Homework 5 solutions

Textbook Problems:

5.8) Media applications that play audio or video files are part of a class of workloads called "streaming" workloads (i.e., they bring in large amounts of data but do not reuse much of it). Consider a video streaming workload that accesses a 512 KiB working set sequentially with the following word address stream:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 …

5.8.1) [10] <§§5.4, 5.8> Assume a 64 KiB direct-mapped cache with a 32-byte block. What is the miss rate for the address stream above? How is this miss rate sensitive to the size of the cache or the working set? How would you categorize the misses this workload is experiencing, based on the 3C model?

Solution:

a) What is the miss rate for the address stream above?

Given: video streaming workload access time = 512 [KiB], sequence of address stream = 0, 2, 4, 6, 8, 10, 12, 14, 16, …, difference between the address streams = 2, direct mapped cache size = 64 [KiB], and size of cache line = 32 [bytes]

Determine block size:

block size = (video streaming workload access time)/(size of cache line) = (512 [KiB])/(32 [bytes]) = 16 [K]

Determine occurrence of miss:

occurrence of miss = ((direct mapped cache size)/(block size))/(difference between the address streams) = ((64 [KiB])/(16 [K])/(2) = (2 [B])*((8 [bits])/(1 [B])) = 16 [bits]

Determine miss rate:

miss rate = (1/occurence of miss)*100% = 1/(16 [bits])*100% = 6.25 [%]

From the calculations, the miss rate for 32 bytes cache line is 6.25 [%]. (ANS)

b) How is the miss rate sensitive to the size of the cache or the working set?

The miss rate is independent of the working set as well as the size of the cache. So, the miss rate does not change with the working set or cache size. (ANS)

c) How would you categorize the misses this workload is experiencing, based on the 3C model?

In a 3C Model, the cache model consists of cache misses classified into one of the following 3 categories:

1) compulsory misses/cold-start misses
2) capacity misses
3) conflict misses/collision misses

As seen, since the cache misses are caused based on the first access to a block and the remaining accesses are not present in the cache, the cache misses fall under the category "cold-start misses". (ANS)

5.8.2) [5] <§§5.1, 5.8> Re-compute the miss rate when the cache block size is 16 bytes, 64 bytes, and 128 bytes. What kind of locality is this workload exploiting?

Solution:

a) Recompute the miss rate when the cache block size is 16 bytes.

Given: video streaming workload access time = 512 [KiB], sequence of address stream = 0, 2, 4, 6, 8, 10, 12, 14, 16, …, difference between the address streams = 2, direct mapped cache size = 64 [KiB], and size of cache line = 16 [bytes]

Determine block size:

block size = (video streaming workload access time)/(size of cache line) = (512 [KiB])/(16 [bytes]) = 32 [K]

Determine occurrence of miss:

occurrence of miss = ((direct mapped cache size)/(block size))/(difference between the address streams) = ((64 [KiB])/(32 [K])/(2) = (1 [B])*((8 [bits])/(1 [B])) = 8 [bits]

Determine miss rate:

miss rate = (1/occurence of miss)*100% = 1/(8 [bits])*100% = 12.5 [%]

From the calculations, the miss rate for 16 bytes cache line is 12.5 [%]. (ANS)

b) Recompute the miss rate when the cache block size is 64 bytes.

Given: video streaming workload access time = 512 [KiB], sequence of address stream = 0, 2, 4, 6, 8, 10, 12, 14, 16, …, difference between the address streams = 2, direct mapped cache size = 64 [KiB], and size of cache line = 64 [bytes]

Determine block size:

block size = (video streaming workload access time)/(size of cache line) = (512 [KiB])/(64 [bytes]) = 8 [K]

Determine occurrence of miss:

occurrence of miss = ((direct mapped cache size)/(block size))/(difference between the address streams) = ((64 [KiB])/(8 [K])/(2) = (4 [B])*((8 [bits])/(1 [B])) = 32 [bits]

Determine miss rate:

miss rate = (1/occurence of miss)*100% = 1/(32 [bits])*100% = 3.125 [%]

From the calculations, the miss rate for 64 bytes cache line is 3.125 [%]. (ANS)

c) Recompute the miss rate when the cache block size is 128 bytes.

Given: video streaming workload access time = 512 [KiB], sequence of address stream = 0, 2, 4, 6, 8, 10, 12, 14, 16, …, difference between the address streams = 2, direct mapped cache size = 64 [KiB], and size of cache line = 128 [bytes]

Determine block size:

block size = (video streaming workload access time)/(size of cache line) = (512 [KiB])/(128 [bytes]) = 4 [K]

Determine occurrence of miss:

occurrence of miss = ((direct mapped cache size)/(block size))/(difference between the address streams) = ((64 [KiB])/(4 [K])/(2) = (8 [B])*((8 [bits])/(1 [B])) = 64 [bits]

Determine miss rate:

miss rate = (1/occurence of miss)*100% = 1/(64 [bits])*100% = 1.5625 [%]

From the calculations, the miss rate for 128 bytes cache line is 1.5625 [%]. (ANS)

d) What kind of locality is this workload exploiting?

The kind of locality this workload is exploiting is spatial locality because the next access is a nearby one for every access. (ANS)

5.8.3) [10] <§5.13> “Prefetching” is a technique that leverages predictable address patterns to speculatively bring in additional cache blocks when a particular cache block is accessed. One example of prefetching is a stream buffer that prefetches sequentially adjacent cache blocks into a separate buffer when a particular cache block is brought in. If the data are found in the prefetch buffer, it is considered as a hit, moved into the cache, and the next cache block is prefetched. Assume a two-entry stream buffer; and, assume that the cache latency is such that a cache block can be loaded before the computation on the previous cache block is completed. What is the miss rate for the address stream above?

Solution:

Given: video streaming workload access time = 512 [KiB], sequence of address stream = 0, 2, 4, 6, 8, 10, 12, 14, 16, …, difference between the address streams = 2

Assumptions:

- A two-entry stream buffer.
- Cache latency is such that one cache block can be preloaded before the completion of the previous cache block.
- A cache is divided into cache blocks also called cache lines.
- Prefetching technique is effectively used in streaming applications.
- In the prefetching approach, future accesses are predicted at an early stage into the cache memory and the cache blocks are obtained on speculation.
- During the execution of the current block a new speculated block is preloaded, through which the miss rate reduces to zero effectively.
- When byte 0 is accessed, it results in a miss. Fetch the entire cache block and cache it. Using the two-entry stream buffer prefetching, the sequentially adjacent cache block is prefetched into separate buffers 1 and 2.
- When the byte 2 is accessed, and if it is already present in prefetch buffer it is considered as a hit.
- Similarly, prefetch the next 2 bytes 3 and 4. The process for the rest of 512 KiB.
- Observing the sequence of addresses, there occurs only 1 miss and rests of accesses are hits.

