# Question 1

Graphical user interface, text

Description automatically generated

How to detect overflow

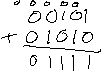
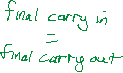
Overflow occurs only if we add two positive numbers and we get a negative number, or if we add two negative numbers and get a positive number. We could also think of it as the final carry in bit is different from the final carry out bit.

How to convert negative numbers to 2’s complement

We convert to 2’s complement by flipping all bits that appear on the left of the least significant 1.



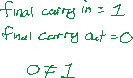
5+10=15



**No overflow has occurred**.



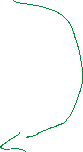
7+13=20



**Overflow has occurred because the addition of two positive numbers gave a negative number**.



11-14=-3



**No overflow has occurred**.



-10+(-13) = -23

But we only have 5 bits, so the actual result is

**Overflow has occurred because the addition of two negative numbers gave a positive number.**



5-10=-5

00101

+ 10110

11011

**No overflow has occurred**.



7-13=-6

00111

+ 10011

11010

**No overflow has occurred**.



-14-11=-25

10010

+ 10101

100111

Actual result =

**Overflow has occurred because subtracting a positive number from a negative number resulted in a positive number.**



-10-(-13) = 3

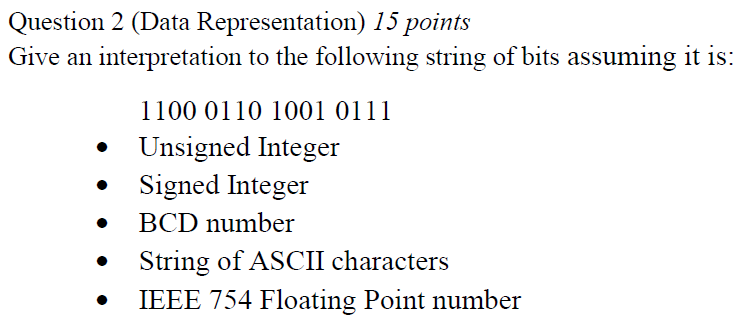
10110

+ 01101

00011

**No overflow has occurred. The final carry in bit and final carry out bit are both 1.**

# Question 2



## Unsigned Integer

Convert to Hex:

= $C

**$C697**

## Signed Integer

Two’s compliment:

1100 = 0100 = -4

**-$4697**

## BCD Number

BCD only goes up to decimal 9. So, any binary number after 1001 is invalid. Therefore, the first four bits (C) are invalid.

**NA 6 9 7**

## String of ASCII Characters

An ASCII character is 8 bits. So, we take $C6 and $97 as the two characters.

Maximum range for ASCII is $7F. Therefore, the two characters are not available as ASCII.

**NA NA**

## IEEE 754 Floating Point Number

S=1 = negative number

E’ = 1000 1101

E’=

E=141-127=14

M = 001 0111

Formula:

Answer:

# Question 3

Text

Description automatically generated

Byte organized memory means each memory location stores 8 bits (1 byte) of data.

Since we’re storing the numbers as words, each number uses 16 bits (2 bytes) of data. Therefore, we need to use two memory locations per number.

-14 in binary = 2’s complement of 14 in binary

14:

-14:

11:

11-14=-3

3:

-3:

|  |  |
| --- | --- |
| **Address** | **Value** |
| **$1000** | **$FF = 1111 1111** |
| **$1001** | **$F2 = 1111 0010** |
| **$1002** | **$00 = 0000 0000** |
| **$1003** | **$0B = 0000 1011** |
| **$1004** | **$FF = 1111 1111** |
| **$1005** | **$FD = 1111 1101** |

|  |  |
| --- | --- |
| **Address** | **Value** |
| **$1000** | **$FFF2** |
| **$1002** | **$000B** |
| **$1004** | **$FFFD** |

# Question 4

Text

Description automatically generated

n=16

k=22

Maximum capacity memory to be connected =

**memory storage**

# Question 5

Text, letter

Description automatically generated

1. Word organized means that each memory location will store 16 bits of data.

k=15

**We need 15 address lines from the CPU.**

12 of the bits go to the actual address lines for the memory modules.

15-12=**3 bits go to the decoder.**

We need 64 KB from multiple 4KB memory chips. 64/4=16 modules are needed.

We have 8 data pins from each module, but we need 16 data pins in total. We can have 2 memory modules that are parallelly read, each containing 16 modules/2 boxes=8 memory modules.

8 bits 8 bits



16 bits



12 bits

15 bits

3 bits

16 bits 8 bits 12 bits 16 bits 15 bits

module system