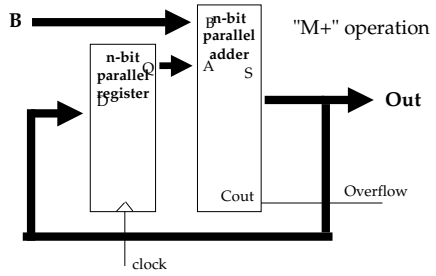
Outline

- Last time:
  - Adders and subtracters: Half adder, full adder
  - Adder organizations: ripple, carry look-ahead (CLA), carry bypass
  - Full adder as a full subtracter
  - Sign-magnitude
  - One's & Two's Complement
- This lecture:
  - Floating Point Formats
  - IEEE Formats
  - Multiplication
  - Division
  - The Arithmetic Logic Unit (ALU)
  - Bandwidth & Latency
  - Scheduling & Allocation, Pipelining

Parallel Addition with Accumulator



Fractions

- The value of a number in any radix is given by:

$$D = \sum_{i=-n}^{p-1} d_i . r^i$$

- Where *di* represents the ith digit of the number, *r* is the *radix* (2 for binary numbers) and there are *p* digits to the left of the decimal point and *n* digits to the right.

Example: 101.1012 =
=

Floating-Point

- Fixed point notation is useful for representing small-valued integers and scaled fractional numbers.
- Floating-point processors handle the scaling of the mantissa automatically
  - Additional hardware for Floating-Point is relatively complex and adds to the cost of the computer
  - But the operation of the machine is more efficient and we can put enough transistors on a single piece of silicon that the incremental cost is reasonable.
- Consider the dynamic range of values that can be expressed by a 16-bit fixed-point number. When interpreted as an integer, the value range is:
  +2^15 - 1 = +32,767

with the high-order bit representing the sign. Maximum negative value is:
  -2^15 = -32,768

Floating-Point

- So how can we represent a calculation like:
  34,200,000,000 × 0.0000000762
  with 16 bit numbers?
- Rewrite in scientific notation: (342 × 10^8) × (7.62 × 10^-8)
- Floating point is nothing more than scientific notationNote that both *magnitude* and *precision* are important aspects of scaling.

$$A = f \times r^e$$

Where *f* and *e* are both signed, fixed-point numbers and *r* is the radix (base). The *fractional* and *exponent* parts of floating-point numbers are sometimes called the *mantissa* and *characteristic*, respectively.

Floating-Point

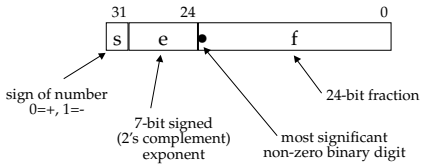
- The fraction and exponent can be represented in any form (1's, 2's complement; sign-magnitude)
- Consider a numerical example in the decimal (r=10) system:
  A = +0.0000068421 × 10 ^(+3)

we can write this in [fraction, exponent] form as:
  A = [+0.0000068421, +3] or [0.6842100000, -2]

- Precision is maintained by *normalizing* a number: shift the fraction (left or right) until the bit to the immediate right of the decimal point is the most significant, non-zero digit. Compute the exponent as appropriate.

Floating-Point

❍ Example of typical 32-bit single-precision binary floating-point format:



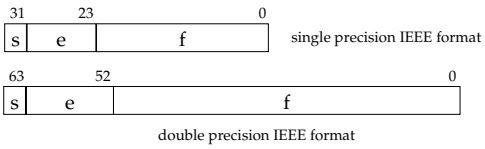
Floating-Point

To make all exponents positive, they are often stored in biased form when the floating-point number is computed:

e biased = e + 2^(m-1)

where *m* is the number of bits in the exponent field.

❍ This helps handle the "multiple zero" problem by defining zero to be : "zero fraction and biased exponent in its most negative form".



Floating-Point Operations

❍ Addition/Subtraction is then:

(1) Check for 0 operands

(2) Align the fractions by selecting the fraction with the smaller exponent. Then shift that fraction to the right and increment smaller exponent until both are equal.

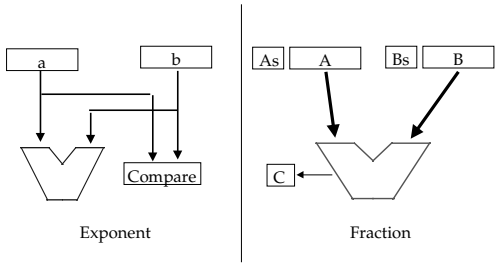
(3) Add or subtract (complement-add) the fractions

(4) Set the exponent of the result to the larger exponent.

(5) Normalize the result, if necessary

(6) Check for overflow or underflow.

Floating-Point Operations



IEEE Floating-Point Standard

❍ The IEEE Standard specifies:

(1) The format for floating-point operands

(2) The accuracy of arithmetic results for addition, subtraction, multiplication, division, square-root generation, obtaining remainders, and comparison operations.

