Computer Organization

- 5 classic components of any computer

- Today we will look at datapaths (adder, multiplier, ...)

ECE232: Hardware Organization and Design

Part 2: Datapath Design – Binary Numbers and Adders

http://www.ecs.umass.edu/ece/ece232/
Unsigned Binary Integers

- Given an n-bit number

\[ x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_12^1 + x_02^0 \]

- Range: 0 to +2^{n-1} - 1
- Example
  - 0000 0000 0000 0000 0000 0000 0000 1011
    - 0 + \cdots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0
    - = 0 + \cdots + 8 + 0 + 2 + 1 = 11_{10}
- Using 32 bits
  - 0 to +4,294,967,295

2’s-Complement Signed Integers

- Given an n-bit number

\[ x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_12^1 + x_02^0 \]

- Bit n-1 is sign bit
  - 1/0 for negative/non-negative numbers
- Range: −2^{n-1} to +2^{n-1} - 1
- Example
  - 1111 1111 1111 1111 1111 1111 1111 1100
    - = -1 \times 2^{31} + 1 \times 2^{30} + \cdots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0
    - = -2,147,483,648 + 2,147,483,644 = -4_{10}
- Using 32 bits
  - -2,147,483,648 to +2,147,483,647
  - Most-negative: 1000 0000 \ldots 0000
  - Most-positive: 0111 1111 \ldots 1111
Signed Negation

- To get $-X$ complement $X$ and add 1
  - Complement means $1 \rightarrow 0$, $0 \rightarrow 1$
  
  $$x + \bar{x} = \text{1111...111}_2 = -1$$

- Example: negate $+2$
  
  $$+2 = \text{0000 0000 ... 0010}_2$$
  $$-2 = \text{1111 1111 ... 1101}_2 + 1$$
  $$= \text{1111 1111 ... 1110}_2$$

- Subtraction: $y - x = y + (\bar{x} + 1)$

Sign Extension

- Representing a number using more bits
  - Preserve the numeric value
- Replicate the sign bit to the left

Examples: 8-bit to 16-bit

- $+5$: $0000 0101 \Rightarrow 0000 0000 0000 0101$
- $-5$: $1111 1011 \Rightarrow 1111 1111 1111 1011$

Disadvantage of Signed-Magnitude Method

- Operation may depend on the signs of the operands
- Example - adding a positive number $X$ and a negative number $-Y$:
  $$X + (-Y)$$
- If $Y > X$, final result is $-(Y-X)$
- Calculation -
  - switch order of operands
  - perform subtraction rather than addition
  - attach the minus sign
- A sequence of decisions must be made, costing excess control logic and execution time
- This is avoided in the 2’s complement method
Overflow in 2’s Comp Add/Subtract (1)

- Example -
  
  \[
  \begin{array}{c c c c c c c}
  &  & 0 & 1 & 0 & 0 & 1 \\
  &  & 1 & 1 & 0 & 0 & 0 \\
  \hline
  & 1 & 0 & 0 & 0 & 0 & 0
  \end{array}
  \]

  \[
  9 \quad -7
  \]

  \[
  2
  \]

  Carry-out discarded - does not indicate overflow

- In general, if X and Y have opposite signs - no overflow can occur regardless of whether there is a carry-out or not

- Examples -
  
  \[
  \begin{array}{c c c c c c c}
  &  & 0 & 0 & 1 & 0 & 1 \\
  &  & 1 & 1 & 0 & 1 & 0 \\
  \hline
  & 1 & 1 & 0 & 1 & 1 & 1
  \end{array}
  \]

  5

  + 10

  \[
  \begin{array}{c c c c c c c}
  &  & 0 & 1 & 0 & 1 & 1 \\
  &  & 1 & 1 & 0 & 0 & 0 \\
  \hline
  & 1 & 0 & 0 & 0 & 1 & 1
  \end{array}
  \]

  -5

  No carry-out

Overflow in 2’s Comp Add/Subtract (2)

