Fixed Point Number System
Fractional Divider

Mohammad-Reza Movahedin
Computer Architecture
EE Dept. Sharif Univ.
1393/12/02

Fixed Point Numbers Representation

- We can extend integer (unsigned and signed) number system to fixed point numbers.
- An ***n***-bit whole and ***m***-bit fractional number $A = \overline{a_{n-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-m}}$ has a value of:

$$\sum_{i=-m}^{n-1} a_i \times 2^i$$

Reminder from decimal: $3.14 = 3\times10^0+1\times10^{-1}+4 \times 10^{-2}$

Fixed Point Numbers

- In order to convert fractional part of a decimal number to a fixed point binary representation, the number should be repeatedly multiplied by 2^N:
- If $A = 0.\overline{a_{-1}a_{-2} \dots a_{-m}}$, then a_{-1} = integer part of $2\times A$
- Example: 3.14 = 11.0010_0011_1101_0111 …

  $0.14\times16=2.24$, $0.24\times16=3.84$, $0.84\times16=13.44$, $0.44\text{x}16=7.04$
- Decimal numbers with limited fractional digits will not be necessarily represented by limited number of binary bits. What about the vice versa?

Fixed Point Add/Sub/Compare

- Addition, subtraction and comparison of fixed point numbers can be accomplished in the same way as integer numbers:

  just retain point location

  – Example: 3.14 ± 4.82 is equivalent to 314 ± 482

  – Overflow can occur, similar to integer operations

    - 3.14 + 9.82 can not be represented with one whole and two fractional digits.

Fixed Point Multiplication

- Multiplication of two fixed point numbers with ***n*** whole and ***m*** fractional digits results in ***2n*** whole and ***2m*** fractional digits.
- The result can be kept as it is,
- or
  - rounded in fractional part, and/or
  - clipped in whole section,
  - with or without overflow assertion.
- Example: $3.14 \times 4.82 =$
  - 15.1348
  - 15.13
  - 9.99, Overflown

Rounding

- Toward – ∞, aka flooring, truncation or chopping
- Toward + ∞, aka ceiling
- Toward nearest, aka rounding
- What about ties (A.5)?
  - Round up, down, random
  - Round to/away from 0
  - Round to nearest even ✓
    - Round to A if A is even
    - Round to A+1 if A is odd

	floor		
3.2	3		
-3.2			
3.7	3		
-3.7			
3.5	3		
4.5	4		

Rounding

- Toward – ∞, aka flooring, truncation or chopping
- Toward + ∞, aka ceiling
- Toward nearest, aka rounding
- What about ties (A.5)?
  – Round up, down, random
  – Round to/away from 0
  – Round to nearest even ✓
    - Round to A if A is even
    - Round to A+1 if A is odd

	floor		
3.2	3		
-3.2	-4		
3.7	3		
-3.7	-4		
3.5	3		
4.5	4		

Rounding

- Toward – ∞, aka flooring, truncation or chopping
- Toward + ∞, aka ceiling
- Toward nearest, aka rounding
- What about ties (A.5)?
  - Round up, down, random
  - Round to/away from 0
  - Round to nearest even ✓
    - Round to A if A is even
    - Round to A+1 if A is odd

	floor	ceil	
3.2	3	4	
-3.2	-4	-3	
3.7	3	4	
-3.7	-4	-3	
3.5	3	4	
4.5	4	5	

Rounding

- Toward – ∞, aka flooring, truncation or chopping
- Toward + ∞, aka ceiling
- Toward nearest, aka rounding
- What about ties (A.5)?
  - Round up, down, random
  - Round to/away from 0
  - Round to nearest even ✓
    - Round to A if A is even
    - Round to A+1 if A is odd

	floor	ceil	round
3.2	3	4	3
-3.2	-4	-3	-3
3.7	3	4	
-3.7	-4	-3	
3.5	3	4	
4.5	4	5	

Rounding

- Toward – ∞, aka flooring, truncation or chopping
- Toward + ∞, aka ceiling
- Toward nearest, aka rounding
- What about ties (A.5)?
  - Round up, down, random
  - Round to/away from 0
  - Round to nearest even ✓
    - Round to A if A is even
    - Round to A+1 if A is odd

	floor	ceil	round
3.2	3	4	3
-3.2	-4	-3	-3
3.7	3	4	4
-3.7	-4	-3	-4
3.5	3	4	
4.5	4	5	

