

assume

Bits Per Track

bpt = x \* r \* K

Track Count

tc = (1-x) \* r \* M

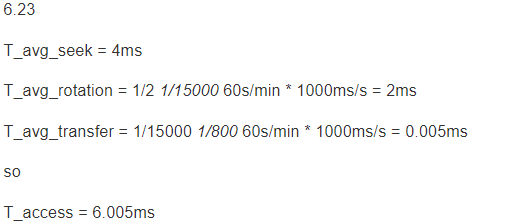
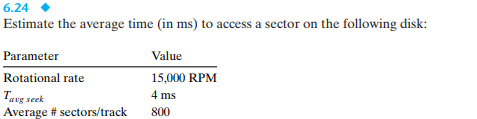
M, K are constant

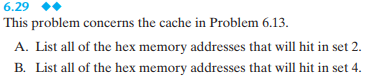
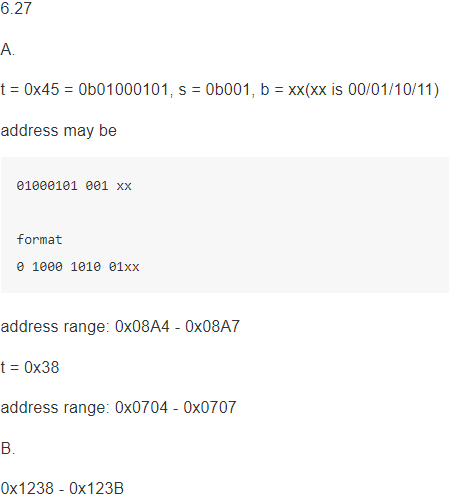
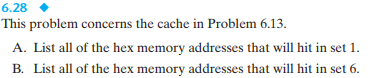
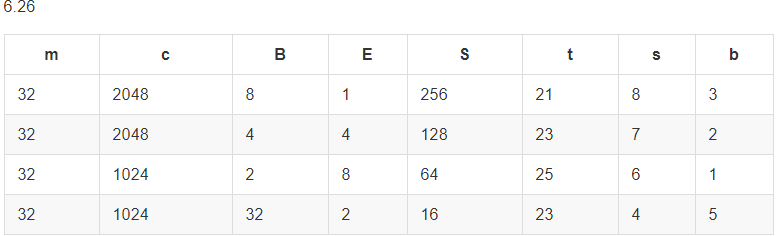
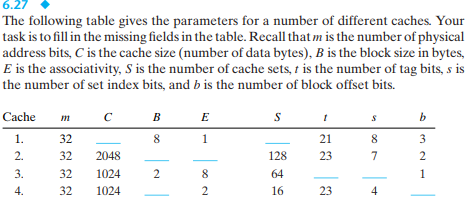
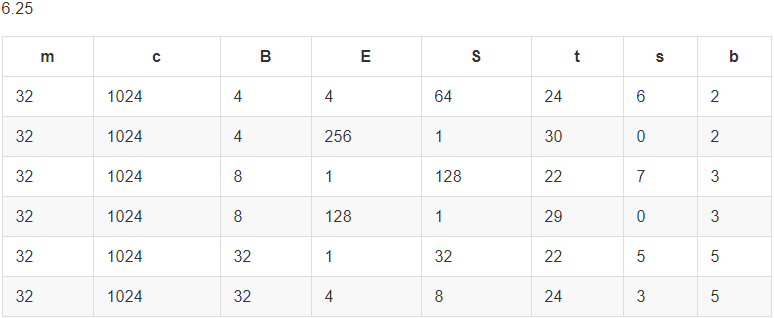
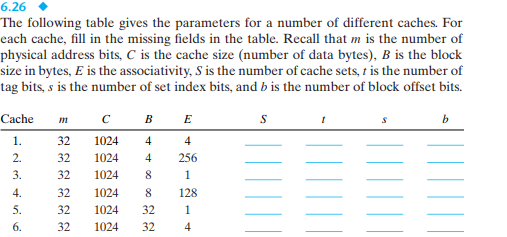
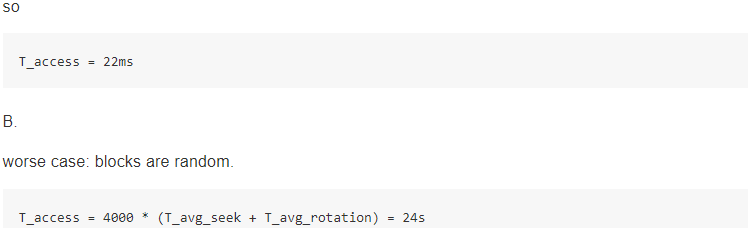
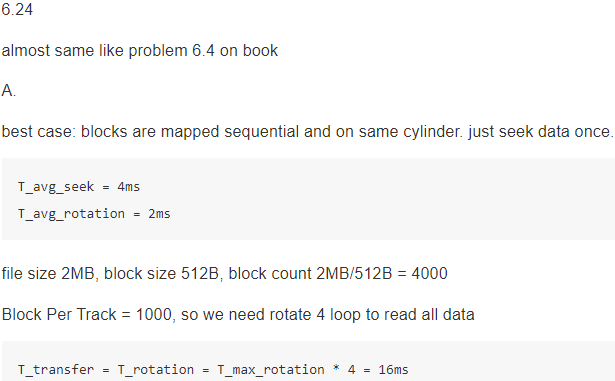
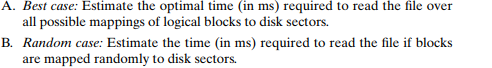
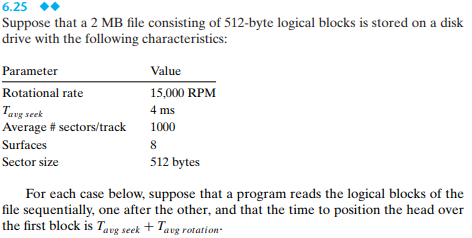
so

