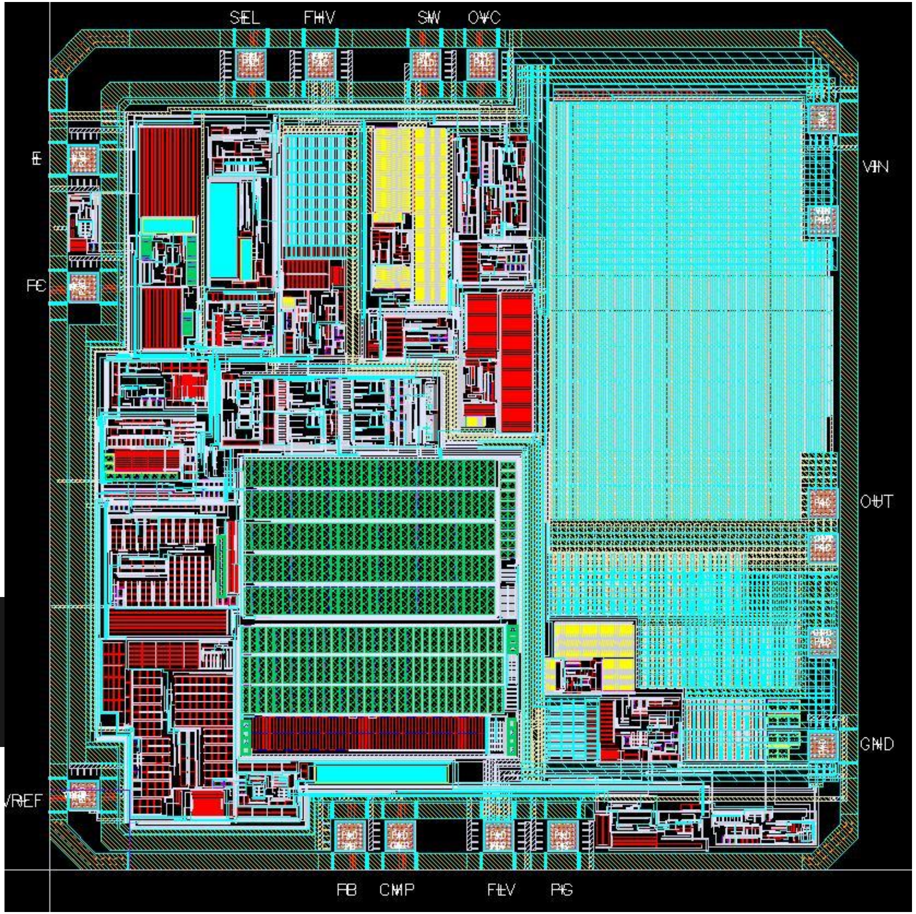
CHAPTER 9

FLOATING POINT ARITHMETIC

ECE4514

DIGITAL DESIGN 2



Sources

This slide set is derived from a number of online resources, not all of them attributed.

There's a Movie

Altera On-Demand Video

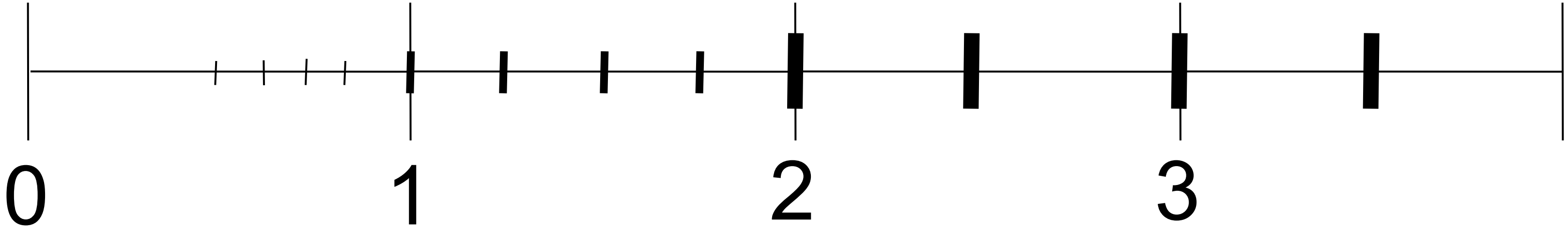
Implementing Floating-Point DSP in an FPGA

Distribution of Floating Point Numbers

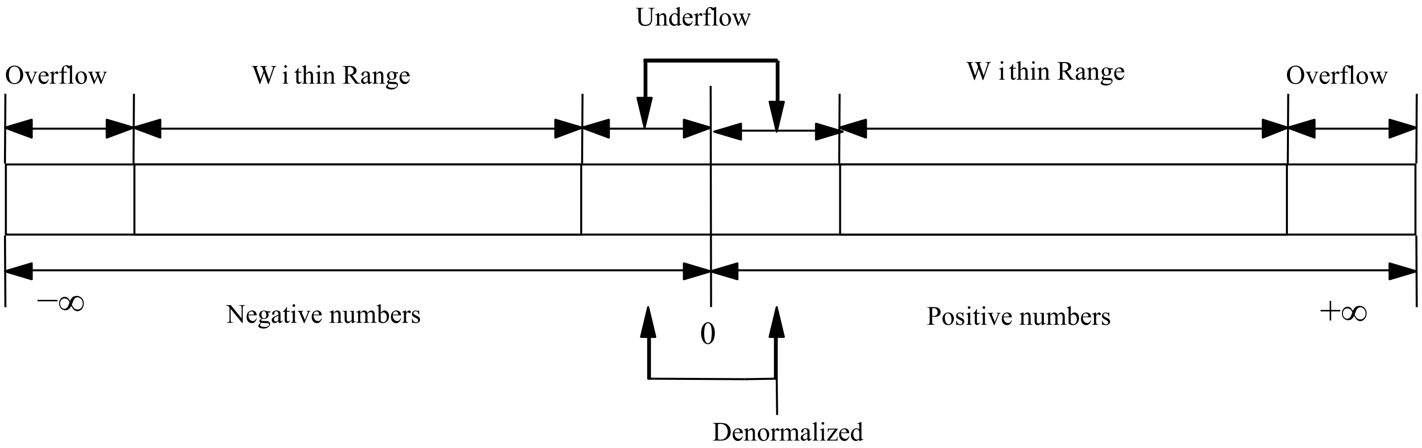
5 Bit Format

- **3 bit mantissa**
- **exponent {-1,0,1}**

e = -1	e = 0	e = 1
1.00 X 2^(-1) = 1/2	1.00 X 2^0 = 1	1.00 X 2^1 = 2
1.01 X 2^(-1) = 5/8	1.01 X 2^0 = 5/4	1.01 X 2^1 = 5/2
1.10 X 2^(-1) = 3/4	1.10 X 2^0 = 3/2	1.10 X 2^1= 3
1.11 X 2^(-1) = 7/8	1.11 X 2^0 = 7/4	1.11 X 2^1 = 7/2



Range of floating point numbers



Floating Point Standard

- Defined by IEEE Std 754-1985
- Developed in response to divergence of representations
  - Portability issues for scientific code
- Two representations
  - Single precision (32-bit)
  - Double precision (64-bit)

Floating Point

- An IEEE floating point representation consists of
  - A Sign Bit (no surprise)
  - An Exponent ("times 2 to the what?")
  - Mantissa ("Significand"), which is assumed to be 1.xxxxx (thus, one bit of the mantissa is implied as 1)
  - This is called a normalized representation
- So a mantissa = 0 really is interpreted to be 1.0, and a mantissa of all 1111 is interpreted to be 1.1111
- Special cases are used to represent denormalized mantissas (true mantissa = 0), NaN, etc.

