EECS151/251A Discussion 12

Christopher Yarp
Apr. 26, 2019
Plan for Today

• Multipliers (including reminders from last lecture)
• Constant Multiplication
• Questions
Multipliers (From Last Week)

• Remember, the mechanics of multiplication in binary are generally the same as decimal multiplication (signed multiply requires a slight tweak).

• 2 Steps to Multiplication:
  • Generation of partial products
  • Adding partial products

• Making faster multipliers mostly involves changing how we deal with generating and adding the partial products
Unsigned Multiplication Example (From Last Week)

4ʼb0011 (3) * 4ʼb0110 (6)

- Partial Products can be generated in parallel

4ʼb0011 (3) * 4ʼb0110 (6)

---

0000
0011
0011
+ 0000
---

00010010 (18)
Number Representations

• Unsigned Binary
  • Each bit place represents a different power of 2
  • Ex: 11 in unsigned binary = $2^3 + 2^1 + 2^0 = 8 + 2 + 1 = 11$

<table>
<thead>
<tr>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• Signed Binary – 2’s Complement
  • Each bit place still represents a different power of 2, except the most significant bit has negative weight
  • Converting to/from 2’s complement can be accomplished by performing a bitwise negation and adding 1. Ex. -11 in 2’s complement = -$2^5 + 2^4 + 2^2 + 2^0 = -32 + 16 + 4 + 1 = -11$

<table>
<thead>
<tr>
<th>-2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Signed Multiplication Example

\[ 4'b0011 \ (3) \times 4'b1100 \ (-4) \]

\[ + \ 00000000 \]
\[ + \ 0000000 \]
\[ + \ 000011 \]
\[ - \ 00011 \]
\[ + \ 00001100 \]
\[ + \ 11100111 + 1 \]
\[ + \ 00001100 \]
\[ + \ 11101000 \]
\[ 11110100 \ (-12) \]

• In 2’s Complement, the MSB is given negative weight
• Need to sign extend numbers when writing partial products
• Need to subtract partial product for MSB
• Carry bit of additions is discarded
Signed Multiplication Example

4ʼb1100 (-4) * 4ʼb0011 (3)

- 11111100
- 1111100
- 000000
- 0000

11111111+1

11110100

11110100

11110100 (-12)

• In 2ʼs Complement, the MSB is given negative weight
• Need to sign extend numbers when writing partial products
• Need to subtract partial product for MSB
• Carry bit of additions is discarded
Signed Multiplication Example

4’b1100 (-4) * 4’b1101 (-3)

- In 2’s Complement, the MSB is given negative weight
- Need to sign extend numbers when writing partial products
- Need to subtract partial product for MSB
- Carry bit of additions is discarded

\[ \begin{align*}
4’b1100 & \quad (\text{-}4) \\
\times & \quad 4’b1101 \quad (\text{-}3) \\
\hline
+ & \quad 11111100 \\
+ & \quad 00000000 \\
+ & \quad 11111000 \\
- & \quad 111000 \\
\hline
\end{align*} \]

\[ \begin{align*}
+ & \quad 11101100 \\
+ & \quad 00011111 + 1 \\
\hline
\end{align*} \]

\[ \begin{align*}
+ & \quad 11101100 \\
+ & \quad 00100000 \\
\hline
\end{align*} \]

\[(1)00001100 \quad (12)\]
Signed Multiplication Example

\[ 4\text{'b0100} \times 4\text{'b0011} \]

- In 2’s Complement, the MSB is given negative weight
- Need to sign extend numbers when writing partial products
- Need to subtract partial product for MSB
- Carry bit of additions is discarded
Accelerating Multiplication
Accelerating the Addition of Partial Products

• Let’s look at an (unsigned) array multiplier

• The products can be computed in parallel but the carry chain when adding partial products is limiting the speed

• How do we improve performance without having a large increase in hardware?
  • We could implement each adder as a parallel prefix or a carry-lookahead adder
  • However, remember that these adders require more logic than a simple carry ripple adder
One Solution: Carry Save Addition

• When we generate a carry in a given column of an addition, we add it to the 2 values in the next column.
  • This addition may in turn generate its own carry

• If adding carries is just like another addition, can we delay adding the carry bits until later?
  • Yes, so long as we remember what the carry bits need to be added

• This is the basis of the carry save adder:
  • Takes in a, b, and carry_in (multi-bit)
  • Produces a sum and carry_out (multi-bit)
Using Carry Save Addition in Multipliers

• Carry now propagates down each column.
  • Carry ripple across rows is eliminated in the array

• Still need to handle carries at the end with a fast adder

Figure from Lecture Slides
Using Carry Save Addition

• Remember, sums are associative and communitive.

