Two’s Complement Representation

Binary Arithmetic
Announcements

• HW4 due tomorrow

• HW5 will be released tonight

• Lab 3 report due next Monday/Tuesday
  – Form group before the submission

• A batch of raw quiz scores released on CMS
Example: Setup Time Analysis

• Assumptions:
  (1) Uniform gate delay = 1ns
  (2) FF propagation delay = 1ns
  (3) Setup time = 3ns, hold time = 2ns

• What’s the best achievable cycle time?
  \[ t_{\text{clk}} \geq t_{\text{ffpd (max)}} + t_{\text{comb (max)}} + t_{\text{setup}} = 1 + 3 + 3 = 7\text{ns} \]
Example: Hold Time Analysis with Clock Skew

Clock may arrive at FF2 up to 3ns later than FF1

<table>
<thead>
<tr>
<th></th>
<th>Prop Delay (ns)</th>
<th>Setup Time (ns)</th>
<th>Hold Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>Time (ns)</td>
</tr>
<tr>
<td>FF</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Comb</td>
<td>3</td>
<td>9</td>
<td>-</td>
</tr>
</tbody>
</table>

• Hold time at FF2 met?
  \[ t_{\text{ffpd}(\text{min})} + t_{\text{comb}(\text{min})} \geq t_{\text{hold}} + t_{\text{skew}(\text{max})} \]
  
  \[ 1 + 3 \geq 1 + 3 \]

The hold time constraint is met
Timing Analysis Discussions

• To achieve a higher clock frequency, would you prefer
  – a smaller **hold time** or a larger one?
  – a smaller **setup time** or a larger one?
  – a negative **clock skew** or a positive one?
    • a smaller skew or a larger one?
Course Content

- Binary numbers and logic gates
- Boolean algebra and combinational logic
- Sequential logic and state machines
- Binary arithmetic
- Memories
- Instruction set architecture
- Processor organization
- Caches and virtual memory
- Input/output
Unsigned Binary Integers

- An \( n \)-bit unsigned number represents \( 2^n \) integer values
  - From 0 to \( 2^n - 1 \)

<table>
<thead>
<tr>
<th>( 2^2 )</th>
<th>( 2^1 )</th>
<th>( 2^0 )</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
Unsigned Binary Fractions

• For the binary number $b_{n-1}b_{n-2} \ldots b_1b_0.b_{-1}b_{-2} \ldots b_{-m}$ the decimal number is

$$D = \sum_{i=-m}^{n-1} b_i \cdot 2^i$$

• Examples

101.001₂ = ?
Unsigned Binary Addition

- **Just like base-10**
  - Add from right to left, propagating carry

\[
\begin{align*}
10010 \quad &18 &+ &01001 \quad &9 &\rightarrow &10111 \quad &15 \\
&27 &+ &01011 \quad &11 &\rightarrow &11111 \quad &18 \\
&23 &+ &111 \quad &7 &\rightarrow &11110 \quad &30 \\
\end{align*}
\]
Signed Magnitude Representation

• Most significant bit is used as a sign bit
  – Sign bit of 0 for positive (0101 = 5)
  – Sign bit of 1 for negative (1101 = -5)

• Range is from -(2^{n-1}-1) to (2^{n-1}-1) for an n-bit number

• Two representations for zero (+0 and -0)

• Does ordinary binary addition still work?

  \[
  \begin{array}{c}
  0010 \quad (2) \\
  + \underline{1010} \quad (-2) \\
  1100 \quad (\text{not 0})
  \end{array}
  \]
Another Encoding of Binary Numbers

Wrap-around point
Two’s Complement Representation

- MSB has weight \(-2^n-1\)
- Range of an n-bit number: \(-2^n-1\) through \(2^{n-1}-1\)
  - Most negative number \((-2^n-1)\) has no positive counterpart

<table>
<thead>
<tr>
<th>-2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Two’s Complement Addition

- Procedure for addition is the same as unsigned addition regardless of the signs of the numbers

\[
\begin{align*}
011 \phantom{0} (3) \\
+ 101 \phantom{0} (-3) \\
\hline
000 \phantom{0} (0)
\end{align*}
\]
Sign Extension

• Replicate the MSB (sign bit)

<table>
<thead>
<tr>
<th>4-bit</th>
<th>8-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100  (4)</td>
<td>00000100 (still 4)</td>
</tr>
<tr>
<td>1100  (-4)</td>
<td>11111100 (still -4)</td>
</tr>
</tbody>
</table>

• Necessary for adding a two’s complement numbers of different lengths
Fixed Size Representation

- Microprocessors usually represent numbers as fixed size $n$-bit values.

- Result of adding two $n$-bit integers is stored as $n$ bits.

