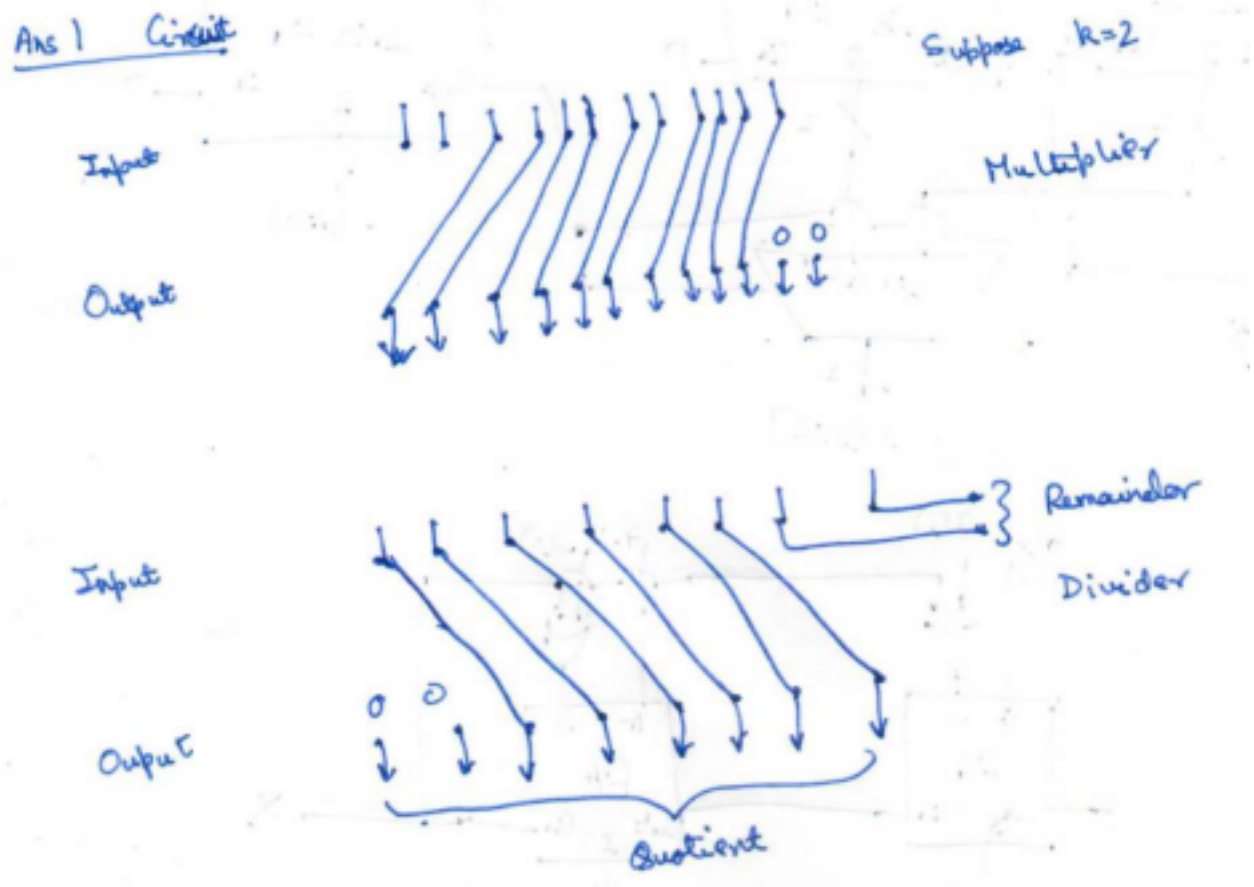
Tutorial 4 Solutions
CSE 112 Computer Organization

Preferred sequence: Q1, Q2, Q5, Q6(a, b), Q7(any one part), Q8(any one part), Q9(a(iii), b(iii)), Q10. Then cover the rest of the questions if time permits. Else clear doubts of rest of the questions in office hours. Try to touch all concepts in the tut.

Q1.



