Arithmetic

D.N.Rakhmatov

Adopted (with modifications) from:
R. Bryant, CMU
B. Leahy, Georgia Tech
D. Patterson, UC-Berkeley
C. Hamacher et al, *Computer Organization*, 6/E, © 2011 McGraw-Hill

N-Bit Number Ranges

NUMBER:	FROM:	TO:
Unsigned	0	$2^N - 1$
Signed Magnitude	$-(2^{N-1} - 1)$	$+(2^{N-1} - 1)$
Two's Complement	$-(2^{N-1})$	$+(2^{N-1} - 1)$
Biased (Bias = B)	$-B$	$2^N - 1 - B$

Example

N = 8

Binary	Unsigned	Signed Magn	One's Compl	Two's Compl	Biased 127	Biased 128
00000000	0	0	0	0	-127	-128
00000001	1	1	1	1	-126	-127
00000010	2	2	2	2	-125	-126
...	...	...	...	...	...	...
01111111	127	127	127	127	0	-1
10000000	128	-0	-127	-128	1	0
10000001	129	-1	-126	-127	2	1
...	...	...	...	...	...	...
11111110	254	-126	-1	-2	127	126
11111111	255	-127	-0	-1	128	127

Number Stored

Number Represented

ASCII

Decimal	Octal	Hex	Binary	Value
000	000	000	00000000	NUL
...				
048	060	030	00110000	0
049	061	031	00110001	1
...				
060	074	03C	00111100	<
061	075	03D	00111101	=
...				
065	101	041	01000001	A
066	102	042	01000010	B
...				
097	141	061	01100001	a
098	142	062	01100010	b
...				
126	176	07E	01111110	~
127	177	07F	01111111	DEL

Key Point

- The same bit pattern can mean different things to hardware depending on software
  - The computer just manipulates bits as instructed
  - Different representations used internally are based on typical time-space tradeoff considerations
- Example:

```
char c = 65;
printf(" %d \n", c);   /* prints 65 */
printf(" %c \n", c);   /* prints A  */
```


Recall Scientific Notation…

$$\underbrace{-}_{\text{Sign}}\underbrace{6.023}_{\text{Normalized* Mantissa}} \times \underbrace{10}_{\text{Base}}^{\overbrace{-}^{\text{Sign of Exponent}}\underbrace{23}_{\text{Exponent}}}$$

* **Normalized** = single non-zero leading digit

IEEE-754 Single Precision

31	30 23	22 0
x	xxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxx
sign	exponent	mantissa (significand)

$$(-1)^S * 1.M * 2^{E-127}$$

Sign
Implicit 1
Mantissa
Base
Exponent

Special Cases

31	30 23	22 0
x	xxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxx
sign	exponent	mantissa (significand)

$$(-1)^S * 2^{E-127} * 1.M$$

	E = 0	0 < E < 255	E = 255
M=0	0	Powers of Two	∞
M!=0	Denormalized: $(-1)^S 2^{-126}$ 0.M (Underflow)	Ordinary FP Numbers	Not a Number

Examples

```
0 00000000 00000000000000000000000 = +0
1 00000000 00000000000000000000000 = -0
0 11111111 00000000000000000000000 = +Infinity
1 11111111 00000000000000000000000 = -Infinity
0 11111111 10101010101010101010101 = NaN
1 11111111 01010101010101010101010 = NaN
0 01111111 10000000000000000000000
      = +1*2^(127-127)*1.1 = 1.5
0 11111110 00000000000000000000000
      = +1*2^(254-127)*1.0 = 2^127
1 00000001 00000000000000000000000
      = -1*2^(1-127)*1.0 = -2^(-126)
0 00000000 00000000000000000000001
      = +1*2^(-126)*0.00000000000000000000001
      = 2^(-149) (denormalized positive, leading 0)
```


