Course:	Computer Organization - ENCM 369
Lab #:	10
Instructor:	N. Bartley
Group Submission for:	B02
Submitted by:	Aarsh Shah
UCID:	30150079
Partner:	William Fraser
UCID:	30158991
Date Submitted:	06-Apr-2023

Exercise A

Address	Tag	Set	Action
0x0040_5b50	0x00405	724	I-cache hit—no I-cache update
0x0040_5b54	0x00405	725	I-cache miss—instruction 0x000a_0533 is copied into instruction field in set 725, V-bit in that set is changed to 1, tag to 0x00405.
0x0040_5b58	0x00405	726	I-cache miss – since tags don't match. The instruction 0x004a_0a13 is copied into the instruction field in set 726, v-bit stays as 1 and the tag is changed to 0x00405
0x0040_4b58	0x00404	726	I-cache miss – tags no longer match as it was just changed for the previous instruction. The instruction 0x0004_2283 is copied into the instruction field in set 726. V-bit stays as 1. The tag is changed to 0x00404
0x0040_4b5c	0x00404	727	I-cache hit – no I-cache update
0x0040_4b60	0x00404	728	I-cache hit – no I-cache update
0x0040_4b64	0x00404	729	I-cache hit – no I-cache update
0x0040_5b5c	0x00405	727	I-cache miss – since tags don't match. The instruction 0x800f_f0ef is copied into the instruction field in set 727, v-bit stays as 1 and the tag is changed to 0x00405
0x0040_5b60	0x00405	728	I-cache miss – since tags don't match. The instruction 0xff5a_1ae3 is copied into the instruction field in set 728, v-bit stays as 1 and the tag is changed to 0x00405
0x0040_5b54	0x00405	725	I-cache hit – no I-cache update
0x0040_5b58	0x00405	726	I-cache miss – since tags don't match. The instruction 0x004a_0a13 is copied into the instruction field in set 726, v-bit stays as 1 and the tag is changed to 0x00405
0x0040_4b58	0x00404	726	I-cache miss – tags no longer match as it was just changed for the previous instruction. The instruction 0x0004_2283 is copied into the instruction field in set 726. V-bit stays as 1. The tag is changed to 0x00404
0x0040_4b5c	0x00404	727	I-cache miss – since tags don't match. The instruction 0xfc02_8313 is copied into the instruction field in set 727, v-bit stays as 1 and the tag is changed to 0x00404
0x0040_4b60	0x00404	728	I-cache miss – since tags don't match. The instruction 0x0065_2023 is copied into the instruction field in set 728, v-bit stays as 1 and the tag is changed to 0x00404
0x0040_4b64	0x00404	729	I-cache hit – no I-cache update
0x0040_5b5c	0x00405	727	I-cache miss – since tags don't match. The instruction 0x800f_f0ef is copied into the instruction field in set 727, v-bit stays as 1 and the tag is changed to 0x00405
0x0040_5b60	0x00405	728	I-cache miss – since tags don't match. The instruction 0xff5a_1ae3 is copied into the instruction field in set 728, v-bit stays as 1 and the tag is changed to 0x00405
0x0040_5b64	0x00405	729	I-cache miss – since tags don't match. The instruction 0x0000_0a33 is copied into the instruction field in set 729, v-bit stays as 1 and the tag is changed to 0x00405

Exercise B

3. Suppose the specification for the cache of textbook Figure 8.12 is changed. The block size is now supposed to be 8 words, and the capacity has been increased to a much more practical size of 16 KiB.

(a) What is S, the number of sets?

$16 \text{ KiBi} = 16 \cdot 2^{10} = 16384$ bytes

Thus $S = 16384 \text{ bytes} \times \frac{1 \text{ word}}{4 \text{ bytes}} \times \frac{1 \text{ set}}{8 \text{ words}} = 512$ sets

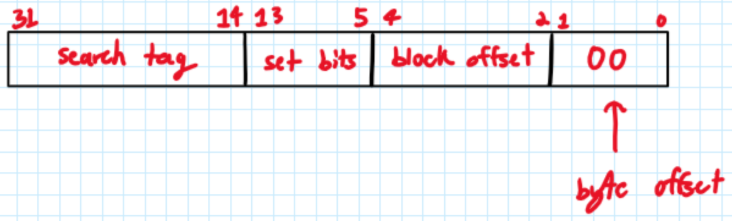
(b) How should addresses be split into parts? Show this with a diagram like the previously given examples—include bit numbers marking the boundaries between parts.

