Question 1: 6

Assume that the elements of the *n* x *n* matrix A are stored row wise in memory (row major numbering) and that the cache size is enough to store only half the elements of A.

a) If the cache **block size is 8 words**, what is the cache miss rate while accessing the elements of A during the following computation of the multiplication of A by two vectors, Y and W?

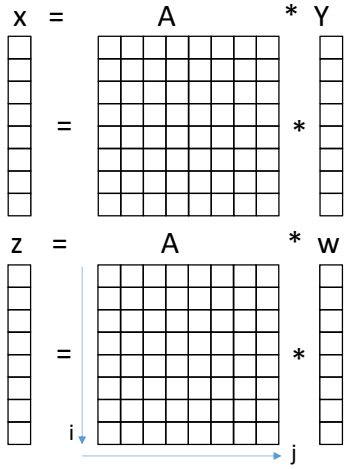
b) How can the computation be rearranged to improve the miss rate while accessing A?

a) The miss rate is 12.5%

b) Fuse the two loops to reduce the miss rate by half

```
for (i = 0; i < n; i = i+1)
{    x[i] = 0 ;
     z[i] = 0 ;
     for (j = 0; j< n; j = j+1)
       { x[i] = x[i] + A[i][j] * Y[j] ;
         z[i] = z[i] + A[i][j] * W[j] ;
       } ;
}
```

```
for (i = 0; i < n; i = i+1)
{    x[i] = 0 ;
     for (j = 0; j< n; j = j+1)
        x[i] = x[i] + A[i][j] * Y[j] ;
} ;
for (i = 0; i < n; i = i+1)
{    z[i] = 0 ;
     for (j = 0; j< n; j = j+1)
        z[i] = z[i] + A[i][j] * W[j] ;
} ;
```



Question 2: 7

Assume a system with a 8KB cache and cache block size of 4 words. The following computation multiply *n* x *n* matrices, where n = 40 (40*40*4 = 6.4KB is enough to store all the elements of one *40* x *40* matrix).

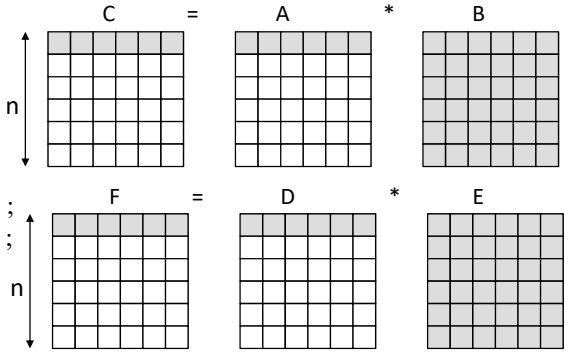
a) What is the miss rate while accessing the elements of B and E?

b) How can you rearrange the computation to improve this miss rate?

a) The miss rate is 25%

b) Splitting the loop into one loop to compute r and one to compute v. This reduces the miss rate by a factor of 40.

```
for (i = 0 ; i < 40 ; i++)
  for (j = 0 ; j < 40 ; j ++)
   { r = 0 ;
     v = 0 ;
     for (k = 0; k < 40 ; k++)
        { r = r +  A[i][k]*B[k][j] ;
          v = v + D[i][k]*E[k][j] ;
        } ;
     C[i][j] = r;
     F[i][j] = v;
   };
```



Explanation: In the loop, each element of B and E is used 40 times, once for each index i. However, if the cache cannot fit both B and E, each element will be evicted before it is reused. If the loop is split, B will fit in the cache and its elements will be reused (40 times), thus reducing the miss rate by a factor of 40. The same applies to the elements of E.

```
for (i = 0 ; i < n ; i++)
  for (j = 0 ; j < n ; j ++)
   { r = 0 ;
     v = 0 ;
     for (k = 0; k < n; k++)
        { r = r +  A[i][k]*B[k][j] ;
          v = v + D[i][k]*E[k][j] ;
        } ;
    C[i][j] = r;
    F[i][j] = v;
   };
```