Rounding

- Toward – ∞, aka flooring, truncation or chopping
- Toward + ∞, aka ceiling
- Toward nearest, aka rounding
- What about ties (A.5)?
  - Round up, down, random
  - Round to/away from 0
  - Round to nearest even ✓
    - Round to A if A is even
    - Round to A+1 if A is odd

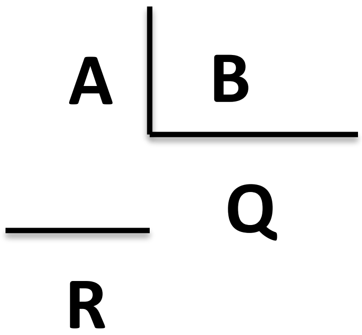
	floor	ceil	round
3.2	3	4	3
-3.2	-4	-3	-3
3.7	3	4	4
-3.7	-4	-3	-4
3.5	3	4	4
4.5	4	5	

Rounding

- Toward – ∞, aka flooring, truncation or chopping
- Toward + ∞, aka ceiling
- Toward nearest, aka rounding
- What about ties (A.5)?
  - Round up, down, random
  - Round to/away from 0
  - Round to nearest even ✓
    - Round to A if A is even
    - Round to A+1 if A is odd

	floor	ceil	round
3.2	3	4	3
-3.2	-4	-3	-3
3.7	3	4	4
-3.7	-4	-3	-4
3.5	3	4	4
4.5	4	5	4

Unsigned Integer Division



A: Dividend (مقسوم)

B: Divisor (مقسوم علیه)

Q: Quotient (خارج قسمت)

R: Reminder (باقیمانده)

$$A = Q \times B + R$$

$$0 \leq R < B$$

Unsigned Fractional Divider

- We focus on unsigned normalized fractional dividers:
- $A = \overline{1.a_{-1}a_{-2}\ .\ .\ .\ a_{-n}}$ or $A = \overline{0.1a_{-2}\ .\ .\ .\ a_{-n}}$
- $B = \overline{1.b_{-1}b_{-2}\ .\ .\ .\ b_{-n}}$ or $B = \overline{0.1b_{-2}\ .\ .\ .\ b_{-n}}$
- $Q = q_0.q_{-1}q_{-2}\ .\ .\ .\ q_{-m}\ (0.5 < Q < 2)$

$$A = Q \times B + R,\ 0 \leq R < B \times 2^{-m}$$

- Above means choose Q <u>as high as possible</u> but ensure:

$$Q \times B \leq A$$

$$q_0 \times B + q_{-1} \times 2^{-1} \times B + q_{-2} \times 2^{-2} \times B + \ .\ .\ .\ \leq A$$

- In order to ensure Q is as high as possible, we should start with q_0 and go forward to q_{-1}, q_{-2}, . . ., q_{-m}

Unsigned Fractional Divider, cont.

- We have to choose q_0 so that: $q_0 \times B \leq A$
- Check if $B \leq A$,
  - when Yes, then q_0=1, continue with A – B
  - otherwise q_0=0, continue with A
- In a better way, first compute R_0 = A – B, check the subtraction borrow output:
  - when 0 means $B \leq A$, then q_0=1, continue with it,
  - when 1 means $B > A$, then q_0=0, restore R_0 to A

Unsigned Fractional Divider, cont.

$$q_0 \times B + q_{-1} \times 2^{-1} \times B + q_{-2} \times 2^{-2} \times B + \ldots \leq A$$

$$q_{-1} \times 2^{-1} \times B + q_{-2} \times 2^{-2} \times B + \ldots \leq A - q_0 \times B$$

$$q_{-1} \times 2^{-1} \times B + q_{-2} \times 2^{-2} \times B + \ldots \leq R_0$$

$$q_{-1} \times B + q_{-2} \times 2^{-1} \times B + q_{-3} \times 2^{-2} \times B + \ldots \leq 2 \times R_0$$

- Compute $R_{-1} = 2 \times R_0 - B = (R_0 << 1) - B$
- Check the borrow
  - Choose q_{-1}
  - Keep or Restore R_{-1}
    - Note: R is partial reminder, i.e. the reminder of division so far
- Continue to extract ***m*** bits.