IEEE FORMAT

	single: 8 bits double: 11 bits	single: 23 bits double: 52 bits
S	Exponent	Fraction

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)
- Normalize significand: 1.0 ≤ |significand| < 2.0
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1023

Single-Precision Range

- Exponents 00000000 and 11111111 reserved
- Smallest value
  - Exponent: 00000001
    $\Rightarrow$ actual exponent = 1 – 127 = –126
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- Largest value
  - exponent: 11111110
    $\Rightarrow$ actual exponent = 254 – 127 = +127
  - Fraction: 111…11 $\Rightarrow$ significand $\approx$ 2.0
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

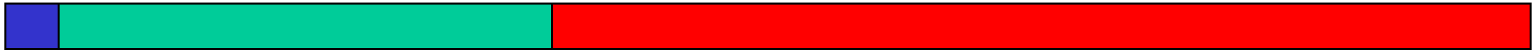
Double-Precision Range

- Exponents 0000…00 and 1111…11 reserved
- Smallest value
  - Exponent: 00000000001
    $\Rightarrow$ actual exponent = 1 – 1023 = –1022
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- Largest value
  - Exponent: 11111111110
    $\Rightarrow$ actual exponent = 2046 – 1023 = +1023
  - Fraction: 111…11 $\Rightarrow$ significand ≈ 2.0
  - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

Representation of Floating Point Numbers

- **IEEE 754 single precision**

31 30 23 22 0



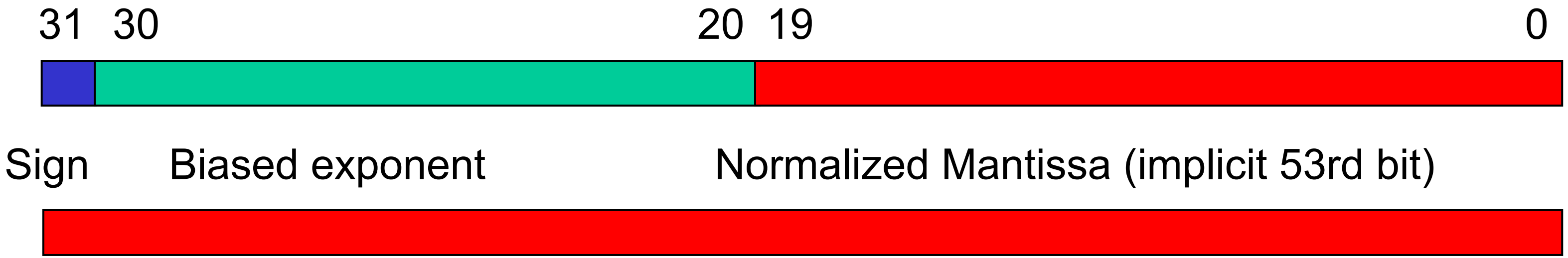
Sign Biased exponent Normalized Mantissa (implicit 24th bit = 1)

$$(-1)^s \times F \times 2^{E-127}$$

Exponent	Mantissa	Object Representec
0	0	0
0	non-zero	denormalized
1-254	anything	FP number
255	0	pm infinity
255	non-zero	NaN

Representation of Floating Point Numbers

- **IEEE 754 double precision**



$$(-1)^{s} \times F \times 2^{E-1023}$$

Exponent	Mantissa	Object Representec
0	0	0
0	non-zero	denormalized
1-2046	anything	FP number
2047	0	pm infinity
2047	non-zero	NaN

Floating Point Arithmetic

- fl(x) = nearest floating point number to x
- Relative error (precision = s digits)
  - $|x - fl(x)|/|x| \leq 1/2\beta^{1-s}$ for $\beta = 2$, 2^{-s}
- Arithmetic
  - $x \oplus y = fl(x+y) = (x + y)(1 + \varepsilon)$ for $\varepsilon < u$
  - $x \otimes y = fl(x \times y)(1 + \varepsilon)$ for $\varepsilon < u$

ULP—*U*nit in the *L*ast *P*lace is the smallest possible increment or decrement that can be made using the machine's FP arithmetic.

Relative Precision

All fraction bits are significant

- Single: approx 2^{-23}
  - Equivalent to $23 \times \log_{10}2 \approx 23 \times 0.3 \approx 6$ decimal digits of precision
- Double: approx 2^{-52}
  - Equivalent to $52 \times \log_{10}2 \approx 52 \times 0.3 \approx 16$ decimal digits of precision

Is FP Addition Associative?

- Associativity law for addition: a + (b + c) = (a + b) + c
- Let a = – 2.7 x 10^{23}, b = 2.7 x 10^{23}, and c = 1.0
- a + (b + c) = – 2.7 x 10^{23} + (2.7 x 10^{23} + 1.0) = – 2.7 x 10^{23} + 2.7 x 10^{23} = 0.0
- (a + b) + c = (– 2.7 x 10^{23} + 2.7 x 10^{23}) + 1.0 = 0.0 + 1.0 = 1.0
- Beware – Floating Point addition not associative!

Why Biased Exponent?

For faster comparisons (for sorting, etc.), allow integer comparisons of floating point numbers:

1/2	0	1111 1111	000 0000 0000 0000 0000 0000
2	0	0000 0001	000 0000 0000 0000 0000 0000

Unbiased exponent

1/2	0	0111 1110	000 0000 0000 0000 0000 0000
2	0	1000 0000	000 0000 0000 0000 0000 0000

Biased exponent

FP Example

- Represent –0.75
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - S = 1
  - Fraction = $1000\ldots00_2$
  - Exponent = –1 + Bias
    - Single: –1 + 127 = 126 = 01111110_2
    - Double: –1 + 1023 = 1022 = 01111111110_2
- Single: 1011111101000…00
- Double: 1011111111101000…00