Binary unsigned representation of
7 : 111_2
10 : 1010_2

After multiplying by 2
7 * 2 = 14 = 1110_2
10 * 2 = 20 = 10100_2

After multiplying by 4
7 * 4 = 28 = 11100_2
10 * 4 = 40 = 101000_2
After division by 2
7/2: Q: 3 = 11_2, R: 1 = 01_2
10/2: Q: 5 = 101_2, R: 0 = 0_2

After division by 4

7/4: Q: 1 = 01_2, R: 3 = 11_2
10/4: Q: 2 = 10_2, R: 2 = 10_2

For multiplication by 2^k, the product is the argument left shifted by k. For example, multiplication of 000111_2 by $2^3$is 111000_2. Notice the left shift by 3.
For division by 2^k, the quotient is the argument right shifted by k, and the remainder is LSBs which are shifted out. For example, if we divide 000111_2 by 2^2, the quotient is 000001_2 and the remainder is 11_2. Notice the right shift by 2. Moreover, the 2 LSBs, which got shifted out, are the remainder.

Note to the TAs - Let students do normal multiplication and division in decimal, and then let them realize the bit shifting patterns in binary format on their own. The motive is to get them acquainted with bit shift patterns while multiplying and dividing by power of 2.

Q2.

Adder Type	Performance	Resource Utilization
CSA	Faster than RCA	Larger than RCA

Reasoning/Explanation: (SOURCE : Dear TAs, please refer to this link for more details)

