Storage Elements, Busses, Integer Arithmetic

Computer Science 104
Lecture 10

Admin

• Homework #3 Due March 2 (two person group)
• Project specification(s) on the web, Due April 20, three person groups (email me today)

Outline
• Review
• Logic for Storage
• Register File
• Busses
• Integer Multiply and Divide

Reading
• Appendix B.7-B.8, Ch 3.3-3.5
• Next week Ch 3.6-3.9, then start Ch5 & B.10
**Review: A 1-bit Full Adder**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</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>

01101100 + 00101100 = 10011001

**Review: The new ALU Slice**

<table>
<thead>
<tr>
<th>A</th>
<th>F</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>a + b</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>a - b</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>NOT b</td>
</tr>
<tr>
<td>-</td>
<td>2</td>
<td>a OR b</td>
</tr>
<tr>
<td>-</td>
<td>3</td>
<td>a AND b</td>
</tr>
</tbody>
</table>
Review: Abstraction--The ALU

- General structure

Input
- Two operands
- Control

Output
- Result
- Overflow
- Zero

Memory Elements

- All the circuits we looked at so far are **combinational circuits**: the output is a Boolean function of the inputs.

- We need circuits that can remember values. (registers)

- The output of the circuit is a function of the input AND a function of a stored values (state).

- Circuits with memory are called **sequential circuits**.
Set-Reset Latch

R  S  Q
0  0  Q
0  1  1
1  0  0
1  1  -

Set-Reset Latch (Continued)

R  S  Q
0  0  Q
0  1  1
1  0  0
1  1  -
Set-Reset Latch (Continued)

Data Latch (D Latch)

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>Q</td>
</tr>
</tbody>
</table>

Does not affect Output
On \( C \uparrow \) D is transferred to the first D latch and the second is stable.

On \( C \downarrow \) the output of the first stage is transferred to the second (output), and the first stage is stable.

Output changes only on the edge of a clock
Register File

- Register File = the set of locations for register values
- How do I build a Register File using D Flip-Flops?
- What other components do I need?

Tri-State Driver

- The Tri-State driver is like a (one directional) switch:
  - When the Enable is on (E=1) it transfers the input to the output.
  - When the Enable is off (E=0) it disconnects the output.

\[
\begin{array}{ccc}
D & E & Q \\
0 & 1 & 0 \\
1 & 1 & 1 \\
- & 0 & Z \\
\end{array}
\]

\[Z := \text{High Impedance}\]
Bus Connections

- The Bus: Many to many connections.
- Mutual exclusion: At most one Enable is on!
- Control must ensure this!

Register Cells on a bus

One can “source” and “sink” from any cell on the bus by activating the right controls, IE--input enable, and OE--output enable.
3-Port Register Cell

- Stores one bit of a register
- Can Read onto Bus-A & Bus-B and Write from Bus-C Simultaneously

3-Port Register File
Address Decode Circuit

Register address: 01

Register File (Four 4-bit Registers)
Digital Logic Summary

- Given Boolean function, generate a circuit that “realize” the function.
- Constructed circuits that can add and subtract.
- The ALU: a circuit that can add, subtract, detect overflow, compare, and do bit-wise operations (AND, OR, NOT)
- Shifter
- Memory Elements: SR-Latch, D Latch, D Flip-Flop
- Tri-state drivers & Bus Communication
- Register Files
- Control Signals modify what circuit does with inputs
  - ALU, Shift, Register Read/Write

Arithmetic

- Integer Addition---Done
- Integer Multiplication
- Integer Division
- Floating Point Addition
- Floating Point Multiplication
**Integer Multiplication**

- **Product** = Multiplicand \( \times \) Multiplier

- **Example**: \( 0011_{\text{ten}} \times 0101_{\text{ten}} \)

<table>
<thead>
<tr>
<th>Multiplicand</th>
<th>0 0 1 1_{\text{ten}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier</td>
<td>0 1 0 1_{\text{ten}}</td>
</tr>
<tr>
<td></td>
<td>0 0 1 1</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>0 0 1 1</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Product</td>
<td>0 0 0 1 1 1_{\text{ten}}</td>
</tr>
</tbody>
</table>

**Multiplication Algorithm #1**

- **From Right-Left**:
  - If multiplier digit = 1: add (shifted) copy of multiplicand to result.
  - If multiplier digit = 0: add 0 to result.

