 … 0
0	00000000	00000000000000000000000
1 bit	8 bits	23 bits

Understanding the Significand (#1/2)

- Method 1 (Fractions):
  - In decimal: $0.340_{10} \Rightarrow 340_{10}/1000_{10}$
    $\Rightarrow 34_{10}/100_{10}$
  - In binary: $0.110_2 \Rightarrow 110_2/1000_2 = 6_{10}/8_{10}$
    $\Rightarrow 11_2/100_2 = 3_{10}/4_{10}$
  - Advantage: less purely numerical, more thought oriented; this method usually helps people understand the meaning of the significand better

Understanding the Significand (#2/2)

- Method 2 (Place Values):
  - Convert from scientific notation
  - In decimal: $1.6732 = (1\times10^{0}) + (6\times10^{-1}) + (7\times10^{-2}) + (3\times10^{-3}) + (2\times10^{-4})$
  - In binary: $1.1001 = (1\times2^{0}) + (1\times2^{-1}) + (0\times2^{-2}) + (0\times2^{-3}) + (1\times2^{-4})$
  - Interpretation of value in each position extends beyond the decimal/binary point
  - Advantage: good for quickly calculating significand value; use this method for translating FP numbers

Example: Converting Binary FP to Decimal

0	0110 1000	101 0101 0100 0011 0100 0011

- Sign: 0 => positive
- Exponent:
  - $0110\ 1000_{two} = 104_{ten}$
  - Bias adjustment: 104 - 127 = -23
- Significand:
  - 1 + $1 \text{x} 2^{-1}$+ $0 \text{x} 2^{-2}$ + $1 \text{x} 2^{-3}$ + $0 \text{x} 2^{-4}$ + $1 \text{x} 2^{-5}$ +...
    $=1+2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-9}+2^{-14}+2^{-15}+2^{-17}+2^{-22}+2^{-23}$
    = 1.0 + 0.6661175
- Represents: $1.6661175_{ten} * 2^{-23} \sim 1.986 * 10^{-7}$

  (about 2/10,000,000)

Continuing Example: Binary to ???

0011 0100 0101 0101 0100 0011 0100 0011

- Convert 2's Comp. Binary to Integer:
  $2^{29}+2^{28}+2^{26}+2^{22}+2^{20}+2^{18}+2^{16}+2^{14}+2^{9}+2^{8}+2^{6}+ 2^{1}+2^{0}$
  $= 878{,}003{,}011_{ten}$
- Convert Binary to Instruction:

0011	01	0	0	0	1	0	1	0101	0100	0011 0100 0011
3	1	0	0	0	1	0	1	5	4	835

```
ldrblo r4, [r5], #-835
```

- Convert Binary to ASCII:

0011 0100	0101 0101	0100 0011	0100 0011
4	U	C	C

Big Idea: Type not associated with Data

0011 0100 0101 0101 0100 0011 0100 0011

- What does bit pattern mean:
  - $1.986 * 10^{-7}$? 878,003,011? "4UCC"?
    `ldrblo r4, [r5], #-835?`
- Data can be anything; operation of instruction that accesses operand determines its type!
  - Side-effect of stored program concept: instructions stored as numbers
- Power/danger of unrestricted addresses/ pointers: use ASCII as Fl. Pt., instructions as data, integers as instructions, ...
  (Leads to security holes in programs)

Converting Decimal to FP (#1/3)

- Simple Case: If denominator is an exponent of 2 (2, 4, 8, 16, etc.), then it's easy.
- Show IEEE 754 representation of -0.75
  - -0.75 = -3/4
  - $-11_{two}/100_{two}$ $-0.11_{two}/1.00_{two} = -0.11_{two}$
  - Normalized to $-1.1_{two} \text{ x } 2^{-1}$
  - $(-1)^S \text{ x } (1 + \text{Significand}) \text{ x } 2^{(\text{Exponent}-127)}$
  - $(-1)^1 \text{ x } (1 + .100\ 0000 \ldots 0000) \text{ x } 2^{(126-127)}$

1	0111 1110	100 0000 0000 0000 0000 0000

Converting Decimal to FP (#2/3)

° Not So Simple Case: If denominator is not an exponent of 2.