FP Example

- What number is represented by the single-precision float

  11000000101000…00

  – S = 1

  – Fraction = $01000\ldots00_2$

  – Exponent = $10000001_2 = 129$

- $x = (-1)^1 \times (1 + .01_2) \times 2^{(129 - 127)}$

  $= (-1) \times 1.25 \times 2^2$

  $= -5.0$

Basic Technique

Basic Technique

- **Represent the decimal in the form +/- $1.xxx_b$ x 2^y**
- **And “fill in the fields”**
  - **Remember biased exponent and implicit “1.” mantissa!**
- **Examples:**
  - **0.0:** 0 00000000 00000000000000000000000
  - **1.0 (1.0 x 2^0):** 0 01111111 00000000000000000000000
  - **0.5 (0.1 binary = 1.0 x 2^-1):** 0 01111110 00000000000000000000000
  - **0.75 (0.11 binary = 1.1 x 2^-1):** 0 01111110 10000000000000000000000
  - **3.0 (11 binary = 1.1*2^1):** 0 10000000 10000000000000000000000
  - **-0.375 (-0.011 binary = -1.1*2^-2):** 1 01111101 10000000000000000000000
  - 1 10000011 01000000000000000000000 = - 1.01_2 * 2^4 = -20.0

IEEE compatible floating point multipliers

- *Algorithm*

Step 1

Calculate the tentative exponent of the product by adding the biased exponents of the two numbers, subtracting the bias, (). bias is 127 and 1023 for single precision and double precision IEEE data format respectively

Step 2

If the sign of two floating point numbers are the same, set the sign of product to '+', else set it to '-'.

Step 3

Multiply the two significands. For p bit significand the product is 2p bits wide (p, the width of significand data field, is including the leading hidden bit (1)). Product of significands falls within range .

Step 4

Normalize the product if MSB of the product is 1 (i.e. product of), by shifting the product right by 1 bit position and incrementing the tentative exponent.

Evaluate exception conditions, if any.

Step 5

Round the product if R(M0 + S) is true, where M0 and R represent the pth and (p+1)st bits from the left end of normalized product and Sticky bit (S) is the logical OR of all the bits towards the right of R bit. If the rounding condition is true, a 1 is added at the pth bit (from the left side) of the normalized product. If all p MSBs of the normalized product are 1's, rounding can generate a carry-out. In that case normalization (step 4) has to be done again.

Exceptions

Exceptions in IEEE 754

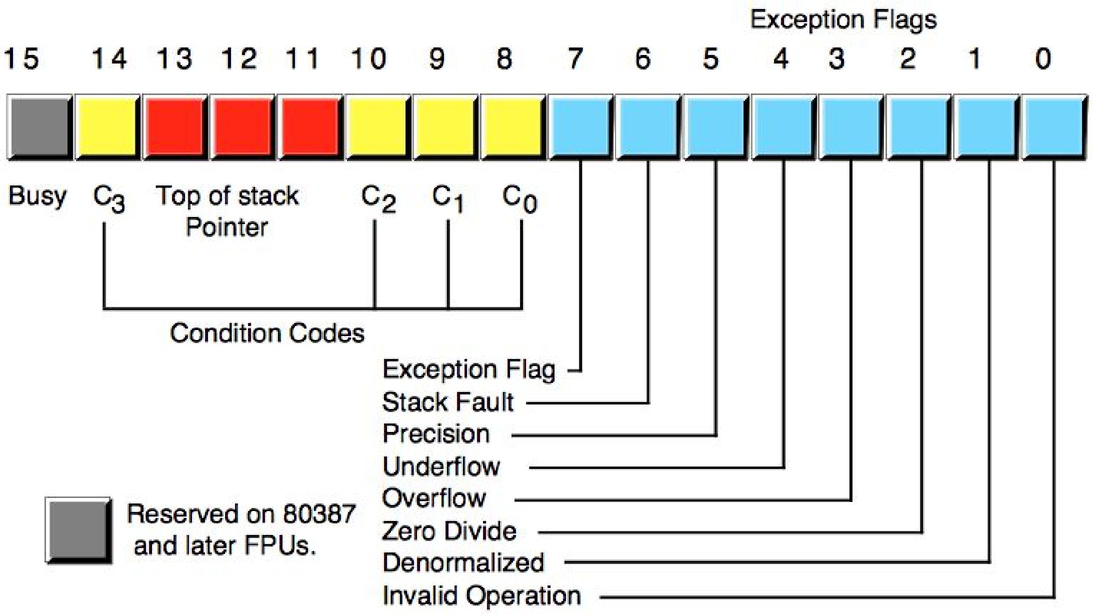
Exception	Remarks
Overflow	Result can be ± ∞ or default maximum value
Underflow	Result can be 0 or denormal
Divide by Zero	Result can be ± ∞
Invalid	Result is NaN
Inexact	System specified rounding may be required

- Operations that can generate Invalid Results

Operations→ Bad Results

Operation	Remarks
Addition/ Subtraction	An operation of the type ∞ ± ∞
Multiplication	An operation of the type 0 x ∞
Division	Operations of the type 0/0 and ∞/∞
Remainder	Operations of the type x REM 0 and ∞ REM y
Square Root	Square Root of a negative number

80x87 Exceptions



Rounding

Rounding

- Guard and round digits and sticky bit
  - When computing result, assume there are several extra digits available for shifting and computation. This improves accuracy of computation.
  - **Guard digit:** first extra digit/bit to the right of mantissa -- used for rounding addition results
  - **Round digit**: second extra digit/bit to the right of mantissa -- used for rounding multiplication results
  - **Sticky bit:** third extra digit/bit to the right of mantissa – used for resolving ties such as 0.50...00 vs. 0.50...01

Rounding Example

Rounding examples

- An example without guard and round digits
  - Add 9.76×10^{25} and 2.59×10^{24} assuming 3 digit mantissa
    - Shift mantissa of the smaller number to the right: 0.25×10^{25}
    - Add mantissas: 10.01×10^{25}
    - Check and normalize mantissa if necessary: 1.00×10^{26}

- An example with guard and round digits
  - Add 9.76×10^{25} and 2.59×10^{24} assuming 3 digit mantissa
    - Internal registers have extra two digits: 9.7600×10^{25} and 2.5900×10^{24}
    - Shift mantissa of the smaller number to the right: 0.2590×10^{25}
    - Add mantissas: 10.0190×10^{25}
    - Check and normalize mantissa if necessary: 1.0019×10^{26}
    - Round the result: 1.00×10^{26}

Rounding Example

- An example without guard and round digits
  - Add 9.78×10^{25} and 8.79×10^{24} assuming 3 digit mantissa
    - Shift mantissa of the smaller number to the right: 0.87×10^{25}
    - Add mantissas: 10.65×10^{25}
    - Normalize mantissa if necessary: 1.06×10^{26}

- An example with guard and round digits
  - Add 9.78×10^{25} and 8.79×10^{24} assuming 3 digit mantissa
    - Internal registers have extra two digits: 9.7800×10^{25} and 8.7900×10^{24}
    - Shift mantissa of the smaller number to the right: 0.8790×10^{25}
    - Add mantissas (note extra digit on the left): 10.6590×10^{25}
    - Check and normalize mantissa if necessary: 1.0659×10^{26}
    - Round the result: 1.07×10^{26}

