ARITHMETIC COMBINATIONAL MODULES AND NETWORKS

- SPECIFICATION OF ADDER MODULES FOR POSITIVE INTEGERS
- HALF-ADDER AND FULL-ADDER MODULES
- CARRY-RIPPLE AND CARRY-LOOKAHEAD ADDER MODULES
- NETWORKS OF ADDER MODULES
- REPRESENTATION OF SIGNED INTEGERS:
  1. sign-and-magnitude
  2. two’s-complement
  3. ones’-complement
- ADDITION AND SUBTRACTION IN TWO’S COMPLEMENT
- ARITHMETIC-LOGIC UNITS (ALU)
- COMPARATOR MODULES AND NETWORKS
- MULTIPLICATION OF POSITIVE INTEGERS
CONVENTIONAL RADIX-2 NUMBER SYSTEM:

\[ x = (x_{n-1}, \ldots, x_0) \leftrightarrow x, \text{ integer} \]

\[ x = \sum_{i=0}^{n-1} x_i \times 2^i \]

• RANGE: 0 to \(2^n - 1\)
ADDER MODULES FOR POSITIVE INTEGERS

Figure 10.1: ADDER MODULE.

\[ x + y + c_{in} = 2^n c_{out} + z \]
A HIGH-LEVEL SPECIFICATION OF ADDER MODULE

INPUTS: \[ x = (x_{n-1}, \ldots, x_0), \quad x_j \in \{0, 1\} \]
\[ y = (y_{n-1}, \ldots, y_0), \quad y_j \in \{0, 1\} \]
\[ c_{in} \in \{0, 1\} \]

OUTPUTS: \[ z = (z_{n-1}, \ldots, z_0), \quad z_j \in \{0, 1\} \]
\[ c_{out} \in \{0, 1\} \]

FUNCTIONS: \[ z = (x + y + c_{in}) \mod 2^n \]
\[ c_{out} = \begin{cases} 1 & \text{if } (x + y + c_{in}) \geq 2^n \\ 0 & \text{otherwise} \end{cases} \]
**EXAMPLE for n=5**

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$c_{\text{in}}$</th>
<th>$z$</th>
<th>$c_{\text{out}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>14</td>
<td>1</td>
<td>$(12 + 14 + 1) \mod 32 = 27$</td>
<td>0 because $(12 + 14 + 1) &lt; 32$</td>
</tr>
<tr>
<td>19</td>
<td>14</td>
<td>1</td>
<td>$(19 + 14 + 1) \mod 32 = 2$</td>
<td>1 because $(19 + 14 + 1) &gt; 32$</td>
</tr>
</tbody>
</table>
### DELAY OF CARRY-RIPPLE ADDER

\[
t_p(\text{net}) = (n - 1)t_c + \max(t_z, t_c)
\]

\[
t_c = \text{Delay}(c_i \rightarrow c_{i+1})
\]

\[
t_z = \text{Delay}(c_i \rightarrow z_i)
\]
HIGH-LEVEL SPECIFICATION OF FULL-ADDER

\[ x_i + y_i + c_i = 2c_{i+1} + z_i \]

INPUTS: \( x_i, y_i, c_i \in \{0, 1\} \)

OUTPUTS: \( z_i, c_{i+1} \in \{0, 1\} \)

FUNCTION: \( z_i = (x_i + y_i + c_i) \mod 2 \)

\[ c_{i+1} = \begin{cases} 
1 & \text{if } (x_i + y_i + c_i) \geq 2 \\
0 & \text{otherwise} 
\end{cases} \]
FULL-ADDER IMPLEMENTATION

\[
\begin{array}{cccc}
  x_i & y_i & C_i & z_i \\
  \hline
  0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 1 \\
  0 & 1 & 0 & 0 & 1 \\
  0 & 1 & 1 & 1 & 0 \\
  1 & 0 & 0 & 0 & 1 \\
  1 & 0 & 1 & 1 & 0 \\
  1 & 1 & 0 & 1 & 0 \\
  1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
z_i = x_i y_i C_i + x_i y_i C_i' + x_i y_i C_i' + x_i y_i C_i
\]

\[
c_i+1 = x_i y_i + x_i C_i + y_i C_i
\]
FULL ADDER TWO-LEVEL IMPLEMENTATION

Figure 10.3: IMPLEMENTATIONS OF FULL-ADDER MODULE: a) TWO-LEVEL.
ALTERNATIVE IMPLEMENTATION

• ADDITION mod 2 → SUM IS 1 WHEN NUMBER OF 1’S IN INPUTS (including the carry-in) IS ODD:

\[ z_i = x_i \oplus y_i \oplus c_i \]

• CARRY-OUT IS 1 WHEN \((x_i + y_i = 2)\) or \((x_i + y_i = 1 \text{ and } c_i = 1)\):

\[ c_{i+1} = x_i y_i + (x_i \oplus y_i)c_i \]

