Data Representation

Supplemental Notes

Yul Williams, D.Sc.

Goals

- At the conclusion of these notes, the student should be able to:
- Understand the basic concepts of computer arithmetic
- Identify the key methods of performing arithmetic functions in hardware
- Perform arithmetic operations using the key methods

Notes Outline

- Introduction to Computer Arithmetic
- Fixed Point Notation
- Unsigned Notation
- Signed Notation
- Binary Coded Decimal
- Special Arithmetic Hardware
- Floating Point Numbers
- Summary

Introduction to Computer Arithmetic

Arithmetic

- **a·rith·me·tic** (-r th m -t k) *n.* The mathematics of integers, rational numbers, real numbers, or complex numbers under addition, subtraction, multiplication, and division. – Webster's

Integers

- Integers are the set of all whole numbers negative, zero and positive
- The set of all integers is represented by the symbol Z
- Z = {-X, …,0,…,X}, where X is a whole number

Natural Numbers

- Natural numbers are the set of all whole numbers from zero and all positive integers
- The set of all natural numbers is represented by the symbol N
- N = 0,…,X}, where X is a whole number

Rational Numbers

- Rational numbers are of the form m/n where $m \in Z$, $n \in Z$ and $n \neq 0$
- They are called rational numbers because they are ratios of integers
- We commonly refer to numbers of this type as fractions
- The set of all rational numbers are represented by the symbol Q

Real Numbers

- The set of all integers and rational numbers
- The set of real numbers is represented by the symbol R
- $Z \subseteq Q \subseteq R$

Complex Numbers

- A complex number is the sum of a real number and an imaginary number
- An example of a complex number is shown in the expression: $a + bi$, where a and b are real numbers and i represents sqrt(-1)
- For the purposes of our discussion, there is no need to discuss complex numbers further

Arithmetic Operations

- Most CPU operations move or copy data
- In comparison, arithmetic operations happen less frequently than data movement operations
- Arithmetic operations are vitally important to realize full CPU functionality

Computer Arithmetic Issues

- As we explore the topic of computer arithmetic, we will examine:
  - Common number formats
  - Arithmetic algorithms
  - Hardware implementation of arithmetic algorithms
- Number representation
- Arithmetic operations
- Performance improvement techniques

Fixed Point Notation

Fixed Point Notation

- Any number in which the number of digits to the right of the decimal point does not change
- Consider an amount of money, for instance
- The fractions of dollars are always represented using 2 digits to the right of the decimal point

Integers as Fixed Point Numbers

- Fixed point notation is used by nearly all computers to represent integer values
- With integers, there are no digits to the right of the decimal point because integers do not have fractional components

Unsigned Notation

Unsigned Notation

- An unsigned number means that a number does not have a separate bit to represent the sign of the number
- Just because a number is unsigned does not mean that we cannot represent negative values
- In some unsigned notation schemes, the numbers may be positive or negative

Non-negative Notation

- Every number is this scheme is treated as a natural number
- This means that the number may be assigned a value of zero or any positive value

Two's Complement Format

- Both positive and negative values are represented in this number format
- For n-bit numbers, values may range from -2^{n-1} to $2^{n-1}-1$

Two’s Complement Cont’d

- In this format, the negative numbers have a 1 in the most significant position in the number
- Positive numbers have a 0 in the most significant bit position in the number

Two’s Complement

4-bit sign plus magnitude	
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000
–0	1000
–1	1001
–2	1010
–3	1011
–4	1100
–5	1101
–6	1110
–7	1111

4-bit 2's complement	
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
–1	1111
–2	1110
–3	1101
–4	1100
–5	1011
–6	1010
–7	1001
–8	1000

2's complement uses only a single representation of zero; sign-plus-magnitude has two.

2's-complement numbers form a regular binary count sequence from full-scale negative to full-scale positive, with a "rollover" or "wraparound" at zero.

2's complement always has one more negative value than it has positive values.

