CPE300: Digital System Architecture and Design

Fall 2011
MW 17:30-18:45 CBC C316

Arithmetic Unit
10102011

http://www.egr.unlv.edu/~b1morris/cpe300/
Chapter 6

- Number Systems and Radix Conversion
- Fixed-Point Arithmetic
- Seminumeric Aspects of ALU Design
- Floating-Point Arithmetic
Outline

- Number Systems
- Fixed Point Arithmetic
Digital Number Systems

- Expanded generalization of lecture 07 topics
- Number systems have a base (radix) \( b \)
- Positional notation of an \( m \) digit base \( b \) number
  - \( x = x_{m-1}x_{m-2} \ldots x_1x_0 \)
  - Value \( (x) = \sum_{i=0}^{m-1} x_i b^i \)
Range of Representation

- Largest number has all digits equal to largest possible base $b$ digit, $(b - 1)$
- Max value in closed form for unsigned $m$ digit base $b$ number
  - $x_{\text{max}} = \sum_{i=0}^{m-1} (b - 1)b^i$
  - $x_{\text{max}} = (b - 1)\sum_{i=0}^{m-1} b^i = (b - 1) \left( \frac{b^{m-1}}{b-1} \right)$
  - $x_{\text{max}} = b^m - 1$

- Sum of geometric series
  - $\sum_{i=0}^{m-1} b^i = \left( \frac{b^{m-1}}{b-1} \right)$
Radix Conversion

• Conversion between different number systems involves computation
  ▫ Base of calculation is c (10 typical for us humans)
  ▫ Other base is b

• Calculation based on division
  ▫ For integers a and d, exist integers q and r such that
    ▫ \( a = q \cdot d + r \)
      • \( 0 \leq r \leq b - 1 \)

• Notation:
  ▫ \( q = [a/d] \)
  ▫ \( r = a \mod b \) \hspace{1mm} (mod is remainder)
Digit Symbol Correspondence Between Bases

- Each base (b or c) has different symbols to represent digits
- Lookup table given for correspondence between symbols
  - Provides mapping between base b and base c symbols
  - May be more than one digit required to represent a larger base symbol

<table>
<thead>
<tr>
<th>Base 12</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>100</td>
<td>101</td>
<td>102</td>
</tr>
</tbody>
</table>
Base Conversion 1

- Convert base b integer to calculator base c

1. Start with base b
   - \( x = x_{m-1}x_{m-2} \ldots x_1x_0 \)
2. Set \( x = 0 \) in base c
3. Left to right, get next symbol \( x_i \)
4. Lookup base c number \( D_i \) for symbol \( x_i \)
5. Calculate in base c
   - \( x = x \cdot b + D_i \)
6. Repeat step 3 until no more digits

Example:
- Convert \( 0x3AF \) to base 10
  - \( x = 0 \)
  - \( x = 16 \cdot 0 + 3 = 3 \)
  - \( x = 16 \cdot 3 + 10 (= A) = 58 \)
  - \( x = 16 \cdot 58 + 15 (= F) = 943 \)
- \( 0x3AF = 943_{10} \)
Base Conversion 2

- Convert calculator base c integer to base b

1. Start with base c integer
   - \( x = x_{m-1}x_{m-2} \ldots x_1x_0 \)

2. Initialize
   - \( i = 0 \)
   - \( v = x \)
   - Produce digits right to left

3. Set
   - \( D_i = v \mod b \)
   - \( v = \lfloor v/b \rfloor \)
   - Lookup \( D_i \) to get \( x_i \)

4. Set
   - \( i = i + 1 \)
   - Repeat step 3 if \( v \neq 0 \)

Example:
- Convert \( 3587_{10} \) to base 12
  - \( \frac{3587}{12} = 298 \ (rem = 11) \Rightarrow x_0 = B \)
  - \( \frac{298}{12} = 24 \ (rem = 10) \Rightarrow x_1 = A \)
  - \( \frac{24}{12} = 2 \ (rem = 0) \Rightarrow x_2 = 0 \)
  - \( \frac{2}{12} = 0 \ (rem = 2) \Rightarrow x_3 = 2 \)
- \( 3587 = 20AB_{12} \)
Fractions and Fixed Point Numbers

