1.a.i)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

29 +27 +26 +24 +23 +22 +20 =512+128+64+16+8+4+1=733

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |

28 +27 +25 +24 +23 +21 +20 =256+128+32+16+8+2+1=443

ii)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

-29+27+26+24+23+22+20= -291

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |

28+27+25+24+23+21+20= 443

b. 1011011101 -291

+0110111011 +443

|  |
| --- |
| 1 |

0010011000 = 15210 152

Carryout not into MSB, without overflow, moreover the answer 152 matches with base 10 too. Thus the arithmetic worked. No the dump computer will not get it.

2.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 210 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

28 +26 +25 +21 +20 =355

00101100011} changing 1s and 0 s

11010011100

+ 1 → -210 +29 +27 +24 +23 +22 +20 =-355(Proved)

11010011101

3.a) 80090000

- 80090F04

F04

F\*163+\*161 +4\*160 [F=15]

=3884

3884/1024

So the space taken up is 3.793

b) 66210= 102A2 [Hex]

80090F04

+ 102A2

800A11A6 [the new memory location]

4.a) The short is signed 16- bit variable where the MSB (most significant bit) is negative or allocated for negative sign thus even adding two positive number it gives us negative number. There is an overflow as two positives is giving a negative number moreover there is a carry into but again not out of MSB thus the arithmetic is not correct.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 215 | 214 | 213 | 212 | 211 | 210 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

0111 1111 1111 1111

+ 1 → MSB is 1 thus the answer is negative

1000000000000000

b) The computer should “Flag set” this. So, the message displayed to the user should be meaningful, thus it should display a message saying some value is incorrect as the value is beyond the range

c) The ushort is un-signed 16- bit variable where the MSB (most significant bit) is positive or allocated for positive sign thus even adding two positive number it gives us positive number. The range for ushort is 0 to 65,535 so now the number falls in that range thus it’s working. There is no overflow and possible carry outs thus arithmetic is correct

d) In x86 there is processor instruction such as JO (jump if overflow flag set) and JC (Jump if carry flag set) and as we know that carries into and out of MSB can decide arithmetic right or wrong thus setting a flag will be easy and thus a loop can be carried whenever it is seen that both or neither carries into and out of the MSB or else exiting loop can in fact help us code in x86 to get correct arithmetic. For, this piece of code to work we need to add the number and check for overflow if there is any overflow we will use JO (jump if overflow flag set) and check it. If, the there is an overflow the computer should display a meaningful message to the user stating that some values are incorrect and to check the number or the range.