**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)

(b) Byte offset: A 32-bit word is 4 bytes, so the width is

Block offset: Each block has 8 words, so the width is

Set bits: S = 256, so we need

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 13 | 12 |  |  |  |  |  |  | 5 | 4 |  | 2 | 1 | 0 |
| search | | | | | | | | | | | | | | | | | | | set | | | | | | | | block | | | byte | |

(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

Block offset: Each block has 64 bytes = 8 words, so the width is

Set bits:

S = 512, so we need

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:

S = 32768, so we need

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

Block offset: Each block has 4 words, so the width is   
 S=1024, so we need

Search tag: 32-10-2-2=18

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 14 | 13 |  |  |  |  |  |  |  |  | 4 | 3 | 2 | 1 | 0 |
| search | | | | | | | | | | | | | | | | | | set | | | | | | | | | | block | | byte | |

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

Block offset: Each block has 4 words, so the width is   
 Set bits:

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 13 | 12 |  |  |  |  |  |  |  | 4 | 3 | 2 | 1 | 0 |
| search | | | | | | | | | | | | | | | | | | | set | | | | | | | | | block | | byte | |

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

\*/

#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;

}