Outline

- Representing Integers
  - Unsigned, 2’s Complement
- Addition and subtraction
- Add/Sub ALU
  - full adder, ripple carry, subtraction,
- Carry lookahead
- Overflow
- Barrel shifter
- Defer: multiplication, division, floating-point

Unsigned Integers

- Recall:
  - n bits give rise to $2^n$ combinations
  - let us call a string of 32 bits as $b_{31} b_{30} \ldots b_2 b_1 b_0$
- $f(b_{31} \ldots b_0) = b_{31} \times 2^{31} + \ldots + b_1 \times 2 + b_0 \times 2^0$
- Treat as normal binary number
  - e.g., 0 . . .01101010= 1 x 2^7 + 1 x 2^6 + 0 x 2^5 + 1 x 2^4 + 0 x 2^3 + 1 x 2^2 + 0 x 2^1 + 1 x 2^0
  - = 128 + 64 + 16 + 4 + 1 = 213
- max f (111 . . . 11) = $2^{32}$ - 1 = 4, 294, 967, 295
- min f (000 . . . 00) = 0
- range [0, $2^{32}-1$] => # values ($2^{32}$ -1) - 0 + 1 = $2^{32}$

More Generally

- Bits have no inherent meaning
- Conventions define meaning
  - E.g., represent negative integers?
  - Seek circuit simplicity & speed
- n bits can represent finite possibilities: $2^n$
  - Integers countably infinite $\rightarrow$ overflow
  - Reals uncountably infinite $\rightarrow$ overflow, underflow, imprecise
Integer Representation

- Sign Magnitude: $000 = +0$
  $001 = +1$
  $010 = +2$
  $011 = +3$
  $100 = -0$
  $101 = -1$
  $110 = -2$
  $111 = -3$
- One's Complement: $000 = +0$, $001 = +1$, $010 = +2$, $011 = +3$, $100 = -0$, $101 = -1$, $110 = -2$, $111 = -3$
- Two's Complement: $000 = +0$, $001 = +1$, $010 = +2$, $011 = +3$, $100 = -0$, $101 = -1$, $110 = -2$, $111 = -3$
- Balance, number of zeros, ease of arithmetic

Two's Complement Integers

- $f(b_{31} b_{30} \ldots b_1 b_0) = -b_{31} \times 2^{31} + \ldots + b_1 \times 2 + b_0 \times 2^0$
- $\text{max } f(0111 \ldots 11) = 2^{31} - 1 = 2^{31} - 1 = 2147483647$
- $\text{min } f(100 \ldots 00) = -2^{31} = -2147483648$
  (asymmetric)
- range $[-2^{31}, 2^{31}-1] \Rightarrow \#\text{values} (2^{31}-1 - (-2^{31} + 1) = 2^{32}$
- E.g., -6
- $000 \ldots 0110 \Rightarrow 111 \ldots 1001 + 1 \Rightarrow 111 \ldots .1010$

Two's Complement Operations

- Negating a two's complement number: invert all bits and add 1
  - $1010 \Rightarrow 0101 + 1 = 0110$
  - $0110 \Rightarrow 1001 + 1 = 1010$
- Converting n bit numbers into numbers with more than n bits:
  - copy the most significant bit (the sign bit)
    $0010 \Rightarrow 0000 0010$
    $1010 \Rightarrow 1111 1010$
  - Called "sign extension"

Sign extension

- Consider representation of -2:
  3bit (decimal) 2-bit (decimal)
  $011 (+3)$  $01 (+1)$
  $010 (+2)$  $00 (0)$
  $001 (+1)$  $11 (-1)$
  $000 (0)$  $10 (-2)$
  $111 (-1)$  $10 (-2)$
  $110 (-2)$  $10 (-2)$
  $101 (-3)$  $10 (-2)$
  $100 (-4)$  $10 (-2)$
Addition and Subtraction

- Similar to decimal (carry/borrow twos instead of tens)
- Identical operation for signed and unsigned
  - E.g. Unsigned vs Signed
    
    |   |   |   |
    |---|---|---|
    | 0011 | 3 | 3 |
    | 1010 | 10 | -6 |
    | 1101 | 13 | -3 |

Interesting cases

- Show computation in 4-bit 2’s complement representation
  
  \[ 4 + (-4) \]

- Overflow: later

ALU bit-slice

- Bit-wise operation
  - and, or, add
  - Full adder?
  - Sub?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CarryIn</td>
<td>and</td>
<td>S-select</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>add</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1-bit Full Adder</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CarryOut</td>
<td>Result</td>
</tr>
</tbody>
</table>

Full adder

- Three inputs and two outputs
  - \( \text{Cout}, s = F(a,b,Cin) \)
    - \( \text{Cout} \): only if at least two inputs are set
    - \( s \): only if exactly one input or all three inputs are set
- Logic?

