COMP2121: Microprocessors and Interfacing

Number Systems

http://www.cse.unsw.edu.au/~cs2121

Lecturer: Hui Wu

Term 2, 2019

1

1

Overview

- Positional notation
- Decimal, hexadecimal, octal and binary
- Converting decimal to any other
- One' complement
- Two's complement
- Two's complement overflow
- Signed and unsigned comparisons
- Strings
- Sign extension
- IEEE Floating Point Number Representation
- Floating Point Number Operations

2

2

Numbers: positional notation

- Number Base B => B symbols per digit:
  - Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Base 2 (Binary): 0, 1
- Number representation:
  - $(a_n a_{n-1} \ldots a_1 . b_1 \ldots b_{m-1} b_m)_B$ is a number of base (radix) B
    - n digits in the integer part and m digits in the fractional part.
    - The base B can be omitted if B=10.
  - value = $a_n \times B^{n-1} + a_{n-1} \times B^{n-2} + \ldots + a_2 \times B^1 + a_1 \times B^0$

    $+ b_1 \times B^{-1} + b_2 \times B^{-2} + \ldots + b_{m-1} \times B^{-(m-1)} + b_m \times B^{-m}$

3

3

Typical Number Systems (1/2)

- Binary system
  - Base 2
  - Digits (bits): 0,1
  - $(11010.101)_2 =$

    $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 0 \times 1 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

    $= 26.625$
- Octal system:
  - Base 8.
  - Digits: 0, 1, 2, 3, 4, 5, 6, 7
  - $(605.24)_8 = 6 \times 8^2 + 0 \times 8^1 + 5 \times 8^0 + 2 \times 8^{-1} + 4 \times 8^{-2} = 389.3125$

4

4

Typical Number Systems (2/2)

- Hexadecimal system
  - Base 16
  - Digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
    - A → 10
    - B → 11
    - C → 12
    - D → 13
    - E → 14
    - F → 15
  - $(8F0D.2C)_{16} = (8\times16^3) + (15\times16^2) + (0\times16^1) + (13\times16^0) + (2\times16^{-1}) + (C\times16^{-2}) = 36621.171875$

5

5

Decimal vs. Hexadecimal vs. Binary

- Examples:
- 1010 1100 0101 (binary)
  = ? (hex)
- 10111 (binary)
  = 0001 0111 (binary)
  = ? (hex)
- 3F9(hex)
  = ? (binary)

00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

6

6

Hex to Binary Conversion

- HEX is a more compact representation of Binary!
- Each hex digit represents 16 decimal values.
- Four binary digits represent 16 decimal values.
- Therefore, each hex digit can replace four binary digits (bits).
- Example:

$(3 \quad B \quad 9 \quad A \quad C \quad A \quad 0 \quad 0)_{16}$

$= (0011\ 1011\ 1001\ 1010\ 1100\ 1010\ 0000\ 0000)_2$

7

7

Octal to Binary Conversion

- Each octal digit represents 8 decimal values.
- Three binary digits represent 8 decimal values.
- Therefore, each octal digit can replace three binary digits (bits).
- Example:

$(3 \quad 7 \quad 1 \quad 2 \quad 4 \quad 5 \quad 0 \quad 1)_8$

$= (011\ 111\ 001\ 010\ 100\ 101\ 000\ 001)_2$

8

8

Converting from Decimal to Any Other (1/7)

- Use division method if a decimal number is an integer.
- Let D be a decimal integer such that

  $D = (a_n a_{n-1} \ldots a_1)_B$

  $= a_n B^{n-1} + a_{n-1} B^{n-2} + \ldots + a_2 B^1 + a_1 B^0$
- Notice that

  $a_1 = D\%B$

  $a_2 = (D/B)\%B$

  …

  In general, $a_i = ((D/B^{i-1})\%B \quad (i=1, 2, \ldots n)$

  Where / is the division operator and % the modulus operator as in C.

9

9

Converting from Decimal to Any Other (2/7)

The conversion procedure is shown in C as follows:

```
D2B-Integer-Converter(int B, long int D)
 { int i, A[];
   long int x;
   i=0;
   x=D;
   while (x!=0)
     { A[i] =x%B ;
       x=x/B ;
       i++;}
 }
```

10

10

Converting from Decimal to Any Other (3/7)

Example 1: Convert 5630 to a hex number.