Determine miss rate:

miss rate = ((occurence of misses)/(total accesses))*100% = (1/((512*1024)/2))*100% =0.000382 [%]

From the calculations, the miss rate for using two-stream buffer prefetching is 0.000382 [%] that shows that it reduces the miss rate nearly to zero. (ANS)

5.14) This exercise examines the single error correcting, double error detecting(SEC/DED) Hamming code.

5.14.1) [5] <§5.5> What is the minimum number of parity bits required to protect a 128-bit word using the SEC/DED code?

Solution:

Assumptions:

- Distance between the members of the code is main important factor.
- For two distances between members is count for an error.
- Two distances error detection is count for single error correcting methods.
- Suppose we have p parity bits for DED. Then the number of bits we can report error on is 2^p. With these parity bits, the total number of bits is p + 128. We need one more bit to indicate "no error". Thus, we have: (note: text book p.437 also explain the following equation)

  2^p >= p + 128 + 1

Process of determining number of parity bits required to protect a 128-bit word:

- First number starts from the leftmost first bit.
- Mark all the position as parity bits that are power of 2.
- All other bit are mark as data bit.

Determine parity bit for 128-bit data:

Power Function	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
Bit Position	1	2	4	8	16	32	64	128

Check parity count of sequence:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Determine minimum number of parity bits using equation for DED:

For DED, by trying out different positive integers for p we found that the smallest p is 8.

Determine minimum number of parity bits using equation for SEC:

For SEC, we need one more bit. Thus, the minimum number of parity bits required is 8+1=9.

From the calculations, the minimum number of parity bits required to protect a 128-bit word using the DED and SEC code is 8 and 9, respectively. (ANS)

5.14.2) [5] <§5.5> Section 5.5 states that modern server memory modules (DIMMs) employ SEC/DED ECC to protect each 64 bits with 8 parity bits. Compute the cost/performance ratio of this code to the code from Exercise 5.14.1. In this case, cost is the relative number of parity bits needed while performance is the relative number of errors that can be corrected. Which is better?

Solution:

a) Compute the cost/performance ratio of this code (128-bit) to the code from 5.14.1 (64-bit).

Given: cost for 64-bit = 8, performance for 64-bit = 2, cost for 128-bit = 9, and performance for 128-bit = 5

Assumptions:

- Modern memory server module use the DIMM or dual in memory module for high configuration ECC checking with dual bit error detection. In SIMM or single in memory module block use SEC ECC pattern for error bit.
- In SEC/DED, for single error, dual error detection pattern is follows in DIMM. So for that reason, 64-bit data is use 8 bit parity checking with dual detection. One extra parity bit is attached for dual detection and extra data allocation.
- Cost is mapped with relative number of parity bits and performance is measured by number of error (i.e. number of 1's).
- Since for 64-bit the parity bits are 8-bit long, the cost is 8. And because the maximum error pattern is 10101010, the total number of error is 4.
- Since for 128-bit the parity bits are 9-bit long, the cost is 9. And because the maximum error pattern is 101010101, the total number of error is 5.

Determine cost/performance ratio for 64-bit code:

cost/performance ratio for 64-bit = (cost for 64-bit)/(performance for 64-bit) = (8)/(4) = 2

Determine cost/performance ratio for 128-bit code:

cost/performance ratio for 128-bit = (cost for 128-bit)/(performance for 128-bit) = (9)/(5) = 1.8

Determine ratio of cost/performance for 64-bit to 128-bit codes:

ratio of cost/performance for 64-bit to 128-bit = (cost/performance ratio for 64-bit)/(cost/performance ratio for 128-bit) = (2)/(1.8) = 10:9 or 1.111

From the calculations, the ratio of cost/performance for 64-bit to 128-bit is 10:9. (ANS)

b) Which code is better?

Typically, the higher value is considered for the better performance because the number of error bit and correct bit ratio is the same in 64-bit SEC/DED pattern. However, in the 128-bit data more than 50% parity bits are used for error. Therefore, the 64-bit with 8 bit parity is better to protect. (ANS)

5.14.3) [5] <§5.5> Consider a SEC code that protects 8 bit words with 4 parity bits. If we read the value 0x375, is there an error? If so, correct the error.

Solution:

Assumptions:

- SEC code considers a single error pattern in data blocks.
- Parity bits are used for the error checking module.
- Refer to parity bits for 8-bit data pattern:

Bit Position	1	2	3	4
Parity	P1	P2	P4	P8

- With this, the binary equivalent for hexadecimal value 0x375 is: $(0x375)_{16}$ = $(001101110101)_2$

Determine received parity values:

Representation	P_1	P_2	D_1	P_4	D_2	D_3	D_4	P_8	D_5	D_6	D_7	D_8
Data	0	0	1	1	0	1	1	1	0	1	0	1
Position	1	2	3	4	5	6	7	8	9	10	11	12

Representation	D_8	D_7	D_6	D_5	P_8	D_4	D_3	D_2	P_4	D_1	P_2	P_1
Data	1	0	1	0	1	1	1	0	1	1	0	0

Position	12	11	10	9	8	7	6	5	4	3	2	1

Determine expected parity values and check if there's an error:

- From parity bit 1, the P_1 is 0 and check positions with data with (1, 3, 5, 7, 9, 11) = 0, 1, 0, 1, 0, 0. From this, the parity is 0 (i.e. even number of 1's). Because the received and expected parity values match, there is no error here.
- From parity bit 2, the P_2 is 0 and check positions with data with (2, 3, 6, 7, 10, 11) = 0, 1, 1, 1, 1, 0. From this, the parity is 0 (i.e. even number of 1's). Because the received and expected parity values match, there is no error here.
- From parity bit 4, the P_4 is 1 and check positions with data with (4, 5, 6, 7, 12) = 1, 0, 1, 1, 1. From this, the parity is 1 (i.e. even number of 1's). Because the received and expected parity values match, there is no error here.
- From parity bit 8, the P_4 is 1 and check positions with data with (8, 9, 10, 11, 12) = 1, 0, 1, 0, 1. From this, the parity is 1 (i.e. odd number of 1's). Because the received and expected parity values do not match, there is an error here.