- Integers are typically 32 or 64 bits (words)
  - 4 or 8 bytes
  - byte = 8 bits
Fixed Size Addition

• Examples with $n = 4$

$$
\begin{array}{c}
2 \ 0010 \\
+ 3 \ 0011 \\
5 \ 0101 \\
\end{array}
\begin{array}{c}
2 \ 0010 \\
+ -3 \ 1101 \\
-1 \ 1111 \\
\end{array}
\begin{array}{c}
-2 \ 1110 \\
+ 6 \ 0110 \\
4 \ 0100 \\
\end{array}
$$

$$
\begin{array}{c}
-2 \ 1110 \\
+ -6 \ 1010 \\
-8 \ 1000 \\
\end{array}
\begin{array}{c}
7 \ 0111 \\
+ 6 \ 0110 \\
-3 \ 1101 \\
\end{array}
\begin{array}{c}
-7 \ 1001 \\
+ -4 \ 1100 \\
5 \ 0101 \\
\end{array}
$$

Something went wrong!
Overflow

• If operands are too big, sum cannot be represented as \( n \)-bit 2’s complement number

\[
\begin{align*}
01000 \quad (8) & \quad 11000 \quad (-8) \\
+ \quad 01001 \quad (9) & \quad + \quad 10111 \quad (-9) \\
10001 \quad (-15) & \quad 01111 \quad (+15)
\end{align*}
\]

• Overflow occurs if
  – Signs of both operands are the same, and
  – Sign of sum is different

• Another test (easy to do in hardware)
  – Carry into MSB does not equal carry out
Did Overflow Occur?

\[
\begin{array}{c}
11110010 \\
01110011 \\
+ 01010011 \\
\hline
11000101 \\
\end{array}
\]

\[
\begin{array}{c}
11110010 \\
11110011 \\
+ 01010011 \\
\hline
01000101 \\
\end{array}
\]

YES

NO
Two’s Complement Representation

• Positive numbers and zero are same as unsigned binary representation

• To get two’s complement negative notation of an integer
  – Flip every bit first
  – Then add one

\[
\begin{align*}
&\text{011} \quad (3) & &\text{01001} \quad (9) \\
&\downarrow & &\downarrow \\
&\text{100} \quad (1’s \ comp) & &\text{10110} \quad (1’s \ comp) \\
+ & \text{1} & + & \text{1} \\
&\text{101} \quad (-3) & &\text{10111} \quad (-9)
\end{align*}
\]
(-X) = (X’+1)

- To get two’s complement negative notation of an integer
  - Flip every bit first
  - Then add one

011 (3)
100 (1’s comp)
+ 1
101 (-3)
Two’s Complement (2’s C) Shortcut

- **To get -X**
  - Copy bits from right to left up to and including the first “1”
  - Flip remaining bits to the left

```
  011010000
  100101111
  + 1
  100110000
```

```
  011010000  (1’s comp)
  100101111  (flip)
  + 1
  100110000  (copy)
```
Converting Binary (2’s C) to Decimal

1. If MSB = 1, take two’s complement to get a positive number
2. Add powers of 2 for bit positions that have a “1”
3. If original number was negative, add a minus sign

\[
X = 11100110_{\text{two}} \\
-X = 00011010 \\
= 2^4 + 2^3 + 2^1 = 16 + 8 + 2 \\
= 26_{\text{ten}} \\
X = -26_{\text{ten}}
\]

Assuming 8-bit 2’s complement numbers

<table>
<thead>
<tr>
<th>( n )</th>
<th>( 2^n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
</tr>
</tbody>
</table>
Converting Decimal to Binary (2’s C)

First Method: *Division*

1. Change to nonnegative decimal number
2. Divide by two – remainder is least significant bit
3. Keep dividing by two until answer is zero, recording remainders from right (LSB) to left
4. **Append a zero as the MSB**;
   if original number $X$ was negative, return $X’+1$

---

$X = 104_{\text{ten}}$

<table>
<thead>
<tr>
<th>Division</th>
<th>Quotient</th>
<th>Remainder</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$104/2$</td>
<td>52</td>
<td>0</td>
<td>$0$</td>
</tr>
<tr>
<td>$52/2$</td>
<td>26</td>
<td>0</td>
<td>$0$</td>
</tr>
<tr>
<td>$26/2$</td>
<td>13</td>
<td>0</td>
<td>$0$</td>
</tr>
<tr>
<td>$13/2$</td>
<td>6</td>
<td>1</td>
<td>$1$</td>
</tr>
<tr>
<td>$6/2$</td>
<td>3</td>
<td>0</td>
<td>$0$</td>
</tr>
<tr>
<td>$3/2$</td>
<td>1</td>
<td>1</td>
<td>$1$</td>
</tr>
<tr>
<td>$1/2$</td>
<td>0</td>
<td>1</td>
<td>$1$</td>
</tr>
</tbody>
</table>

$X = 01101000_{\text{two}}$
Converting Decimal to Binary (2’s C)

Second Method: *Subtract Powers of Two*
1. Change to nonnegative decimal number
2. Subtract largest power of two less than or equal to number
3. Put a one in the corresponding bit position
4. Keep subtracting until result is zero
5. **Append a zero as MSB;**
   if original was $X$ negative, return $X’+1$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
</tr>
</tbody>
</table>

$X = 104_{\text{ten}}$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>104 -</td>
<td>64</td>
<td>40</td>
</tr>
<tr>
<td>40 -</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>8 - 8</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

$X = 01101000_{\text{two}}$
Full Adder (1 bit Adder with Carry)

- **Inputs:** A, B and $C_{\text{in}}$ (carry-in)
- **Outputs:** S (sum) and $C_{\text{out}}$ (carry-out)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$C_{\text{in}}$</th>
<th>S</th>
<th>$C_{\text{out}}$</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>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Full-Adder Circuit Symbol

A          B
Cin        S

Cout

FA

OR

Cout
Cin
S

A     B

Lecture 12: 26
Before Next Class

- H&H 5.5

Next Time

More Binary Arithmetic
ALU