Source: Fundamentals of Embedded Software, Daniel W. Lewis

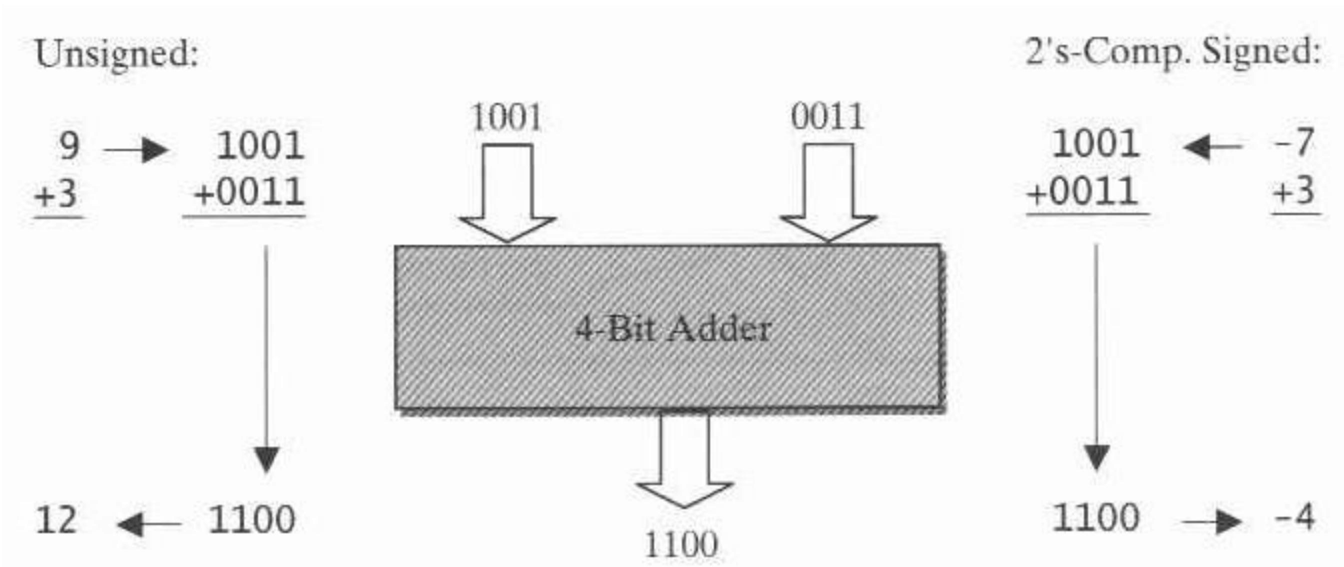
Values in Unsigned Notation

Binary Representation	Unsigned Non-Negative	Unsigned Two's Complement
0000 0000	0	0
0000 0001	1	1
…	…	…
0111 1111	127	127
1000 0000	128	-128
1000 0001	129	-127
…	…	…
1111 1111	255	-1

