Storage Elements, Busses, Integer Arithmetic

CPS 104
Lecture 10

Homework #4 Due March 6
Project specification on the web, Due April 14
Partners determined by Thursday or we assign
Email kevin (yike@cs.duke.edu)

Outline
• Review (work problems)
• Logic for Storage
• Register File
• Busses
• Integer Multiply and Divide

Reading
Appendix B.5, 4.6-4.8

Review: Boolean Functions
• Boolean functions have arguments that take two values (T,F) or (0,1) and they return a single or a set of (T,F) or (0,1) value(s).
• Boolean functions can always be represented by a table called a “Truth Table”
• Example: \( F: \{0,1\}^3 \rightarrow \{0,1\}^2 \)

\begin{array}{ccc}
\textbf{a} & \textbf{b} & \textbf{c} \\
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
1 & 1 & 1 \\
\end{array}

Review: Boolean Functions and Expressions

\[ F(A, B, C) = (A \cdot B) + (\neg A \cdot C) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Review: Boolean Gates
• Gates are electronics devices that implement simple Boolean functions

Examples

Review: Boolean Functions, Gates and Circuits
• Circuits are made from a network of gates. (function compositions)
Practice Problems

1. (5 pts) Write a logic function that is true if and only if \( X \) contains at least two 1s.

2. (5 pts) Implement the logic function from problem 1, using only AND, OR and NOT gates. (Note there are no constraints on the number of gate inputs.) By implement, I mean draw the circuit diagram.

3. (5 pts) Write a logic function that is true if and only if \( X \), when interpreted as an unsigned binary number, is greater than the number 5.

4. (5 pts) Implement the logic function from problem 3, using only AND, OR and NOT gates. (Note there are no constraints on the number of gate inputs.)

Logic Functions

1. (5 pts) Write a logic function that is true if and only if \( X \), when interpreted as a signed two's complement number is less than the number -2.

2. (5 pts) Implement the logic function from problem 5, using only AND, OR and NOT gates. (Note there are no constraints on the number of gate inputs.)

Parity Example

1. (15 pts) The parity code of a binary word counts the number of ones in a word. If there are an even number of ones the parity code is 0, if there are an odd number of ones the parity code is 1. For example, the parity of \( 0101 \) is 0, and the parity of \( 1101 \) is 1.

Construct the truth table for a function that computes the parity of a four-bit word. Implement this function using AND, OR and NOT gates. (Note there are no constraints on the number of gate inputs.)

Circuit to Function

• What is Truth Table and Function for this Circuit?

Review: A 1-bit Full Adder

Add/sub

Review: The new ALU Slice
ALU Modifications

1. (30 pts) Modify the 32-bit ALU of Figure 4.18 to support a new instruction (addpar) that performs four 8-bit additions in parallel. The 8-bits are taken from each of the 32-bit operands. For example, 32-bit input operand A is divided into four 8-bit operands A3 (bits 31-24), A2 (bits 23-16), A1 (bits 15-8), A0 (bits 0-7). For this new instruction, the addition is performed such that A0+B0, A1+B1, A2+B2, and A3+B3 occur simultaneously and independently (i.e., the operation of A0+B0 should not affect A1+B1, etc.) Draw a diagram similar to Figure 4.18 clearly showing your modifications to support this new instruction.

Review: Abstraction—The ALU

• General structure
  • Input
    • Two operands
    • Control
  • Output
    • Result
    • Overflow
    • Zero

Review: Circuit Example: Decoder

<table>
<thead>
<tr>
<th>I0</th>
<th>I1</th>
<th>Q0</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Review: Example 4x1 MUX

<table>
<thead>
<tr>
<th>S0</th>
<th>S1</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>
Set-Reset Latch (Continued)

\[
\begin{array}{c|c|c|c}
R & S & Q \\
0 & 0 & Q \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 1 & - \\
\end{array}
\]

Data Latch (D Latch)

\[
\begin{array}{c|c|c|c}
D & E & Q \\
0 & 1 & 0 \\
1 & 1 & 1 \\
- & 0 & Q \\
\end{array}
\]

D Flip-Flop

- On C, D is transferred to the first D latch and the second is stable.
- On C, 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

- 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.

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

\[Z \Rightarrow \text{High Impedance}\]

---

**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
### 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 (Ch 4.6)
- Integer Division (Ch 4.7)
- Floating Point Addition (Ch 4.8)
- Floating Point Multiplication (Ch 4.8)

### Integer Multiplication

- Product = Multiplicand x Multiplier
- Example: $0011_{10} \times 0101_{10}$

```
Multiplicand
\[ \begin{array}{c}
0 & 0 & 1 & 1 \\
\end{array} \]
Multiplier
\[ \begin{array}{c}
0 & 1 & 0 & 1 \\
\end{array} \]
Produce
\[ \begin{array}{c}
0 & 0 & 0 & 1 & 1 & 1 \\
\end{array} \]
```

### 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 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
- Possible to combine multiplier and product registers

### Multiplication Algorithm #2
1. Test
   - Multiplier0
2. Shift the Product register right 1 bit
3. Shift the Multiplier register right 1 bit
   - 32nd repetition?

### 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 AFTER the last 1
- Example 1: $7_{10}$
  - $7_{10} = -110 + 8_{10}$
  - $0111_2 = -0001_2 + 1000_2$
- Example 2: $11010 = 01101110_2$
  - $11010 = (-2_{10} + 16_{10}) + (-32_{10} + 128_{10})$
  - $01101110_2 = (-00000010_2 + 00010000_2) + (-00100000_2 + 10000000_2)$

### 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 × Divisor + Remainder
• Example: $1,001,010_{\text{ten}} / 1000_{\text{ten}}$

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Divisor} & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{Quotient} & 1 & 0 & 0 & 1 & 0 & 1 \\
\text{Dividend} & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
\text{Remainder} & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

Division Hardware #1

• Divisor starts in left half of divisor register

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 #4 Due March 6
• Project: form groups and read description.