(3) Conversion between integers and floating-point numbers

(4) Conversion between different floating-point formats as specified in the standard

(5) Conversion between binary floating-point numbers and decimal numbers

(6) Floating-point exceptions

❍ It has three major considerations:

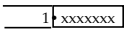
(1) The format of the floating-point numbers

(2) The accuracy of the arithmetic computations

(3) The handling of exception conditions

IEEE Floating-Point Standard: Terminology

❍ Significand: The portion of the binary floating-point number that consists of an explicit or implicit leading bit (0 or 1) to the immediate left of the radix point and a fraction to the right of the radix point.



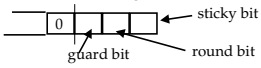
❍ Normal 0: Minimum exponent and 0 significand

❍ Denormalized: Minimum exponent, implicit or explicit leading bit 0 but number is not a normal 0.

❍ Unnormalized: Occurs in IEEE extended format

❍ Normalized: Non-zero number where leftmost significant bit is a 1. Normalization does not change the sign of a number.

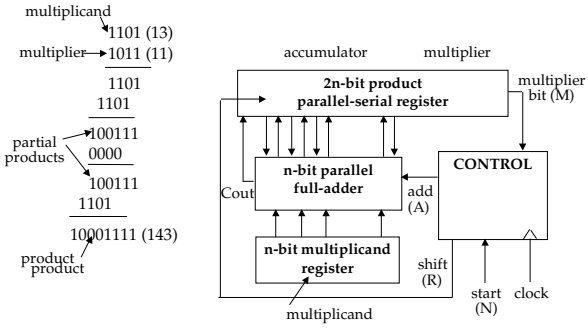
❍ Special guard digits are used in rounding:



But How to Compute Pij?

			a3	a2	a1	a0	
			b3	b2	b1	b0	
			a3b0	a2b0	a1b0	a0b0	
		a3b1	a2b1	a1b1	a0b1		
	a3b2	a2b2	a1b2	a0b2			
a3b3	a2b3	a1b3	a0b3				
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

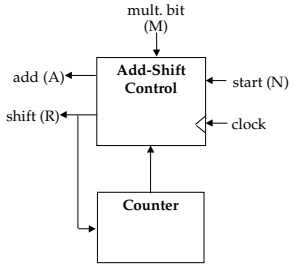
Multiplication



Multiplication

initial contents	00000	1011	(11)
add (M=1)	1101		(13)
after add.	01101	1011	
after shift	001101	101	
add (M=1)	1101		
after add.	100111	101	
after shift	0100111	10	
skip add (M=0)			dividing line between product and multiplier
after shift	00100111	1	
add (M=1)	1101		
after add.	10001111	1	
after shift	010001111		(143)

Multiplication



Multiplication: Some Variations

Consider:

$P = A \times B$

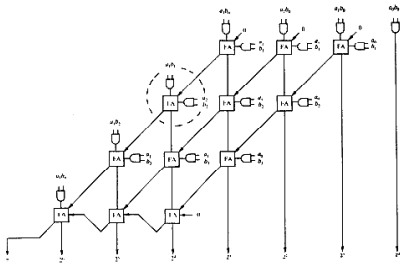
$= (A_H A_L) \times (B_H B_L)$

$= A_H \times B_H + A_H \times B_L + A_L \times B_H + A_L \times B_L$

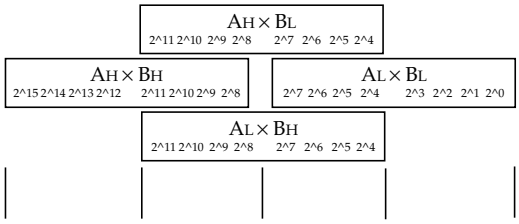
$= P_{HH} + P_{HL} + P_{LH} + P_{LL}$

where H represents high-order bits and L represents low-order bits.

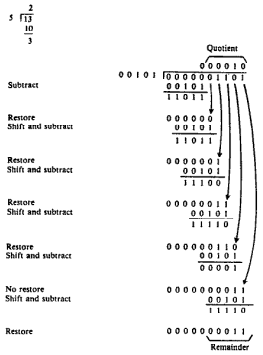
4x4 Non-Additive Multiply Module (NMM)



Wallace Tree Using 4x4 NMM



Fixed-Point Division



Divisor Reciprocation

f(x) = 1/x - B so f(x)=0 when x = 1/B

f′(x) = -1/x^2

So xi+1 = xi - ((1/xi) - B)/(-1/x^2))

xi+1 = xi (2 - Bxi)

But must form Bxi product every time (waste), so multiply both sides by B and let ai = Bxi :

xi+1 = xi (2 - ai)

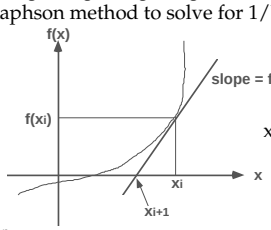
ai+1 = ai (2 - ai)

Convergence Division

- Generate reciprocal of the divisor by an iterative process and then use:

