Exercise A

Tags and Sets

Address	Tag	Set
0x0040_17c0	0x00401	496
0x0040_17c4	0x00401	497
0x0040_17c8	0x00401	498
0x0040_17cc	0x00401	499
.	.	.
.	.	.
.	.	.
0x0040_27b8	0x00402	494
0x0040_27bc	0x00402	495
0x0040_27c0	0x00402	496
0x0040_27c4	0x00402	497
0x0040_27c8	0x00402	498
0x0040_27cc	0x00402	499

I-Cache State

Set	Valid	Tag	Instruction
494	1	0x00402	0x1253_0004
495	1	0x00402	0x0240_2021
496	1	0x00401	0x8c99_0000
497	1	0x00402	0x2652_0004
498	1	0x00402	0x1653_fffc
499	1	0x00402	0x2414_0000

Instruction Fetches on the Next Page…

Instruction Fetches

Address	Tag	Set	Action
0x0040_27b8	0x00402	494	I-cache hit – no I-cache update
0x0040_27bc	0x00402	495	I-cache miss – instruction 0x0240_2021 is copied into instruction field in set 495, V-bit in that set is changed to 1, tag to 0x00402
0x0040_27c0	0x00402	496	I-cache miss – instruction 0x0c10_0f50 is copied into instruction field in set 496, V-bit in that set is changed to 1, tag to 0x00402
0x0040_17c0	0x00401	496	I-cache miss – instruction 0x8c99_0000 is copied into instruction field in set 496, V-bit in that set is changed to 1, tag to 0x00401
0x0040_17c4	0x00401	497	I-cache hit – no I-cache update
0x0040_17c8	0x00401	498	I-cache hit – no I-cache update
0x0040_17cc	0x00401	499	I-cache hit – no I-cache update
0x0040_27c4	0x00402	497	I-cache miss – instruction 0x2652_0004 is copied into instruction field in set 497, V-bit in that set is changed to 1, tag to 0x00402
0x0040_27c8	0x00402	498	I-cache miss – instruction 0x1653_fffc is copied into instruction field in set 498, V-bit in that set is changed to 1, tag to 0x00402
0x0040_27bc	0x00402	495	I-cache hit – no I-cache update
0x0040_27c0	0x00402	496	I-cache miss – instruction 0x0c10_0f50 is copied into instruction field in set 496, V-bit in that set is changed to 1, tag to 0x00402
0x0040_17c0	0x00401	496	I-cache miss – instruction 0x8c99_0000 is copied into instruction field in set 496, V-bit in that set is changed to 1, tag to 0x00401
0x0040_17c4	0x00401	497	I-cache miss – instruction 0x0019_c023 is copied into instruction field in set 497, V-bit in that set is changed to 1, tag to 0x00401
0x0040_17c8	0x00401	498	I-cache miss – instruction 0xac98_0000 is copied into instruction field in set 498, V-bit in that set is changed to 1, tag to 0x00401
0x0040_17cc	0x00401	499	I-cache hit – no I-cache update
0x0040_27c4	0x00402	497	I-cache miss – instruction 0x2652_0004 is copied into instruction field in set 497, V-bit in that set is changed to 1, tag to 0x00402
0x0040_27c8	0x00402	498	I-cache miss – instruction 0x1653_fffc is copied into instruction field in set 498, V-bit in that set is changed to 1, tag to 0x00402
0x0040_27cc	0x00402	499	I-cache miss – instruction 0x2414_0000 is copied into instruction field in set 499, V-bit in that set is changed to 1, tag to 0x00402

Exercise B

3. C = 32 KB = 32000 bytes = 2^{15} bytes
   words per block = 8 words/block = 2^3 words/block
   bytes per word = 4 bytes/word = 2^2 bytes/word
   S = ?

   a) Number of Sets:

   $C = S \times words\ per\ block \times bytes\ per\ word$

   $2^{15} = S \times 2^3 \frac{words}{block} \times 2^2 \frac{bytes}{word}$

   $S = \frac{2^{15}\ bytes}{2^3 \frac{words}{block} \times 2^2 \frac{bytes}{word}} = 2^{15-3-} \ = 1024\ blocks$

   b) Address Split:

   The address split is: byte offset, $\log_2 4 = 2\ bits$;
   block offset, $\log_2 8 = 3\ bits$;
   set bits, $\log_2 1024 = 10\ bits$;
   tag bits, $32 - 10 - 3 - 2 = 17\ bits$.

   c) Bits Per Block:

   A block has: 1 V-bit,
   17 tag bits,
   8 words/block x 32 bits/word = 256 data bits.

