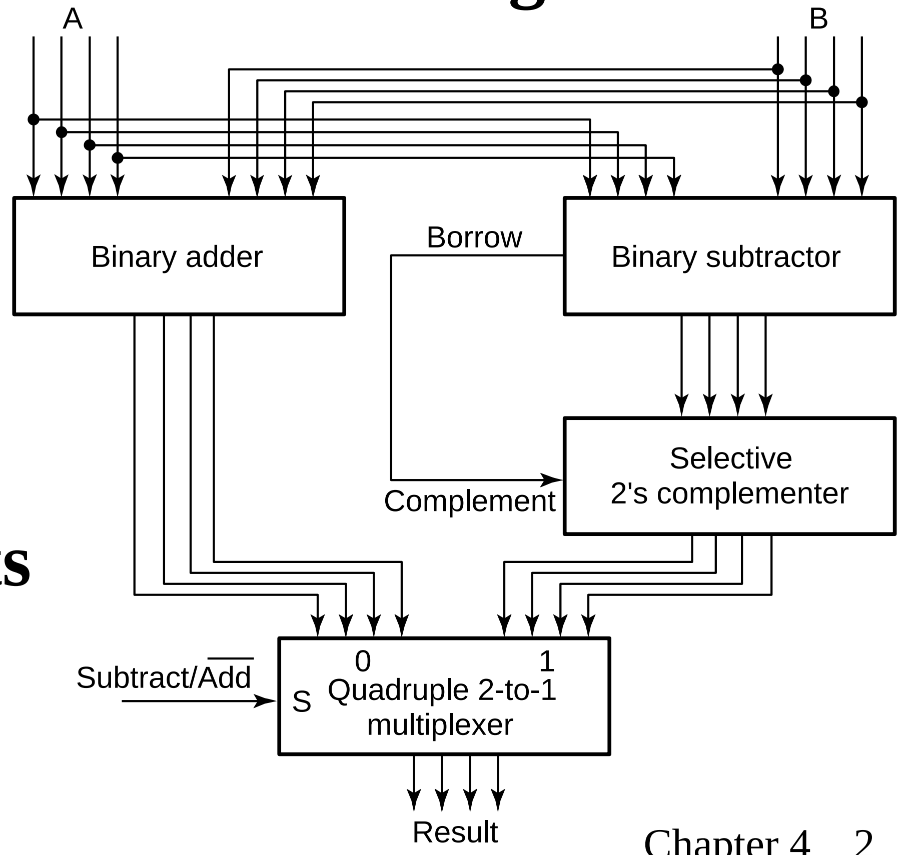
Chapter 3: part 3 Binary Subtraction

- Iterative combinational circuits
- Binary adders
  - Half and full adders
  - Ripple carry and carry lookahead adders
- Binary subtraction
- Binary adder-subtractors
  - Signed binary numbers
  - Signed binary addition and subtraction
  - Overflow
- Other Arithmetic Functions

Unsigned Subtraction (continued)

- The subtraction, 2^n – N, is taking the 2's complement of N
- To do both unsigned addition and unsigned subtraction requires:
- Quite complex!
- Goal: Shared simpler logic for both addition and subtraction
- Introduce complements as an approach


A
B
Binary adder
Borrow
Binary subtractor
Selective 2's complementer
Complement
Subtract/$\overline{\text{Add}}$
S
0
1
Quadruple 2-to-1 multiplexer
Result

Complements

- Two complements:
  - Diminished Radix Complement of N
    - (r – 1)’s complement for radix r
    - 1’s complement for radix 2
    - Defined as $(r^n - 1) - N$
  - Radix Complement
    - r’s complement for radix r
    - 2’s complement in binary
    - Defined as $r^n - N$
- Subtraction is done by adding the complement of the subtrahend
- If the result is negative, takes its 2’s complement

Binary 1's Complement

- For $r = 2$, $N = 01110011_2$, $n = 8$ (8 digits):

  $(r^n - 1) = 256 - 1 = 255_{10}$ or 11111111_2
- The 1's complement of 01110011_2 is then:

$$\begin{array}{r} 11111111 \\ -\ \underline{01110011} \\ 10001100 \end{array}$$

- Since the $2^n - 1$ factor consists of all 1's and since $1 - 0 = 1$ and $1 - 1 = 0$, the one's complement is obtained by complementing each individual bit (bitwise NOT).