• We can add the partial products in a tree structure using carry save adders!
  • Now have a number of layers that scales logarithmically!

• This is the basis of the Wallace Tree Multiplier
Radix and Multiplication

• Binary arithmetic has some advantages
  • Partial product generation is just a series of AND gates (including sign extension)

• However, there are also disadvantages
  • There is a partial product for each bit of the multiplier
  • That leads to a lot of partial products (a lot of additions)

• Ex. 3*4
  • single partial product in base 10
  • 4 partial products in base 2.

• Why don’t we consider a larger radix?
Radix 4 Multiplication

• Let’s consider 2 bits at a time
  • Halve the number of partial products we generate

• Radix 4 multiplication A*B
  • Partial Product Shift By 2 bits each time

<table>
<thead>
<tr>
<th>B Digit</th>
<th>Partial Product</th>
<th>Partial Product (Rewritten)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0*A</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1*A</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2*A</td>
<td>4<em>A - 2</em>A</td>
</tr>
<tr>
<td>3</td>
<td>3*A</td>
<td>4*A - A</td>
</tr>
</tbody>
</table>

• Recall: Multiplications by powers of 2 are left shifts
• Can we use this property?
Booth Recoding

- Uses radix 4 arithmetic
- Modification: Partial Products for B==2 and B==3 can be separated into 4*A – {2, 1}A
- 4*A can be implemented as a shift to the left by 2
- 2*A can be implemented as a shift to the left by 1
- Recall that we are doing radix 4 multiplication, we shift left by 2 positions for the next partial product
- Therefore, any 4*A term can be handled in the next partial product!
  - To do this, the multiplier needs to look at 3 (rather than just 2) bits. The extra bit is the MSB of the previous

<table>
<thead>
<tr>
<th>B Digit</th>
<th>Partial Product</th>
<th>Partial Product (Rewritten)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0*A</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1*A</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2*A</td>
<td>4<em>A - 2</em>A</td>
</tr>
<tr>
<td>3</td>
<td>3*A</td>
<td>4*A - A</td>
</tr>
</tbody>
</table>
### Booth Recoding

<table>
<thead>
<tr>
<th>B_{i+1}</th>
<th>B_i</th>
<th>B_{i-1}</th>
<th>Action</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Add 0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Add A</td>
<td>Includes +4*A from previous radix 4 digit = +A in this position due to left shift by 2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Add A</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Add 2*A</td>
<td>Includes +4*A from previous round (+A in this position). *2 is implemented as a left shift by 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Sub 2*A</td>
<td>4*A will be added in when handling next radix 4 digit. *2 is implemented as a left shift by 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Sub A</td>
<td>4*A will be added in when handling next radix 4 digit. Includes +4*A from previous radix 4 digit (+A in this position)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Sub A</td>
<td>4*A will be added in when handling next radix 4 digit.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Add 0</td>
<td>4*A will be added in when handling next radix 4 digit. Includes +4*A from previous radix 4 digit (+A in this position)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B Digit</th>
<th>Partial Product</th>
<th>Partial Product (Rewritten)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0*A</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1*A</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2*A</td>
<td>4<em>A - 2</em>A</td>
</tr>
<tr>
<td>3</td>
<td>3*A</td>
<td>4*A - A</td>
</tr>
</tbody>
</table>
Booth Recoding Example (Unsigned)

- Example: 6*4
- B_{-1} = 0

\[
\begin{array}{c}
\text{4'b0110 } (6) \\
\times \text{4'b0111 } (7) \\
\hline
\text{0110 (Sub A)} \\
+ \text{01100 (Add 2A)} \\
+ \text{0000 (Add 0)} \\
\hline
\text{11111010 (Sub A)} \\
+ \text{01100 (Add 2A)} \\
+ \text{0000 (Add 0)} \\
\hline
(1)\text{00101010 (42)}
\end{array}
\]

<table>
<thead>
<tr>
<th>B_{i+1}</th>
<th>B_i</th>
<th>B_{i-1}</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Add 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Add A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Add A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Add 2*A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Sub 2*A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Sub A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Sub A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Add 0</td>
</tr>
</tbody>
</table>
Additional Methods