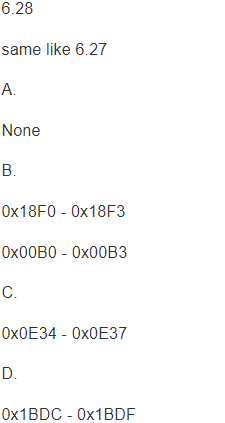
Bit Count

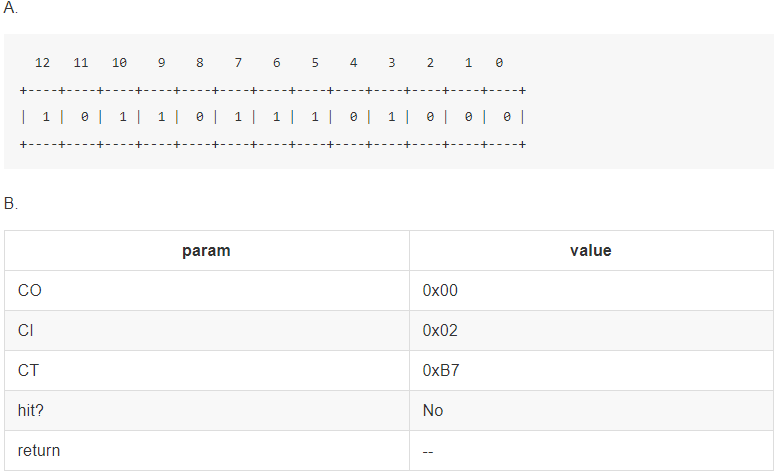
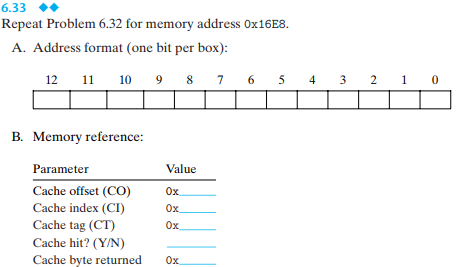
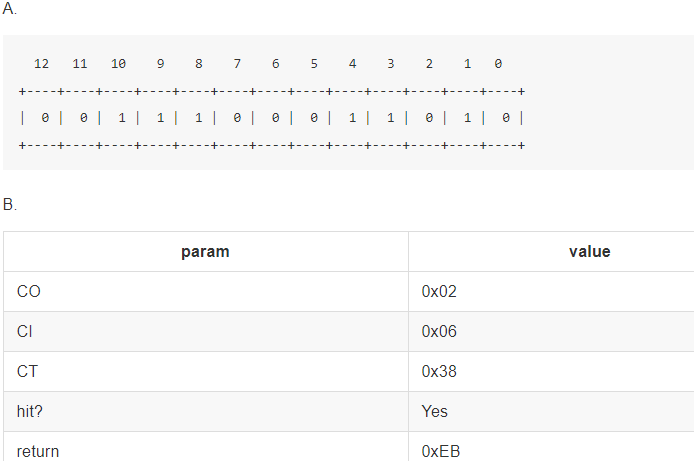
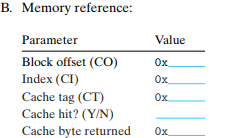
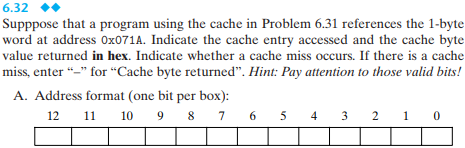
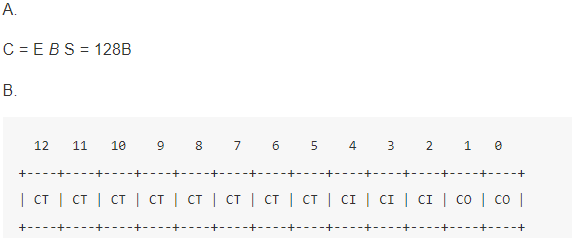