From the calculations, because the expected parity value is different from the received parity values, an error is present in the received data. (ANS)

Determine corrected hexadecimal value:

- Because the parity bits forms $(1000)_2$ pattern that equal to 8, the bit 8 must be wrong.
- To correct the received parity values, it needs to invert the bit 8 as shown:

Representation	P_1	P_2	D_1	P_4	D_2	D_3	D_4	P_8	D_5	D_6	D_7	D_8
Data	0	0	1	1	0	1	1	**0**	0	1	0	1
Position	1	2	3	4	5	6	7	8	9	10	11	12

Representation	D_8	D_7	D_6	D_5	P_8	D_4	D_3	D_2	P_4	D_1	P_2	P_1
Data	1	0	1	0	**0**	1	1	0	1	1	0	0
Position	12	11	10	9	8	7	6	5	4	3	2	1

From the calculations, the corrective hexadecimal value is $(0x365)_{hex}$. (ANS)

5.16) As described in Section 5.7, virtual memory uses a page table to track the mapping of virtual addresses to physical addresses. This exercise shows how this table

must be updated as addresses are accessed. The following data constitute a stream of virtual byte addresses as seen on a system. Assume 4 KiB pages, a four-entry fully associative TLB, and true LRU replacement. If pages must be brought in from disk, increment the next largest page number.

Decimal	4669	2227	13916	34587	48870	12608	49225
hex	0x123d	0x08b3	0x365c	0x871b	0xbee6	0x3140	0xc049

TLB

Valid	Tag	Physical Page Number	Time Since Last Access
1	0xb	12	4
1	0x7	4	1
1	0x3	6	3
0	0x4	9	7

Page table

Index	Valid	Physical Page or in Disk
0	1	5
1	0	Disk
2	0	Disk
3	1	6
4	1	9
5	1	11
6	0	Disk
7	1	4
8	0	Disk
9	0	Disk
a	1	3
b	1	12

5.16.1) [10] <§5.7> For each access shown above, list

- ■whether the access is a hit or miss in the TLB,
- ■whether the access is a hit or miss in the page table,
- ■whether the access is a page fault,
- ■the updated state of the TLB.

Solution:

Access 0x123d:

Virtual Page Number: 0x1

Tag: 0x1
TLB Hit: No
Page Table hit: No -> Page fault
New TLB:

Valid	Tag	Phys Page #	Last used
1	0xb	12	5
1	0x7	4	2
1	0x3	6	4
1	0x1	13	1

Access 0x08b3:

Virtual Page Number: 0x0
Tag: 0x0
TLB Hit: No
Page Table hit: Yes -> No Page fault
New TLB:

Valid	Tag	Phys Page #	Last used
1	0x0	5	1
1	0x7	4	3
1	0x3	6	5
1	0x1	13	2

Access 0x365c:

Virtual Page Number: 0x3
Tag: 0x3
TLB Hit: Yes
Page Table hit: Not checked
New TLB:

Valid	Tag	Phys Page #	Last used
1	0x0	5	2
1	0x7	4	4
1	0x3	6	1
1	0x1	13	3

Access 0x871b:

Virtual Page Number: 0x8
Tag: 0x8
TLB Hit: No
Page Table hit: No -> Page fault
New TLB:

Valid	Tag	Phys Page #	Last used
1	0x0	5	3
1	0x8	14	1
1	0x3	6	2
1	0x1	13	4

Access 0xbeeg:

Virtual Page Number: 0xb
Tag: 0xb
TLB Hit: No
Page Table hit: Yes -> No Page fault
New TLB:

Valid	Tag	Phys Page #	Last used
1	0x0	5	4
1	0x8	14	2
1	0x3	6	3
1	0xb	12	1

Access 0x3140:

Virtual Page Number: 0x3
Tag: 0x3
TLB Hit: Yes
Page Table hit: Not checked
New TLB:

Valid	Tag	Phys Page #	Last used
1	0x0	5	5
1	0x8	14	3

1	0x3	6	1
1	0xb	12	2

Access 0xc049:

Virtual Page Number: 0xc
Tag: 0xc
TLB Hit: No
Page Table hit: No -> Page fault
New TLB:

Valid	Tag	Phys Page #	Last used
1	0xc	15	1
1	0x8	14	4
1	0x3	6	2
1	0xb	12	3

5.16.2) [15] <§5.7> Repeat Exercise 5.16.1, but this time use 16 KiB pages instead of 4 KiB pages. What would be some of the advantages of having a larger page size? What are some of the disadvantages?

Solution:

Access 0x123d:

Virtual Page Number: 0x0
Tag: 0x0
TLB Hit: No
Page Table hit: Yes -> No Page fault
New TLB:

Valid	Tag	Phys Page #	Last used
0	0xb	12	5
0	0x7	4	2
1	0x3	6	4
1	0x0	5	1

Access 0x08b3:

Virtual Page Number: 0x0
Tag: 0x0
TLB Hit: Yes
Page Table hit: Not checked
New TLB:

Valid	Tag	Phys Page #	Last used
0	0xb	12	6
0	0x7	4	3
1	0x3	6	5
1	0x0	5	1

Access 0x365c:

Virtual Page Number: 0x0
Tag: 0x0
TLB Hit: Yes
Page Table hit: Not checked
New TLB:

Valid	Tag	Phys Page #	Last used
0	0xb	12	7
0	0x7	4	4
1	0x3	6	6
1	0x0	5	1

Access 0x871b:

Virtual Page Number: 0x2
Tag: 0x8
TLB Hit: No
Page Table hit: No -> Page fault
New TLB:

Valid	Tag	Phys Page #	Last used
1	0x2	13	1
0	0x7	4	5

1	0x3	6	7
1	0x0	5	2

Access 0xbeeg:

Virtual Page Number: 0x2
Tag: 0x2
TLB Hit: Yes
Page Table hit: Not checked
New TLB:

Valid	Tag	Phys Page #	Last used
1	0x2	13	1
0	0x7	4	6
1	0x3	6	8
1	0x0	5	3

Access 0x3140:

Virtual Page Number: 0x0
Tag: 0x0
TLB Hit: Yes
Page Table hit: Not checked
New TLB:

Valid	Tag	Phys Page #	Last used
1	0x2	13	2
0	0x7	4	7
1	0x3	6	9
1	0x0	5	1

Access 0xc049:

Virtual Page Number: 0x3
Tag: 0x3
TLB Hit: Yes
Page Table hit: Not checked
New TLB:

Valid	Tag	Phys Page #	Last used
1	0x2	13	3
0	0x7	4	8
1	0x3	6	1
1	0x0	5	2

Larger page sizes generally means fewer page table/TLB misses because more data is moved at once and there are fewer page numbers to keep track of. This tends to come at a cost of slower read operations and an even higher penalty for when you do page fault. You also use main memory less efficiently so you may not be able to store as many pages in memory. (ANS)

5.16.3) [15] <§5.7> Repeat Exercise 5.16.1, but this time use 4 KiB pages and a two-way set associative TLB.