Addition and Subtraction

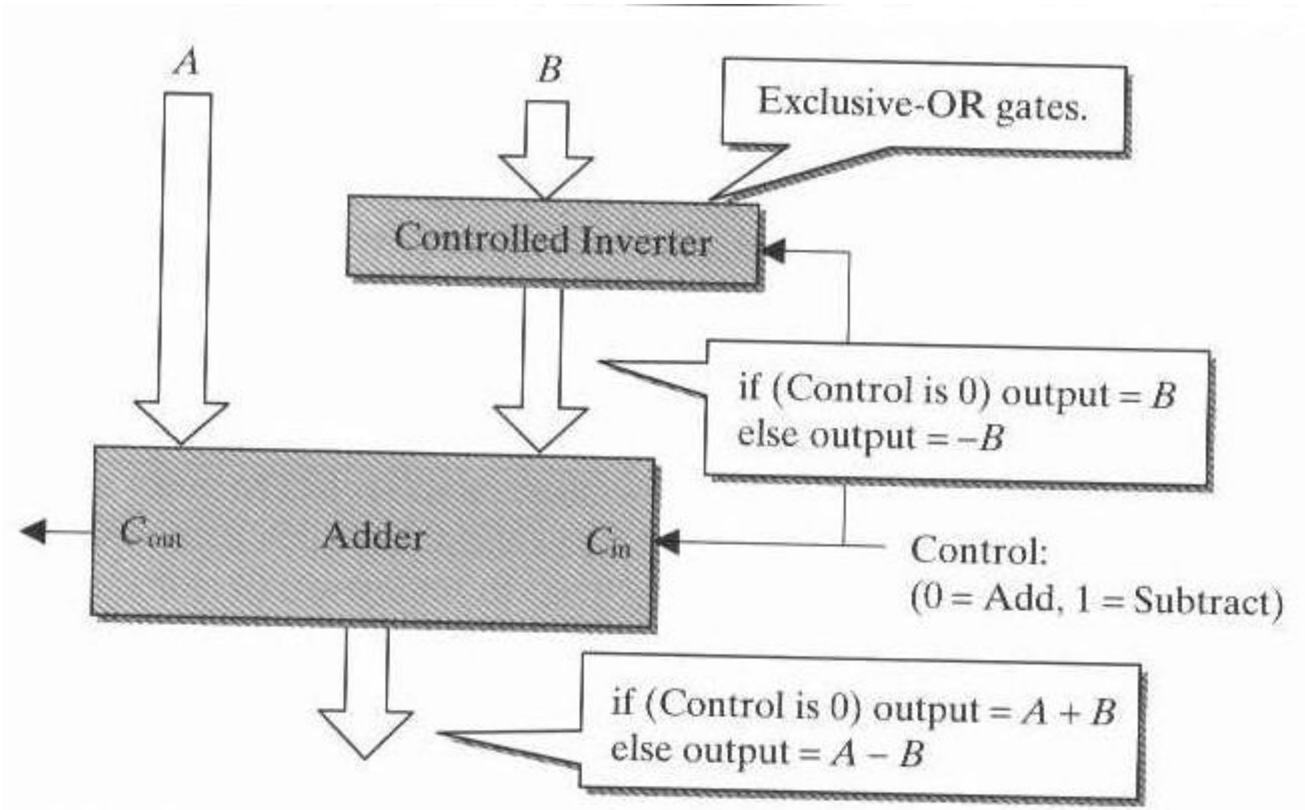
Unsigned Notation

Adding Two 4-bit Numbers



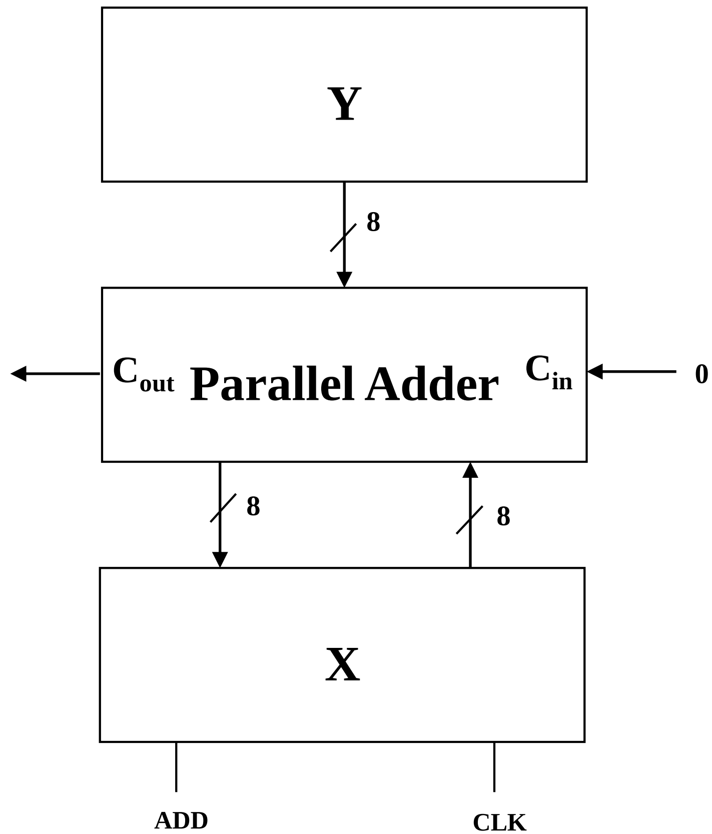
Source: Fundamentals of Embedded Software, Daniel W. Lewis

Performing Subtraction With an Adder?



Source: Fundamentals of Embedded Software, Daniel W. Lewis

X ← X + Y Implementation



Arithmetic Overflow

- What happens when a result from an arithmetic operation is larger than the register that holds the result?
- An overflow condition occurs
- A extra bit generates a carry out. This bit can be used to signal the rest of the system that an overflow has occurred

Overflow Condition for Unsigned Non-Negative Numbers

255 1111 1111 (8-bit value)

+ 1 0000 0001 (8-bit value)

------ -------------

256 10000 0000 (9-bit result)

9-bit result cause a carry out to be generated

The Double Whammy!

- In two’s complement notation, an overflow can occur at either end of the numeric range

Two's Complement
The Positive End

127	**0111 1111 (8-bit value)**
+ 1	**0000 0001 (8-bit value)**
------	**-------------**
128	**1000 0000 (8-bit result)**

In two's complement, this result is –128 (not +128)

Two's Complement
The Negative End

-128	**1000 0000 (8-bit value)**
+ (-1)	**1111 1111 (8-bit value)**
------	**-------------**
-129	**0111 1111 (8-bit result)**

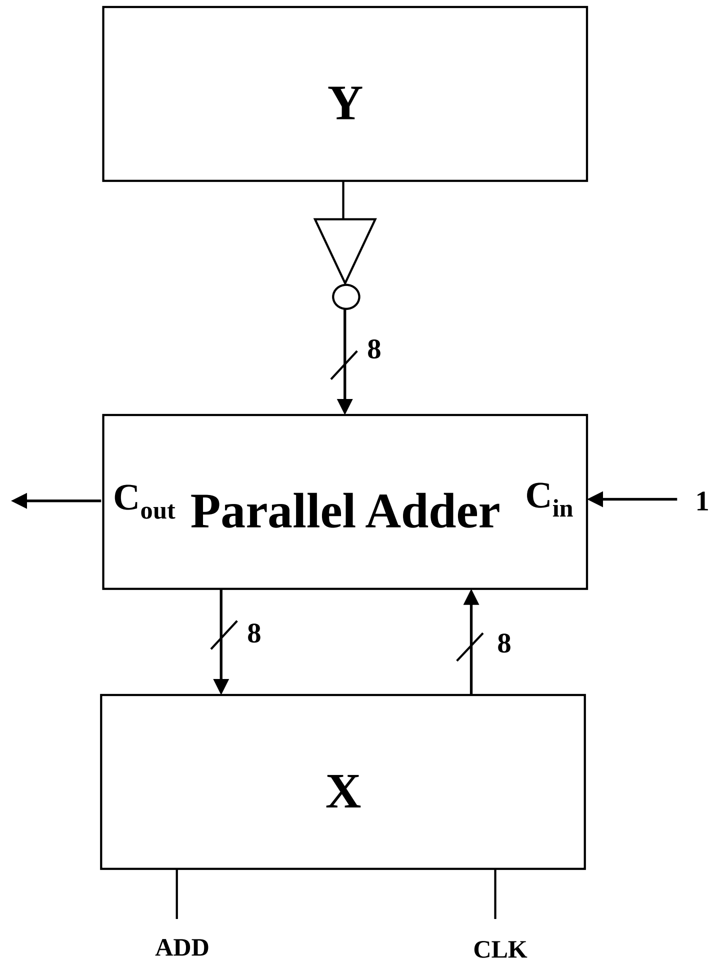
The result is +127 (not –129)

Overflow Condition in Unsigned Two's Complement Addition

127	**0111 1111 (8-bit value)**
+ 1	**0000 0001 (8-bit value)**
------	**-------------**
0	**10000 0000 (9-bit result)**

Overflow condition can occur only when two numbers are added. There is no way to represent + 128 in two's complement when using 8-bit registers.

