

# Computer Arithmetic

## Ch 8

ALU

Integer Representation

Integer Arithmetic

Floating-Point Representation

Floating-Point Arithmetic

19/09/2001

Copyright Teemu Kerola 2001

1

## Arithmetic Logical Unit (ALU) <sup>(2)</sup>

(aritmeettis-looginen  
yksikkö)

- Does all “work” in CPU Rest is management!
  - integer & floating point arithmetic's
  - copy values from one register to another
  - comparisons
  - left and right shifts
  - branch and jump address calculations
  - load/store address calculations
- Control signals from CPU control unit
  - what operation to perform and when

19/09/2001

Copyright Teemu Kerola 2001

2

## ALU Operations (5)

- Data from/to internal registers (latches)
  - input data may have been copied from normal registers, or it may have come from memory
  - output data may go to normal registers, or to memory
- Wait for maximum gate delay
- Result is ready
- Result may (also) be in flags
- Flags may cause an interrupt

Fig. 8.1

(lipuke)

19/09/2001

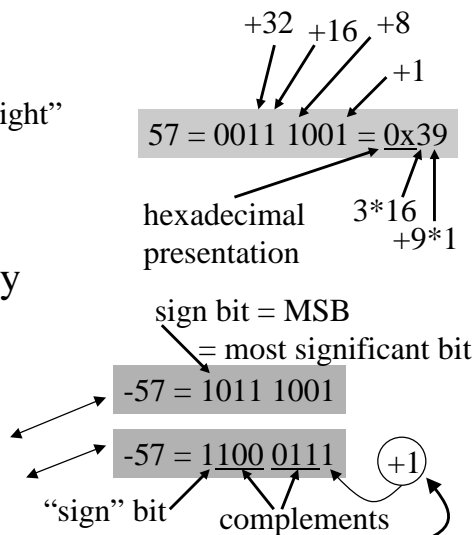
Copyright Teemu Kerola 2001

3

## Integer Representation (8)

Everything with 0 and 1  
 no plus/minus signs  
 no decimal periods  
 assumed “on the right”

- Unsigned integers
- Positive numbers easy
  - normal binary form
- Negative numbers
  - sign-magnitude
  - two's complement



19/09/2001

Copyright Teemu Kerola 2001

4

# Twos Complement

(kahden komplementti)

- Most used
- Have space for 8 bits?
  - use 7 bits for data and 1 bit for sign

+2 = 0000 0010  
 +1 = 0000 0001  
 0 = 0000 0000  
 -1 = 1111 1111  
 -2 = 1111 1110

– just like in sign-magnitude or in one's complement (but presentation is different)

ones complement: -0 = 1111 1111

19/09/2001
Copyright Teemu Kerola 2001
5

# Why Two's Complement Presentation? <sup>(4)</sup>

- Math is easy to implement
  - subtraction becomes addition
- Have just one zero
  - comparisons to zero easy
- Easy to expand to presentation with more bits
  - simple circuit

$X - Y = X + (-Y)$

easy to do,  
simple circuit

57 = 0011 1001 = 0000 0000 0011 1001

-57 = 1100 0111 = 1111 1111 1100 0111

↑  
sign extension

19/09/2001
Copyright Teemu Kerola 2001
6

## Why Two's Complement Presentation? <sup>(3)</sup>

- Range with n bits:  $-2^{n-1} \dots 2^{n-1} - 1$ 

8 bits:  $-2^7 \dots 2^7 - 1 = -128 \dots 127$   
 32 bits:  $-2^{31} \dots 2^{31} - 1 = -2\,147\,483\,648 \dots 2\,147\,483\,647$
- Overflow easy to recognise
  - add positive & negative: overflow not possible!
  - add 2 positive/negative numbers
    - if sign bit of result is different?  $\Rightarrow$  overflow!