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{Cin}
\end{array} \xrightarrow{\text{Cout}} \begin{array}{c}
\text{a} \\
\text{b} \\
\text{Cin}
\end{array} \xrightarrow{s}
\]
### Subtract

- A - B = A + (–B)
- Form two complement by invert and add one

### Ripple-carry adder

![Diagram of ripple-carry adder](image)

### Problem: Slow

- Is a 32-bit ALU as fast as a 1-bit ALU?
  - Delay = 32x CP(Fast adder) + XOR
- Is there more than one way to do addition?
  - Two extremes: ripple carry and sum-of-products
  - Flatten expressions to two levels

Can you see the ripple? How could you get rid of it?

\[
\begin{align*}
c_0 &= b_0c_0 + a_0c_0 + a_0b_0 \\
c_1 &= b_1c_1 + a_1c_1 + a_1b_1 \\
c_2 &= b_2c_2 + a_2c_2 + a_2b_2 \\
c_3 &= b_3c_3 + a_3c_3 + a_3b_3 \\
c_4 &= b_4c_4 + a_4c_4 + a_4b_4 \\
c_5 &= b_5c_5 + a_5c_5 + a_5b_5 \\
c_6 &= b_6c_6 + a_6c_6 + a_6b_6 \\
c_7 &= b_7c_7 + a_7c_7 + a_7b_7 \\
c_8 &= b_8c_8 + a_8c_8 + a_8b_8 \\
c_9 &= b_9c_9 + a_9c_9 + a_9b_9 \\
c_{10} &= b_{10}c_{10} + a_{10}c_{10} + a_{10}b_{10} \\
c_{11} &= b_{11}c_{11} + a_{11}c_{11} + a_{11}b_{11} \\
c_{12} &= b_{12}c_{12} + a_{12}c_{12} + a_{12}b_{12} \\
c_{13} &= b_{13}c_{13} + a_{13}c_{13} + a_{13}b_{13} \\
c_{14} &= b_{14}c_{14} + a_{14}c_{14} + a_{14}b_{14} \\
c_{15} &= b_{15}c_{15} + a_{15}c_{15} + a_{15}b_{15} \\
c_{16} &= b_{16}c_{16} + a_{16}c_{16} + a_{16}b_{16} \\
c_{17} &= b_{17}c_{17} + a_{17}c_{17} + a_{17}b_{17} \\
c_{18} &= b_{18}c_{18} + a_{18}c_{18} + a_{18}b_{18} \\
c_{19} &= b_{19}c_{19} + a_{19}c_{19} + a_{19}b_{19} \\
c_{20} &= b_{20}c_{20} + a_{20}c_{20} + a_{20}b_{20} \\
c_{21} &= b_{21}c_{21} + a_{21}c_{21} + a_{21}b_{21} \\
c_{22} &= b_{22}c_{22} + a_{22}c_{22} + a_{22}b_{22} \\
c_{23} &= b_{23}c_{23} + a_{23}c_{23} + a_{23}b_{23} \\
c_{24} &= b_{24}c_{24} + a_{24}c_{24} + a_{24}b_{24} \\
c_{25} &= b_{25}c_{25} + a_{25}c_{25} + a_{25}b_{25} \\
c_{26} &= b_{26}c_{26} + a_{26}c_{26} + a_{26}b_{26} \\
c_{27} &= b_{27}c_{27} + a_{27}c_{27} + a_{27}b_{27} \\
c_{28} &= b_{28}c_{28} + a_{28}c_{28} + a_{28}b_{28} \\
c_{29} &= b_{29}c_{29} + a_{29}c_{29} + a_{29}b_{29} \\
c_{30} &= b_{30}c_{30} + a_{30}c_{30} + a_{30}b_{30} \\
c_{31} &= b_{31}c_{31} + a_{31}c_{31} + a_{31}b_{31} \\
\end{align*}
\]