$\log_2(512) = 9$ set bits

$\log_2(8) = 3$ block offset bits

A 32-bit word is 4-bytes, so the byte offset is $\log_2(4) = 2$

tag width = address width − number of set bits − block offset width − byte offset width
$= 32 - 9 - 3 - 2$
$= 18$



(c) The cache will be built using SRAM cells, one SRAM cell for every V-bit, every bit within a tag, and every bit within a block of data or instruction words. How many SRAM cells are needed for the whole cache? Show your work carefully.

$$512 \text{ sets} \times \left(\underset{\text{V-bit}}{1} + \underset{\text{tag}}{18} + \underset{\text{data/Instruction}}{32}\right)$$

$512 \times 51 = 26{,}112$ SRAM cells are needed.

4. The term "64-bit processor" usually describes a processor in which general-purpose registers are 64 bits wide, and memory addresses *within the processor core and in pointer variables* are managed as 64-bit patterns. However, 2^{64} bytes of DRAM is enormously larger than the amount of DRAM that can practically be connected to a single processor chip, so a typical 64-bit design might use only the least significant 44 bits of an address to access caches and main memory. This is illustrated in Figure 6.

   Do the following calculations for the computer of Figure 6. We'll assume direct-mapped design for all three caches, and we'll consider the word size to be *64 bits*.

   (a) The capacity of the L1 D-cache is 32 KiB. The block size is 64 *bytes*. Draw a diagram to show how a 44-bit address input to the cache would be split into these fields: tag, set bits, block offset, and byte offset. Indicate exactly how wide each field is.

Block size = 64 bytes $\times \frac{1 \text{ word}}{8 \text{ bytes}}$ = 8 words

32 KiB = $32 \cdot 2^{10}$ = 32768 Bytes

32,768 bytes $\times \frac{1 \text{ word}}{8 \text{ bytes}} \times \frac{1 \text{ set}}{8 \text{ words}}$ = 512 sets

$\log_2(512)$ = 9 set bits

Block offset, 8 word block size $\Rightarrow \log_2(8)$ = 3 bits

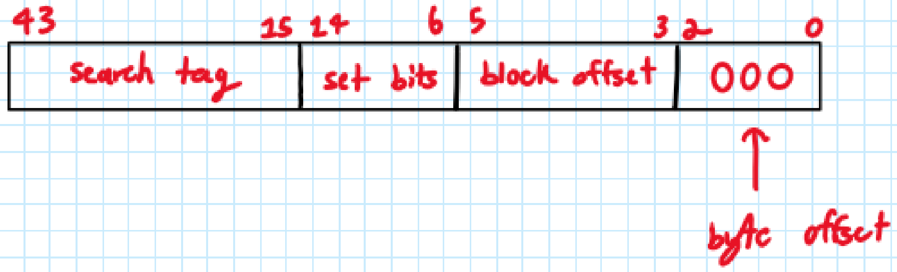
Byte offset $\Rightarrow$ Since each word is 64 bits

There are $\frac{64 \text{ bits}}{8}$ = 8 bytes in each word

Thus byte offset = $\log_2(8)$ = 3 bits

tag width = 44 - 9 - 3 - 3
= 29 bits

Thus:



(b) Repeat part (a) for the L2 cache, which has a capacity of 4 MiB. The block size is again 64 *bytes*.

$$\text{Block size} = 64 \text{ bytes} \times \frac{1 \text{ word}}{8 \text{ bytes}} = 8 \text{ words}$$

$$4 \text{MiB} = 4 \cdot 2^{20} = 4{,}194{,}304 \text{ bytes}$$

$$4{,}194{,}304 \text{ bytes} \times \frac{1 \text{ word}}{8 \text{ bytes}} \times \frac{1 \text{ set}}{8 \text{ words}} = 65{,}536 \text{ sets}$$