• INTERMEDIATE VARIABLES

   PROPAGATE \[ p_i = x_i \oplus y_i \]
   GENERATE \[ g_i = x_i \cdot y_i \]

• HALF-ADDER

<table>
<thead>
<tr>
<th>(x_i)</th>
<th>(y_i)</th>
<th>(g_i)</th>
<th>(p_i)</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>
FA EXPRESSIONS IN TERMS OF \( p'_i \), \( g'_i \), and \( c'_i \)

\[
\begin{align*}
Z_i &= p_i \oplus c_i \\
C_{i+1} &= g_i + p_i \cdot c_i
\end{align*}
\]

Figure 10.3: IMPLEMENTATIONS OF FULL-Adder MODULE: b) MULTILEVEL GATE NETWORK WITH XORs, ANDs and OR; c) WITH XORs and NANDs.
WORST-CASE DELAY

\[ t_p = t_{XOR} + 2(n - 1)t_{NAND} + \max(2t_{NAND}, t_{XOR}) \]

Figure 10.2: CARRY-RIPPLE ADDER MODULE.
CHARACTERISTICS OF FULL-ADDER IN CMOS FAMILY

<table>
<thead>
<tr>
<th>Input</th>
<th>[standard loads]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_i$</td>
<td>1.3</td>
</tr>
<tr>
<td>$x_i$</td>
<td>1.1</td>
</tr>
<tr>
<td>$y_i$</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Size: 7 [equivalent gates]

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Propagation delays</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$t_{pLH}$ [ns]</td>
</tr>
<tr>
<td>$c_i$</td>
<td>$z_i$</td>
<td>$0.43 + 0.03L$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$z_i$</td>
<td>$0.68 + 0.04L$</td>
</tr>
<tr>
<td>$y_i$</td>
<td>$z_i$</td>
<td>$0.68 + 0.04L$</td>
</tr>
<tr>
<td>$c_i$</td>
<td>$c_{i+1}$</td>
<td>$0.36 + 0.04L$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$c_{i+1}$</td>
<td>$0.73 + 0.04L$</td>
</tr>
<tr>
<td>$y_i$</td>
<td>$c_{i+1}$</td>
<td>$0.37 + 0.04L$</td>
</tr>
</tbody>
</table>

$L$: load on the gate output
CARRY-LOOKAHEAD ADDER IMPLEMENTATION

- FASTER IMPLEMENTATION

- ADDITION AS A TWO-STEP PROCESS:

  1. DETERMINE THE VALUES OF ALL THE CARRIES

  2. SIMULTANEOUSLY COMPUTE ALL THE RESULT BITS
Figure 10.4: CARRY-LOOKAHEAD ADDER MODULE.
SWITCHING EXPRESSIONS

- **INTERMEDIATE CARRIES:**

\[ c_{i+1} = g_i + p_i \cdot c_i \]

**BY SUBSTITUTION,**

\[ c_1 = g_0 + p_0 c_0 \]
\[ c_2 = g_1 + p_1 c_1 \]
\[ = g_1 + p_1 g_0 + p_1 p_0 c_0 \]
\[ c_3 = g_2 + p_2 c_2 \]
\[ = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \]
\[ c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \]
Figure 10.5: CARRY-LOOKAHEAD ADDER: a) 4-BIT CARRY-LOOKAHEAD GENERATOR WITH \( P \) and \( G \) OUTPUTS (CLG-4).
Figure 10.5: CARRY-LOOKAHEAD ADDER: b) 4-BIT MODULE (CLA-4); (CLG-4).

\[
\begin{align*}
  t_p(x_0 \rightarrow c_4) &= t_{pg} + t_{CLG-4} \\
  t_p(c_0 \rightarrow c_4) &= t_{CLG-4} \\
  t_p(x_0 \rightarrow P, G') &= t_{pg} + t_{CLG-4} \\
  t_p(x_0 \rightarrow z_3) &= t_{pg} + t_{CLG-4} + t_{XOR}
\end{align*}
\]
\[ P = 1: \ c_{\text{in}} \ \text{PROPAGATED BY THE MODULE} \]

\[ G = 1: \ c_{\text{out}} = 1 \ \text{GENERATED BY THE MODULE,} \]
\[ \text{IRRESPECTIVE OF} \ c_{\text{in}} \]

\[ P = \begin{cases} 
1 & \text{if } \ x + y = 2^4 - 1 \\
0 & \text{otherwise}
\end{cases} \]

\[ G = \begin{cases} 
1 & \text{if } \ x + y \geq 2^4 \\
0 & \text{otherwise}
\end{cases} \]

\[ c_{\text{out}} = G + P \cdot c_{\text{in}} \]

\[ P = p_3p_2p_1p_0 \]

\[ G = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 \]
ITERATIVE (CARRY-RIPPLE) ADDER NETWORK