57 = 0011 1001	outside range
+ 80 = 0101 0000	
137 = <u>1000</u> 1001	

19/09/2001
7
Copyright Teemu Kerola 2001

## Why Two's Complement Presentation? <sup>(5)</sup>

- Addition easy if one or both operands negative
  - treat them all as unsigned integers

Same circuit works for both (except for overflow check)

13 = 1101
+1 = 0001
14 = 1110

Digits represent 4 bit unsigned numbers

-3 = 1101
+1 = 0001
-2 = 1110

Digits represent 4 bit two's complement numbers

+3 = 0011
1100
+1
1101

19/09/2001
8
Copyright Teemu Kerola 2001

## Integer Arithmetic Operations

- Negation X = -Y
- Addition X = Y+Z
- Subtraction X = Y-Z
- Multiplication X = Y\*Z
- Division X = Y / Z

19/09/2001

Copyright Teemu Kerola 2001

9

## Integer Negation <sup>(6)</sup>

- Step 1: negate all bits
- Step 2: add 1
  - **Step 3: special cases**
    - ignore carry bit
      - negate 0?
    - check that sign bit really changes
      - can not negate smallest negative
      - results in exception

57 = 0011 1001

1100 0110

+1

1100 0111

0 = 0000 0000

1111 1111

+1

-0 = 1 0000 0000

-128 = 1000 0000

bitwise not: 0111 1111

add 1: 1000 0000

19/09/2001

Copyright Teemu Kerola 2001

10

## Integer Addition and Subtraction <sup>(4)</sup>

- Normal binary addition
  - 32 bit full adder?
- Ignore carry & monitor sign bit for overflow
- In case of SUB, complement 2nd operand
- 2 circuits
  - addition
  - complement

Fig. 8.6

19/09/2001

Copyright Teemu Kerola 2001

11

## Integer Multiplication <sup>(4)</sup>

- Complex
- Operands 32 bits  $\Rightarrow$  result 64 bits
- “Just like” you learned at school
  - optimised for binary data
    - it is easy to multiply with 0 or 1!
- Simpler case with unsigned numbers
  - simple circuits
    - adder
    - shifter
    - wires

