CS47 - Lecture 21

Kaushik Patra
(kaushik.patra@sjsu.edu)

1

- Floating Point Representation
- Floating Point Challenges
- MIPS Floating Point instruction

Reference Books:

1) *Chapter 3, section 6 of 'Computer Organization & Design by Hennessy, Patterson*

Floating Point Representation ...

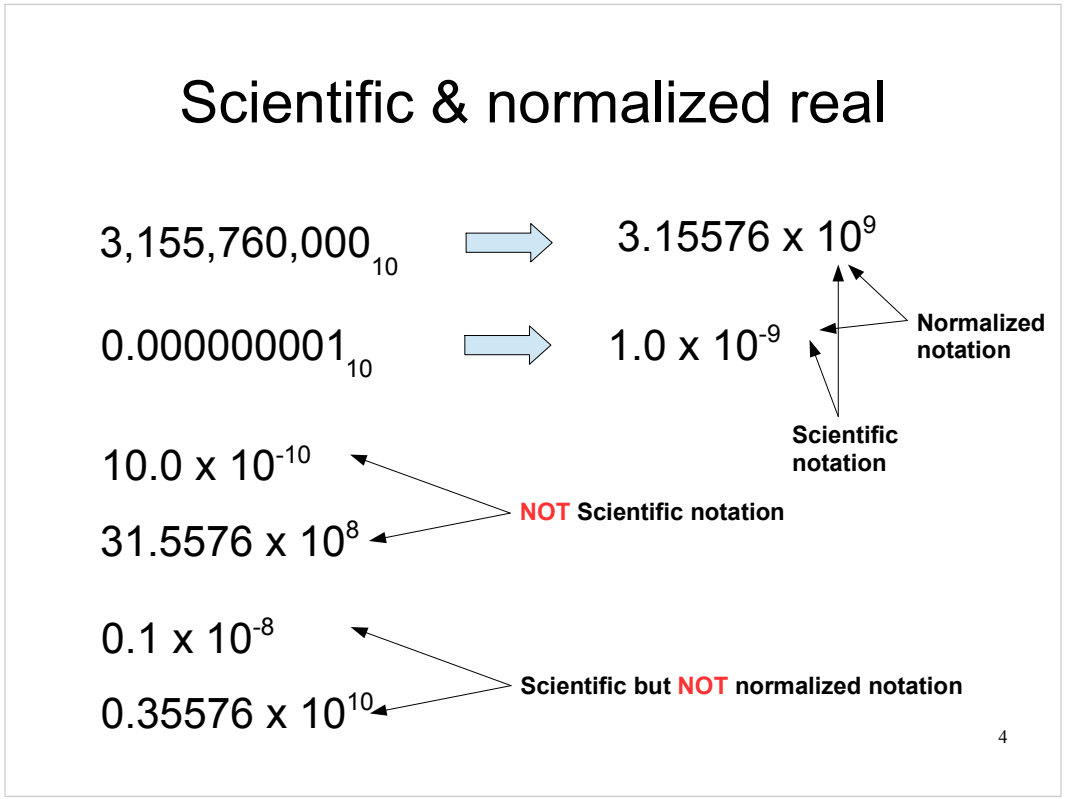
2

Real numbers

- Just the integer support is not sufficient
  - Everything about our world (and beyond) is 'real'.
- Example real numbers
  - Pi (π) : $3.14159265..._{10}$
  - Euler's number (e) : $2.71828..._{10}$
  - Gravitational acceleration (g) : 9.80665_{10} m/s^2

3

- Having just integer support may be sufficient for certain applications, but not all the application can be implement with just integers. Especially, application in scientific and financial computation is all about real numbers containing fractions. It important that computer supports mathematical operations involving real numbers.



- A scientific notation of a real number is expressed as a multiplication of a fraction with only one digit to the left of the decimal point and corresponding power (exponent) of 10. Re-constructing original number is simple because for a multiplication of a fraction number with nth power of 10, we just need to shift the decimal point to right by n if n is positive. Similarly, if n is negative, we need to shift the decimal point to left by n.

- A normalized notation is a scientific notation, where there no leading 0. To keep a number in normalized form, we need to adjust the decimal point position with adjustment of the power value of 10.

Real numbers in Binary

- Similar to the decimal normalized notation, numbers can be represented in binary.
  - $1.xxxxxxxx_2 \times 2^{yyyy}$.
  - Where 'xxxxxxxx' is the fraction bit pattern and 'yyyy' is the exponent bit pattern.
- Similar to decimal normalized notation, original value of a binary can be calculated by shifting the point to left or right by 'yyyy' amount.
  - It is called a 'floating point' in digital world.

