Digital Electronics COE328

Lecture 9

Dr. Shazzat Hossain

Arithmetic Overflow

- When the digits are not enough for the result.
- Example: using 4 bits to perform

$$\begin{array}{r} +7 \\ +6 \\ \hline +13 \end{array} \qquad \begin{array}{r} -7 \\ + \ -6 \\ \hline -13 \end{array} \qquad \begin{array}{r} -7 \\ +3 \\ \hline -4 \end{array} \qquad \begin{array}{r} +7 \\ -3 \\ \hline +4 \end{array}$$

Overflow detection:
Overflow must be detected in two conditions:
1: C_3=1 and C_4=0
2: C_3=0 and C_4=1

$$v = \bar{C}_3 C_4 + C_3 \bar{C}_4$$

Practice: Design a 4-bit ripple carry adder with arithmetic overflow.

$$\begin{array}{r} +7 \\ +6 \\ \hline 13 \end{array}$$

$$\begin{array}{ccccc} C_4 & C_3 & C_2 & C_1 & C_0 \\ 0 & 1 & 1 & 0 & 0 \\ & 0 & 1 & 1 & 1 \\ + & 0 & 1 & 1 & 0 \\ \hline & \boxed{1} & 1 & 0 & 1 \end{array}$$

sign bit '1'

inconsistent

wrong answer

$$\begin{array}{r} +7 \\ -3 \\ \hline +4 \end{array}$$

$$\begin{array}{ccccc} C_4 & C_3 & C_2 & C_1 & C_0 \\ 1 & 1 & 1 & 1 & 0 \\ & 0 & 1 & 1 & 1 \\ & 1 & 1 & 0 & 1 \\ \hline & \boxed{0} & 1 & 0 & 0 \end{array}$$

sign bit '0'

correct answer $\quad C_3 = C_4$

$$\begin{array}{r} -7 \\ +3 \\ \hline -4 \end{array}$$

$$\begin{array}{ccccc} C_4 & C_3 & C_2 & C_1 & C_0 \\ 0 & 0 & 1 & 1 & 0 \\ & 1 & 0 & 0 & 1 \\ + & 0 & 0 & 1 & 1 \\ \hline & \boxed{1} & 1 & 0 & 0 \end{array}$$

sign bit '1' consistent

correct answer.

correct

$$\begin{array}{r} -7 \\ + -6 \\ \hline -13 \end{array}$$

$$\begin{array}{ccccc} C_4 & C_3 & C_2 & C_1 & C_0 \\ 1 & 0 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 1 \\ + & 1 & 0 & 1 & 0 \\ \hline & 0 & 0 & 1 & 1 \end{array}$$

sign bit '0'