Floating-Point Multiplication

Consider a 4-digit decimal example

- $1.110 \times 10^{10} \times 9.200 \times 10^{-5}$

1. Add exponents
   - For biased exponents, subtract bias from sum
   - New exponent = 10 + –5 = 5
2. Multiply significands
   - $1.110 \times 9.200 = 10.212 \Rightarrow 10.212 \times 10^{5}$
3. Normalize result & check for over/underflow
   - 1.0212×10^{6}
4. Round and renormalize if necessary
   - 1.021×10^{6}
5. Determine sign of result from signs of operands
   - $+1.021 \times 10^{6}$

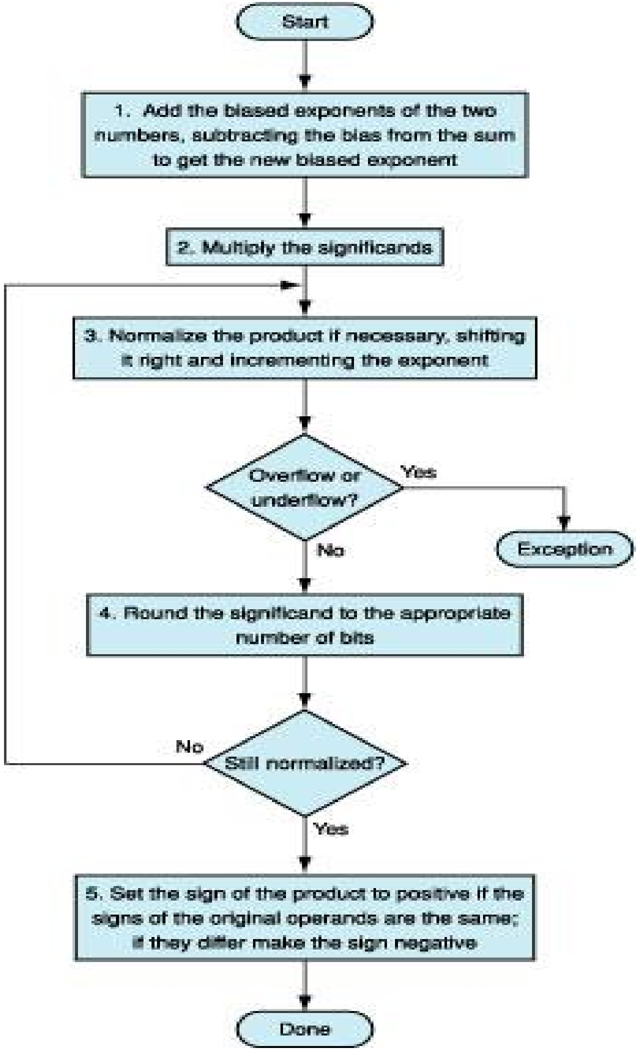
FP Multiplication

Now consider a 4-digit binary example

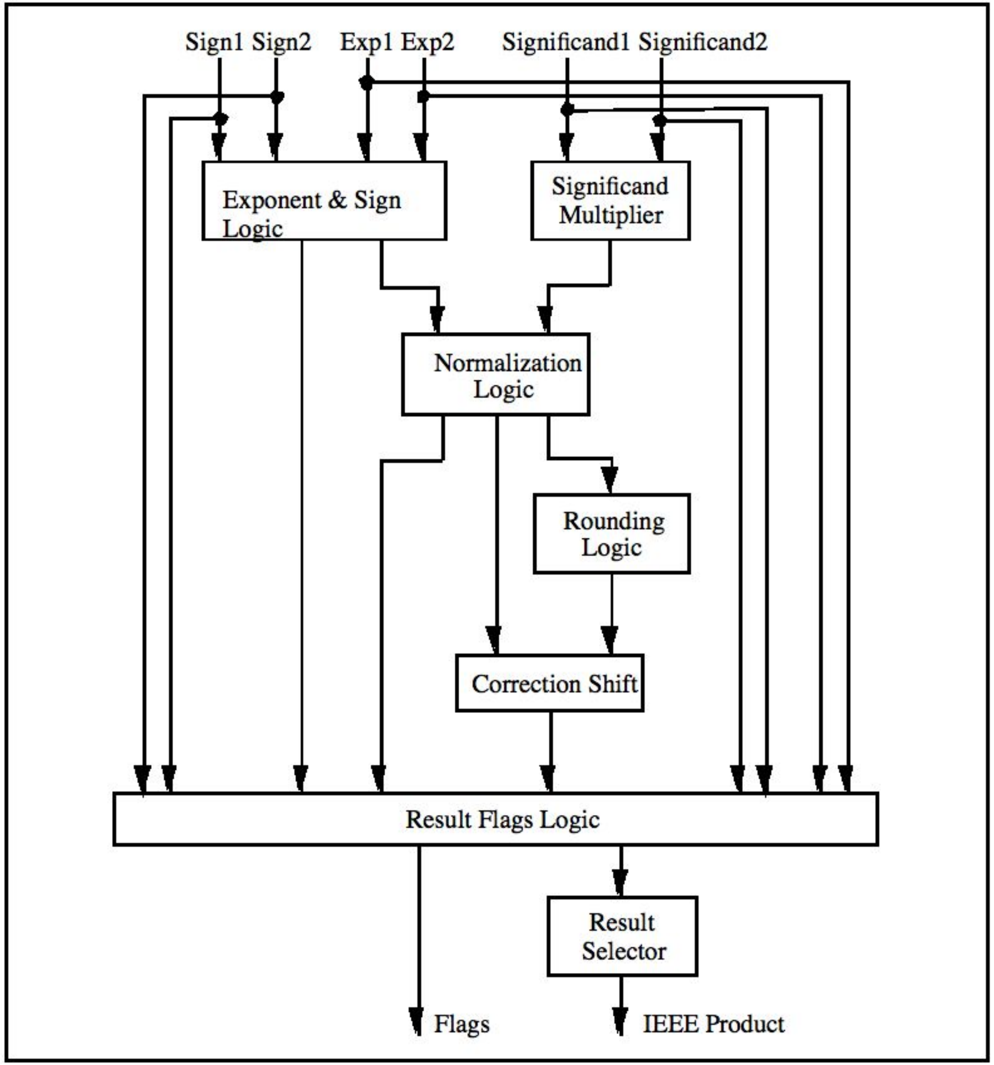
- $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$ (0.5×-0.4375)

1. Add exponents
   - Unbiased: $-1 + -2 = -3$
   - Biased: $(-1 + 127) + (-2 + 127) = -3 + 254 - 127 = -3 + 127$
2. Multiply significands
   - $1.000_2 \times 1.110_2 = 1.1102 \;\Rightarrow\; 1.110_2 \times 2^{-3}$
3. Normalize result & check for over/underflow
   - $1.110_2 \times 2^{-3}$ (no change) with no over/underflow
4. Round and renormalize if necessary
   - $1.110_2 \times 2^{-3}$ (no change)
5. Determine sign: +ve × –ve ⇒ –ve
   - $-1.110_2 \times 2^{-3} = -0.21875$