Division	Quotient	Remainder	Remainder in hex
5630/16	352	14	E
351/16	21	15	F
21/16	1	5	5
1/16	0	1	1

Therefore, $5630=(15FE)_{16}$

11

11

Converting from Decimal to Any Other (4/7)

Example 2: Convert 138 to a binary number.

Division	Quotient	Remainder
138/2	69	0
69/2	34	1
34/2	17	0
17/2	8	1
8/2	4	0
4/2	2	0
2/2	1	0
1/2	0	1

Therefore, $138=(10001010)_2$

12

12

Converting from Decimal to Any Other (5/7)

- Use multiplication method if the decimal number is a fractional number.
- Let D be a fractional decimal number such that

$D=(0.b_1b_2 \dots b_{m-1}b_m)_B$

$=b_1B^{-1} + b_2B^{-2} + \dots + b_{m-1}B^{-(m-1)} + b_mB^{-m}$

- Notice that

$b_1 = \text{floor}(D\times B)$

$b_2 = \text{floor}(\text{frac}(D\times B)\times B)$

…

In general, $b_i = \text{floor}(\text{frac}(D\times B^{i-1})\times B)$ $(i=1, 2, \dots m)$

Where floor(x) is the integer part of x and frac(x) is the fractional part of x.

13

13

Converting from Decimal to Any Other (6/7)

The conversion procedure is shown in C as follows:

```
D2B-Fractional-Converter(int B, double D)
 { int i, A[];
    double x;
    i=0;
    x=D;
    while (x!=0)
      { A[i] = floor(x*B) ;
         x = x*B-A[i] ;
         i++ ;}
  }
```

14

14

Converting from Decimal to Any Other (7/7)

Example 3: Convert 0.6875 to a binary number.

Multiplication	Integer	Fractional
$0.6875 \times 2 = 1.375$	1	0.375
$0.375 \times 2 = 0.75$	0	0.75
$0.75 \times 2 = 1.5$	1	0.5
$0.5 \times 2 = 1.0$	1	0.0

Therefore, $0.6875=(0.1011)_2$

15

15

Which Base Should We Use?

- Decimal: Great for humans; most arithmetic is done with these.
- Binary: This is what computers use, so get used to them. Become familiar with how to do basic arithmetic with them (+,-,*,/).
- Hex: Terrible for arithmetic; but if we are looking at long strings of binary numbers, it's much easier to convert them to hex in order to look at four bits at a time.

16

16

How Do We Tell the Difference?

- When dealing with AVR microcontrollers:
  - Hex numbers are preceded with "\$" or "0x"
    - -\$10 == 0x10 == 10_{16} == 16_{10}
  - Binary numbers are preceded with "0b"
  - Octal numbers are preceded with "0" (zero)
  - Everything else by default is Decimal

17

17

Inside the Computer

- To a computer, numbers are always in binary; all that matters is how they are printed out: binary, decimal, hex, etc.
- As a result, it doesn't matter what base a number in C is in...
  - 32_{10} == 0x20 == 100000_2
- Only the value of the number matters.

18

18

Bits Can Represent Everything

- Characters?
  - 26 letter => 5 bits
  - upper/lower case + punctuation
    => 7 bits (in 8) (ASCII)
  - Rest of the world's languages => 16 bits (unicode)
- Logical values?
  - 0 -> False, 1 => True
- Colors ?
- Locations / addresses? commands?
- But N bits => only 2^N things

19

19

What If Too Big?

- Numbers really have an infinite number of digits
  - with almost all being zero except for a few of the rightmost digits: e.g: 0000000 … 000098 == 98
  - Just don't normally show leading zeros
- Computers have fixed number of digits
  - Adding two n-bit numbers may produce an (n+1)-bit result.
  - Since registers' length (8 bits on AVR) is fixed, this is a problem.
  - If the result of add (or any other arithmetic operation), cannot be represented by a register, overflow is said to have occurred

20

20

An Overflow Example

- Example (using 4-bit numbers):

+15	1111
+3	0011
+18	10010

  - But we don't have room for 5-bit solution, so the solution would be 0010, which is +2, which is wrong.

21

21

How To Handle Overflow?