Solution:

Initial TLB:

Index	Tag	Valid	Phys Page #	Last used	Tag	Valid	Phys Page #	Last used
0	0x2	1	9	7	0x5	0	12	4
1	0x3	1	4	1	0x1	1	6	3

Access 0x123d:

Virtual Page Number: 0x1
Tag: 0x0 Index: 1
TLB Hit: No
Page Table hit: No-> Page fault
New TLB:

Index	Tag	Valid	Phys Page #	Last used	Tag	Valid	Phys Page #	Last used
0	0x2	1	9	7	0x5	0	12	4
1	0x3	1	4	2	0x0	1	13	1

Access 0x08b3:

Virtual Page Number: 0x0
Tag: 0x0 Index:0
TLB Hit: No
Page Table hit: Yes -> No Page fault
New TLB:

Index	Tag	Valid	Phys Page #	Last used	Tag	Valid	Phys Page #	Last used
0	0x2	1	9	8	0x0	1	5	1
1	0x3	1	4	2	0x0	1	13	1

Access 0x365c:

Virtual Page Number: 0x3
Tag: 0x1 Index: 1
TLB Hit: No
Page Table hit: Yes -> No Page fault
New TLB:

Index	Tag	Valid	Phys Page #	Last used	Tag	Valid	Phys Page #	Last used
0	0x2	1	9	8	0x0	1	5	1
1	0x1	1	6	1	0x0	1	13	2

Access 0x871b:

Virtual Page Number: 0x8
Tag: 0x4 Index: 0
TLB Hit: No
Page Table hit: No-> Page fault
New TLB:

Index	Tag	Valid	Phys Page #	Last used	Tag	Valid	Phys Page #	Last used
0	0x4	1	14	1	0x0	1	5	2
1	0x1	1	6	1	0x0	1	13	2

Access 0xbeeg:

Virtual Page Number: 0xb
Tag: 0x5 Index: 1

TLB Hit: No
Page Table hit: Yes -> No page fault
New TLB:

Index	Tag	Valid	Phys Page #	Last used	Tag	Valid	Phys Page #	Last used
0	0x4	1	14	1	0x0	1	5	2
1	0x1	1	6	2	0x5	1	12	1

Access 0x3140:

Virtual Page Number: 0x3
Tag: 0x1 Index: 1
TLB Hit: Yes
Page Table hit: Not checked
New TLB:

Index	Tag	Valid	Phys Page #	Last used	Tag	Valid	Phys Page #	Last used
0	0x4	1	14	1	0x0	1	5	2
1	0x1	1	6	1	0x5	1	12	2

Access 0xc049:

Virtual Page Number: 0xc
Tag: 0x5 Index:0
TLB Hit: No
Page Table hit: No -> Page fault
New TLB:

Index	Tag	Valid	Phys Page #	Last used	Tag	Valid	Phys Page #	Last used
0	0x4	1	14	2	0x5	1	15	1
1	0x1	1	6	1	0x5	1	12	2

5.16.4) [15] <§5.7> Repeat Exercise 5.16.1, but this time use 4 KiB pages and a direct mapped TLB.

Solution:

Initial TLB:

Index	Valid	Tag	Phys Page #
0	0	0x1	9
1	0	0x0	6
2	0	0x2	12
3	1	0x2	4

Access 0x123d:

Virtual Page Number: 0x1
Tag: 0x0 Index: 1
TLB Hit: No
Page Table hit: No-> Page fault
New TLB:

Index	Valid	Tag	Phys Page #
0	0	0x1	9
1	1	0x0	13
2	0	0x2	12
3	1	0x2	4

Access 0x08b3:

Virtual Page Number: 0x0
Tag: 0x0 Index: 0
TLB Hit: No
Page Table hit: Yes -> No Page fault
New TLB:

Index	Valid	Tag	Phys Page #
0	1	0x0	5
1	1	0x0	13
2	0	0x2	12
3	1	0x2	4

Access 0x365c:

Virtual Page Number: 0x3
Tag: 0x0 Index: 3
TLB Hit: No
Page Table hit: Yes -> No Page fault
New TLB:

Index	Valid	Tag	Phys Page #
0	1	0x0	5
1	1	0x0	13
2	0	0x2	12
3	1	0x0	6

Access 0x871b:

Virtual Page Number: 0x8
Tag: 0x2 Index: 0
TLB Hit: No
Page Table hit: No -> Page fault
New TLB:

Index	Valid	Tag	Phys Page #
0	1	0x2	14
1	1	0x0	13
2	0	0x2	12
3	1	0x0	6

Access 0xbee6:

Virtual Page Number: 0xb
Tag: 0x2 Index: 3
TLB Hit: No
Page Table hit: Yes -> No Page fault
New TLB:

Index	Valid	Tag	Phys Page #
0	1	0x2	14
1	1	0x0	13

2	0	0x2	12
3	1	0x2	12

Access 0x3140:

Virtual Page Number: 0x3
Tag: 0x0 Index: 3
TLB Hit: No
Page Table hit: Yes -> No Page fault
New TLB:

Index	Valid	Tag	Phys Page #
0	1	0x2	14
1	1	0x0	13
2	0	0x2	12
3	1	0x0	6

Access 0xc049:

Virtual Page Number: 0xc
Tag: 0x2 Index: 2
TLB Hit: No
Page Table hit: No -> Page fault
New TLB:

Index	Valid	Tag	Phys Page #
0	1	0x2	14
1	1	0x0	13
2	1	0x2	15
3	1	0x0	6

5.16.5) [10] <§§5.4, 5.7> Discuss why a CPU must have a TLB for high performance. How would virtual memory accesses be handled if there were no TLB?

Solution:

Without using a TLB you would have to look up physical addresses in the Page Table on every memory access. This would be equivalent to a TLB miss on every memory access and would dramatically slow performance. (ANS)

5.20) In this exercise, we will examine how replacement policies affect miss rate. Assume a two-way set associative cache with four one-word blocks. Consider the following word address sequence: 0, 1, 2, 3, 4, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0.

Consider the following address sequence: 0, 2, 4, 8, 10, 12, 14, 16, 0

5.20.1) [5] <§§5.4, 5.8> Assuming an LRU replacement policy, which accesses are hits?