Binary 2's Complement

- For *r* = 2, *N* = 01110011_2, *n* = 8 (8 digits), we have:

  $(r^n) = 256_{10}$ or 100000000_2

- The 2's complement of 01110011 is then:

$$\begin{array}{r} 100000000 \\ -\ \underline{01110011} \\ 10001101 \end{array}$$

- Note the result is the 1's complement plus 1, a fact that can be used in designing hardware

Subtraction with 2's Complement

- **For n-digit, <u>unsigned</u> numbers M and N, find M – N in base 2:**
  - **Add the 2's complement of the subtrahend N to the minuend M:**
    $$M + (2^n - N) = M - N + 2^n$$
  - **If $M \geq N$, the sum produces end carry r^n which is discarded; from above, M – N remains.**
  - **If M < N, the sum does not produce an end carry and, from above, is equal to 2^n – (N – M), the 2's complement of (N – M).**
  - **To obtain the result – (N – M) , take the 2's complement of the sum and place a – to its left.**

Important Observation

- The complement of the complement restores the number to its original value.
- This is true for both 1’s complement and 2’s complement.

Unsigned 2's Complement Subtraction Example 1

- Find $01010100_2 - 01000011_2$

$$\begin{array}{r} 01010100 \\ -\ \underline{01000011} \end{array} \xrightarrow{\text{2's comp}} \begin{array}{r} {}^{1}01010100 \\ +\ \underline{10111101} \\ 00010001 \end{array}$$

- The carry of 1 indicates that no correction of the result is required.

Unsigned 2's Complement Subtraction Example 2

- Find $01000011_2 – 01010100_2$

$$\begin{array}{rr} & 01000011 \\ - & \underline{01010100} \end{array} \xrightarrow{\text{2's comp}} \begin{array}{rr} & {}^{0}01000011 \\ + & \underline{10101100} \\ & 11101111 \\ & 00010001 \end{array}$$

$$11101111 \xrightarrow{\text{2's comp}} 00010001$$

- The carry of 0 indicates that a correction of the result is required.
- Result = – (00010001)

Signed Integers

- Positive numbers and zero can be represented by unsigned n-digit, radix *r* numbers. We need a representation for negative numbers.
- To represent a sign (+ or –) we need exactly one more bit of information (1 binary digit gives $2^1 = 2$ elements which is exactly what is needed).
- Since computers use binary numbers, by convention, the most significant bit is interpreted as a sign bit:

$$s\ a_{n-2} \ldots a_2 a_1 a_0$$

where:

s = 0 for Positive numbers

s = 1 for Negative numbers

and a_i = 0 or 1 represent the magnitude in some form.

Signed Integer Representations

- ***Signed-Magnitude*** **– the number consists of a magnitude and a symbol (0 for +, 1 for -).**
- ***Signed-Complement*** **– a negative number is represented by its complement.   There are two possibilities here:**
  - ***Signed 1's Complement***
    - **Uses 1's Complement Arithmetic**
  - ***Signed 2's Complement***
    - **Uses 2's Complement Arithmetic**

Signed Integer Representation Example

- **r =2, n=3**

Number	Sign -Mag.	1's Comp.	2's Comp.
+3	011	011	011
+2	010	010	010
+1	001	001	001
+0	000	000	000
– 0	100	111	—
– 1	101	110	111
– 2	110	101	110
– 3	111	100	101
– 4	—	—	100

Corresponding positive number?