X ← X - Y Implementation



Overflow Condition in Unsigned Two’s Complement Subtraction

1	**0000 0001 (8-bit value)**
- 2	**1111 1110 (8-bit value)**
------	**-------------**
-1	**01111 1111 (9-bit result)**

Overflow condition can occur only when the two numbers are added. The result of the operation is 255. This is incorrect.

Multiplication Techniques

Unsigned Notation

Iterative Multiplication

Objective: Multiply 16 x 32

Method: Iterative method.

Let X = 16; Multiplicand

Let Y = 32; Multiplier

Let Z = X x Y; Product

Algorithm:

```
Z = 0;
for i = 0 to Y DO
   Z = Z + X;  // Add X to itself Y times
end for
```

- **In the simplest sense, multiplication may be considered as a series of additions to yield a final product.**

- **Although this approach is effective in producing the correct results, it is time-dependent. As either the number X or Y is increased, the time to complete the final result is increased.**

Shift-Add Multiplication

Objective: Multiply 16 x 32

Method: Shift-add method.

Let X = 16; Multiplicand

Let Y = 32; Multiplier

Let Z = X x Y; Product

Calculation:

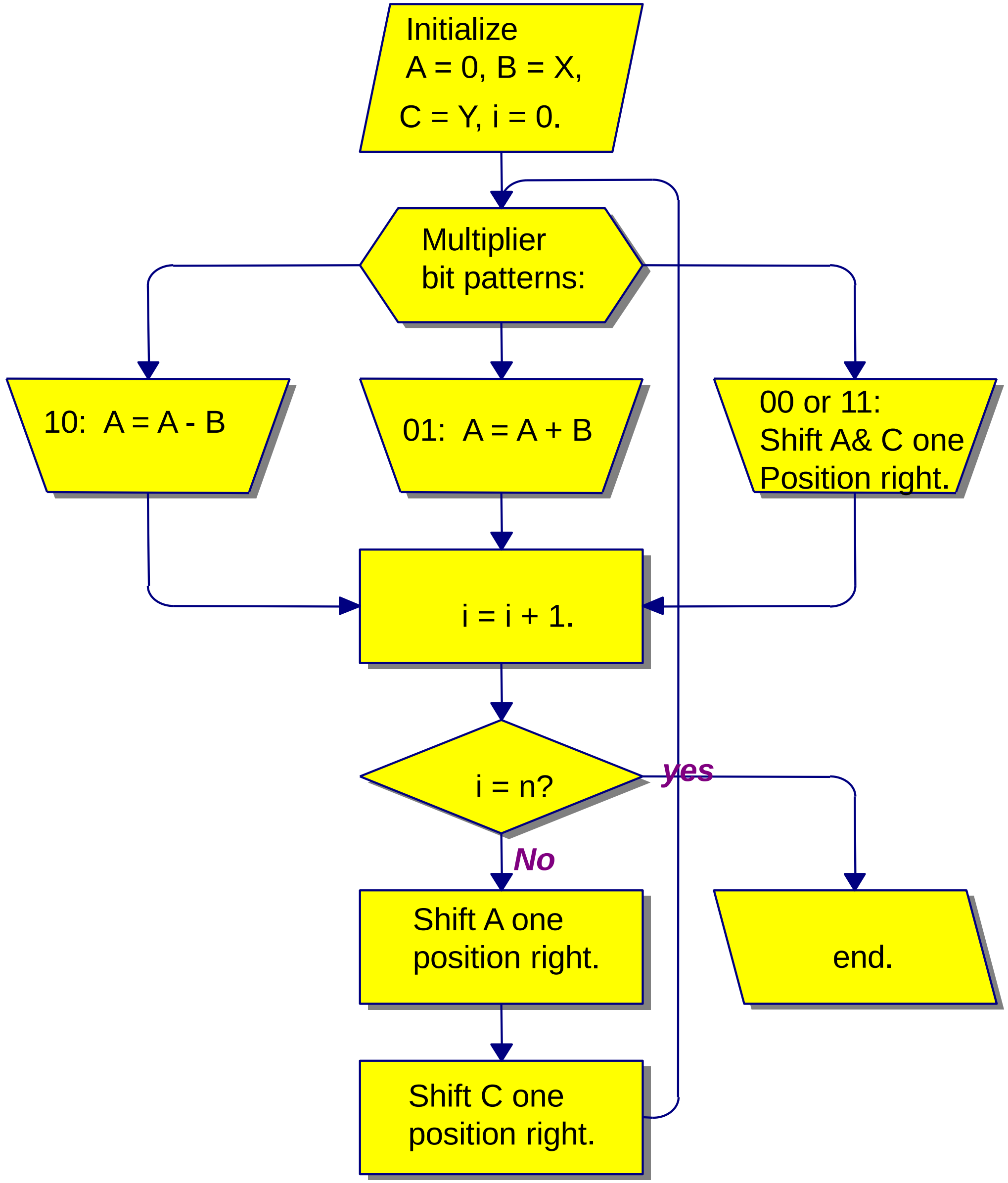
```
   16
x  32
------
   32    Partial Product 1
  48     Partial Product 2
-------
  512    Final Product
```