- If X and Y have the same sign and result has different sign - overflow occurs

- Examples -
  
  \[
  \begin{array}{c c c c c c c}
  &  & 1 & 0 & 1 & 1 & 1 \\
  &  & 1 & 0 & 1 & 1 & 1 \\
  \hline
  & 1 & 0 & 1 & 1 & 0 & 0
  \end{array}
  \]

  \[
  -9 \quad -9
  \]

  \[
  14 = -18 \mod 32
  \]

  - Carry-out and overflow

  \[
  \begin{array}{c c c c c c c}
  &  & 0 & 1 & 0 & 0 & 1 \\
  &  & 0 & 0 & 1 & 1 & 1 \\
  \hline
  & 0 & 1 & 0 & 0 & 0 & 0
  \end{array}
  \]

  \[
  9 \quad 7
  \]

  \[
  -16 = 16 \mod 32
  \]

  - No carry-out but overflow
Ripple Carry Adder

- **Addition**
  - most frequent operation
  - used also for multiplication and division
  - fast two-operand adder essential

- **Simple parallel adder**
  - for adding $X_{n-1}, X_{n-2}, \ldots, X_0$ and $Y_{n-1}, Y_{n-2}, \ldots, Y_0$
  - using $n$ full adders

- **Full adder**
  - combinational digital circuit with input bits $X_i, Y_i$ and incoming carry bit $C_i$, producing output sum bit $S_i$ and outgoing carry bit $C_{i+1}$
  - incoming carry for next FA with input bits $X_{i+1}, Y_{i+1}$
  - $S_i = X_i \oplus Y_i \oplus C_i$
  - $C_{i+1} = X_i \cdot Y_i + C_i \cdot (X_i + Y_i)$

---

Full-Adder (FA)

- Examine the Full Adder table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>Cin</th>
<th>Cout</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$Cout = x \cdot y + Cin \cdot (x + y)$

$S = x'y'c + x'y'c' + xy'c' + xyc$

$x \oplus y \oplus c$

In general, for bit $i$:

$C_{i+1} = x_i \cdot y_i + C_i \cdot (x_i + y_i)$

where $C_{i+1} = Cout, C_i = Cin$

Half adder has 2 inputs. In principle HA is same as FA, with Cin set to 0.
Parallel Adder: Ripple Carry

- In a parallel arithmetic unit
  - All $2^n$ input bits available at the same time
  - Carry propagates from the FA to the right to FA to the left
  - Carries ripple through all $n$ FAs before we can claim that the sum outputs are correct and may be used in further calculations
- Each FA has a finite delay

Example

- $x_3, x_2, x_1, x_0 = 1111$
- $y_3, y_2, y_1, y_0 = 0001$
- $\Delta FA$ - operation time - delay
- Assuming equal delays for sum and carry-out
- Longest carry propagation chain when adding two 4-bit numbers
- In synchronous arithmetic units - time allowed for adder's operation is worst-case delay - $n\Delta FA$

$$
\begin{align*}
T &= 0 + 1111 \\
T &= \Delta FA + 0001 \\
T &= 2\Delta FA + 1110 \\
T &= 3\Delta FA + 0111 \\
T &= 4\Delta FA + 1111 \\
\end{align*}
$$
Subtraction using Ripple Carry Adder

- Suppose you are performing X-Y operation
  - Complement Y bits
  - Force C0 to 1
  - Add

Example: X = 0101, Y = 0010; Compute X – Y
- First step: Complement Y
  - 1101
- Second step: add 0101 + 1101 + 1 = 0011