\[
\begin{align*}
\mathbf{x} &= (x^{(3)}, x^{(2)}, x^{(1)}, x^{(0)}) \\
x^{(3)} &= (x_{15}, x_{14}, x_{13}, x_{12}) \\
x^{(2)} &= (x_{11}, x_{10}, x_{9}, x_{8}) \\
x^{(1)} &= (x_{7}, x_{6}, x_{5}, x_{4}) \\
x^{(0)} &= (x_{3}, x_{2}, x_{1}, x_{0})
\end{align*}
\]

where

\[
x = 2^{12}x^{(3)} + 2^8x^{(2)} + 2^4x^{(1)} + x^{(0)}
\]
Figure 10.6: 16-BIT CARRY-RIPPLE ADDER NETWORK USING 4-BIT ADDER MODULES.
Figure 10.7: 32-BIT CARRY-LOOKAHEAD ADDER USING CLA-4 AND CLG-4 MODULES.

- PROPAGATION DELAY:

\[ t_p(\text{net}) = t_{PG} + 2t_{CLG-4} + t_{ADD} \]
CLA ADDER (cont.)

\[ c_4 = G_0 + P_0 c_0 \]
\[ c_8 = G_1 + P_1 G_0 + P_1 P_0 c_0 \]
\[ c_{12} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0 \]
\[ c_{16} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0 \]

\[ P_0 = p_3 \cdot p_2 \cdot p_1 \cdot p_0 \]
\[ G_0 = g_3 + g_2 p_3 + g_1 p_3 p_2 + g_0 p_3 p_2 p_1 \]
TWO COMMON REPRESENTATIONS:
- SIGN-AND-MAGNITUDE (SM)
- TRUE-AND-COMPLEMENT (TC)
SIGN-AND-MAGNITUDE (SM) SYSTEM

- $x$ REPRESENTED BY PAIR $(x_s, x_m)$
  
  **sign:**
  
  $x_s = \begin{cases} 
  0 & \text{if } x \geq 0 \\
  1 & \text{if } x \leq 0 
  \end{cases}$

  **magnitude:**
  
  $x_m$

- RANGE OF SIGNED INTEGERS

  - total number of bits: $n$
  - sign: 1
  - magnitude: $n - 1$

  $$-(2^{n-1} - 1) \leq x \leq 2^{n-1} - 1$$

- TWO REPRESENTATIONS OF ZERO:

  - $x_s = 0, x_m = 0$ (positive zero)
  - $x_s = 1, x_m = 0$ (negative zero)
TWO’S-COMPLEMENT SYSTEM

- NO SEPARATION BETWEEN THE REPRESENTATION OF SIGN AND REPRESENTATION OF MAGNITUDE
- SIGNED INTEGER $x$ REPRESENTED BY *POSITIVE* INTEGER $x_R$
- MAP 2: BINARY REPRESENTATION OF $x_R$

$$x_R = \sum_{i=0}^{n-1} x_i 2^i, \quad 0 \leq x_R \leq 2^n - 1$$

- MAP 1: TWO’S COMPLEMENT

$$x_R = x \mod 2^n$$

BY DEFINITION OF $\mod$, FOR $|x| < 2^n$: equivalent to

$$x_R = \begin{cases} x & \text{if } x \geq 0 \\ 2^n - |x| & \text{if } x < 0 \end{cases}$$

FOR UNAMBIGUOUS SYMMETRICAL REPRESENTATION

$$|x|_{max} \leq 2^{n-1} - 1$$

$x$ | -4  -3  -2  -1  0  1  2  3  
---|---|---|---|---|---|---|---|---|
$x_R$ |  4  5  6  7  0  1  2  3
Figure 10.8: SIGNED INTEGER REPRESENTED BY POSITIVE INTEGER.
Figure 10.9: TWO’S COMPLEMENT REPRESENTATION FOR $n = 4$. 
### MAPPING IN TWO’S-COMPLEMENT SYSTEM

<table>
<thead>
<tr>
<th>( x )</th>
<th>( x_R )</th>
<th>( x )</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00...000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>00...001</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00...010</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>( 2^{n-1} - 1 )</td>
<td>( 2^{n-1} - 1 )</td>
<td>01...111</td>
<td></td>
</tr>
<tr>
<td>( -2^{n-1} )</td>
<td>( 2^{n-1} )</td>
<td>10...000</td>
<td></td>
</tr>
<tr>
<td>( -(2^{n-1} - 1) )</td>
<td>( 2^{n-1} + 1 )</td>
<td>10...001</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>( -2 )</td>
<td>( 2^n - 2 )</td>
<td>11...110</td>
<td></td>
</tr>
<tr>
<td>(-1)</td>
<td>( 2^n - 1 )</td>
<td>11...111</td>
<td></td>
</tr>
</tbody>
</table>

