1. **Assume a floating-point normalized representation similar to IEEE 754 representation. This representation has 1 bit for sign, 4 bits for exponent with bias 8 and 3 bits for mantissa. Compute the largest positive floating point number that can be represented with this format.**

Ans:

0 1110 111

Mantissa/significand value: 1.0 (implicit due to normalization) + 0.5+0.25+0.125 = 1.875

Exponent: (14- 8 for bias) = 6.

Largest positive number: 1.875 x 2^6

2. **In the IEEE 754 format with 32 bits, show the binary representation for -2178.375**

Ans:

Binary representation of 2178.375 = 100010000010.011

Normalized representation: 1.00010000010011 x 2^{11}

Sign bit: 1 (for negative)

Exponent (8 bits): (11+127) = 138 = 10001010

Mantissa (23 bits with leading 1 as implicit due to normalization): 00010000010011000000000

Final binary representation: 1 100 01010 00010000010011000000000

3. **Consider a 8-way set associative cache of size 64k bytes, block size of 32 bytes, and memory address of 32 bits. Find the total number of index bits required.**

Ans:

Cache size – 64 K Bytes.

Block size – 32 bytes

Number of blocks or number of lines in the cache – 64 K / 32 = 2K

Each set has 8 lines since it is 8-way set associative. Thus, total number of sets = 2K / 8 = 256

Number of index bits = 8

4. **Consider a 16-way set associative cache of size 256K bytes, block size of 32 bytes, and memory address of 64 bits. Find the number of tag bits required.**

Ans:

Cache size – 256 K Bytes.

Block size – 32 bytes. Hence number of offset bits = 5

Number of blocks or number of lines in the cache – 256 K / 32 = 8K

Each set has 16 lines since it is 16-way set associative. Thus, total number of sets = 8K / 16 = 512

Number of index bits = 9.

Thus, number of tag bits = 64 – (index+offset bits) = 64 – (5+9) = 50 bits.

5. **Consider a matrix-matrix multiplication program written in C in which the matrices are of type double, where the size of a double is considered to be 8 bytes. The program multiplies a 4x4 matrix A with a 4x4 matrix B to produce a 4x4 matrix C. Matrices A, B and C are stored in memory one after the other, and matrix A starts at address 0. Let the size of a block be 32 bytes and the size of memory address be 16 bits. Consider a 2-way set associative cache of size 256 bytes.**

   Find the total number of cache misses for the **kij variant** of matrix-matrix multiplication for the entire program.

Ans:

```
double A[4][4], B[4][4], C[4][4];

for(k=0; k<4; k++){
  for(i=0; i<4; i++){
    tmp = A[i][k];
    for(j=0; j<4; j++){
      C[i][j] += tmp*B[k][j];
    }
  }
}
```

Block size – 32. Size of double – 8. Hence, a block can accommodate 4 elements. Number of offset bits = 5
Number of blocks or number of lines = Size of cache / Block size = 256/32 = 8
Each set has 2 lines since the cache is 2-way set associative. Thus, number of sets = 8/2 = 4.
Number of index bits = 2
Number of tag bits = 16 – (5+2) = 9