- **The shift-add method is the approach that is used by people performing manual multiplications on paper. This is the method that we were all taught is school.**
- **This approach is effective in producing the correct results. In this method, partial products are produced and then summed at the end to produce a final product.**
- **The final product is a sum of the partial products.**

Multiplying Positive and Negative Values

- Until now, we have only considered multiplying positive values
- A well-known algorithm for producing good results from multiplying two's complement numbers is called Booth's Algorithm
- Let's take a look at how it works…

Booth's Algorithm



Division Techniques

Unsigned Notation

Division

- Division may be viewed as a series of subtractions
- This is an iterative approach (just like in the case of iterative additions for the multiplication technique)
- The following is an example of the iterative division algorithm

Iterative Division

Objective: Divide 512 x 8

Method: Iterative method.

Let X = 512; Dividend

Let Y = 2; Divisor

Let Z = X ÷ Y; Quotient

Algorithm:

```
Z = 0;
while (X ≥ Y) loop
   Z = Z + 1;
   X = X – Y; //Subtract Y from X, Z times
end while
```

Shift-Subtract Division

Objective: Divide 512 x 2

Method: Shift-substract method.

Let X = 512; Dividend

Let Y = 2; Divisor

Let Z = X ÷ Y; Quotient

Calculation:

```
   256
2 | 512
    4
    ------
    11
    10
   --------
     12
     12
```

Signed Notation

Signed Magnitude Notation

- Signed-magnitude notation has two parts:
  - The sign. A 0 indicates positive and a 1 indicates negative
  - The magnitude – an n-bit value that holds the absolute value of the number
- Example
  - +5 and –5 have different signs but their magnitudes are equal

Observation of Signed Notation

- Although this notation is very human-readable, it requires more hardware to hold the sign and the magnitude values
- Signed notation is more complicated than unsigned notation because their respective signs determine how the numbers should be handled

Addition and Subtraction of Signed-Magnitude Numbers

Operation	X_s	Y_s	*AS*	*PM*	$X = 3, Y = 5$	$X = 5, Y = 3$
$(+X) + (+Y)$	0	0	0	0	$(+3) + (+5) = +8$	$(+5) + (+3) = +8$
$(+X) - (+Y)$	0	0	1	1	$(+3) - (+5) = -2$	$(+5) - (+3) = +2$
$(+X) + (-Y)$	0	1	0	1	$(+3) + (-5) = -2$	$(+5) + (-3) = +2$
$(+X) - (-Y)$	0	1	1	0	$(+3) - (-5) = +8$	$(+5) - (-3) = +8$
$(-X) + (+Y)$	1	0	0	1	$(-3) + (+5) = +2$	$(-5) + (+3) = -2$
$(-X) - (+Y)$	1	0	1	0	$(-3) - (+5) = -8$	$(-5) - (+3) = -8$
$(-X) + (-Y)$	1	1	0	0	$(-3) + (-5) = -8$	$(-5) + (-3) = -8$
$(-X) - (-Y)$	1	1	1	1	$(-3) - (-5) = +2$	$(-5) - (-3) = -2$

- **AS = 0 for addition and AS = 1 for subtraction**
- **X_s is the sign bit for the number X,**
- **Y_s is the sign bit for the number Y and**
- **PM = $X_s \oplus AS \oplus Y$**

Multiplication and Division of Signed-Magnitude Numbers

- The procedures to multiply and divide signed numbers are nearly identical to those of the unsigned number notation
- The only exception is that the sign bit must be set for each operation

Binary Coded Decimal

BCD

Binary Coded Decimal

- When encoding decimal data, we sometimes use a notation referred to as Binary Coded Decimal or BCD
- In BCD notation, each decimal digit requires 4 bits of binary data
- Each byte of data can represent 2 decimal digits. The high nibble and the low nibble of the byte are each 4 bits long

BCD Notation

In order to represent the decimal number 41, the BCD representation becomes 0100 0001. Since we have 4 binary digits, our range of number that each nibble can represent is from 0 to 15. Since 4 binary digits can represent 16 numbers, each nibble covers the range of digits in the Hexadecimal number system. The hexadecimal number system is a base 16 number system that is frequently used in digital design and in programming operations. In BCD, however, we use only the first 10 numbers are 0 through 9 (just as in the decimal number system). Numbers 11 through 15 are ignored.

Binary	Decimal	BCD
0	0	0000
1	1	0001
10	2	0010
11	3	0011
100	4	0100
101	5	0101
110	6	0110
111	7	0111
1000	8	1000
1001	9	1001
1010	10	1010

More on BCD

- BCD is a signed notation
- Its values may be positive and negative
- Its digits are not stored in two’s complement format. One bit is used to store the sign of the number

BCD Addition and Subtraction

- The algorithms for performing addition and subtraction are very similar to the signed-magnitude notations
- We simply need to change the hardware to account for the BCD representation. Remember that each BCD digit is represented by 4 binary bits.
- Another change is the manner in which the complements are calculated.

The Nine's Complement

Objective: Calculate the nine's complement of 631.