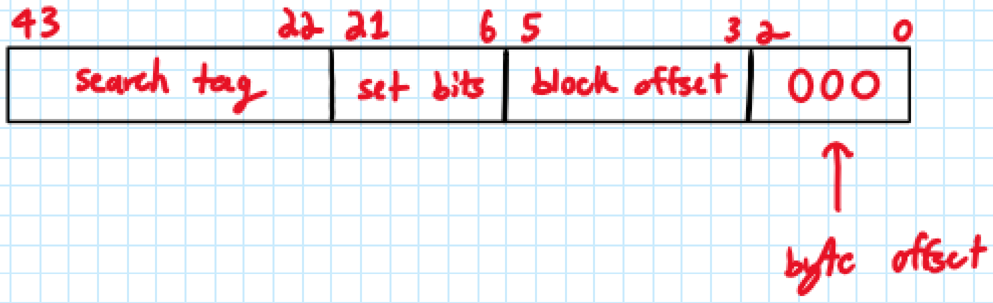
$$\log_2(65{,}536) = 16 \text{ set bits}$$

Block offset ⇒ 3 bits same as above

Byte offset ⇒ 3 bits same as above

$$\text{tag width} = 44 - 16 - 3 - 3$$
$$= 22 \text{ bits}$$

Thus:



(c) Use your results from (b) to determine how many one-bit SRAM cells are needed for all the V-bits, tags, and data/instruction blocks in the L2 cache.

$$65{,}536 \text{ sets} \times \left(\overset{1}{\text{v-bit}} + \overset{22}{\text{tag width}} + \overset{64}{\text{data/information}}\right)$$

$65{,}536 \times 86 = 5{,}701{,}632$ SRAM cells.

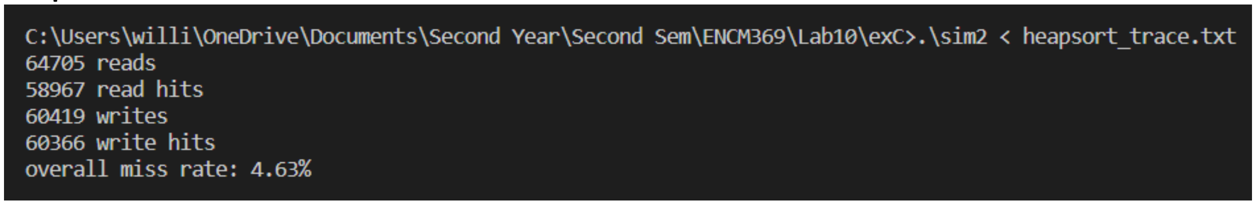
Exercise C

Source Code *sim2.c*:

```c
// sim2.c

#include <stdio.h>
#include <stdlib.h>

int read_one_line(unsigned *p)
{
  int nscan, rw;
  char buf[2];

  nscan = scanf("%1s%x", buf, p);
  if (nscan == EOF)
    return 'e'; /* indicate end-of-file */
  else if (nscan != 2)
  {
    fprintf(stderr, "Format error in input stream.\n");
    exit(1);
  }

  rw = buf[0];
  if (rw != 'r' && rw != 'w')
  {
    fprintf(stderr, "Read/write character was neither r nor w.\n");
    exit(1);
  }
  return rw;
}

// These two arrays keep track of all the V-bits and stored tags in
// the array.  We don't need an array for data to count hits and misses.
// Because these arrays are external variables, it's safe to assume
// that they will be initialized to all zeros before main starts.
char v_bit[128];
unsigned stored_tag[128];

int main(void)
{
  int read_count = 0, read_hits = 0;
  int write_count = 0, write_hits = 0;
  int access_count, miss_count;
  int rw;
  unsigned address, search_tag, set_bits;
  int hit;

  while (1)
  {
    rw = read_one_line(&address);
    if (rw == 'e')
      break;

    set_bits = (address & 0xfe0) >> 5; // bits 11-5
    search_tag = address >> 12;       // bits 31-12

    // Note: Next line results in either hit == 1 or hit == 0.
    hit = v_bit[set_bits] == 1 && stored_tag[set_bits] == search_tag;
    if (rw == 'r')
    {
      read_count++;
      read_hits += hit;
    }
    else
    {
      write_count++;
      write_hits += hit;
    }
    if (!hit)
    { // On a miss, update V-bit and search_tag.
      v_bit[set_bits] = 1;
      stored_tag[set_bits] = search_tag;
    }
  }

  printf("%d reads\n", read_count);
  printf("%d read hits\n", read_hits);
  printf("%d writes\n", write_count);
  printf("%d write hits\n", write_hits);

  access_count = read_count + write_count;
  miss_count = access_count - read_hits - write_hits;
  printf("overall miss rate: %.2f%%\n",
         100.0 * (double)miss_count / access_count);

  return 0;
}
```

