Outline

1. Addition and subtraction
2. Multiplication
3. Homework
Section outline

1. Addition and subtraction
   Overview
   Number systems
   Adders
Arithmetic/logic operations

- Increment
- Addition
- Negation
- Subtraction
- Multiplication
  - Slow or large
- Division
  - Slow or large
Arithmetic/logic operations

- Shift left
  - Fast, multiplication by two
- Shift right
  - Fast, division by two
- Bit-wise operations
  - AND, OR, NOT, NAND, NOR, XOR, and XNOR
Arithmetic

- Number systems review
- Adders
- Multipliers
- Memory overview
Arithmetic circuits

- Administration
- Number systems
- Adders
  - Ripple carry
  - Carry lookahead
  - Carry select
Section outline

1. Addition and subtraction
   Overview
   Number systems
   Adders
Number systems

- Representation of positive numbers same in most systems
- A few special-purpose alternatives exist, e.g., Gray code
- Alternatives exist for signed numbers
Given an $n$-bit number in which $d_i$ is the $i$th digit, the number is 
$$\sum_{i=1}^{n} 2^{i-1} d_i$$
Consider adding 9 (1001) and 3 (0011)

\[
\begin{array}{cccc}
1 & 0 & 0 & 1 \\
+ & 0 & 0 & 1 \\
\hline
0 & 0 & 1 & 1 \\
\end{array}
\]
Unsigned addition

Consider adding 9 (1001) and 3 (0011)

```
  1 0 0 1
+ 0 0 1 1
  __________
     0 0 1 1
```

Why an extra column?
Consider adding 9 (1001) and 3 (0011)

\[
\begin{array}{cccc}
1 & 1 \\
1 & 0 & 0 & 1 \\
+ & 0 & 0 & 1 & 1 \\
\hline
0 & 0 & 0 & 0 \\
\end{array}
\]
Consider adding 9 (1001) and 3 (0011)

\[
\begin{array}{c}
1 \\
1 \\
1 \\
+ \\
0 \\
0 \\
0 \\
\hline
1 \\
0 \\
0 \\
\end{array}
\]
Consider adding 9 (1001) and 3 (0011)

\[
\begin{array}{cccc}
1 & 1 \\
1 & 0 & 0 & 1 \\
+ & 0 & 0 & 1 & 1 \\
\hline
1 & 1 & 0 & 0 \\
\end{array}
\]
Consider adding 9 (1001) and 3 (0011)

\[
\begin{array}{cccc}
1 & 1 \\
1 & 0 & 0 & 1 \\
+ & 0 & 0 & 1 & 1 \\
\hline
1 & 1 & 0 & 0
\end{array}
\]

Why an extra column?
Overflow

- If the result of an operation can’t be represented in the available number of bits, an overflow occurs.
- E.g., $0110 + 1011 = 10001$
- Need to detect overflow.
Overflow

- If the result of an operation can't be represented in the available number of bits, an overflow occurs
- E.g., $0110 + 1011 = 10001$
- Need to detect overflow
Gray code

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0
1
3
2
7
6
4
5
15 (f)
14 (e)
12 (c)
13 (d)
8
9
11 (b)
10 (a)

Robert Dick
Advanced Digital Logic Design
Gray code

- To convert from a standard binary number to a Gray code number XOR the number by it’s half (right-shift it)
- To convert from a Gray code number to a standard binary number, XOR each binary digit with the parity of the higher digits

Given that a number contains $n$ digits and each digit, $d_i$, contributes $2^{i-1}$ to the number

$$P_j^k = d_j \oplus d_{j+1} \cdots \oplus d_{k-1} \oplus d_k$$

$$d_i = d_i \oplus P_{i+1}^n$$
Gray code

- Converting from Gray code to standard binary is difficult
  - Take time approximately proportional to $n$
- Doing standard arithmetic operations using Gray coded numbers is difficult
- Generally slower than using standard binary representation
- E.g., addition requires two carries
- Why use Gray coded numbers?
  - Analog to digital conversion
  - Reduced bus switching activity
Signed number systems

- Three major schemes
  - Sign and magnitude
  - One’s complement
  - Two’s complement
Number system assumptions

- Four-bit machine word
- 16 values can be represented
- Approximately half are positive
- Approximately half are negative
Sign and magnitude
Sign and magnitude

- $d_n$ represents sign
  - 0 is positive, 1 is negative
- Two representations for zero
- What is the range for such numbers?

