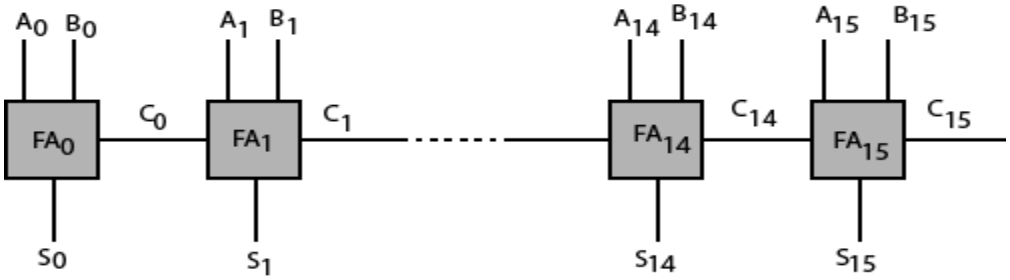
Tutorial 4
Computer Organisation

1. A 16-bit Ripple carry adder (RCA) is made up of 16 identical full adders (FA), as shown in the figure below. The carry-propagation of each FA is 13 ns and the sum-propagation delay of each FA is 16ns. The worst-case delay (in ns) of this 16-bit adder will be?



Ans: T = (n-1) T_{Carry} + Max (T_{Carry}, T_{sum})

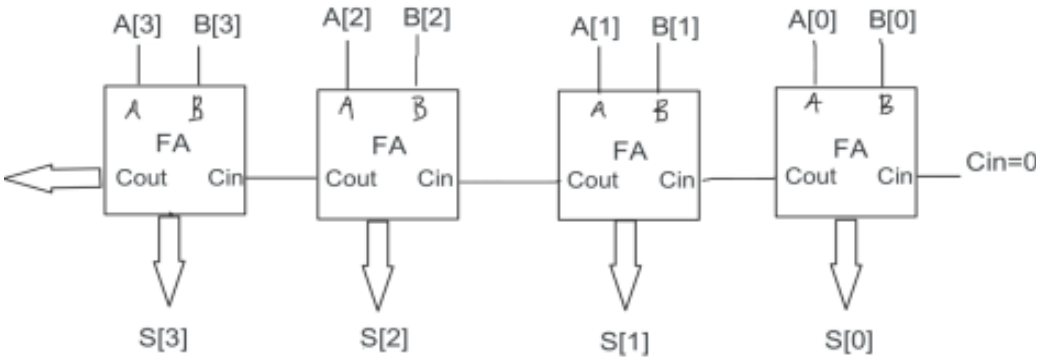
= 15 x 13 + 16 = 211ns

Explanation:

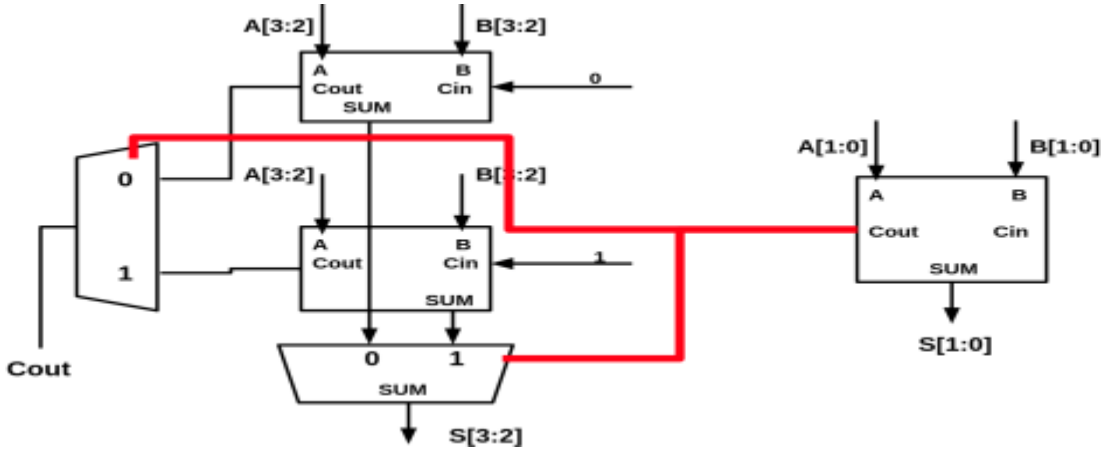
C_{14} will be available after 15 x 13 ie, 195ns. The FA_{15} adder takes 13 ns and 16 ns to produce carry and sum, respectively. So, the worst delay is (195 ns +16 ns)

