TUT-VII

25.09.2012

Add binary numbers

1. 0.1011+0.1111

   Ans: 1.1010

2. 1001 1011+1001 1101

   Ans: 1 0011 1000

2. What is 2's complement of each of the following numbers? Determine the decimal value in each case

b) 11001
c) 1001
d) 111001
e) 01001

Ans: -7,-7,-7 and +9

Binary Addition with sign bit

- Consider a number uses 5 bits to represent its magnitude while its 6^{th} bit indicates its sign.
- What is representation range in this case?
- 2^5 (Thirty-two integer steps from 0 to max)
- This means that we can represent a number as high as $+ 31_{10} (011111_2)$ or as low as $-32_{10} (100000_2)$

Binary Addition with sign bit

- Add 17_{10} and 19_{10}
- Answer should be $+36_{10}$
  - $17_{10} = 10001_2$
  - $19_{10} = 10011_2$

```
  1   11   <--- Carry bits
  010001
+ 010011
--------
  100100    -28(10)
```

What is decimal equivalent of answer?

- Add (-17_{10}) and (-19_{10})

$-17_{10} = 101111_2$ $\qquad$ $-19_{10} = 101101_2$

```
                        1 1111  <--- Carry bits
(Showing sign bits)      101111
                       + 101101
                       --------
                        1011100
                        |
                  Discard extra bit
```

ANSWER: $011100_2 = +28_{10}$

Overflow condition

With the restrictions of the (six-bit) number field , the magnitude of the true and proper sum (36_{10}) exceeds the allowable limit called OVERFLOW CONDITION

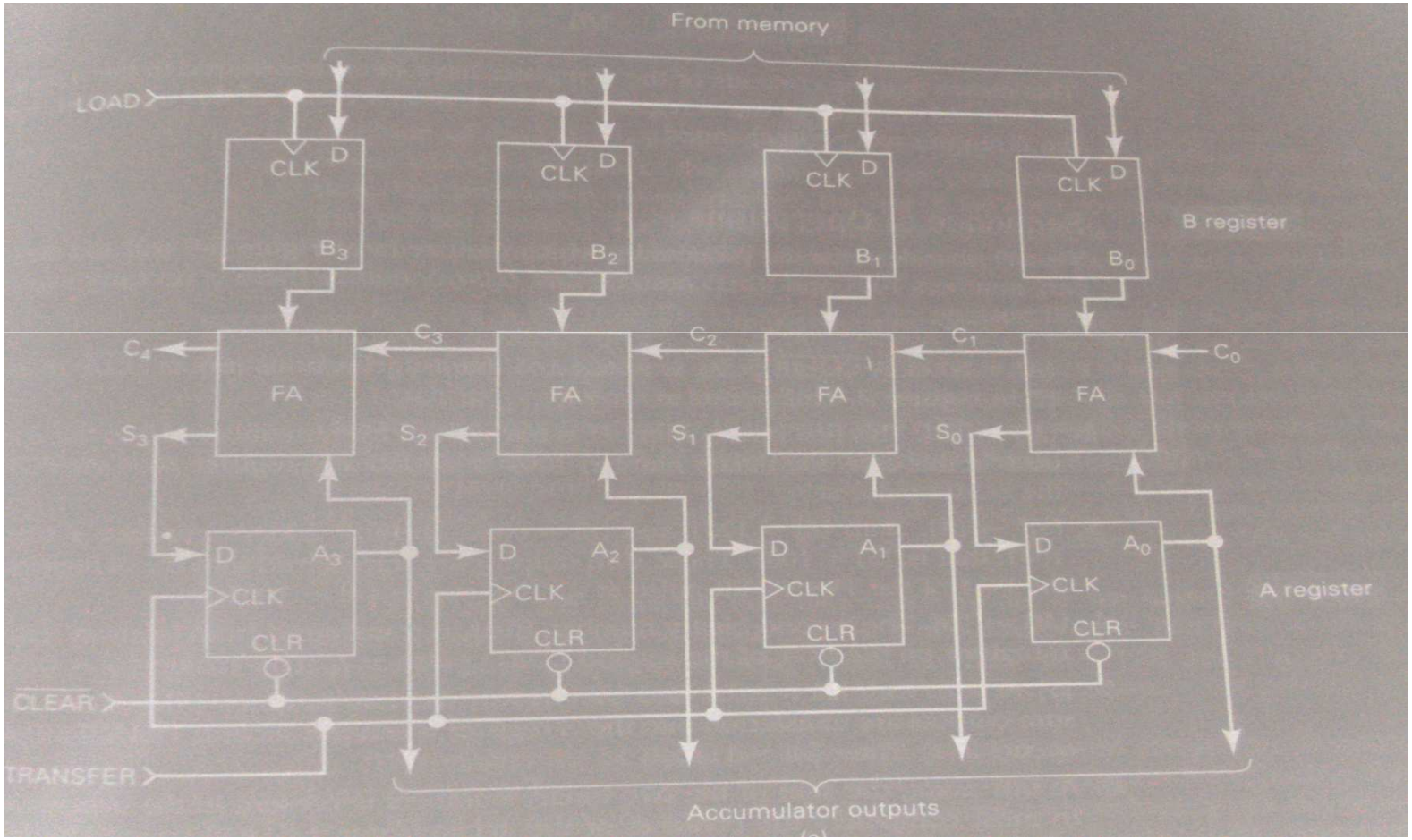
- Add the following in binary inn which cases OVERFLOW OCCURS?
  1. $3_{10}+2_{10}$
  2. $5_{10}+4_{10}$
  3. $-4_{10}+(-6_{10})$