- The two's complement of a negative number is the corresponding positive value. For example, inverting the bits of –5 (1111 1011) gives:
  - 0000 0100
- And adding one gives the final value:
  - 0000 0101 (5)

Signed-Complement Arithmetic

- **Addition:**

  1. Add the numbers including the sign bits, discarding a carry out of the sign bits (2's Complement)

  2. If the sign bits were the same for both numbers and the sign of the result is different, an overflow has occurred.

  3. The sign of the result is computed in step 1.

- **Subtraction:**

  Form the 2's complement of the number you are subtracting and follow the rules for addition.

Signed 2's Complement Examples

- **Example 1:** $\begin{array}{r} \mathbf{1101} \\ \mathbf{+\underline{0011}} \end{array}$

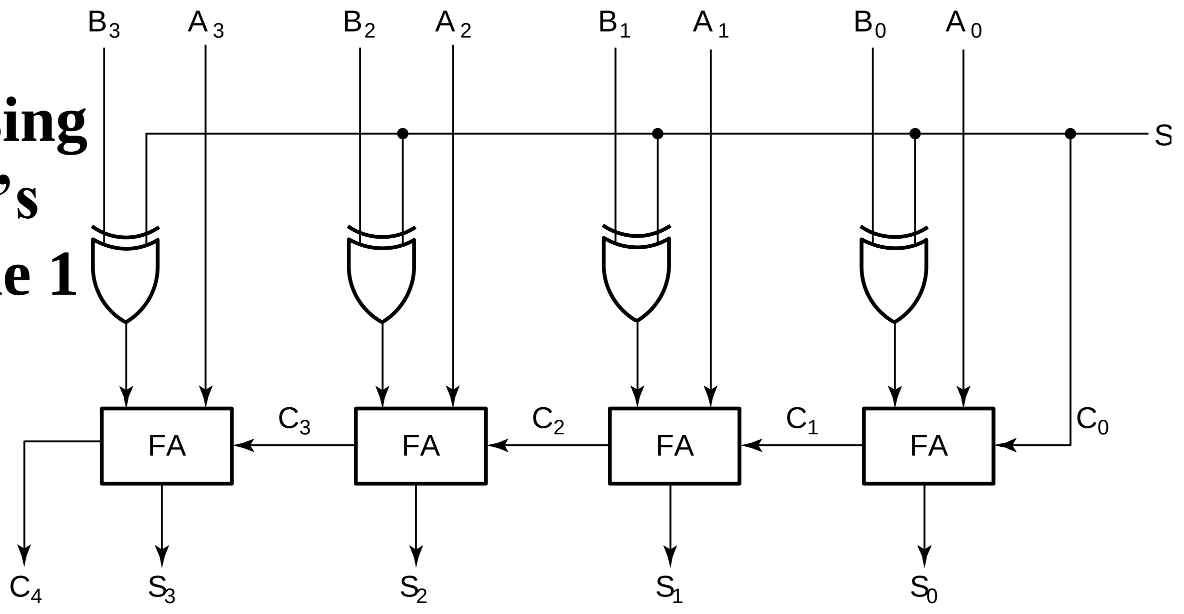
Result is 0000. The carry out of the MSB is discarded.

- **Example 2:** $\begin{array}{r} \mathbf{1101} \\ \mathbf{-\underline{0011}} \end{array}$

Result is 1010. The carry out of the MSB is discarded.

2’s Complement Adder/Subtractor

- Subtraction can be done by addition of the 2's Complement.
  1. Complement each bit (1's Complement.)
  2. Add 1 to the result.
- The circuit shown computes A + B and A – B:
- For S = 1, subtract, the 2’s complement of B is formed by using XORs to form the 1’s comp and adding the 1 applied to C_0.
- For S = 0, add, B is passed through unchanged



Overflow Detection

- *Overflow* occurs if *n* + 1 bits are required to contain the result from an n-bit addition or subtraction
- Overflow can occur for:
  - Addition of two operands with the same sign
  - Subtraction of operands with different signs
- Signed number overflow cases with correct result sign

$$\begin{array}{rrrr} 0 & 0 & 1 & 1 \\ +\,\underline{0} & -\,\underline{1} & -\,\underline{0} & +\,\underline{1} \\ 0 & 0 & 1 & 1 \end{array}$$

