Chapter 7
Arithmetic Logic

Jin-Fu Li
Department of Electrical Engineering
National Central University
Jungli, Taiwan
Outline

- Addition and Subtraction of Signed Numbers
- Multiplication of Positive Numbers
Content Coverage

Main Memory System

Central Processing Unit (CPU)
- Cache memory
  - Operational Registers
    - Program Counter
  - Control Unit
- Arithmetic and Logic Unit

Instruction Sets

Input/Output System

Advanced Reliable Systems (ARES) Lab. Jin-Fu Li, EE, NCU
An Example of Binary Addition

```
0  0  0  1  0  1  1  0  (44)
1  0  1  1  0  0  1  0  (356)
```

```
1 1 1 1 1 1 1 1
```

```
0 0 0 1 0 1 1 0  (44)
1 0 1 1 0 0 1 0  (356)
```

```
0 1 1 0 0 1 0 0 0  (400)
```

```
0 1 1 0 0 1 0 0 0  (400)
```
1-bit Full Adder

Adder Truth Table

<table>
<thead>
<tr>
<th>C</th>
<th>A</th>
<th>B</th>
<th>A.B(G)</th>
<th>A+B(P)</th>
<th>A⊕B</th>
<th>SUM</th>
<th>CARRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Generate Signal G(A.B): occurs when a carry output (CARRY) is internally generated within the adder.

Propagate Signal P(A+B): when it is true, the carry in signal C is passed to the carry output (CARRY) when C is true.
**Logic for a 1-bit Full Adder**

\[ \text{SUM} = A \oplus B \oplus C \]
\[ \text{CARRY} = AB + AC + BC \]

Single-bit schematic of SUM

Single-bit schematic of CARRY
An N-bit Ripple Carry Adder

Disadvantage: long delay time
Binary Addition-Subtraction

\[ Y - X = Y + X' + 1 \]
Carry-LookAhead Addition

\[ C_{i+1} = A_iB_i + (A_i + B_i)C_i = G_i + P_iC_i \]

\[
C_1 = G_0 + P_0C_0 \\
C_2 = G_1 + P_1G_0 + P_1P_0C_0 \\
C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \\
C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0
\]
Multi-Level Carry LookAhead Addition

\[ c_1 = G_0 + P_0 c_0 \]
\[ c_2 = G_1 + G_0 P_1 + P_1 P_0 c_0 \]
\[ \vdots \]
\[ c_n = G_n + P_n G_{n-1} + P_n P_{n-1} G_{n-2} + \cdots + P_n P_{n-1} \cdots P_0 c_0. \]

\[ G_0^* = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3, \]
\[ P_0^* = P_0 P_1 P_2 P_3. \]

\[ c_4 = G_0^* + c_0 P_0^*, \]
\[ c_8 = G_1^* + G_0^* P_1^* + c_0 P_0^* P_1^*, \]
\[ c_{12} = G_2^* + G_1^* P_2^* + G_0^* P_1^* P_2^* + c_0 P_0^* P_1^* P_2^*, \]
\[ c_{16} = G_3^* + G_2^* P_3^* + G_1^* P_2^* P_3^* + G_0^* P_1^* P_2^* P_3^* + c_0 P_0^* P_1^* P_2^* P_3^*. \]
Multi-Level Carry LookAhead Addition

\[ \text{PG Unit} \]

level-1

\[ \text{CLA Unit} \]

level-2

\[ \text{Carry-Lookahead (CLA) Unit} \]

\[ a_{15-12} \]
\[ b_{15-12} \]
\[ a_{11-8} \]
\[ b_{11-8} \]
\[ a_{7-4} \]
\[ b_{7-4} \]
\[ a_{3-0} \]
\[ b_{3-0} \]
Multiplication of Positive Numbers

Bit-level multiplier

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>axb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Multiplication of two 4-bit words
Consider two unsigned binary integers $X$ and $Y$

$$X = \sum_{i=0}^{n-1} X_i 2^i \quad Y = \sum_{j=0}^{n-1} Y_j 2^j$$

$$P = X \times Y = \sum_{i=0}^{n-1} X_i 2^i \cdot \sum_{j=0}^{n-1} Y_j 2^j$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (X_i Y_j) 2^{i+j}$$

$$= \sum_{k=0}^{n+n-1} P_k 2^k$$
Array Multiplication
Register-Based Multiplication

\[ a_{n-1} a_0 \]
\[ q_{n-1} q_0 \]
\[ m_{n-1} \ldots m_0 \]

Control sequencer

Shift right

Add/Noadd control

Register A (initially 0)

n-bit adder

0

0

MUX

Shift right

Multiplier Q

Shift right

Add/Noadd control

Control sequencer

Multiplicand M

0
Example of Register-Based Multiplication

- **n-bit adder**
- **MUX**
- **Control sequencer**
- **Register A** (initially 0)
- **Multiplier Q**
- **Add/Noadd control**
- **Shift right**