inconsistent

wrong answers

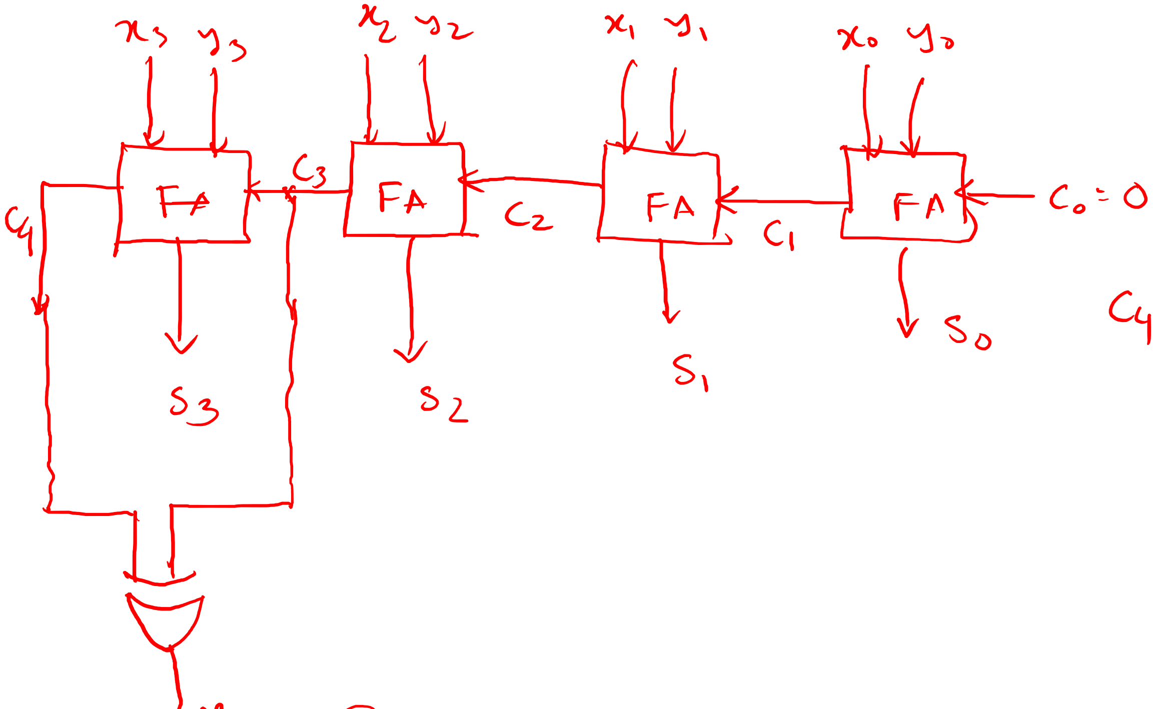
1. If $C_3 = 1,\ C_4 = 0$

or $C_3 = 0,\ C_4 = 1$

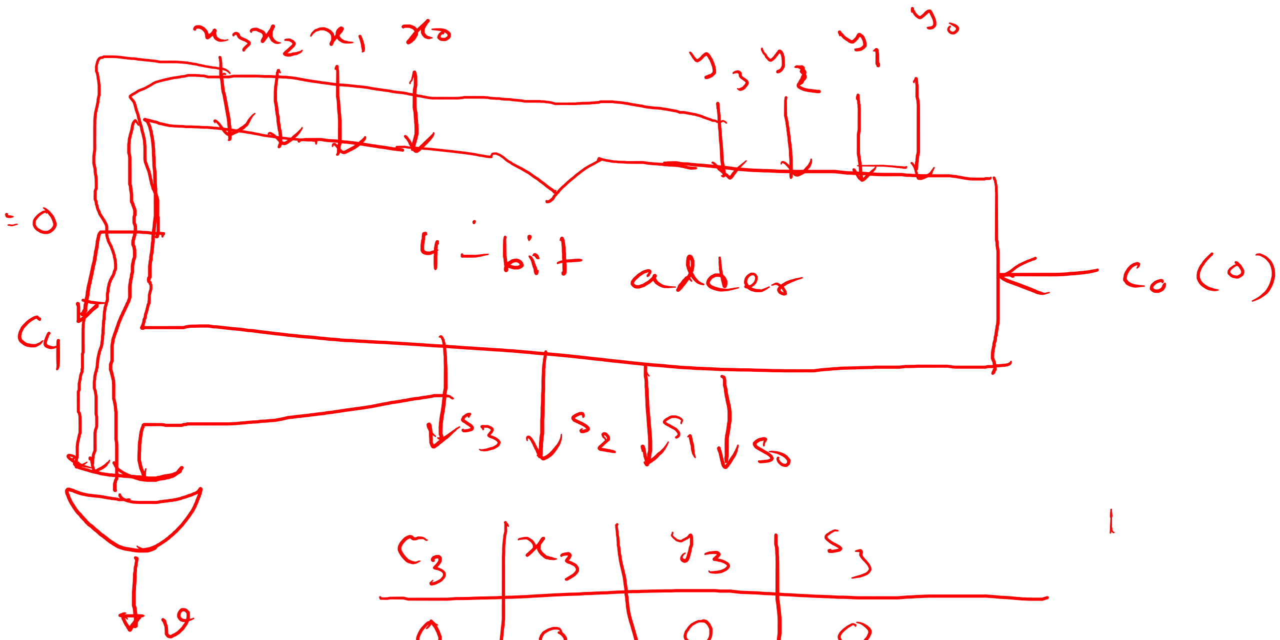
$v = C_3 \overline{C_4} + \overline{C_3} C_4$