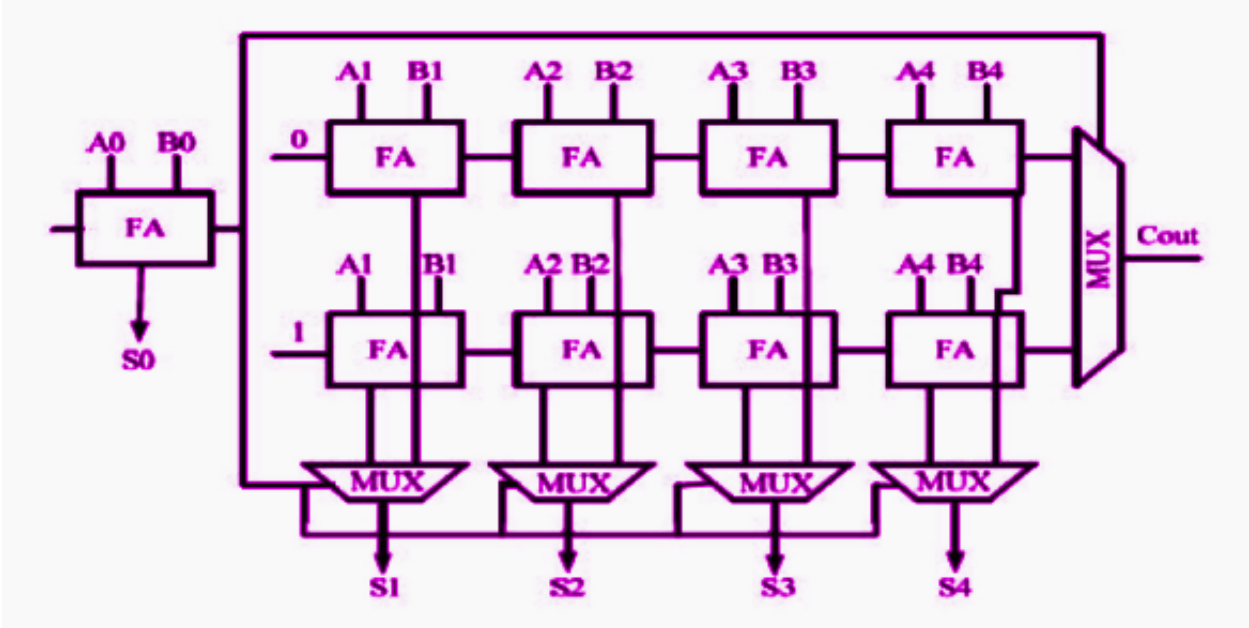
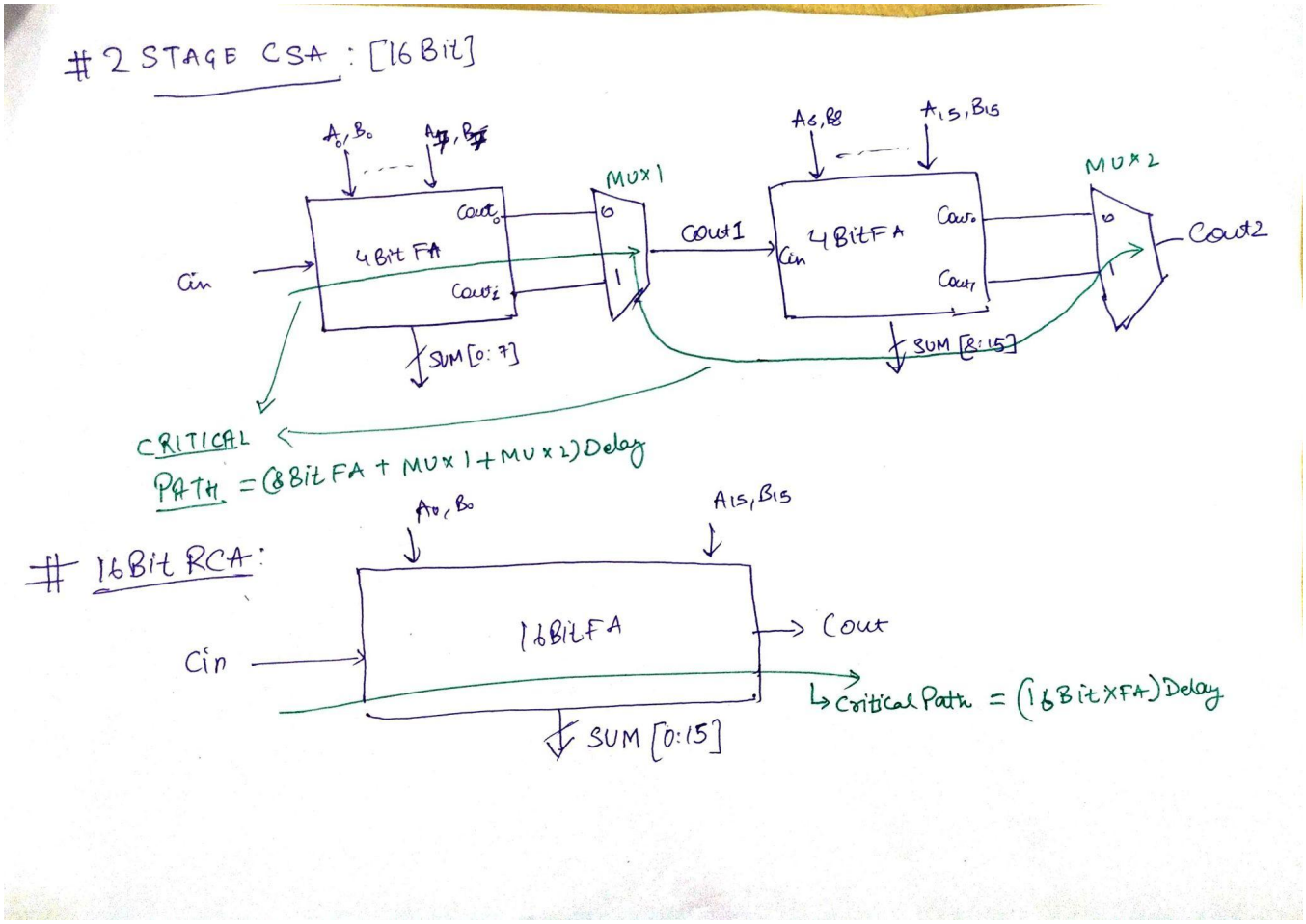


Fig. Logic Diagram of CSA

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 case of conventional adder is N-bit carry propagation path and one sum generating stage while in 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:



