BLM5207
Computer Organization

Prof. Dr. Nizamettin AYDIN
naydin@yildiz.edu.tr
http://www3.yildiz.edu.tr/~naydin

Computer Arithmetic

1

1

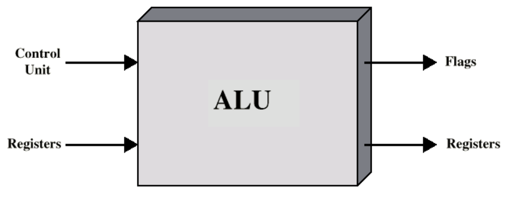
Arithmetic & Logic Unit

- Does the calculations
- Everything else in the computer is there to service this unit
- Handles integers
- May handle floating point (real) numbers
- May be separate FPU (maths co-processor)
- May be on chip separate FPU (486DX +)

2

2

ALU Inputs and Outputs



3

3

Integer Representation

- Only have 0 & 1 to represent everything
- Positive numbers stored in binary
  - e.g. 41=00101001
- No minus sign
- No period
- Sign-Magnitude
- Two's complement

4

4

Sign-Magnitude

- Left most bit is sign bit
- 0 means positive
- 1 means negative
- +18 = 00010010
- -18 = 10010010
- Problems
  - Need to consider both sign and magnitude in arithmetic
  - Two representations of zero (+0 and -0)

$$A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{if } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{if } a_{n-1} = 0 \end{cases}$$

5

5

Two's Complement

- +3 = 00000011
- +2 = 00000010
- +1 = 00000001
- +0 = 00000000
- -1 = 11111111
- -2 = 11111110
- -3 = 11111101

$$-2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

6

6

Characteristics of Twos Complement Representation and Arithmetic

Range	-2^{n-1} through $2^{n-1} - 1$
Number of Representations of Zero	One
Negation	Take the Boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer.
Expansion of Bit Length	Add additional bit positions to the left and fill in with the value of the original sign bit.
Overflow Rule	If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the result has the opposite sign.
Subtraction Rule	To subtract *B* from *A*, take the twos complement of *B* and add it to *A*.

7

7

Benefits

- One representation of zero
- Arithmetic works easily (see later)
- Negating is fairly easy
  - 3 = 00000011
  - Boolean complement gives 11111100
  - Add 1 to LSB 11111101

8

8

Negation Special Case 1

- 0 = 00000000
- Bitwise not 11111111
- Add 1 to LSB +1
- Result 1 00000000
- Overflow is ignored, so:
- - 0 = 0 √

9

9

Negation Special Case 2

- -128 = 10000000
- bitwise not 01111111
- Add 1 to LSB +1
- Result 10000000
- So:
- -(-128) = -128 X
- Monitor MSB (sign bit)
- It should change during negation

10

10

Range of Numbers

- 8 bit 2s complement
  - +127 = 01111111 = 2^7 -1
  - -128 = 10000000 = -2^7
- 16 bit 2s complement
  - +32767 = 011111111 11111111 = 2^{15} - 1
  - -32768 = 100000000 00000000 = -2^{15}

11

11

Conversion Between Lengths

- Positive number pack with leading zeros
- +18 = 00010010
- +18 = 00000000 00010010
- Negative numbers pack with leading ones
- -18 = 10010010
- -18 = 11111111 10010010
- i.e. pack with MSB (sign bit)

12

12

Fixed-Point Representation

- Number representation discussed so far also referred as fixed point.
  - Because the radix point (binary point) is fixed and assumed to be to the right of the rightmost digit (least significant digit).

13

13

Integer Arithmetic

- Negation:
  - In sign magnitude, simply invert the sign bit.
  - In twos complement:
    - Apply twos complement operation (take bitwise complement including sign bit, and add 1)

14

14

Addition and Subtraction

- Normal binary addition
- Monitor sign bit for overflow
- Take twos complement of subtrahend and add to minuend
  - i.e. a - b = a + (-b)
- So we only need addition and complement circuits

15

15

Addition and Subtraction

- Overflow rule
  - If two numbers are added and they are both positive or both negative, then overflow occurs if and only if the result has the opposite sign
- Subtraction rule
  - To subtract one number(subrahend) from another (minuhend), take twos complement (negation) of the subtrahend and add it to the minuhend