Solution:

Cache:

Line	Set 0	Set 1
0	00	01
1	10	11

Accesses:

Address	Hit/Miss	Change	Cache
0	Miss	0->00	0 - / - -
1	Miss	1 ->10	0 - / 1 -
2	Miss	2 -> 01	0 2 / 1 -
3	Miss	3 -> 11	0 2 / 1 3
4	Miss	4 -> 00	4 2 / 1 3
2	Hit	-	4 2 / 1 3
3	Hit	-	4 2 / 1 3
4	Hit	-	4 2 / 1 3
5	Miss	5 -> 10	4 2 / 5 3
6	Miss	6 -> 01	4 6 / 5 3
7	Miss	7 -> 11	4 6 / 5 7

0	Miss	0 -> 00	0 6 / 5 7
1	Miss	1 -> 10	0 6 / 1 7
2	Miss	2 -> 01	0 2 / 1 7
3	Miss	3 -> 11	0 2 / 1 3
4	Miss	4 -> 00	0 4 / 1 3
5	Miss	5 -> 10	0 4 / 5 3
6	Miss	6 -> 00	6 4 / 5 3
7	Miss	7 -> 11	6 4 / 5 7
0	Miss	0 -> 01	6 0 / 5 7

From the calculations, the number of hits using LRU replacement policy for this address sequence is 3. (ANS)

5.20.2) [5] <§§5.4, 5.8> Assuming an MRU (most recently used) replacement policy, which accesses are hits?

Solution:

Accesses:

Address	Hit/Miss	Change	Cache
0	Miss	0->00	0 - / - -
1	Miss	1 ->10	0 - / 1 -
2	Miss	2 -> 01	0 2 / 1 -
3	Miss	3 -> 11	0 2 / 1 3
4	Miss	4 -> 01	0 4 / 1 3
2	Miss	2 -> 01	0 2 / 1 3
3	Hit	-	0 2 / 1 3
4	Miss	4 -> 01	0 4 / 1 3
5	Miss	5 -> 11	0 4 / 1 3

6	Miss	6 -> 01	0 6 / 1 3
7	Miss	7 -> 11	0 6 / 1 7
0	Hit	-	0 6 / 1 7
1	Hit	-	0 6 / 1 7
2	Miss	2 -> 00	2 6 / 1 7
3	Miss	3 -> 10	2 6 / 3 7
4	Miss	4 -> 00	4 6 / 3 7
5	Miss	5 -> 10	4 6 / 5 7
6	Hit	-	4 6 / 5 7
7	Hit	-	4 6 / 5 7
0	Miss	0 -> 01	4 0 / 5 7

From the calculations, the number of hits using MRU replacement policy for this address sequence is 5. (ANS)

5.20.3) [5] <§§5.4, 5.8> Simulate a random replacement policy by flipping a coin. For example, “heads” means to evict the first block in a set and “tails” means to evict the second block in a set. How many hits does this address sequence exhibit?

Solution:

Accesses:

Address	Hit/Miss	Change	Cache
0	Miss	0->00	0 - / - -
1	Miss	1 ->10	0 - / 1 -
2	Miss	2 -> 01	0 2 / 1 -
3	Miss	3 -> 11	0 2 / 1 3
4	Miss	4 -> 01	0 4 / 1 3
2	Miss	2 -> 01	0 2 / 1 3

3	Hit	-	0 2 / 1 3
4	Miss	4 -> 01	0 4 / 1 3
5	Miss	5 -> 11	0 4 / 1 5
6	Miss	6 -> 00	6 4 / 1 5
7	Miss	7 -> 11	6 4 / 1 7
0	Miss	0 -> 00	0 4 / 1 7
1	Hit	-	6 4 / 1 7
2	Miss	2 -> 00	2 4 / 1 7
3	Miss	3 -> 10	2 4 / 3 7
4	Hit	-	2 4 / 1 7
5	Miss	5 -> 10	2 4 / 5 7
6	Miss	6 -> 01	2 6 / 5 7
7	Hit	-	2 6 / 5 7
0	Miss	0 -> 00	0 6 / 5 7

From the calculations, the number of hits using random replacement policy for this address sequence is 4. (ANS)

5.20.4) [10] <§§5.4, 5.8> Describe an optimal replacement policy for this sequence. Which accesses are hits using this policy?

Solution:

Interestingly the best strategy seems to be no replacement. Don't boot anything once the cache fills up.

Address	Hit/Miss	Change	Cache
0	Miss	0->00	0 - / - -
1	Miss	1 ->10	0 - / 1 -
2	Miss	2 -> 01	0 2 / 1 -

3	Miss	3 -> 11	0 2 / 1 3
4	Miss	-	0 2 / 1 3
2	Hit	-	0 2 / 1 3
3	Hit	-	0 2 / 1 3
4	Miss	-	0 2 / 1 3
5	Miss	-	0 2 / 1 3
6	Miss	-	0 2 / 1 3
7	Miss	-	0 2 / 1 3
0	Hit	-	0 2 / 1 3
1	Hit	-	0 2 / 1 3
2	Hit	-	0 2 / 1 3
3	Hit	-	0 2 / 1 3
4	Miss	-	0 2 / 1 3
5	Miss	-	0 2 / 1 3
6	Miss	-	0 2 / 1 3
7	Miss	-	0 2 / 1 3
0	Hit	-	0 2 / 1 3

From the calculations, the optimal replacement policy is to have no replacement. This results in 7 hits. (ANS)

5.20.5) [10] <§§5.4, 5.8> Describe why it is difficult to implement a cache replacement policy that is optimal for all address sequences.

Solution:

Different programs will access memory in different ways. Some programs will stick to the most recently accessed pages for a long time and can leverage an LRU strategy. Some programs will march through pages and then circle back around and can leverage MRU. Others still have no discernable pattern and won't consistently benefit from either

replacement strategy. An optimal algorithm, by definition, is to replace page that will not be used for longest period of time. With this, the optical replacement policy looks for the pages that have least chances of getting used in the future. Therefore, it is necessary for the CPU to know what page frames would be demanded in the future, which is somewhat impractical. In other words, the difficulty of implementing cache replacement policy that is optimal for all address sequences is that it requires future knowledge of the reference string. Furthermore, since there are multiple processes on a system without following any patterns on page usage, optimal page replacement is difficult to implement and mainly used only for comparison purposes. (ANS)

5.20.6) [10] <§§5.4, 5.8> Assume you could make a decision upon each memory reference whether or not you want the requested address to be cached. What effect could this have on miss rate?

Solution:

This would help a lot because as the programmer you know if you will be accessing the same memory (or close by) in the near future. Since cache and pages sizes vary between machines however, this would be very difficult to program for anything but a single machine. You would also need to know exactly which addresses will be accessed and in what order to make much use of it. By making a decision upon each memory reference whether or not the address is cached, this would bring down the miss rate to be equivalent to the optical algorithm. This would simply mean that the optimal algorithm is possibly implemented giving the best possible hit rate. (ANS)

Part 1 Problems:

1) *Cache*: "A fully associative cache of total capacity 128 bytes, with 32 byte cache blocks is the same a 128 byte, 4 way set associative cache with 32 byte cache blocks." Is this statement true or false? Explain.

Solution:

No. The second one leverages indexing based on the address and will behave differently than a fully associative cache. Also a fully associative cache usually requires more bits for tracking LRU than a 4 way set associative and more tag bits as there are no index bits to reduce the tag size.