Floating Point Multiplication Algorithm



A Simple FP Multiplier



Floating-Point Addition

Consider a 4-digit decimal example

- $9.999 \times 10^{1} + 1.610 \times 10^{-1}$

1. Align decimal points
   - Shift number with smaller exponent
   - $9.999 \times 10^{1} + 0.016 \times 10^{1}$
2. Add significands
   - $9.999 \times 10^{1} + 0.016 \times 10^{1} = 10.015 \times 10^{1}$
3. Normalize result & check for over/underflow
   - 1.0015×10^{2}
4. Round and renormalize if necessary
   - 1.002×10^{2}

Floating-Point Addition

Now consider a 4-digit binary example

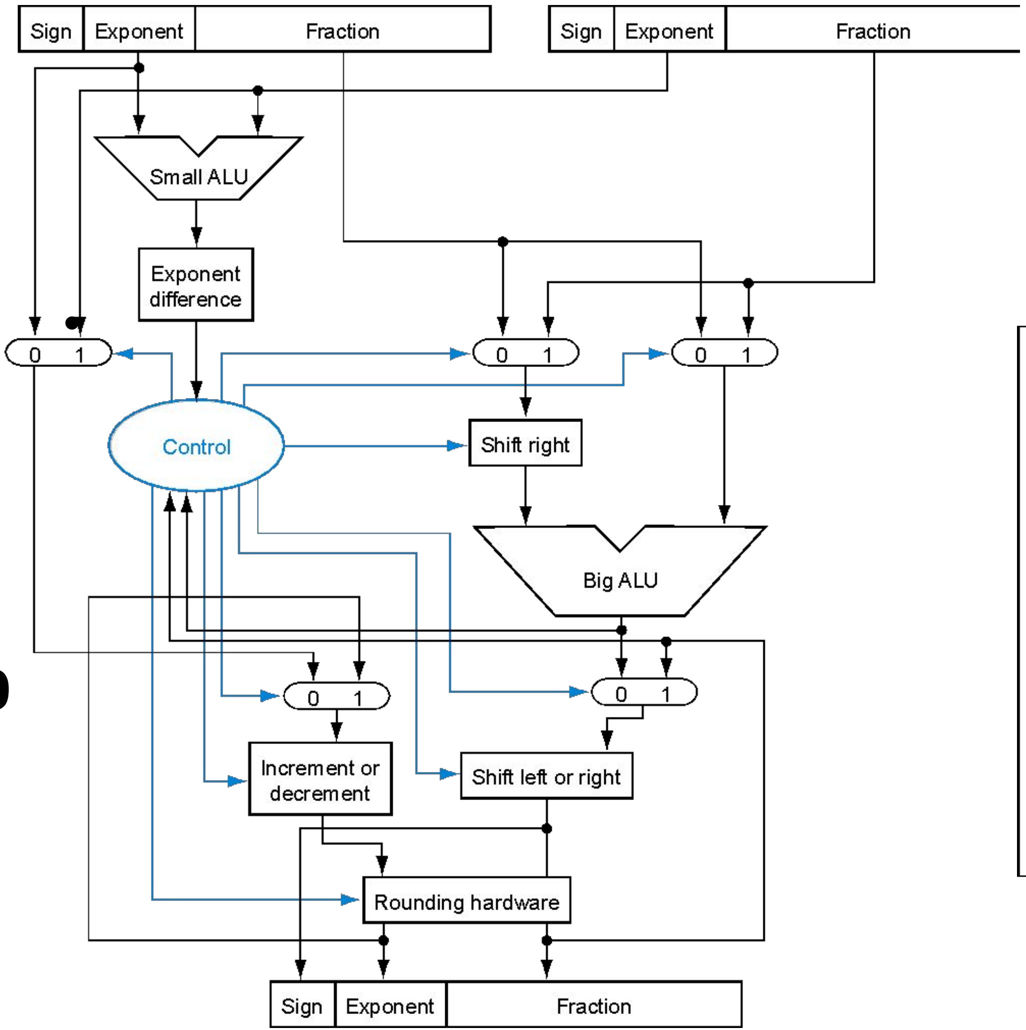
- $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ (0.5 + –0.4375)

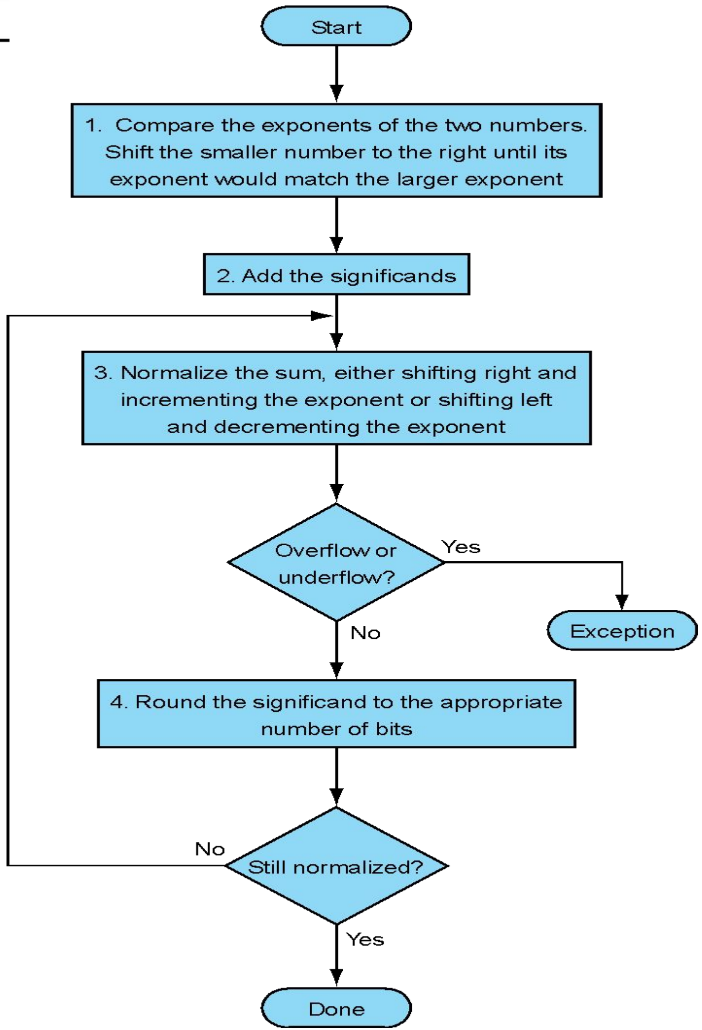
1. Align binary points
   - Shift number with smaller exponent
   - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$
2. Add significands
   - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$
3. Normalize result & check for over/underflow
   - $1.000_2 \times 2^{-4}$, with no over/underflow
4. Round and renormalize if necessary
   - $1.000_2 \times 2^{-4}$ (no change) = 0.0625