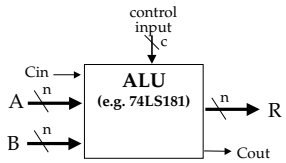
  A/B = A × (1/B)

- Use(great, great, great grand-uncle) Newton-Raphson method to solve for 1/B:



$x_{i+1} = x_i - f(x_i)/f'(x_i)$

Arithmetic Logic Unit (ALU)



Possible Operations:

A + B

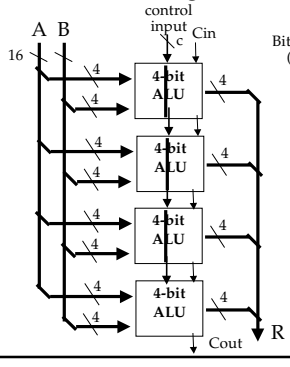
A - B

A + 1

-A

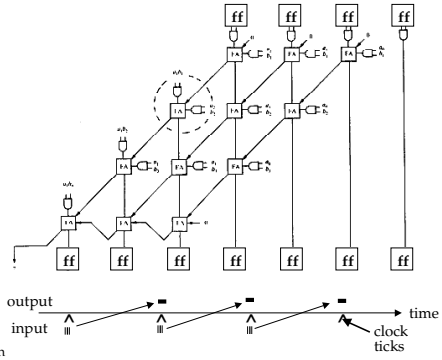
A && B, A || B, A′

Arithmetic Logic Unit (ALU)

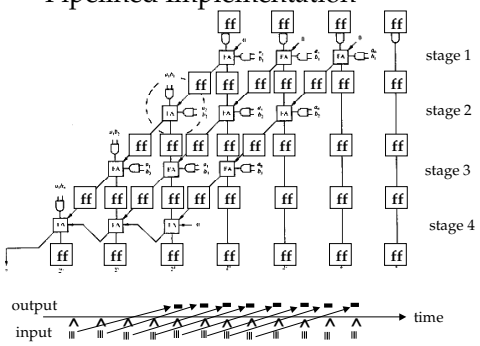


Bit-slice implementation (here, "nibble-slice")

NMM: Combinational Implementation



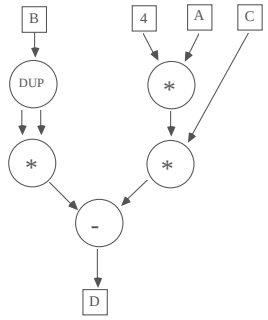
Pipelined Implementation



Bandwidth and Latency

- *Latency*: The minimum time to get the first result (sec.) - the one-time cost.
- *Bandwidth*: The maximum rate at which results can be produced in the steady-state (values/sec.) - the incremental cost
- For example, disc drives, RAM (normal, video), networks, signal-processors (e.g. HDTV, radar)

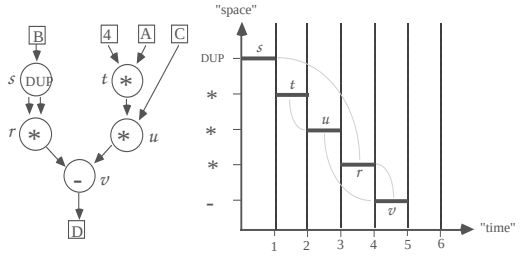
Simple Dataflow Description: B^2 - 4AC



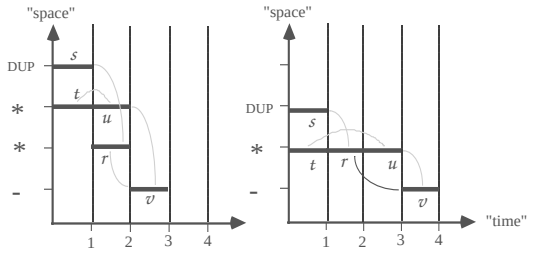
Scheduling of Functional Units

- Given a certain number of functional units (e.g. ALU's, RAM's), schedule a particular computation (from the HDL or software) onto a particular functional unit at a particular time relative to other operations.
- Analogous problem scheduling "wires"
- What about the values on the wires?

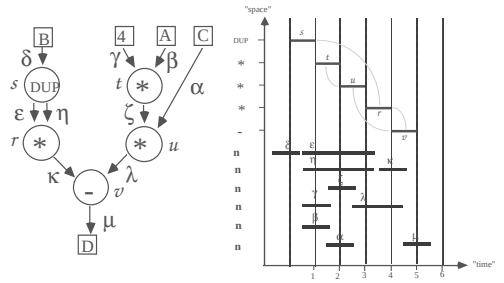
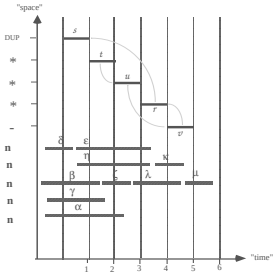
Allocation of Functional Units: Data Dependencies



Allocation of Functional Units: Time-Space Tradeoffs



Allocation of Functional Units: Sharing of Signal Links



Allocation of Functional Units: More Efficient Use of Links