Number Formats

CSE 410 - Computer Systems

October 15, 2001

Readings and References

- Reading
  - Sections 4.1 through 4.4, 4.8 through page 280, 4.11, 4.12, Patterson and Hennessy, Computer Organization & Design

- Other References

Signed Numbers

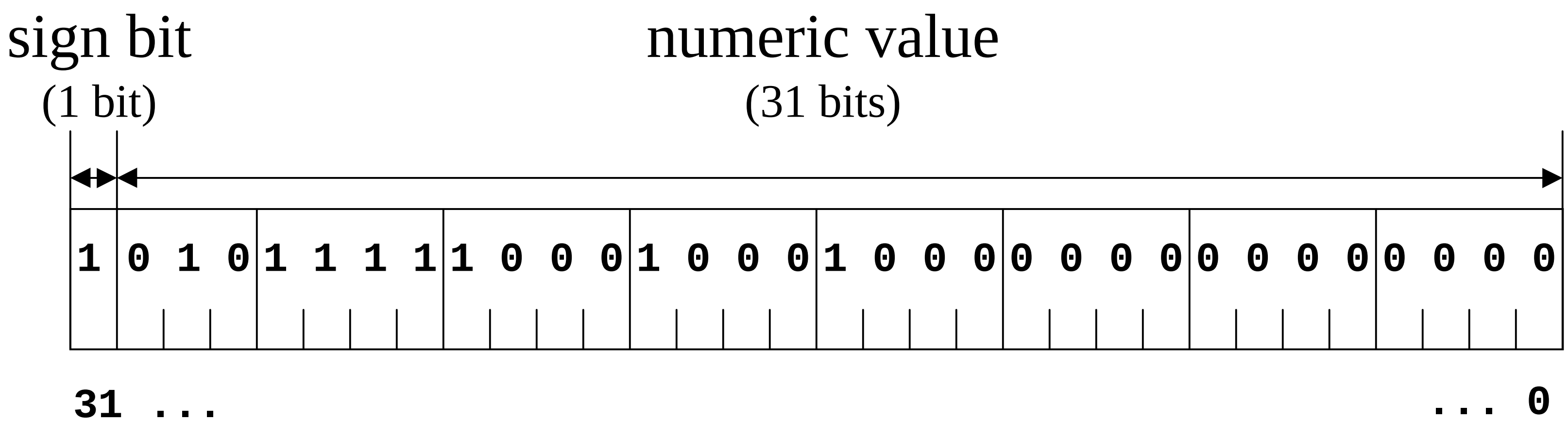
- We have already talked about unsigned binary numbers
  - each bit position represents a power of 2
  - range of values is 0 to 2^n-1
- How can we indicate negative values?
  - two states: positive or negative
  - a binary bit indicates one of two states: 0 or 1

  $\Rightarrow$ use one bit for the sign bit

Where is the sign bit?

- Could use an additional bit to indicate sign
  - each value would require 33 bits
  - would really foul up the hardware design
- Could use any bit in the 32-bit word
  - any bit but the left-most (high order) would complicate the hardware tremendously
- The high order bit (left-most) is the sign bit
  - remaining bits indicate the value

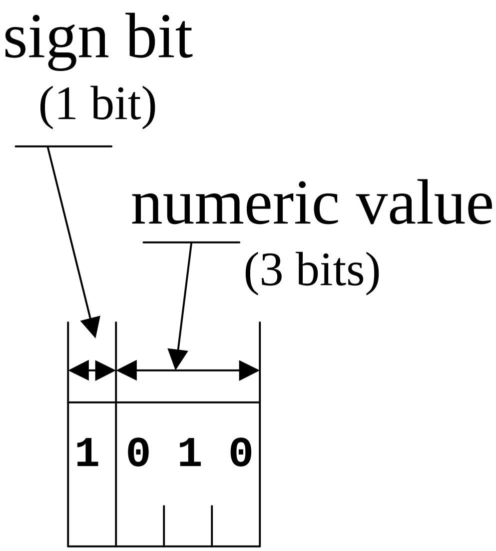
Format of 32-bit signed integer



- Bit 31 is the sign bit
  - 0 for positive numbers, 1 for negative numbers
  - aka most significant bit (msb), high order bit

Example: 4-bit signed numbers

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



Two's complement notation

- Note special arrangement of negative values
- One zero value, one extra negative value
- The representation is exactly what you get by doing a subtraction

```
Decimal        Binary

    1            0001
-   7        -   0111
----             ----
   -6            1010
```

Why “two’s” complement?

- In an n-bit word, negative x is represented by the value of 2^n-x
- 4-bit example

  $2^4 = 16$. What is the representation of -6?

```
Decimal        Binary

   16           10000
-   6        -   0110
----             ----
   10            1010
```

Negating a number

- Given x, how do we represent negative x?

```
          negative(x) = 2^n-x
and x+complement(x) = 2^n-1
so        negative(x) = 2^n-x = complement(x)+1
```

- The easy shortcut
  - write down the value in binary
  - complement all the bits
  - add 1

Example: the negation shortcut

```
 decimal 6 = 0110 = +6
complement = 1001
     add 1 = 1010 = -6

decimal -6 = 1010 = -6
complement = 0101
     add 1 = 0110 = +6
```

Signed and Unsigned Compares

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

```
add     $t0,$zero,-1

li      $t1,7

slt     $t2,$t0,$t1    # t2 = 1

sltu    $t3,$t0,$t1    # t3 = 0
```

Note: using 4-bit signed numbers in this example. The same relationships exist with 32-bit signed values.