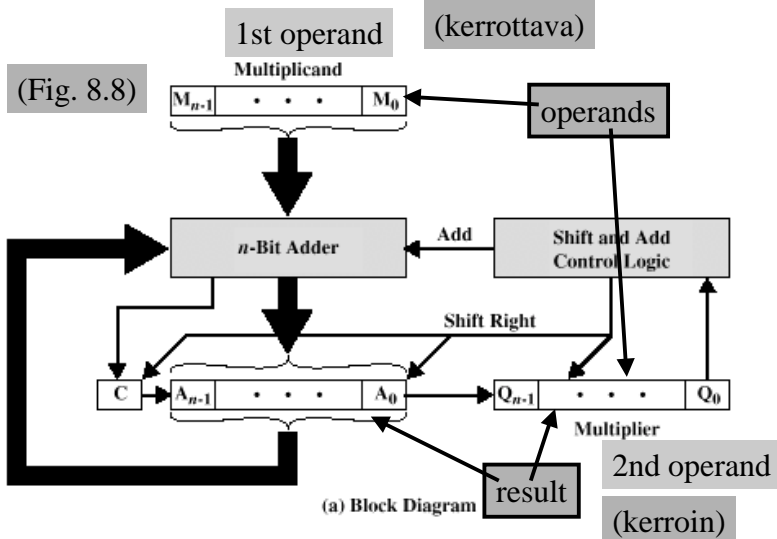
Fig. 8.7

19/09/2001

Copyright Teemu Kerola 2001

12

## Unsigned Multiplication Example



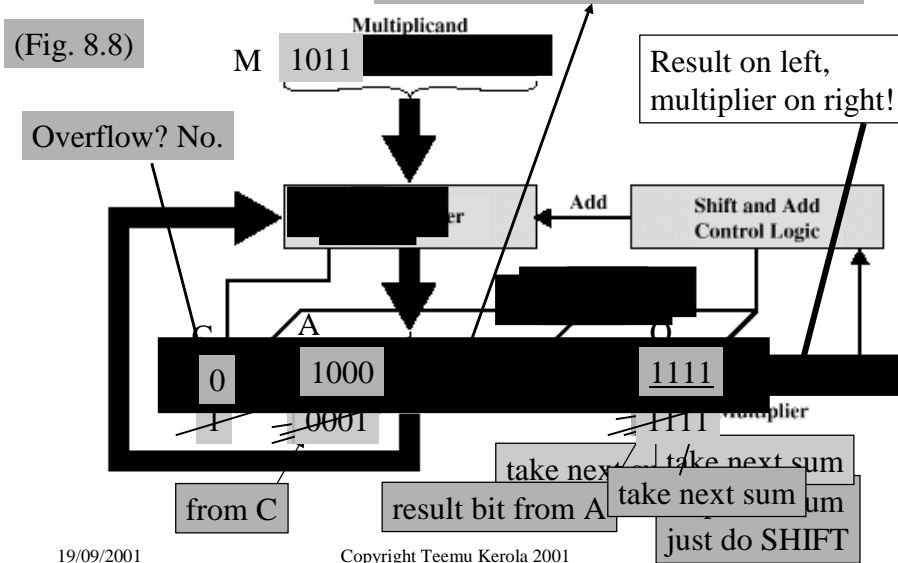
19/09/2001

Copyright Teemu Kerola 2001

13

## Unsigned Multiplication Example (19)

$$13 * 11 = ??? = 1000\ 1111 = 128 + 8 + 4 + 2 + 1 = 143$$



19/09/2001

Copyright Teemu Kerola 2001

## Multiplication with Negative Values

- Multiplication for unsigned numbers does not work for negative numbers
  - algorithm applies only for unsigned integer representation
  - not the same case as with addition
- Could do it all with unsigned values
  - change operands to positive values
  - do multiplication with positive values
  - negate result if needed
  - OK, but can do better, I.e., faster

19/09/2001

Copyright Teemu Kerola 2001

15

## The Gist in Booth's Algorithm <sup>(7)</sup>

Unsigned multiplication:  
addition for every "1" bit  
in multiplicand

$$5 * 7 \Rightarrow 0101 * 0111 \Rightarrow \begin{array}{r} 0101 \\ + 01010 \\ + 010100 \\ \hline = 100011 \end{array}$$

- Booth's algorithm:
  - combine all adjacent 1's in multiplicand together, replace all additions by one subtraction and one addition (to result)