$v = C_3 \oplus C_4$ overflow

4-bit ripple carry adder



$v = C_3 \oplus C_4$



$C_3 = x_3 \oplus y_3 \oplus S_3$

$$\boxed{v = x_3 \oplus y_3 \oplus S_3 \oplus C_4}$$

C_3	x_3	y_3	S_3
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Carry lookahead adder

$$\begin{array}{cccc} c_3 & c_2 & c_1 & c_0 \\ x_3 & x_2 & x_1 & x_0 \\ y_3 & y_2 & y_1 & y_0 \\ \hline \end{array}$$

$i = 0$

$s_0 = x_0 \oplus y_0 \oplus c_0$

$g_0 = x_0 y_0 \qquad p_0 = x_0 + y_0$

$c_1 = g_0 + c_0 p_0$

$i = 1$

$$s_i = x_i \oplus y_i \oplus c_i$$

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$= g_i + c_i (x_i + y_i) = g_i + c_i p_i$$

$$g_i = x_i y_i$$

$$p_i = x_i + y_i$$

$$s_{i+1} = x_{i+1} \oplus y_{i+1} \oplus c_{i+1}$$

$$c_{i+2} = x_{i+1} \cdot y_{i+1} + c_{i+1} (x_{i+1} + y_{i+1})$$

$$= g_{i+1} + c_{i+1} p_{i+1} = g_{i+1} + (g_i + c_i p_i) p_{i+1}$$

$$= g_{i+1} + g_i p_{i+1} + c_i p_i p_{i+1}$$

Carry Lookahead Adder

To reduce the delay caused by the effect of carry propagation through the ripple-carry adder, we can attempt to evaluate quickly for each stage whether the carry-in from the previous stage will have a value 0 or 1. If a correct evaluation can be made in a relatively short time, then the performance of the complete adder will be improved.

From Figure 5.4*b* the carry-out function for stage i can be realized as

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

If we factor this expression as

$$c_{i+1} = x_i y_i + (x_i + y_i) c_i$$

then it can be written as

$$c_{i+1} = g_i + p_i c_i \qquad \textbf{[5.3]}$$

where

$$g_i = x_i y_i$$
$$p_i = x_i + y_i$$

The function g_i is equal to 1 when both inputs x_i and y_i are equal to 1, regardless of the value of the incoming carry to this stage, c_i. Since in this case stage i is guaranteed to generate a carry-out, g is called the *generate* function. The function p_i is equal to 1 when at least one of the inputs x_i and y_i is equal to 1. In this case a carry-out is produced if $c_i = 1$. The effect is that the carry-in of 1 is propagated through stage i; hence p_i is called the *propagate* function.

Carry Lookahead Adder

Expanding the expression 5.3 in terms of stage $i-1$ gives

$$c_{i+1} = g_i + p_i(g_{i-1} + p_{i-1}c_{i-1})$$
$$= g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1}$$

The same expansion for other stages, ending with stage 0, gives

$$c_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots + p_i p_{i-1} \cdots p_2 p_1 g_0 + p_i p_{i-1} \cdots p_1 p_0 c_0 \quad \textbf{[5.4]}$$

2-stage carry lookahead adder

The generate and propagate functions:

$$C_1 = g_o + C_o p_o$$
$$C_1 = g_1 + g_o p_1 + C_o p_1 p_o$$

$$g_o = x_o y_o$$
$$g_1 = x_1 y_1$$
$$p_o = x_o + y_o$$
$$p_1 = x_1 + y_1$$

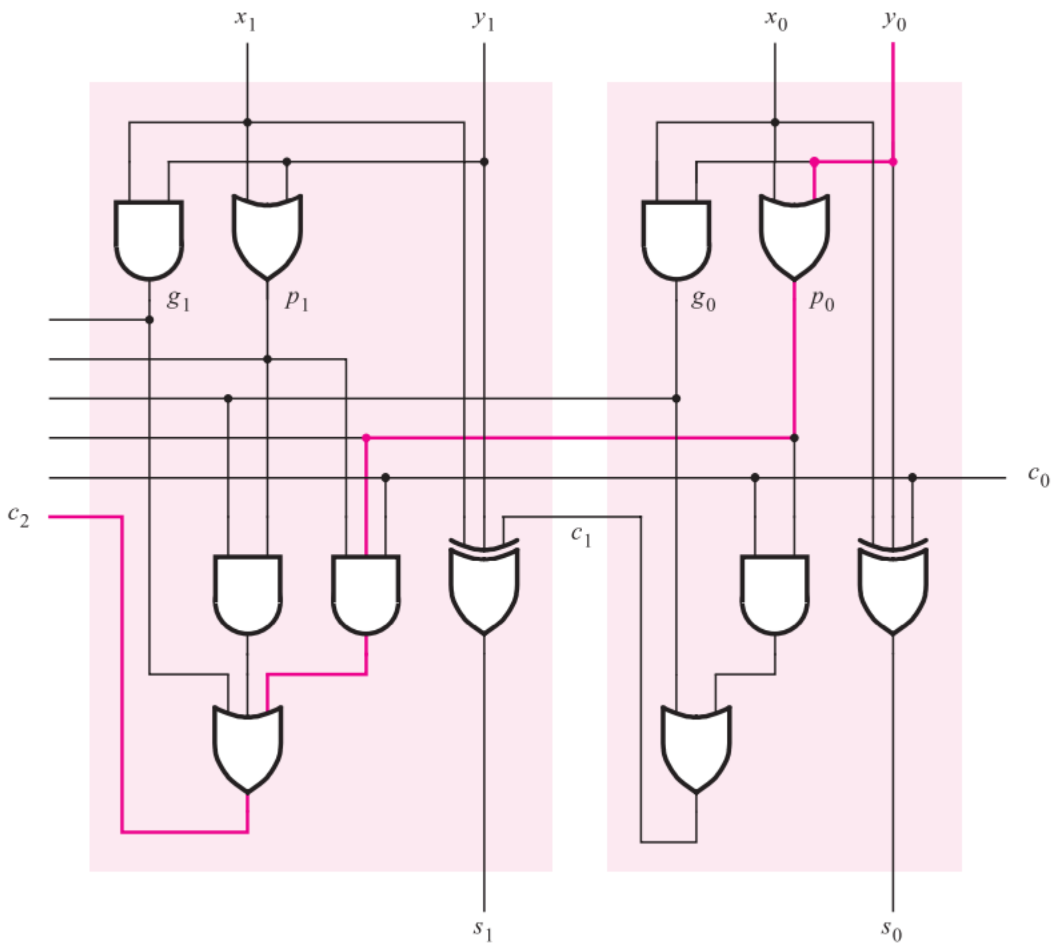


Figure 5.16 The first two stages of a carry-lookahead adder.

Design of Arithmetic Circuits Using VHDL

- VHDL code for full adder

```
LIBRARY ieee;
USE ieee_std_logic_1164.all;

ENTITY fulladd IS
        PORT(Cin,x,y    :IN  STD_LOGIC;
             s,Cout     :OUT STD_LOGIC);
END fulladd;


ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
  s<= x XOR y XOR Cin;
  Cout<=(x AND y) OR (Cin AND x) OR (Cin AND y);
END LogicFunc;
```

VHDL for 4-Bit Adder

```
LIBRARY ieee;
USE ieee_std_logic_1164.all;

ENTITY adder4 IS
    PORT(Cin     :IN STD_LOGIC;
         X,Y     :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         S       :OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
         Cout, OVF :OUT STD_LOGIC);
END adder4;

ARCHITECTURE Behavior OF adder4 IS
     SIGNAL Sum: STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
   Sum<= ("0" & X) +Y + Cin;
   S <= Sum(3 DOWNTO 0);
   Cout <=Sum(4);
   OVF<=Sum(4) XOR X(3) XOR Y(3) XOR Sum(3);
END Behavior;
```