2. Suppose the value of A[3:0] = 0011 and B[3:0] = 0001. Write down 0 or 1 to denote the state of each wire in the following Ripple carry Adder (RCA) and Carry Select Adder (CSA) circuits:

a.) RCA

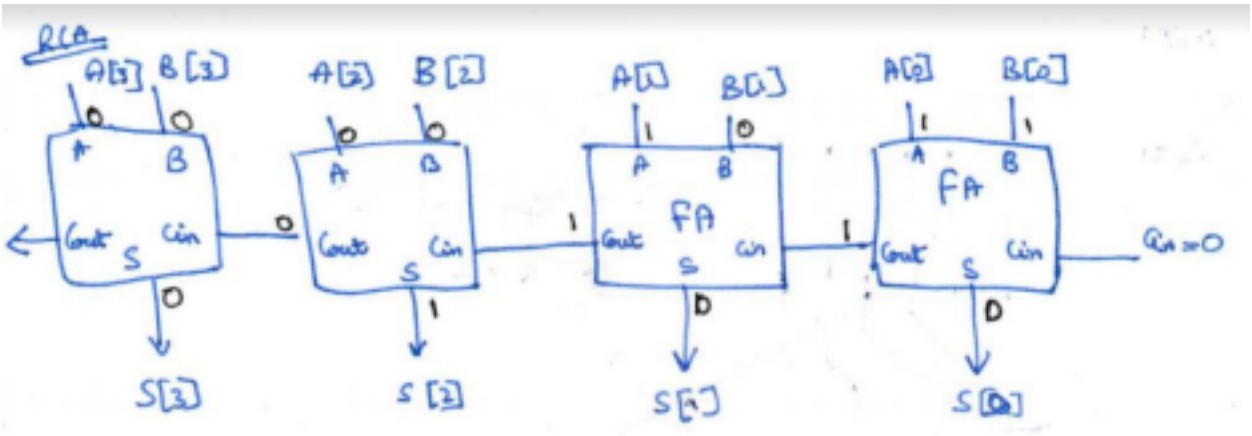


b.) CSA

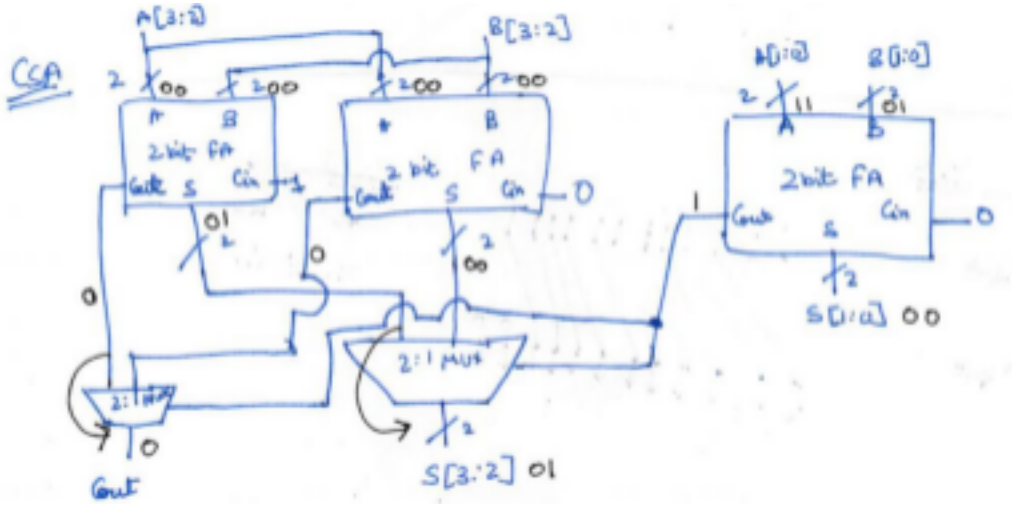


Ans:

a) RSA



b) CSA



All output wires are marked in black pen.

3. Compare Carry Select Adder (CSA) with Ripple Carry Adder (RCA) by filling the following table with faster/slower in the performance column and larger/smaller in the resource utilization column. Also, explain your answer with valid reasoning

Adder Type	**Performance**	**Resource Utilisation**
CSA		
RCA		

Ans:

Adder Type	Performance	Resource Utilisation
CSA	Faster	More Area
RCA	Slower	Less Area

Explanation:

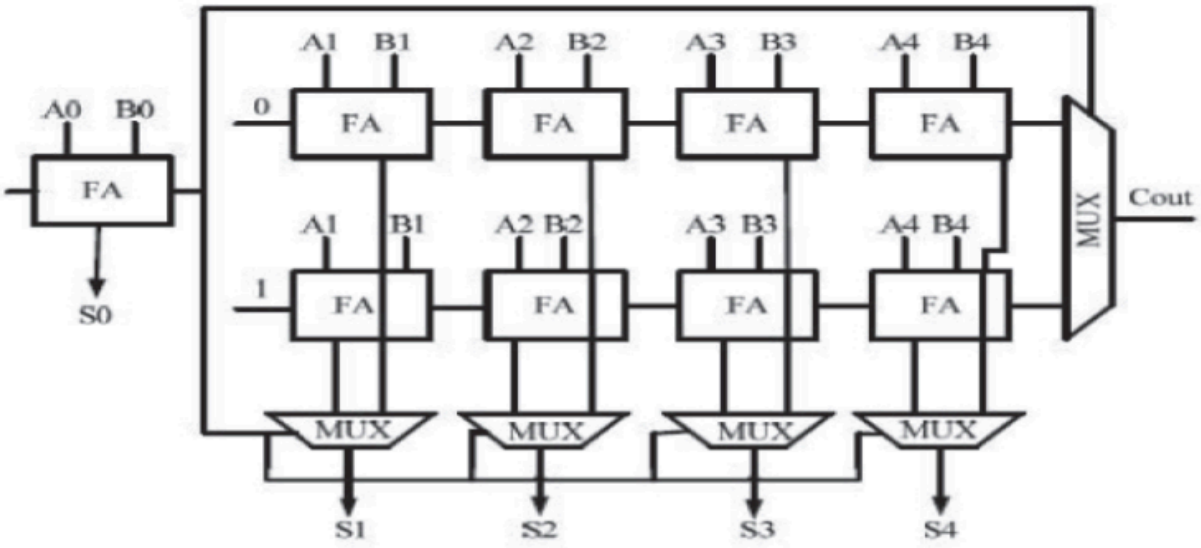
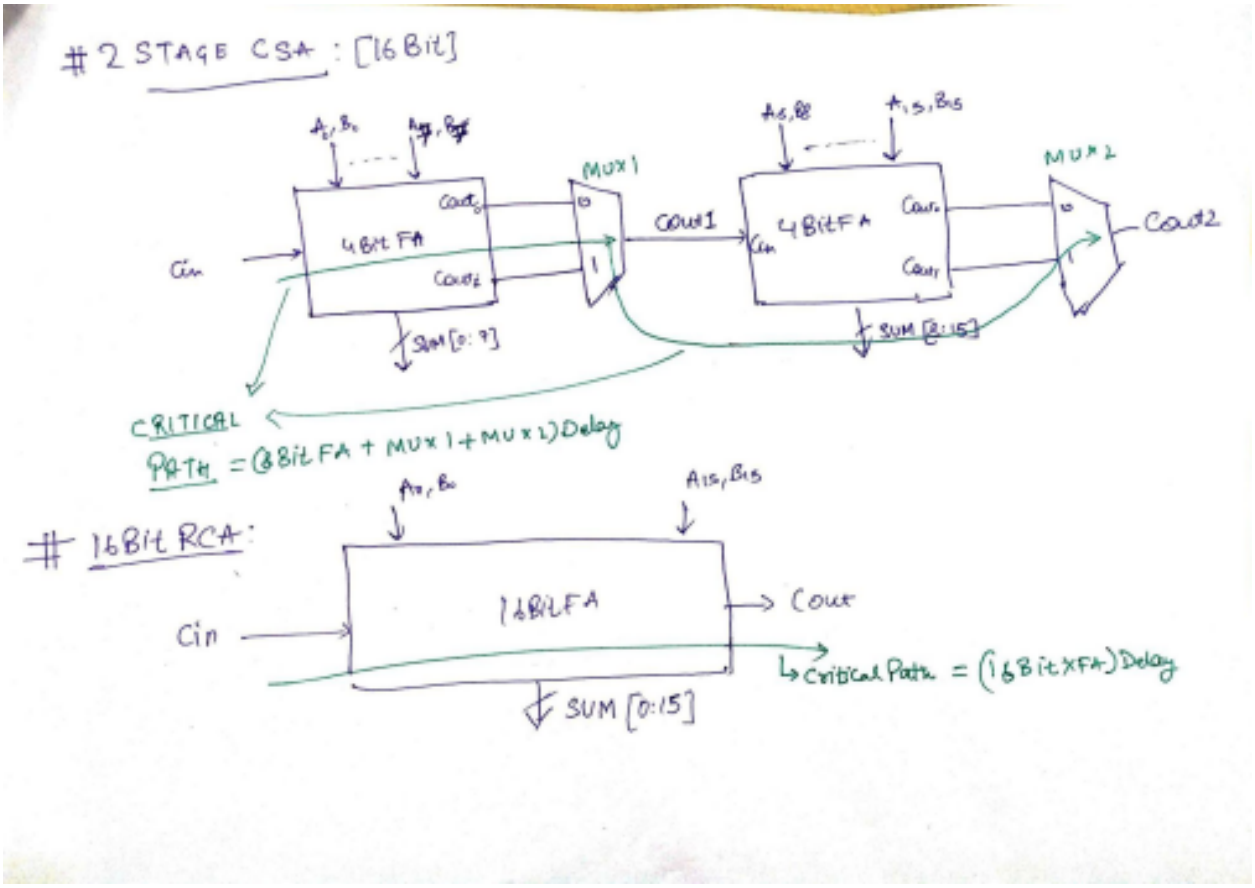


Fig. Logic Diagram of CSA

Each RCA pair in CSA can compute in parallel the value of sum before the previous stage carry comes. Thus, the critical path of an N bit adder is reduced. Delay in CSA is much lesser than RCA because the critical path in the case of a conventional adder is N-bit carry propagation path and one sum generating stage, while in the case of CSA, the critical path is (N/L)-bit carry propagation path and L stage multiplexer with one sum generating stage in the N- bit CSA, where L is number of stages in CSA as shown in above figure. Since L is much less than N and multiplexer delay is less than the delay in full adder, hence the delay in the CSA is much less than that in the RCA but there exists a copy of hardware in each stage which leads to an increase in the amount of power consumption and cost.