- Some languages detect overflow (Ada), some don't (C and JAVA)
- AVR has N, Z, C and V flags to keep track of overflow
  - Will cover details later

22

22

Comparison

- How do you tell if X > Y ?
- See if X - Y > 0

23

23

How to Represent Negative Numbers?

- So far, unsigned numbers
- Obvious solution: define leftmost bit to be sign!
  - 0 => +, 1 => -
  - Rest of bits can be numerical value of number
- Representation called sign and magnitude
- On AVR $+1_{ten}$ would be: 0000 0001
- And -1_{ten} in sign and magnitude would be: 1000 0001

24

24

Shortcomings of Sign and Magnitude?

- Arithmetic circuit more complicated
  - Special steps depending whether signs are the same or not
- Also, two zeros.
  - $0x00 = +0_{ten}$
  - $0x80 = -0_{ten}$ (assuming 8 bit integers).
  - What would it mean for programming?
- Sign and magnitude abandoned because another solution was better

25

25

Another Try: Complement the Bits

- Examples: $7_{10} = 00000111_2$ $-7_{10} = 11111000_2$
- Called one's Complement.
- The one's complement of an integer X is
  2^p-X-1, where p is the number of integer bits.

Questions:

- What is -00000000_2 ?
- How many positive numbers in N bits?
- How many negative numbers in N bits?

26

26

Shortcomings of Ones Complement?

- Arithmetic not too hard

- Still two zeros
  - $0x00 = +0_{ten}$
  - $0xFF = -0_{ten}$ (assuming 8 bit integers).
- One's complement was eventually abandoned because another solution is better

27

27

Two's Complement

- The two's complement of an integer X is

  2^p-X,

  where p is the number of integer bits
- Bit p is the "sign" bit. Negative number if it is 1; positive number otherwise.
- Examples:
  - $7_{10} = 00000111_2$ $-1_{10} = 11111111_2$
  - $-2_{10} = 11111110_2$ $-7_{10} = 11111001_2$

28

28

Two's Complement Formula

- Given a two's complement representation $d_p d_{p-1} \ldots d_1 d_0$, its value is $d_p\,(-2^p) + d_{p-1}\,2^{p-1} + \ldots + d_1\,2^1 + d_0\,2^0$
- Example:
  - Two's complement representation 11110011

$= 1\times(-2^7) + 1\times 2^6 + 1\times 2^5 + 1\times 2^4 + 0\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0$

$= 00001101_2$

29

29

Two's Complement's Arithmetic Examples

- Example 1: 20 – 4 = 16
- Assume 8 bit architecture.

20 – 4 = 20 + (–4)

$= 0001\ 0100_{two} - 0000\ 0100_{two}$

$= \ \ 0001\ 0100_{two}$

$+\ 1111\ 1100_{two}$

$= 1\,0001\ 0000_{two}$

Carry   Most significant bit (msb)   No overflow.

30

30

Two's Complement's Arithmetic Examples

- Example 2: $-127 - 2 = -129$?
- $-127 - 2$

$= -0111\ 1111_{two} - 0000\ 0010_{two}$

$= \quad 1000\ 0001_{two}$

$\;\; + 1111\ 1110_{two}$

$= \quad 10111\ 1111_{two}$

Carry msb Overflow

31

31

Two's Complement's Arithmetic Examples

- Example 3: $127 + 2 = 129$?
- $127 + 2$

$= \quad 0111\ 1111_{two} + 0000\ 0010_{two}$

$= \quad 0111\ 1111_{two}$

$\;\; + 0000\ 0010_{two}$

$= \quad 1000\ 0001_{two}$

msb Overflow

32

32

When Overflow Occurs?

The ‘two’s complement overflow’ occurs when:

• both the msb’s being added are 0 and the msb of the result is 1

• both the msb’s being added are 1 and the msb of the result is 0

33

How AVR Computes Overflow Flag V?

Instruction: *add Rd, R*

V=Rd7•Rr7• NOT(R7)+NOT(Rd7)•NOT(Rr7)•R7

NOT : negation

+ : bit-wise or

• : bit-wise and

34

Signed vs. Unsigned Numbers

- C declaration int
  - Declares a signed number
  - Uses two's complement
- C declaration unsigned int
  - Declares a unsigned number
  - Treats 32-bit number as unsigned integer, so most significant bit is part of the number, not a sign bit
- NOTE:
  - Hardware does all arithmetic in 2's complement.
  - It is up to programmer to interpret numbers as signed or unsigned.