FP Adder Hardware

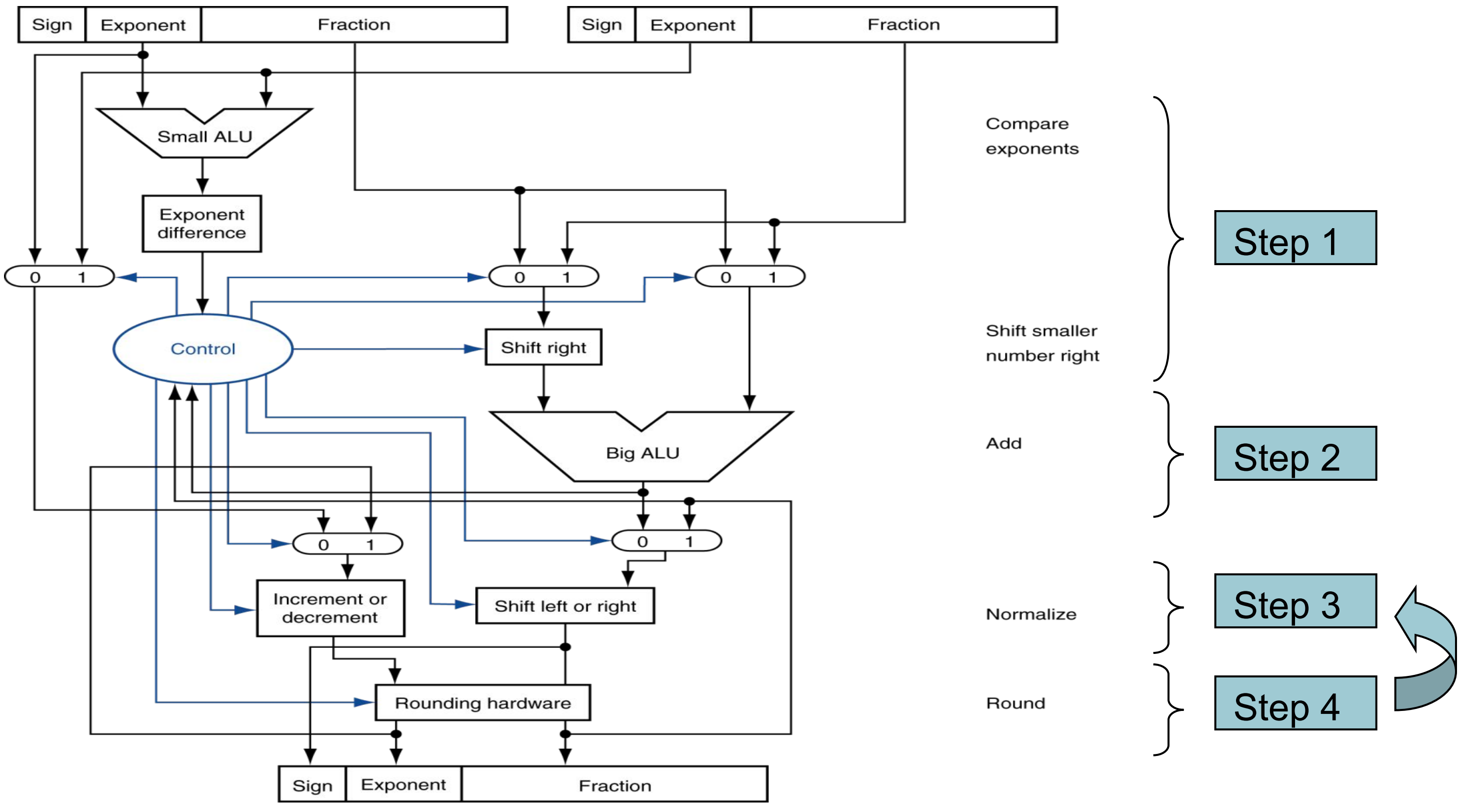
- Much more complex than integer adder
- Doing it in one clock cycle would take too long
  - Much longer than integer operations
  - Slower clock would penalize all instructions
  - Can be pipelined