   Thus, there are 1 + 17 + 256 = 274 bits per block.

   SRAM Cells:

   $Total\ number\ of\ bits = 274\ \frac{bits}{block} \times 1024\ blocks = 280576\ bits$

   Thus, 280576 SRAM cells are needed for the whole cache.

4. bytes per block = 64 bytes/block
   words per block = ?
   bytes per word = 8 bytes/word = 2^3 bytes/word
   S = ?

   a) C = 64 KB = 64000 bytes = 2^{16} bytes

   Words Per Block:

   $$Bytes\ per\ block = bytes\ per\ word \times words\ per\ block$$

   $$64\frac{bytes}{block} = 8\frac{bytes}{word} \times words\ per\ block$$

   $$Words\ per\ block = \frac{64\frac{bytes}{block}}{8\frac{bytes}{word}} = 8\frac{words}{block} = 2^3\frac{words}{block}$$

   Number of Sets:

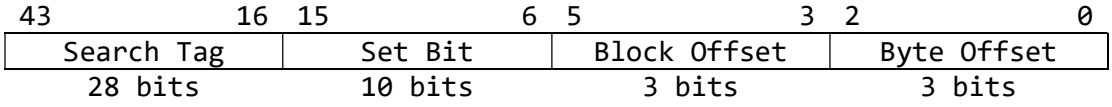
   $$C = S \times words\ per\ block \times bytes\ per\ word$$

   $$2^{16} = S \times 2^3\frac{words}{block} \times 2^3\frac{bytes}{word}$$

   $$S = \frac{2^{16}\ bytes}{2^3\frac{words}{block} \times 2^3\frac{bytes}{word}} = 2^{16-3-} \ = 1024\ blocks$$

   Address Split:

   The address split is: byte offset, $\log_2 8 = 3\ bits$;
   block offset, $\log_2 8 = 3\ bits$;
   set bits, $\log_2 1024 = 10\ bits$;
   tag bits, $44 - 10 - 3 - 3 = 28\ bits$.



b) C = 2 MB = 2000000 bytes = 2^{21} bytes

Words Per Block:

$$Bytes\ per\ block = bytes\ per\ word \times words\ per\ block$$

$$64\frac{bytes}{block} = 8\frac{bytes}{word} \times words\ per\ block$$

$$Words\ per\ block = \frac{64\frac{bytes}{block}}{8\frac{bytes}{word}} = 8\frac{words}{block} = 2^3\frac{words}{block}$$

Number of Sets:

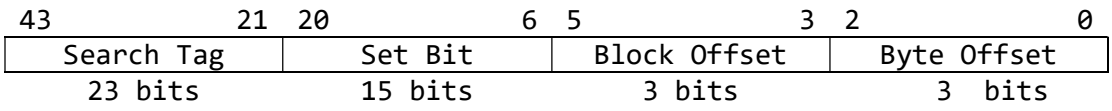
$$C = S \times words\ per\ block \times bytes\ per\ word$$

$$2^{21} = S \times 2^3\frac{words}{block} \times 2^3\frac{bytes}{word}$$

$$S = \frac{2^{21}\ bytes}{2^3\frac{words}{block} \times 2^3\frac{bytes}{word}} = 2^{21-3-3} = 32768\ blocks$$

Address Split:

The address split is: byte offset, $\log_2 8 = 3\ bits$;
block offset, $\log_2 8 = 3\ bits$;
set bits, $\log_2 32768 = 15\ bits$;
tag bits, $44 - 15 - 3 - 3 = 23\ bits$.



c) Bits Per Block:

A block has: 1 V-bit,
23 tag bits,
8 words/block x 64 bits/word = 512 data bits.

Thus, there are 1 + 23 + 512 = 536 bits per block.