2) *TLB* : Consider a processor which has the entire page table and all the pages in the physical memory. Suppose it takes 5 nanoseconds to search the TLB and 60 nanoseconds to access the physical memory (inclusive of cache access time), with a TLB hit rate of 60%, what is the average memory access time?

Solution:

Tavg = 0.6*(Ttlb + Tmem) + (1-0.6)*(Ttlb + Tmem +Tmem)= .6*65 + .4*125 = 89 ns

3) *Disk access time*. Data access time on a hard disk drive can be expressed by

$$Tdata = Tseek + Trotation + Ttransfer$$

Consider a disk with an average seek time of 5 milliseconds (time to move the head to the desired track), and rotational speed of 7200 rotations per minute (RPM). On average a disk platter needs to spin 1/2 revolution to locate the target sector on a track. The transfer rate is 60 MB/s once the disk head finds the target sector. Assume a sector size is 512 bytes. Answer the following questions.

a) Compute T_{data} for transfering one sector from the disk.

Solution:

Tseek = 5ms

Transfer rate = 60MB/s

1 sector is 0.0005MB. =>Ttransfer = 0.0005/60 = 0.0083333ms

7200 rotation takes 1 minute = 60 seconds

=>0.5 revolution takes (0.5*60)/7200 seconds = 4.167 ms

=>Tdata = 5ms + 0.00833ms + 4.167ms = 9.17553ms

b) Compute T_{data} for reading 8 contiguous sectors from the disk.

Solution:

Tseek = 5ms

Transfer rate = 60MB/s

8 sector is 8*0.0005MB. =>Ttransfer = 8*0.0005/60 = 0.06666ms

7200 rotation takes 1 minute = 60 seconds

=>0.5 revolution takes (0.5*60)/7200 seconds = 4.167 ms

=>Tdata = 5ms + 0.06666ms + 4.167ms = 9.2336ms

c) Compute T_{data} for reading 8 random sectors from the disk, such that every sector access necessitates a seek time and rotational latency.

Solution:

From part A, Tdata = 9.17553ms for 1 sector

8 random sectors

=>Tdata = 8*Tdata_1sector = 78.4ms

4) *Two caches*, A and B have the following specs:

Cache A: 16 byte, 2-way set-associative cache with 2 byte block size.

Cache B: 16 byte, fully associative cache with 4 byte block size.

It receives memory requests to the below addresses(binary) in the given order: 00000, 10110, 10001, 00001, 11000, 10111, 00110, 11100, 00001, 11111

Assumptions:

Both the caches are initially empty.

The processor uses an LRU replacement policy.

How many offset, index and tag bits are in Cache A and Cache B?

Solution:

Cache A

Offset = log2(2) = 1 bit

C = A B S

16 byte = 2-way * 2bytes *S

=>S = 4

Index = log2(4) = 2 bits

Tag = 2 bits (Remaining bits)

Cache B

Offset = log2(4) = 2 bits

Index = 0

Tag = 3 bits (The remaining bits)

a) For each address, determine if it would be a cache hit or a cache miss for Cache A. Categorize if a cache miss is compulsory, conflict, or capacity.

Solution:

binary	tag	index	hit/miss
00000	00	00	Compulsory miss
10110	10	11	Compulsory miss
10001	10	00	Compulsory miss
00001	00	00	Hit
11000	11	00	Conflict miss
10111	10	11	Hit
00110	00	11	Compulsory miss
11100	11	10	Compulsory miss

00001	00	00	Hit
11111	11	11	Conflict Miss

b) For each address, determine if it would be a cache hit or a cache miss for Cache B. Categorize if a cache miss is compulsory, conflict, or capacity.

Solution:

binary	tag	hit/miss
00000	000	Compulsory miss
10110	101	Compulsory miss
10001	100	Compulsory miss
00001	000	Hit
11000	110	Compulsory miss
10111	101	Hit
00110	001	Capacity miss
11100	111	Capacity miss
00001	000	Capacity miss
11111	111	Hit

5) A *TLB* is 2-way set associative and holds 8 translations. It has the following entries. Each virtual memory address is 32 bits.

Index	Page Number	Frame Number	Page Number	Frame Number
0	0xADBDC	0x1234567	0x87034	0x7642867
1	0x00001	0x7123454	0xA999D	0x2345647
2	0x65432	0xDBD234D	0x44996	0x234BDCE
3	0x77777	0xAB578BC	0x98743	0x0187109

Fill out the rightmost column of the memory access table below to denote whether the memory access was a hit or a miss in the TLB. The cache uses LRU replacement, and you may assume that to start with, the right entry is the least recently used.

Virtual Mem Access	Result (Physical Mem Address)	Hit/Miss
0xBEEDDAD2	0x3092834AD2	M
0x4321A9DC	0xDA234CC9DC	M
0x9874312E	0x018710912E	H
0x10218000	0xAA234BC000	M
0x339541D0	0xDB32D781D0	M
0xBEEDD020	0x3092834020	H

What are the contents of the TLB after processing the page accesses?

The yellow-shaded entries are the ones that got replaced

Index	Page Number	Frame Number	Page Number	Frame Number
0	0x33954	0xDB32D78	0x10218	0xAA234BC

1	0x00001	0x7123454	0xBEEDD	0x3092834
2	0x65432	0xDBD234D	0x4321A	0xDA234CC
3	0x77777	0xAB578BC	0x98743	0x0187109

6) *Page Table*: The following are the page table entries of processor A.

Valid	Virtual Page (20 bits)	Physical Page(24 bits)
1	0xAAAA5	0x123456
1	0xAAAA6	0xA75B23
1	0xAAAA7	0xBABABA
0	0xAAAA8	0xBADDAD
0	0xAAAA9	0xEEEDAD

Your processor receives the instruction ldr r1, [r2]. List the complete physical memory address being accessed given the following values of r1 (32 bits), or indicate if the access causes a page fault. Assume 32 bit virtual addresses.

i. 0xAAAA9EAD
   Page number = 0xAAAA9 - Page Fault

ii. 0xAAAA7523
   Page number = 0xAAAA7 - Physical address =0xBABABA523

iii. 0xAAAAAA33
   Page number = 0xAAAAA - Page Fault

How many bytes does a single page in this system hold?

12 page offset bits = 2^12 bit pages = 4 KiB pages

7) *Cache Layout* : A processor has a separate D-cache and an I-cache. D-cache: 32KB, 2-way set associative, block size of 1 word (32 bits), *write-back* policy

I- cache: 16KB, direct mapped cache, block size of 1 word (32 bits)

Ignoring the bits required to implement the replacement policy, answer the following questions.

(a) Calculate the number of tag, index and offset bits for the D-cache.