Calculation:

```
   631
 - 999
--------
   368   (nine's Complement of 631)
```

Note: Adding 1 to the nine's complement result yields the ten's complement. The Ten's complement in BCD is equivalent to the two's complement in binary

Multiplication and Division of BCD Numbers

- The procedures to multiply and divide signed numbers are nearly identical to those of the unsigned number notation
- More extensive changes to the hardware implementation are required
- The nine's complement hardware is required in addition to changes to the shifting hardware

Special Arithmetic Hardware

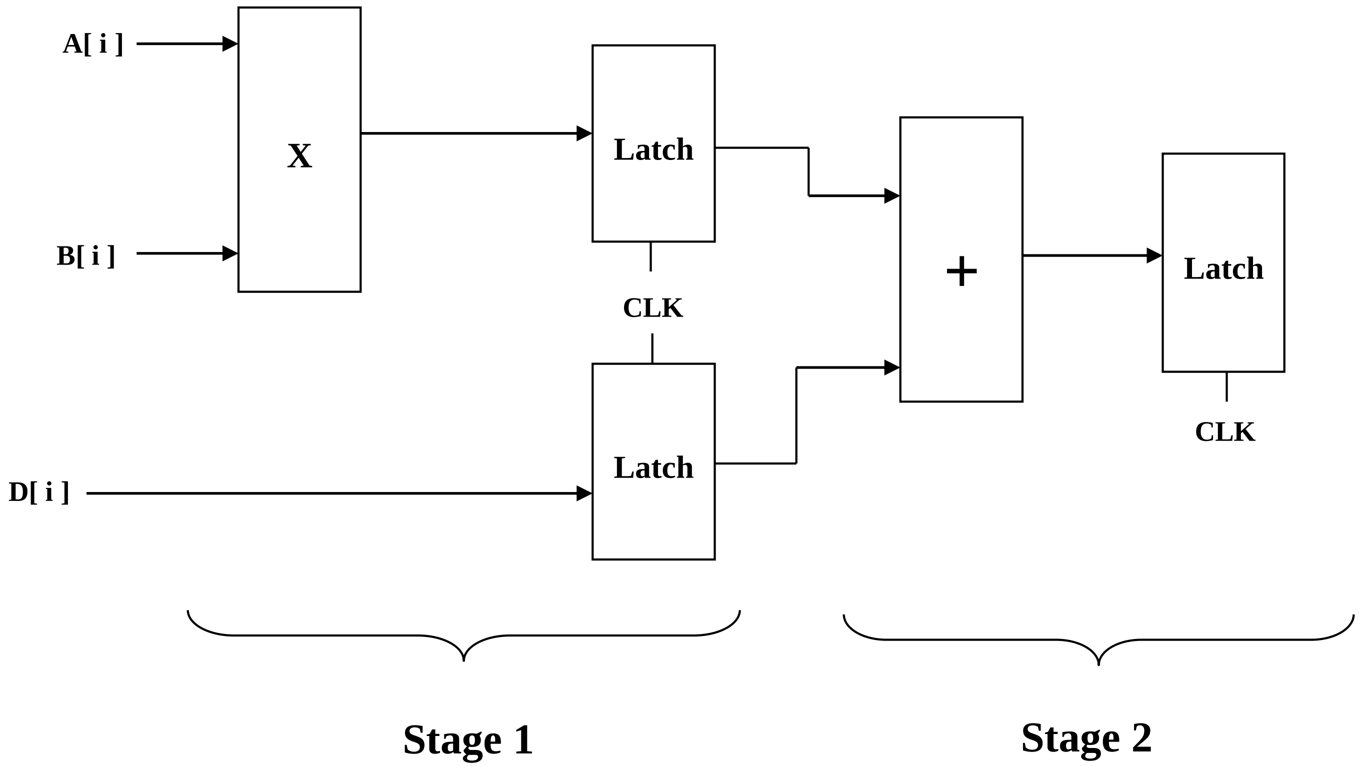
Pipelining

Specialized Hardware

Arithmetic Pipeline

- Data enters a stage of the pipeline where an arithmetic operation is performed
- The results are then passed to a subsequent stage
- Concurrently, a new set of data enters the first stage
- Arithmetic operations are overlapped for each computation
- This increases the throughput. The number of operations performed per unit time.

Pipelining Example



Speedup

- Speedup is defined to be the amount of time needed to process n pieces of data using a single processing element, divided by the time required by a k-stage pipeline
- T_1 is the time required by a single processing element to process the data
- T_k is the clock period of the k-stage pipeline. If the stages have different minimum clock periods, T_k is the largest of these periods

Example Speedup Calculation

Given a for-loop: FOR i = 1 to 100 DO {A[i] $\leftarrow$ (B[i] * C[i]) + D[i] }

Assumption: A non-pipelined uniprocessor takes 20 ns to calculate A[i]. This means that for 100 iterations, the total time required is 100 * 20 ns, or 2000 ns.

For a 2-stage pipeline, where stage 1 calculates the (B[i] * C[i]) component of A[i] and stage 2 calculates the (BC_Result[i]) + D[i]).

Let T_1 = 20 ns, the time required to calculate each A[i].

Let T_k = 10 ns, the clock period of each pipeline stage .

