Assignment 4

Name : Fahid Imran BSAI-B Id: 23i-0061

Main Memory

Address	Data				Address	Data			
0x0000	aa	bb	cc	dd	0x011C	ab	bc	ab	bc
0x0004	12	bc	12	bc	0x0120	ab	bb	ab	bb
0x0008	ab	bb	cd	bb	0x0124	ab	bc	ab	bc
0x000C	1b	bc	1c	bc	0x0128	ab	bb	ab	bb
0x0010	ab	bb	cb	bb	0x012C	ab	bc	ab	bc
0x0014	1b	bc	1b	be	0x0130	ab	bc	ab	bc
0x0018	12	bc	12	bf	0x0134	ab	bb	ab	bb
0x001C	1b	bb	1b	bd	0x0138	ab	bc	ab	bc
0x0020	1b	bc	1b	b1	0x013C	ab	bb	ab	bb
0x0024	1b	bb	1b	bb	0x0140	ab	bc	ab	bc
0x0028	1b	bc	11	bc	0x0144	ab	bc	ab	bc
0x002C	12	bc	1b	bb	0x0148	ab	bb	ab	bb
0x0030	1b	bb	1b	be	0x014C	ab	bc	ab	bc
0x0034	1b	bc	13	bf	0x0150	ab	bb	ab	bb
0x0038	1b	bb	1b	bd	0x0154	ab	bc	ab	bc
0x003C	1b	bc	1b	b2	0x0158	ab	bc	ab	bc
0x0040	~~11~~ 01	~~bc~~ 01	~~1b~~ 01	~~bb~~ 01	0x015C	ab	bb	ab	bb
0x0044	1b	bb	1b	bc	0x0160	ab	bc	ab	bc
0x0048	1b	bc	1b	bb	0x0164	ab	bb	ab	bb
0x004C	1b	bb	1b	be	0x0168	ab	bc	ab	bc
0x0050	1b	bc	14	bf	0x016C	ab	bc	ab	bc
0x0054	ab	bc	1b	bd	0x0170	ab	bb	ab	bb
0x0058	1b	bb	1b	b3	0x0174	ab	bc	ab	bc
0x005C	ab	bc	1b	bc	0x0178	ab	bb	ab	bb
0x0060	1b	bb	(1b)	bb	0x017C	ab	bc	ab	bc
0x0064	ab	bc	1b	bc	0x0180	ab	bc	ab	bc
0x0068	~~ab~~ 98	~~bc~~ 11	~~ab~~ 22	~~bc~~ 33	0x0184	ab	bb	ab	bb
0x006C	ab	bb	1b	bb	0x0188	ab	bc	ab	bc
0x0070	ab	bc	ab	bc	0x018C	ab	bb	ab	bb
0x0074	ab	bb	1b	bb	0x0190	ab	bc	ab	bc
0x0078	ab	bc	ac	bc	0x0194	ab	bc	ab	bc
0x007C	ab	bc	1b	bc	0x0198	ab	bb	ab	bb
0x0080	ab	bb	ac	bb	0x019C	ab	bc	ab	bc
0x0084	ab	bc	1c	bc	0x01A0	ab	bb	ab	(bb)
0x0088	ab	bb	ab	bb	0x01A4	ab	bc	ab	bc