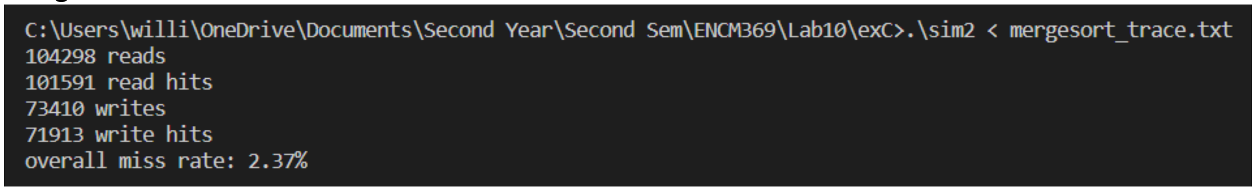
Outputs for sim2:

Heapsort:



```
C:\Users\willi\OneDrive\Documents\Second Year\Second Sem\ENCM369\Lab10\exC>.\sim2 < heapsort_trace.txt
64705 reads
58967 read hits
60419 writes
60366 write hits
overall miss rate: 4.63%
```

Mergesort:



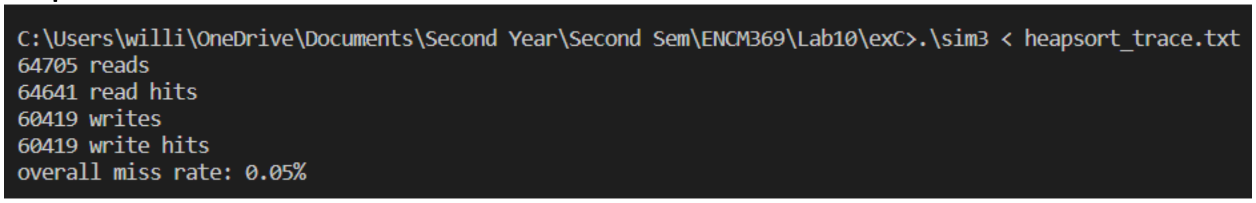
```
C:\Users\willi\OneDrive\Documents\Second Year\Second Sem\ENCM369\Lab10\exC>.\sim2 < mergesort_trace.txt
104298 reads
101591 read hits
73410 writes
71913 write hits
overall miss rate: 2.37%
```

Source Code *sim3.c*:

```
// sim3.c

#include <stdio.h>
#include <stdlib.h>

int read_one_line(unsigned *p)
{
  int nscan, rw;
  char buf[2];

  nscan = scanf("%1s%x", buf, p);
  if (nscan == EOF)
    return 'e'; /* indicate end-of-file */
  else if (nscan != 2)
  {
    fprintf(stderr, "Format error in input stream.\n");
    exit(1);
  }

  rw = buf[0];
  if (rw != 'r' && rw != 'w')
  {
    fprintf(stderr, "Read/write character was neither r nor w.\n");
    exit(1);
  }
  return rw;
}

// These two arrays keep track of all the V-bits and stored tags in
// the array.  We don't need an array for data to count hits and misses.
// Because these arrays are external variables, it's safe to assume
// that they will be initialized to all zeros before main starts.
char v_bit[1024];
unsigned stored_tag[1024];

int main(void)
{
  int read_count = 0, read_hits = 0;
  int write_count = 0, write_hits = 0;
  int access_count, miss_count;
  int rw;
  unsigned address, search_tag, set_bits;
  int hit;
```