How to detect Overflow condition?

- Examine sign bits of two numbers being added
- Examine the sign bit of the result
- Overflow occurs whenever the numbers being added are
  - both positive AND the sign bit of the result is 1

    OR
  - Both negative and the sign bit of the result is 0

Design a logic circuit for the figure ,that will produce output=1 whenever the overflow condition occurs

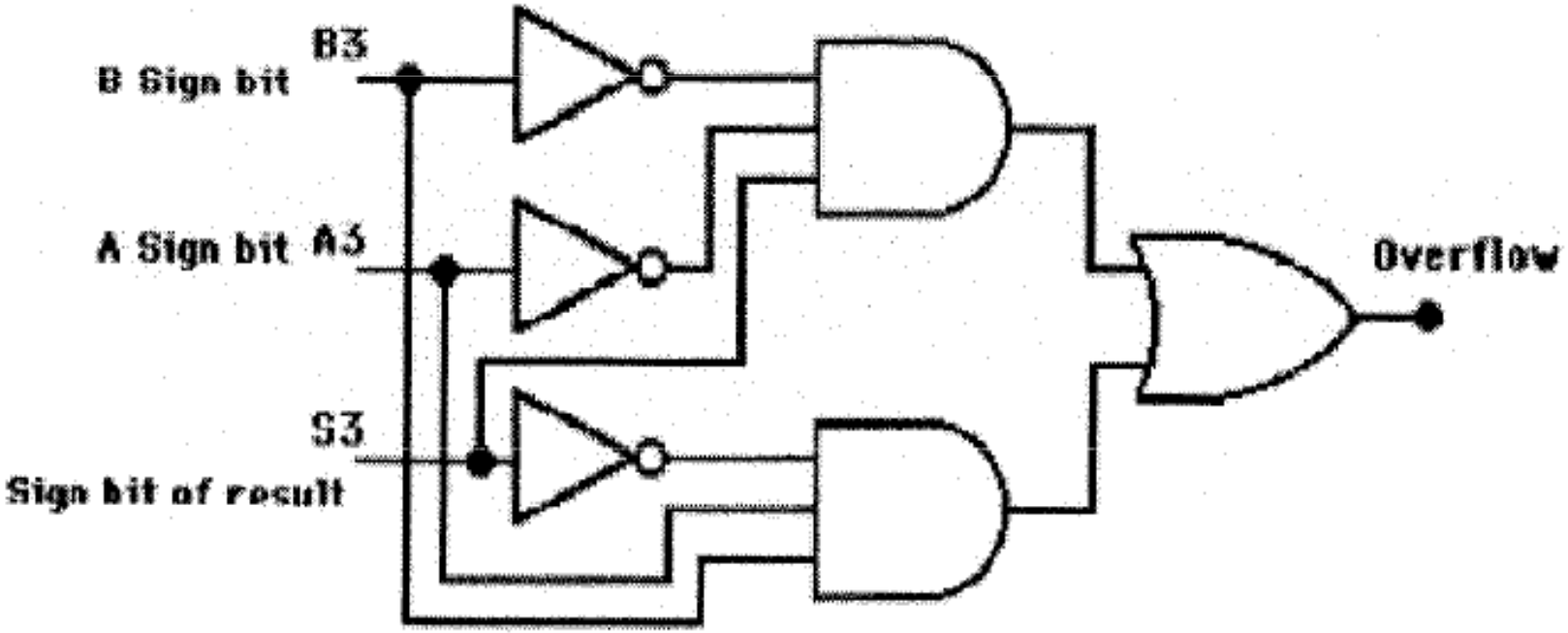


Logic Design of Overflow Circuit

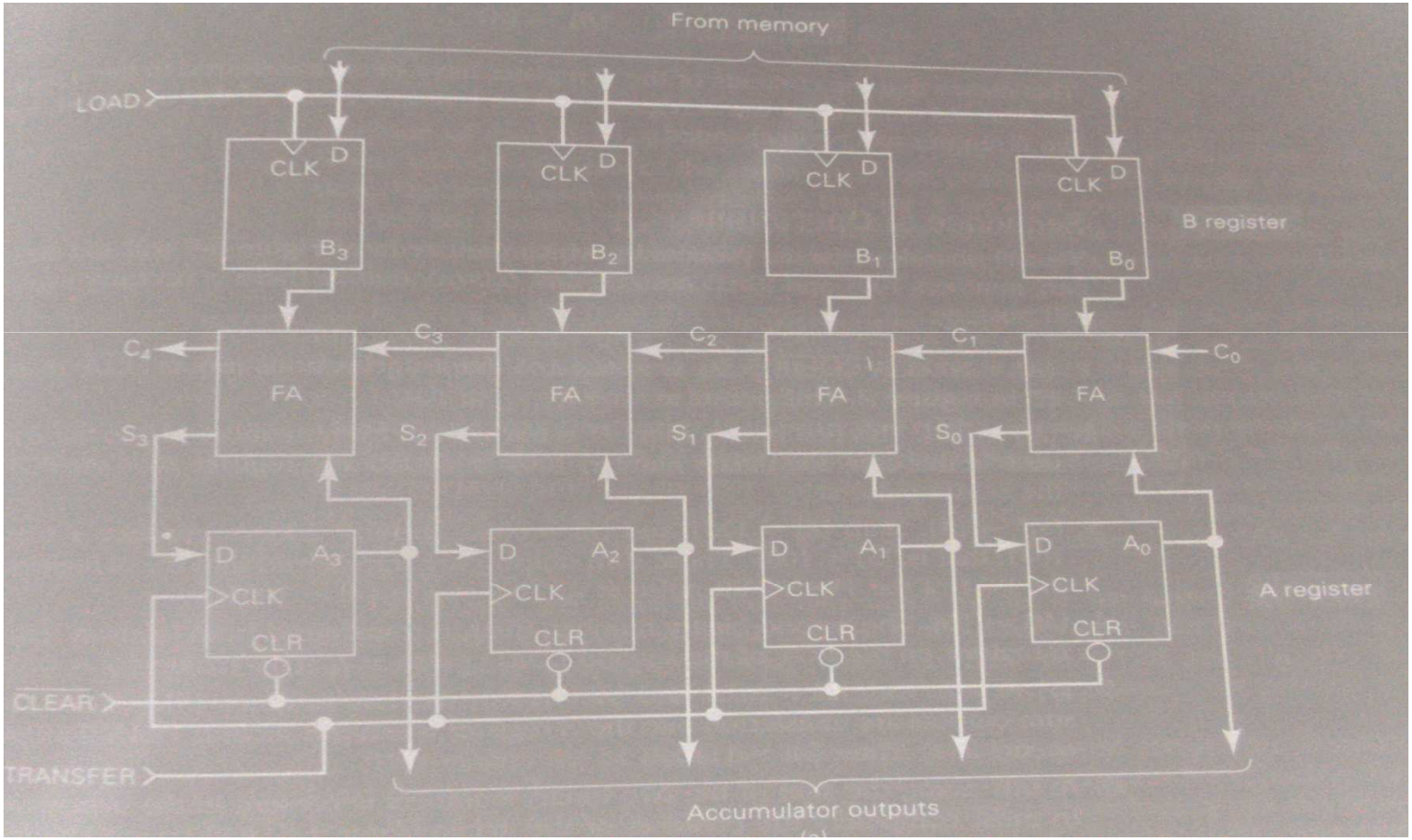
– Both nos. +ve AND the sign bit of the result is 1