Element	Address Address number - **tagindexoffset**	Set	Access Sequence	Cache hit or miss
A[0][0]	0 - 0000000000000000	0	1	M
A[0][1]	8 - 0000000000001000	0	53	M
A[0][2]	16 - 0000000000010000	0	105	H
A[0][3]	24 - 0000000000011000	0	157	H
A[1][0]	32 - 0000000000100000	1	14	M
A[1][1]	40 - 0000000000101000	1	66	M
A[1][2]	48 - 0000000000110000	1	118	M
A[1][3]	56 - 0000000000111000	1	170	H
A[2][0]	64 - 0000000001000000	2	27	M
A[2][1]	72 - 0000000001001000	2	79	H
A[2][2]	80 - 0000000001010000	2	131	M
A[2][3]	88 - 0000000001011000	2	183	M
A[3][0]	96 - 0000000001100000	3	40	M
A[3][1]	104 - 0000000001101000	3	92	H
A[3][2]	112 - 0000000001110000	3	144	H,
A[3][3]	120 - 0000000001111000	3	196	M
B[0][0]	128 - 0000000010000000	0	2,15,28,41	M,H,H,H
B[0][1]	136 - 0000000010001000	0	5,18,31,44	H,H,H,H
B[0][2]	144 - 0000000010010000	0	8,21,34,47	H,H,H,H
B[0][3]	152 - 0000000010011000	0	11,24,37,50	H,H,H,H
B[1][0]	160 - 0000000010100000	1	54,67,80,93	M,H,H,H
B[1][1]	168 - 0000000010101000	1	57,70,83,96	H,M,H,H
B[1][2]	176 - 0000000010110000	1	60,73,86,99	H,H,H,H
B[1][3]	184 - 0000000010111000	1	63,76,89,102	H,H,H,H
B[2][0]	192 - 0000000011000000	2	106,119,132,145	M,H,H,H
B[2][1]	200 - 0000000011001000	2	109,122,135,148	H,H,M,H
B[2][2]	208 - 0000000011010000	2	112,125,138,151	H,H,H,H
B[2][3]	216 - 0000000011011000	2	115,128,141,154	H,H,H,H
B[3][0]	224 - 0000000011100000	3	158,171,184,197	M,H,H,H
B[3][1]	232- 0000000011101000	3	161,174,187,200	H,H,H,M
B[3][2]	240 - 0000000011110000	3	164,177,190,203	H,H,H,H
B[3][3]	248 - 0000000011111000	3	167,180,193,206	H,H,H,H
C[0][0]	256 - 0000000100000000	0	3 (ld),4 (st),55,56,107,108,159,160	M,H,H,H,H,H,H,H
C[0][1]	264 - 0000000100001000	0	6,7,58,59,110,111,162,163	H,H,H,H,H,H,H,H
C[0][2]	272 - 0000000100010000	0	9,10,61,62,113,114,165,166	H,H,H,H,H,H,H,H
C[0][3]	280 - 0000000100011000	0	12,13,64,65,116,117,168,169	H,H,H,H,H,H,H,H
C[1][0]	288 - 0000000100100000	1	16,17,68,69,120,121,172,173	M,H,M,H,M,H,H,H
C[1][1]	296 - 0000000100101000	1	19,20,71,72,123,124,175,176	H,H,H,H,H,H,H,H
C[1][2]	304 - 0000000100110000	1	22,23,74,75,126,127,178,179	H,H,H,H,H,H,H,H
C[1][3]	312 - 0000000100111000	1	25,26,77,78,129,130,181,182	H,H,H,H,H,H,H,H
C[2][0]	320 - 0000000101000000	2	29,30,81,82,133,134,185,186	M,H,H,H,M,H,H,H
C[2][1]	328 - 0000000101001000	2	32,33,84,85,136,137,188,189	H,H,H,H,H,H,H,H
C[2][2]	336 - 0000000101010000	2	35,36,87,88,139,140,191,192	H,H,H,H,H,H,H,H
C[2][3]	344 - 0000000101011000	2	38,39,90,91,142,143,194,195	H,H,H,H,H,H,H,H

C[3][0]	352 - 0000000101100000	3	42,43,94,95,146,147,198,199	M,H,H,H,H,H,H,H
C[3][1]	360- 0000000101101000	3	45,46,97,98,149,150,201,202	H,H,H,H,H,H,H,H
C[3][2]	368 - 0000000101110000	3	48,49,100,101,152,153,204,205	H,H,H,H,H,H,H,H
C[3][3]	372 - 0000000101111000	3	51,52,103,104,155,156,207,208	H,H,H,H,H,H,H,H

