ENEE244-010x
Digital Logic Design

Lecture 2
Announcements

• Check updated UTF Office Hours on Syllabus/Webpage
• First homework assigned (see course webpage). Due date: Sept. 9 in class.
• Readings now up on course webpage
• First recitation is tomorrow (Thursday)!
Agenda

• Last time:
  – Positional Number Systems (2.1)
  – Basic Arithmetic Operations (2.3)
  – Polynomial Method of Number Conversion (2.4)

• This time:
  – Polynomial Method of Number Conversion (2.4)
  – Iterative Method of Number Conversion (2.5)
  – Special Conversion Procedures (2.6)
  – Signed numbers and Complements
  – Addition and Subtraction with Complements
Polynomial method of number conversion

- Convert from base $r_1$ to base $r_2$
- Express number as polynomial in base $r_1$
  \[ -N = d_2 \times r_1^2 + d_1 \times r_1^1 + d_0 \times r_1^0 \]
- Switch each digit symbol $d_i$ to its base $r_2$ representation and each base symbol $r_1$ to its base $r_2$ representation.
- Evaluate the polynomial in base $r_2$. 
Polynomial Method of Number Conversion

• Example: convert from hexadecimal to decimal
• Hexadecimal number: C53B

\[- C53B = C \times (10_{16})^3 + 5 \times (10_{16})^2 + 3 \times (10_{16})^1 + B \times (10_{16})^0 \]
\[- C53B = (12) \times (16)^3 + (5) \times (16)^2 + (3) \times (16)^1 + (11) \times (16)^0 \]
\[- C53B = 50491 \]

• **Use this method when converting a number into decimal form (e.g. binary to decimal)**

• Why?
Iterative Method of Number Conversion

• Convert from base $r_1$ to base $r_2$.
• Perform repeated division by $r_2$. The remainder is the digit of the base $r_2$ number.
• Example: Convert 50 from decimal to binary
  – Divide 50 by 2, get 25 remainder 0
  – Divide 25 by 2, get 12 remainder 1
  – Divide 12 by 2, get 6 remainder 0
  – Divide 6 by 2, get 3 remainder 0
  – Divide 3 by 2, get 1 remainder 1
  – Divide 1 by 2, get 0 remainder 1
• Answer is: 110010
• Can verify using the polynomial method
• **Use when converting from decimal to another base. (e.g. decimal to binary)
• Why?
Iterative Method for Converting Fractions

• Convert from base $r_1$ to base $r_2$.
• Perform repeated multiplication by $r_2$. The integer part is the digit of the base $r_2$ number.
• Ex: Convert .40625 from decimal to binary
  – Multiply .40625 by 2, get 0 + .8125
  – Multiply .8125 by 2, get 1 + .625
  – Multiply .625 by 2, get 1 + .25
  – Multiply .25 by 2, get 0 + .5
  – Multiply .5 by 2, get 1 + 0
• Answer is: .01101
• Can verify using the polynomial method
Special Conversion Procedures

• When converting between two bases in which one base is a power of the other, conversion is simplified.

• Ex: Convert from 1101 0110 1111 1001 from binary to hexadecimal:
  – 1101 = 13 = D
  – 0110 = 6
  – 1111 = 15 = F
  – 1001 = 9

• Answer: D6F9
Signed Numbers and Complements
Range of represented numbers

• Let $\ell$ be the number of binary digits that can be stored.

• Example: Store data in a single byte (8 bits).

• Using a single byte can represent unsigned numbers from 0 to 255 ($2^8 = 256$ different values).

• Alternatively, can represent the signed numbers from -128 to 127 in same amount of space ($2^7 = 128$).
Signed Numbers and Complements

• How to denote if a number is positive or negative?
  – Use a sign bit: $0_s 1001$ denotes positive 9, $1_s 1001$ denotes negative 9. This representation is called the sign-magnitude representation.
  – This works, but it will be convenient to use a different representation of negative numbers.

• Two methods: 2’$s$ complement and 1’$s$ complement.
  – Idea: Subtraction is hard! Addition is easy!
  – Convert every subtraction problem to an addition problem
    • Example: Instead of computing 01000101 − 00110100, instead compute 01000101 + (−00110100).
2’s Complement

• 2’s complement of $N = 2^\ell - N = (10)_2^\ell - N$

• In our example (one byte of memory), to represent -9, (where 9 = 1001 in binary), compute $(10_2)_8 - 1001 = 100000000 - 1001 = 11110111$
1’s Complement

• 1’s complement of $N = 2^\ell - 1 - N = (10_2)^\ell - 1 - N$

• In our example, to represent -9, (where 9 = 1001 in binary), compute $(10_2)^8 - 1 - 1001 = 11111111 - 1001 = 11110110$

• This corresponds to flipping the bits of 00001001.
In-Class Exercise

• Subtraction using 2’s complement, 1’s complement
2’s Complement

• Notice for negative numbers, most significant bit is always 1. For positive numbers, most significant bit is always 0.

• This bit is therefore called the sign bit.
Subtraction Using 2’s Complement

• Just do addition as usual
• Ignore highest order carry
• Aside: This is equivalent to doing arithmetic modulo $2^\ell$. 
1’s Complement

• Again, for negative numbers, nth digit is always 1. For positive numbers, nth digit is always 0.

• There are now two ways to represent 0: 00000000 or 11111111
Subtraction using 1’s complement

• Do addition as usual
• If there is an end carry, add it to the least significant bit.
• Most significant bit tells you the sign.
Fast(er) way to compute 2’s complement

• To form the 2’s complement of 0110 1010:
  – Take the 1s complement:  1001 0101
  – Then add 1:  1001 0110
Advantages/Disadvantages of 1’s vs. 2’s complement

<table>
<thead>
<tr>
<th>1s complement</th>
<th>2s complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to compute (just flip bits)</td>
<td>Harder to compute (flip bits and add one)</td>
</tr>
<tr>
<td>Harder to manipulate (e.g., for subtraction, need to add in extra carry.)</td>
<td>Easy to manipulate (e.g., subtraction is the same as addition—no extra hardware needed)</td>
</tr>
</tbody>
</table>