Floating Point Addition



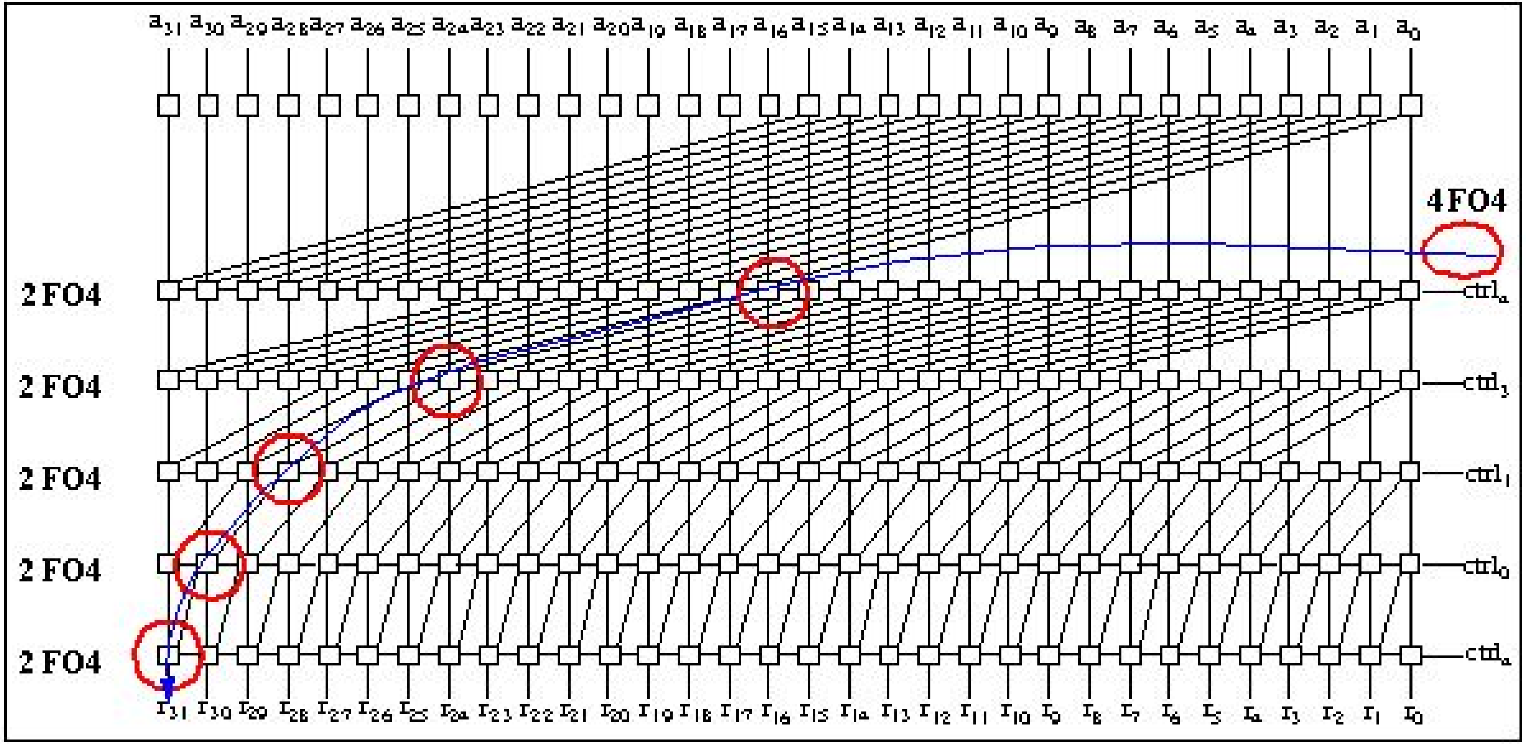


FP Adder Hardware

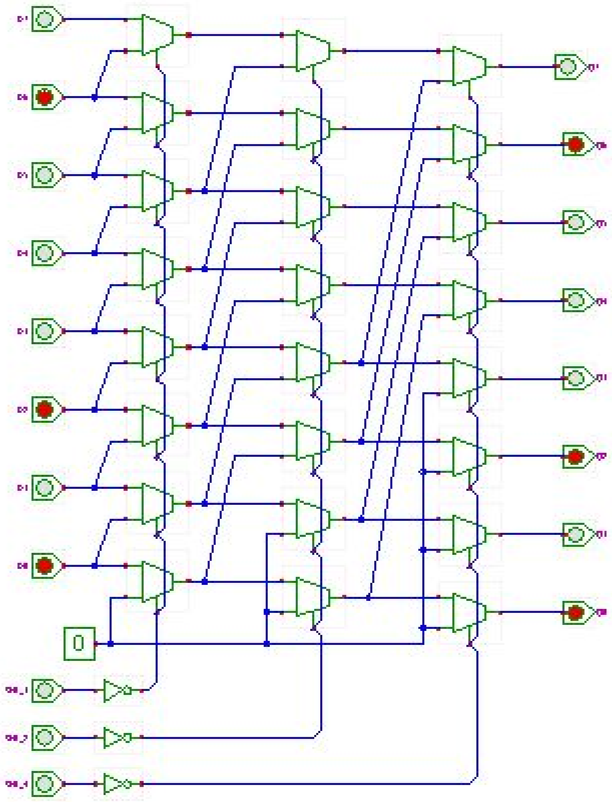


Barrel Shifters

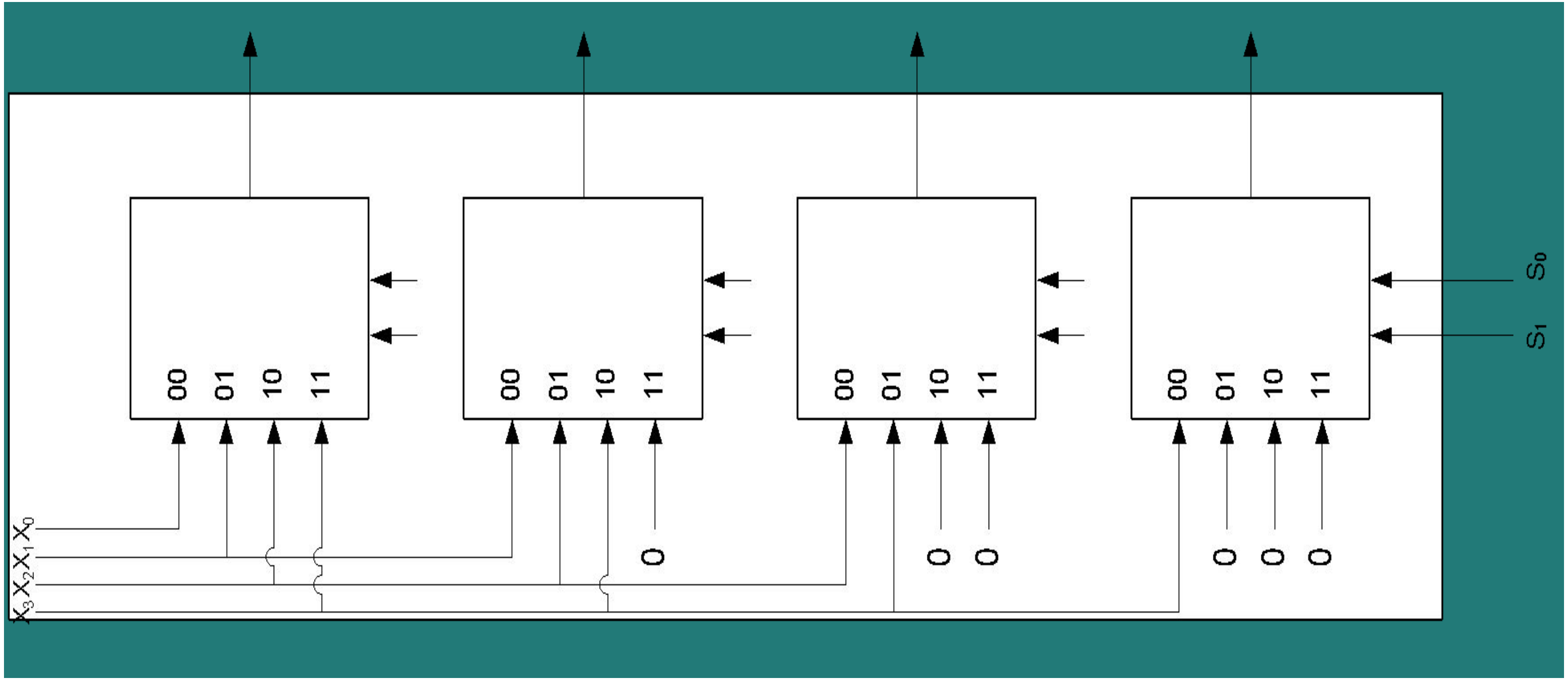
More Complex



Simple Barrel Shifter

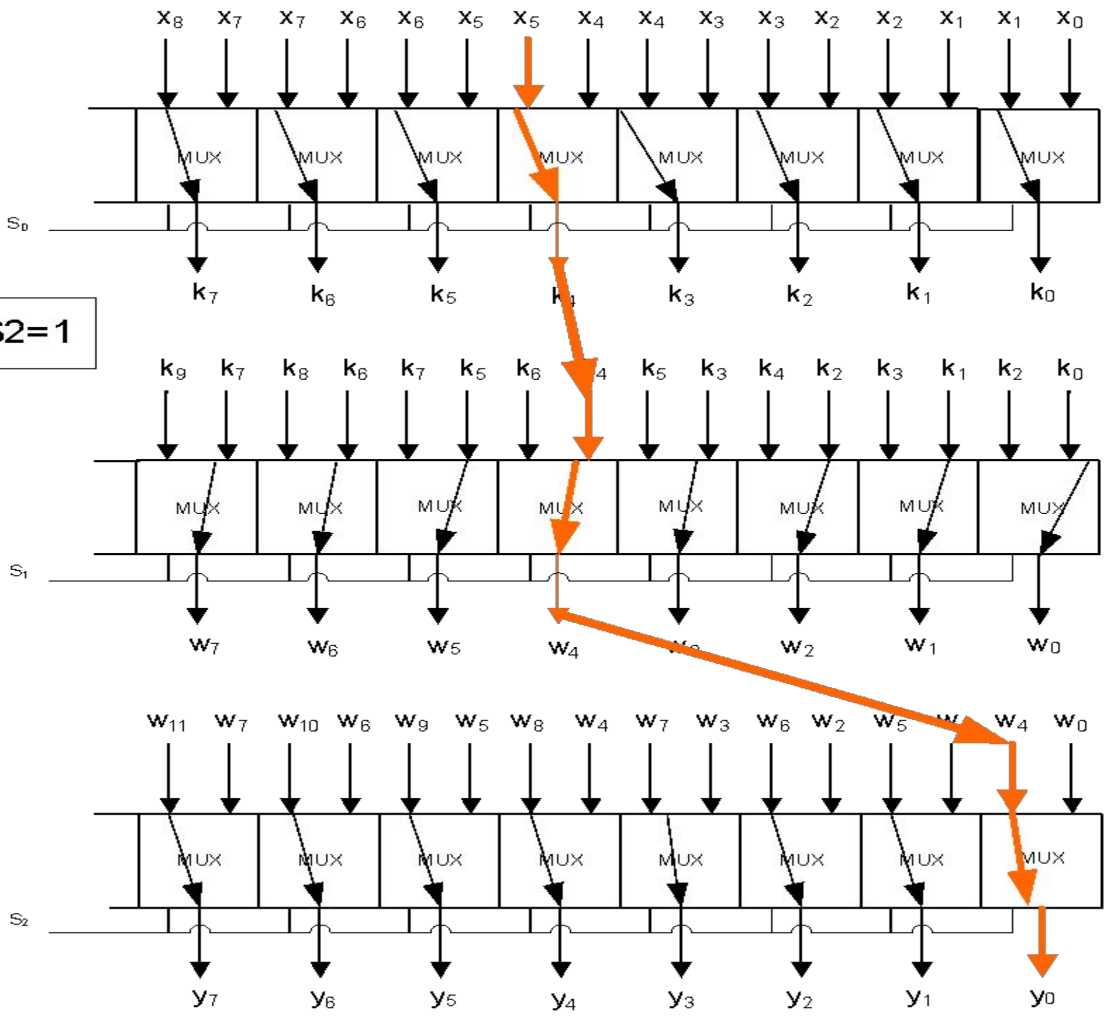


Right Shift Barrel Shifter with 4:1 MUX