- Detection can be performed by examining the result signs which should match the signs of the top operand

Overflow Detection

- **Signed number cases with carries C_n and C_{n-1} shown for correct result signs:**

$$\begin{array}{cccccccc} 0 & 0\;0 & 0\;1 & 1\;1 & 1 \end{array}$$

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ +\,\underline{0} & -\,\underline{1} & -\,\underline{0} & +\,\underline{1} \\ 0 & 0 & 1 & 1 \end{array}$$

- **Signed number cases with carries shown for erroneous result signs (indicating overflow):**

$$\begin{array}{cccccccc} 0 & 1\;0 & 1\;1 & 0\;1 & 0 \end{array}$$

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ +\,\underline{0} & -\,\underline{1} & -\underline{0} & +\,\underline{1} \\ 1 & 1 & 0 & 0 \end{array}$$

- **Simplest way to implement overflow $V = C_n \oplus C_{n-1}$**

Overflow Example (1)

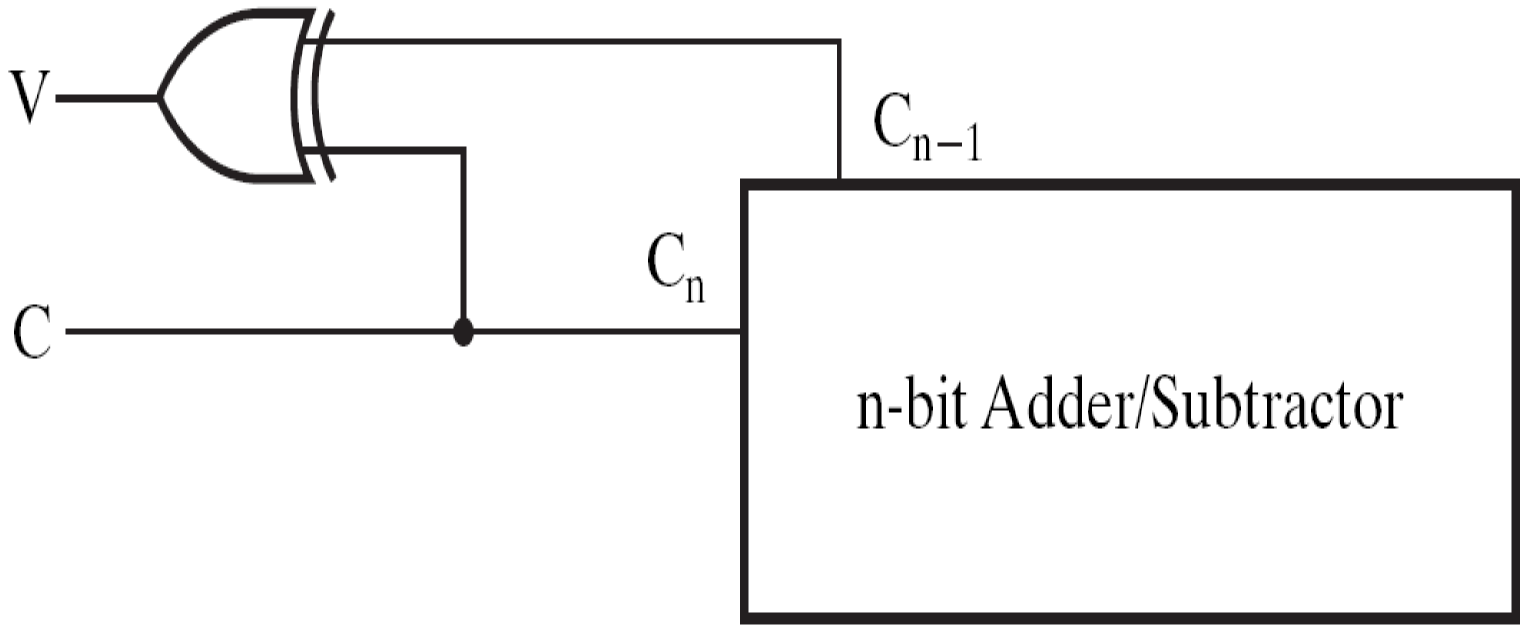
Carries:	0	1
+70	0	1000110
+80	0	1010000
+150	1	0010110