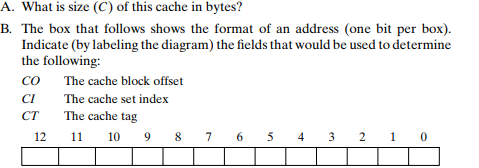
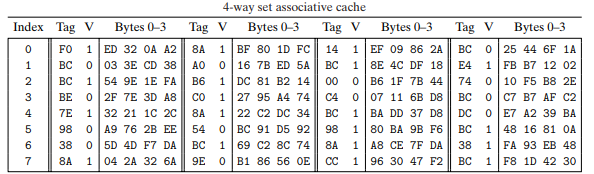
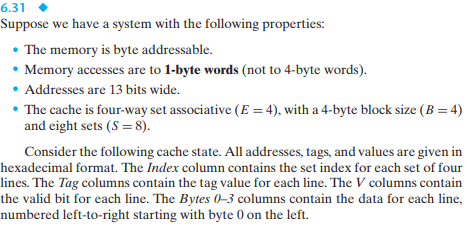
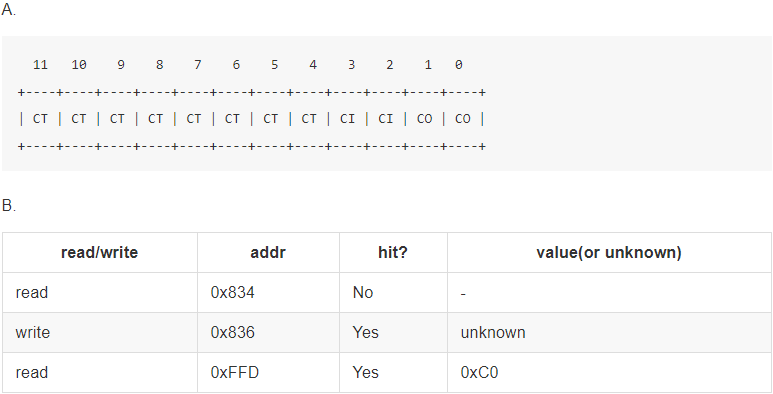
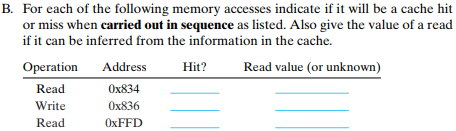
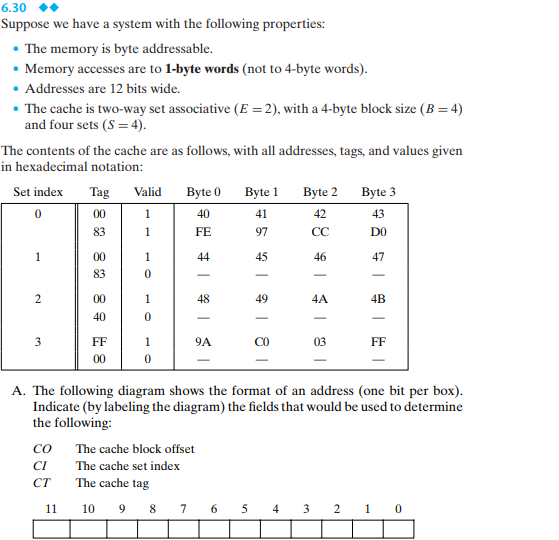
bc = K \* M \* r^2 \* (1-x) \* x