SRAM Cells:

$$Total\ number\ of\ bits = 536\ \frac{bits}{block} \times 32768\ blocks = 17563648\ bits$$

Thus, 17563648 SRAM cells are needed for the whole cache.

Exercise C

Part II

Capacity:

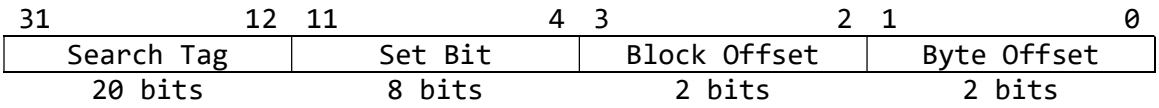
C = ?
words per block = 4 words/block
bytes per word = 4 bytes/word
S = 256 blocks

$$C = S \times words\ per\ block \times bytes\ per\ word$$

$$C = 256\ blocks \times 4\frac{words}{block} \times 4\frac{bytes}{word} = 4096\ bytes$$

Address Split:

The address split is: byte offset, $\log_2 4 = 2\ bits$;
block offset, $\log_2 4 = 2\ bits$;
set bits, $\log_2 256 = 8\ bits$;
tag bits, $32 - 8 - 2 - 2 = 20\ bits$.



Program Runs:

	heapsort_trace.txt	mergesort_trace.txt
sim1.c	64705 reads 49791 read hits 60419 writes 60401 write hits overall miss rate: 11.9%	104298 reads 88429 read hits 73410 writes 64076 write hits overall miss rate: 14.2%
sim2.c	64705 reads 57046 read hits 60419 writes 60387 write hits overall miss rate: 6.1%	104298 reads 99955 read hits 73410 writes 70863 write hits overall miss rate: 3.9%

There is significant spatial locality of reference in the memory accesses done by both algorithms because, even though both caches are the same size, the 4-word block performs a lot better than the 1-word block cache.

Part III

Capacity:

C = 4 KB = 4000 bytes = 2^{12} bytes
words per block = 16 words/block = 2^4 words/block
bytes per word = 4 bytes/word = 2^2 bytes/word
S = ?

$$C = S \times words\ per\ block \times bytes\ per\ word$$

$$2^{12}\ bytes = S \times 2^4 \frac{words}{block} \times 2^2 \frac{bytes}{word}$$

$$S = \frac{2^{12}\ bytes}{2^4 \frac{words}{block} \times 2^2 \frac{bytes}{word}} = 2^{12-4-2} = 64\ blocks$$

Address Split:

The address split is: byte offset, $\log_2 4 = 2\ bits$;
block offset, $\log_2 16 = 4\ bits$;
set bits, $\log_2 64 = 6\ bits$;
tag bits, $32 - 6 - 4 - 2 = 20\ bits$.

31 — 12	11 — 6	5 — 2	1 — 0
Search Tag	Set Bit	Block Offset	Byte Offset
20 bits	6 bits	4 bits	2 bits

Program Runs:

	heapsort_trace.txt	mergesort_trace.txt
sim3.c	64705 reads 59875 read hits 60419 writes 60324 write hits overall miss rate: 3.9%	104298 reads 101949 read hits 73410 writes 72243 write hits overall miss rate: 2.0%

Part II Source File:

```
// sim2.c
// ENCM 369 Winter 2018 Lab 10 Exercise C
// Author: S. Norman
// Edited by: R. Lim
//
// If you build an executable using gcc -Wall sim2.c -o sim2
// you can run it by redirecting input to come from a data file,
// as in
//         ./sim2 < 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[256];
unsigned tag_array[256];

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, set;
    int hit;

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

        set = (address & 0xff0) >> 4; // bits 11-4
        tag = address >> 12;          // bits 31-12

        // 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;
}
```

Part III Source File:

```
// sim3.c
// ENCM 369 Winter 2018 Lab 10 Exercise C
// Author: S. Norman
// Edited by: R. Lim
//
// If you build an executable using gcc -Wall sim3.c -o sim3
// you can run it by redirecting input to come from a data file,
// as in
//         ./sim3 < 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[64];
unsigned tag_array[64];

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, set;
    int hit;

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

        set = (address & 0xfc0) >> 6; // bits 11-6
        tag = address >> 12;          // bits 31-12

        // 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;
}
```