**Representation of numbers**

Digits in Base 10

1. Let every memory location has 6 digits. Every digit is between 0 and 9. Let abcdef is stored in some memory. There are two interpretations of it. First one is normal. Second means 0.0bcdefx10a. Let some memory has 314562. In second representation it means 0.014562x103=14.562. Alternate representations of 14.562 are 401456, 500145. However 314562 is compact representation (b0).
2. There are two methods for addition. The first is ordinary. However when final result is in 7 digits the first digit is ignored. 432567+312456=745023 621345+771234=392579.
3. Following procedure is used in second method: Let abcdef and pqrstu be two representations (ap). We write alternate representations with first digit as (a+1). Now the numbers formed with digits (2 to 6) are added. When a<p then numbers are exchanged. Example: 482314+257134=482880.

Reason: 508231+500057. 08231+00057=08288. Answer 508288 or 482880.

1. Example: Let 24.4352 and 7123.653 are stored using second representation. They are added using first representation. The final answer is 9567100? Reason: 24.4353 in second representation is 324435(24.4353=0.0244353 x 103). 324435+571236=895671. It is 0.095671x108=9567100.
2. Let 234314 is stored in second representation and 212456 is stored in first representation. They are added using first representation. What is the answer in the second representation? 234314 is 0.0234314 x 107=723431. 723431+212456=935887. It is 35887000.

Digits in base 2

1. Every memory has 8 bits. Every bit is 0 or 1. Let abcdefgh is stored in some memory. First interpretation is normal. In second it means 1d.efgh x2abc. Example: 01001101 mean 77 in first interpretation and 11.25 in second. [reason 10.1101 x 2010=10.1101 x 22=1011.01]
2. In the first method of addition is ordinary. When the result is in 9 bits then the first bit is ignored.
3. The second method of addition can be understood using following example: 01010010 + 10001101 = 10011001 [12.5+45=57.5] Reason: 01010010 is 010(10010)=1(1.0010)x 2010=11.0010 x2010. 11.0010 x2010 + 10.1101 x2100 = (.110010 + 10.1101)x2100=11.1001 x2100=10011001.
4. Let us take one more example: 00100010+01011001=10.0010 x 2001 + 11.1001 x 2010 = (1.00010 + 11.1001)x 2010=100.1010 x 2010 = 10.01010 x 2011= 01100101 [4.25+14.25=18.5]
5. Let 9.37 is stored in 2nd representation and 9.37+9.37 is done using first method of addition. The final answer is 42.96. Reason: Let 0.37 is 0.abc in binary. 9.37 is 1001.abc=10.01abc x 2010. In second representation it is 010001ab. Addition 010001ab with itself is 10001ab0. It is 10.1ab0 x 2100=101.ab x 2011=5.37 x 23 = 5.37 \* 8= 42.96. 10001ab0=100(01ab0)=1(0.1ab0)x2100

Access Exponent

1. Every memory has 10 bits. Let abcdefghij is stored in some memory. First interpretation is normal. In second it means 1e.fghij x2abcd-1000. Example: 1001001101 mean 589 in first interpretation and 4.8125 in second. [reason 10.01101 x 21001-1000=10.01101 x 21=100.1101] [0.1101=1/2+1/4+0/8+1/16=0.8125]

Example: 0110101000 means 0.8125 in second because 11.01000 x 20110-1000 = 0.1101

1. In the first method of addition is ordinary. When the result is in 11 bits the first bit is ignored.
2. The second method of addition can be understood using following example: 0101100101+1000011010=11.00101 x 20101-1000 + 10.11010 x 21000-1000 = (.01100 + 10.11010)x 21000-1000= 11.00110 x 21000-1000=1000100110. [0.375+2.8125=3.1875]

Sign

1. Let abcdefghij is stored in some memory. The first bit is the sign bit in both representations. ‘1’ is negative number and ‘0’ is positive number.
2. In the first representation when a=1 then magnitude of the number is obtained by complementing every bit and adding one. 1011010010 is –302 because 0100101101 is 301.
3. Addition is normal. 0010110110+1000010100=1011001010 (+182)+(-492)=-310.
4. When the result is in 11 bits then the first bit is ignored. 1100000100+1100000000=11000000100. After ignoring first bit we get 1000000100. (-252)+(-256)=-508.
5. We get the wrong answer when final result is not within -512..+511.
6. In the second representation abcdefghij is 1f.ghij x2bcde-1000. a=0Sign positive negative(a=1).
7. The first representation is used for integers and the second for float.

int p,q,r,\*x,\*y,\*z; float a,b,c,d,\*t; a=9.37; b=9.37; c=2.0;

x=&a; y=&b; z=&c; p=(\*x)+(\*y)+(\*z); t=&p; d=\*t; printf(“%f”,d); output is -42.96

9.37 is 1001.pqr=10.01pqr x 22 = 10.01pqr x 20010 = 10.01pqr x 21010-1000 = 01010001pq

2 is 10.00= 10.00 x 20 = 10.00 x 21000-1000=0100000000

9.37+9.37+2=1110001pq0= 10.1pq0 x 21100-1000= 101.pq0 x 2011= 5.37\*8= 42.96

|  |
| --- |
| Question 1: Write output (A) a=3.13 b=3.13 (B) a=9.29 b=3 (C) a=13.41 b=13.41 |

Total size 32 bits

1. In both representations first bit is the sign bit. In case of negative integers the absolute value is obtained by complement+1.
2. Let us see the representation of +12. 00000000 00000000 00000000 00001100
3. -12 11111111 11111111 11111111 11110100
4. +256\*47 00000000 00000000 00101111 00000000
5. +(2562\*6+256\*17+9)=397577 00000000 00000110 00010001 00001001
6. -(2562\*6+256\*17+9)=-397577 11111111 11111001 11101110 11110111
7. In the representation of float first bit is the sign bit. Next 8 bits are exponent in access 128. Remaining 23 bits are mantissa (except 1st bit, dot is put after second bit) .
8. +9.25 is 1001.01=10.0101x22. Exponent=2. access128 is 130 i.e. 10000010 mantissa is 00101.