**Multiplicand M**

```
1 1 0 1
```

```
0 0 0 0
```

```
1 0 0 0
```
Integer Division

\[
\begin{array}{c}
21 \\
\hline
13 \overline{274} \\
\underline{26} \\
14 \\
\hline
13 \\
\underline{13} \\
1
\end{array}
\]

\[
\begin{array}{c}
10101 \\
\hline
1101 \overline{100010010} \\
\underline{1101} \\
10000 \\
\underline{1101} \\
1110 \\
\underline{1101} \\
1
\end{array}
\]
Register-Based Division

- **Register A** (initially 0)
- **Dividend Q**
- **Divisor M**
- **Control sequencer**
- **Add/Subtract**
- **Shift left**

Diagram:
- **Adder** for adding or subtracting
- **Register A**
- **Dividend Q**
- **Divisor M**
- **Control sequencer**
- **Shift left**
A Restoring-Division Example

Initially: 0 0 0 0 0
Shift: 0 0 0 1 1
Subtract: 1 1 0 1
Set q₀: 1 1 1 0
Restore: 1 1
Subtract: 0 0 0 1 0
Shift: 0 0 1 0 0
Subtract: 1 1 0 1
Set q₀: 0 0 0 1
Shift: 0 0 1 0 0
Subtract: 1 1 1 0 1
Set q₀: 1 1 1 1

First cycle

Second cycle

Third cycle

Fourth cycle

Remainder: 0 0 1 0
Quotient: 1 0 0 0
A Nonrestoring-Division Example

Initially: 0 0 0 0 0  1 0 0 0

Shift: 0 0 0 1 1
Subtract: 1 1 1 0 1
Set q₀: 1 1 1 1 0  0 0 0

Shift: 1 1 1 0 0  0 0 0
Add: 0 0 0 1 1
Set q₀: 1 1 1 1 0  0 0 0

Shift: 1 1 1 1 0  0 0 0
Add: 0 0 0 1 1
Set q₀: 0 0 0 0 1  0 0 0

Shift: 0 0 0 1 0  0 0 0
Subtract: 1 1 1 0 1
Set q₀: 1 1 1 1 1  0 0 0
Add: 0 0 0 1 1
Set q₀: 0 0 0 0 1  0 0 0

Remainder: 0 0 0 1 0
Quotient: 100011

First cycle
Second cycle
Third cycle
Fourth cycle
Floating Point (a brief look)

- We need a way to represent
  - numbers with fractions, e.g., 3.1416
  - very small numbers, e.g., .000000001
  - very large numbers, e.g., 3.15576 \( \times 10^9 \)

- Representation:
  - sign, exponent, significand: \((-1)^{\text{sign}} \times \text{significand} \times 2^{\text{exponent}}\)
  - more bits for significand gives more accuracy
  - more bits for exponent increases range

- IEEE 754 floating point standard:
  - single precision: 8 bit exponent, 23 bit significand
  - double precision: 11 bit exponent, 52 bit significand
IEEE 754 Floating-Point Standard

- Leading “1” bit of significand is implicit

- Exponent is “biased” to make sorting easier
  - all 0s is smallest exponent all 1s is largest
  - bias of 127 for single precision and 1023 for double precision
  - summary: \((-1)^{\text{sign}} \times (1+\text{significand}) \times 2^{\text{exponent} - \text{bias}}\)

- Example:
  - decimal: 
    - 
  - binary: 
    - 
  - floating point: exponent = 126 = 01111110
  - IEEE single precision: 

Advanced Reliable Systems (ARES) Lab.

Jin-Fu Li, EE, NCU
Floating Point Addition

1. Compare the exponents of the two numbers. Shift the smaller number to the right until its exponent would match the larger exponent.

2. Add the significands.

3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent.

4. Round the significand to the appropriate number of bits.

Start

Small ALU

Exponent difference

Control

Shift right

Big ALU

Increment or decrement

Shift left or right

Rounding hardware

Sign Exponent Fraction

Overflow or underflow?

Exception

Yes

No

Still normalized?

No

Yes

Done
Floating Point Complexities

- Operations are somewhat more complicated
- In addition to overflow we can have “underflow”
- Accuracy can be a big problem
  - IEEE 754 keeps two extra bits, guard and round
    - For example, add 2.56x100 to 2.34x102 → 2.37x102 (with guard and round bits) or 2.36 (without guard and round bits)
  - four rounding modes
  - positive divided by zero yields “infinity”
  - zero divide by zero yields “not a number”
  - other complexities