Course: ENCM 369
Lab Section: B03
Lab 10
Student Name: Mitchell Sawatzky
Student Name: Connor Newman
Date Submitted: April 4, 2016

Exercise A

Address	Tag	Set	Action
0x0040_2634	0x00402	397	I-cache hit—no I-cache update
0x0040_2638	0x00402	398	I-cache miss—instruction 0x0200_2021 is copied into “instruction” field in set 398, V-bit in that set is changed to 1, tag to 0x00402
0x0040_263c	0x00402	399	I-cache miss-instruction 0x0c10_0590 is copied into “instruction” field in set 399, V-bit in that set is changed to 1, tag to 0x00402
0x0040_1640	0x00401	400	I-cache hit-no I-cache update
0x0040_1644	0x00401	401	I-cache hit-no I-cache update
0x0040_1648	0x00401	402	I-cache hit-no I-cache update
0x0040_164c	0x00401	403	I-cache miss-instruction 0x03e0_0008 is copied into “instruction” field in set 403, tag is changed to 0x00401
0x0040_2640	0x00402	400	I-cache miss-instruction 0x2610_0004 is copied into “instruction” field in set 400, tag is changed to 0x00402
0x0040_2644	0x00402	401	I-cache miss-instruction 0x1611_fffc is copied into “instruction” field in set 401, tag is changed to 0x00402
0x0040_2638	0x00402	398	I-cache hit-no I-cache update
0x0040_263c	0x00402	399	I-cache hit-no I-cache update
0x0040_1640	0x00401	400	I-cache miss-instruction 0x8c88_0000 is copied into “instruction” field in set 400, tag is changed to 0x00401
0x0040_1644	0x00401	401	I-cache miss-instruction 0x0008_48c0 is copied into “instruction” field in set 401, tag is changed to 0x00401
0x0040_1648	0x00401	402	I-cache hit-no I-cache update
0x0040_164c	0x00401	403	I-cache hit-no I-cache update
0x0040_2640	0x00402	400	I-cache miss-instruction 0x2610_0004 is copied into “instruction” field in set 400, tag is changed to 0x00402
0x0040_2644	0x00402	401	I-cache miss-instruction 0x1611_fffc is copied into “instruction” field in set 401, tag is changed to 0x00402
0x0040_2648	0x00402	402	I-cache miss-instruction 0x2652_0000 is copied into “instruction” field in set 402, tag is changed to 0x00402

Exercise B

3. (a)

$$8KB = 8 * 2^{10} bytes = S * 2^3 \frac{words}{block} * 2^2 \frac{bytes}{word}$$

$$S = \frac{8 * 2^{10} bytes}{2^3 \frac{words}{block} * 2^2 \frac{bytes}{word}} = 2^8 blocks = 256\ blocks$$

(b) Byte offset: A 32-bit word is 4 bytes, so the width is $\log_2 4 = 2$

Block offset: Each block has 8 words, so the width is $\log_2 8 = 3$

Set bits: S = 256, so we need $\log_2 256 = 8$

Search tag: 32-8-3-2=19

<table>
<tr><td>31</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>13</td><td>12</td><td></td><td></td><td></td><td></td><td></td><td></td><td>5</td><td>4</td><td></td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="19">search</td><td colspan="8">set</td><td colspan="3">block</td><td colspan="2">byte</td></tr>
</table>

(c) There is 1 V-bit per set, and S = 256, so 256 cells
The search tag is 19 bits long, so there are 19 cells
The block offset is 3 bits long, so overall there are 19+3+1=23 cells per set
Overall, SRAM cells = 256 * 23 = 5888 cells needed.

4. (a) Byte offset: A 64-bit word is 8 bytes, so the width is $\log_2 8 = 3$
Block offset: Each block has 64 bytes = 8 words, so the width is $\log_2 8 = 3$
Set bits:

$$32KB = 32 * 2^{10} bytes = S * 2^3 \frac{words}{block} * 2^3 \frac{bytes}{word}$$
$$S = \frac{32 * 2^{10} bytes}{2^3 \frac{words}{block} * 2^3 \frac{bytes}{word}} = 2^9 blocks = 512\ blocks$$

S = 512, so we need $\log_2 512 = 9$
Search tag: 32-9-3-3=17

31																15	14								6	5		3	2		0
search																	set									block			byte		

(b) From part (a),
Byte offset: 3
Block offset: 3
Set bits:

$$2MB = 2 * 2^{20} bytes = S * 2^3 \frac{words}{block} * 2^3 \frac{bytes}{word}$$
$$S = \frac{2 * 2^{20} bytes}{2^3 \frac{words}{block} * 2^3 \frac{bytes}{word}} = 2^{15} blocks = 32768\ blocks$$

S = 32768, so we need $\log_2 32768 = 15$
Search tag: 32-15-3-3=11

31										21	20														6	5		3	2		0
search											set															block			byte		

(c) There is 1 V-bit per set, and S=32768, so 32768 cells
The search tag is 11 bits long, so there are 11 cells
The block offset is 3 bits long, so overall there are 11+3+1=15 cells per set
Overall, SRAM cells = 32768 * 15 = 491520 cells needed.