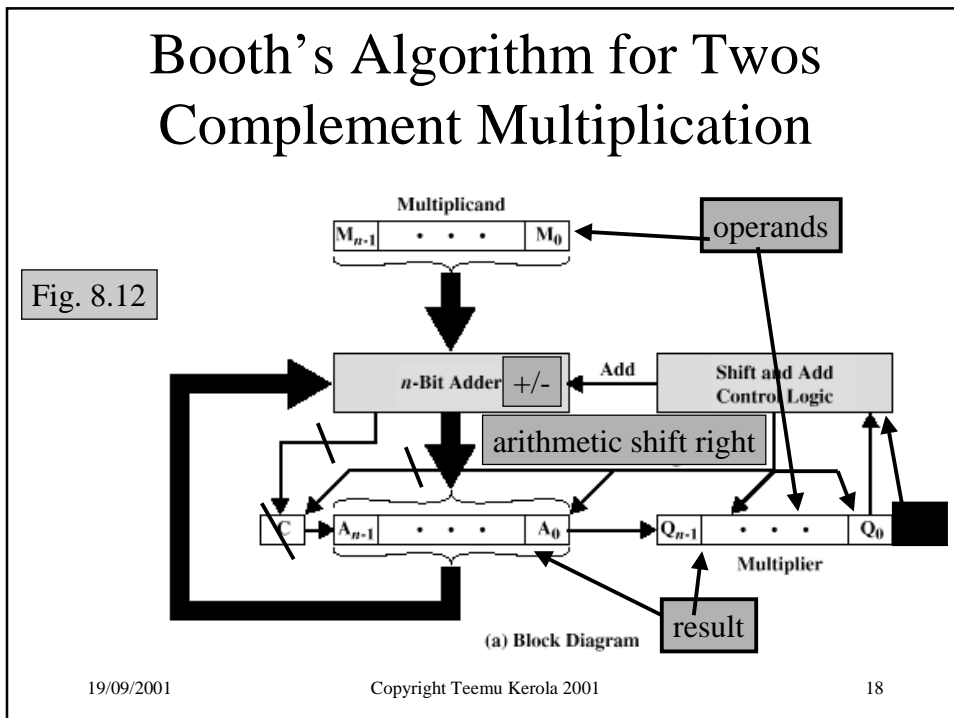
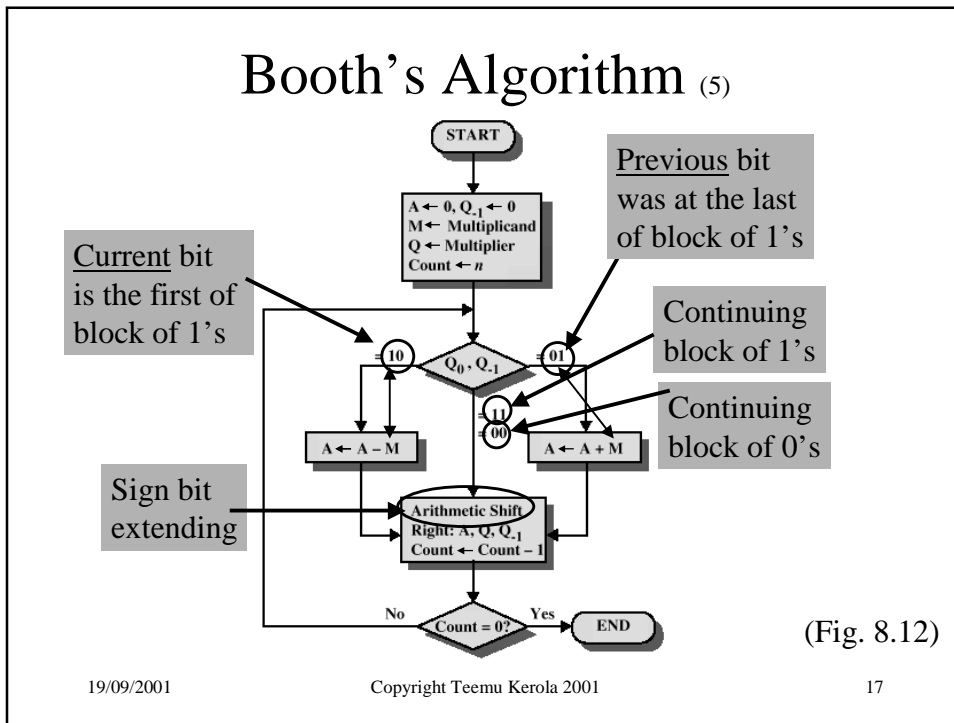
$$\begin{array}{l} 5 * 7 \Rightarrow 0101 * 0111 \\ \Rightarrow 0101 * (-0001 + 1000) \Rightarrow \begin{array}{r} +0101000 \\ - 0101 \\ \hline = 100011 \end{array} \end{array}$$