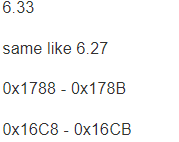
when x == 1/2, bc is maximum

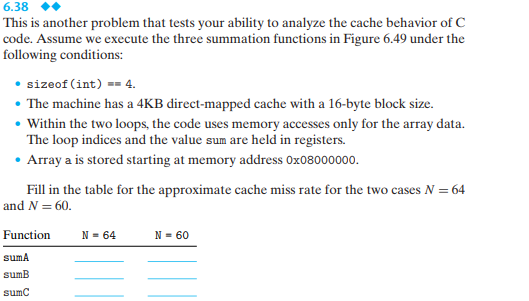
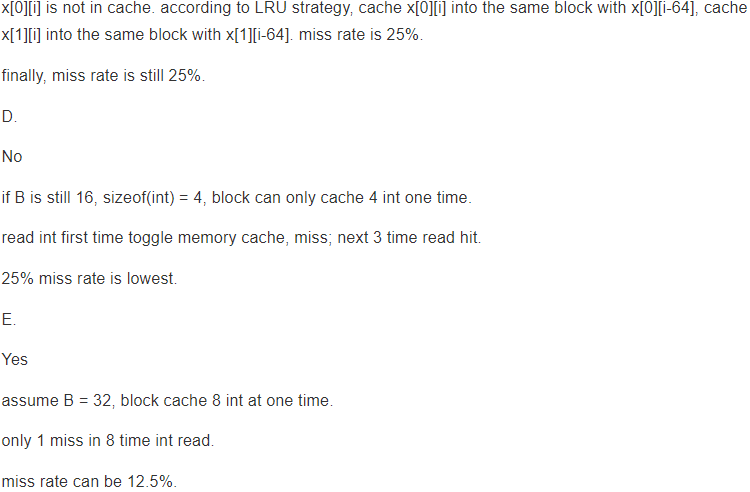
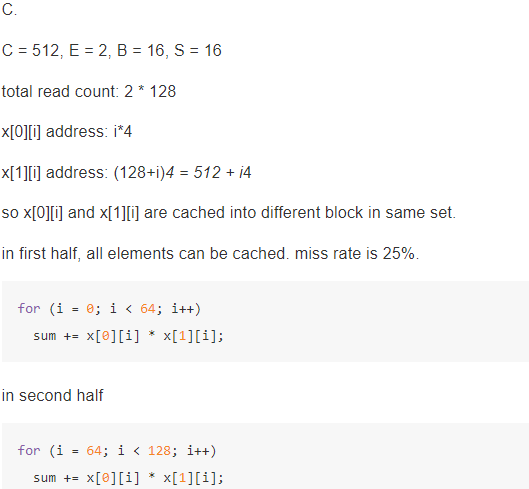
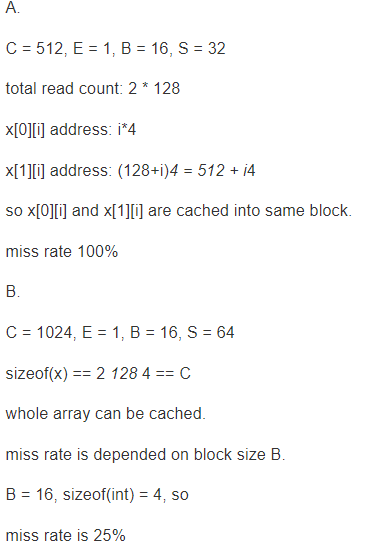
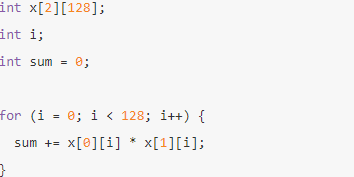