Number of blocks = 32 (KB) / 4 (B/block) = 8 K blocks

Number of lines = 8 K blocks / 2 (blocks/line) = 4 K lines

Index bits/block = log2(4K lines) = 12 bits

Offset bits/block = log2(4 B/word) + log2(1 word/block) = 2 bits

Tag bits/block = 32 bit addr -12 index bits - 2 offset bits = 18 bits

(b) Calculate the number of tag, index and offset bits for the I-cache.

Number of blocks = 16 (KB) / 4 (B/block) = 4 K blocks

Number of lines = 4 K blocks / 1 (blocks/line) = 4 K lines

Index bits/block = log2(4K lines) = 12 bits

Offset bits/block = log2(4 B/word) + log2(1 word/block) = 2 bits

Tag bits/block = 32 bit addr - index bits - offset bits = 18 bits

(c) How many bits are needed to implement the D-cache? I cache?

D cache:

Bits/ Block = 18 (tag bits) + 32 (bits/ block) + 1 (valid bit) + 1 (dirty bit) = 52

Total Bits = 52 (bits/block) * 8 K (blocks/cache) = 425984

I cache:

Bits/ Block = 18 (tag bits) + 32 (bits/ block) + 1 (valid bit) = 51

Total Bits = 51 (bits/block) * 4 K (blocks/cache) = 208896

Part 2 Problems:

1) For a SECDED code, how does one determine whether there are 0, 1, or 2 errors? In the case of a 1-bit error, how does one determine whether the data needs to be corrected?

Solution:

In the following answer, suppose the code is `p1, p2, d1, p3, d2, d3, d4, p4`, where `p4` is the overall parity bit, we call `p1p2p3` the SEC (single error correction) bits, and `p4` the DED (dual error detection bit).

- 0 error: SEC = 0, DED = 0
- 1 error: DED = 1
- 2 errors: SEC != 0, DED = 0

For 1-bit error,

- if SEC = 0, then the only error is at the DED bit, and we don't need correction to the data
- Otherwise if SEC = power of 2, then the error is at one of the SEC bit, and we don't need correction to the data
- Otherwise, the error is in the data and we need correction.

2) The following are 4:8 Hamming SECDED codes. For each, determine whether no errors, one parity error, one data error, or 2 errors of either type were present.

For 0 errors or 1 parity error, just fill in the original (correct) 4-bit value.

For 1 data error, write the corrected data value.

For 2 errors, just leave this field blank.

In the following solutions, we will show solutions with indexing from left to right in red, and indexing from right to left in blue. **In the final exam, we will make it clear about what format to use.**

00000001

Solution:

Indexing from left to right

1	2	3	4	5	6	7	8
p1	p2	d1	p3	d2	d3	d4	p4

0	0	0	0	0	0	0	1

p1 controls bits 1, 3, 5, 7, which are 0 ^ 0 ^ 0 ^ 0 = 0. No error.

p2 controls bits 2, 3, 6, 7, which are 0 ^ 0 ^ 0 ^ 0 = 0. No error.

p3 controls bits 4, 5, 6, 7, which are 0 ^ 0 ^ 0 ^ 0 = 0. No error.

p4 (DED bit) controls all bits, which are 0 ^ 0 ^ 0 ^ 0 ^ 0 ^ 0 ^ 0 ^ 1 = 1. ERROR!

Thus, we have SEC=000_{binary}=$0_{decimal}$ and DED=1, which mean we only have 1 parity error. The correct message is 0000_{binary}

Indexing from right to left

8	7	6	5	4	3	2	1
p4	d4	d3	d2	p3	d1	p2	p1
0	0	0	0	0	0	0	1

p1 controls bits 1, 3, 5, 7, which are 1 ^ 0 ^ 0 ^ 0 = 1. ERROR!

p2 controls bits 2, 3, 6, 7, which are 0 ^ 0 ^ 0 ^ 0 = 0. No error.

p3 controls bits 4, 5, 6, 7, which are 0 ^ 0 ^ 0 ^ 0 = 0. No error.

p4 (DED bit) controls all bits, which are 0 ^ 0 ^ 0 ^ 0 ^ 0 ^ 0 ^ 0 ^ 1 = 1. ERROR!

Thus, we have SEC=001_{binary}=$1_{decimal}$ and DED=1, which means we have error at bit 1 (p2), which is a parity bit. The correct message is 0000_{binary}

00101111

Solution:

Indexing from left to right

1	2	3	4	5	6	7	8
p1	p2	d1	p3	d2	d3	d4	p4
0	0	1	0	1	1	1	1

p1 controls bits 1, 3, 5, 7, which are 0 ^ 1 ^ 1 ^ 1 = 1. ERROR!

p2 controls bits 2, 3, 6, 7, which are 0 ^ 1 ^ 1 ^ 1 = 1. ERROR!

p3 controls bits 4, 5, 6, 7, which are 0 ^ 1 ^ 1 ^ 1 = 1. ERROR!.

p4 (DED bit) controls all bits, which are 0 ^ 0 ^ 1 ^ 0 ^ 1 ^ 1 ^ 1 ^ 1 = 1. ERROR!

Thus, we have SEC = 111_{binary} = $7_{decimal}$ and DED = 1, which mean we have error at bit 7 (d4). For the correct data message, we just need to flip the bit and get 1110_{binary}

Indexing from right to left

8	7	6	5	4	3	2	1
p4	d4	d3	d2	p3	d1	p2	p1
0	0	1	0	1	1	1	1

p1 controls bits 1, 3, 5, 7, which are 1 ^ 1 ^ 0 ^ 1 = 1. ERROR!

p2 controls bits 2, 3, 6, 7, which are 1 ^ 1 ^ 1 ^ 0 = 1. ERROR!

p3 controls bits 4, 5, 6, 7, which are 1 ^ 0 ^ 1 ^ 0 = 0. No error.

p4 (DED bit) controls all bits, which are 0 ^ 0 ^ 1 ^ 0 ^ 1 ^ 1 ^ 1 ^ 1 = 1. ERROR!

Thus, we have SEC=011_{binary}=$3_{decimal}$ and DED=1, which means we have error at bit 3 (d1). For the correct data message, we just need to flip the bit and get 0100_{binary}

11110111

Solution:

1	2	3	4	5	6	7	8
p1	p2	d1	p3	d2	d3	d4	p4
1	1	1	1	0	1	1	1

p1 controls bits 1, 3, 5, 7, which are 1 ^ 1 ^ 0 ^ 1 = 1. ERROR!

p2 controls bits 2, 3, 6, 7, which are 1 ^ 1 ^ 1 ^ 1 = 0. No error.

p3 controls bits 4, 5, 6, 7, which are 1 ^ 0 ^ 1 ^ 1 = 1. ERROR!.

p4 (DED bit) controls all bits, which are 1 ^ 1 ^ 1 ^ 1 ^ 0 ^ 1 ^ 1 ^ 1 = 1. ERROR!