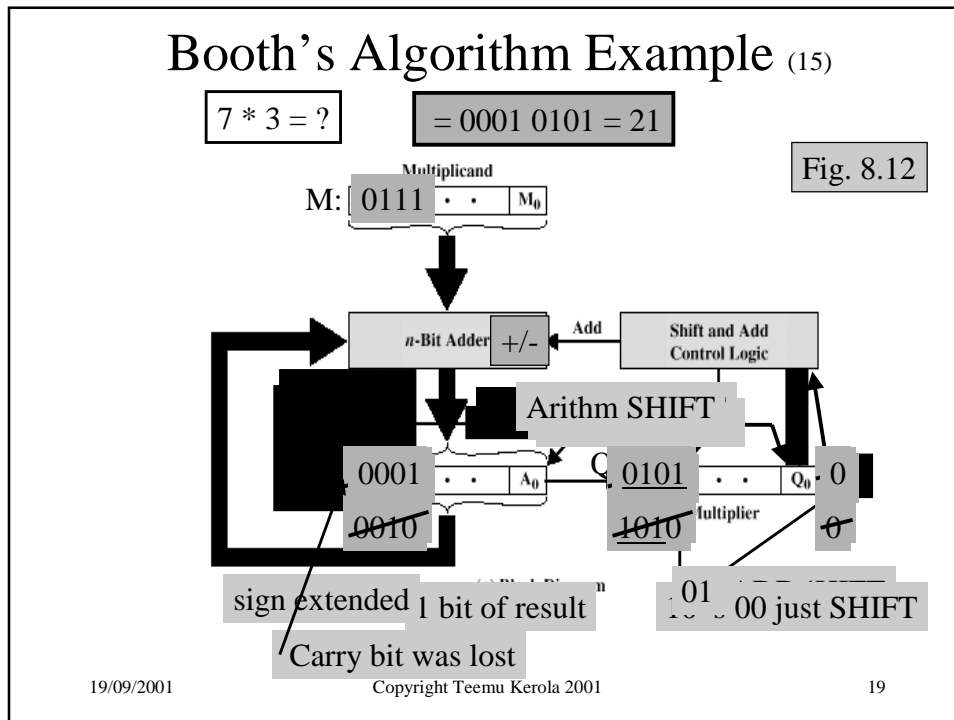
19/09/2001

Copyright Teemu Kerola 2001

16



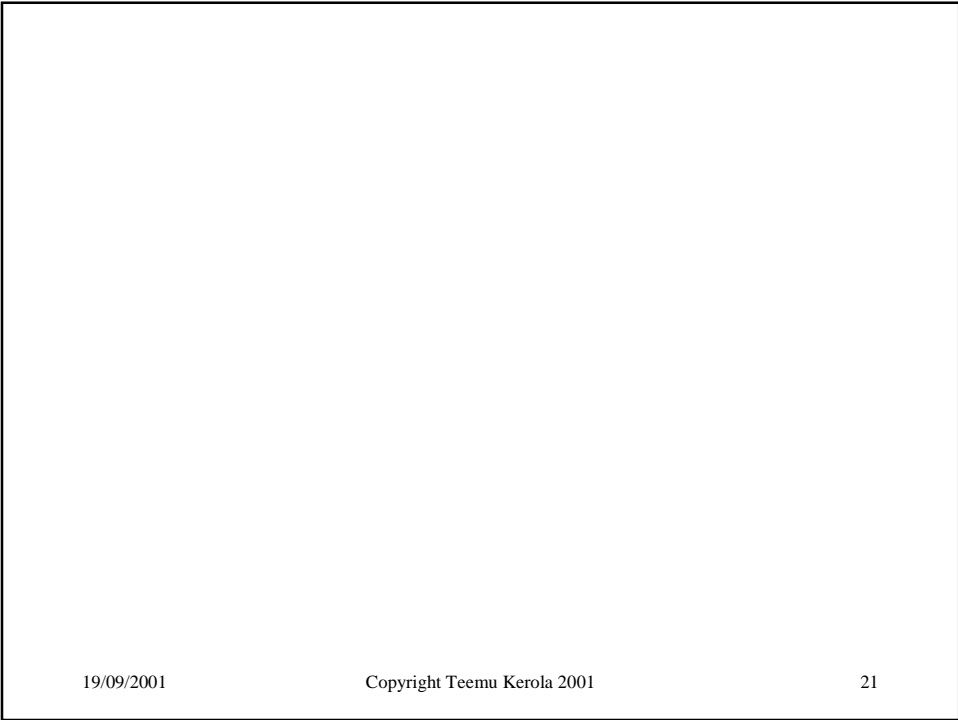




## Integer Division

- Like in school algorithm Fig. 8.15
  - easy: new quotient digit 0 or 1
  - M register for dividend (jaettava)
  - Q register for divisor & quotient (jakaja, osamäärä)
  - A register for (partial) remainder (jakojäännös)