Range: $[-2^{n-1} + 1, 2^{n-1} - 1]$
Sign and magnitude

- $d_n$ represents sign
  - 0 is positive, 1 is negative
- Two representations for zero
- What is the range for such numbers?
  - Range: $[-2^{n-1} + 1, 2^{n-1} - 1]$
Sign and magnitude

- How is addition done?
- If both numbers have the same sign, add them like unsigned numbers and preserve sign
- If numbers have differing signs, subtract smaller magnitude from larger magnitude and use sign of large magnitude number
Sign and magnitude

- Consider $5 + (-6)$
- Note that signs differ
- Use magnitude comparison to determine large magnitude: $6 - 5$
- Subtract smaller magnitude from larger magnitude: $1$
- Use sign of large magnitude number: $-1$
Direct subtraction

Consider subtracting 5 (0101) from 6 (0110)

\[
\begin{array}{cccc}
0 & 1 & 1 & 0 \\
- & 0 & 1 & 0 \\
\hline
0 & 1 & 0 & 1 \\
\end{array}
\]

- Note that this operation is different from addition
- Sign and magnitude addition is complicated
Direct subtraction

Consider subtracting 5 (0101) from 6 (0110)

\[
\begin{array}{cccc}
  & b & 0 & 1 & 1 & 0 \\
 0 & 1 & 1 & 0 & 1 \\
- & 0 & 1 & 0 & 1 \\
\hline
  & 0 & 1 & 0 & 1 \\
\end{array}
\]

- Note that this operation is different from addition
- Sign and magnitude addition is complicated
Direct subtraction

Consider subtracting 5 (0101) from 6 (0110)

\[
\begin{array}{cccc}
  & b & 0 & 1 & 1 & 0 \\
\hline
- & 0 & 1 & 0 & 1 \\
\hline
  & 0 & 1 & 0 & 1 \\
\end{array}
\]

- Note that this operation is different from addition
- Sign and magnitude addition is complicated
Direct subtraction

Consider subtracting 5 (0101) from 6 (0110)

\[
\begin{array}{cccc}
& 0 & 1 & 1 & 0 \\
- & 0 & 1 & 0 & 1 \\
\hline
  & 0 & 0 & 1 \\
\end{array}
\]

- Note that this operation is different from addition
- Sign and magnitude addition is complicated
Direct subtraction

Consider subtracting 5 (0101) from 6 (0110)

\[
\begin{array}{cccc}
  & b & 0 & 1 & 1 & 0 \\
 0 & 1 & 1 & 0 \\
\hline
- & 0 & 1 & 0 & 1 \\
\hline
  & 0 & 0 & 0 & 1 \\
\end{array}
\]

- Note that this operation is different from addition
- Sign and magnitude addition is complicated
One’s compliment

![One's compliment representation](image)

-0000
-0001
-0010
-0011
-0100
-0101
-0110
-0111
-1000
-1001
-1010
-1011
-1100
-1101
-1110
-1111
-2
-3
-4
-5
-6
-7
0
1
2
3
4
5
6
7
One’s compliment

- If negative, complement all bits
- Addition somewhat simplified
- Do standard addition except wrap around carry back to the 0th bit
- Potentially requires two additions of the whole width
  - Slow
One’s complement addition

Consider adding -5 (1010) and 7 (0111)

\[
\begin{array}{c}
1 & 0 & 1 & 0 \\
+ & 0 & 1 & 1 & 1 \\
\hline
0 & 1 & 1 & 1
\end{array}
\]
One’s complement addition