OR

– Both nos. -ve and the sign bit of the result is 0

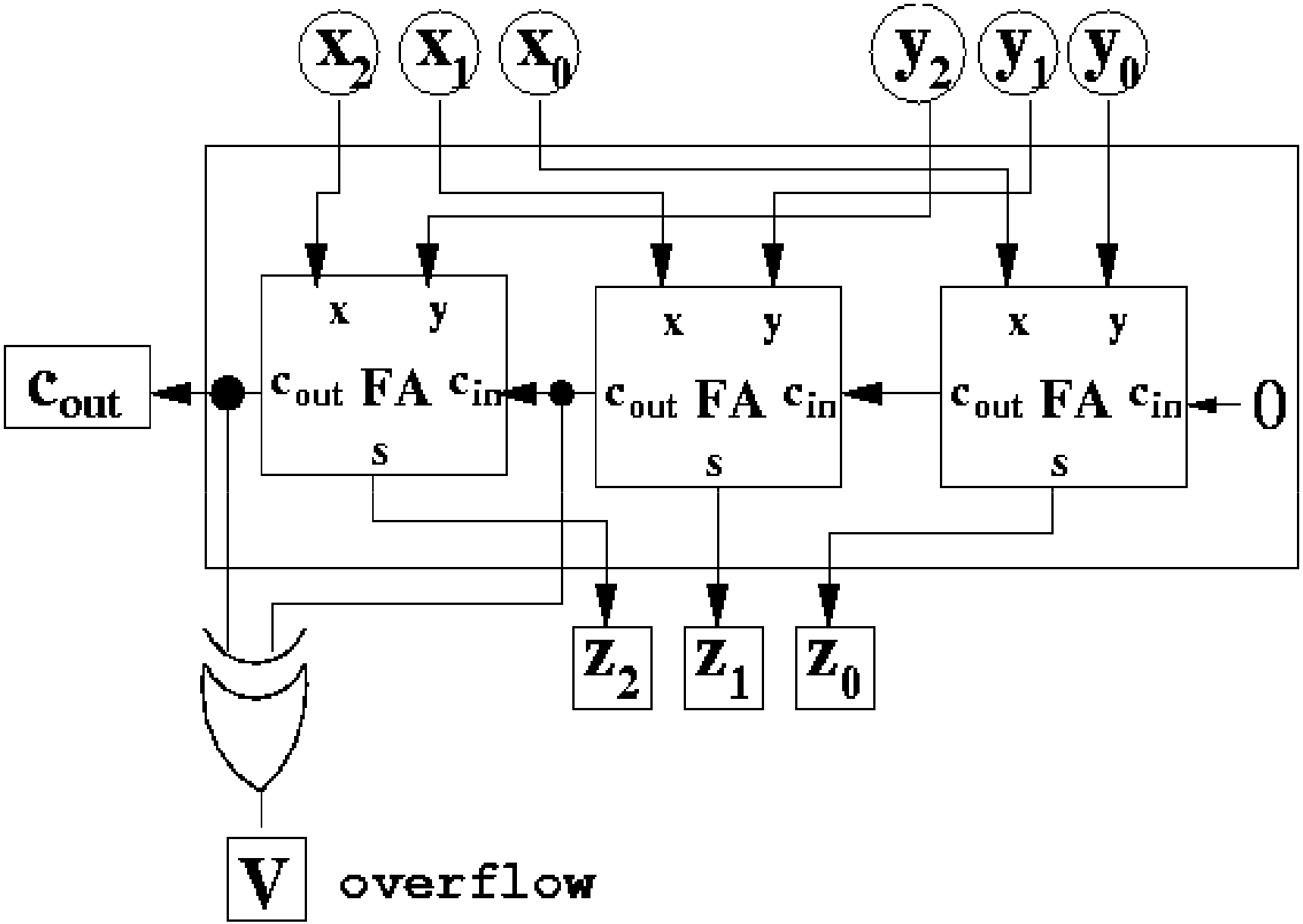


Design a logic circuit for the figure ,that will produce output=1 whenever the overflow condition occurs