0x008C	ab	bc	1c	bc	0x01A8	ab	bc	ab	bc
0x0090	ab	bc	ab	bc	0x01AC	ab	bb	ab	bb
0x0094	ab cc	bb cc	1c cc	bb cc	0x01B0	ab	bc	ab	bc
0x0098	ab	bc	ac	bc	0x01B4	ab dd	bb dd	ab dd	bb dd
0x009C	ab	bb	1b	bb	0x01B8	ab	bc	ab	bc
0x00A0	ab 02	bc 02	ac 02	bc 02	0x01BC	ab	bc	ab	bc
0x00A4	ab	bc	1b	bc	0x01C0	ab	bb	ab	bb
0x00A8	ab	bb	ac	bb	0x01C4	ab	bc	ab	bc
0x00AC	ab	bc	1c	bc	0x01C8	ab	bb	ab	bb
0x00B0	ab 15	bb bb	ab 16	bb bb	0x01CC	ab	bc	ab	bc
0x00B4	ab	bc	1c	bc	0x01D0	ab	bc	ab	bc
0x00B8	ab	bc	ab	bc	0x01D4	ab	bb	ab	bb
0x00BC	ab	bb	1c	bb	0x01D8	ab	bc	ab	bc
0x00C0	ab	bc	ac	bc	0x01DC	ab	bb	ab	bb
0x00C4	ab	bb	1b	bb	0x01E0	2c	bc	ab	bc
0x00C8	ab	bc	ac	bc	0x01E4	ab	bc	ab	bc
0x00CC	ab	bc	1b	bc	0x01E8	2b	bb	ab	bb
0x00D0	ab	bb	ac	bb	0x01EC	ab	bc	ab	bc
0x00D4	fb	bc	1c	bc	0x01F0	2b e0	bb e0	ab 0e	bb 0e
0x00D8	ab	bb	ab	bb	0x01F4	ab	bc	ab	bc
0x00DC	fb	bc	1c	bc	0x01F8	ab	bc	ab	bc
0x00E0	ab	bc	ab	bc	0x01FC	ab	bb	ab	bb
0x00E4	fb	bb	1c	bb	0x0200	ab	bc	ab	bc
0x00E8	ab	bc	ab	bc	0x0204	ab	bb	ab	bb
0x00EC	fb	bb	1b	bb	0x0208	ab	bc	ab	bc
0x00F0	ab	bc	ab	bc	0x020C	ab	bc	ab	bc
0x00F4	ab	bc	ab	bc	0x0210	1c	bb	ab	bb
0x00F8	ab	bb	ab	bb	0x0214	ab	bc	ab	bc
0x00FC	ab	bc	ab	bc	0x0218	1b	bb	ab	bb
0x0100	ab	bb	ab	ff	0x021C	ab	bc	ab	bc
0x0104	ab	bc	ab	bc	0x0220	ab	bc	ab	bc
0x0108	ab	bc	ab	bc	0x0224	ab	bb	ab	bb
0x010C	ab	bb	ab	bb	0x0228	ab	bc	ab	bc
0x0110	ab	bc	ab	bc	0x022C	ab	bb	ab	bb
0x0114	ab	bb	ab	bb	0x0230	ab	bc	ab	bc
0x0118	ab	bc	ab	bc	0x0234	ff	ff	ff	ff

Memory access sequence:

Hit/Miss (4)	Op	Address	Addresses in binary	Data (Hex)	Hit/Miss (a)	Hit/Miss (b)	Hit/Miss (c)
Miss	LD	0x00E0	0000 0000 1110 0000	ab bc ab bc	Miss	Miss	Miss
Miss	LD	0x0050	0000 0000 0101 0000	1b bc 14 bf	Miss	Miss	Miss
Miss	ST	0x0040	0000 0000 0100 0000	01 01 01 01	Miss	Miss	Miss
Miss	ST	0x00A3	0000 0000 1010 0011	02 02 02 02	Miss	Miss	Miss
Miss	LD	0x01E7	0000 0001 1110 0111	abbc abbc	Miss	Miss	Miss
Miss	ST	0x01B4	0000 0001 1011 0100	dd dd dd dd	Miss	Miss	Miss
Miss	ST	0x01F0	0000 0001 1111 0000	e0 e0 0e 0e	Miss	Miss	Miss
Miss	LD	0x0130	0000 0001 0011 0000	abbcabbc	Miss	Miss	Miss
Miss	LD	0x0064	0000 0000 0110 0100	abbc1bbc	Miss	Miss	Miss
Miss	ST	0x00B0	0000 0000 1011 0000	15 bb 16 bb	Miss	Miss	Miss
Hit	ST	0x0066	0000 0000 0110 0110	98 11 22 33	Hit	Hit	Hit
Miss	ST	0x0094	0000 0000 1001 0100	Cc cc cc cc	Miss	Miss	Miss
Hit	LD	0x01E4	0000 0001 1110 0100	abbc abbc	Miss	Hit	Hit

Now consider three different 32-bytes caches shown below. Assume that each of the four caches was used independently to facilitate memory access for the sequence above. For each cache type, assume that the cache is initially empty. Block-size= 4 bytes. Cache capacity=32 bytes. Note: If any byte requested from the processor is not given in the table below, just write 'g' as data at that address. Calculate **hit rate** and **miss rate**. For part b and c and d use the initial values of memory, not the updated values after part a.

