Arithmetic

Hakim Weatherspoon
CS 3410, Spring 2012
Computer Science
Cornell University

See P&H 2.4 (signed), 2.5, 2.6, C.6, and Appendix C.6
Goals for today

Binary (Arithmetic) Operations

- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Subtraction (two’s compliment)
- Performance
Binary Addition

Addition works the same way regardless of base

- Add the digits in each position
- Propagate the carry

Unsigned binary addition is pretty easy

- Combine two bits at a time
- Along with a carry
1-bit Adder

Half Adder
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded
4-bit Adder

4-Bit Full Adder
- Adds two 4-bit numbers and carry in
- Computes 4-bit result and carry out
- Can be cascaded
4-bit Adder

- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out

- Carry-out = overflow indicates result does not fit in 4 bits
Arithmetic with Negative Numbers

Negative Numbers Complicate Arithmetic

Recall addition with negatives:
Arithmetic with Negative Numbers

Negative Numbers Complicate Arithmetic

Recall addition with negatives:

• pos + pos → add magnitudes, result positive
• neg + neg → add magnitudes, result negative
• pos + neg → subtract smaller magnitude, keep sign of bigger magnitude
First Attempt: Sign/Magnitude Representation

• 1 bit for sign (0=positive, 1=negative)
• N-1 bits for magnitude
Two’s Complement Representation

Better: Two’s Complement Representation

• Leading 1’s for negative numbers
• To negate any number:
  – complement all the bits
  – then add 1
## Two’s Complement

<table>
<thead>
<tr>
<th>Non-negatives (as usual):</th>
<th>Negatives (two’s complement: flip then add 1):</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0 = 0000</td>
<td></td>
</tr>
<tr>
<td>+1 = 0001</td>
<td></td>
</tr>
<tr>
<td>+2 = 0010</td>
<td></td>
</tr>
<tr>
<td>+3 = 0011</td>
<td></td>
</tr>
<tr>
<td>+4 = 0100</td>
<td></td>
</tr>
<tr>
<td>+5 = 0101</td>
<td></td>
</tr>
<tr>
<td>+6 = 0110</td>
<td></td>
</tr>
<tr>
<td>+7 = 0111</td>
<td></td>
</tr>
<tr>
<td>+8 = 1000</td>
<td></td>
</tr>
</tbody>
</table>
# Two’s Complement

<table>
<thead>
<tr>
<th>Non-negatives (as usual):</th>
<th>Negatives (two’s complement: flip then add 1):</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0 = 0000</td>
<td>~0 = 1111 -0 = 0000</td>
</tr>
<tr>
<td>+1 = 0001</td>
<td>~1 = 1110 -1 = 1111</td>
</tr>
<tr>
<td>+2 = 0010</td>
<td>~2 = 1101 -2 = 1110</td>
</tr>
<tr>
<td>+3 = 0011</td>
<td>~3 = 1100 -3 = 1101</td>
</tr>
<tr>
<td>+4 = 0100</td>
<td>~4 = 1011 -4 = 1100</td>
</tr>
<tr>
<td>+5 = 0101</td>
<td>~5 = 1010 -5 = 1011</td>
</tr>
<tr>
<td>+6 = 0110</td>
<td>~3 = 1001 -6 = 1010</td>
</tr>
<tr>
<td>+7 = 0111</td>
<td>~7 = 1000 -7 = 1001</td>
</tr>
<tr>
<td>+8 = 1000</td>
<td>~8 = 0111 -8 = 1000</td>
</tr>
</tbody>
</table>
Two’s Complement Facts

Signed two’s complement
• Negative numbers have leading 1’s
• zero is unique: +0 = -0
• wraps from largest positive to largest negative

N bits can be used to represent
• unsigned:
  – eg: 8 bits ⇒
• signed (two’s complement):
  – ex: 8 bits ⇒
Sign Extension & Truncation

Extending to larger size

Truncate to smaller size
Two’s Complement Addition

Addition with two’s complement signed numbers

- Perform addition as usual, regardless of sign (it just works)
Two’s Complement Addition

Addition with two’s complement signed numbers

- Perform addition as usual, regardless of sign (it just works)
Overflow

Overflow

• adding a negative and a positive?

• adding two positives?

• adding two negatives?
Overflow

Overflow

• adding a negative and a positive?

• adding two positives?

• adding two negatives?

Rule of thumb:

Overflow happened iff
   carry into msb != carry out of msb
Two’s Complement Adder

Two’s Complement Adder with overflow detection

\[ \text{overflow} \]

\[ A_3 \quad B_3 \quad A_2 \quad B_2 \quad A_1 \quad B_1 \quad A_0 \quad B_0 \]

\[ R_3 \quad R_2 \quad R_1 \quad R_0 \]

\[ 0 \]
Binary Subtraction

Two’s Complement Subtraction
Binary Subtraction

Two’s Complement Subtraction

\[ A - B = A + (-B) = A + (\overline{B} + 1) \]

Q: What if \((-B)\) overflows?
A Calculator

A 8
B 8
S
0=add
1=sub
A Calculator

0=add
1=sub
Efficiency and Generality

- Is this design fast enough?
- Can we generalize to 32 bits? 64? more?
Performance

Speed of a circuit is affected by the number of gates in series (on the *critical path* or the *deepest level of logic*)

![Diagram showing inputs, combinational logic, and expected outputs with the time delay $t_{combinational}$]
4-bit Ripple Carry Adder

- First full adder, 2 gate delay
- Second full adder, 2 gate delay
- ...
Critical Path

Which operation is the critical path?