• Pipelining!
  • Used in many high performance systems
  • Upside: Increased throughput
  • Downside: Increased latency
  • Good if you have many independent multiplications to perform and latency is acceptable

Figure from Lecture Slides
Signed Multiplication Tricks

• 2 things we need to do for signed multiplication:
  • Sign extend partial products
  • Subtract last partial products

• How can we simplify matters?
  • Sign extension requires additional logic
  • Add constants that allows us to eliminate the sign extension logic
  • Merge with the constant that is added when negating the last partial product
Trick with Sign Extension

• Ex. Sign Extend 1100 to 8 bits: 11111100
  • Add 1000
  • Causes a carry to ripple
    11111100
    + 00001000
    ------------
    (1)00000100
  • Results in the original input with the MSB Inverted

• Ex. Sign Extend 0100 to 8 bits: 00000100
  • Add 1000
  • No carry ripple
    00000100
    + 00001000
    ------------
    00001100
  • Results in the original input with the MSB inverted
  • Allows us to eliminate the 4 AND gates required for sign extension
  • Need an inverter and to subtract the constant later
Application of Sign Extension Trick

1) Invert Last Partial Product (From Lecture Slides)

\[
\begin{array}{cccc}
X_3 & X_2 & X_1 & X_0 \\
* & Y_3 & Y_2 & Y_1 & Y_0 \\
\end{array}
\]

\[
+ \begin{array}{cccc}
X_3Y_0 & X_3Y_0 & X_3Y_0 & X_3Y_0 \\
X_2Y_0 & X_1Y_0 & X_0Y_0 \\
\end{array}
+ \begin{array}{cccc}
X_3Y_1 & X_3Y_1 & X_3Y_1 & X_3Y_1 \\
X_2Y_1 & X_1Y_1 & X_0Y_1 \\
\end{array}
+ \begin{array}{cccc}
X_3Y_2 & X_3Y_2 & X_3Y_2 & X_3Y_2 \\
X_2Y_2 & X_1Y_2 & X_0Y_2 \\
\end{array}
- \begin{array}{cccc}
X_3Y_3 & X_3Y_3 & X_3Y_3 & X_3Y_3 \\
X_2Y_3 & X_1Y_3 & X_0Y_3 \\
\end{array}
\]

\[
\begin{array}{cccc}
Z_7 & Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0 \\
\end{array}
\]

2) Add Constants (From Lecture Slides)

\[
\begin{array}{cccc}
X_3 & X_2 & X_1 & X_0 \\
* & Y_3 & Y_2 & Y_1 & Y_0 \\
\end{array}
\]

\[
+ \begin{array}{cccc}
X_3Y_0 & X_3Y_0 & X_3Y_0 & X_3Y_0 \\
X_2Y_0 & X_1Y_0 & X_0Y_0 \\
\end{array}
+ \begin{array}{cccc}
1 & 0 & 0 & 0 \\
\end{array}
+ \begin{array}{cccc}
X_3Y_1 & X_3Y_1 & X_3Y_1 & X_3Y_1 \\
X_2Y_1 & X_1Y_1 & X_0Y_1 \\
\end{array}
+ \begin{array}{cccc}
1 & 0 & 0 & 0 \\
\end{array}
+ \begin{array}{cccc}
X_3Y_2 & X_3Y_2 & X_3Y_2 & X_3Y_2 \\
X_2Y_2 & X_1Y_2 & X_0Y_2 \\
\end{array}
+ \begin{array}{cccc}
1 & 0 & 0 & 0 \\
\end{array}
+ \begin{array}{cccc}
X_3Y_3 & X_3Y_3 & X_3Y_3 & X_3Y_3 \\
X_2Y_3 & X_1Y_3 & X_0Y_3 \\
\end{array}
+ \begin{array}{cccc}
1 & (+1 \text{ from Neg}) \\
\end{array}
+ \begin{array}{cccc}
1 & 0 & 0 & 0 \\
\end{array}
- \begin{array}{cccc}
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
Z_7 & Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0 \\
\end{array}
\]
Application of Sign Extension Trick

