Arithmetic Circuits

• Numbers as bits: two's complement
• Addition: ripple-carry adders
• Multiplication: unsigned and signed
• Intro to registers
Encoding numbers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an n-bit number encoded in this fashion is given by the following formula:

\[ v = \sum_{i=0}^{n-1} 2^i b_i \]

\[ \begin{array}{cccccccccccc}
2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0
\end{array} \]

\( = 2000_{10} \)

Oftentimes we will find it convenient to cluster groups of bits together for a more compact notation. Two popular groupings are clusters of 3 bits and 4 bits.

Seems natural to me!

Octal - base 8

<table>
<thead>
<tr>
<th>Octal</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
</tr>
</tbody>
</table>

Hexadecimal - base 16

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 - 0</td>
<td>0000 - 0</td>
</tr>
<tr>
<td>0001 - 1</td>
<td>1001 - 9</td>
</tr>
<tr>
<td>0010 - 2</td>
<td>1010 - a</td>
</tr>
<tr>
<td>0011 - 3</td>
<td>1011 - b</td>
</tr>
<tr>
<td>0100 - 4</td>
<td>1100 - c</td>
</tr>
<tr>
<td>0101 - 5</td>
<td>1101 - d</td>
</tr>
<tr>
<td>0110 - 6</td>
<td>1110 - e</td>
</tr>
<tr>
<td>0111 - 7</td>
<td>1111 - f</td>
</tr>
</tbody>
</table>
Representing negative integers

To keep our arithmetic circuits simple, we'd like to find a representation for negative numbers so that we can use a single operation (binary addition) when we wish to find the sum of two integers, independent of whether they are positive or negative.

We certainly want \( A + (-A) = 0 \). Consider the following 8-bit binary addition where we only keep 8 bits of the result:

\[
\begin{array}{c}
11111111 \\
+ 00000001 \\
\hline
00000000
\end{array}
\]

which implies that the 8-bit representation of -1 is 11111111.

More generally

\[
-A = 0 - A \\
= (-1 + 1) - A \\
= (-1 - A) + 1 \\
= \sim A + 1
\]

\( \sim \) means bit-wise complement.
Signed integers: 2’s complement

- $2^{N-1}$  $2^{N-2}$  ...  ...  ...  $2^3$  $2^2$  $2^1$  $2^0$

Range: $-2^{N-1}$ to $2^{N-1} - 1$

8-bit 2’s complement example:

11010110 = $-2^7 + 2^6 + 2^4 + 2^2 + 2^1 = -128 + 64 + 16 + 4 + 2 = -42$

If we use a two’s complement representation for signed integers, the same binary addition mod $2^n$ procedure will work for adding positive and negative numbers (don’t need separate subtraction rules). The same procedure will also handle unsigned numbers!

By moving the implicit location of “decimal” point, we can represent fractions too:

1101.0110 = $-2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} = -8 + 4 + 1 + 0.25 + 0.125 = -2.625$
Sign extension

Consider the 8-bit 2’s complement representation of:

\[
42 = 00101010 \\
-5 = \sim00000101 + 1 \\
= 11111010 + 1 \\
= 11111011
\]

What is their 16-bit 2’s complement representation?

\[
42 = 0000000000101010 \\
-5 = 1111111111111011
\]

Extend the MSB (aka the “sign bit”) into the higher-order bit positions.
Adder: a circuit that does addition

Here's an example of binary addition as one might do it by “hand”:

\[
\begin{array}{c}
1101 \\
\phantom{1}+ 0101 \\
\hline
10010
\end{array}
\]

Adding two N-bit numbers produces an (N+1)-bit result.

If we build a circuit that implements one column:

we can quickly build a circuit two add two 4-bit numbers...