Converting to IEEE-754

- $1/3 = 0.33333\ldots_{10}$
  $= 0.25 + 0.0625 + 0.015625 + 0.00390625 + 0.0009765625 + \ldots$
  $= 1/4 + 1/16 + 1/64 + 1/256 + 1/1024 + \ldots$
  $= 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + 2^{-10} + \ldots$
  $= 0.0101010101\ldots_2 * 2^0$
  $= 1.0101010101\ldots_2 * 2^{-2}$
- Sign: 0
- Exponent: $-2 + 127 = 125_{10} = 01111101_2$
- Significand: 0101010101…

0	01111101	01010101010101010101010

Converting from IEEE-754

0	01101000	10101010100001101000010

- Sign: 0 (positive number)
- Exponent:
  - 01101000 = 104
  - Bias adjustment: 104 - 127 = -23
- Significand:
  - $1+2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-9}+2^{-14}+2^{-15}+2^{-17}+2^{-22}$
    $= 1.0 + 0.666115...$
- $1.666115*2^{-23} \sim 1.986*10^{-7}$

IEEE-754 Double Precision

63	62 52	51 31 0
x	xx...xxx	xxxxxxxxxx...xxxxxxxxxx
sign	exponent	mantissa (significand)

$$(-1)^S * 1.M * 2^{E-1023}$$

Sign
Implicit 1
Mantissa
Base
Exponent

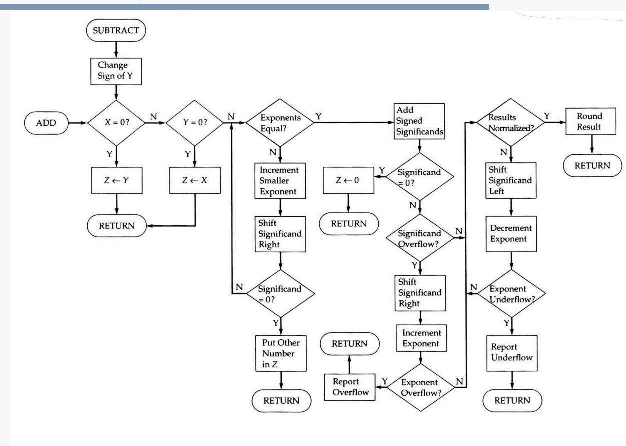
Special Cases

63	62 … 52	51 … 31 … 0
x	xx...xxx	xxxxxxxxxxx...xxxxxxxxxxx
sign	exponent	mantissa (significand)

$(-1)^S * 2^{E-1023} * 1.M$

	E = 0	0<E<2047	E = 2047
M=0	0	Powers of Two	∞
M!=0	Denormalized: $(-1)^S\,2^{-1022}\,0.M$ (Underflow)	Ordinary FP Numbers	Not a Number

Floating-Point Add/Subtract



Example I

- `18.75 + 0.1875 = 18.9375`
  - $18.75_{10} = 10010.11_2 = 1.001011 * 2^4$
    `= (-1)^0 * 2^(131-127) * 1.001011`
    `= 0 10000011 00101100000000000000000`
  - $0.1875_{10} = 0.0011_2 = 1.1 * 2^{-3}$
    `= (-1)^0 * 2^(124-127) * 1.1`
    `= 0 01111100 10000000000000000000000`
  - Don't forget implicit 1:
    `0 10000011 100101100000000000000000`
    `0 01111100 110000000000000000000000`
  - Next: match the exponents before addition!
    - The difference: `10000011 - 01111100 = 00000111 = 7`
    - Need to right-shift the smaller number by **7** bits