16

16

Addition of Numbers in Twos Complement Representation

1001 = −7 +0101 = 5 1110 = −2 (a) (−7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 (b) (−4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = −4 +1111 = −1 11011 = −5 (d) (−4) + (−1)
0101 = 5 +0100 = 4 1001 = Overflow (e) (+5) + (+4)	1001 = −7 +1010 = −6 10011 = Overflow (f) (−7) + (−6)

17

17

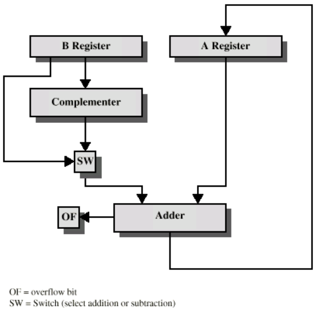
Subtraction of Numbers in Twos Complement Representation (M – S)

0010 = 2 +1001 = −7 1011 = −5 (a) M = 2 = 0010 S = 7 = 0111 −S = 1001	0101 = 5 +1110 = −2 10011 = 3 (b) M = 5 = 0101 S = 2 = 0010 −S = 1110
1011 = −5 +1110 = −2 11001 = −7 (c) M = −5 = 1011 S = 2 = 0010 −S = 1110	0101 = 5 +0010 = 2 0111 = 7 (d) M = 5 = 0101 S = −2 = 1110 −S = 0010
0111 = 7 +0111 = 7 1110 = Overflow (e) M = 7 = 0111 S = −7 = 1001 −S = 0111	1010 = −6 +1100 = −4 10110 = Overflow (f) M = −6 = 1010 S = 4 = 0100 −S = 1100

18

18

Hardware for Addition and Subtraction



19

19

Multiplication

- Complex
- Work out partial product for each digit
- Take care with place value (column)
- Add partial products

20

20

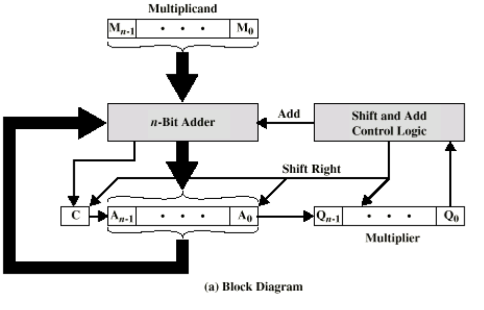
Multiplication Example

- 1011 Multiplicand (11 dec)
- x 1101 Multiplier (13 dec)
- 1011 Partial products
- 0000 Note: if multiplier bit is 1 copy
- 1011 multiplicand (place value)
- 1011 otherwise zero
- 10001111 Product (143 dec)
- Note: need double length result

21

21

Unsigned Binary Multiplication



22

22

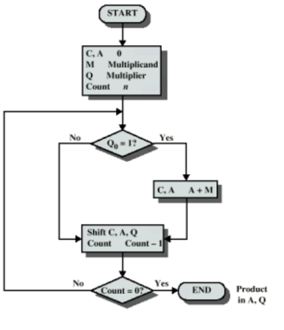
Execution of Example

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

23

23

Flowchart for Unsigned Binary Multiplication



24

24

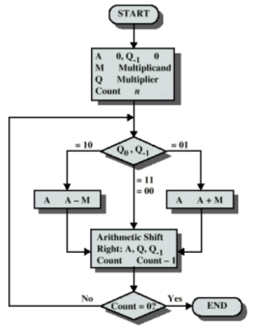
Multiplying Negative Numbers

- This does not work!
- Solution 1
  - Convert to positive if required
  - Multiply as above
  - If signs were different, negate answer
- Solution 2
  - Booth's algorithm

25

25

Booth's Algorithm



26

26

Example of Booth's Algorithm

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

27

27

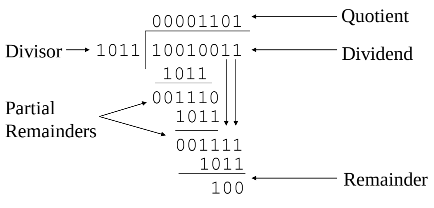
Division

- More complex than multiplication
- Negative numbers are really bad!
- Based on long division