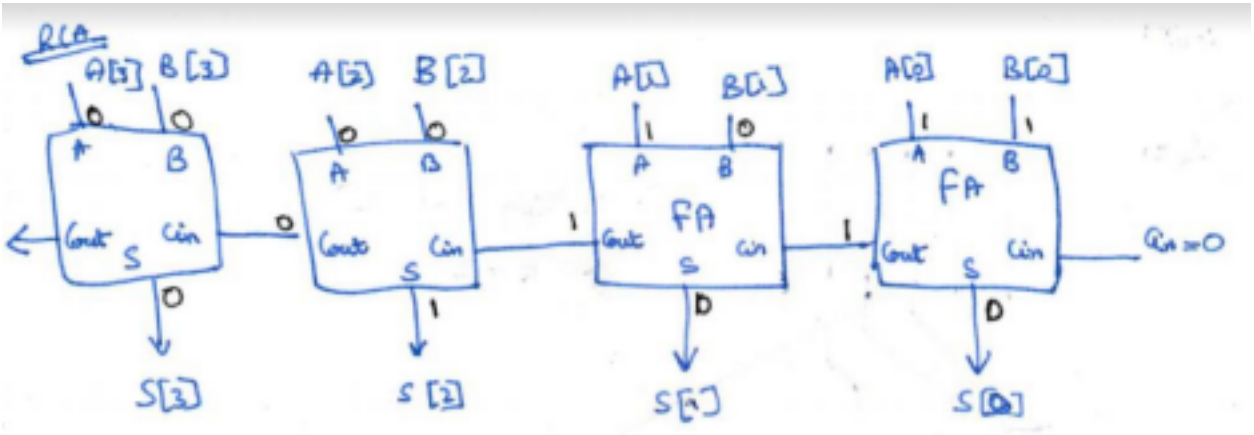
For above figure, if we have to do 16 Bit Addition and there is MUX after 8 Bits to send Cout to next stage, this means we have 2 stages in total, hence, while the 1st stage computes Cout1 for Cin = 1 & 0, 2nd stage also computes it parallelly (under the assumption 16 bit summands are applied simultaneously). 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, final Cout will 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 -

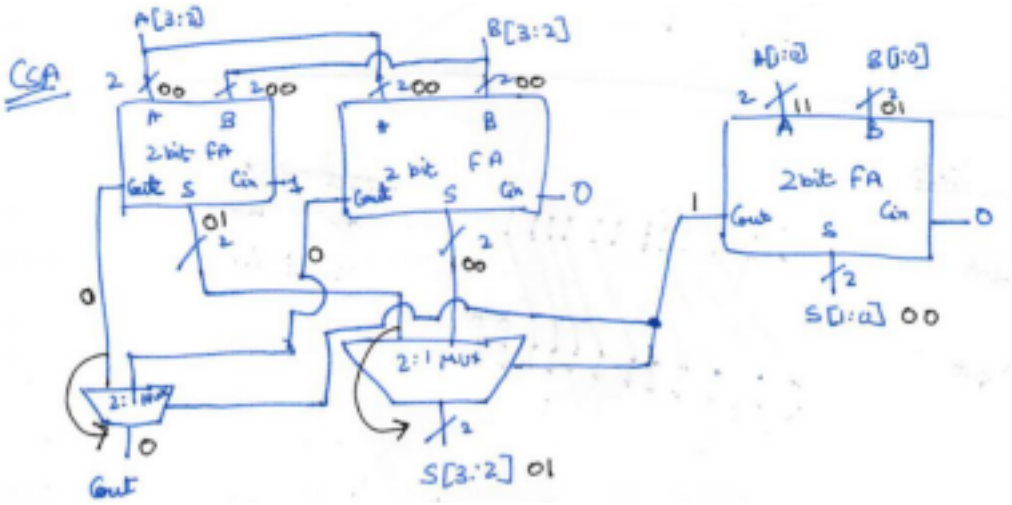
Dear TAs, discuss them briefly in terms of performance and resource requirement. The motive is to get them acquainted with different adders so that they are able to differentiate between them in terms of performance and resource requirements. The students should be aware of the design tradeoffs. You can just mention that there are other types of adders as well, with different tradeoffs (don't discuss the details of other adders).

Q3.

a. RCA



b. CSA



All output wires are marked in black pen.

Q4.