Other number Representation

- **Floating Point**

8 bit → 256

0 - 126

Value = + −M×2^{E-127}

M is mantesa which is LS 23 Bits

E is the exponent and is 8 MS bits

The sign bit is the MSB

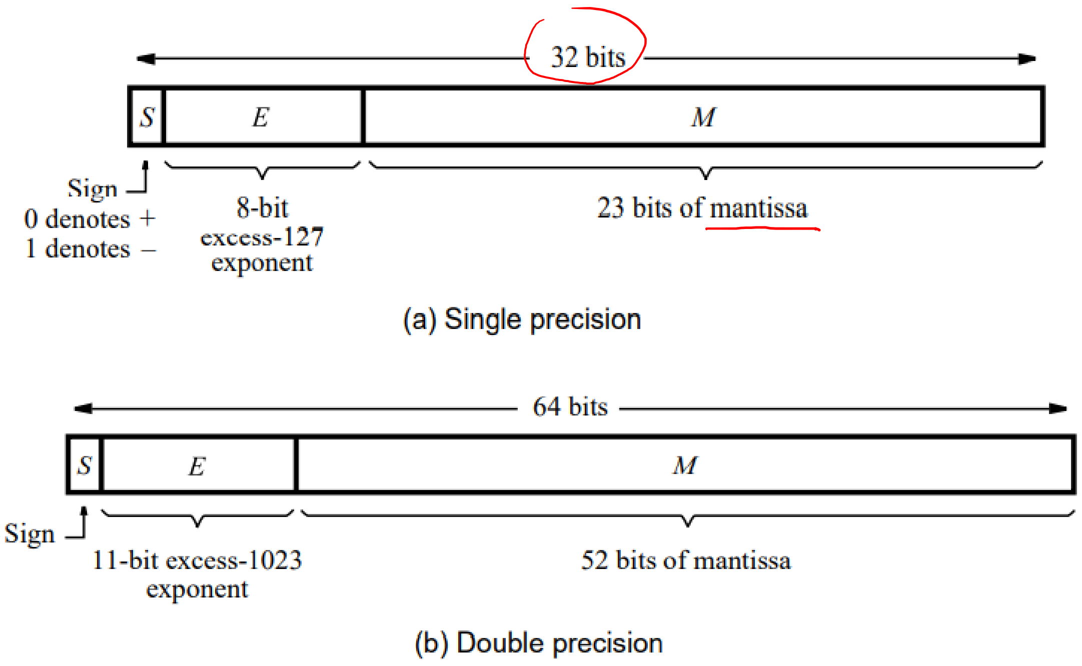


Figure 5.34 IEEE Standard floating-point formats.

Evaluate: $(3CC80000)_H$= 0.0244

$0.25 \times 10^{\pm 25}$

1

- **BCD (Binary Coded Decimal)**

It is a code for decimal numbers.

Convert each decimal digit into a 4-bit binary.

Example: Find the BDC to 58

BCD=0101 1000

A → 10
B → 11
C → 12

$(3CC80000)_H$

L

exponent

0 | 011 1100 1 | 100 1000 (16 Zeros)