- 32 steps when multiplier is 32-bit number.

- **Example**: \( 3_{10} \times 5_{10} \) or \( 0011_{2} \times 0101_{2} \)
  
  Product = \( 00001111_{2} \)
Multiplication Algorithm #1

1. Test Multiplier
   a. Add multiplicand to product and place the result in Product register

2. Shift the Multiplicand register left 1 bit

3. Shift the Multiplier register right 1 bit

Do 32nd repetition?

Start

Multiplier0 = 0
Multiplier0 = 1

No: < 32 repetitions
Yes: 32 repetitions

Done

Figure Copyright Morgan Kaufmann

Multiplication Hardware #1

- Multiplicand starts in right half of register
- MIPS: 64-bit product in Hi & Lo Regs
  - Move from Lo (mflo) to get 32-bit product
  - Move from hi (mfhi) to get upper 32-bits & test for overflow
**Multiplication Hardware #2**

- Shift Multiplicand Left ~ Shift Product Right
- Only need 32 bits for multiplicand

![Diagram of multiplication hardware](image)

- Possible to combine multiplier and product registers

**Multiplication Algorithm #2**

1. Test Multiplier0
   - Add multiplicand to the left half of the product and place the result in the left half of the Product register
2. Shift the Product register right 1 bit
3. Shift the Multiplier register right 1 bit

Start

![Flowchart of multiplication algorithm](image)

© Alvin R. Lebeck

© Alvin R. Lebeck
Booth Encoding

• Observation:
  ➢ Can write number as difference of two numbers.
  ➢ In particular: Can replace a string of 1s with initial subtract when we see a 1, and then an add when we see the bit (which will be a zero) AFTER the last 1

• Example 1: $7_{10}$
  ➢ $7_{10} = -1_{10} + 8_{10}$
  ➢ $0111_2 = -0001_2 + 1000_2$

• Example 2: $110_{10} = 01101110_2$
  ➢ $110_{10} = (-2_{10} + 16_{10}) + (-32_{10} + 128_{10})$
  ➢ $01101110_2 = (-00000010_2 + 00010000_2) + (-00100000_2 + 10000000_2)$

• Works for signed numbers as well!

Booth’s Algorithm

• Similar to previous multiply algorithm.

• (Current, Previous) bits of Multiplier:
  ➢ 0,0: middle of string of 0s; do nothing
  ➢ 0,1: end of a string of 1s; add multiplicand
  ➢ 1,0: start of string of 1s; subtract multiplicand
  ➢ 1,1: middle of string of 1s; do nothing

• Shift Product/Multiplier right 1 bit (as before)
**Signed Multiplication**

- Convert negative numbers to positive and remember the original signs.
- In 2s-complement, can multiply directly using Booth’s Algorithm.
  - Sign extend when shifting.

**Integer Division**

- Dividend = Quotient x Divisor + Remainder
- Example: 1,001,010\text{_{ten}} / 1000\text{_{ten}}

\[
\begin{array}{c|c|c|c|c|c}
\text{Divisor} & \text{1000}_{\text{ten}} & \text{Quotient} \\
\hline
\text{1} & \text{0} & \text{0} & \text{1}_{\text{ten}} & \text{1010}_{\text{ten}} & \text{Dividend} \\
\hline
-1 & 0 & 0 & 0 & & \\
\hline
1 & 0 & & & & \\
1 & 0 & 1 & & & \\
1 & 0 & 1 & 0 & & \\
-1 & 0 & 0 & 0 & & \\
\hline
1 & 0 & & & \text{Remainder} & \\
\end{array}
\]
Division Hardware #1

- Divisor starts in left half of divisor register

![Diagram of division hardware](image)

Division (contd.)

- Similar to multiplication
  - Shift remainder left instead of shifting divisor right
  - Combine quotient register with right half of remainder register
  - MIPS: Hi contains remainder, Lo contains quotient

- Signed Division
  - Remember the signs and negate quotient if different.
  - Make sign of remainder match the dividend

- Same hardware can be used for both multiply and divide.
  - Need 64-bit register that can shift left and right
  - ALU that adds or subtracts
  - Optimizations possible
Summary

- Storage elements
  - S-R latch, D-Latch, D Flip-Flop
- Register File
- Integer Multiplication & Division

- Homework #3 Due March 2
- Project: read description.