Carry Look Ahead Adder

- Problem with Ripple Carry Adder
  - Slow
  - How much is the delay for a 64 bit adder?
- Solution
  - Shorten carry propagation delay
  - Wouldn’t it be great to generate all carry signals in parallel?
  - How do you do that?
- Observation
  1. If \( X_i = Y_i = 1 \), a carry will be generated, Cin does not matter
  2. If \( X_i = Y_i = 0 \), no carry will be generated by the FA, Cin does not matter
  3. When does Cin matter in Cout generation?
    - \( X_i Y_i = 01 \)
    - \( X_i Y_i = 10 \)
- Carry Look Ahead Adder uses the above observation to generate carry signals in parallel
**Carry Look Ahead Adder**

- $G_i = X_i, Y_i$: generated carry; $P_i = X_i + Y_i$: propagated carry
- $c_1 = G_0 + c_0 P_0$,
- $c_2 = G_1 + G_0 P_1 + c_0 P_0 P_1$,
- $c_3 = G_2 + G_1 P_2 + G_0 P_1 P_2 + c_0 P_0 P_1 P_2$,
- $c_4 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + c_0 P_0 P_1 P_2 P_3$

**Plumbing analogy**

- $c_1 = g_0 + c_0 p_0$
- $c_2 = g_1 + g_0 p_1 + c_0 p_0 p_1$
- $c_4 = g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3 + c_0 p_0 p_1 p_2 p_3$
Delay of Carry Look Ahead Adders

- Let \( \tau \) be the delay of a gate.

- If inputs are available at time \( t=0 \), when are \( p \) and \( g \) signals available?

\[
\begin{align*}
X_3 &\quad Y_3 \\
\downarrow &\quad \downarrow \\
p_3 &\quad g_3
\end{align*}
\]
\[
\begin{align*}
X_2 &\quad Y_2 \\
\downarrow &\quad \downarrow \\
p_2 &\quad g_2
\end{align*}
\]
\[
\begin{align*}
X_1 &\quad Y_1 \\
\downarrow &\quad \downarrow \\
p_1 &\quad g_1
\end{align*}
\]
\[
\begin{align*}
X_0 &\quad Y_0 \\
\downarrow &\quad \downarrow \\
p_0 &\quad g_0
\end{align*}
\]

- Which signal will be generated last?
- How long will it take?

\[
\begin{align*}
c_1 &= G_0 + c_0 P_0, \\
c_2 &= G_1 + G_0 P_1 + c_0 P_0 P_1, \\
c_3 &= G_2 + G_1 P_2 + G_0 P_1 P_2 + c_0 P_0 P_1 P_2, \\
c_4 &= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + c_0 P_0 P_1 P_2 P_3
\end{align*}
\]
Gates are limited to two inputs

- \( C_4 = g_3 + p_3 g_2 + p_2 p_3 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \)

What if there were 6 inputs?
What if there were 7 inputs?
What if there were 8 inputs?
What if there were 9 inputs?

Total Delay

- \( \tau + 3\tau + \tau + 2\tau = 7\tau \)
- What is the delay of a 5 bit CLA?
- 6 bit CLA? 7 bit CLA?
- 8 bit CLA?

\[ G^* = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 \]
\[ P^* = P_0 P_1 P_2 P_3. \]
2-level Carry Look Ahead (16-bit)

- n=16 - 4 groups, 4-bit each

\[
\begin{align*}
    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^*
\end{align*}
\]

Plumbing Analogy
**Carry Select Adder**

- Principle: speculative

Carry propagate delay

\[ \text{CP}(2n) = 2 \times \text{CP}(n) \]

\[ \text{CP}(2n) = \text{CP}(n) + \text{CP}(\text{mux}) \]

**Summary**

- Throw hardware for performance
- Ripple Carry: least hardware, slowest
- CLA: faster, more hardware
- Carry Select: even faster, even more hardware
- Other techniques available, e.g., Carry skip adder
  - See [http://www.ecs.umass.edu/ece/koren/arith/simulator/](http://www.ecs.umass.edu/ece/koren/arith/simulator/)
- Combination of these techniques – hybrid adders

- Reading: Chapter 2 - Section 2.4
Different circuit implementation of a CLL

**MCC - Manchester Carry module**

64-bit Hybrid Adder