35

35

Signed and Unsigned Numbers in AVR(1/2)

- AVR microcontrollers support only 8 bit signed and unsigned integers.
- Multi-byte signed and unsigned integers can be implemented by software.
- Question: How to compute

$$10001110\ 01110000\ 11100011\ 00101010_{two}$$
$$+\ 01110000\ 11001000\ 10001100\ 01110001_{two}$$

on AVR?

36

36

Signed and Unsigned Numbers in AVR (2/2)

° Solution: Four-byte integer addition can be done by using four one-byte integer additions taking carries into account (lowest bytes are added first).

10001110	01110000	11100011	00101010
+ 01110000	+ 11001000	+ 10001100	+ 01110001
= 11111110	100111000	101101111	010011011

Carry bits

The result is 11111111 00111001 01101111 10011011_{two}

37

37

Signed v. Unsigned Comparison

- X = 1111 1100_{two}
- Y = 0000 0010_{two}

- Is X > Y?
  - unsigned: YES
  - signed: NO

38

38

Signed v. Unsigned Comparison (Hardware Help)

- $X = 1111\ 1100_{two}$
- $Y = 0000\ 0010_{two}$
- Is X > Y? Do the Subtraction X – Y and check result

$X - Y = 1111\ 1100_{two} - 0000\ 0010_{two}$

$= 1111\ 1100_{two}$

$+ 1111\ 1110_{two}$

$= 11111\ 1010_{two}$

Hardware needs to keep

- a special bit (S flag in AVR) which indicates the result of signed comparison, and
- a special bit (C flag in AVR) which indicates the result of unsigned comparison. 39

39

Signed v. Unsigned Comparison (Hardware Help)

- $X = 1111\ 1100_{two}$
- $Y = 0000\ 0010_{two}$
- Is X > Y? Do the Subtraction X – Y and check result

$X - Y = 1111\ 1100_{two} - 0000\ 0010_{two}$

$= 1111\ 1100_{two}$

$+ 1111\ 1110_{two}$

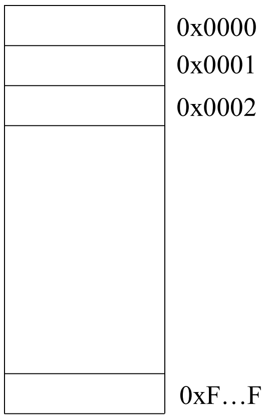
$= 11111\ 1010_{two}$

Hardware needs to keep

- a special bit (S flag in AVR) which indicates the result of signed comparison, and
- a special bit (C flag in AVR) which indicates the result of unsigned comparison. 40

40

Numbers Are Stored at Addresses



- Memory is a place to store bits
- A word is a fixed number of bits (eg, 16 in AVR assembler) at an address
- Addresses have fixed number of bits
- Addresses are naturally represented as unsigned numbers
- How multi-byte numbers are stored in memory is determined by the endianness.
- On AVR, programmers choose the endianess.

41

Beyond Integers (Characters)

- 8-bit bytes represent characters, nearly every computer uses American Standard Code for Information Interchange (ASCII)

No.	char	No.	char	No.	char	No.	char	No.	char	No.	char
32		48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
...		...		...		...		...		...	
47	/	63	?	79	O	95	_	111	o	127	DEL

- Uppercase + 32 = Lowercase (e.g, B+32=b)
- tab=9, carriage return=13, backspace=8, Null=0

42

Strings

- Characters normally combined into strings, which have variable length
  - e.g., "Cal", "M.A.D", "COMP3221"
- How to represent a variable length string?
  1) 1st position of string reserved for length of string (Pascal)
  2) an accompanying variable has the length of string (as in a structure)
  3) last position of string is indicated by a character used to mark end of string (C)
- C uses 0 (Null in ASCII) to mark the end of a string

43

43

Example String

- How many bytes to represent string "Popa"?
- What are values of the bytes for "Popa"?