28

28

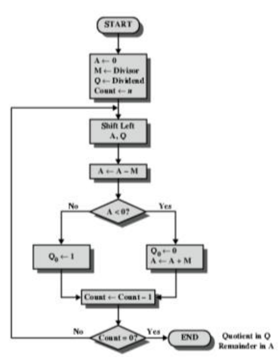
Division of Unsigned Binary Integers



29

29

Flowchart for Unsigned Binary Division



30

30

Example

A	Q	M = 0011	A	Q	M = 1101
0000	0111	Initial value	0000	0111	Initial value
0000	1110	Shift	0000	1110	Shift
1101		Subtract	1101		Add
0000	1110	Restore	0000	1110	Restore
0001	1100	Shift	0001	1100	Shift
1110		Subtract	1110		Add
0001	1100	Restore	0001	1100	Restore
0011	1000	Shift	0011	1000	Shift
0000		Subtract	0000		Add
0000	1001	Set $Q_0 = 1$	0000	1001	Set $Q_0 = 1$
0001	0010	Shift	0001	0010	Shift
1110		Subtract	1110		Add
0001	0010	Restore	0001	0010	Restore

(a) (7)/(3) (b) (7)/(−3)

31

31

Real Numbers

- Numbers with fractions
- Could be done in pure binary
  - 1001.1010 = $2^4 + 2^0 + 2^{-1} + 2^{-3}$ =9.625
- Where is the binary point?
- Fixed?
  - Very limited
- Moving?
  - How do you show where it is?

32

32

- 123 000 000 000 000

  1.23 X 10^{14}

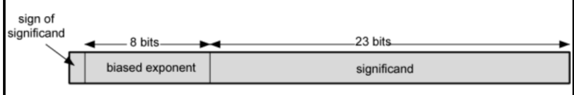
- 0.0000000000000123

  1.23 X 10^{-14}

33

33

Floating Point



(a) Format

- +/- .significand x $2^{exponent}$
- Misnomer
- Point is actually fixed between sign bit and body of mantissa
- Exponent indicates place value (point position)

34

34

Floating Point Examples

(a) Format

$$
\begin{aligned}
1.1010001 \times 2^{10100} &= 0\ 10010011\ 10100010000000000000000 = 1.638125 \times 2^{20} \\
-1.1010001 \times 2^{10100} &= 1\ 10010011\ 10100010000000000000000 = -1.638125 \times 2^{20} \\
1.1010001 \times 2^{-10100} &= 0\ 01101011\ 10100010000000000000000 = 1.638125 \times 2^{-20} \\
-1.1010001 \times 2^{-10100} &= 1\ 01101011\ 10100010000000000000000 = -1.638125 \times 2^{-20}
\end{aligned}
$$

(b) Examples

35

35

Signs for Floating Point

- Mantissa is stored in 2s complement
- Exponent is in excess or biased notation
  - e.g. Excess (bias) 128 means
  - 8 bit exponent field
  - Pure value range 0-255
  - Subtract 128 to get correct value
  - Range -128 to +127

36

36

Normalization

- FP numbers are usually normalized
- i.e. exponent is adjusted so that leading bit (MSB) of mantissa is 1
- Since it is always 1 there is no need to store it
- (c.f. Scientific notation where numbers are normalized to give a single digit before the decimal point
- e.g. 3.123×10^3)

37

37

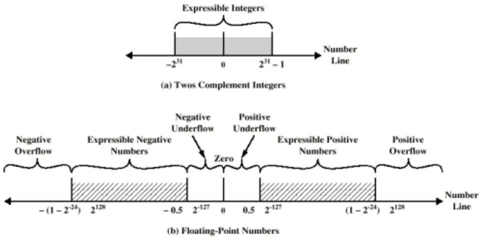
FP Ranges

- For a 32 bit number
  - 8 bit exponent
  - $+/- 2^{256} \approx 1.5 \times 10^{77}$
- Accuracy
  - The effect of changing lsb of mantissa
  - 23 bit mantissa $2^{-23} \approx 1.2 \times 10^{-7}$
  - About 6 decimal places