Illustration:



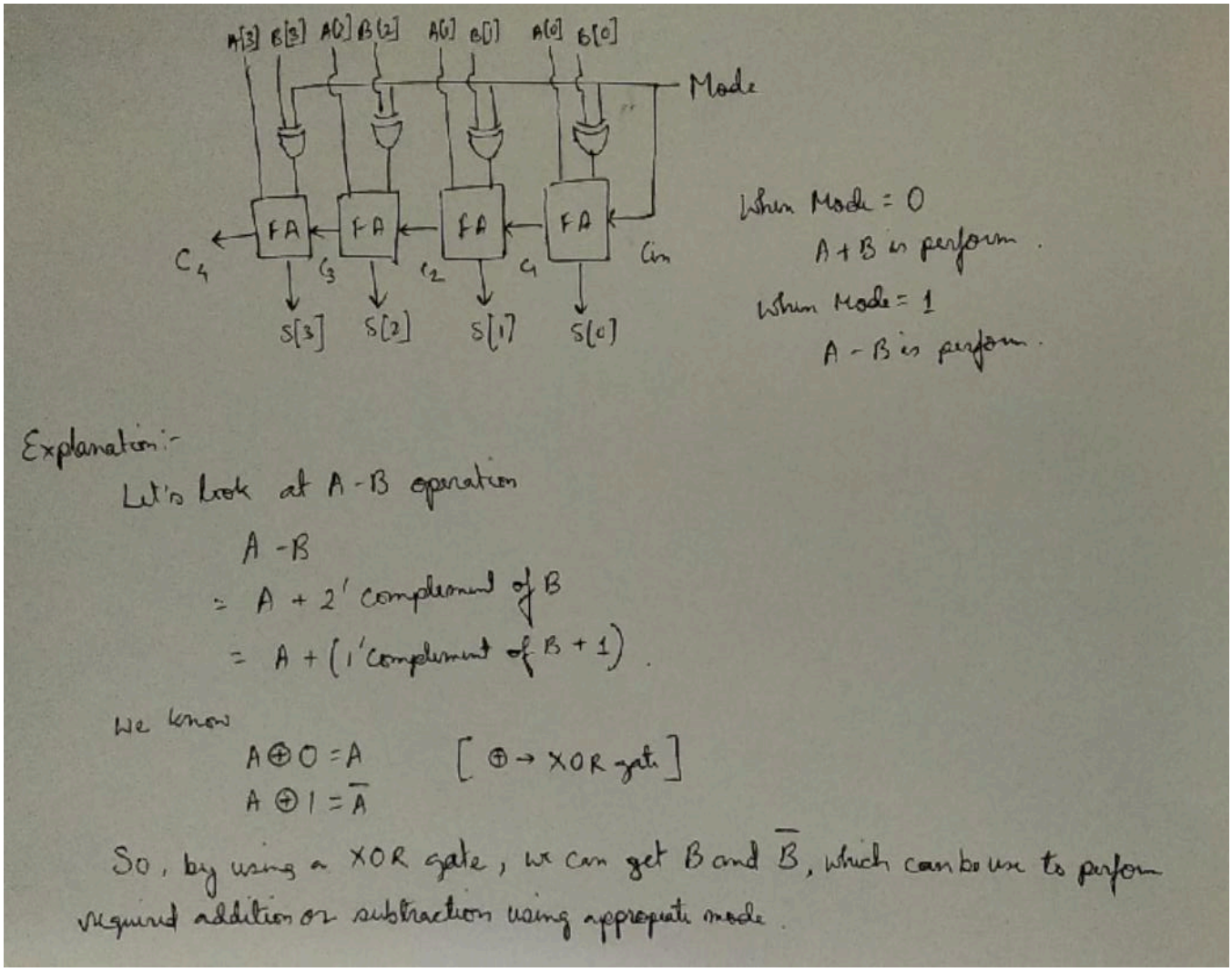
For the above figure, if we have to do 16 Bit Addition and there is MUX after 8 Bits to send Cout to the next stage, this means we have 2 stages in total; hence, while the 1st stage computes Cout1 for Cin = 1 & 0, 2nd stage also computes it parallelly (under the assumption 16-bit summands are applied simultaneously). Now, as soon as 1st stage selects the Cout1 corresponding to Cin, the second stage will immediately give the final Cout2 after just 1 MUX delay. But, if it was a RCA, the final Cout would only come after 16 Stages of Full Adders corresponding to 16-bit addition and the rippling of Cin to Cout. Hence, this means that 16 Bit critical Path delay has been reduced to 16/2 (N/L) + 2 MUX delay. And since the MUX delay is very small compared to FA, the performance(Time) gain is substantially large compared to RCA.