No.	char	No.	char	No.	char	No.	char	No.	char	No.	char
32		48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
...		...		...		...		...		...	
47	/	63	?	79	O	95	_	111	o	127	DEL

- 80, 111, 112, 97, 0 DEC
- 50, 6F, 70, 61, 0 HEX

44

44

Strings in C: Example

° String simply an array of char

```
void strcpy (char x[],char y[])
  {
   int i=0;  /* declare and
initialize i*/
   while ((x[i]=y[i])!='\0') /* 0 */
   i=i+1;  /* copy and test byte */
  }
```

45

45

String in AVR Assembly Language

- .db “Hello\n” ; This is equivalent to

  .db ‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\n’

- What does the following instruction do?

  ldi r4, ‘1’

46

46

Sign Extension (1/4)

- Remember that negative numbers in computers are represented in 2's complements.
- How to extend a binary number of m bits in 2's complement to an equivalent binary number of m+n bits?

Example 1: $x = (0100)_2 = 4$

Since x is a positive number,

$x = (0000\ 0100)_2$

$= (0000\ 0000\ 0100)_2$

In general, if a number is positive, add n 0's to its left. This procedure is called sign extension.

47

47

Sign Extension (2/4)

- **Example 2:** $x = (1100)_2 = -4$

Since x is negative,

$x = (1111\ 1100)_2$

$= (1111\ 1111\ 1100)_2$

In general, if a number is negative, add n 1's to its left. This procedure is called sign extension.

48

48

Sign Extension (3/4)

- How to add two binary numbers of different lengths?
  - Sign-extend the shorter number such that it has the same length as the longer number, and then add both numbers.

Example 3: $x = (11010100)_2 = -44$,

$y = (0100)_2 = 4$

$x+y=?$

Since y is positive, $y=(00000100)_2$.

$x+y = (11010100)_2 + (00000100)_2$

$= (11011000)_2 = -40$

49

49

Sign Extension (4/4)

Example 4: $x = (11010100)_2 = -44$,

$y = (1100)_2 = -4$

$x+y=?$

Since y is negative, $y=(11111100)_2$.

$x+y = (11010100)_2 + (11111100)_2$

$= (11010000)_2 = -48$

50

50

Scientific Notation

Integer → **6.02 x 10^{23}** ← Exponent

Decimal point   Radix (base)

- Normalized form: no leadings 0
  (exactly one non-zero digit to the left of decimal point)
- Alternatives to representing 1/1,000,000,000
  - Normalized: $1.0 * 10^{-9}$
  - Not normalized: $0.1 * 10^{-8}$,$10.0 * 10^{-10}$

How to represent 0 in Normalized form?

51

Scientific Notation for Binary Numbers

Integer → **1.01 x 2^{-12}** ← Exponent

Binary point   Radix (base)

- Computer arithmetic that supports it is called floating point, because it represents numbers where binary point is not fixed, as it is for integers
  - Declare such variables in C as float (single precision floating point number) or double (double precision floating point number).

52

Floating Point Representation

Normal form: +(-) 1.x * 2 ^{y}

Sign bit　Significand　Exponent

- How many bits for significand (mantissa) x?
- How many bits for exponent y
- Is y stored in its original value or in transformed value?
- How to represent +infinity and –infinity?
- How to represent 0?

53

53

Overflow and Underflow

- What if result is too large?
  - ❑ Overflow!
  - ❑ Overflow => Positive exponent larger than the value that can be represented in exponent field
- What if result too small?
  - ❑ Underflow!
  - ❑ Underflow => Negative exponent smaller than the value that can be represented in Exponent field
- How to reduce the chance of overflow or underflow?

54

54

IEEE 754 FP Standard—Single Precision

Sign bit Biased Exponent Significand

S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF

Bits 31 30 23 22 1 0

- Bit 31 for sign
  - ❑ S=1 for negative numbers, 0 for positive numbers
- Bits 23-30 for biased exponent
  - ❑ The real exponent = E –127
  - ❑ 127 is called bias.
- Bits 0-22 for significand

55

55

IEEE 754 FP Standard—Single Precision (Cont.)