- **Base b fraction**
  - \( f = .f_{-1} f_{-2} \ldots f_m \)
  - Value is integer \( f_{-1} f_{-2} \ldots f_m \) divided by \( b^m \)

- **Mixed fixed point number**
  - \( x_{n-1} x_{n-2} \ldots x_1 x_0 . x_{-1} x_{-2} \ldots x_{-m} \)
  - Value of \( n+m \) digit integer
    - \( x_{n-1} x_{n-2} \ldots x_1 x_0 x_{-1} x_{-2} \ldots x_{-m} \)
  - Divided by \( b^m \)

- **Moving radix point one place left divides by b**
  - Right shift for fixed radix point position

- **Moving radix point one place right multiplies by b**
  - Left Shift for fixed radix point position
Converting Fractions to Calculator Base

- Can use integer conversion and divide result by $b^m$
- Alternative algorithm
  1. Let base b number be
     - $f = .f_{-1}f_{-2} \ldots f_{-m}$
  2. Initialize
     - $f = 0.0$
     - $i = -m$
  3. Find base c equivalent of $D$ of digit $f_i$
  4. Update
     - $f = \frac{f + D}{b}$
     - $i = i + 1$
  5. If $i = 0$, result is $f$; otherwise repeat step 3

- Example
  - Convert $0.413_8$ to base 10
    - $f = \frac{0+3}{8} = 0.375$
    - $f = \frac{(0.375+1)}{8} = 0.171875$
    - $f = \frac{0.171875+4}{8} = 0.521484375$
- Notice: there will be precision errors due to numerical round-off
  - Only a fixed number of digits can be retained
Converting Fractions to Base b

1. Start with fraction \( f \) in base \( c \)
   \[ f = .f_{-1}f_{-2} \ldots f_{-m} \]

2. Initialize
   - \( v = f \)
   - \( i = 1 \)

3. Set
   - \( D_{-i} = \lfloor b \cdot v \rfloor \)
   - \( v = b \cdot v - D_{-i} \)
   - Get base \( b \) digit \( f_{-i} \) for \( D_{-i} \)
     with table

4. Increment
   - \( i = i + 1 \)
   - Repeat Step 3 until
     - \( v = 0 \)
     - Enough digits generated

Example
- Convert \( 0.31_{10} \) to base 8
  \[ 0.31 \times 8 = 2.48 \Rightarrow f_{-1} = 2 \]
  \[ 0.48 \times 8 = 3.84 \Rightarrow f_{-2} = 3 \]
  \[ 0.84 \times 8 = 6.72 \Rightarrow f_{-3} = 6 \]

- \( f = 0.236_8 \)

Notice:
- Since \( 8^3 > 10^2 \), \( 0.236_8 \) has more accuracy than \( 0.31_{10} \)
Digit Grouping for Related Bases

- Base $b = c^k$
- Can convert between bases by replacing single digit symbol in base $b$ with corresponding digits in base $c$

(Our favorite method to change base e.g. binary to hex)

Examples
- $102130_4 = 102130_4 = 0x49C$
Negative Numbers and Complements

- Two complement operations defined
- Two complement number systems
  - Represent both positive and negative numbers

- Given m digit base b number x
- Radix complement (b’s complement)
  - \( x^c = (b^m - x) \mod b^m \)
  - mod \( b^m \) only has effect for \( x=0 \)
    - What is radix complement of \( x = 0 \)?
- Diminished radix complement ((b-1)’s complement)
  - \( \hat{x}_c = b^m - 1 - x \)
Complement Number Systems