“Ripple-carry adder”
“Full Adder” building block

The “half adder” circuit has only the A and B inputs.

\[ S = A \oplus B \oplus C \]
\[ CO = \overline{A}BC + \overline{A}BC + AB\overline{C} + ABC \]
\[ = (\overline{A} + A)BC + (B + B)AC + AB(C + C) \]
\[ = BC + AC + AB \]
Subtraction: $A - B = A + (-B)$

Using 2’s complement representation: $-B = \sim B + 1$

So let's build an arithmetic unit that does both addition and subtraction. Operation selected by control input:
Speed: $t_{PD}$ of Ripple-carry Adder

Worse-case path: carry propagation from LSB to MSB, e.g., when adding 11...111 to 00...001.

$\Theta(N)$ is read “order $N$” : means that the latency of our adder grows in proportion to the number of bits in the operands.

$$t_{PD,N\text{-BIT RIPPLE}} = N \times t_{PD,FA} \approx \Theta(N)$$
How about the $t_{PD}$ of this circuit?

Isn't the $t_{PD}$ of this circuit $= 2 \times t_{PD,N\text{-BIT RIPPLE}}$?

Nope! $t_{PD}$ of this circuit $= t_{PD,N\text{-BIT RIPPLE}} + t_{PD,FA}$ !!!

Timing analysis is tricky!
Simple Multiplication

Unsigned Multiplication

\[
\begin{array}{cccc}
A_3 & A_2 & A_1 & A_0 \\
\times & B_3 & B_2 & B_1 & B_0 \\
\hline
A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 \\
A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 \\
A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 \\
+ A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 \\
\end{array}
\]

AB\_i called a “partial product”

Multiplying N-bit number by M-bit number gives (N+M)-bit result

Easy part: forming partial products

(just an AND gate since B\_i is either 0 or 1)

Hard part: adding M N-bit partial products
Combinational Multiplier (unsigned)

\[
\begin{array}{cccccc}
X3 & X2 & X1 & X0 & \text{ multiplicand} \\
* & Y3 & Y2 & Y1 & Y0 & \text{ multiplier} \\
\hline
X3Y0 & X2Y0 & X1Y0 & X0Y0 \\
+ & X3Y1 & X2Y1 & X1Y1 & X0Y1 \\
+ & X3Y2 & X2Y2 & X1Y2 & X0Y2 \\
+ & X3Y3 & X2Y3 & X1Y3 & X0Y3 \\
\hline
Z7 & Z6 & Z5 & Z4 & Z3 & Z2 & Z1 & Z0
\end{array}
\]

Partial products, one for each bit in multiplier (each bit needs just one AND gate)

- Propagation delay ~2N
Combinational Multiplier (signed!)

\[
\begin{array}{cccccc}
X_3 & X_2 & X_1 & X_0 \\
\times & Y_3 & Y_2 & Y_1 & Y_0 \\
\hline
X_3Y_0 & X_3Y_0 & X_3Y_0 & X_3Y_0 & X_2Y_0 & X_1Y_0 & X_0Y_0 \\
+ & X_3Y_1 & X_3Y_1 & X_3Y_1 & X_3Y_1 & X_2Y_1 & X_1Y_1 & X_0Y_1 \\
+ & X_3Y_2 & X_3Y_2 & X_3Y_2 & X_2Y_2 & X_1Y_2 & X_0Y_2 \\
- & X_3Y_3 & X_3Y_3 & X_2Y_3 & X_1Y_3 & X_0Y_3 \\
\hline
Z_7 & Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0
\end{array}
\]

\[x_3 \quad x_2 \quad x_1 \quad x_0\]
\[y_3 \quad y_2 \quad y_1 \quad y_0\]

NB: There are tricks we can use to eliminate the extra circuitry we added...
Our last component: the D register

The edge-triggered D register: 
*on the rising edge of CLK*, the value of D is saved in the register and then shortly afterwards appears on Q.
D-Register Timing - I

$t_{PD}$: maximum propagation delay, $CLK \rightarrow Q$

$t_{SETUP}$: setup time
How long $D$ must be stable before the rising edge of $CLK$

$t_{HOLD}$: hold time
How long $D$ must be stable after the rising edge of $CLK$
D-Register Timing - II

\[ \text{t}_{\text{PD,reg1}} + \text{t}_{\text{PD,logic}} + \text{t}_{\text{SETUP,reg2}} < \text{t}_{\text{CLK}} \]