3) Add Constants (From Lecture Slides)

\[ X_3 \quad X_2 \quad X_1 \quad X_0 \]
\[ \ast \quad Y_3 \quad Y_2 \quad Y_1 \quad Y_0 \]

\[ + \quad X_3Y_0 \quad X_2Y_0 \quad X_1Y_0 \quad X_0Y_0 \]
\[ + \quad X_3Y_1 \quad X_2Y_1 \quad X_1Y_1 \quad X_0Y_1 \]
\[ + \quad X_3Y_2 \quad X_2Y_2 \quad X_1Y_2 \quad X_0Y_2 \]
\[ + \quad X_3Y_3 \quad X_2Y_3 \quad X_1Y_3 \quad X_0Y_3 \]
\[ + \quad 1 \quad (+1 \text{ from Neg}) \]
\[ - \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \]

\[ Z_7 \quad Z_6 \quad Z_5 \quad Z_4 \quad Z_3 \quad Z_2 \quad Z_1 \quad Z_0 \]

4) Negate Last Term (From Lecture Slides)

\[ X_3 \quad X_2 \quad X_1 \quad X_0 \]
\[ \ast \quad Y_3 \quad Y_2 \quad Y_1 \quad Y_0 \]

\[ + \quad X_3Y_0 \quad X_2Y_0 \quad X_1Y_0 \quad X_0Y_0 \]
\[ + \quad X_3Y_1 \quad X_2Y_1 \quad X_1Y_1 \quad X_0Y_1 \]
\[ + \quad X_3Y_2 \quad X_2Y_2 \quad X_1Y_2 \quad X_0Y_2 \]
\[ + \quad X_3Y_3 \quad X_2Y_3 \quad X_1Y_3 \quad X_0Y_3 \]
\[ + \quad 1 \quad (+1 \text{ from Neg}) \]
\[ + \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \]

\[ Z_7 \quad Z_6 \quad Z_5 \quad Z_4 \quad Z_3 \quad Z_2 \quad Z_1 \quad Z_0 \]
Application of Sign Extension Trick

5) Add Constants (From Lecture Slides)

\[
\begin{array}{cccc}
X_3 & X_2 & X_1 & X_0 \\
* & Y_3 & Y_2 & Y_1 & Y_0 \\
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
+ & X_3Y_0 & X_2Y_0 & X_1Y_0 & X_0Y_0 \\
+ & X_3Y_1 & X_2Y_1 & X_1Y_1 & X_0Y_1 \\
+ & X_3Y_2 & X_2Y_2 & X_1Y_2 & X_0Y_2 \\
+ & X_3Y_3 & X_2Y_3 & X_1Y_3 & X_0Y_3 \\
\hline
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
Z_7 & Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0 \\
\hline
\end{array}
\]

- Can be implemented with limited modifications to the unsigned multiplier!
- Requires passing some constants to full adders and inverting some terms

Figure from Lecture Slides
Constant Coefficient Multipliers
Multiplying by a Constant

• Observation: Every number can be factored into a sum of powers of 2
  • This is exactly what we do when we write a number in binary!
    • Ex. 11 = 2^3 + 2^1 + 2^0 = 8 + 2 + 1
• Can we leverage this to help us multiply by constants?
  • Yes!

• Use the distributive property
  • Ex. A*11 = A*(2^3 + 2^1 + 2^0) = A*2^3 + A*2^1 + A*2^0
• Use the fact that power of 2 multiplies are shifts
  • Ex. A*11 = A<<3 + A<<1 + A<<0
  • Turned a multiply into shifts by fixed amounts and additions
Extending to Use Subtraction

• This concept can be extended to use subtraction
  • Ex. $15 = 2^3 + 2^2 + 2^1 + 2^0 = 2^4 - 2^0 = 16 - 1$
  • $A*15 = A*2^4 - A*2^0 = A<<4 - A<<0$

• This is denoted by drawing a line over digits with negative weight
  • Ex. $15 = 001111 = 01000\bar{1}$
Canonical Signed Digit

- CSD Represents Numbers using 1, 0, $\overline{1}$ digits
- Minimizes the number of nonzero digits
  - Minimizes the number of additions needed when multiplying by a constant

Procedure (2 Passes):
1. Replace any occurrence of 2 or more 1’s (01…10) with 10…$\overline{1}$0
2. Replace any occurrence of 2 or more 1’s (01…10) with 10…$\overline{1}$0
   and Replace 0110 with 0010
   and Replace 0110 with 0010

Ex (From Lecture).
0010111 = 23
001100$\overline{1}$ (Pass 1)
010$\overline{1}$00$\overline{1}$ (Pass 2) = 32 – 8 – 1