38

38

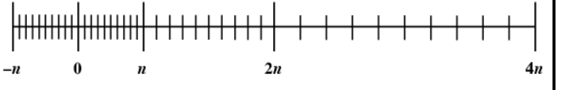
Expressible Numbers



39

39

Density of Floating Point Numbers



40

40

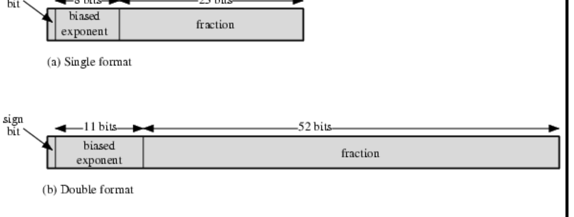
IEEE 754

- Standard for floating point storage
- 32 and 64 bit standards
- 8 and 11 bit exponent respectively
- Extended formats (both mantissa and exponent) for intermediate results

41

41

IEEE 754 Formats



42

42

IEEE 754 Format Parameters

Parameter	Format: Single	Single Extended	Double	Double Extended
Word width (bits)	32	≥ 43	64	≥ 79
Exponent width (bits)	8	≥ 11	11	≥ 15
Exponent bias	127	unspecified	1023	unspecified
Maximum exponent	127	≥ 1023	1023	≥ 16383
Minimum exponent	–126	≤ –1022	–1022	≤ –16382
Number range (base 10)	10^{-38}, 10^{+38}	unspecified	10^{-308}, 10^{+308}	unspecified
Significand width (bits)*	23	≥ 31	52	≥ 63
Number of exponents	254	unspecified	2046	unspecified
Number of fractions	2^{23}	unspecified	2^{52}	unspecified
Number of values	1.98×2^{31}	unspecified	1.99×2^{63}	unspecified

* not including implied bit

43

Interpretation of IEEE 754 Floating-Point Numbers

	Single Precision (32 bits)				Double Precision (64 bits)			
	Sign	Biased exponent	Fraction	Value	Sign	Biased exponent	Fraction	Value
positive zero	0	0	0	0	0	0	0	0
negative zero	1	0	0	–0	1	0	0	–0
plus infinity	0	255 (all 1s)	0	∞	0	2047 (all 1s)	0	∞
minus infinity	1	255 (all 1s)	0	–∞	1	2047 (all 1s)	0	–∞
quiet NaN	0 or 1	255 (all 1s)	≠ 0	NaN	0 or 1	2047 (all 1s)	≠ 0	NaN
signaling NaN	0 or 1	255 (all 1s)	≠ 0	NaN	0 or 1	2047 (all 1s)	≠ 0	NaN
positive normalized nonzero	0	0 < e < 255	f	$2^{e-127}(1.f)$	0	0 < e < 2047	f	$2^{e-1023}(1.f)$
negative normalized nonzero	1	0 < e < 255	f	$-2^{e-127}(1.f)$	1	0 < e < 2047	f	$-2^{e-1023}(1.f)$
positive denormalized	0	0	f ≠ 0	$2^{e-126}(0.f)$	0	0	f ≠ 0	$2^{e-1022}(0.f)$
negative denormalized	1	0	f ≠ 0	$-2^{e-126}(0.f)$	1	0	f ≠ 0	$-2^{e-1022}(0.f)$

44

Floating-Point Numbers and Arithmetic Operations

Floating Point Numbers	Arithmetic Operations
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$X + Y = \left(X_S \times B^{X_E - Y_E} + Y_S\right) \times B^{Y_E}$ $X - Y = \left(X_S \times B^{X_E - Y_E} - Y_S\right) \times B^{Y_E}$ $\}\ X_E \le Y_E$ $X \times Y = \left(X_S \times Y_S\right) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_S}{Y_S}\right) \times B^{X_E - Y_E}$

Examples:

$X = 0.3 \times 10^2 = 30$
$Y = 0.2 \times 10^3 = 200$

$X + Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3 = 0.23 \times 10^3 = 230$
$X - Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3 = (-0.17) \times 10^3 = -170$
$X \times Y = (0.3 \times 0.2) \times 10^{2+3} = 0.06 \times 10^5 = 6000$
$X \div Y = (0.3 \div 0.2) \times 10^{2-3} = 1.5 \times 10^{-1} = 0.15$

45

A floating-point operation may produce one of these conditions:

- Exponent overflow:
  - A positive exponent exceeds the maximum possible expo-nent value. In some systems, this may be designated as +∞ or —∞.
- Exponent underflow:
  - A negative exponent is less than the minimum possible exponent value (e.g., —200 is less than —127). This means that the number is too small to be represented, and it may be reported as 0.
- Significand underflow:
  - In the process of aligning significands, digits may flow off the right end of the significand. Some form of rounding is required.
- Significand overflow:
  - The addition of two significands of the same sign may result in a carry out of the most significant bit. This can be fixed by realignment.