Set 0	Line 1	After access 1 A[0][0]- A[0][3]	After access 3 C[0][0]- C[0][3]			
	Line 2	After access 2 B[0][0]- B[0][3]	After access 53 A[0][0]- A[0][3]			
Set 1	Line 1	After access 14 A[1][0]- A[1][3]	After access 54 B[1][0]- B[1][3]	After access 68 C[1][0]- C[1][3]	After access 117 A[1][0]- A[1][3]	
	Line 2	After access 16 C[1][0]- C[1][3]	After access 66 A[1][0]- A[1][3]	After access 70 B[1][0]- B[1][3]	After access 120 C[1][0]- C[1][3]	
Set 2	Line 1	After access 27 A[2][0]- A[2][3]	After access 106 B[2][0]- B[2][3]	After access 133 C[2][0]- C[2][3]		
	Line 2	After access 29 C[2][0]- C[2][3]	After access 131 A[2][0]- A[2][3]	After access 106 B[2][0]- B[2][3]	After access 183 A[2][0]- A[2][3]	
Set 3	Line 1	After access 40 A[3][0]- A[3][3]	After access 158 B[3][0]- B[3][3]	After access 198 C[3][0]- C[3][3]		
	Line 2	After access 42 C[3][0]- C[3][3]	After access 196 A[3][0]- A[3][3]	After access 200 B[3][0]- B[3][3]		

k=0, i=0, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (1M), 4 load accesses to C (1 M), 4 store accesses to C (0 M)
k=0, i=1, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (0M), 4 load accesses to C (1 M), 4 store accesses to C (0 M) – since B[0,0]-B[3,3] still in memory.
k=0, i=2, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (0M), 4 load accesses to C (1 M), 4 store accesses to C (0 M)
k=0, i=3, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (0M), 4 load accesses to C (1 M), 4 store accesses to C (0 M)
Total accesses – 52, A misses – 4, B misses – 1, C misses – 4. Total misses – 9/52

k=1, i=0, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (1M), 4 load accesses to C (0 M), 4 store accesses to C (0 M)
k=1, i=1, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (1M), 4 load accesses to C (1 M), 4 store accesses to C (0 M)
k=1, i=2, j=0,1,2,3 - 1 load access to A (0M), 4 load accesses to B (0M), 4 load accesses to C (0 M), 4 store accesses to C (0 M)
k=1, i=3, j=0,1,2,3 - 1 load access to A (0M), 4 load accesses to B (0M), 4 load accesses to C (0 M), 4 store accesses to C (0 M)
Total accesses – 52, A misses – 2, B misses – 2, C misses – 1. Total misses – 5/52

k=2, i=0, j=0,1,2,3 - 1 load access to A (0M), 4 load accesses to B (1M), 4 load accesses to C (0 M), 4 store accesses to C (0 M)
k=2, i=1, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (0M), 4 load accesses to C (1 M), 4 store accesses to C (0 M)
k=2, i=2, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (1M), 4 load accesses to C (1 M), 4 store accesses to C (0 M)
k=2, i=3, j=0,1,2,3 - 1 load access to A (0M), 4 load accesses to B (0M), 4 load accesses to C (0 M), 4 store accesses to C (0 M)
Total accesses – 52, A misses – 2, B misses – 2, C misses – 2. Total misses – 6/52

k=3, i=0, j=0,1,2,3 - 1 load access to A (0M), 4 load accesses to B (1M), 4 load accesses to C (0 M), 4 store accesses to C (0 M)
k=3, i=1, j=0,1,2,3 - 1 load access to A (0M), 4 load accesses to B (0M), 4 load accesses to C (0 M), 4 store accesses to C (0 M)
k=3, i=2, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (0M), 4 load accesses to C (0 M), 4 store accesses to C (0 M)
k=3, i=3, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (1M), 4 load accesses to C (1 M), 4 store accesses to C (0 M)
Total accesses – 52, A misses – 2, B misses – 2, C misses – 1. Total misses – 5/52

Total misses – 25/208

Full points will be given if the first few misses are obtained correctly and this is used to extrapolate to the full set of accesses.