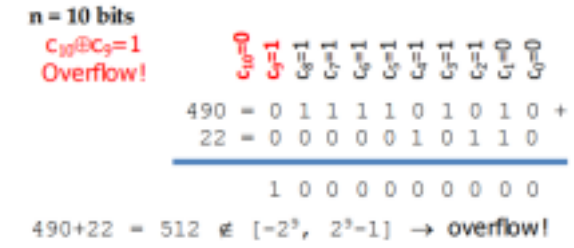
X is the add/sub line. If X is 1, then C_{in} is 1 and B is inverted. This effectively takes the 2's complement of B (Recall that to take 2's complement of a number, we first invert it, and then add 1 to it).

Therefore, if X is 1, then the output is A-B.

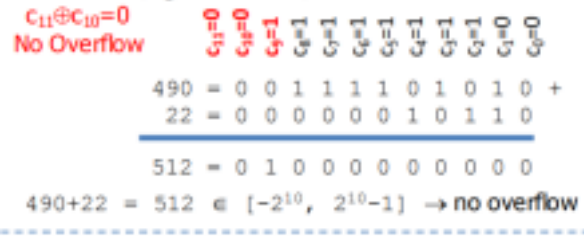
If X is 0, the output is A+B.

Q5.

1, 2, 3, 4:



To avoid overflow:
n = 11 bits (sign-extension)



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

NOTE to theTAs: This is a very important question which discusses how arithmetic in 2's complement systems work. Please make sure that everyone is able to solve this question.

Q6. (i)

6a)
```
  0 0111
+ 0011
------
  1010
```
Overflow! ⇒
```
  0011
+ 0011
------
  0110
```

$0111 - 0011 = 0111 + (-0011)$

-0011 = 2's Complement of 0011

$0011 \rightarrow 1100$

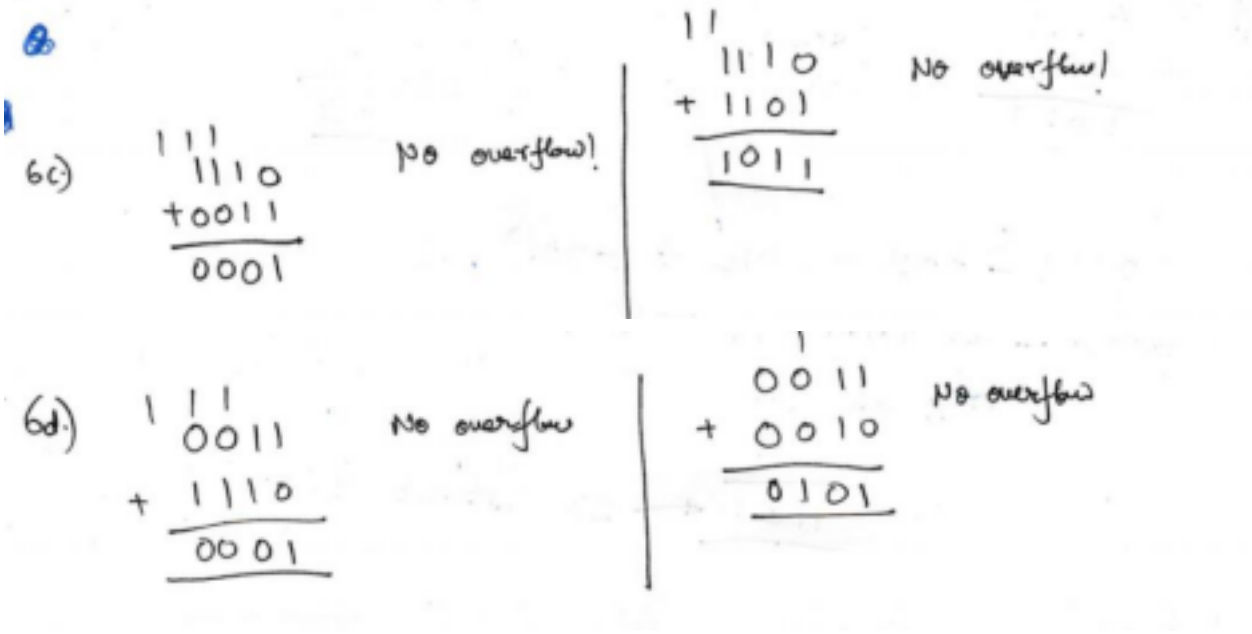
$1100 + 1 = 1101$ ← 2's Complement of 0011

```
  111
  0111
+ 1101
------
 0100
```
No overflow.