- Both positive and negative numbers represented in m digits
  - Range of m digit base b unsigned number:
    - $0 \leq x \leq b^m - 1$
- First half of range used for positive and second half for negative numbers
  - Complement of number range
    - Positive: 0 to $b^m/2$
    - Negative: $b^m/2$ to $b^m-1$
  - Radix complement has extra negative number for even b (think b=2)
  - Diminished radix complement has equal numbers of positive and negative representations
Utility of Complement System

- Sign-magnitude system requires extra +/- symbols in addition to digits
  - Binary has easy mapping
    - + := 0
    - - := 1
  - If b > 2 a whole digit for the 2 +/- symbols is wasteful

- Easy to do signed addition and subtraction using the complement number systems
Complement Representation of Negative Numbers

<table>
<thead>
<tr>
<th>Radix Complement</th>
<th>Diminished Radix Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Representation</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$0 &lt; x &lt; b^m/2$</td>
<td>$x$</td>
</tr>
<tr>
<td>$-b^m/2 \leq x &lt; 0$</td>
<td>$</td>
</tr>
</tbody>
</table>

- Radix complement has one more negative than positive for even base $b$
- Diminished radix complement has 2 zeros but same number of positive and negative values
## Base 2 Complement Representations

<table>
<thead>
<tr>
<th>8 Bit Radix (2’s) Complement</th>
<th>8 bit Diminished Radix (1’s) Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Representation</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$0 &lt; x &lt; 128$</td>
<td>$x$</td>
</tr>
<tr>
<td>$-128 \leq x &lt; 0$</td>
<td>$256 -</td>
</tr>
</tbody>
</table>

- 1’s complement 255 (or -0)
  - $255 = 1111 \ 1111_2$
- 2’s complement
  - $-128 = 1000 \ 0000_2$ is valid
  - Negation gives overflow
Negation in Complement Systems

- Negative of any m digit value is also m digits
  - Exception: \(-b^m/2\)
- Negative of any number is obtained by applying the b’s or \((b-1)’s\) complement operation
- The complement operations are related
  \[ x^c = (\hat{x}^c + 1) \mod b^m \]
  - Given one, easy to compute other
Digitwise Computation of Diminished Radix Complement

- \( \hat{x}_c = bm - 1 - x \)
- \( \hat{x}_c = \sum_{i=0}^{m-1} (b - 1)b^i - \sum_{i=0}^{m-1} (x_i)b^i \)
- \( \hat{x}_c = \sum_{i=0}^{m-1} (b - 1 - x_i)b^i \)

- Diminished radix number is an m digit base b number
  - Each digit is obtained (as diminished complement) from corresponding digit in x
Base 5 Complements

<table>
<thead>
<tr>
<th>Base 5 Digit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4’s Comp.</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Examples**
- 4’s complement of $201341_5$
  - $243103_5$
- 5’s complement of $201341_5$
  - $243103_5 + 1 = 243104_5$
- 5’s complement of $44444_5$
  - $00000_5 + 1 = 00001_5$
  - $(44444_5 + 1) \mod 5^5 = 00000_5$
Complement Fractions

- m digit fraction is same as m digit integer divided by $b^m$,
  - The $b^m$ in complement definitions corresponds to 1 for fractions
- Radix complement of $f = .f_{-1}f_{-2}...f_{-m}$
  - $(1-x) \mod 1$
    - Where mod 1 means discard integer
- The range of fractions is roughly $-1/2$ to $+1/2$
- This can be inconvenient for a base other than 2
- The b’s comp. of a mixed number
  - $x = x_{m-1}x_{m-2}...x_1x_0.x_{-1}x_{-2}...x_{-n} = b^m - x$,
  - Both integer and fraction digits are subtracted
Scaling Complement Numbers

- Dividing by $b$ corresponds to moving radix point one place left
  - Shift number one place right
- Multiplying by base $b$ corresponds to moving radix point one place right (roughly)
  - Shift number one place left

- Issues:
  - What is new left digit on right shift?
  - When does left shift overflow?
Right Shift for Divide