Sign

Mantissa

$2^6 + 2^5 + 2^4 + 2^3 + 2^0$

$64 + 32 + 16 + 8 + 1$

$1.(2^{-1} + 2^{-4}) \times 2^{121-127}$

$= 1.(0.5 + \frac{1}{16}) \times 2^{-6}$

$(46)_{10}$

$(0100 \quad 0110)_{BCD}$

87

$(1000 \quad 0111)_{BCD}$

$$\begin{array}{r} 46 \\ +36 \\ \hline 82 \end{array}$$

→

0100	0110	BCD
0011	0110	BCD
(0111	1100)	
	+ 0110	← Correction
(1000	0010)	
8	2	

BCD Adder

Add each digit in binary, if the result is > 9, then add 6

Example: 46 + 36

46 = 0100 0110

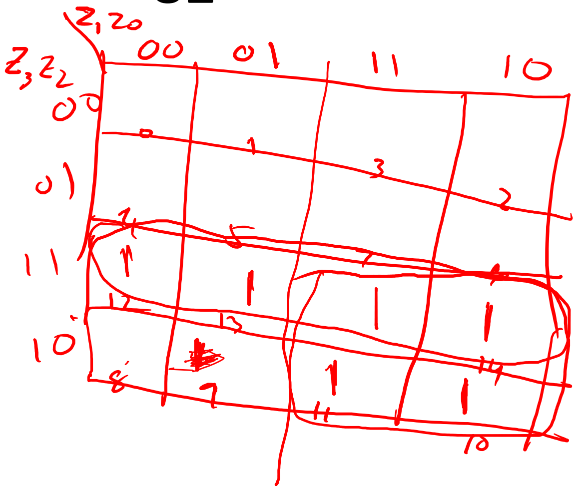
36 = 0011 0110

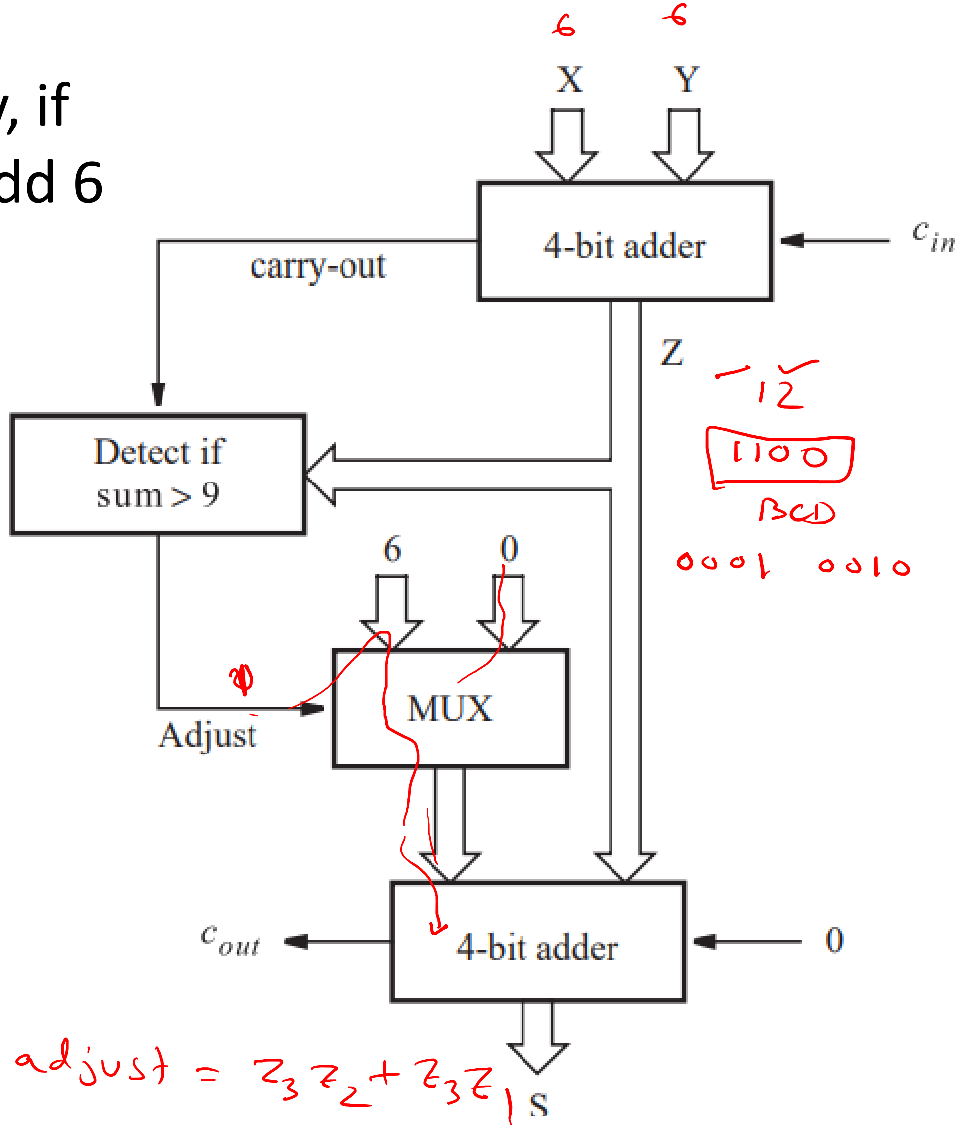
= 0111 1100

add 6 0110

= 1000 0010

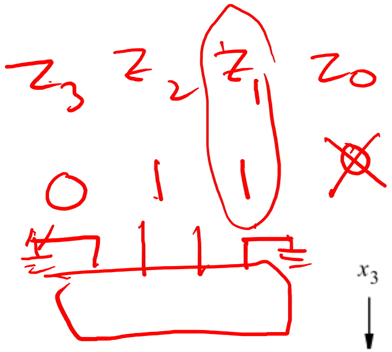
= 82

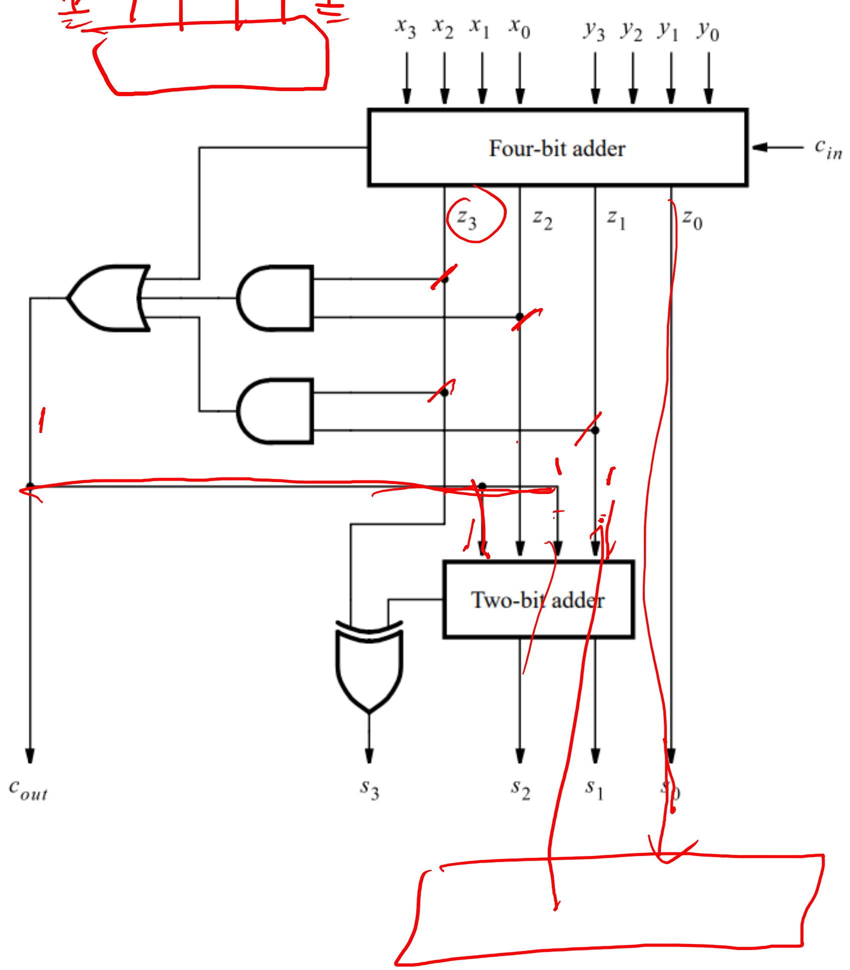




adjust = $z_3 z_2 + z_3 z_1$

Figure 5.36 Block diagram for a one-digit BCD adder.