Overflow Example (2)

Carries:	1	0
-70	1	0111010
-80	1	0110000
-150	0	1101010

A simple logic that provides overflow detection is shown in Figure 4-8 (see next slide)

Figure 4-8



Class Exercises (1)

- **Perform the indicated subtraction with the following unsigned binary numbers by taking the 2s complement of the subtrahend: (please see slide 6)**
- **a) 11010 – 10001**

11010 + 01111 = 01001

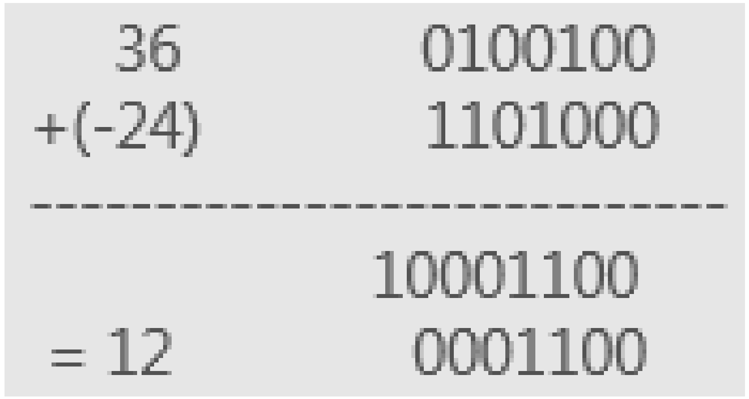
- **b) 11110 - 1110**

11110 + 10010 = 10000

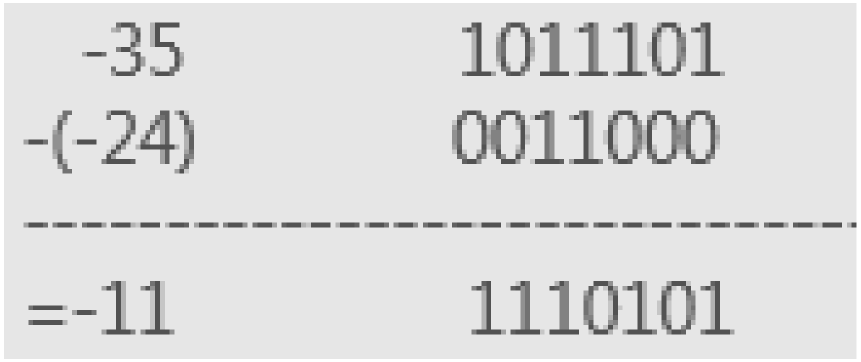
Class Exercises (2)

- **Perform the arithmetic operations (+36) + (-24) and (-35) – (-24) in binary using signed 2s complement representation for negative numbers. (see slide 14)**

+36=0100100; -24=1101000; -35=1011101



```
   36            0100100
+(-24)           1101000
-----------------------------
                10001100
 = 12            0001100
```



```
  -35            1011101
-(-24)           0011000
-----------------------------
=-11             1110101
```