Consider adding -5 (1010) and 7 (0111)

\[
\begin{array}{c}
1 \\
+ 0 1 1 1 \\
\hline
0 1 1 1
\end{array}
\]

1
One’s complement addition

Consider adding -5 (1010) and 7 (0111)

```
  1
+ 1 0 1 0
+ 0 1 1 1
-----
  0 1 1
```
One’s complement addition

Consider adding -5 (1010) and 7 (0111)

\[
\begin{array}{c c c c c c c c c}
1 & 1 \\
1 & 0 & 1 & 0 \\
+ & 0 & 1 & 1 & 1 \\
\hline
0 & 0 & 1 \\
\end{array}
\]
One’s complement addition

Consider adding -5 (1010) and 7 (0111)

\[
\begin{array}{c}
1 \\
1 \\
1 \\
0 \\
0 \\
+ \\
0 \\
\end{array}
\begin{array}{c}
1 \\
0 \\
1 \\
1 \\
\hline
1 \\
0 \\
0 \\
0 \\
\end{array}
\begin{array}{c}
1 \\
0 \\
0 \\
0 \\
1 \\
\end{array}
\]
One’s complement addition

Consider adding -5 (1010) and 7 (0111)

```
  1 1 1 1
  1 0 1 0
```

```
+ 0 1 1 1
```

```
  0 0 0 0
  0 0 0 0
```
Consider adding -5 (1010) and 7 (0111)

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 \\
+ & 0 & 1 & 1 & 1 \\
\hline
0 & 0 & 1 & 0 \\
\end{array}
\]
Two’s complement

- To negate a number, invert all its bits and add 1
- Like one’s complement, however, rotated by one bit
- Counter-intuitive
  - However, has some excellent properties
Two’s complement
Two’s complement

- Only one zero
  - Leads to more natural comparisons
- One more negative than positive number
  - This difference is irrelevant as $n$ increases
- Substantial advantage – Addition is easy!
Two’s complement addition

Consider adding -4 (1100) and 6 (0110)

\[
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
\hline
+ & 0 & 1 & 1 & 0 \\
\hline
\end{array}
\]

\[
0 & 1 & 1 & 0
\]
Two’s complement addition

Consider adding -4 (1100) and 6 (0110)

\[
\begin{array}{cccccc}
1 & 1 & 0 & 0 & \\
+ & 0 & 1 & 1 & 0 \\
\hline
0 & 1 & 1 & 0 & 0
\end{array}
\]
Two’s complement addition

Consider adding -4 (1100) and 6 (0110)

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 \\
+ & 0 & 1 & 1 & 0 \\
\hline
1 & 0
\end{array}
\]
Two’s complement addition

Consider adding -4 (1100) and 6 (0110)

\[
\begin{array}{c}
1 \\
1 1 0 0 \\
+ 0 1 1 0 \\
\hline \\
0 1 0
\end{array}
\]
Two’s complement addition

Consider adding -4 (1100) and 6 (0110)

```
  1  1
  1 1 0 0
+ 0 1 1 0
  ___________
  0 0 1 0
```
Two’s complement

- No looped carry – Only one addition necessary
- If carry-in to most-significant bit $\neq$ carry-out to most-significant bit, overflow occurs
- What does this represent?
- Both operands positive and have carry-in to sign bit
- Both operands negative and don’t have carry-in to sign bit
Two’s complement overflow

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>
# Two’s complement overflow

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>cin</th>
<th>cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>
Section outline

1. Addition and subtraction
   Overview
   Number systems
   Adders
## Half adder review

For two’s complement, don’t need subtracter

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>cout</th>
<th>sum</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\text{cout} = AB
\]

\[
\text{sum} = A \oplus B
\]
Half adder review

```
A
--|--
 | 
| | 
---|--
| |
| |
| |
B
--|--
 | 
| | 
---|--
| |
| |
| |
```
### Full adder review

Need to deal with carry-in

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>cin</th>
<th>cout</th>
<th>sum</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Full adder

\[
\text{sum} = A \oplus B \oplus \text{cin} \\
\text{cout} = AB + A \text{ ci} + B \text{ ci}
\]
Cascaded full-adders
Full adder standard implementation

Six logic gates
Full adder composed of half-adders

\[ AB + ci(A \oplus B) = AB + B \, ci + A \, ci \]
Adder/subtractor

Consider input to cin
The critical path (to $cout$) is two gate delays per stage

Consider adding two 32-bit numbers

- 64 gate delays
  - Too slow!