Assume that the least-recently used (LRU) scheme is used where appropriate, where the least recently inserted or accessed element is replaced on conflict. Also, when inserting an element into the cache, if there are multiple empty slots for one index, you should insert the new element into the (first available slot).

1]. Use the direct-mapped cache to facilitate memory access for the memory sequence above. You should fill in the binary form of the Tag values. Show the final contents of the cache in the table below, and compute the hit rate and miss rate. For cache write hits use write through and for cache write misses use write no allocate **[15 marks]**

update block directly in memory then bring it in cache

store in memory at same time

Block size = 4

No. of Blocks = 8

No. of offset bits = $\log_2 4$

= 2 bits

No. of set bits = $\log_2 8$

= 3 bits

No. of Tag bits = 16 - 2 - 3 = 11 bits

Set bits	V	Tag bits	Data			
			00	01	10	11
000	1	~~0000 0000 111~~ ~~0000 0000 010~~ 0000 0000 101	ab 01 02	bc 01 02	ab 01 02	bc 01 02
001	1	~~0000 0001 111~~ ~~0000 0000 011~~ 0000 0001 111	ab ab 98 ab	bc bc 11 bc	ab 1b 22 ab	bc bc 33 bc
010						
011						
100	1	~~0000 0000 101~~ ~~0000 0000 010~~ ~~0000 0001 111~~ 0000 0001 001	~~15~~ 1b e0 ab	~~bb~~ bc e0 bc	~~16~~ 14 0e ab	~~bb~~ ~~bf~~ 0e bc
101	1	~~0000 0001 101~~ 0000 0000 100	dd cc	dd cc	dd cc	dd cc
110						
111						

2].Use 4-way associative cache to facilitate memory access for the memory sequence above. You should fill in the binary form of the Tag values. Show the final contents of the cache in the table below, and compute the

1 set = 4 blocks

No. of Set bits = $\log_2$ (no. of sets) = 1 bit

No. of offset bits = will remain same

Q no:- 1

Total Requests = 13

Miss = 12

Hit = 1

Miss Rate = $\frac{12}{13} \times 100 = 92.31\ \%$

Hit Rate = $\frac{1}{13} \times 100 = 7.69\ \%$

write in memory if block entirly replace

change in cache then in memory

hit rate and miss rate. For cache write hits use write back and for cache write misses use write allocate. **You may add any column you find necessary. [15 marks]**

Set bits	V	Dirty	LRU	Tag bits	Data 00	01	10	11
0	1	0	0	~~0000 0000 1110 0~~	~~ab~~	~~bc~~	~~ab~~	~~bc~~
		1	1	0000 0001 1111 0	e0	e0	0e	0e
	1	0	0	~~0000 0000 0101 0~~	~~1b~~	~~bc~~	~~14~~	~~bf~~
			1	0000 0001 0011 0	ab	bc	ab	bc
	1	1	0	~~0000 0000 0100 0~~	~~01~~	~~01~~	~~01~~	~~01~~
			1	0000 0000 1011 0	15	bb	16	bb
	1	1	0	0000 0000 1010 0	02	02	02	02
1	1	0	0	0000 0001 1110 0	~~ab~~	~~bc~~	~~ab~~	~~bc~~
			1		ab	bc	ab	bc
	1	1	0	0000 0001 1011 0	dd	dd	dd	dd
	1	0	0	0000 0000 0110 0	~~ab~~	~~bc~~	~~1b~~	~~bc~~
		1	1		98	11	22	33
	1	1	0	0000 0000 1001 0	cc	cc	cc	cc