Other Arithmetic Functions

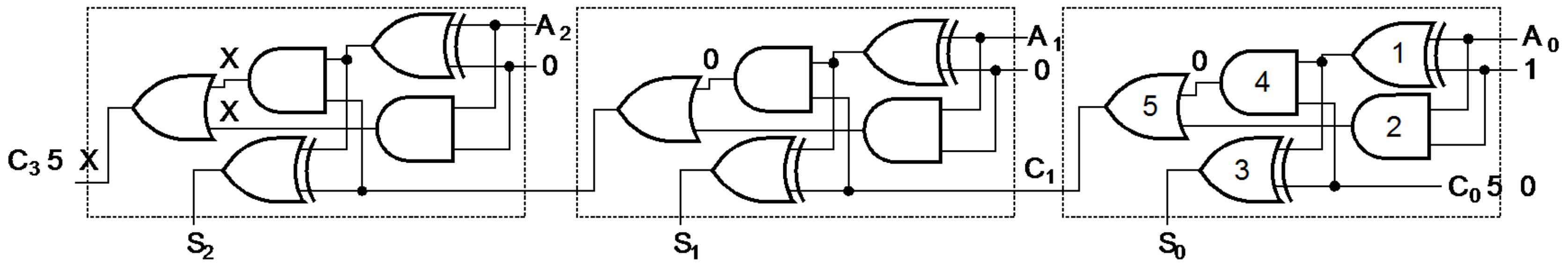
- Convenient to design the functional blocks by *contraction* - removal of redundancy from circuit to which input fixing has been applied
- Functions
  - Incrementing
  - Decrementing
  - Multiplication by Constant
  - Zero Fill and Extension

Design by Contraction

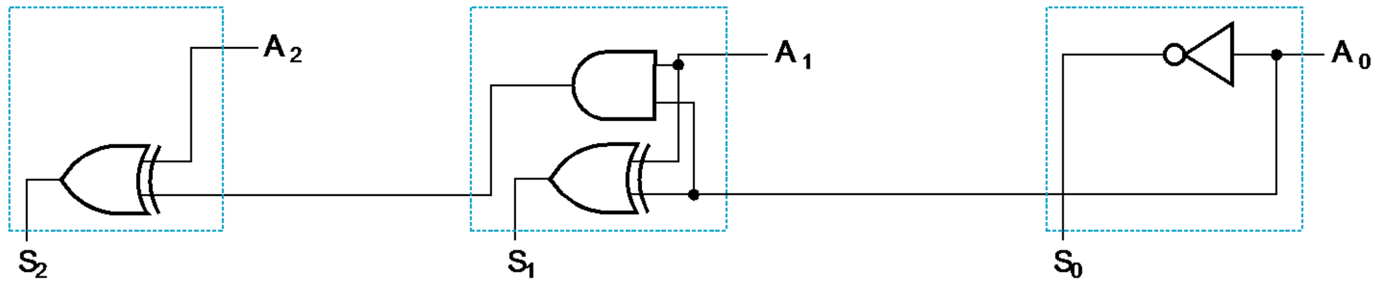
- **Contraction is a technique for simplifying the logic in a functional block to implement a different function**
  - **The new function must be realizable from the original function by applying rudimentary functions to its inputs**
  - **Contraction is treated here only for application of 0s and 1s (not for X and $\overline{X}$)**
  - **After application of 0s and 1s, equations or the logic diagram are simplified by using rules given on page 168 of the text.**

Design by Contraction Example

- **Contraction of a ripple carry adder to incrementer for *n* = 3**
  - **Set B = 001**



(a)



(b)