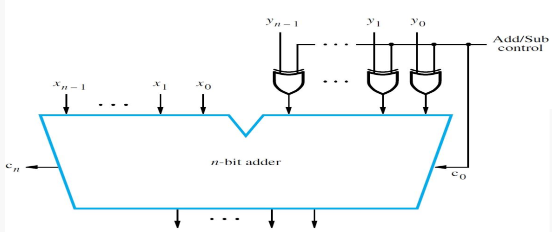
Example II

- `18.75 + 0.1875 = 18.9375`
  - Match the exponents:
    `0 10000011 100101100000000000000000`
    `0 10000011 000000011000000000000000`
  - Add:
    `0 10000011 100101111000000000000000`
  - Sum is already normalized (leading 1)
  - Actual bits stored:
    `0 10000011 00101111000000000000000`
  - Actual meaning:
    `0 10000011 00101111000000000000000`
    `+ 2^(131-127) * 1.18359375 = 18.9375`

Signed Adder/Subtractor

x_i	y_i	Carry-in c_i	Sum s_i	Carry-out c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

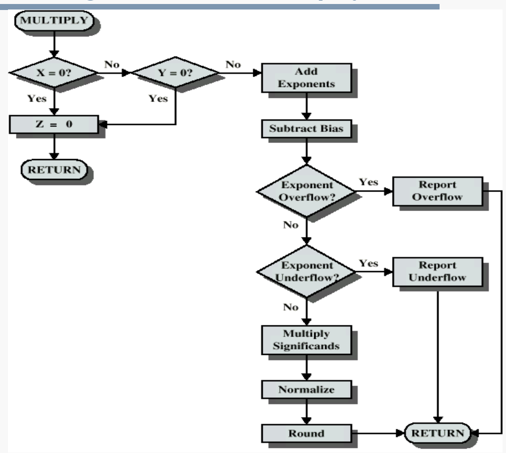
$$s_i = \overline{x}_i \overline{y}_i c_i + \overline{x}_i y_i \overline{c}_i + x_i \overline{y}_i \overline{c}_i + x_i y_i c_i = x_i \oplus y_i \oplus c_i$$
$$c_{i+1} = y_i c_i + x_i c_i + x_i y_i$$



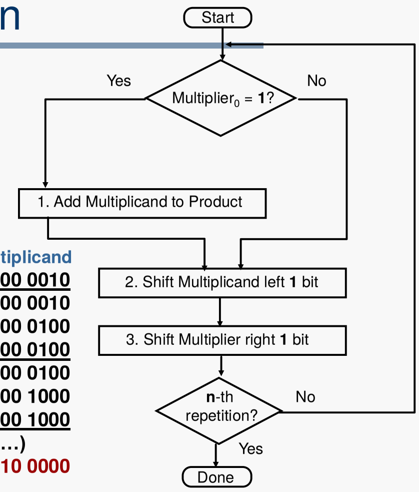
Two's Complement

- 2's complement: negation = bitwise inversion + 1
  - Example:
    -94_{10} `= -01011110 = 10100001+1 = 10100010`
- Example: adding two two's complement numbers
  $01011001_2 = 89_{10}$
  $\underline{11001101_2} = \underline{-51}_{10}$
  $100100110_2 \quad 38_{10}$
  - Note:
    - Overflow with 2's complements: positive + positive = negative, or negative + negative = positive
    - Overflow occurs when carry-in to sign bit position is NOT equal to carry-out
    - When there is no overflow, carry-out can be ignored

Floating-Point Multiply



Multiplication



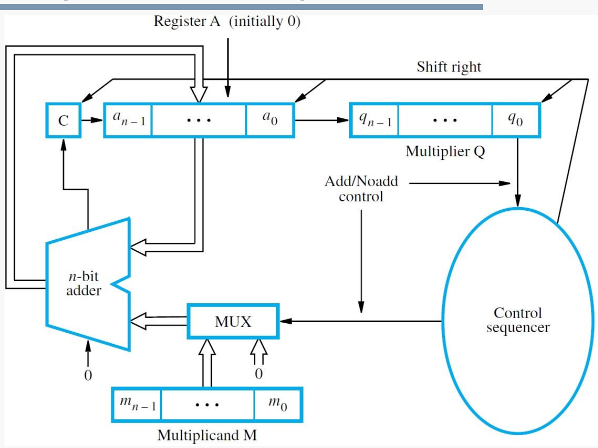
Example: **3** × **2**

	Product	Multiplier	Multiplicand
	0000 0000	0011	0000 0010
1:	0000 0010	0011	0000 0010
2:	0000 0010	0011	0000 0100
3:	0000 0010	0001	0000 0100
1:	0000 0110	0001	0000 0100
2:	0000 0110	0001	0000 1000
3:	0000 0110	0000	0000 1000