Sign 0 representation 0**1000001 0**0010100 00000000 00000000 exponent is bold.

1. -9.25 is 11000001 00010100 00000000 00000000 same as 9.25 except first bit.
2. 0.75/2=0.011=11.00x2-3. exponent-3. Access128=125=0**0111110 1**1000000 00000000 00000000
3. +(2562\*6+256\*17+9)=110 00010001 00001001=11.00001000100001001 x 217=

0**1001000 1**1000010 00100001 00100000

Float seeeeeee emmmmmmm mmmmmmmm mmmmmmmm s:sign e:exponent m:mantissa

|  |
| --- |
| 1. Question 2: What is the output of following?   float a,\*t; int q; q=256\*256\*256\*194+256\*256\*82; t=&q; a=\*t; printf(“%f”,a);  Example: q= 256\*256\*256\*66+256\*256\*17+256\*64 gives 36.3125  q=01000010 00010001 01000000 00000000 a=10.01000101…x 210000100-10000000=100100.0101  =100100.0101=36.3125 (0.0101 is 0/2+1/22 +0/23 + 1/24)=0.25+0.0625=0.3125 |

|  |  |
| --- | --- |
| 1. Question 3: What is the output of the given program?   Example: a=12.625 and b=a/64 gives 62 74  a=12.625=1100.101=11.00101 x 22 b=11.00101 x 2-4.  b=00111110 01001010 00000000 00000000=q  p=00111110 01001010 x=00111110=62 y=01001010=74 | float a,b;int \*k,p,q,x,y;  a=21.5; b=a/32; k=&b;  q=\*k; p=q/256/256;  x=p/256; y=p%256;  printf(“%d %d”,x,y); |

|  |  |
| --- | --- |
| W Question 4: What is the output of the given program?  Example:a=9.29;q=256\*256\*64; o/p13.29. Reason9.29= 1001.abc  a= 01000001 0001abc0 00000000 00000000 = \*x  q= 00000000 01000000 00000000 00000000  p= 01000001 0101abc0 00000000 00000000=11.01abc x 22 | float a,\*t,d;int q,\*x,p;  a=514.41; q=256\*256\*43;  x=&a; p=(\*x)+q;  t=&p; d=\*t;printf(“%f”,d); |
| Question 5: What is the output of given program?  Example: a=62+256\*256\*128; b=80+256\*256\*128; gives 71.  int a,b,c,\*d; float \*p,\*q,r,s,t; a=58+256\*256\*128; b=86+256\*256\*256;  p=&a; q=&b; r=\*p; s=\*q; t=r+s; d=&t; c=\*d; printf(“%d”,c%256);  a=00000000 10000000 00000000 00111110 = 10.0000000000000000111110 x 2-127 =r  b=00000000 10000000 00000000 01010000 = 10.0000000000000001010000 x 2-127 =s  r+s = 100.0000000000000010001110 x 2-127= 10.00000000000000010001110 x 2-126  c=00000001 00000000 00000000 01000111 = mod 256 is 01000111 = 71 | |

During rounding when last digit is less than 5 then increase. When more than 5 then unchanged. When equal to 5 then second last digit is seen. It is increased when odd.

Let 5 significant digits: 34.285+100=134.28 34.275+100=134.85 during rounding when last digit is 5 Binary let 5 significant bits: 1.0011+100=101.01 1.0001+100=101.00

10.101+100=110.10 10.011+100=110.10 1.0010+100=101.00 1.0110+100=101.10

1. What is missing in following? Use smallest magnitude integer in numerator. The output of the program is shown for various values of ‘k’. Float has 24 significant bits. ES19RF

int k; scanf(“%d”,&k); double x=missing; float y=x+k; if(y>x+k) printf(“A”); else printf(“B”);

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | 0 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
| x=missing | B | A | B | A | A | B | A | B | A | A | B | A | B | A | A |
| x=34/63.0 | A | B | A | B | B | B | A | B | A | B | B | B | A | B | A |

1. Representations of int 2563\*64 and float 2.0 are same. Similarly 2563\*66+2562\*96 and 56.0 are same. State 2563\*68+2562\*85 is similar to which float. MS18RF
2. float a,b,c; int x,y; a=5+1/3.0; x=22; b=a+x+y; c=a+(x+y); For which values of y b>c. MS16RF

Example: a=10+1/3.0; x=17 when 37y100 b<c x=6 to 53 y=54-x to 117-x [24 bits float]

1. In round-off operation when last digit is 5 then 1 is added with probability ‘p’. What is the probability that x+1+10+100+1000+10000 will be 11111.3? Find it for x=0.255445. Assume 6 significant digits.

Example: For x=0.244445 it is p5. For x=0.255555 it is 1-(1p)5. ES15RF

For x=0.254545 it is p2(1-(1-p)2)+(1-p2)(p2+p(1-p2))=p+p2-2p4+p5.