Q no:- 2

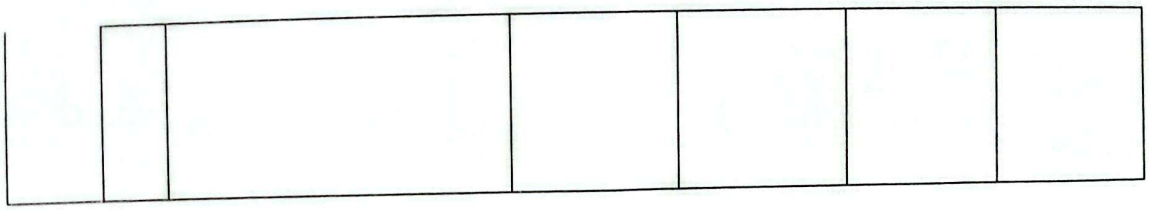
Total Requests = 13

Miss = 11

Hit = 2

Miss Rate = $\frac{11}{13} \times 100 = 84.61\ \%$

Hit Rate = $\frac{2}{13} \times 100 = 15.39\ \%$



3]. Use fully associative cache to facilitate memory access for the memory sequence above. You should fill in the binary form of the Tag values. Show the final contents of the cache in the table below, and compute the hit rate and miss rate. For cache write hits use write back and for cache write misses use write allocate. **You may add any column you find necessary. [15 marks]**

D	LRU	V	Tag bits				Data			
							00	01	10	11
0	0 6	1	0000	0000	1110	00	ab	bc	ab	bc
1	1 7						ab	bc	1b	bc
	2 0		0000	0000	0110	01	98	11	22	33
	3 1									
	4 2									
	5 3									
0	0 6	1	0000	0000	0101	00	1b	bc	14	bf
1	1 7									
	2 0		0000	0000	1011	00	15	bb	16	bb
	3 1									
	4 2									
	5 3									
1	0 6	1	0000	0000	0100	00	01	01	01	01
	1 7									
	2 0		0000	0000	1001	01	cc	cc	cc	cc
	3 1									
	4 2									
	5 3									
1	0 4	1	0000	0000	1010	00	02	02	02	02
	1 5									
	2 6									
	3 7									
	4									
0	0 6	1	0000	0001	1110	01	ab	bc	ab	bc
	1 0						ab	bc	ab	bc
	2									
	3									
	4									
	5									
1	0	1	0000	0001	1011	01	dd	dd	dd	dd
	1									
	2 6									
	3									
	4									
	5									

There will not be any Set bits in fully associative cache.

1	0 1 2 3 4 5	1	0000 0001 1111 00	e0	e0	0e	0e
0	0 1 2 3 4	1	0000 0001 0011 00	ab	bc	ab	bc

4]. Use 4-way associative cache to facilitate memory access for the memory sequence above. You should fill in the binary form of the Tag values. Show the final contents of the cache in the table below, and compute the hit rate and miss rate. For cache write hits use write through and for cache write misses use write no-allocate. Design Cache and fill the Cache with the memory sequence given above. **[15 Marks]**

5]. Consider a machine with a byte addressable main memory of 64Kbyte with 8byte Block size. Assume that direct mapped cache consisting 32 lines. Write answer of the following questions **[20 Marks]**

no. of blocks → on next page

a). How 64Kbyte Main memory address divided into tag, block/line number, and byte number

b). Into what line would bytes with each of the following addresses be stored?