- True forms (positive): \( x_R = x \)
- Complement forms (negative): \( x_R = 2^n - |x| \)
### EXAMPLE 10.2: MAPPINGS FOR $-4 \leq x \leq 3$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$x_R$</th>
<th>$\overline{x}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>-1</td>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>-2</td>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>-3</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>-4</td>
<td>4</td>
<td>100</td>
</tr>
</tbody>
</table>
**CONVERSE MAPPING**

\[
x = \begin{cases} 
  x_R & \text{if } x_R \leq 2^{n-1} - 1 \ (x \geq 0) \\
  x_R - 2^n & \text{if } x_R \geq 2^{n-1} \ (x < 0)
\end{cases}
\]

IN TERMS OF BIT VECTOR \((x_{n-1}, x_{n-2}, \ldots, x_1, x_0)\)

i) For \(x_R < 2^{n-1}\), bit \(x_{n-1}\) is 0 and \(x \geq 0\).

\[
x = x_R = 0 \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i
\]

ii) For \(x_R \geq 2^{n-1}\) bit \(x_{n-1}\) is 1 and \(x < 0\).

\[
x = x_R - 2^n = (1 \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i) - 2^n = -1 \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i
\]

**COMBINING BOTH CASES**

\[
x = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i
\]
\[ x = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i2^i \]

8-BIT EXAMPLES:

<table>
<thead>
<tr>
<th>( x )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>01000101</td>
<td>0 + 69 = 69</td>
</tr>
<tr>
<td>11000101</td>
<td>-128 + 69 = -58</td>
</tr>
</tbody>
</table>

SIGN DETECTION:

\[ x \geq 0 \text{ if } x_{n-1} = 0 \]
\[ x < 0 \text{ if } x_{n-1} = 1 \]
ONES’-COMPLEMENT SYSTEM

\[ x_R = x \mod C \]

ONES’-COMPLEMENT SYSTEM: \( C = 2^n - 1 \)

- the ones’-complement system symmetrical, with the range \(- (2^n - 1) \leq x \leq 2^n - 1\);

- two representations for zero, namely \( x_R = 0 \) and \( x_R = 2^n - 1 \);

- the sign also detected by the most-significant bit

\[ x \geq 0 \quad \text{if} \quad (x_{n-1} = 0) \quad \text{or} \quad (x_R = 2^n - 1) \]
## MAPPING IN ONES’-COMPLEMENT SYSTEM

<table>
<thead>
<tr>
<th>$x$</th>
<th>$x_R$</th>
<th>$x$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00...000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>00...001</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00...010</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>True forms</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(positive)</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>$2^{n-1} - 1$</td>
<td>$2^{n-1} - 1$</td>
<td>01...111</td>
<td>$x_R = x$</td>
</tr>
<tr>
<td>$-(2^{n-1} - 1)$</td>
<td>$2^{n-1}$</td>
<td>10...000</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Complement forms</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(negative)</td>
</tr>
<tr>
<td>$-2$</td>
<td>$2^n - 3$</td>
<td>11...101</td>
<td>$x_R = 2^n - 1 -</td>
</tr>
<tr>
<td>$-1$</td>
<td>$2^n - 2$</td>
<td>11...110</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$2^n - 1$</td>
<td>11...111</td>
<td></td>
</tr>
</tbody>
</table>

### True forms (positive)

- $x_R = x$

### Complement forms (negative)

- $x_R = 2^n - 1 - |x|$
TO GET

\[ z = x + y \]

COMPUTE

\[ z_R = (x_R + y_R) \mod 2^n \]

CORRECT IF \(-2^{n-1} \leq (x + y) \leq 2^{n-1} - 1\)

PROOF: CONSIDER

\[ (x_R + y_R) \mod 2^n \]

AND SHOW THAT IT CORRESPONDS TO \(z_R\)

BY DEFINITION OF THE REPRESENTATION,

\[ x_R = x \mod 2^n \text{ and } y_R = y \mod 2^n \]

THEREFORE,

\[ (x_R + y_R) \mod 2^n = (x \mod 2^n + y \mod 2^n) \mod 2^n = (x + y) \mod 2^n = z \mod 2^n \]

BY DEFINITION OF REPRESENTATION

\[ z \mod 2^n = z_R \]
2’s COMPLEMENT ADDITION: A SUMMARY

1. ADD $x_R$ AND $y_R$ (use adder for positive operands)

2. PERFORM THE $\text{mod}$ OPERATION

- DOES NOT DEPEND ON THE RELATIVE MAGNITUDES OF THE OPERANDS AND ON THEIR SIGNS (simpler than in $S+M$)