19/09/2001
Copyright Teemu Kerola 2001
20



19/09/2001

Copyright Teemu Kerola 2001

21

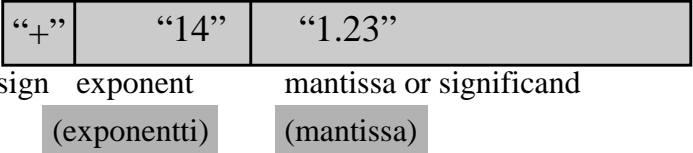
# Floating Point Representation

$$-0.000\ 000\ 000\ 123 = -1.23 * 10^{-10}$$

$$+0.123 = +1.23 * 10^{-1}$$

$$+123.0 = +1.23 * 10^2$$

$$+123\ 000\ 000\ 000\ 000 = +1.23 * 10^{14}$$



19/09/2001

Copyright Teemu Kerola 2001

22

## IEEE 32-bit Floating Point Standard

IEEE  
Standard 754

“+”	“14”	“1.1875” = “1.0011”
sign	exponent	mantissa or significand

- 1 bit for sign, 1  $\Rightarrow$  “-”, 0  $\Rightarrow$  “+”
- I.e., Stored value  $S \Rightarrow$  Sign value =  $(-1)^S$

19/09/2001
Copyright Teemu Kerola 2001
23

## IEEE 32-bit FP Standard

“+”	“15”	“1.1875” = “1.0011”
sign	exponent	mantissa or significand

- 8 bits for exponent,  $2^{8-1}-1= 127$  biased form

exponent = 5	$\xrightarrow{\text{store}}$	5+127 = 132 = 1000 0100
exponent = -1	$\xrightarrow{\text{store}}$	-1+127 = 126 = 0111 1110
exponent = 0	$\xrightarrow{\text{store}}$	0+127 = 127 = 0111 1111

– stored exponents 0 and 255 are special cases

- stored range: **1 - 254**  $\Rightarrow$  true range: **-126 - 127**

19/09/2001
Copyright Teemu Kerola 2001
24

### IEEE 32-bit FP Standard <sup>(7)</sup>

“+”	“15”	“0.1875” = “0.0011”
sign	exponent	mantissa or significand

$1/8 = 0.1250$

$1/16 = 0.0625$

$\frac{0.0625}{0.1875}$

- 23 bits for mantissa, stored so that
  - 1) Binary point (.) is assumed just right of first digit
  - 2) Mantissa is normalised, so that leftmost digit is 1
  - 3) Leftmost (most significant) digit (1) is not stored (implied bit)

0.0011	“15”
1.100	“12”
1000	“12”

mantissa exponent

24 bit mantissa!

19/09/2001
Copyright Teemu Kerola 2001
25

### IEEE 32-bit FP Values

$23.0 = +10111.0 * 2^0 = +1.0111 * 2^4 = ?$

$4+127=131$

0	1000 0011	011 1000 0000 0000 0000 0000
sign	exponent	mantissa or significand
1 bit	8 bits	23 bits

$1.0 = +1.0000 * 2^0 = ?$

$0+127 = 127$

0	0111 1111	000 0000 0000 0000 0000 0000
sign	exponent	mantissa or significand
1 bit	8 bits	23 bits