  - **The middle cell can be repeated to make an incrementer with *n* > 3.**

Discussion

- **Can we make the three cells identical?**

- **How?**

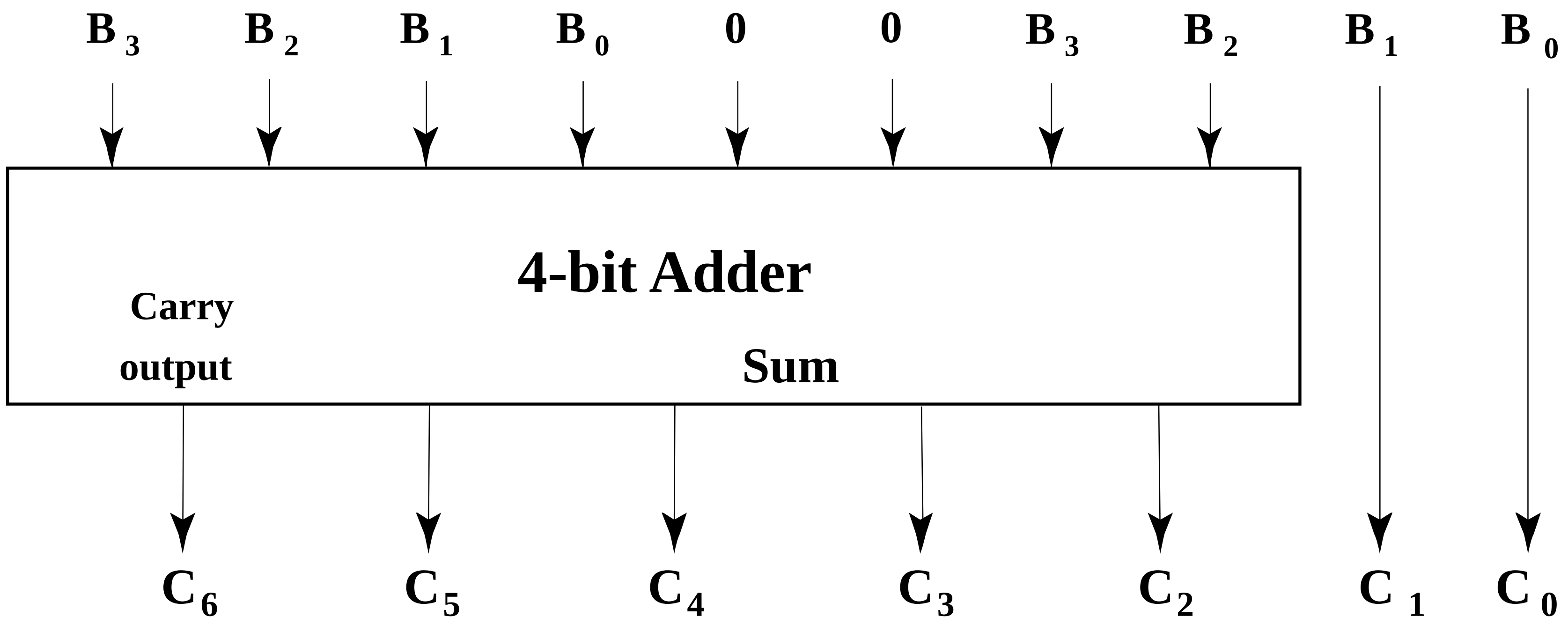
The rightmost cell in position 0 can be replaced with the cell in position 1 with B0=0 and C0=1. Likewise, the output C3 could be generated but no used. In both cases, logic cost and power efficiency are sacrificed to make all of the cells identical.

Incrementing & Decrementing

- ***Incrementing***
  - **Adding a fixed value to an arithmetic variable**
  - **Fixed value is often 1, called *counting (up*)**
  - **Examples: A + 1, B + 4**
  - **Functional block is called *incrementer***
- ***Decrementing***
  - **Subtracting a fixed value from an arithmetic variable**
  - **Fixed value is often 1, called *counting (down*)**
  - **Examples: A – 1, B – 4**
  - **Functional block is called *decrementer***

Multiplication by a Constant

- Multiplication of B(3:0) by 101
- See text Figure 4-10 (a) for contraction



Zero Fill

- ***Zero fill*** **- filling an *m*-bit operand with 0s to become an *n*-bit operand with *n* > *m***
- **Filling usually is applied to the MSB end of the operand, but can also be done on the LSB end**
- **Example: 11110101 filled to 16 bits**
  - **MSB end: 0000000011110101**
  - **LSB end: 1111010100000000**

Extension

- *Extension* - increase in the number of bits at the MSB end of an operand by using a complement representation
  - Copies the MSB of the operand into the new positions
  - Positive operand example - 01110101 extended to 16 bits:

    0000000001110101
  - Negative operand example - 11110101 extended to 16 bits:

    1111111111110101

Extension Example

- 01101011 (107 in decimal)

0000000001101011

- 10010101 (how much in 2's complement?)

1111111110010101

Weekly Exercises

- **3-50**
- **3-51**
- **3-54**