The value V of a single precision FP number is determined as follows:

- If 0<E<255 then $V=(-1)^{S} * 2^{E-127} * 1.F$ where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E = 255 and F is nonzero, then V=NaN ("Not a number")
- If E = 255 and F is zero and S is 1, then V= -Infinity
- If E = 255 and F is zero and S is 0, then V=Infinity
- If E = 0 and F is nonzero, then $V=(-1)^{S} * 2^{-126} * 0.F$. These are unnormalized numbers or subnormal numbers.
- If E = 0 and F is 0 and S is 1, then V=-0
- If E = 0 and F is 0 and S is 0, then V=0

56

56

IEEE 754 FP Standard—Single Precision (Cont.)

Subnormal numbers reduce the chance of underflow.

• Without subnormal numbers, the smallest positive number is 2^{-127}

• With subnormal numbers, the smallest positive number is $0.00000000000000000000001 * 2^{-126} = 2^{-(126+23)} = 2^{-149}$

57

57

IEEE 754 FP Standard—Double Precision

Sign bit Biased Exponent Significand

S EEEEEEEEEEE FFFFFFFFFF…FFFFFFFFFFFF

Bits 63 62 52 51 1 0

- Bit 63 for sign
  - S=1 for negative numbers, 0 for positive numbers
- Bits 52-62 for biased exponent
  - The real exponent = E –1023
  - 1023 is called bias.
- Bits 0-51 for significand

58

58

IEEE 754 FP Standard—Double Precision (Cont.)

The value V of a double precision FP number is determined as follows:

- If 0<E<2047 then $V=(-1)^S * 2^{E-1023} * 1.F$ where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E = 2047 and F is nonzero, then V=NaN ("Not a number")
- If E = 2047 and F is zero and S is 1, then V= -Infinity
- If E = 2047 and F is zero and S is 0, then V=Infinity
- If E = 0 and F is nonzero, then $V=(-1)^S * 2^{-1022} * 0.F$. These are unnormalized numbers or subnormal numbers.
- If E = 0 and F is 0 and S is 1, then V=-0
- If E = 0 and F is 0 and S is 0, then V=0

59

59

Implementing FP Addition by Software

How to implement x+y where x and y are two single precision FP numbers?

Step 1: Convert x and y into IEEE format

Step 2: Align two significands if two exponents are different.

- Let e1 and e2 are the exponents of x and y, respectively, and assume e1> e2. Shift the significand (including the implicit 1) of y right e1–e2 bits to compensate for the change in exponent.

Step 3: Add two (adjusted) significands.

Step 4: Normalize the result.

60

60

An Example

How to implement x+y where x=2.625 and y= – 4.75?

Step 1: Convert x and y into IEEE format

x=2.625 $\rightarrow$ 10.101 (Binary)

$\rightarrow$ 1.0101 * 2^1 (Normal form)

$\rightarrow$ 1.0101 * 2^{128} (IEEE format)

$\rightarrow$ 0 10000000 01010000000000000000000

Comments: The fraction part can be converted by multiplication. (This is the inverse of the division method for integers.)

0.625 × 2 = 1.25 1 (the most significant bit in fraction)

0.25 × 2 = 0.5 0

0.5 × 2 = 1.0 1 (the least significant bit in fraction)

61

61

An Example (Cont.)

y= – 4.75 $\rightarrow$ – 100.11 (Binary)

$\rightarrow$ – 1.0011 * 2^2 (Normal form)

$\rightarrow$ – 1.0011 * 2^{129} (IEEE format)

$\rightarrow$ 1 10000001 00110000000000000000000

Step 2: Align two significands.

The significand of x = 1.0101 $\rightarrow$ 0.10101 (After shift right 1 bit)

Comments: x=0.10101*2 ^{129} and y= –1.0011 *2 ^{129} after the alignment.

62

62

An Example (Cont.)

Step 3: Add two (adjusted) significands.

0.10101 ← The adjusted significand of x

– 1.00110 ← The significand of y

= – 0. 10001 ← The significand of x+y

Step 4: Normalize the result.

Result = – 0. 10001 * 2^{129} → – 1.0001 * 2^{128}

→ 1 10000000 00010000000000000000000

(Normal form)

63

63

Reading

1. http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/index.html.
2. http://www.cs.berkeley.edu/~wkahan/ieee754status/754story.html.

64

64

Reading Material

1. Appendix A in *Microcontrollers ands Microcomputers.*

65

65