Consider faster alternatives
Carry lookahead adder

- Carry generate: $G = AB$
- Carry propagate: $P = A \oplus B$
- Represent *sum* and *cout* in terms of $G$ and $P$
Carry lookahead adder

\[\text{sum} = A \oplus B \oplus \text{cin} \]
\[= P \oplus \text{cin}\]

\[\text{cout} = AB + A \text{ cin} + B \text{ cin}\]
\[= AB + \text{ cin}(A + B)\]
\[= AB + \text{ cin}(A \oplus B)\]
\[= G + \text{ cin} P\]
Carry lookahead adder

Flatten carry equations

\[
\begin{align*}
\text{cin}_1 &= G_0 + P_0 \text{ cin}_0 \\
\text{cin}_2 &= G_1 + P_1 \text{ cin}_1 = G_1 + P_1 G_0 + P_1 P_0 \text{ cin}_0 \\
\text{cin}_3 &= G_2 + P_2 \text{ cin}_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 \text{ cin}_0 \\
\text{cin}_4 &= G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 \text{ cin}_0
\end{align*}
\]

Each \text{cin} can be implemented in three-level logic
Carry lookahead building block

- $A_i$, $B_i$, $C_i$ are inputs.
- $P_i$, $S_i$, $G_i$ are outputs.
- $1$ gate delay for $P_i$.
- $2$ gate delays for $S_i$.
- $1$ gate delay for $G_i$. 

1 gate delay

2 gate delays

1 gate delay
Carry lookahead adder
Assume a 4-stage adder with CLA
- Propagate and generate signals available after 1 gate delays
- Carry signals for slices 1 to 4 available after 3 gate delays
- Sum signal for slices 1 to 4 after 4 gate delays
Carry lookahead

- No carry chain slowing down computation of most-significant bit
  - Computation in parallel
- More area required
- Each bit has more complicated logic than the last
- Therefore, limited bit width for this type of adder
- Can chain multiple carry lookahead adders to do wide additions
- Note that even this chain can be accelerated with lookahead
  - Use internal and external carry lookahead units
Cascaded carry lookahead adder
Delay analysis for cascaded carry lookahead

- Four-stage 16-bit adder
  - \texttt{cin} for MSB available after five gate delays
  - \texttt{sum} for MSB available after eight gate delays
- 16-bit ripple-carry adder takes 32 gate delays
- Note that not all gate delays are equivalent
- Depends on wiring, driven load
- However, carry lookahead is usually much faster than ripple-carry
Carry select adders

- Trade even more hardware for faster carry propagation
- Break a ripple carry adder into two chunks, low and high
- Implement two high versions
  - $high_0$ computes the result if the carry-out from low is 0
  - $high_1$ computes the result if the carry-out from low is 1
- Use a MUX to select a result once the carry-out of low is known
  - $high_0$’s cout is never greater than $high_1$’s cout so special-case MUX can be used
Carry select adder
Delay analysis of carry select adder

- Consider 8-bit adder divided into 4-bit stages
- Each 4-bit stage uses carry lookahead
- The 2:1 MUX adds two gate delays
- 8-bit sum computed after 6 gate delays
- 7 gate delays for carry lookahead
- 16 gate delays for ripple carry
Outline

1. Addition and subtraction
2. Multiplication
3. Homework
Digital logic circuits frequently need to carry out arithmetic operations
- Addition, subtraction, and multiplication
- A number of design decisions affect the performance, area, and power consumption of arithmetic sub-circuits
- Number systems
- Trade-off between area/power consumption and speed
To understand why these trade-offs exist, we need to understand the fundamentals of arithmetic circuits.