6. **For the same matrix-matrix multiplication and the same configuration of memory addresses, cache configuration and other parameters in the above question, imagine that memory addresses are split into (index, tag, offset) instead of the (tag, index, offset) organization we have seen in the class. Find the cache hit ratio. Explain your answer.**

Element	**Address** **Address number -** **indextagoffset**	**Set**	**Access Sequence**	**Cache hit or miss**
A[0][0]	0 - 0000000000000000	0	1	M
A[0][1]	8 - 0000000000001000	0		
A[0][2]	16 - 0000000000010000	0		
A[0][3]	24 - 0000000000011000	0		
A[1][0]	32 - 0000000000100000	0	14	M
A[1][1]	40 - 0000000000101000	0		
A[1][2]	48 - 0000000000110000	0		
A[1][3]	56 - 0000000000111000	0		
A[2][0]	64 - 0000000001000000	0	27	M
A[2][1]	72 - 0000000001001000	0		
A[2][2]	80 - 0000000001010000	0		
A[2][3]	88 - 0000000001011000	0		
A[3][0]	96 - 0000000001100000	0		
A[3][1]	104 - 0000000001101000	0		
A[3][2]	112 - 0000000001110000	0		
A[3][3]	120 - 0000000001111000	0		
B[0][0]	128 - 0000000010000000	0	2,15,28	M,M,M
B[0][1]	136 - 0000000010001000	0	5,18,31	H,H,H
B[0][2]	144 - 0000000010010000	0	8,21,34	H,H,H
B[0][3]	152 - 0000000010011000	0	11,24,37	H,H,H
B[1][0]	160 - 0000000010100000	0		
B[1][1]	168 - 0000000010101000	0		
B[1][2]	176 - 0000000010110000	0		
B[1][3]	184 - 0000000010111000	0		
B[2][0]	192 - 0000000011000000	0		
B[2][1]	200 - 0000000011001000	0		
B[2][2]	208 - 0000000011010000	0		
B[2][3]	216 - 0000000011011000	0		
B[3][0]	224 - 0000000011100000	0		
B[3][1]	232- 0000000011101000	0		

B[3][2]	240 - 0000000011110000	0		
B[3][3]	248 - 0000000011111000	0		
C[0][0]	256 - 0000000100000000	0	3,4	M,H
C[0][1]	264 - 0000000100001000	0	6,7	H,H
C[0][2]	272 - 0000000100010000	0	9,10	H,H
C[0][3]	280 - 0000000100011000	0	12,13	H,H
C[1][0]	288 - 0000000100100000	0	16,17	M,H
C[1][1]	296 - 0000000100101000	0	19,20	H,H
C[1][2]	304 - 0000000100110000	0	22,21	H,H
C[1][3]	312 - 0000000100111000	0	25,26	H,H
C[2][0]	320 - 0000000101000000	0	29,30	M,H
C[2][1]	328 - 0000000101001000	0	32,33	H,H
C[2][2]	336 - 0000000101010000	0	35,36	H,H
C[2][3]	344 - 0000000101011000	0	38,39	H,H
C[3][0]	352 - 0000000101100000	0		
C[3][1]	360- 0000000101101000	0		
C[3][2]	368 - 0000000101110000	0		
C[3][3]	372 - 0000000101111000	0		

Set 0	Line 1	After access 1 A[0][0]-A[0][3]	After access 3 C[0][0]-C[0][3]	After access 15 B[0][0]-B[0][3]	After access 27 A[2][0]-A[2][3]	After access 29 C[2][0]-C[2][3]
	Line 2	After access 2 B[0][0]-B[0][3]	After access 14 A[1][0]-A[1][3]	After access 16 C[1][0]-C[1][3]	After access 28 B[0][0]-B[0][3]	
Set 1	Line 1					
	Line 2					
Set 2	Line 1					
	Line 2					
Set 3	Line 1					
	Line 2					

k=0, i=0, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (1M), 4 load accesses to C (1 M), 4 store accesses to C (0 M)
k=0, i=1, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (1M), 4 load accesses to C (1 M), 4 store accesses to C (0 M) – since B[0,0]-B[3,3] still in memory.
k=0, i=2, j=0,1,2,3 - 1 load access to A (1M), 4 load accesses to B (1M), 4 load accesses to C (1 M), 4 store accesses to C (0 M)
It is clear that for each k,i pair, there will be 1 miss each for load A, B and C since all addresses map to set 0.
Thus, total misses = (16x3)/(16x13) = 48 misses out of a total of 208 accesses.
Hence, total hits = 160/208. Cache hit ratio = 76.9%