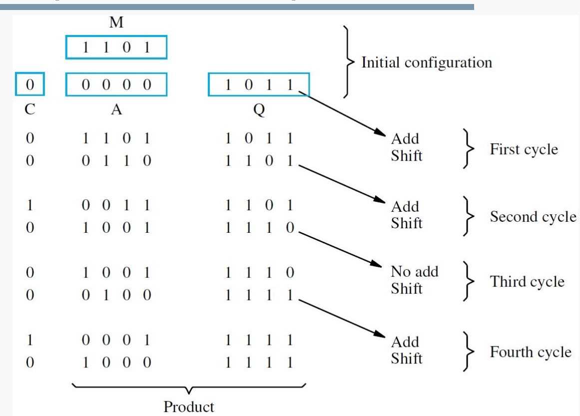
(two more iterations…)

0000 0110	0000	0010 0000

Sequential Multiplier I

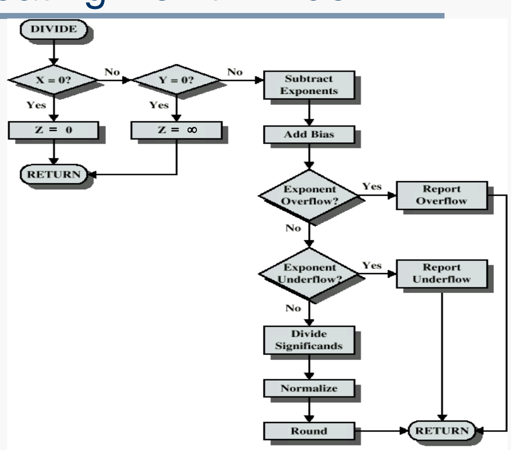


Sequential Multiplier II

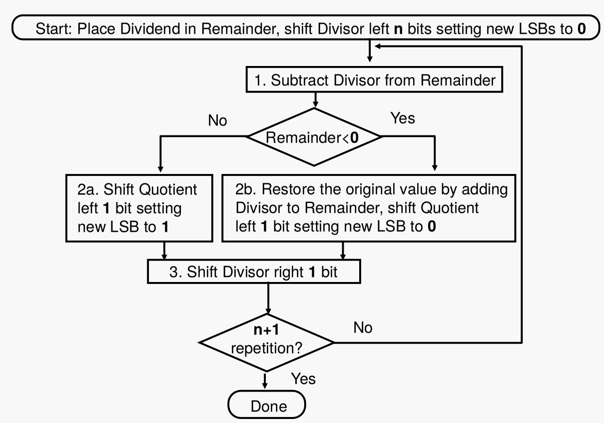


(b) Multiplication example

Floating-Point Divide



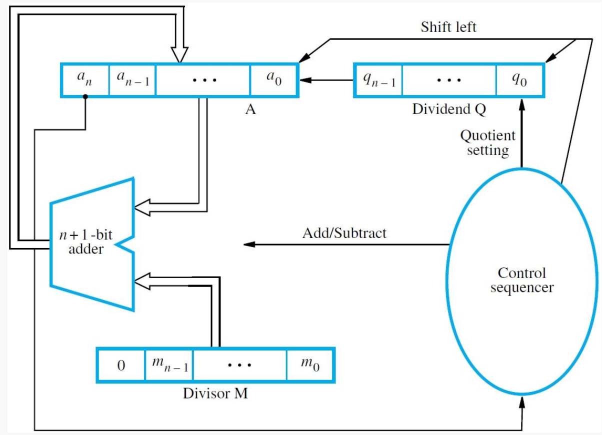
Division



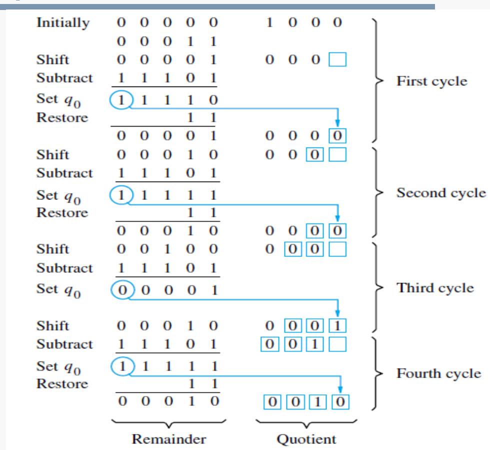
Example: 7 / 2

	Remainder	Quotient	Divisor
	0000 0111	0000	0010 0000
1:	1110 0111	0000	0010 0000
2:	0000 0111	0000	0010 0000
3:	0000 0111	0000	0001 0000
1:	1111 0111	0000	0001 0000
2:	0000 0111	0000	0001 0000
3:	0000 0111	0000	0000 1000
1:	1111 1111	0000	0000 1000
2:	0000 0111	0000	0000 1000
3:	0000 0111	0000	0000 0100
1:	0000 0011	0000	0000 0100
2:	0000 0011	0001	0000 0100
3:	0000 0011	0001	0000 0010
1:	0000 0001	0001	0000 0010
2:	0000 0001	0011	0000 0010
3:	0000 0001	0011	0000 0001
	0000 0001	0011	0000 0001

Sequential Divider I



Sequential Divider II



IEEE-754 Rounding

- Round towards **+infinity**
  - ALWAYS round up: **2.001 → 3**, **–2.001 → –2**
- Round towards **–infinity**
  - ALWAYS round down: **1.999 → 1**, **–1.999 → –2**
- Truncate
  - Drop the last bits (round towards 0)
- Round to (nearest) even
  - **2.5 → 2**, **3.5 → 4**
  - Ensures fairness of calculations on tie