Note to the TAs -

Dear TAs, discuss them briefly in terms of performance and resource requirements. The motive is to get them acquainted with different adders so that they are able to differentiate between them in terms of performance and resource requirements. The students should be aware of the design tradeoffs.

For more detail, please refer: Source

4. Design a 4- bit Ripple carry adder that can perform addition and subtraction operations. Explain how it work.



5. We need to perform the following operations, where numbers are represented in 2's complement:

a) -87 + 256

b) 490 + 22

For each case:

1. Determine the minimum number of bits required to represent both summands. You might need to sign-extend one of the summands, since for proper summation, both summands must have the same number of bits.

2. Perform the binary addition in 2's complement arithmetic. The result must have the same number of bits as the summands.

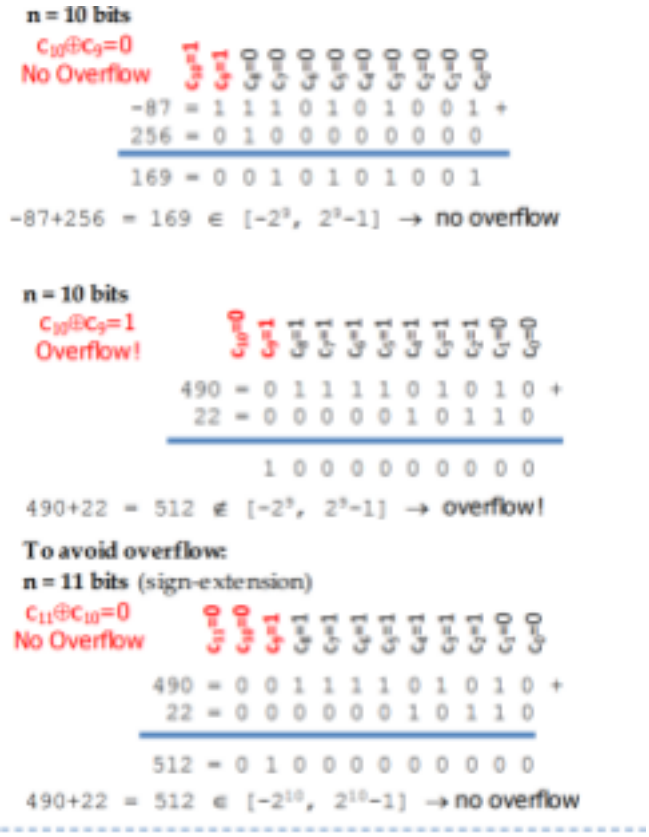
3. Determine whether there is overflow.

4. If there was an overflow, then redo the computation by sign extending both summands.

5. If we want to avoid overflow, what is the minimum number of bits required to represent both the summands and the result?

Ans:

1,2,3,4:



How to check for overflows: To determine if a computation overflowed or not, check the last two carries generated (the C_{in} and the C_{out} of the last adder stage, for example C_9 and C_{10} in part a respectively). If C_{in} XOR C_{out} is 1, then there was an overflow.

5. If the number of bits of the summands is n and m, then we need 1+max(n,m) bits for the result to ensure no overflows.

Source:

https://www.secs.oakland.edu/~llamocca/Courses/ECE2700/W21/Solutions%20-%20Homework%202.pdf

6. (i)For the following values of A and B, compute A+B and A-B. Note that both are denoted using 2's complement notation:

a. A = 0111 and B = 0011

b. A = 1110 and B = 1101

c. A = 1110 and B = 0011

d. A = 0011 and B = 1110

(ii)Propose a logic to detect Overflow in 2's Complement Addition.