- Then we can't represent number precisely, but that's why we have so many bits in significand: for precision
- Once we have significand, normalizing a number to get the exponent is easy.
- So how do we get the Significand of a never ending number?

Converting Decimal to FP (#3/3)

° Fact: All rational numbers have a repeating pattern when written out in decimal (eg 1/7 =0.142857142857…)

° Fact: This still applies in binary as well (eg 1/111 =0.001001001…)

° To finish conversion:

- Write out binary number with repeating pattern.
- Cut it off after correct number of bits (different for single vs double precision).
- Derive Sign, Exponent and Significand fields.

Hairy Example (#1/2)

° How to represent 1/3 in IEEE 754?

° 1/3

$= 0.33333\ldots_{10}$

$= 0.25 + 0.0625 + 0.015625 + 0.00390625 + 0.0009765625 + \ldots$

$= 1/4 + 1/16 + 1/64 + 1/256 + 1/1024 + \ldots$

$= 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + 2^{-10} + \ldots$

$= 0.0101010101\ldots_2 * 2^0$

$= 1.0101010101\ldots_2 * 2^{-2}$ (Normalized)

Hairy Example (#2/2)

° $1/3 = 1.0101010101\ldots_2 * 2^{-2}$

° Sign: 0

° Exponent = $-2 + 127 = 125_{10} = 01111101_2$

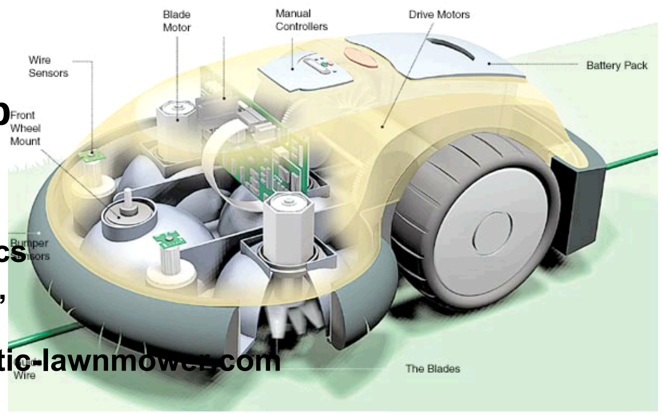
° Significand = 0101010101…

0	0111 1101	0101 0101 0101 0101 0101 010

What's this stuff good for? Mow Lawn?

- Robot lawn mower: "Robomow RL-800"
- Surround lawn, trees with perimeter wire
- Sense tall grass to spin blades faster: up to 5800 RPM
- Slow if senses object, stop if bumps
- US$700

Friendly Robotics of Even Yehuda, Israel, http://www.robotic-lawnmower.com



Representation for +/- Infinity

- In FP, divide by zero should produce +/- infinity, not overflow.
- Why?
  - OK to do further computations with infinity

    e.g., $1/(X/0) = (1/\infty) = 0$ is a valid Operation

    or, $X/0 > Y$ may be a valid comparison

    (Ask math prof.)
- IEEE 754 represents +/- infinity
  - Most positive exponent reserved for infinity
  - Significands all zeroes

Two Representation for 0!

- Represent 0?
  - exponent all zeroes
  - significand all zeroes too
  - What about sign?
  - `+0: 0 00000000 00000000000000000000000`
  - `-0: 1 00000000 00000000000000000000000`
- Why two zeroes?
  - Helps in some limit comparisons
  - Ask math prof.

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	+/- infinity
255	nonzero	???