0001000100011011 : line = 00011

1100001100110100 : line = 00110

1101000000011101 : line = 00011

1011111101010001: line = 01010

FA02: 1111 1010 0000 0010 line = 00000

EC4F: 1110 1100 0100 1111 line = 01001

c). Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of other bytes stored along with it? → on next page

d). How many total bytes of memory can be stored in the cache? → on next page

6]. Assume a **4-way set associative** cache of size **32KB**, each block having four 64-bit words. The system is based on 64-bit physical address and uses byte addressing. Find the total number of offsets, index and tag bits. **[20 Marks]**

- Main Memory Size : $2^{64} = 2^{24} \times 2^{40} B = 16777216$ TB
- Tag Bits: 64 − 5 − 8 = 51 bits
- Offset Bits: $\log_2 32 = 5$ bits
- Index Bits: $\log_2 256 = 8$ bits

Block size = 4 x 64 bits = 4 x 8 B = 32 Bytes

capacity = 32 KB = 32×2^{10} = 32768 Bytes

no. of blocks = $\frac{32768}{32}$ = 1024

no. of sets = $\frac{1024}{4}$ = 256 → set = 4 Bytes

Total Requests = 13

Miss = 11

Hit = 2

Miss Rate = $\frac{11}{13} \times 100 = 84.61\ \%$

Hit Rate = $\frac{2}{13} \times 100 = 15.39\ \%$

Qno:- 5 (a)

Memory Size = 64 KB = $2^6 \times 2^{10}$ B = 2^{16} B

Address bus size = 16 bits

So each memory address will be of 16 bit

⊙ No. of offset bits = $\log_2$ block size

= $\log 2^8$ = 3 bits

⊙ No. of set bits = $\log_2$ no. of blocks

= $\log 2^{32}$ = 5 bits

⊙ No. of Tag bits = 16 − 5 − 3 = 8 bits

(c) As each line in cache is of 8 bytes. So according to special locality it will bring 8 bytes from memory with it all time.

Addresses of other Bytes are as follows:

0001 1010 0001 1000
0001 1010 0001 1001
0001 1010 0001 1010 → required address
0001 1010 0001 1011
0001 1010 0001 1100
0001 1010 0001 1101
0001 1010 0001 1110
0001 1010 0001 1111

(d)

capacity = no. of block x size of block

= 32 x 8

= 256 Bytes

Question No:- 4

Set Bits	V	Dirty	LRU	Tag bits	Data			
					00	01	10	11
0	1	0 1	0 1 2 3 2	0000 0000 1110 0 0000 0001 1111 0	ab e0	bc e0	ab 0e	bc 0e
	1	0	0 1 2 3 0 1	0000 0000 0101 0 0000 0001 0011 0	1b ab	bc bc	14 ab	bf bc
	1	1	0 1 2 3 0	0000 0000 0100 0 0000 0000 1011 0	01 15	01 bb	01 16	01 bb
	1	1	0 1 2 3	0000 0000 1010 0	02	02	02	02
1	1	0	0 1 2 3 0	0000 0001 1110 0	ab ab	bc bc	ab ab	bc bc
	1	1	0 1 2 3	0000 0001 1011 0	dd	dd	dd	dd
	1	0 1	0 1 2	0000 0000 0110 0	ab 98	bc 11	1b 22	bc 33
	1	1	0 1	0000 0000 1001 0	cc	cc	cc	cc

Total Requests = 13

Miss = 11 Hit = 2

Miss Rate = $11/13 \times 100 = 84.61\ \%$

Hit Rate = $2/13 \times 100 = 15.39\ \%$

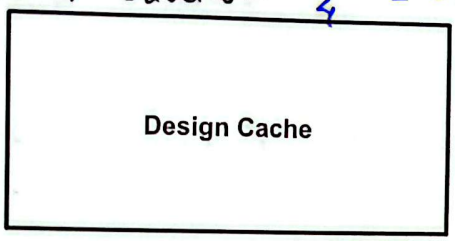
7]. Assume a direct-mapped cache of 256Byte and data access sequence given below where the memory and its contents are shown in the table on the right. Assume 4byte addresses and a 4-byte data block. You are required to design a cache and calculate a Cache Hit and Miss %.