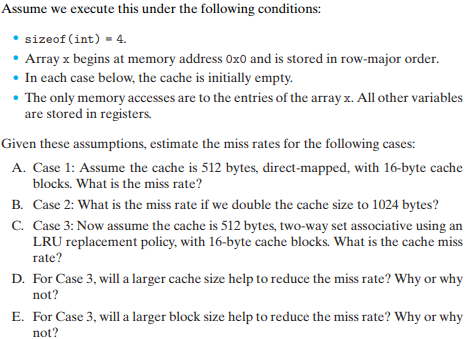
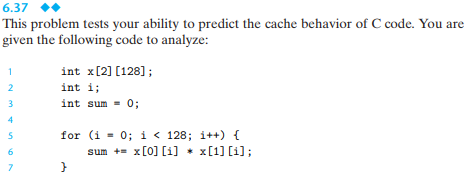
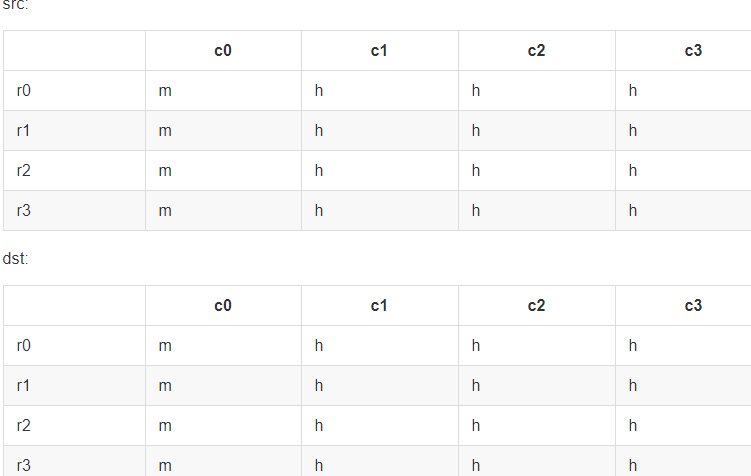
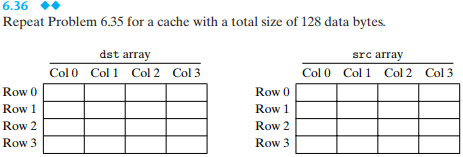
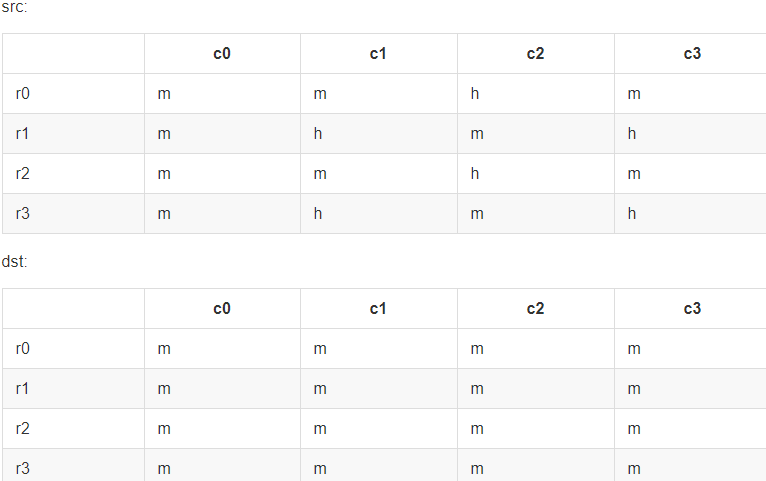
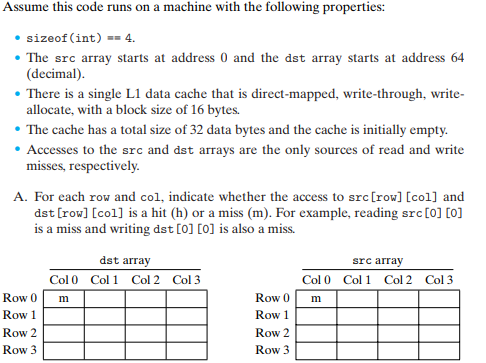
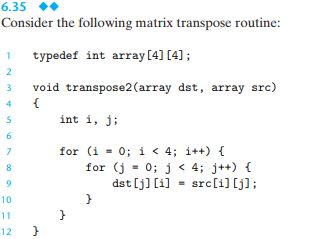