5

- Please note that '2' has been chosen as the base of the power (exponent), since in binary system it is easier to calculate any binary number multiplied with 2^n. We just need to adjust the position of floating point similar to what we do for the positioning of decimal point for normalized notation in decimal system.

Real numbers in Binary

Power value in 3-bit 2s complement

$1.0110101*2^{011}$ ⇒ 1011.0101

$1.0110101*2^{101}$ ⇒ 0.00101101101

1011.0101 ⇒

$$2^3\times1+2^2\times0+2^1\times1+2^0\times1+2^{-1}\times0+2^{-2}\times1+2^{-3}\times0+2^{-4}\times1$$
$$=(8+0+2+1+0+0.25+0+0.0625)$$
$$=11.3125$$

0.00101101101 ⇒

$$2^{-1}\times0+2^{-2}\times0+2^{-3}\times1+2^{-4}\times0+2^{-5}\times1+2^{-6}\times1+$$
$$2^{-7}\times0+2^{-8}\times1+2^{-9}\times1+2^{-10}\times0+2^{-11}\times1$$
$$=0+0+0.125+0+0.03125+0.015625+0+0.00390625+$$
$$0.001953125+0+0.00048828125$$
$$=0.1630859375$$

6

- Assuming powers (exponents) are expressed in 2's complement (which is not) we can determine how much displacement (and which direction) floating point needs to get back original number from its normalized form. Process is similar to decimal point displacement – observe the magnitude and sign of the exponent and move the floating point accordingly.

- To convert a floating point number from its binary form two decimal form, convert the whole number part (i.e. left of the floating point) as integer and add it to converted fraction part. To convert the fraction part (i.e. the right of the floating point) convert using ($b * 2^{-n}$), where b is the bit value at the position n. n=1 at the very first bit position after the floating point and increases thereafter.

Format for floating point binary

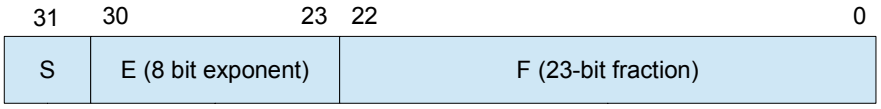
- There are only fixed & limited number of bits (e.g. 32-bit) to represent fraction and the exponent.
  - Increasing number of bits for fraction will increase the precision of represented numbers.
  - Increasing number of bits for exponent will increase the range of the represented numbers.
- Since the complete set of bit patterns will be divided into fraction and exponent, 2's complement method for representing -ve will no longer work.

7

- In reality a fraction value can be never ending in representation (think of Pi or Euler's number or recurring numbers). It depends on how many digits we are using to represent it, more the digit is closer the value is. Example 7/3 = 2.33333.... the fraction part can not be represented using finite number of digits. If we use total five digits (2.3333) and total 10 digits (2.333333333), the later is obviously is closure to the actual values. This same concept applies to binary representation of floating point to. More the number of bits for the fraction value, more precise is the representation.

- In decimal normalized notation, exponent value determines what is the range of the fraction value that can be represented. For example highest number represented by one digit exponent is order of 1thousand million. With a 2 digit exponent the highest number presented is 99 zeros followed be a 1 (100000...) Same applies to binary exponent – more the number of bits in exponent, more the range of number represented.

IEEE 754 floating point standard



$$(-1^{S})\times(1+F)\times 2^{(E-127)}$$

- Explicit sign bit
- Biased exponent with constant bias of 127
- Magnitude range
  - Smallest magnitude can be almost 2.0e-38
  - Highest magnitude can be almost 2.0e38
    - Beyond integer representation of 32-bit (almost 4.3e9 for unsigned)

8

- Initially different systems assumed different format for real number representation using bits. This led into great incompatibility between systems in terms of analysis results. During 1980, IEEE enforced standard format of floating point to resolve discrepancies of arithmetic analysis result from different computer systems. As of now, this is the standard supported by any computer system.

- For a negative number S=1 and for a positive number S=0. Therefore (-1^{S}) will either be -1 or 1 while computing the resultant value represented by the bit pattern.

- Since it is a normalized form, the fraction part assumes implicit 1 before the floating point. The F here represent the fraction part after the floating point. For example if F part is 1011 (rest of the bits are 0) then the fraction values in binary we are representing is 1.1011 (and all the 0s afterwards.)