Not feasible! Why? Exponential fanin

### Carry look-ahead

- An approach in-between our two extremes
- Motivation:
  - If we didn’t know the value of carry-in, what could we do?
  - When would we always generate a carry?
    - \( g_i = a_i b_i \)
  - When would we propagate the carry?
    - \( p_i = a_i + b_i \)
- Did we get rid of the ripple?
CLA: Plumbing Analogy

Carry-Lookahead Adder

- Waitaminute!
  - Nothing has changed
  - Fanin problems if you flatten!
    - Linear fanin, not exponential
  - Ripple problem if you don’t!
- Enables divide-and-conquer
- Figure out Generate and Propagate for 4-bits together
- Compute hierarchically

Carry Lookahead adder

- Height of tree = O(lg(n))
- 32 bit addition: \( k \times \text{lg}(32) = k \times 5 \)
Carry-Lookahead Adder

- Hierarchy
  - $G_{i,k} = G_{j+1,k} + P_{j+1,k} \cdot G_{i,j}$ (assume $i < j + 1 < k$)
  - $P_{i,k} = P_{i,j} \cdot P_{j+1,k}$
  - $G_{0,7} = G_{4,7} + P_{4,7} \cdot G_{0,3}$
  - $P_{0,7} = P_{0,3} \cdot P_{4,7}$
  - $G_{8,15} = G_{12,15} + P_{12,15} \cdot G_{8,11}$
  - $P_{8,15} = P_{8,11} \cdot P_{12,15}$
  - $G_{0,15} = G_{8,15} + P_{8,15} \cdot G_{0,7}$
  - $P_{0,15} = P_{0,7} \cdot P_{8,15}$

Parallel algorithm in hardware.

Computing G's and Ps

- $G_{12,15} P_{12,15}$
- $G_{8,11} P_{8,11}$
- $G_{0,7} P_{0,7}$
- $G_{0,3} P_{0,3}$
- $G_{15,8} P_{15,8}$
- $G_{7,0} P_{7,0}$
- $G_{15,0} P_{15,0}$
Computing C’s

- Different tree.
- Note propagation order
- Worth spending some time to think about the intricacies

Overflow

- Examples: $7 + 3 = 10$ but ...
- $-4 - 5 = -9$ but ...

Overflow

- Overflow: the result is too large (or too small) to represent properly
- Example: $-8 = 4$-bit binary number $<= 7$
- When adding operands with different signs, overflow cannot occur!
- Overflow occurs when adding:
  - 2 positive numbers and the sum is negative
  - 2 negative numbers and the sum is positive
- On your own: Prove you can detect overflow by:
  - Carry into MSB $+$ Carry out of MSB
Overflow detection

- Carry into MSB \( \oplus \) Carry out of MSB
  - For N-bit ALU: Overflow = CarryIn\[N - 1\] \( \oplus \) CarryOut\[N - 1\]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X XOR Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Negative, Zero

- Required for conditional branches
- Zero
  - How?
  - NOR all 32 bits
  - Avoid 33\textsuperscript{rd} bit (carry out)
- Negative may be required on overflow
  - If \( (a < b) \) jump : jump taken if \( a - b \) is negative
- Tempting to consider MSB
  - E.g. if \((\,-5 < 4)\) branch
  - Branch should be taken, but \((-5 - 4)\) computation results in overflow... so MSB is 0
  - E.g. if \((7 < -3)\) branch
  - Branch should not be taken but \((7 - (-3))\) results in overflow... so MSB is 1.

Shift

- E.g., Shift left logical for \( d\{7:0\} \) and \( shamt\{2:0\} \)
  - Using 2-1 muxes called Mux(select, in0, in1)
  - \( stage0\{7:0\} = Mux(shamt\{0\}, d\{7:0\}, 0 || d\{7:1\}) \)
  - \( stage1\{7:0\} = Mux(shamt\{1\}, stage0\{7:0\}, 00 || stage0\{6:2\}) \)
  - \( dout\{7:0\} = Mux(shamt\{2\}, stage1\{7:0\}, 0000 || stage1\{3:0\}) \)
- Other operations
  - Right shift
  - Arithmetic shifts
  - Rotate

Barrel Shifter

Stage 0

\[
\begin{array}{c}
\text{Stage 0}
\end{array}
\]

Stage 1

\[
\begin{array}{c}
\text{Stage 1}
\end{array}
\]

Stage 2

\[
\begin{array}{c}
\text{Stage 2}
\end{array}
\]