EXAMPLES OF ADDITION FOR $C=64$ and $-32 \leq x, y, z \leq 31$

<table>
<thead>
<tr>
<th>Signed operands</th>
<th>Representation</th>
<th>Two’s-complement addition</th>
<th>Signed result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$ $y$</td>
<td>$x_R$ $y_R$</td>
<td>$(x_R + y_R) \mod 64 = z_R$</td>
<td>$z$</td>
</tr>
<tr>
<td>13 9</td>
<td>13 9</td>
<td>22 mod 64 = 22</td>
<td>22</td>
</tr>
<tr>
<td>13 -9</td>
<td>13 55</td>
<td>68 mod 64 = 4</td>
<td>4</td>
</tr>
<tr>
<td>-13 9</td>
<td>51 9</td>
<td>60 mod 64 = 60</td>
<td>-4</td>
</tr>
<tr>
<td>-13 -9</td>
<td>51 55</td>
<td>106 mod 64 = 42</td>
<td>-22</td>
</tr>
</tbody>
</table>
THE mod OPERATION

• Let $w_R = x_R + y_R$. Then

$$x_R, y_R < 2^n \Rightarrow w_R < 2 \times 2^n$$

$$z_R = w_R \mod 2^n = \begin{cases} w_R & \text{if } w_R < 2^n \\ w_R - 2^n & \text{if } 2^n \leq w_R < 2 \times 2^n \end{cases}$$

• Since $w_R < 2 \times 2^n$

$$w = (w_n, w_{n-1}, \ldots, w_0)$$

$$w_R = \begin{cases} < 2^n & \text{if } w_n = 0 \\ \geq 2^n & \text{if } w_n = 1 \end{cases}$$

Case 1. $w_R < 2^n$. Then $w_R \mod 2^n = w_R \Leftrightarrow (w_{n-1}, \ldots, w_0)$.

Case 2. $w_R \geq 2^n$

$$w_R \mod 2^n = w_R - 2^n \Leftrightarrow (1, w_{n-1}, \ldots, w_0) - (1, 0, \ldots, 0)$$

$$= (w_{n-1}, \ldots, w_0)$$

• CONCLUSION: $w_R \mod 2^n = (w_{n-1}, \ldots, w_0)$
- \textit{mod} OPERATION PERFORMED BY DISCARDING MOST-SIGNIFICANT BIT OF \( w \)

- 2'S COMPLEMENT ADDITION:
RESULT CORRESPONDS TO OUTPUT OF ADDER, DISCARDING THE CARRY-OUT

\[
z = ADD(x, y, 0)
\]

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{two_complement_adder.png}
\caption{TWO’S-COMPLEMENT ADDER MODULE.}
\end{figure}
### EXAMPLES OF 2’S COMPLEMENT ADDITION

<table>
<thead>
<tr>
<th>( n = 4 )</th>
<th>Bit-level computation</th>
<th>Positive representation</th>
<th>Signed values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( x = 1011 )</td>
<td>( x_R = 11 )</td>
<td>( x = -5 )</td>
</tr>
<tr>
<td></td>
<td>( y = 0101 )</td>
<td>( y_R = 5 )</td>
<td>( y = 5 )</td>
</tr>
<tr>
<td></td>
<td>( w = 10000 )</td>
<td>( w_R = 16 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( z = 0000 )</td>
<td>( z_R = 0 )</td>
<td>( z = 0 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( n = 8 )</th>
<th>Bit-level computation</th>
<th>Positive representation</th>
<th>Signed values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( x = 11011010 )</td>
<td>( x_R = 218 )</td>
<td>( x = -38 )</td>
</tr>
<tr>
<td></td>
<td>( y = 11110001 )</td>
<td>( y_R = 241 )</td>
<td>( y = -15 )</td>
</tr>
<tr>
<td></td>
<td>( w = 111001011 )</td>
<td>( w_R = 459 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( z = 11001011 )</td>
<td>( z_R = 203 )</td>
<td>( z = -53 )</td>
</tr>
</tbody>
</table>
CHANGEOFSIGNINTWO’SCOMPLEMENTSYSTEM

• \( z = -x \)

\[
z_R = (2^n - x_R) \mod 2^n
\]

\( x = 0: \) since \( z = -x = 0 \) we have \( z_R = x_R = 0 \). Moreover,

\[
z_R = (2^n - 0) \mod 2^n = 0
\]

\( x > 0: \) since \( z = -x \) is negative,

\[
z_R = 2^n - |z| = 2^n - |x|
\]

Moreover, \( x \) is positive so that

\[
x_R = x
\]

Substitute: \( z_R = 2^n - x_R \).

\( x < 0: \) since \( z = -x \) is positive,

\[
z_R = z = -x
\]

Moreover, \( x \) is negative so that

\[
x_R = 2^n - |x| = 2^n + x
\]

Substitute: \( z_R = 2^n - x_R \).
CHANGE OF SIGN (cont.)