Unsigned Fractional Divider, cont.

- Restoring Division:
  - Compute $R_{-k} = 2 \times R_{-(k-1)} - B = (R_{-(k-1)} << 1) - B$
  - Check the borrow, choose q_{-k}, keep or Restore R_{-k}

- Non-Restoring Division:
  - Compute $R_{-k} = 2 \times R_{-(k-1)} +/- B = (R_{-(k-1)} << 1) +/- B$
    - Subtract when $q_{-(k-1)}=1$, otherwise Add
    - When $q_{-(k-1)}=0$, $R_{-(k-1)}$ is incorrect by $-B$, thus $2 \times R_{-(k-1)}$ is also incorrect by $-2B$
    - Error is compensated by adding B to $-2B$ to get desired $-B$
  - Check the sign
    - Choose q_{-k}
    - Always Keep R_{-k}, i.e. Non-Restore

Unsigned Fractional Divider, Restoring Example

1.00 : 1.10 (1 : 1.5)

1.00 – 1.10 $\rightarrow$ Borrow $\rightarrow$ q_0=0, Restore R_0=1.00

10.00 ($2R_0$) – 1.10 $\rightarrow$ No Borrow $\rightarrow$ q_{-1}=1, R_{-1}=0.10

1.00 ($2R_{-1}$) – 1.10 $\rightarrow$ Borrow $\rightarrow$ q_{-2}=0, Restore R_{-2}=1.00

This can be repeated for an arbitrary number of iteration

Q = 0.10101010101010 . . .

Note: R should be one bit more than A and B (Why?)

Unsigned Fractional Divider, Non-Restoring Decimal Example

$1 : 1.5$

$1 – 1.5 = – 0.5 \rightarrow q_0 = 0, R_0 = – 0.5$

$-1\ (2R_0) + 1.5 = + 0.5 \rightarrow q_1 = 1, R_{-1} = + 0.5$

$1\ (2R_1) – 1.5 = – 0.5 \rightarrow q_2 = 0, R_{-2} = -0.5$

This can be repeated for an arbitrary number of iteration

$Q = 0.10101010101010\ .\ .\ .$

Note: R should be two bits more than A and B (Why?)

Unsigned Fractional Divider, Restoring Verilog Model

```
module frac_divider #(parameter ni = 32, parameter no = 40) (
        input clk, start,
        input [0 : -ni] a, b,                    // a and b should be normalized, i.e. a[0]=b[0]=1
        output reg [0 : -no] q)                  // quotient: q[0].q[-1] ... q[-no] = a / b
    reg [9:0] cntr; reg [1 :-ni] pr; reg [0 :-ni] br;    // pr is partial reminder, br is divisor register
    wire borrow; wire [0:-ni] sub;
    assign {borrow, sub} = pr - br;
    always @(posedge clk)
        if(start) begin br <= b; pr <= a; cntr <= no + 1; end
        else if(cntr) begin
            cntr <= cntr - 1;
            q <= (q << 1) | (borrow ? 1'b0 : 1'b1);
            pr <= (borrow ? pr : sub) << 1;
        end
endmodule
```

Radix-N Division, aka SRT

- What is presented so far is Radix-2 division
  - A single bit per iteration
- Radix-N division extracts N bits per iteration
  - SRT (Sweeney, Robertson, and Tocher) algorithm
    - Estimate N-bit per iteration
    - Compensate for error in the next iteration
  - SRT-4, i.e. 2 bits per iteration, is widely used
- Pentium SRT-4 Bug cost Intel ca. $1B in 1994

Thank You

- Mid-Term Exam Date, Thu. 94/02/03, 9:00 am
- Online Weekly Quiz Day, Sats, 9:00 pm
- Lab Instr. Manual, tonight, God Willing
- Group #4 Instructor Assignment Issue Explanation
- Take Quiz

15 minutes Quiz

- Assume A and B are 8-bit signed numbers,
- A = 8’hAC, B=8’hCF
- Calculate:

  $A + B$

  $A - B$

  $A \times B$

- Note: addition and subtraction results should be calculated in both 8 and 9 bits. Multiplication result must be in 16 bits.