    - Half the time we round up, the other half time we round down
  - This is the default rounding mode

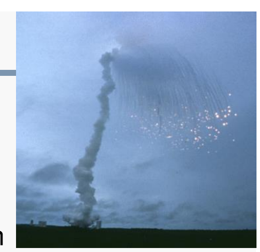
Integer to Float to Integer

```
float f;                /* IEEE-754 */
int i, j;               /* 2's complement */
i = 1073742079; j = i; /* j = 1073742079 */
f = (float)i;
i = (signed int)f;
if (i == j)
  { /* never executed, i = 1073742080 */ }
if (j/i == 1)
  { /* never executed, j/i = 0 */ }
```

- Loss of precision:

```
int:     SXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
float:   SEEEEEEEEXXXXXXXXXXXXXXXXXXXXXXX
```


Example



- Ariane-5 launcher (1996)
  - Exploded 37 sec after liftoff
    - Automatic self-destruction
  - Cargo worth $500 million
- Why? – Premature optimization
  - Software converted 64-bit floating-point horizontal acceleration to 16-bit signed integer
    - Worked OK for Ariane-4, but overflowed for faster Ariane-5
    - Ariane-5 reused the software from Ariane-4, assuming it was bug-free
      - ✓ Overflow caused an exception in the Inertial Reference System, which was not programmed to catch it and hence crashed
      - ✓ After crashing, the Inertial Reference System started writing diagnostic data on the internal bus, which continued to be interpreted as valid navigational data, thus leading to erratic maneuvering and explosion