6b)
```
  11
  1110
+ 1101
------
 1011
```
No overflow!

$1101 \rightarrow 0011$ (2's Complement)

```
  111
  1110
+ 0011
------
 0001
```
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

So the boolean logic for overflow becomes = An'Bn'Sn + AnBnSn'

(Where A' means complement_A)

Q7.

7a)

```
  011
x 100
-----
  000
 000x
011xx
-----
01100
-----
```

7b)

```
  1010
x 0001
------
  1010
 0000x
0000xx
------
00101 0
------
```

7c)

```
  1001
x 0011
------
  1001
 1001x
-----
11011
-----
```

NOTE to the TAs: Solve only one of the parts if time is less, and explain that part fully.

Q8.

8a)

1001 × 1010

magnitude magnitude

```
  001
x 010
-----
  000
 001x
-----
 0010
-----
```

Sign: +ve ⇒ 0

∴ Ans: 010

8b)

```
    0 1 1
  x 1 1 1
  -------
  1 1 0 1 1
  0 1 1 x
0 1 1 x x
-----------
1 0 1 0 1
-----------
```

Sign : -ve ⇒ 1

∴ Ans: 110101

8c)

```
    1 0 1
  x 1 1 0
  -------
    0 0 0
  1 0 1 x
1 0 1 x x
---------
1 1 1 1 0
---------
```

Sign : +ve ⇒ 0

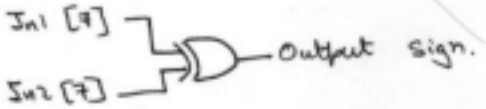
∴ Ans: 01110

Sign bit of output is:

Input 1	Input 2	Output
+ (0)	+ (0)	+ (0)
+ (0)	- (1)	- (1)
- (1)	+ (0)	- (1)
- (1)	- (1)	+ (0)

∴ Output sign = Input 1 Sign XOR Input 2 Sign.

∴



NOTE to the TAs: Difference is only in sign bit with 7th question. You can mention this and avoid solving question number 8 if time is limited.

Q9.

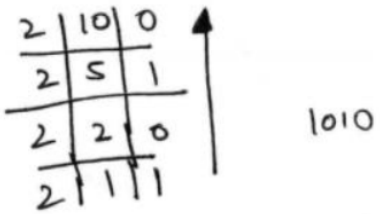
a) (i) **(ii)**

(ii) 19.25

2	19	1
2	9	1
2	4	0
2	2	0
2	1	1

10011

4)a) (i) 10.5



for 0.5 :

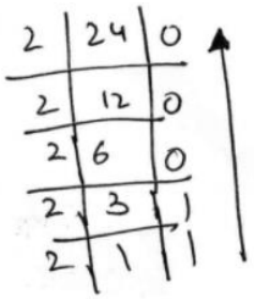
$0.5 \times 2 \Rightarrow 1.0$ | 10

$0 \times 2 \Rightarrow 0$

$\therefore$ 1010.10

(iii)

(iii) 24.6



$0.6 \times 2 \Rightarrow 1.2$

$0.2 \times 2 \Rightarrow 0.4$

$0.4 \times 2 \Rightarrow 0.8$

$0.8 \times 2 \Rightarrow 1.6$

$0.6 \times 2 \Rightarrow 1.2$

$= \overline{1001}$ (1001 1001 ...)