• DIRECT SUBTRACTION $2^n - x_R$ COMPLEX

EXAMPLE:

\[
\begin{array}{c|c}
2^8 & 100000000 \\
\hline
x_R & 01011110 \\
\hline
& 10100010
\end{array}
\]

• INSTEAD, USE $2^n = (2^n - 1) + 1$

\[
z_R = (2^n - 1 - x_R) + 1
\]

• THE COMPLEMENT WITH RESPECT TO $2^n - 1$: COMPLEMENT EACH BIT OF $x$

\[
x_R = 17 \quad 010001 \\
63 - x_R \quad 111111 - 010001 \quad 101110
\]
CHANGE-OF-SIGN OPERATION

TWO-STEP OPERATION:

1. COMPLEMENT EACH BIT OF $x$ denoted $x'$.
2. ADD 1 (set carry-in $c_0 = 1$)

• DESCRIPTION:

$$z = ADD(x', 0, 1)$$

EXAMPLE FOR $n = 4$, $x = -3$:

<table>
<thead>
<tr>
<th>$x$</th>
<th>1101 $x = -3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x'$</td>
<td>0010</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>$c_0$</td>
<td>1</td>
</tr>
<tr>
<td>$z$</td>
<td>0011 $z = 3$</td>
</tr>
</tbody>
</table>
SUBTRACTION IN TWO’S COMPLEMENT SYSTEM

- \( z = x - y = x + (-y) \)
  
  \[
  z_R = (x_R + (2^n - 1 - y_R) + 1) \mod 2^n
  \]

- **THE CORRESPONDING DESCRIPTION**
  
  \[
  z = ADD_R(x, y', 1)
  \]

**EXAMPLE:**

<table>
<thead>
<tr>
<th>( x )</th>
<th>01100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
<td>00110001</td>
</tr>
<tr>
<td>( y' )</td>
<td>11001110</td>
</tr>
<tr>
<td>( z )</td>
<td>00101111</td>
</tr>
</tbody>
</table>

**SUMMARY OF 2’S COMPLEMENT OPERATIONS**

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>2’s COMPLEMENT SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z = x + y )</td>
<td>( z = ADD(x, y, 0) )</td>
</tr>
<tr>
<td>( z = -x )</td>
<td>( z = ADD(x', 0, 1) )</td>
</tr>
<tr>
<td>( z = x - y )</td>
<td>( z = ADD(x, y', 1) )</td>
</tr>
</tbody>
</table>
OVERFLOW DETECTION IN ADDITION

- OVERFLOW – result exceeds most positive or negative representable integer

$$-2^{n-1} \leq z \leq 2^{n-1} - 1$$

- BOTH OPERANDS SAME SIGN, RESULT OPPOSITE SIGN

$$v = x'_{n-1}y'_{n-1}z_{n-1} + x_{n-1}y_{n-1}z'_{n-1}$$

Figure 10.11: OVERFLOW IN TWO’S-COMPLEMENT SYSTEM.
TWO’S COMPLEMENT ARITHMETIC UNIT

INPUTS: \[ x = (x_{n-1}, \ldots, x_0), \quad x_j \in \{0, 1\} \]
\[ y = (y_{n-1}, \ldots, y_0), \quad y_j \in \{0, 1\} \]
\[ c_{in} \in \{0, 1\} \]
\[ F = (f_2, f_1, f_0) \]

OUTPUTS: \[ z = (z_{n-1}, \ldots, z_0), \quad z_j \in \{0, 1\} \]
\[ c_{out}, sgn, zero, ovf \in \{0, 1\} \]

FUNCTIONS:

<table>
<thead>
<tr>
<th>( F )</th>
<th>Operation</th>
<th>( z = x + y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>ADD</td>
<td>add</td>
</tr>
<tr>
<td>011</td>
<td>SUB</td>
<td>subtract</td>
</tr>
<tr>
<td>101</td>
<td>ADDC</td>
<td>add with carry</td>
</tr>
<tr>
<td>110</td>
<td>CS</td>
<td>change sign</td>
</tr>
<tr>
<td>010</td>
<td>INC</td>
<td>increment</td>
</tr>
</tbody>
</table>

\[ sgn = 1 \quad \text{if} \quad z < 0, \quad 0 \quad \text{otherwise} \quad \text{(the sign)} \]
\[ zero = 1 \quad \text{if} \quad z = 0, \quad 0 \quad \text{otherwise} \]
\[ ovf = 1 \quad \text{if} \quad z \text{ overflows,} \quad 0 \quad \text{otherwise} \]
Figure 10.12: IMPLEMENTATION OF TWO’S-COMPLEMENT ARITHMETIC UNIT.
CONTROL OF TWO’S-COMPLEMENT ARITHMETIC OPERATIONS

- OPERATION IDENTIFIED BY BIT-VECTOR $F = (f_2, f_1, f_0)$
- COMPLEMENT OPERATION $a = \text{COMPL}(b, K)$:
  