- **Positive number** $x = x_{m-1}x_{m-2} \ldots x_1x_0$
  - Zero fill: $x/b = 0x_{m-1}x_{m-2} \ldots x_1$

- **Negative number**
  - $(b-1)$ file: $x/b = (b - 1)x_{m-1}x_{m-2} \ldots x_1$

- **Fill rule for even** $b$
  - Zero fill when $x_{m-1} < b/2$
  - $(b-1)$ fill when $x_{m-1} \geq b/2$
Left Shift for Multiply

- Overflow can occur (loss of information)
  - Positive numbers
    - Any digit other than 0 shifts off left end
    - After shift, left-most digit makes number look negative (digit $\geq b/2$ for even b)
  - Negative numbers
    - Any digit other than $(b-1)$ shifts off left end
    - After shift, left-most digit makes number look positive (digit $< b/2$ for even b)
Left Shift Examples

- **Non-overflow cases:**
  - $762_8 << 1 = 620_8$ ; $-14 \times 8 = -112$
  - $031_8 << 1 = 310_8$ ; $-25 \times 8 = -200$

- **Overflow cases**
  - $241_8 << 1 = 410_8$ ; $2 \neq 0$ off left
  - $041_8 << 1 = 410_8$ ; changes from + to −
  - $713_8 << 1 = 130_8$ ; changes from − to +
  - $662_8 << 1 = 620_8$ ; $2 \neq 7$ off left
Fixed Point Addition and Subtraction

- When radix point is in the same position for both operands
  - Add/Sub acts as if numbers were integers
- Addition of signed numbers in radix complement system only needs an unsigned adder
  - Must design m digit base b unsigned adder
- Radix complement signed addition theorem
  - $s = \text{rep}(x) + \text{rep}(y) = \text{rep}(x+y)$
  - $\text{rep}(x) := b$’s complement representation of $x$
  - Does not consider overflow
Unsigned Addition Hardware

- Perform operation on each digit of m digit base b number
- Each digit cell requires operands $x_j$ and $y_j$ as well as a carry in $c_j$
- Sum
  - $s_j = (x_j + y_j + c_j) \mod b$
- Carry-out
  - $c_{j+1} = \lfloor (x_j + y_j + c_j) / b \rfloor$
  - All carries are less than or equal to 1 regardless of b
- Works for any fixed radix point location (e.g. fractions)
Unsigned Addition Example

With fixed number of digits, overflow occurs on carry from leftmost digit

Carries are 0 or 1 in all cases

Addition is defined by a table of sum and carry for $b^2$ digit pairs
Adder Implementation Alternatives

- For base $b=2^k$, each digit is equivalent to $k$ bits.
- Adder can be viewed as a logic circuit with $2k+1$ inputs and $k+1$ outputs.

Ripple carry adder

- Choice of $k$ affects computation delay.
- When 2 level logic is used, what is the max gate delay for $m$ digit addition?
  - $2m$
Complement Subtractor

- Subtraction in radix complement is addition with negated (complemented) second input
  - Must supply overflow detection
- Radix complement is addition of 1 to diminished radix complement
  \( x^c = (\hat{x}^c + 1) \mod b^m \)
  - Easy to take diminished radix complement and use carry in of adder to supply +1 for radix complement
Overflow Detection

- Occurs when adding number of like sign and the result seems to have opposite sign
- For even b: sign determined by the leftmost digit
  - Overflow detector only requires $x_{m-1}, y_{m-1}, s_{m-1}$

Ripple-carry adder/subtractor

XOR gates select $y$ for addition or complement of $y$ for subtraction in base 2
Carry Lookahead

- Speed of addition depends on carries
  - Carries need to propagate from lsb to msb
- Two level logic for base $b$ digit becomes complex quickly for increasing $k$ ($b=2^k$)
  - Length of carry chain divided by $k$
- Need to compute carries quickly
  1. Determine if addition in position $j$ generates a carry
  2. Determine if carry is propagated from input to output of digit $j$
Binary Generate and Propagate Signals