Ans:

(i)

6a)
```
  0 0111
+ 0011
------
  1010
```
overflow! ⇒
```
  00 111
+ 00011
-------
  01010
```

0111 − 0011 = 0111 + (−0011)

−0011 = 2's Complement of 0011

0011 → 1100

+ 1

1101 ← 2's Complement of 0011

```
  111
  0111
+ 1101
------
 10100
```
No overflow.

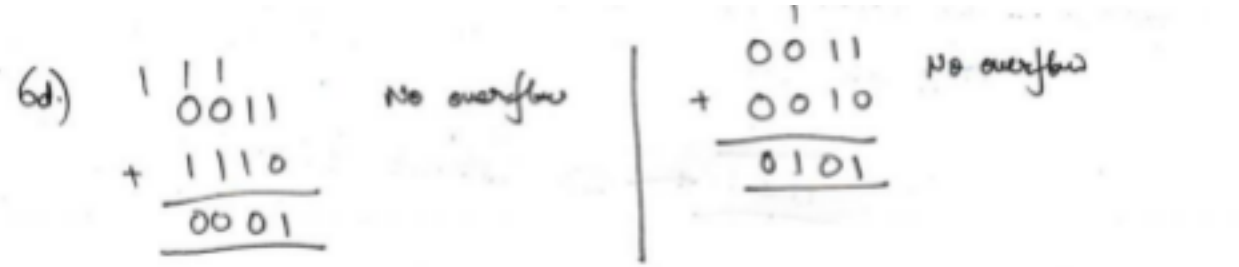
6b)
```
  11
  1110
+ 1101
------
 11011
```
No overflow!

1101 → 0011 (2's Complement)

```
  111
  1110
+ 0011
------
 10001
```
No overflow!

6c)
```
  111
  1110
+ 0011
------
 10001
```
No overflow!

```
  11
  1110
+ 1101
------
 11011
```
No overflow!



(ii) Circuit to detect overflow:

If suppose we have to add two numbers in binary of "n" bits each, A + B i.e., and let the MSBs for the Summands be An & Bn and the MSB of SUM be Sn

Now, overflow occurs during the following conditions:

1. Two negative numbers are added and an answer comes positive or
2. Two positive numbers are added and an answer comes as negative

Summarizing in a table:

An	Bn	Sn	Overflow
0	0	1	1
1	1	0	1
0	1	x	0
1	0	x	0

7.

(a) 0.2 = 0.001100110011…
0.1 = 0.0001100110011…
0.3 = 0.01001100110011…
(i) 0.2 = 0.00110 (5-bits)
0.2 = 0.001100 (6-bits)
(ii) 0.1 = 0.00011 (5b)
0.1 = 0.000110 (6b)
(iii) 0.3 = 0.01001 (5b)
0.3 = 0.010011 (5b)

(b) 0.2 + 0.1 = 0.3 (decimal)
(i) 0.00110
\+ 0.00011
0.01001
(ii) 0.001100
\+ 0.000110
0.010010

(c) Comparison:
(i) 0.3 (5b) = 0.01001
0.3 (b-(i)) = **0.01001**
Found Equal ⇒ No error
(ii) 0.3 (5b) = 0.010011
0.3 (b-(i)) = **0.010010**
Found Unequal ⇒ Error!
Explanation:

- Even though we might assume the decimal arithmetic operations will fetch exactly the same result in computer binary, there is a high probability of error in the final results, which depends on the precision limitations of the hardware, such as the size of the register used to represent the result.
- If a 5-bit register were only used, we would not have observed any errors as compared to the use of 6-bit register.
- This experiment can be extended to various data structures used in programming machines, such as float or double, where the selection of different datatypes will have different implications.