Thus, we have SEC = 101_{binary} = $5_{decimal}$ and DED = 1, which means we have error at bit 5 (d2). For the correct data message, we just need to flip the bit and get 1111_{binary}

Indexing from right to left

8	7	6	5	4	3	2	1
p4	d4	d3	d2	p3	d1	p2	p1
1	1	1	1	0	1	1	1

p1 controls bits 1, 3, 5, 7, which are 1 ^ 1 ^ 1 ^ 1 = 0. No error.

p2 controls bits 2, 3, 6, 7, which are 1 ^ 1 ^ 1 ^ 1 = 0. No error.

p3 controls bits 4, 5, 6, 7, which are 0 ^ 1 ^ 1 ^ 1 = 1. ERROR!

p4 (DED bit) controls all bits, which are 1 ^ 1 ^ 1 ^ 1 ^ 0 ^ 1 ^ 1 ^ 1 = 1. ERROR!

Thus, we have SEC=100_{binary}=$4_{decimal}$ and DED=1, which means we have error at bit 4 (p3), which is a parity bit. The correct message is 1111_{binary}

10100101

Solution:

Indexing from left to right

1	2	3	4	5	6	7	8
p1	p2	d1	p3	d2	d3	d4	p4
1	0	1	0	0	1	0	1

p1 controls bits 1, 3, 5, 7, which are 1 ^ 1 ^ 0 ^ 0 = 0. No error.

p2 controls bits 2, 3, 6, 7, which are 0 ^ 1 ^ 1 ^ 0 = 0. No error.

p3 controls bits 4, 5, 6, 7, which are 0 ^ 0 ^ 1 ^ 0 = 1. ERROR!.

p4 (DED bit) controls all bits, which are 1 ^ 0 ^ 1 ^ 0 ^ 0 ^ 1 ^ 0 ^ 1 = 0.

Thus, we have SEC = 001_{binary} = $1_{decimal}$ and DED = 0, which means we have 2 errors and we cannot restore the correct message.

Indexing from right to left

8	7	6	5	4	3	2	1
p4	d4	d3	d2	p3	d1	p2	p1
1	0	1	0	0	1	0	1

p1 controls bits 1, 3, 5, 7, which are 1 ^ 1 ^ 0 ^ 0 = 0. No error.

p2 controls bits 2, 3, 6, 7, which are 0 ^ 1 ^ 1 ^ 0 = 0. No error.

p3 controls bits 4, 5, 6, 7, which are 0 ^ 0 ^ 1 ^ 0 = 1. ERROR!

p4 (DED bit) controls all bits, which are 1 ^ 0 ^ 1 ^ 0 ^ 0 ^ 1 ^ 0 ^ 1 = 0.

Thus, we have SEC=100_{binary}=$4_{decimal}$ and DED=0, which means we have 2 errors and we cannot restore the correct message.

11110011

Solution:

Indexing from left to right

1	2	3	4	5	6	7	8
p1	p2	d1	p3	d2	d3	d4	p4
1	1	1	1	0	0	1	1

p1 controls bits 1, 3, 5, 7, which are 1 ^ 1 ^ 0 ^ 1 = 1. ERROR!

p2 controls bits 2, 3, 6, 7, which are 1 ^ 1 ^ 0 ^ 1 = 1. ERROR!

p3 controls bits 4, 5, 6, 7, which are 1 ^ 0 ^ 0 ^ 1 = 0. No error.

p4 (DED bit) controls all bits, which are 1 ^ 1 ^ 1 ^ 1 ^ 0 ^ 0 ^ 1 ^ 1 = 0.

Thus, we have SEC = 110_{binary} = $5_{decimal}$ and DED = 0, which means we have 2 errors and we cannot restore the correct message.

Indexing from right to left

8	7	6	5	4	3	2	1
p4	d4	d3	d2	p3	d1	p2	p1
1	1	1	1	0	0	1	1

p1 controls bits 1, 3, 5, 7, which are 1 ^ 0 ^ 1 ^ 1 = 1. ERROR!

p2 controls bits 2, 3, 6, 7, which are 1 ^ 0 ^ 1 ^ 1 = 1. ERROR!

p3 controls bits 4, 5, 6, 7, which are 0 ^ 1 ^ 1 ^ 1 = 1. ERROR!

p4 (DED bit) controls all bits, which are 1 ^ 1 ^ 1 ^ 1 ^ 0 ^ 0 ^ 1 ^ 1 = 0.

Thus, we have SEC=111_{binary}=$7_{decimal}$ and DED=0, which means we have 2 errors and we cannot restore the correct message.

01101001

Solution:

Indexing from left to right

1	2	3	4	5	6	7	8
p1	p2	d1	p3	d2	d3	d4	p4
0	1	1	0	1	0	0	1

p1 controls bits 1, 3, 5, 7, which are 0 ^ 1 ^ 1 ^ 0 = 0. No error.

p2 controls bits 2, 3, 6, 7, which are 1 ^ 1 ^ 0 ^ 0 = 0. No error.

p3 controls bits 4, 5, 6, 7, which are 0 ^ 1 ^ 0 ^ 0 = 1. ERROR!.

p4 (DED bit) controls all bits, which are 0 ^ 1 ^ 1 ^ 0 ^ 1 ^ 0 ^ 0 ^ 1 = 0

Thus, we have SEC = $001_{binary} = 1_{decimal}$ and DED = 0, which means we have 2 errors and we cannot restore the correct message.

Indexing from right to left

8	7	6	5	4	3	2	1
p4	d4	d3	d2	p3	d1	p2	p1
0	1	1	0	1	0	0	1

p1 controls bits 1, 3, 5, 7, which are 1 ^ 0 ^ 0 ^ 1 = 0. No error.

p2 controls bits 2, 3, 6, 7, which are 0 ^ 0 ^ 1 ^ 1 = 0. No error.

p3 controls bits 4, 5, 6, 7, which are 1 ^ 0 ^ 1 ^ 1 = 1. ERROR!

p4 (DED bit) controls all bits, which are 0 ^ 1 ^ 1 ^ 0 ^ 1 ^ 0 ^ 0 ^ 1 = 0

Thus, we have SEC=100_{binary}=$4_{decimal}$ and DED=0, which means we have 2 errors and we cannot restore the correct message.

3) We have seen 4:7 and 11:15 SEC codes, as well as 4:8 and 11:16 SECDED. What is the equation for finding the parity and data field bit widths in a given SEC code? SECDED?

Solution:

SEC: **2^p >= p + d + 1**

SECDED: **2^(p-1) >= p + d**