  $a_i = \begin{cases} 
  b_i & \text{if } K = 0 \\
  b'_i & \text{if } K = 1 
  \end{cases}$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Op-code</th>
<th>Control Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>001</td>
<td>$K_x, K_y, K_{MX}$</td>
</tr>
<tr>
<td>SUB</td>
<td>011</td>
<td>$K_x, K_y, K_{MX}$</td>
</tr>
<tr>
<td>ADDC</td>
<td>101</td>
<td>$K_x, K_y, K_{MX}$</td>
</tr>
<tr>
<td>CS</td>
<td>110</td>
<td>$K_x, K_y, K_{MX}$</td>
</tr>
<tr>
<td>INC</td>
<td>010</td>
<td>$K_x, K_y, K_{MX}$</td>
</tr>
</tbody>
</table>

- CONTROL SIGNALS:

  \[
  K_x = f_2 f_1 \\
  K_y = f_1 \\
  K_{MX} = f_0 \\
  c_0 = f_1 + f_2 f_0 c_{in}
  \]
• **ARITHMETIC-LOGIC UNIT**
  module realizing set of arithmetic and logic functions

• Why build ALUs?
  1. Use in many different applications
  2. ALU modules used in processors: function selected by control unit
## TYPICAL EXAMPLE OF ALU

<table>
<thead>
<tr>
<th>Control ((S'))</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERO</td>
<td>(z = 0)</td>
</tr>
<tr>
<td>ADD</td>
<td>(z = (x + y + c_{in}) \mod 16)</td>
</tr>
<tr>
<td>SUB</td>
<td>(z = (x + y' + c_{in}) \mod 16)</td>
</tr>
<tr>
<td>EXSUB</td>
<td>(z = (x' + y + c_{in}) \mod 16)</td>
</tr>
<tr>
<td>AND</td>
<td>(z = x \cdot y)</td>
</tr>
<tr>
<td>OR</td>
<td>(z = x + y)</td>
</tr>
<tr>
<td>XOR</td>
<td>(z = x \oplus y)</td>
</tr>
<tr>
<td>ONE</td>
<td>(z = 1111)</td>
</tr>
</tbody>
</table>

\(a'\) denotes the integer represented by vector \(a'\)

\(\cdot, +, \text{ and } \oplus\) are applied to the corresponding bits
Figure 10.13: 4-bit ALU.
• MODULE HAS NO CARRY-OUT SIGNAL
  – cannot be used directly in an iterative (carry-ripple) network
  – carry-out signal implemented as

\[ c_{out} = G + P \cdot c_{in} \]

– carry-skip network
16-bit ALU

Figure 10.14: 16-bit ALU.
• **HIGH-LEVEL DESCRIPTION OF AN \( n \)-BIT COMPARATOR:**

**INPUTS:** \( \mathbf{x} = (x_{n-1}, \ldots, x_0), \ x_j \in \{0, 1\} \)
\( \mathbf{y} = (y_{n-1}, \ldots, y_0), \ y_j \in \{0, 1\} \)
\( c_{\text{in}} \in \{G,E,S\} \)

**OUTPUT:** \( z \in \{G,E,S\} \)

**FUNCTION:**
\[
  z = \begin{cases} 
    G & \text{if } (x > y) \text{ or } (x = y \text{ and } c_{\text{in}} = G) \\
    E & \text{if } (x = y) \text{ and } (c_{\text{in}} = E) \\
    S & \text{if } (x < y) \text{ or } (x = y \text{ and } c_{\text{in}} = S) 
  \end{cases}
\]

\( x \) and \( y \) – the integers represented \( \mathbf{x} \) and \( \mathbf{y} \)

• **IMPLEMENTATION OF 4-bit COMPARATOR MODULE**

\[
  \mathbf{c_{\text{in}}} = (c_{\text{in}}^G, c_{\text{in}}^E, c_{\text{in}}^S), \quad c_{\text{in}}^G, c_{\text{in}}^E, c_{\text{in}}^S \in \{0, 1\}
\]
\[
  \mathbf{z} = (z^G, z^E, z^S), \quad z^G, z^E, z^S \in \{0, 1\}
\]
Figure 10.15: 4-BIT COMPARATOR MODULE: a) block diagram; b) gate-network implementation.
\[ S_i = x'_i y_i \]
\[ E_i = (x_i \oplus y_i)', \quad i = 0, \ldots, 3 \]
\[ G_i = x_i y'_i \]

\[ z^G = G_3 + E_3 G_2 + E_3 E_2 G_1 + E_3 E_2 E_1 G_0 + E_3 E_2 E_1 E_0 c_{in}^G \]
\[ z^E = E_3 E_2 E_1 E_0 c_{in}^E \]
\[ z^S = S_3 + E_3 S_2 + E_3 E_2 S_1 + E_3 E_2 E_1 S_0 + E_3 E_2 E_1 E_0 c_{in}^S \]
Figure 10.16: 16-BIT ITERATIVE COMPARATOR NETWORK.
Figure 10.17: 16-BIT TREE COMPARATOR NETWORK.
$z^G = \begin{cases} 1 & \text{if } g > s \\ 0 & \text{otherwise} \end{cases}$