- Professor Kahan had clever ideas; "Waste not, want not"
  - We will talk about Exp=0,255 & Significand !=0 later

Recall: arithmetic in scientific notation

Addition:

° $3.2 \times 10^4 + 2.3 \times 10^3$ => common exp

° $= 3.2 \times 10^4 + 0.23 \times 10^4$ => add

° $= 3.43 \times 10^4$ => normalize and round

° ~ 3.4×10^4

Multiplication:

° $3.2 \times 10^4 \times 2.3 \times 10^5$

° $= 3.2 \times 2.3 \times 10^9 = 7.36 \times 10^9 \sim 7.4 \times 10^9$

Basic Fl. Pt. Addition Algorithm

•Much more difficult than with integers

•For addition (or subtraction) of X to Y (X<Y):

(1) Compute $D = Exp_Y - Exp_X$ (align binary point)

(2) Right shift $(1+Sig_X)$ D bits=>$(1+Sig_X)*2^{(ExpX-ExpY)}$

(3) Compute $(1+Sig_X)*2^{(ExpX - ExpY)} + (1+Sig_Y)$

Normalize if necessary; continue until MS bit is 1

(4) Too small (e.g., 0.001xx...)
  left shift result, decrement result exponent

(4') Too big (e.g., 101.1xx…)
  right shift result, increment result exponent

(5) If result significand is 0, set exponent to 0

FP Addition/Subtraction Problems

° Problems in implementing FP add/sub:
  - If signs differ for add (or same for sub), what will be the sign of the result?

° Question: How do we integrate this into the integer arithmetic unit?

° Answer: We don't!

ARM's Floating Point Architecture (#1/4)

° Separate floating point instructions:
  - Single Precision:
    `fcmps, fadds, fsubs, fmuls, fdivs`
  - Double Precision:
    `fcmpd, faddd, fsubd, fmuld, fdivd`

° These instructions are far more complicated than their integer counterparts, so they can take much longer.

ARM's Floating Point Architecture (#2/4)

° **Problems:**

- It's inefficient to have different instructions take vastly differing amounts of time.
- Generally, a particular piece of data will not change from FP to int, or vice versa, within a program. So only one type of instruction will be used on it.
- Some programs do no floating point calculations
- It takes lots of hardware relative to integers to do Floating Point fast

ARM Floating Point Architecture (#3/4)

° ARM Solution: Make completely separate Co-processors that handles only IEEE-754 FP.

° Coprocessor 10 (for SP) & 11 (for DP):

- Actually a Single hardware used differently for Single Precision and Double Precision
- contains 32 32-bit registers: `s0 – s31`
- Arithmetic instructions use this register set
- separate load and store: `flds` and `fsts` ("Float loaD Single Coprocessor 10", "Float STore …")
- Double Precision: even/odd pair overlap one DP FP number: `s0/s1 = d0`, `s2/s3 = d1`, … , `s30/s31 =d15`
- separate double load and store: `fldd` and `fstd` ("Float loaD Double Coprocessor 11", "Float STore …")

ARM Floating Point Architecture (#4/4)

° ARM Solution:

- Processor: handles all the normal stuff
- Coprocessor 10 & 11: handles FP and only FP;
- more coprocessors?… Yes, later
- Today, cheap chips may leave out FP HW (Example: Chip on Lab's DSLMU Board)

° Instructions to move data between main processor and coprocessors:

- `fmsr (Sn = Rd)`, `fmrs (Rd = Sn)`, etc.

° Check ARM instruction reference manual on CD-ROM for many, many more FP operations.

"In Conclusion…"

° Floating Point numbers *approximate* values that we want to use.

° IEEE 754 Floating Point Standard is most widely accepted attempt to standardize interpretation of such numbers ($1T)

° New ARM registers(`s0-s31`), instruct.:

- Single Precision (32 bits, 2×10^{-38}… 2×10^{38}): `fcmps, fadds, fsubs, fmuls, fdivs`
- Double Precision (64 bits , 2×10^{-308}…2×10^{308}): `fcmpd, faddd, fsubd, fmuld, fdivd`

° Type is not associated with data, bits have no meaning unless given in context