46

FP Arithmetic +/-

- Check for zeros
- Align significands (adjusting exponents)
- Add or subtract significands
- Normalize result

47

FP Arithmetic +/- Phase 1

- Zero check

Because addition and subtraction are identical except for a sign change, the process begins by changing the sign of the subtrahend if it is a subtract operation. Next, if either operand is 0, the other is reported as the result.

48

FP Arithmetic +/- Phase 2

- Significand alignment
- Numbers needs to be manipulated so that the two exponents are equal.
  - To see the need for aligning exponents, consider the following decimal addition:
  - $(123 \times 10^0) + (456 \times 10^{-2})$
  - Clearly, we cannot just add the significands. The digits must first be set into equivalent positions, that is, the 4 of the second number must be aligned with the 3 of the first. Under these conditions, the two exponents will be equal,which is the mathematical condition under which two numbers in this form can be added. Thus,
  - $(123 \times 10^0) + (456 \times 10^{-2}) = (123 \times 10^0) + (4.56 \times 10^0) = 127.56 \times 10^0$

49

49

FP Arithmetic +/- Phase 2

Alignment may be achieved by shifting either the smaller number to the right (increasing its exponent) or shifting the larger number to the left. Because either operation may result in the loss of digits, it is the smaller number that is shifted; any digits that are lost are therefore of relatively small significance. The alignment is achieved by repeatedly shifting the magnitude portion of the significand right 1 digit and incrementing the exponent until the two exponents are equal. (Note that if the implied base is 16, a shift of 1 digit is a shift of 4 bits.) If this process results in a 0 value for the significand, then the other number is reported as the result. Thus, if two numbers have exponents that differ significantly, the lesser number is lost.

50

50

FP Arithmetic +/- Phase 3

- Addition

The two significands are added together, taking into account their signs. Because the signs may differ, the result may be 0. There is also the possibility of significand overflow by 1 digit. If so, the significand of the result is shifted right and the exponent is incremented. An exponent overflow could occur as a result; this would be reported and the operation halted.

51

51

FP Arithmetic +/- Phase 4

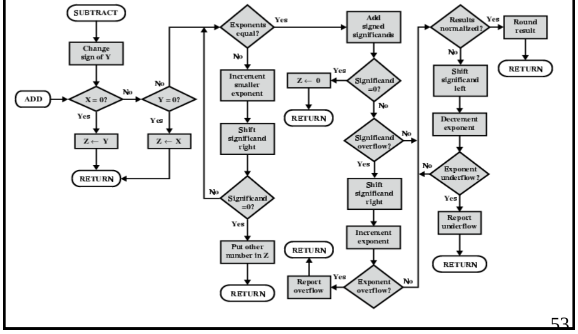
- Normalization

Normalization consists of shifting significand digits left until the most significant digit (bit, or 4 bits for base-16 exponent) is nonzero. Each shift causes a decrement of the exponent and thus could cause an exponent underflow. Finally, the result must be rounded off and then reported. We defer a discussion of rounding until after a discussion of multiplication and division.

52

52

FP Addition & Subtraction Flowchart



53

53

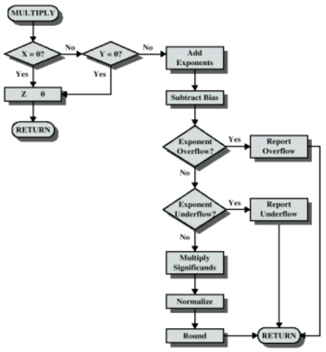
FP Arithmetic x/÷

- Check for zero
- Add/subtract exponents
- Multiply/divide significands (watch sign)
- Normalize
- Round
- All intermediate results should be in double length storage

54

54

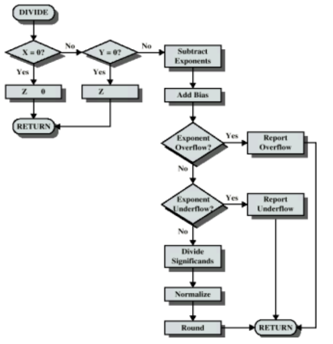
Floating Point Multiplication



55

55

Floating Point Division



56

56

57

57