Then, $$S_n = nT_1 / [(k + n - 1) T_k]$$

$$= (100 * 20 \text{ ns}) / [(2 + 100 - 1) * 10 \text{ ns}] = 1.98$$

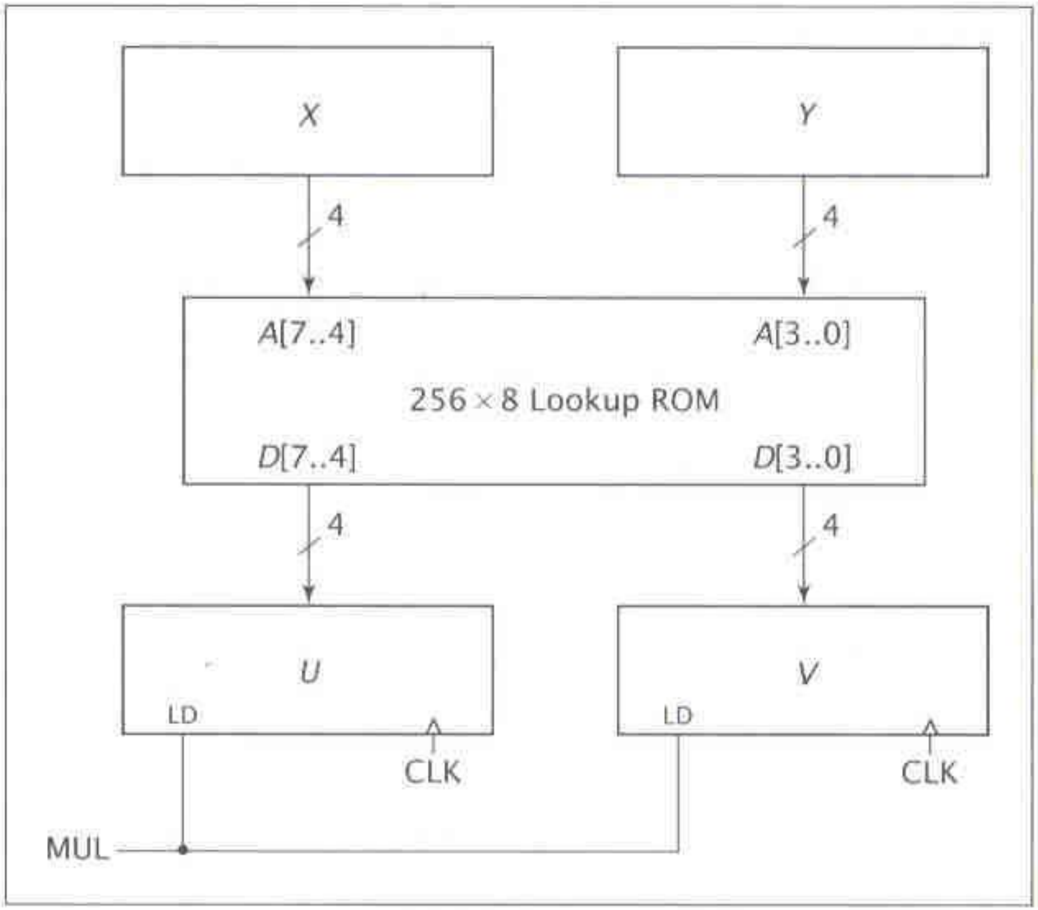
Lookup Tables

Specialized Hardware

Lookup Tables

- Theoretically, any combinational logic may be implemented in ROM
- As we observed in the design of the microsequencer for the Very Simple CPU, the ROM was used to hold micro-operations as well as raw signal values
- A lookup table allows pre-calculated data to be stored in ROM and accessed when the known components are provided as inputs

Lookup Table Example



Lookup Tables: Pros and Cons

- Pros
  - This approach is much quicker than the shift-add or shift-subtract methods
  - It has less hardware complexity than the other approaches
- Con
  - The size of the ROM grows rapidly as the number of operands increases

Wallace Trees

Specialized Hardware

The Wallace Tree

- The Wallace Tree is a combinational circuit used to multiply two numbers
- It uses a lot more logic than the shift-add multipliers and produces products faster
- The Wallace tree uses carry-save adders and a single parallel adder
  - Carry-save adders can add 3 values simultaneously
  - It can output a sum and a set of carry bits