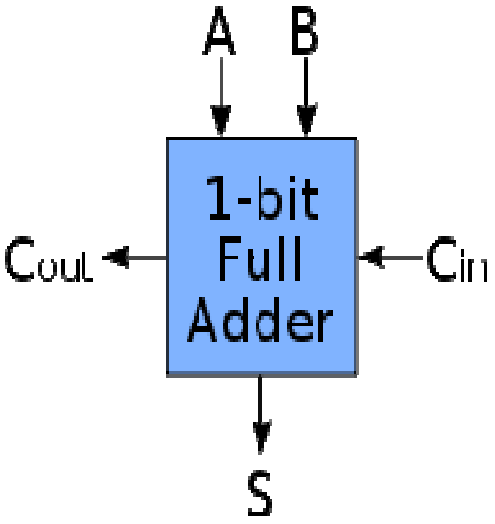


Alternate logic for Overflow detection

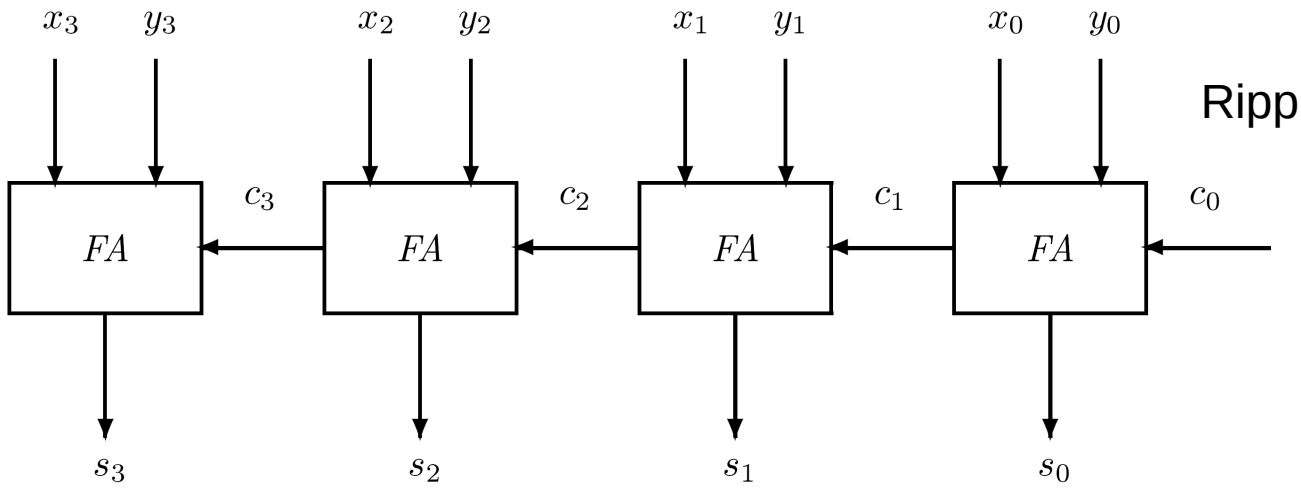
- **0 carried in, and 1 carried out**
- Only if both **A & B bit inputs are 1**.
- In this case,the sum is 0, and the carry out is 1.
- This is the case when to add two negative numbers, but the result is non-negative.
- **1 carried in, and 0 carried out**
- Only if both **A & B bit inputs are 0**.
- 0 is carried out, and the sum is 1. This is the case when you add two non-negative numbers and get a negative result.


x2
x1
x0
y2
y1
y0
x
y
cout FA cin
s
cout
0
Z2
Z1
Z0
V
overflow

ADDER



- Full adder - combinational digital circuit with input bits A,B (x_i,y_i)and incoming carry bit Cin (c_i) and output sum bit S (s_i)and outgoing carry bit Cout (C_{i+1})- incoming carry for next FA with input bits A_{i+1},B_{i+1} (X_{i+1},y_{i+1})
- $s_i = x_i \oplus y_i \oplus c_i$



Ripple carry adder

FA for subtraction

- FA used for adding 1 in least-significant position to implement subtract operation in two's complement
- One's complement of subtrahend taken and a forced carry added to FA in position 0 by setting $c_0=1$

Example

$T = 0$			1111
		+	0001
$T = \Delta_{FA}$	Carry		0001
	Sum		1110
$T = 2\Delta_{FA}$	Carry		0011
	Sum		1100
$T = 3\Delta_{FA}$	Carry		0111
	Sum		1000
$T = 4\Delta_{FA}$	Carry		1111
	Sum		0000

- x_0=1111 ; y_3,y_2,y_1,y_0=0001
- Δ_{FA} - operation time - delay
- Assuming equal delays for sum and cout
- Longest carry propagation chain when adding two 4-bit numbers

- worst-case delay - $n\Delta_{FA}$
- Adder produce correct sum after this fixed delay even for very short carry propagation time as in 0101+0010
- Subtract 0101-0010
  - adding two's complement of subtrahend to minuend
  - one's complement of 0010 is 1101
  - forced carry c_0=1 * result 0011

Reducing Carry Propagation Time

- Shorten carry propagation delay (accelerating carry propagation exist)
- Exercise - Find a technique for detection of carry completion

Carry-Look-Ahead Adders

- generate all incoming carries in parallel
- carries depend only on $x_{n-1},x_{n-2},\ldots,x_0$ and $y_{n-1},y_{n-2},\ldots,y_0$ - information available to all stages for calculating incoming carry and sum bit
- find out from inputs whether new carries will be generated and whether they will be propagated