$z^E = \begin{cases} 1 & \text{if } g = s \\ 0 & \text{otherwise} \end{cases}$

$z^S = \begin{cases} 1 & \text{if } g < s \\ 0 & \text{otherwise} \end{cases}$

- $g$ and $s$ are the integers represented by the vectors $g$ and $s$, respectively.
• $n \times m$ bits multiplier:
  
  $0 \leq x \leq 2^n - 1$ (the multiplicand)
  
  $0 \leq y \leq 2^m - 1$ (the multiplier),
  
  $0 \leq z \leq (2^n - 1)(2^m - 1)$ (the product).

• The high-level function:

  \[
  z = x \times y
  \]

  \[
  z = x \left( \sum_{i=0}^{m-1} y_i 2^i \right) = \sum_{i=0}^{m-1} x y_i 2^i
  \]

  Since $y_i$ is either 0 or 1, we get

  \[
  xy_i = \begin{cases} 
  0 & \text{if } y_i = 0 \\
  x & \text{if } y_i = 1
  \end{cases}
  \]
MULTIPLICATION BIT MATRIX

\[
\begin{align*}
    &x_7 y_0 & x_6 y_0 & x_5 y_0 & x_4 y_0 & x_3 y_0 & x_2 y_0 & x_1 y_0 & x_0 y_0 \\
    &x_7 y_1 & x_6 y_1 & x_5 y_1 & x_4 y_1 & x_3 y_1 & x_2 y_1 & x_1 y_1 & x_0 y_1 \\
    &x_7 y_2 & x_6 y_2 & x_5 y_2 & x_4 y_2 & x_3 y_2 & x_2 y_2 & x_1 y_2 & x_0 y_2 \\
    &x_7 y_3 & x_6 y_3 & x_5 y_3 & x_4 y_3 & x_3 y_3 & x_2 y_3 & x_1 y_3 & x_0 y_3 \\
    &x_7 y_4 & x_6 y_4 & x_5 y_4 & x_4 y_4 & x_3 y_4 & x_2 y_4 & x_1 y_4 & x_0 y_4 \\
    &x_7 y_5 & x_6 y_5 & x_5 y_5 & x_4 y_5 & x_3 y_5 & x_2 y_5 & x_1 y_5 & x_0 y_5
\end{align*}
\]

Multiplier implementation:

- \( m \) arrays of \( n \) AND gates

- \( m - 1 \) n-bit adders
Figure 10.18: IMPLEMENTATION OF AN $8 \times 6$ MULTIPLIER: a) PRIMITIVE MODULES; b) NETWORK.
MULTIPLIER DELAY

- delay of the buffer which connects signal $y_0$ to the $n$ AND gates
- delay of the AND gate
- delay of the adders

$$t_{adders} = t_c(n - 1) + t_s + (t_c + t_s)(m - 2)$$

If $t_s = t_c$, we get

$$t_{adders} = (n + 2(m - 2))t_s = (n + 2m - 4)t_s$$

FOR THE $8 \times 6$ CASE: $t_{adders} = (8 + 12 - 4)t_s = 16t_s$
EXAMPLE OF NETWORKS WITH STANDARD ARITHMETIC MODULES

Inputs: \( a[3], a[2], a[1], a[0], b[3], b[2], b[1], b[0] \in \{0, ..., 2^{16} - 1\} \)
\( e = (e_3, e_2, e_1, e_0) \), \( e_i \in \{0, 1\} \)

Outputs: \( c[3], c[2], c[1], c[0] \in \{0, ..., 2^{17} - 1\} \)
\( d \in \{0, 1, 2, 3\} \)
\( f \in \{0, 1\} \)

Function:

\[
f = \begin{cases} 
1 & \text{if at least one } e_j = 1 \\
0 & \text{otherwise}
\end{cases}, \quad j = 0, 1, 2, 3
\]

\[
d = \begin{cases} 
i & \text{if } e_i \text{ is the highest priority event} \\
0 & \text{if no event occurred}
\end{cases}
\]

\[
c[i] = \begin{cases} 
a[i] + b[i] & \text{if } e_i \text{ is the highest priority event} \\
0 & \text{otherwise}
\end{cases}
\]
MODULAR IMPLEMENTATION

CONSISTS OF

- a PRIORITY ENCODER to determine the highest-priority event;

- an ADDER;

- two SELECTORS (multiplexers) to select the corresponding inputs to the adder;

- a DISTRIBUTOR (demultiplexer) to send the output of the adder to the corresponding system output; and

- an OR gate to determine whether at least one event has occurred.
Figure 10.19: NETWORK IN EXAMPLE 10.5