19/09/2001
Copyright Teemu Kerola 2001
26

## IEEE 32-bit FP Values

0	1000 0000	111 1000 0000 0000 0000 0000
---	-----------	------------------------------

sign 1 bit      exponent 8 bits      mantissa or significand 23 bits

$X = ?$

$X = (-1)^0 * 1.1111 * 2^{(128-127)}$

$= 1.1111_2 * 2$

$= (1 + 1/2 + 1/4 + 1/8 + 1/16) * 2$

$= (1 + 0.5 + 0.25 + 0.125 + 0.0625) * 2$

$= 1.9375 * 2$        **$= 3.875$**

19/09/2001
Copyright Teemu Kerola 2001
27

### IEEE-754 Floating-Point Conversion

Christopher Vickery  
Computer Science Department at  
Queens College of CUNY  
(The City University of New York)

19/09/2001

## IEEE FP Standard

- Single Precision (SP) 32 bits
- Double Precision (DP) 64 bits

(yksin- ja  
kaksinkertainen  
tarkkuus)

Table 8.3

- Special values
  - -0,  $+\infty$ ,  $-\infty$ , NaN
  - denormalized values

Table 8.4

Not a Number

19/09/2001

Copyright Teemu Kerola 2001

29

## IEEE SP FP Range

- Range
  - 8 bit exponent, effective range: -126 ... +127
  - range  $2^{-126} \dots 2^{127} \approx -10^{-38} \dots 10^{38}$
- Accuracy
  - 23 bit mantissa, 24 bit effective mantissa
  - change least significant digit in mantissa?
  - $2^{24} \approx 1.7 * 10^{-7} \approx 6$  decimal digits

19/09/2001

Copyright Teemu Kerola 2001

30

## Floating Point Arithmetic <sup>(4)</sup>

- Relatively simple Table 8.5
- Done from internal registers with all bits
  - implied bit included
- Add/subtract
  - more complex than multiplication
  - denormalize first one operand so that both have same exponent
- Multiplication/Division
  - handle mantissa and exponent separately

19/09/2001

Copyright Teemu Kerola 2001

31

## FP Add or Subtract <sup>(4)</sup>

- Check for zeroes  $1.234 \cdot 10^4$  +  $4.444 \cdot 10^6$ 
  - trivial if one or both operands zero
- Align mantissas  $0.01234 \cdot 10^6$   $4.444 \cdot 10^6$ 
  - same exponent
- Add/subtract  $4.45634 \cdot 10^6$ 
  - carry?
  - ⇒ shift right and add increase exponent
- Normalize result  $4.45634 \cdot 10^6$ 
  - shift left, reduce exponent

19/09/2001

Copyright Teemu Kerola 2001

32



## FP Special Cases

- Exponent overflow (ylivuoto)
  - above max Exception Or  $\pm\infty$  ?
- Exponent underflow (alivuoto)
  - below min Exception or zero or denormalized?
- Mantissa (significant) underflow
  - in denormalizing may move bits too much right
  - all significant bits lost? Oooops, lost data!
- Mantissa (significant) overflow Fix it
  - result of adding mantissas may have carry

19/09/2001

Copyright Teemu Kerola 2001

33

## FP Multiplication (Division) <sup>(7)</sup>

Check for zeroes

Result 0,  $\pm\infty$  ??

Add exponents

Subtract extra bias

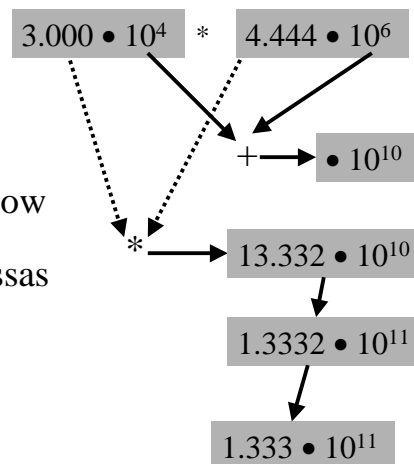
Report overflow/underflow

Multiply (divide) mantissas

Normalise

Round

(pyöristä)



19/09/2001

Copyright Teemu Kerola 2001

34

## Rounding (4)

- Guard bits
  - extra padding with zeroes
  - used with computations only
  - computations with more accuracy than data