b) Fuse the two loops to reduce the miss rate by half

```
for (i = 0 ; i < n ; i++)
  for (j = 0 ; j < n ; j ++)
   { r = 0 ;
     for (k = 0; k < n; k++)
        r = r +  A[i][k]*B[k][j] ;
     C[i][j] = r;
   } ;
for (i = 0 ; i < n ; i++)
  for (j = 0 ; j < n ; j ++)
   { v = 0 ;
     for (k = 0; k < n; k++)
       v = v + D[i][k]*E[k][j] ;
     F[i][j] = v;
   };
```

5

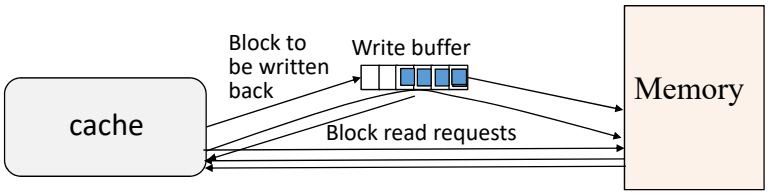
Question 3:

a) Give two reasons to argue that write buffers reduce the miss penalty.

1: do not have to wait until a dirty block is written back before bringing the missed block into cache
2: if the missing block is still in the write buffer, it will be returned to the cache quicker

b) Can you modify the write buffer scheme to maximize the benefit that you may get from one of the two reasons indicated in part a?

Keep the blocks in the write buffer as long as possible before writing them back (the idea of what is called a “victim cache”)



5

Question 4: (a) This question is about a Hamming code where each 4-bit data word is coded as a 7-bits code word. Generate the code word for each of the following data words:

0000 → 0000000

	p1	p2	0	p3	0	0	0
p1	0		0		0		0
p2		0	0			0	0
p3				0	0	0	0

0010 → 0101010

	p1	p2	0	p3	0	1	0
p1	0		0		0		0
p2		1	0			1	0
p3				1	0	1	0

0110 → 1100110

	p1	p2	0	p3	1	1	0
p1	1		0		1		0
p2		1	0			1	0
p3				0	1	1	0

1100 → 0111100

	p1	p2	1	p3	1	0	0
p1	0		1		1		0
p2		1	1			0	0
p3				1	1	0	0

1111 → 1111111

	p1	p2	1	p3	1	1	1
p1	1		1		1		1
p2		1	1			1	1
p3				1	1	1	1

7

Question 4:

b) What is the minimum Hamming distance between the code words corresponding to the five data words specified in (a)?

The distance between any two code words is at least 3

0000 → 0000000
0010 → 0101010 (3)
0110 → 1100110 (3)
1100 → 0111100 (4)
1111 → 1111111 (3)

(4)

c) Assuming at most a single bit error, retrieve the correct data word corresponding to the code words: 1010101 and 1110111

1010101 → syndrome = 000 →
no errors detected →
data word is 1101

1110111 → syndrome = 100 →
error is in bit position 4 →
the corrected code word is 1111111 →
the decoded data word is 1111

	1	0	1	0	1	0	1
s1=0	1		1		1		1
s2=0		0	1			0	1
s3=0				0	1	0	1

Syndrome = s3 s2 s1

	1	1	1	0	1	1	1
s1=0	1		1		1		1
s2=0		1	1			1	1
s3=1				0	1	1	1

Syndrome = s3 s2 s1

Question 4:

d) As described above, 1100 is encoded as 0111100. Assume that two bits are flipped while reading 0111100 from memory resulting in 1011100, what will be the outcome of the decoding process?

1011100 → syndrome = 011 →
error is in bit position 3 →
the corrected code word is 1001100 →
the decoded data word is 0100,
which is not the original word 1100

	1	0	1	1	1	0	0
s1=1	1		1		1		0
s2=1		0	1			0	0
s3=0				1	1	0	0

Syndrome = s3 s2 s1