typedef int array\_t[N][N];

int sumA(array\_t a) {

int i, j;

int sum = 0;

for (i = 0; i < N; i++)

for (j = 0; j < N; j++)

sum += a[i][j];

return sum;

}

int sumB(array\_t a) {

int i, j;

int sum = 0;

for (i = 0; i < N; i++)

for (j = 0; j < N; j++)

sum += a[j][i];

return sum;

}

int sumC(array\_t a) {

int i, j;

int sum = 0;

for (i = 0; i < N; i+=2)

for (j = 0; j < N; j+=2)

sum += (a[j][i] + a[j][i+1] + a[j+1][i] + a[j+1][i+1])

}

C = 4096, B = 16, E = 1, S = 256

**N = 64**

sizeof(array\_t) == 64 *64 == 4096 == 4*C

memory-cache graph

memory address start from 0 and end to 4096\*4.

cell size is 16B. number(0-255) in cell means cache block number that the cell

will be cached.

0 +---------+

| 0 |

16 +---------+

| 1 |

32 +---------+

| 2 |

48 +---------+

| . |

| . |

| . |

| . |

| . |

4096-16 +---------+

| 255 |

4096 +---------+

| 0 |

4096+16 +---------+

| 1 |

4096+32 +---------+

| . |

| . |

| . |

| . |

| . |

| . |

4096\*4-16+---------+

| 255 |

4096\*4 +---------+

A. sumA

sum += a[i][j];

read memory address order:

0, 4, 8, 12, ....., 4096\*4-4

read cache order:

0, 0, 0, 0, 1, 1, 1, 1,.....255, 255, 255, 255, 0, 0, 0, 0,.....255, 255, 255, 255

first cache read miss, next 3 time read hit.

miss rate: 25%

B. sumB

sum += a[j][i];

read memory address order:

0, 64*4, 64*4*2, .... 64*4*63, 4, 64*4+4, 64*4*2+4, .... 4096\*4-4

read cache order:

0, 16, 32, 48, ... 240,(4 times) 1, 17, 33, ... 241,(4 times) 15, 31, 47, ... 255(4 times)

let's see first read loop:

read cache order loop 4 times

0, 16, 32, 48, ... 240,(4 times)

first loop all miss, next 3 loop all hit

so miss rate is 25%.

C. sumC

for (i = 0; i < N; i+=2)

for (j = 0; j < N; j+=2)

sum += (a[j][i] + a[j][i+1] + a[j+1][i] + a[j+1][i+1]);

easy to see that read a[j][i+1] and a[j+1][i+1] always hit

same like

for (i = 0; i < N; i+=2)

for (j = 0; j < N; j+=2)

sum += (a[j][i] + a[j+1][i]);

same like

for (i = 0; i < N; i+=2)

for (j = 0; j < N; j++)

sum += a[j][i];

total read count = 64\*64

because of i+=2,

read cache order only loop 2 times

0, 16, 32, 48, ... 240,(2 times)

so miss rate is 50%

totol read miss count = 64/2 *64*50% = 64\*64/4

so miss rate is still 25%.

**N = 60**

A. sumA

sum += a[i][j];

read memory by step 1

miss rate 25%

B. sumB

for (i = 0; i < N; i++)

for (j = 0; j < N; j++)

sum += a[j][i];

it's interesting.

let's see first inner loop a[0][0] -> a[59][0]

read memory address order:

0, 60*4, 60*4*2, .... 60*4\*59

read cache order:

0, 15, 30, ...., 225, (17 numbers) 255, 14, 29, ....., 224, (17 numbers) 254, 13, 28, ....., 223, (17 numbers) 253, 12, 27, 42, 57, 72, 87, 102, 117 (9 numbers)

all read miss and store into different blocks

next 3 loops: a[0][1] -> a[59][1], a[0][2] -> a[59][2], a[0][3] -> a[59][3]

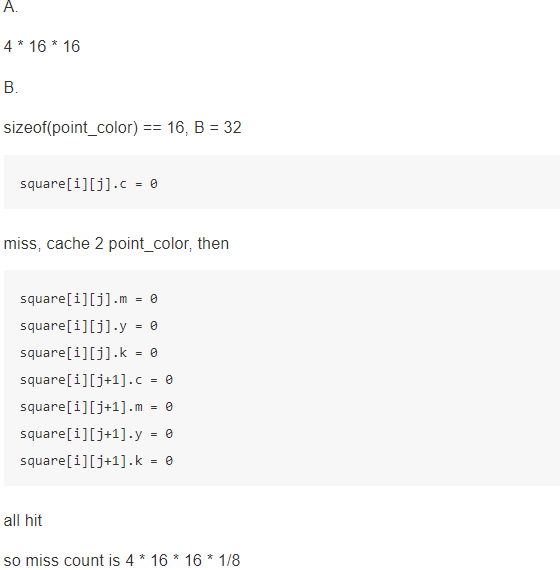
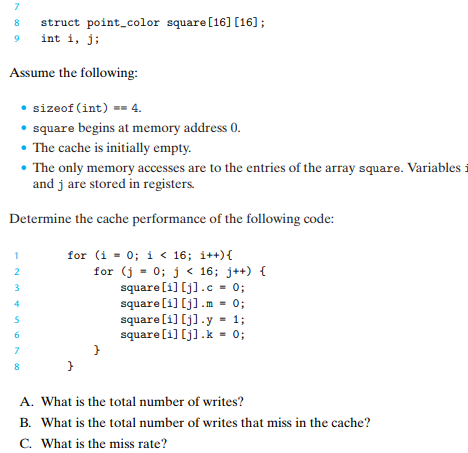
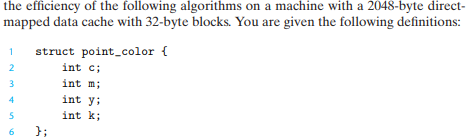
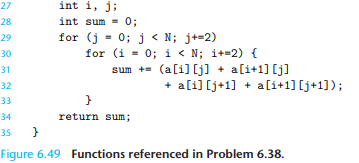
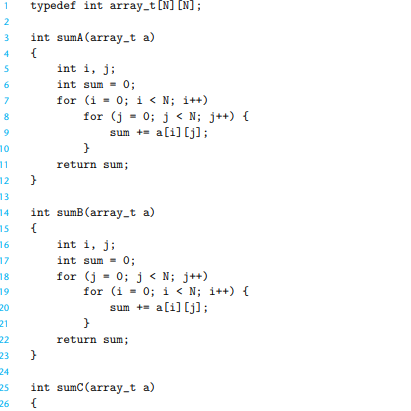
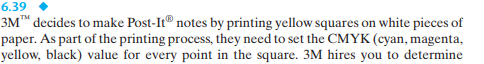
all hit