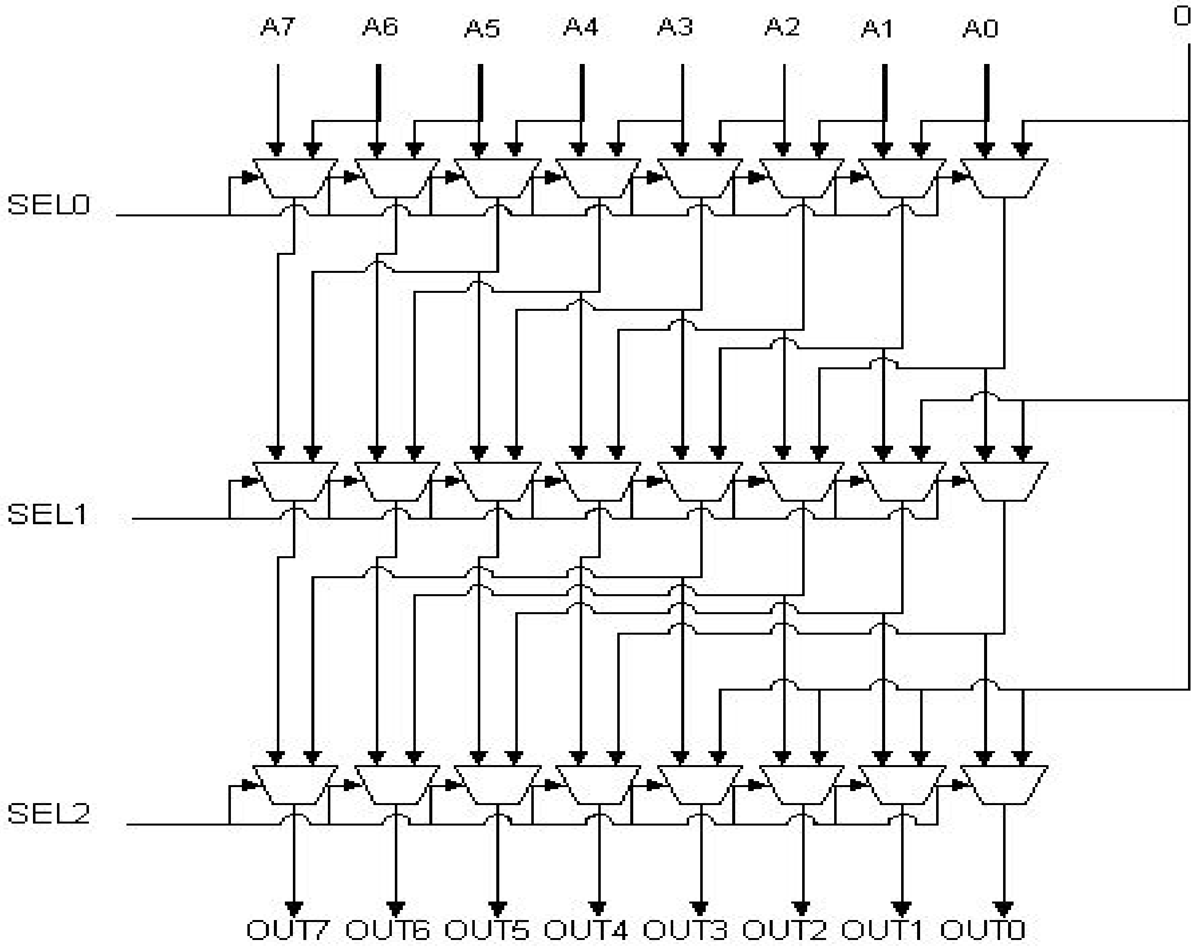
Paths of the Distributed Barrel Shifter

S0=1 S1=0 S2=1



Note that in this case if we have 8 bits of data then inputs to MUXes greater than 7 should be be set to a desired value

A Normalization Shifter for FP Arithmetic



Pentium FP Jokes

http://www.netjeff.com/humor/item.cgi?file=PentiumJokes