- The format has significance in value comparison.
  - Explicit sign bit made it easier for comparing a result is less than or greater than zero.
  - Since the exponent is made biased, a pattern (00...0) represents the most negative and (11...1) represents the most positive exponent value.
  - Since the exponent field is placed before the fraction, this magnitude part can be compared with simple integer comparison.
    - A number with higher exponent will surely result in higher number with this arrangement.
    - The higher the value of the fraction part's bit pattern is, higher would be the integer values corresponding to that bit pattern.

IEEE 754 floating point standard

- Allows special values to represent unusual events instead of interrupting the program.
  - Event like divide by 0, software can set a bit pattern to represent –∞ or +∞ [E=255, F=0].
  - Events like invalid operation (like 0/0) can return a bit pattern representing NaN (not a number) [E=255, F=non-zero]

9

- Though the number represented by E=255 be a big one, IEEE 754 drops that big number support to represent infinity. Please note, this is not really a mathematical infinity but representing a value beyond scope of the 32-bit representation.

Floating Point Challenges ...

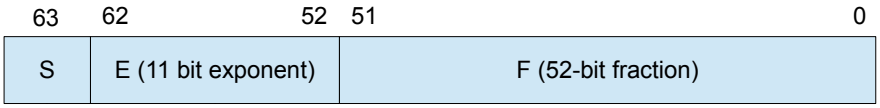
10

A good note can be found in the following:

https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

Many challenges ...

- Underflow error if result is too small to be represented by format.
  - To avoid frequent underflow error double precision (double in C/Java) format is recommended in IEEE754.
  - Biased exponent with constant bias of 1023
  - Magnitude range
    - Smallest magnitude can be almost 2.0e-308
    - Highest magnitude can be almost 2.0e308



11

- In this case, infinity and NaN bit pattern is also modified. An infinity bit pattern has E=2047 and F=0. A NaN bit patter has E=2047 and F=non-zero.

Many challenges ...

- Rounding error
  - Trying to pack infinite number of real number within finite number of bit pattern will always raise to approximation to certain extend.
    - A value 7/3 = 2.3333... can not be represented by any finite bit pattern of floating numbers (32, 64, 128 or beyond). It will always be an approximation towards 2.333... with certain error due to rounding the value.
    - The nearest representation of 0.10 in 24-bit system is 0.099999904632568359375, which is off by 0.000000095367431640625.
      - Not seems much, but this caused Patriot missile to fail to intercept Scud missile during Gulf ware 1991 causing death of 28 soldiers. After 100 hours of operation it caused 0.34s off from target, enough to cause havoc.

Many challenges ...

- Now we have -0 and +0. Digital circuit must take care of this signed zero in operation.
- Equality test of floating point may not be reliably done.
  - If (float_a == float_b) { … }
    - Not recommended.
  - Recommended way is following.
    - If (fabs(float_a – float_b) < float_small_threshold) { … }

MIPS Floating Point Support ...

MIPS instruction support

- Early days, floating point support in program used to be through software libraries (e.g. math.h/libmath).
  - Many of the common functions (like addition, subtraction, etc) are moved into hardware in a separate processor (called co-processor) in a system.
    - For MARS it is the Coproc 1
  - Depending on the system, co-processor can accommodate different sets of floating point math functions.
    - Simplest set contains add, sub, mul, div, comparison and branching.

MIPS instruction support

- Two varieties of math function – single and double precision (*.s and *.d).
  - Addition: add.s / add.d
  - Subtraction: sub.s / sub.d
  - Multiplication: mul.s / mul.d
  - Division: div.s / div.d
- Comparison instruction c.x.s and c.x.d where x
  - 'eq', 'neq', 'lt', 'le', 'gt', 'ge'
- Floating point branch, true (bclt) and false(bclf)
  - Test the status of the co-processor from c.x.(s|d) operation.

MIPS Co-Processor Registers

- 32 Separate co-processor registers
  - $f0 - $f31
  - Double precision instructions (*.d) can only access registers with even index.
    - Even-odd pair makes the double precision numbers (64-bits)
      - For example add.d $f1, $f2, $f4 is invalid instruction since the destination register index is odd ($f1).
      - Correct instruction would be add.d $f6, $f4, $f8 (where all the referred register indices are even).
      - ($f6, $f7) together makes the double precision register.
  - Separate load and store instructions
    - Load: lwc1 (e.g. lwc1 $f4, x($sp))
    - Store: swc1 (e.g. swc1 $f7, y($sp))

CS47 - Lecture 21

Kaushik Patra
(kaushik.patra@sjsu.edu)

18

- Floating Point Representation
- Floating Point Challenges
- MIPS Floating Point instruction

Reference Books:

1) *Chapter 3, section 6 of 'Computer Organization & Design by Hennessy, Patterson*