althrough N is smaller and not power of 2, miss rate is still 25%

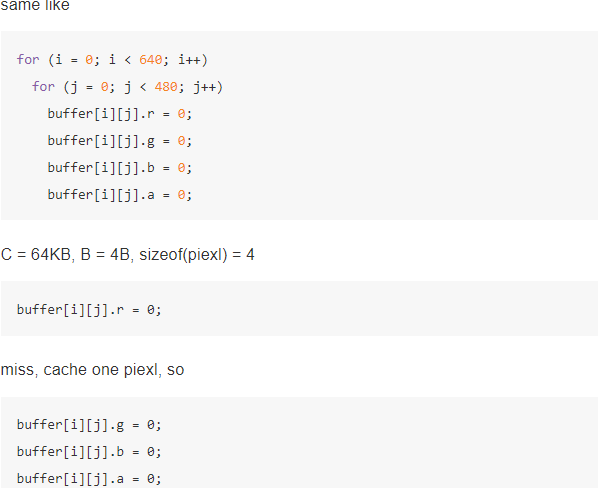
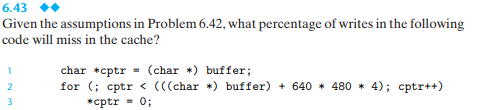
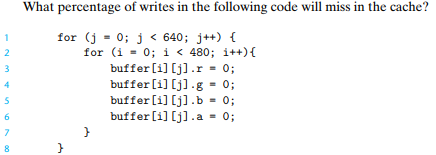
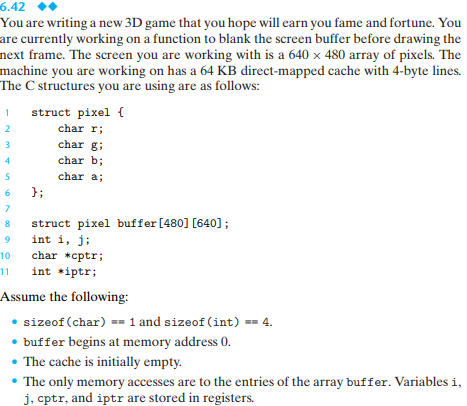
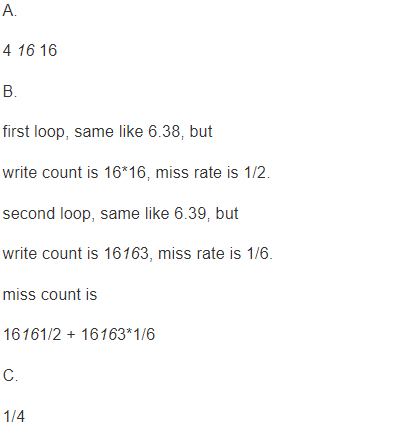
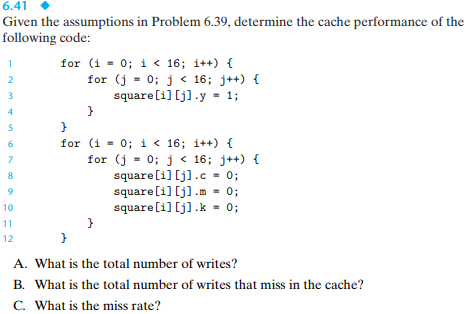
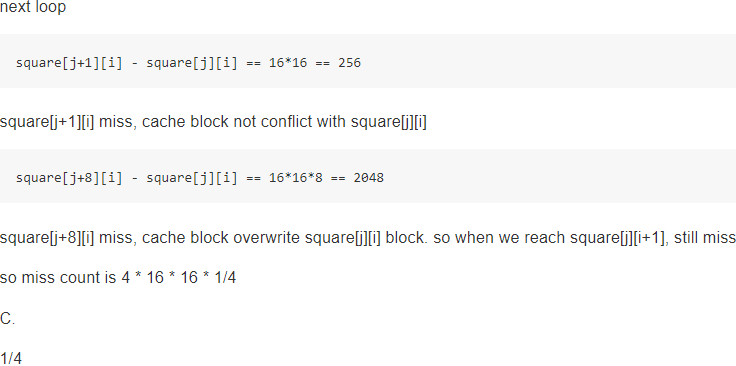
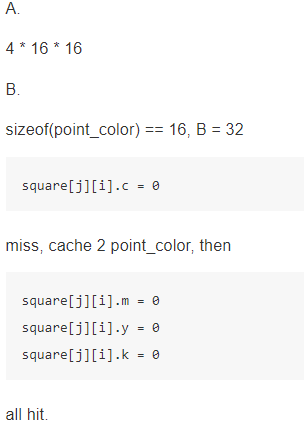
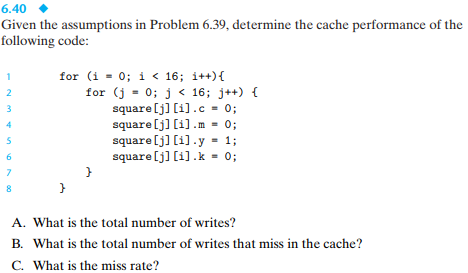
C. sumC