$\therefore$ 11000.$\overline{1001}$

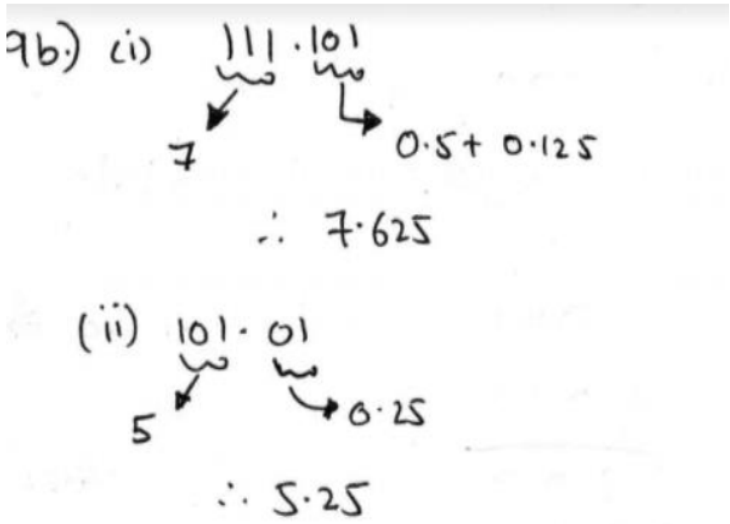
NOTE:

For part (a), (iii) subpart: We can see that we need an infinite number of bits to represent 24.6. Suppose we stop the representation after 1st repetition. We get 11000.1001. This is 24.5625. The error in this representation is 0.0375 (0.15%). To get a lower error rate, we can

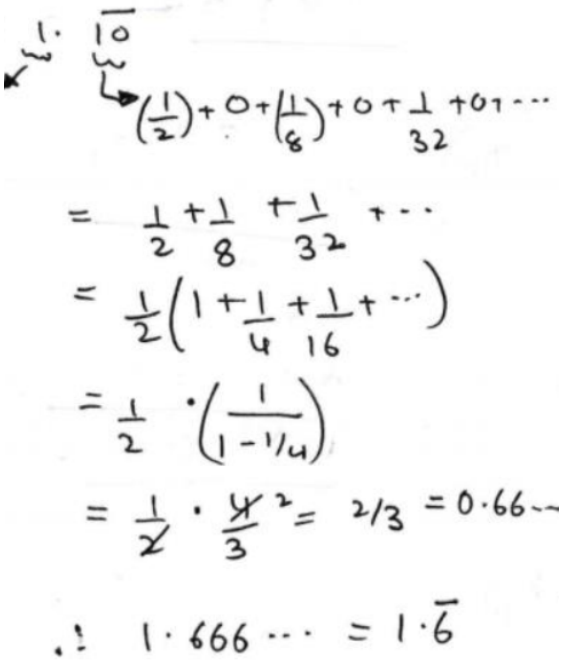
use more bits for the representation.

Note to TAs: The last explanation shows we require more bits if we want higher precision in fixed point representation. This arbitrary notation for the sake of precision can hinder general purpose computers. So people shifted towards standards like IEEE 754 single precision to get good enough precision for the same number of bits.

b.) (i) & (ii)



iii)



Q10

(a) Single Precision

-12.25_{10}
= 1100.01_2 (Convert into unsigned binary representation)
= **$1.10001_2 \times 2^3$** (Separate the mantissa and the exponent via normalization such that the binary starts with only one before the decimal)
Sign = -ve = **1_2**
Exponent = 3, Therefore 127 + 3 = 130 = **1000_0010_2** (The exponent is added to 127)
Mantissa = **$100\_0100\_0000\_0000\_0000\_0000_2$** (Mantissa is the value after the decimal followed by 0s to fill 23 bits)
IEEE Single Precision Representation is :
1 1000_0010 100_0100_0000_0000_0000_0000$_2$

HexaDecimal Representation is:
60A440000

(b) Double Precision

-12.25_{10}
= 1100.01_2 (Convert into unsigned binary representation)
= **1.10001_2** x 2^3 (Separate the mantissa and the exponent via normalization such that the binary starts with only one before the decimal)
Sign = -ve = **1_2**
Exponent = 3, Therefore 1023 + 3 = 1026 = **$100\_0000\_0010_2$** (The exponent is added to 1023)
Mantissa =
1000_1000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000$_2$
(Mantissa is the value after the decimal followed by 0s to fill 52 bits)

IEEE Double Precision Representation is :
1 10000000010 1000100000000000000000000000000000000000000000000000

HexaDecimal Representation is:
C028800000000000

THE END