Example: 3x3 Wallace Tree

$x =$ 111

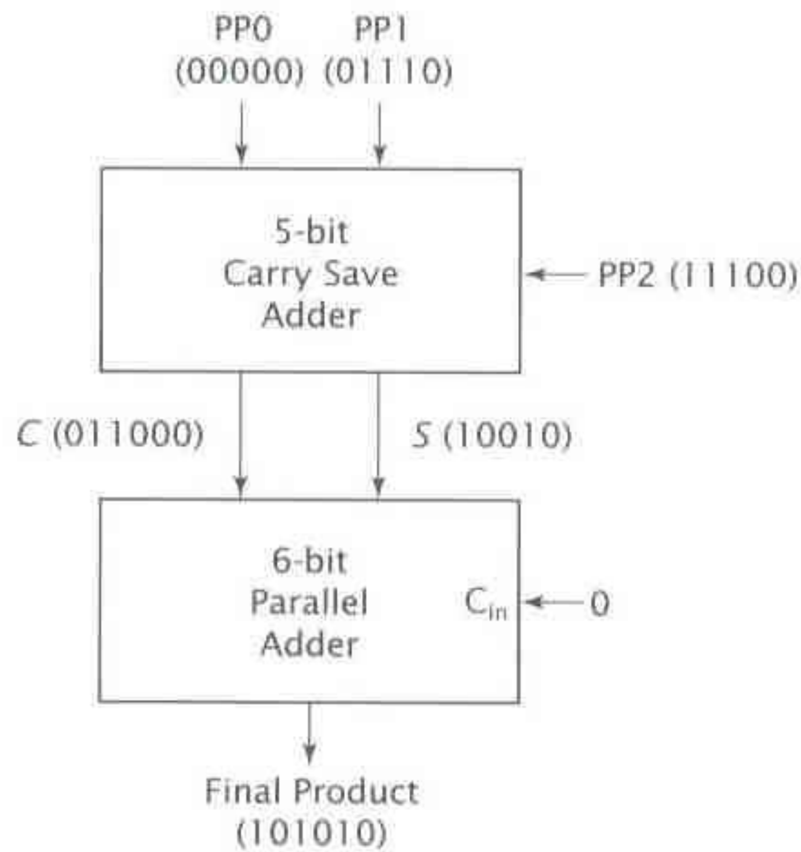
$y =$ 110

000 ← *PP0*

111 ← *PP1*

111 ← *PP2*

101010 ← *Final sum calculated*



Floating Point Numbers

Floating Pointer Format and Scientific Notation

- Floating point format is very similar to scientific notation
- A number in scientific notation number has:
  - A sign
  - A significand or mantissa
  - An exponent
- Number in scientific notation may be expressed in many different ways
- -1234.5678 = -1.2345678 x 10^3
- -1234.5678 = -1234567.8E-3

Floating Pointers

- Floating point numbers must be normalized. Each number's significand is a fraction with no leading zeroes
- This works well for all values except 0 and $\pm\propto$. Special values are assigned to represent them.
- Not a Number or NaN represents values such as $\propto \div \propto$ or sqrt(-1)

Precision

- Precision is defined to be the number of digits in the significand
- A computer that uses 8 bits in the its significand is defined as having 8-bit precision
- Double precision means that double the number of bits are present within the significand as compared to a single precision value on the same machine.

Standardizing Floating Point Notation

- In an effort to make floating point notation uniform across all computers, the Institute of Electrical and Electronics Engineers (I.E.E.E) created a standard for floating point notation.
- This standard was named the I.E.E.E. Standard 754
- It is widely used on nearly all modern computers

I.E.E.E. 754

Institute of Electrical and Electronics Engineers (I.E.E.E.)

IEEE Std. 754 Floating Point Number Representation

- Current standard on almost all machines.
- Two forms specified by the standard
  - *Single-precision*
    - One 32-bit word: 24 bit significand, 8 bit exponent.
    - Range: $1.17549435 \times 10^{-38} \ldots 3.40282347 \times 10^{+38}$
  - *Double-precision*
    - Two 32-bit words: 53 bit significand, 11 bit exponent.
    - Range: $2.2250738585072014 \times 10^{-308} \ldots 1.797693134862315 7 \times 10^{+308}$

Biasing

- A signed exponent may be represented in 2's complement
- An 8-bit exponent's minimum value 10000000 = -2^7 = 128 and its maximum value = 01111111 = (2^7-1) = +127
- Such a system will accommodate binary floating point numbers within the approximate range of $2^{\pm 127}$

FP Representation: IEEE 754

$$1.001101110 \times 2^5 =$$

significand sign & magnitude (omitting leading 1)

0 10000100 00110111000000000000000

exponent, biased representation

<u>Bias of 127</u>

00000000	= -127
	…
01111110	= -1
01111111	= 0
10000000	= 1
	…
11111111	= 128

FP Representation: IEEE 754

- FP ops easier if treat significand sign separately.
- FP ordering ≈ binary ordering.
  - Except for significand sign bit
  - Simplifies ordering comparison
- Omitting leading 1 saves one bit of representation.
  - But makes it impossible to represent 0.0

IEEE 754 Special Numbers

Exponents 00000000 and 11111111

- Don't represent -127 and 128 as expected
- Used as indicators of special values
  - +0.0, -0.0 (+0.0 = all zero bits)
  - $+\infty$, $-\infty$
  - NaN: "Not a number"
  - Denormalized numbers – more near-zero fractions

$(+1.0 \times 10^{+38})^2 = +\infty$

$+1.0 \div +0.0 = +\infty$

$+1.0 \div -0.0 = -\infty$

$+0.0 \div +0.0 = \text{NaN}$

$+\infty - +\infty = \text{NaN}$

$\sqrt{-1} = \text{NaN}$

Expressing a Floating Point Number using IEEE 754 Standard

Value of a real number = (-1)$^{\text{sign bit}}$ * 2$^{(\text{biased exponent} - \text{bias constant})}$ * actual significand

+19.5 = (-1)^{0} * 2$^{(4 - (-127))}$ * 001 1100 0000 0000 0000 0000, single precision

$$\mathbf{\underset{\text{Sign bit}}{0}\ \underbrace{1000\ 0011}_{\text{Exponent}}\ \underbrace{001\ 1100\ 0000\ 0000\ 0000\ 0000}_{\text{Significand}}}$$

Summary

- Computer arithmetic is essential to any computer system
- Computers must not only be able to move and copy data. They must be able to manipulate the data with arithmetic operations
- It is important to consider the various types of number notation schemes and their pros and cons