Exercise C

Part II:

Byte offset: A 32-bit word is 4 bytes, so the width is $\log_2 4 = 2$
Block offset: Each block has 4 words, so the width is $\log_2 4 = 2$
S=1024, so we need $\log_2 1024 = 10$
Search tag: 32-10-2-2=18

<table>
<tr><td>31</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>14</td><td>13</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="18">search</td><td colspan="10">set</td><td colspan="2">block</td><td colspan="2">byte</td></tr>
</table>

Terminal Output:

```
Mitchell@ttys000 12:30 {0} [exC]$ gcc sim1.c
Mitchell@ttys000 12:32 {0} [exC]$ ./a.out < heapsort_trace.txt
64705 reads
49791 read hits
60419 writes
60401 write hits
overall miss rate: 11.9%
Mitchell@ttys000 12:33 {0} [exC]$ ./a.out < mergesort_trace.txt
104298 reads
88429 read hits
73410 writes
64076 write hits
overall miss rate: 14.2%
Mitchell@ttys000 12:33 {0} [exC]$ gcc sim2.c
Mitchell@ttys000 12:33 {0} [exC]$ ./a.out < heapsort_trace.txt
64705 reads
63955 read hits
60419 writes
60419 write hits
overall miss rate: 0.6%
Mitchell@ttys000 12:33 {0} [exC]$ ./a.out < mergesort_trace.txt
104298 reads
102187 read hits
73410 writes
72039 write hits
overall miss rate: 2.0%
```

sim2.c

```
/* sim1.c
 * ENCM 369 Winter 2016 Lab 10 Exercise C
 * Author: S. Norman
 *
 * If you build an executable using gcc -Wall sim1.c -o sim1
 * you can run it by redirecting input to come from a data file,
 * as in
 *         ./sim1 < heapsort_trace.txt
 */
```

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

int read_one_line(unsigned *p)
/* Read one line of the input stream.
 * Return value is normally 'r' or 'w' to indicate read or write.
 * In that case, *p contains the address read from the input line.
 * Return value is 'e' to indicate that input failed at the end of
 * the input stream.
 */
{
  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
// it will be initialized to all zeros before main starts.
char v_array[1024];
unsigned tag_array[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, tag, block, set;
  int hit;

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

    block = (address & 0xc) >> 2; // bits 3-2
    set = (address & 0x3ff0) >> 4; // bits 13-4
    tag = address >> 14;          // bits 31-14

    // Note: Next line results in either hit == 1 or hit == 0.
    hit = v_array[set] == 1 && tag_array[set] == tag;
    if (rw == 'r') {
      read_count++;
      read_hits += hit;
    }
    else {
      write_count++;
      write_hits += hit;
    }
    if (!hit) {                 // On a miss, update V-bit and tag.
      v_array[set] = 1;
      tag_array[set] = 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: %.1f%%\n",
         100.0 * (double) miss_count / access_count);

  return 0;
}
```

Yes. With a block size of 1 word, and a high degree of spatial locality, the cache must contain a lot of values that get re-loaded. However when the block size is increased, there are more cache hits with values close together.

Part III:

Byte offset: A 32-bit word is 4 bytes, so the width is $\log_2 4 = 2$

Block offset: Each block has 4 words, so the width is $\log_2 4 = 2$

Set bits:

$$8KB = 8 * 2^{10} bytes = S * 2^2 \frac{words}{block} * 2^2 \frac{bytes}{word}$$

$$S = \frac{8 * 2^{10} bytes}{2^2 \frac{words}{block} * 2^2 \frac{bytes}{word}} = 2^9 blocks = 512\ blocks$$

Search tag: 32-9-2-2=19

<table>
<tr><td>31</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>13</td><td>12</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="19">search</td><td colspan="9">set</td><td colspan="2">block</td><td colspan="2">byte</td></tr>
</table>

Terminal Output:

```
Mitchell@ttys000 12:39 {0} [exC]$ gcc sim3.c
Mitchell@ttys000 12:49 {0} [exC]$ ./a.out < heapsort_trace.txt
64705 reads
62165 read hits
60419 writes
60419 write hits
overall miss rate: 2.0%
Mitchell@ttys000 12:49 {0} [exC]$ ./a.out < mergesort_trace.txt
104298 reads
101724 read hits
73410 writes
71840 write hits
overall miss rate: 2.3%
```

sim3.c

```
/* sim1.c
 * ENCM 369 Winter 2016 Lab 10 Exercise C
 * Author: S. Norman
 *
 * If you build an executable using gcc -Wall sim1.c -o sim1
 * you can run it by redirecting input to come from a data file,
 * as in
 *         ./sim1 < heapsort_trace.txt
 */
```

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

int read_one_line(unsigned *p)
/* Read one line of the input stream.
 * Return value is normally 'r' or 'w' to indicate read or write.
 * In that case, *p contains the address read from the input line.
 * Return value is 'e' to indicate that input failed at the end of
 * the input stream.
 */
{
  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
// it will be initialized to all zeros before main starts.
char v_array[8192];
unsigned tag_array[8192];

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, tag, block, set;
  int hit;

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

    block = (address & 0xc) >> 2; // bits 3-2
    set = (address & 0x1ff0) >> 4; // bits 12-4
    tag = address >> 13;          // bits 31-13

    // Note: Next line results in either hit == 1 or hit == 0.
    hit = v_array[set] == 1 && tag_array[set] == tag;
    if (rw == 'r') {
      read_count++;
      read_hits += hit;
    }
    else {
      write_count++;
      write_hits += hit;
    }
    if (!hit) {                 // On a miss, update V-bit and tag.
      v_array[set] = 1;
      tag_array[set] = 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: %.1f%%\n",
         100.0 * (double) miss_count / access_count);
  return 0;
}
```