We have already discussed the selection of number systems and the design of adders/subtracters.

Similar alternatives exist for multipliers.
Multiplication

- Multiplication is the repeated application addition of ANDed bits and shifting (multiplying by two)
- Multiplication is the sum of the products of each bit of one operand with the other operand
- Consequence: A product has double the width of its operands
Recall that multiplying a number by two shifts it to the left one bit.

\[
6 \cdot 3 = 6 \cdot (2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 1)
\]

\[
= 6 \cdot 2^2 \cdot 0 + 6 \cdot 2^1 \cdot 1 + 6 \cdot 2^0 \cdot 1
\]

\[
110 \cdot 011 = 11000 \cdot 0 + 1100 \cdot 1 + 110 \cdot 1
\]

\[
= 110 + 1100
\]

\[
= 10010
\]

\[
= 18
\]
Multiplication

\[
\begin{array}{c}
A_2 B_2 \\
A_1 B_2 \\
A_0 B_2 \\
\hline
sum_5 \\
sum_4 \\
sum_3 \\
\hline
A_2 B_1 \\
A_1 B_1 \\
A_0 B_1 \\
\hline
sum_2 \\
sum_1 \\
sum_0 \\
\hline
A_2 B_0 \\
A_1 B_0 \\
A_0 B_0 \\
\hline
\end{array}
\]
Consider multiplying 6 (110) by 3 (011)

\[
\begin{array}{c}
1 \\
\times \\
0 \\
\hline
1 \\
1 \\
1 \\
\hline
0 \\
1 \\
0 \\
\hline
0 \\
1 \\
0 \\
0 \\
1 \\
0 \\
\end{array}
\]
Multiplier implementation

- Direct implementation of this scheme possible
- Partial products formed with ANDs
- For four bits, 12 adders and 16 gates to form the partial products
  - 88 gates
- Note that the maximum height (number of added bits) is equal to the operand width
Combinational multiplier
Multiplier building block

Diagram showing the connections between Sum In, X, Cin, Y, FA, Cout, and Sum Out.
Combinational multiplier

- Addition and subtraction
- Multiplication
- Homework

Combinational multiplier diagram with inputs A2, A1, A0, B2, B1, B0 and outputs P5, P4, P3, P2, P1, P0.
Sequential multiplier

- Can iteratively one row of adders to carry out multiplications
- Advantage: Area reduced to approximately its square root
- Disadvantage: Takes \( n \) clock cycles, where \( n \) is the operand bit width
2X2 sequential multiplier

Operands already in registers
Adder flip-flops cleared
2X2 sequential multiplier

Operands already in registers
Adder flip-flops cleared

Could be HA

Operands

Could be HA
Arithmetic/logic units

- Possible to implement functional units that can carry out many arithmetic and logic operations with little additional area or delay overhead
- Already saw example: Combined adder/subtractor
- Other operations possible
- Could you generalize the approach used for two’s compliment addition and subtraction to another pair of operations?
Arithmetic/logic operations

- Increment
- Addition
- Negation
- Subtraction
- Multiplication
  - Slow or large
- Division
  - Slow or large
Arithmetic/logic operations

- **Shift left**
  - Fast, multiplication by two
- **Shift right**
  - Fast, division by two
- **Bit-wise operations**
  - AND, OR, NOT, NAND, NOR, XOR, and XNOR
Memory types

- ROM, PROM, EPROM, EEPROM: Already know these
- SRAM: Fast, low-density, relies on feedback
- DRAM: Fast, high-density, requires refresh, relies on stored charge
- Flash: Already know these – Non-volatile, slow, relies on floating gate
Outline

1. Addition and subtraction
2. Multiplication
3. Homework
Recommended reading

- Chapter 9