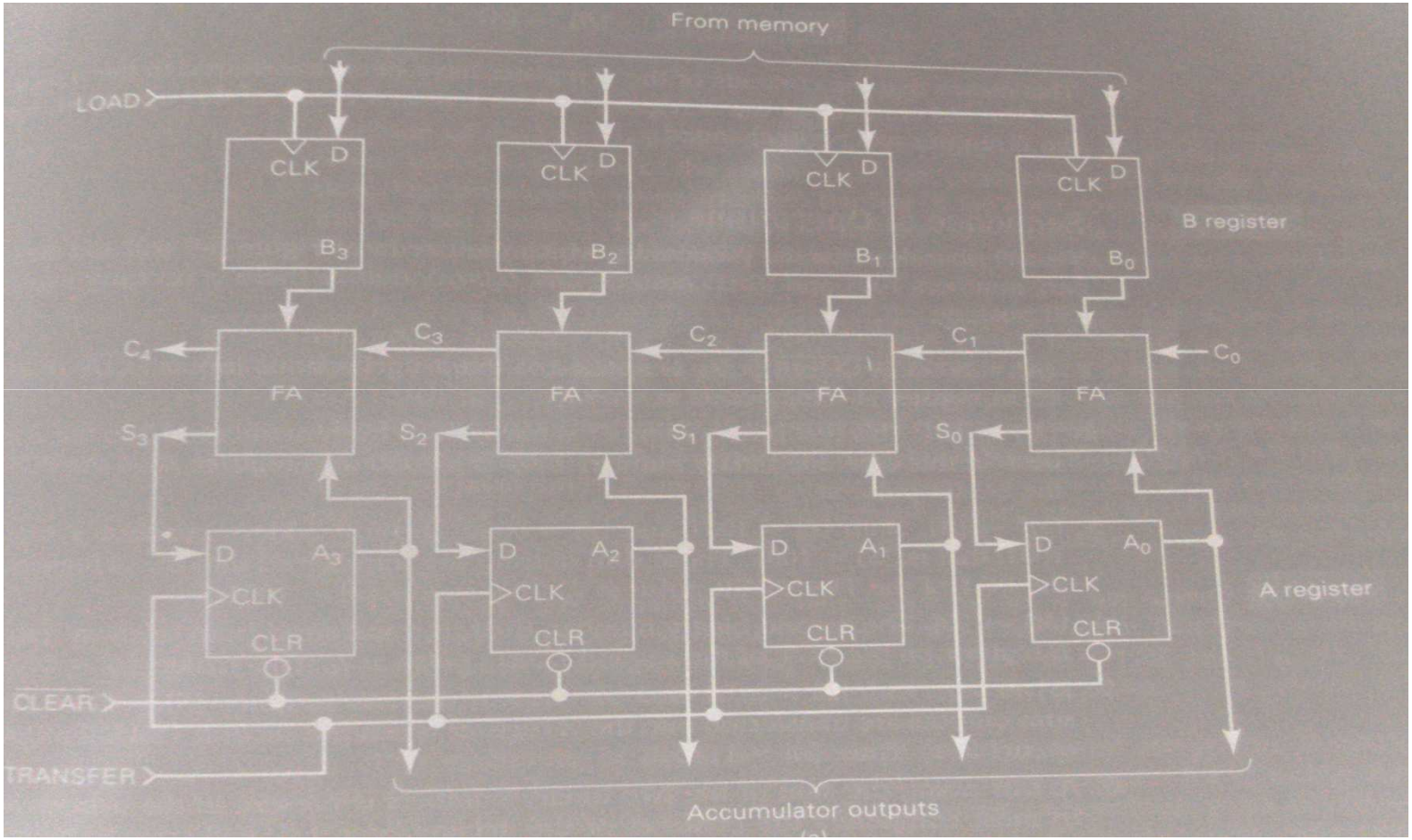
Carry Propagation

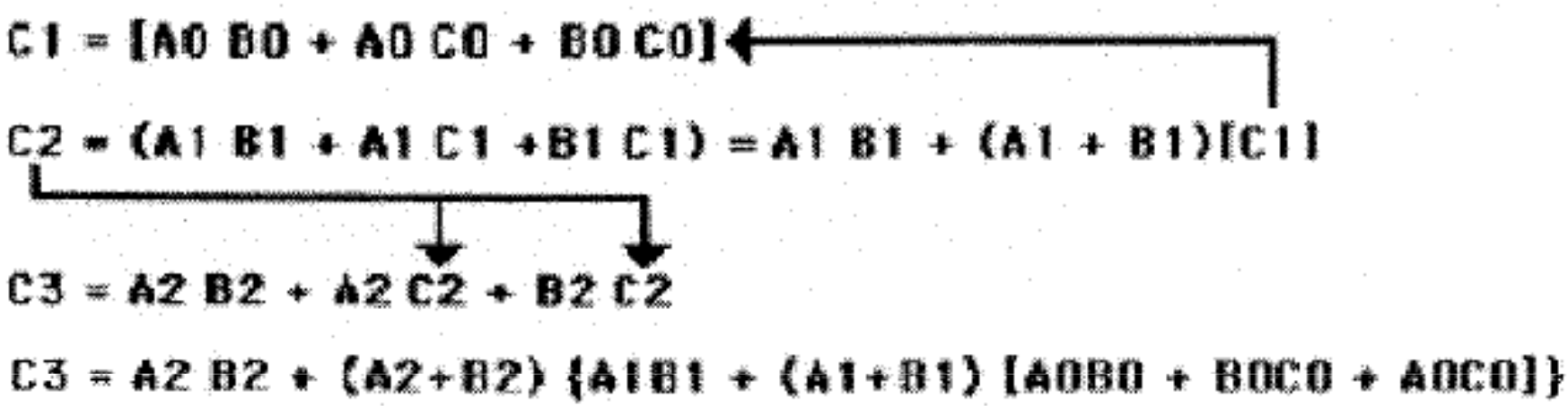
- If $x_i=y_i=1$ - carry-out generated regardless of incoming carry - no additional information needed
- If $x_i,y_i=10$ or $x_i,y_i=01$ - incoming carry propagated
- If $x_i=y_i=0$ - no carry propagation
- Let $G_i=x_i\,y_i$ - generated carry ;
- Let $P_i=x_i+y_i$ - propagated carry
- $c_{i+1}= x_i\,y_i + c_i\,(x_i + y_i) = G_i + c_i\,P_i$
- Substituting $c_i=G_{i-1}+c_{i-1}P_{i-1} \longrightarrow c_{i+1}=G_i+G_{i-1}Pi+c_{i-1}P_{i-1}P_i$
- $$\begin{aligned} c_{i+1} &= G_i + G_{i-1}P_i + G_{i-2}P_{i-1}P_i + c_{i-2}P_{i-2}P_{i-1}P_i = \cdots \\ &= G_i + G_{i-1}P_i + G_{i-2}P_{i-1}P_i + \cdots + c_0P_0P_1\cdots P_i. \end{aligned}$$
- All carries can be calculated in parallel from $x_{n-1},x_{n-2},\ldots,x_0$, $y_{n-1},y_{n-2},\ldots,y_0$, and forced carry c_0

Example - 4-bit Adder

$$
\begin{aligned}
c_1 &= G_0 + c_0 P_0, \\
c_2 &= G_1 + G_0 P_1 + c_0 P_0 P_1, \\
c_3 &= G_2 + G_1 P_2 + G_0 P_1 P_2 + c_0 P_0 P_1 P_2, \\
c_4 &= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + c_0 P_0 P_1 P_2 P_3
\end{aligned}
$$

Design a look-ahead carry circuit for the adder shown in fugure.
Derive an expression for c3 in terms of A0,B0,C0,A1,B1,A2,B2




C1 = [A0 B0 + A0 C0 + B0 C0]
C2 = (A1 B1 + A1 C1 + B1 C1) = A1 B1 + (A1 + B1)[C1]
C3 = A2 B2 + A2 C2 + B2 C2
C3 = A2 B2 + (A2+B2) {A1B1 + (A1+B1) [A0B0 + B0C0 + A0C0]}