Loading bytes

- Unsigned: `lbu $reg, a($reg)`
  - the byte is 0-extended into the register

0000 0000	0000 0000	0000 0000	xxxx xxxx

- Signed: `lb $reg, a($reg)`
  - bit 7 is extended through bit 31

0000 0000	0000 0000	0000 0000	0xxx xxxx

1111 1111	1111 1111	1111 1111	1xxx xxxx

Why Floating Point?

- The numbers we have talked about so far have all been integers in the range 0 to 4B or -2B to +2B
- What about numbers outside that range?
  - population of the planet: 6 billion+
- What about numbers that have a fractional part in addition to the integer part?
  - $\pi = 3.1415926535...$

Could use scaling to get fractions

- Assume that every numeric value in memory was scaled by a factor of 1000

  3000 => represents 3.000

  3010 => represents 3.010

- Problems
  - one scale factor for all numbers?
  - impossible to choose one "best" scale factor for all numbers that we might want to represent

A scale factor for each number

- This is the same as scientific notation
  - 6×10^9, 3.1415926535×10^0
- A floating point number contains two parts
  - mantissa (or significand): the value
  - exponent: the exponent of the scale factor
- Normalized form
  - a non-zero single digit before the decimal point

"Binary scientific notation"

- The computer only stores binary numbers
  - So we use powers of 2 rather than 10
  - Normalized numbers have a leading 1
- 6,000,000,000 = 6.0×10^9
  - $1.3969838619_{10} \times 2^{32}$
- $\pi \cong 3.14159265358979323846264338$3
  - $1.5707963267948966192313216916397 5 \times 2^1$

Storage format: fixed width fields

- How big can the exponent be?
  - what is the range it represents?
- How big can the mantissa be?
  - what are the values it represents?
- We have to select a storage format and allocate specific fields to various purposes
  - single precision: one 32-bit word
  - double precision: two 32-bit words

IEEE 754 Standard

- Chaos in the 70s and 80s as each system designer chose new formats and rules
- IEEE 754 standard
  - format of the fields
  - rounding: up, down, towards 0, nearest
  - exceptional values: ±infinity, NaN (not a number)
  - action to take on exceptional values

Floating Point Storage

- Single Precision
  - one word (32 bits)
- Double Precision
  - two words (64 bits)
  - the order of the words depends on endianness of the machine being used
- Defined by IEEE 754

Single Precision Format

s	**exponent**	**mantissa**
1	*8 bits*	*23 bits*
0	**1 0 0 0 0 0 1 1**	**1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

Double Precision Format

s	exponent	mantissa (high order bits)
1	*11 bits*	*20 bits*
0	1 0 0 0 0 0 1 1 1 1 1	0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

mantissa (low order bits)
32 bits
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Double Precision Mantissa Fields

- Sign bit
  - 1 bit sign for the value
- Mantissa
  - 52 bits for the value
  - by definition, the leading digit is always a 1
  - so we don’t need to actually store it
  - and we actually have 53 bits of information

Double Precision Exponent Field

- Field range
  - 11 bits: range $2^{11} = 2048$ possible values
- Special values
  - exponent = 2047 $\Rightarrow$ value=special (inf, NaN)
  - exponent = 0 $\Rightarrow$ value=0

Biased Notation

- Need exponent range - negative and positive
- If positive exponents are bigger numbers than the negative exponents, then floating point numbers can be sorted as integers
- Exponent is stored as (E+1023)
  - most positive exponent is +1023 (stored as 2046)
  - most negative exponent is -1022 (stored as 1)
  - this is <u>not</u> two's complement notation

Example: 6,174,015,488

- 6174015488

  $= 6.174015488 \times 10^9 = 1.4375_{10} \times 2^{32}$

- Exponent

  $= 32+1023 = 1055 = 41F_{16}$

- Mantissa

  $= .4375_{10} = .0111_2 = 7_{16}$

6,174,015,488

s	exponent	mantissa (high order bits)
1	*11 bits*	*20 bits*
0	1 0 0 0 0 0 1 1 1 1 1	0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

hex: 4 1 F 7 0 0 0 0

mantissa (low order bits)
32 bits
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Roundoff Error

- Adding a very small floating point number to a very large floating point number may not have any effect
  - any one number has only 53 significant bits
- Adding a number with a fractional part to another number over and over will probably never yield an exactly integer result
  - so don't use floating point loop indexes

Loss of precision

```
1101 0000 0000 0000.0000 0000 0000 0000   =  1.101₂ x 2^15
0000 0000 0000 0000.0000 0000 0000 1101   =  1.101₂ x 2^-13
```

- These are not unusual numbers

  53248 and 0.0001983642578125

- Very few bits of mantissa required
- But their sum requires a mantissa with at least 32 bits or there will lost significant bits