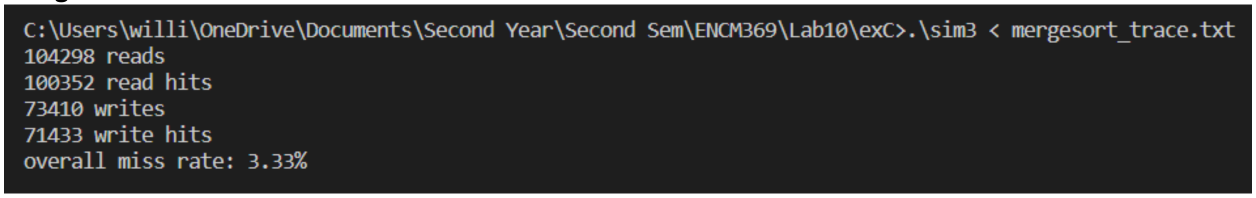
```c
  while (1)
  {
    rw = read_one_line(&address);
    if (rw == 'e')
      break;

    set_bits = (address & 0x7fe) >> 5; // bits 14-5
    search_tag = address >> 15;        // bits 31-15

    // Note: Next line results in either hit == 1 or hit == 0.
    hit = v_bit[set_bits] == 1 && stored_tag[set_bits] == search_tag;
    if (rw == 'r')
    {
      read_count++;
      read_hits += hit;
    }
    else
    {
      write_count++;
      write_hits += hit;
    }
    if (!hit)
    { // On a miss, update V-bit and search_tag.
      v_bit[set_bits] = 1;
      stored_tag[set_bits] = search_tag;
    }
  }

  printf("%d reads\n", read_count);
  printf("%d read hits\n", read_hits);
  printf("%d writes\n", write_count);
  printf("%d write hits\n", write_hits);

  access_count = read_count + write_count;
  miss_count = access_count - read_hits - write_hits;
  printf("overall miss rate: %.2f%%\n",
         100.0 * (double)miss_count / access_count);

  return 0;
}
```

Outputs for sim3:
Heapsort:



```
C:\Users\willi\OneDrive\Documents\Second Year\Second Sem\ENCM369\Lab10\exC>.\sim3 < heapsort_trace.txt
64705 reads
64641 read hits
60419 writes
60419 write hits
overall miss rate: 0.05%
```

MergeSort:



```
C:\Users\willi\OneDrive\Documents\Second Year\Second Sem\ENCM369\Lab10\exC>.\sim3 < mergesort_trace.txt
104298 reads
100352 read hits
73410 writes
71433 write hits
overall miss rate: 3.33%
```

Question in part 2:
Yes, as the miss rates significantly decreased in sim2, this indicates that there is significant special locality. We know this as the simulated caches in sim1 and sim2 contain the same amount of data, but sim2 is implemented with eight-word blocks, rather than one-word blocks as in sim1. The fact that the miss rate is lower indicates that when a given block was accessed, multiple words in that block would be used by the processor, this is known as special locality.

Exercise D

a) Word size: 64 bits $= 8 \frac{\text{bytes}}{\text{word}}$

Block size: 64 bytes $= 8 \frac{\text{words}}{\text{block}}$

Capacity: 32KiB = 32 768 bytes

Number of ways: $8 \frac{\text{blocks}}{\text{set}}$

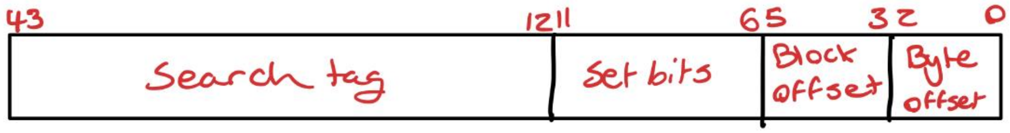
$$S = \frac{C}{N \cdot Bpl} = \frac{32768}{8 \cdot 8 \cdot 8} = 64 \text{ sets}$$

$\log_2 64 =$ 6 set bits

$\log_2 8 =$ 3 bit byte offset

$\log_2 8 =$ 3 bit block offset

tag bits $= 44 - 6 - 3 - 3 = 32$ tag bits



b) Block size: $8 \frac{\text{words}}{\text{block}}$

Word size: $8 \frac{\text{bytes}}{\text{word}}$

Capacity: 4 MiB = 4194304 bytes

N: $16 \frac{\text{blocks}}{\text{set}}$

$$S = \frac{C}{N \times Bpl} = \frac{4194304}{16 \cdot 8 \cdot 8}$$

$= 4096$ sets

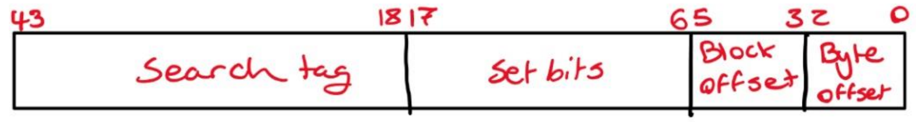
$\log_2 4096 =$ 12 set bits

$\log_2 8 =$ 3 bit byte offset

$\log_2 8 =$ 3 bit block offset

$44 - 12 - 3 - 3$

$= 26$ tag bits



c) 4096 sets × (1 v-bit + 26 tag width + 64 data)

= 372736 SRAM cells