Radix Complement

- Can use 10’s complement to decimal numbers:
- Example: 74 – 36
- 10’s complement of 36 = (99-36)+1=64
- 74-36=74+64=(1)38 ignore 1

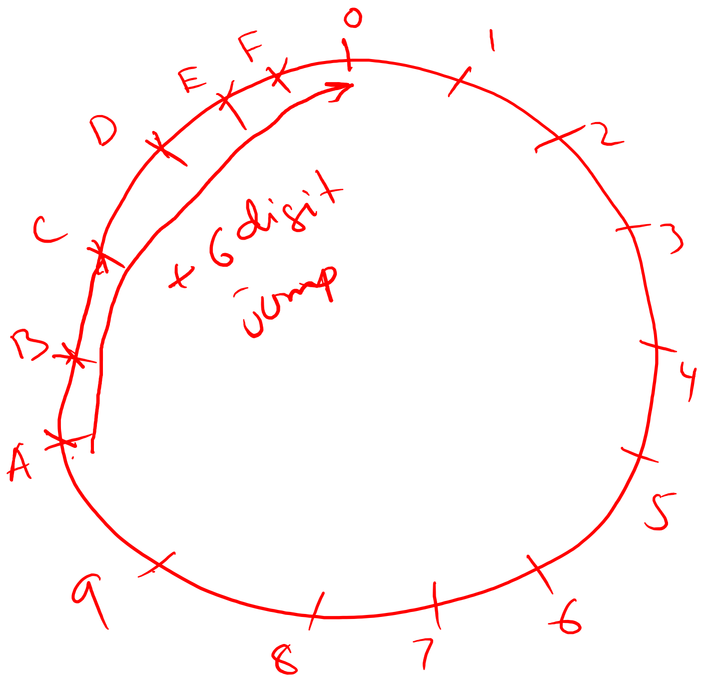
$10^2 - K = 100 - 36$

$$\begin{array}{r} 74 \\ 36 \\ \hline \end{array}$$

$100 - 36 = 64$

$$\begin{array}{r} 74 \\ +64 \\ \hline 138 \end{array}$$

Answer



ASCII Code

- The most popular code for representing information in digital systems is used for both letters and numbers, as well as for some control characters. It is known as the ASCII code, which stands for the American Standard Code for Information Interchange.
- Uses 7 bits for 128 characters number 1 = 0110001 = 49

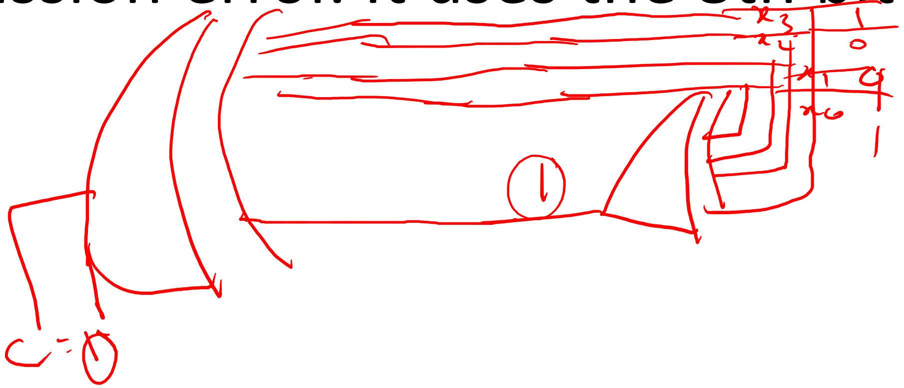
Table 5.3 The seven-bit ASCII code.

Bit positions	Bit positions 654							
3210	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	—	o	DEL

Parity and Parity Generator

- **Parity**

Parity is used to check data transmission error. It uses the 8th bit for parity check.



Even parity: number of 1's = even

Odd parity: number of 1's = odd

- **Parity Generator and Check**

For 4-bit generator use XOR (XOR generates a 1 if the number of 1's is odd):

1 ← 1 1 0 1

$$p = x_3 \oplus x_2 \oplus x_1 \oplus x_0$$

At the receiving end the checking is done using

$$c = p \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0$$

If C = 0 no error, if C=1 then an error occurred

Example

Q1. Given the 8-bit binary number 11011001, find the following:

a) The decimal value if the 8-bit number is an unsigned integer.

b) The decimal value if the 8-bit number is a signed integer.

c) The decimal value if the 8-bit number is 2's complement.

d) Covert the 8-bit number to a hexadecimal number.

Q2. Given A=1001, B=0011. Find the output for A+B and A-B.

4-bit ripple adder / 4-bit adder/subtractor

overflow

BCD adder

Carry lookahead adder → Not for test