• A) ADD/SUB
• B) AND
• C) OR
• D) LT
Critical Path

What is the length of the critical path (in gates)? (counting inverters)

- A) 3
- B) 5
- C) 9
- D) 11
What is the length of the critical path for a 32-bit ALU (in gates)? (counting inverters)

- A) 11
- B) 32
- C) 64
- D) 71
Recap

We can now implement any combinational (combinatorial) logic circuit

• Decompose large circuit into manageable blocks
  – Encoders, Decoders, Multiplexors, Adders, ...

• Design each block
  – Binary encoded numbers for compactness

• Can implement circuits using NAND or NOR gates

• Can implement gates using use P- and N-transistors

• And can add and subtract numbers (in two’s compliment)!

• Next, state and finite state machines...
Administrivia

Make sure you are
Registered for class, can access CMS
Have a Section you can go to
Have project partner in same Lab Section

Lab1 and HW1 are out
Both due in one week, next Monday, start early
Work alone
But, use your resources
• Lab Section, Piazza.com, Office Hours, Homework Help Session,
• Class notes, book, Sections, CSUGLab

Homework Help Session
Wednesday and Friday from 3:30-5:30pm
Location: 203 Thurston
Administrivia

Check online syllabus/schedule
• http://www.cs.cornell.edu/Courses/CS3410/2012sp/schedule.html

Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims (in evenings):
• Tuesday, February 28th
• Thursday, March 29th
• Thursday, April 26th

Schedule is subject to change
Stateful Components

Until now is combinatorial logic

- Output is computed when inputs are present
- System has no internal state
- Nothing computed in the present can depend on what happened in the past!

Need a way to record data
Need a way to build stateful circuits
Need a state-holding device

Finite State Machines
How can we store and change values?

(a) Ballots

(b) A → B

(c) S → Q → R

(d) All the above

(e) None

How do we create vote counter machine?
Unstable Devices
Bistable Devices

- Stable and unstable equilibria?

- In stable state, $\bar{A} = B$

- How do we change the state?
SR Latch

Set-Reset (SR) Latch
Stores a value Q and its complement Q

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>( \overline{Q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SR Latch

Set-Reset (SR) Latch
Stores a value $Q$ and its complement $\bar{Q}$

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>$\bar{Q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Unclocked D Latch

Data (D) Latch

<table>
<thead>
<tr>
<th>D</th>
<th>Q</th>
<th>(\overline{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Unclocked D Latch

Data (D) Latch

Data Latch

• Easier to use than an SR latch
• No possibility of entering an undefined state

When D changes, Q changes
  – ... immediately (after a delay of 2 Ors and 2 NOTs)

Need to control when the output changes
D Latch with Clock

Level Sensitive D Latch
Clock high:
set/reset (according to D)
Clock low:
keep state (ignore D)
D Latch with Clock

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

\[
\begin{array}{c|c|c|c}
\text{clk} & D & Q & \overline{Q} \\ \hline
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\end{array}
\]
D Latch with Clock

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

\[
\begin{array}{c|c|c|c}
clk & D & Q & \overline{Q} \\
0 & 0 & Q & \overline{Q} \\
0 & 1 & Q & \overline{Q} \\
1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\end{array}
\]
Clocks

Clock helps coordinate state changes

• Usually generated by an oscillating crystal
• Fixed period; frequency = 1/period

\[ \text{1} \quad \text{0} \]

- Diagram of clock waveform
Edge-triggering

- Can design circuits to change on the rising or falling edge
- Trigger on rising edge = positive edge-triggered
- Trigger on falling edge = negative edge-triggered
- Inputs must be stable just before the triggering edge

input _________
clock _________

Valid input: ✅
Invalid input: ❌
Clock Methodology

Clock Methodology

• Negative edge, synchronous

\[
\text{clk} \quad t_{\text{combinational}} \quad t_{\text{setup}} \quad t_{\text{hold}}
\]

– Signals must be stable near falling clock edge

• Positive edge synchronous

• Asynchronous, multiple clocks, . . .
Edge-Triggered D Flip-Flop

D Flip-Flop
- Edge-Triggered
- Data is captured when clock is high
- Outputs change only on falling edges
Clock Disciplines

Level sensitive
• State changes when clock is high (or low)

Edge triggered
• State changes at clock edge

positive edge-triggered

negative edge-triggered
Registers

- D flip-flops in parallel
- Shared clock
- Extra clocked inputs: write_enable, reset, ...

4-bit reg
An Example: What will this circuit do?