same as miss rate when N = 64

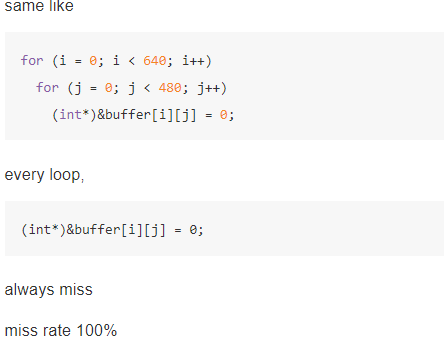
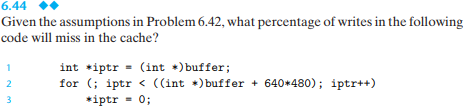
25%











Clock frequency is approx. 2500.0 MHz

Memory mountain (MB/sec)

s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15

128m 12824 7552 5119 3776 2981 2452 2080 1799 1663 1563 1483 1410 1372 1334 1304

64m 12880 7575 5121 3790 2976 2456 2079 1797 1663 1563 1489 1415 1366 1333 1302

32m 12849 7635 5137 3785 2996 2456 2075 1795 1671 1570 1486 1419 1372 1321 1299

16m 12906 7656 5174 3826 3000 2504 2090 1808 1700 1579 1506 1442 1397 1373 1357

8m 13045 7832 5321 4072 3121 2577 2194 1885 1787 1706 1663 1633 1622 1621 1703

4m 13303 8352 5602 4210 3326 2765 2352 2033 1931 1859 1824 1812 1817 1906 2076

2m 16265 11003 9150 7368 6024 5014 4310 3816 3795 3761 3848 3877 3909 3946 4044

1024k 16708 11597 10180 8150 6605 5544 4772 4169 4109 4150 4136 4127 4113 4127 4119

512k 16674 11613 10179 8160 6595 5543 4787 4182 4245 4242 4300 4360 4416 4508 4668

256k 16929 12872 11826 10661 8538 7360 6383 5724 6516 6951 7181 7297 7041 7634 7768

128k 16992 15381 14141 13145 12497 11444 10028 9734 9642 9638 9671 9696 9606 9438 9117

64k 17109 15238 14163 13563 12263 11302 9736 9412 9175 9162 9239 9200 9267 9229 12245

32k 18060 17655 17503 16954 16785 16489 16718 14382 16135 16512 16471 15799 14318 16250 15337

16k 17809 17534 17326 16516 16783 16406 16250 13333 14043 15280 13890 12546 13347 12394 17062