Operations	Address
Read	0x01B0
Read	0x00C8
Read	0x0214
Read	0x0234
Read	0x0160
Read	0x0030
Read	0x018C
Read	0x0068
Read	0x0062
Read	0x01A4
Read	0x01CC
Read	0x01A3
Read	0x0230
Read	0x01CC
Read	0x0212

Block size = 4 B

Capacity = 256 B

no. of blocks = $\frac{256}{4}$ = 64



no. of offset bits = $\log_2 4$ = 2 bits

no. of set bits = $\log_2 64$ = 6 bits

Hits: 1

Hit rate: $\frac{1}{15} \times 100 = 6.67\ \%$

Miss: 14

Miss rate: $\frac{14}{15} \times 100 = 93.33\ \%$

8]. A computer system has a two-level cache hierarchy consisting of a Level 1 (L1) cache and a Level 2 (L2) cache. The system services memory requests with the following characteristics:

- L1 cache serves 3000 memory requests, out of which 2550 requests result in a cache hit.
- L2 cache serves 2800 memory requests, out of which 2520 requests result in a cache hit.
- L1 cache access time: 1 nanosecond (ns)
- L2 cache access time: 10 ns
- Main memory access time (miss penalty): 100 ns

Calculate the following:

1. The percentage of memory accesses that result in a cache hit in the L1 cache.
2. The percentage of memory accesses that result in a cache hit in the L2 cache.
3. The number of memory accesses that result in a cache miss in both the L1 and L2 caches.
4. The effective access time (EAT) for this cache hierarchy.

① Percentage of cache hit for L1 = $\frac{2550}{3000} \times 100 = 85\%$

② Percentage of cache hit for L2 = $\frac{2520}{2800} \times 100 = 90\ \%$

Q2

(3) Request Miss from L1 will go to cache L2:

L1 cache Miss = 3000 − 2550
= 450
= 450 − (450 × L2 hit Rate)
= 450 − (450 × 0.9)
= 450 − 405
= 45

for L2 Miss Requests = 2800 − 2520
= 280

(4) EAT = (L1 hit rate × L1 access time) +
L1 Miss rate × [L1 access time + (L2 hit rate × L2 access time)
+ L2 miss rate × [L2 access time + Memory access Time)]

= (0.85 × 1) + 0.15 [1 + (0.9 × 10) + (0.1 (10 + 100))]
= 0.85 + 0.15 (1 + 9 + 11)
= 4 ns

Question No:-7

Set Bits	V	Tag Bits	Data			
			00	01	10	11
000000						
⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮
000100	1	0000 0000 0000 0000 0000 0010	1c	bb	ab	bb
000101	1	0000 0000 0000 0000 0000 0010	ab	bc	ab	bc
⋮	⋮	⋮	⋮	⋮	⋮	⋮
001100	1	~~0000 0000 0000 0000 0000 0000~~ 0000 0000 0000 0000 0000 0010	~~1b~~ ab	~~bb~~ bc	~~1b~~ ab	~~bc~~ bc
001101		0000 0000 0000 0000 0000 0010	~~ff~~	~~ff~~	~~ff~~	~~ff~~
⋮	⋮	⋮	⋮	⋮	⋮	⋮
011000	1	~~0000 0000 0000 0000 0000 0001~~ 0000 0000 0000 0000 0000 0000	~~ab~~ 1b	~~bc~~ bb	~~ab~~ 1b	~~bc~~ bb
⋮	⋮	⋮	⋮	⋮	⋮	⋮
011010	1	0000 0000 0000 0000 0000 0000	ab	bc	ab	bc
⋮	⋮	⋮	⋮	⋮	⋮	⋮
100011	1	0000 0000 0000 0000 0000 0001	ab	bb	ab	bb
⋮	⋮	⋮	⋮	⋮	⋮	⋮
101000	1	0000 0000 0000 0000 0000 0001	ab	bb	ab	bb
101001	1	0000 0000 0000 0000 0000 0001	ab	bc	ab	bc
⋮	⋮	⋮	⋮	⋮	⋮	⋮
101100	1	0000 0000 0000 0000 0000 0001	ab	bc	ab	bc
⋮	⋮	⋮	⋮	⋮	⋮	⋮
110010	1	0000 0000 0000 0000 0000 0000	ab	bc	ac	bc
110011	1	0000 0000 0000 0000 0000 0001	~~ab~~ ab	~~bc~~ bc	~~ab~~ ab	~~bc~~ bc
⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮
111111						