- Generate: digit at position $j$ will have a carry
  - $G_j = x_j y_j$
- Propagate: carry in passes through to carry out
  - $P_j = x_j + y_j$
- Carry is defined as 1 if the sum generates a carry or if a carry is propagated
  - $c_{j+1} = G_j + P_j c_j$
Carry Lookahead Speed

- 4 bit carry equations
  - $c_1 = G_0 + P_0c_0$
  - $c_2 = G_1 + P_1G_0 + P_1P_0c_0$
  - $c_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0c_0$
  - $c_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$

- Carry lookahead delay
  - One gate delay for to calculate G or P
  - 2 levels of gates for a carry
  - 2 gate delays for full adder ($s_j$)

- The number of OR gate inputs (terms) and AND gate inputs (literals in a term) grows as the number of carries generated by lookahead
Recursive Carry Lookahead

- Apply lookahead logic to groups of digits
- Group of 4 digits (level 1)
  - Group generate:
    - \( G_0^1 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 \)
  - Group propagate:
    - \( P_0^1 = P_3P_2P_1P_0 \)
  - Can further define level 2 signals which are groups of level 1 groups
- Group k terms at each level \( \rightarrow \log_k m \) levels for m bit addition
  - Each level introduces 2 more gate delays
  - \( k \) chosen to trade-off reduced delay and complexity of \( G \) and \( P \) logic
    - Typically \( k \geq 4 \) however structure easier to see for \( k=2 \)
Carry Lookahead Adder Diagram

- Group size $k=2$
Digital Multiplication

- Based on digital addition
  - Generate partial products (from each digit) and sum for the complete product
  - “Pencil and paper addition”

\[
\begin{array}{cccccc}
  x_3 & x_2 & x_1 & x_0 & \text{Multiplicand} \\
  y_3 & y_2 & y_1 & y_0 & \text{Multiplier} \\
\end{array}
\]

\[
\begin{array}{cccccc}
  (xy_0)_4 & (xy_0)_3 & (xy_0)_2 & (xy_0)_1 & (xy_0)_0 \\
  pp_0 & \\
  (xy_1)_4 & (xy_1)_3 & (xy_1)_2 & (xy_1)_1 & (xy_1)_0 \\
  pp_1 & \\
  (xy_2)_4 & (xy_2)_3 & (xy_2)_2 & (xy_2)_1 & (xy_2)_0 \\
  pp_2 & \\
  (xy_3)_4 & (xy_3)_3 & (xy_3)_2 & (xy_3)_1 & (xy_3)_0 \\
  pp_3 & \\
  p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
\]
Accumulated Partial Product

- Partial products accumulated rather than collected and added in the end

1. for $i := 0$ step 1 until $2m-1$
2. \hspace{1em} $p_i := 0$
3. for $j := 0$ step 1 until $m-1$
4. \hspace{2em} begin
5. \hspace{3em} $c := 0$
6. \hspace{3em} for $i := 0$ step 1 until $m-1$
\hspace{4em} begin
7. \hspace{5em} $p_{j+i} := (p_{j+i} + x_i y_j + c) \mod b$
8. \hspace{5em} $c := \lfloor (p_{j+i} + x_i y_j + c)/b \rfloor$
9. \hspace{4em} end;
10. \hspace{3em} end;
11. \hspace{2em} $p_{j+m} := c$
12. \hspace{1em} end;

$c$ is a single base $b$ digit
(no longer $0, 1$ as in addition)
Parallel Array Multiplier
Parallel Array Multiplier Operation

- Each box in array does the base b digit calculations
  - $p_k(\text{out}): = (p_k(\text{in}) + xy + c(\text{in})) \mod b$
  - $c(\text{out}): = \lceil(p_k(\text{in}) + xy + c)/b \rceil$
- Inputs and outputs of boxes are single base b digits (including carries)
- Worst case path from input to output is about 6m gates if each box is a 2 level circuit
  - In binary, each box is a full adder with an extra AND gate to compute $xy$