$4.444 \cdot 10^6$

$4.44400 \cdot 10^6$

$$2.0 - 1.9999 \approx 1.000000 \cdot 2^1 - 0.1111111 \cdot 2^1$$

$$= 1.000000 \cdot 2^1 - 1.111111 \cdot 2^0$$

6 bit mantissa

$\begin{array}{r} 1.000000 \cdot 2^1 \\ - 0.111111 \cdot 2^1 \\ \hline = 0.000001 \cdot 2^1 \\ = 1.000000 \cdot 2^{-5} \end{array}$	Different accuracy!	$\begin{array}{r} 1.000000 \ 00 \cdot 2^1 \\ - 0.111111 \ 10 \cdot 2^1 \\ \hline = 0.000000 \ 10 \cdot 2^1 \\ = 1.000000 \ 00 \cdot 2^{-6} \end{array}$
-------------------------------------------------------------------------------------------------------------------------------------	---------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------

normalised

Align mantissas

2 guard bits

19/09/2001
Copyright Teemu Kerola 2001
35

## Rounding Choices (4)

4 digit accuracy in memory?

- Nearest representable 3.123 or -4.568
- Toward  $+\infty$  3.124 or -4.567
- Toward  $-\infty$  3.123 or -4.568
- Toward 0 3.123 or -4.567

19/09/2001
Copyright Teemu Kerola 2001
36

## IEEE $\infty$ and NaN

- $\infty$ 
  - outside range of finite numbers
  - rules for arithmetic with  $\infty$ :  $\infty + \infty = \infty$ , etc.
- NaN
  - invalid operation (E.g., 0.0/0.0) can result to NaN or exception
 

Table 8.6

    - user control
    - quiet NaN, or exception?
  - un-initialized data?
  - programming language support?

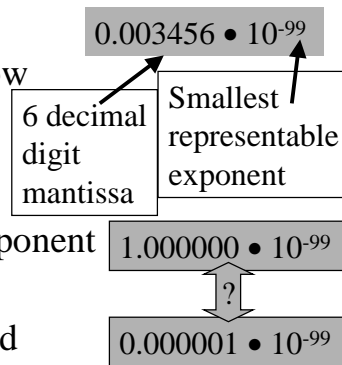
19/09/2001

Copyright Teemu Kerola 2001

37

## IEEE Denormalized Numbers (4)

- Problem: What to do when can not normalize any more?
  - Exponent would underflow
- Answer: Denormalized representation
  - smallest representable exponent reserved for this purpose
  - mantissa is not normalized
  - smallest (closest to zero) value is now much smaller than with normalized representation



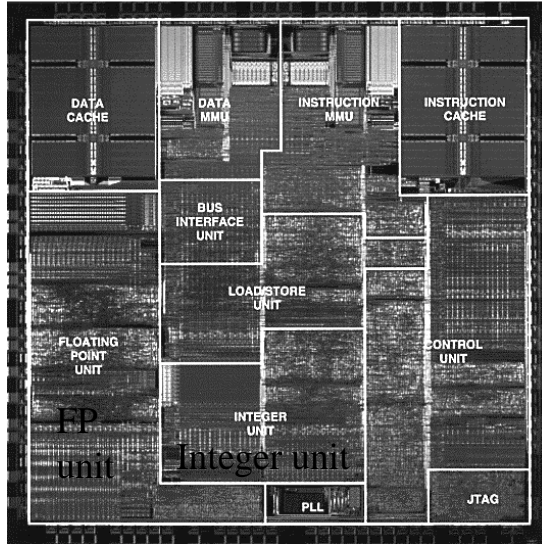
19/09/2001

Copyright Teemu Kerola 2001

38

-- End of Chapter 8: Arithmetic --

Motorola's PowerPC™ 602 RISC Microprocessor



[http://infopad.eecs.berkeley.edu/CIC/die\\_photos/](http://infopad.eecs.berkeley.edu/CIC/die_photos/)

19/09/2001

Copyright Teemu Kerola 2001

39