CSED211 Homework 4

20190084 권민재

1. Exercise 6.25

The following table gives the prarmeters for a number of different caches. For each cache, fill in the missing fields in the table. Recall taht m is the number of physical address bits, C is the cache size (number of data bytes), B is the block size in bytes, E is asscociativity, S is the number of cache sets, t is the number of tag bits, s is the numbner of set index bits, and b is the number of offset bits.

Solution

- $S = C \div B \div E$
  - 총 캐시 사이즈에서 블록 사이즈를 나누면 블록의 개수가 되고, 그 블록의 개수를 다시 asscociativity의 수로 나누면 set의 개수가 된다.
- $s = \log_2 S$
  - set index 비트의 크기는 S에 밑이 2인 로그를 취한 값을 가진다.
- $b = \log_2 B$
  - 오프셋 비트의 크기는 B에 밑이 2인 로그를 취한 값을 가진다.
- $t = m - (s + b)$
  - 태그 비트의 크기는 주소의 전체 길이에서 $s + b$를 뺀 값을 가진다.

Answer

Cache	m	C	B	E	S	t	s	b
1	32	1024	4	4	64	24	6	2
2	32	1024	4	256	1	30	0	2
3	32	1024	8	1	128	22	7	3
4	32	1024	8	128	1	29	0	3
5	32	1024	32	1	32	22	5	5
6	32	1024	32	4	8	24	3	5

2. Exercise 6.29

Suppose we have system with the follwing properties:

- The memory is byte addressable.
- Memory accesses are to 1-byte words (not to 4-byte words).
- Addresses are 12 bits wide.
- The cache is two-way set associative ($E = 2$), with a 4-byte block size ($B = 4$) and four sets ($S = 4$)

The contents of the cache are as follows, with all addresses, tags, and values given in hexadecial notation.

Set index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0				
2	00	1	48	49	4A	4B
	40	0				
3	FF	1	9A	C0	03	FF
	00	0				

A.

The following diagram shows the format of an address (1 bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the follwing:

- CO. The cache block offset
- CI. The cache set index
- CT. the cache tag

Solution

- CI. The cache set index
  - $s = \log_2 S = \log_2 4 = 2$
- CO. The cache block offset
  - $b = \log_2 B = \log_2 4 = 2$
- CT. the cache tag
  - $t = m - (s + b) = 12 - (2 + 2) = 8$

Answer

12	11	10	9	8	7	6	5	4	3	2	1	0
	CT.	CT.	CT.	CT.	CT.	CT.	CT.	CT.	CI.	CI.	CO.	CO.

B.

For each of the follwing memory accesses, indicate if it will be a cache hit or miss when *carried out in sequence* as listed. Also give the value of a read if it can be inferred from the information in the cache.

Solution

Read 0x834

Binary	Tag	Index	Offset
100000110100	10000011	01	00
0x834	0x83	0x1	0x0

Set Index가 1인 set에 tag가 0x83인 블럭은 존재하지만, 해당 블럭은 valid 하지 않으므로, **hit가 발생하지 않는다**.

Write 0x836

Binary	Tag	Index	Offset
100000110110	10000011	01	10
0x836	0x83	0x1	0x2

Set Index가 1인 set에 tag가 0x83인 블럭은 존재하고, 앞서 **Read 0x834**를 수행하면서 해당 블럭은 valid하게 되었으므로 **hit가 발생한다**.

Read 0xFFD

Binary	Tag	Index	Offset
111111111101	11111111	11	01
0xFFD	0xFF	0x3	0x1

Set Index가 3인 set에 tag가 0xFF인 블럭이 있고, 해당 블럭은 valid 하므로 **hit가 발생한다.** 이때 오프셋은 0x1이므로, 읽을 값은 **0xC0**으로 추론된다.

Answer

Operation	Address	Hit?	Read value (or unknown)
Read	0x834	No.	unknown
Write	0x836	Yes.	unknown
Read	0xFFD	Yes.	0xC0

3. Exercise 6.30

Suppose we have a system with the following properties:

- The memory is byte addressable.
- Memory accesses are to 1-byte words (not to 4-byte words).
- Address are 13 bits wide.
- The cache is 4-way set associative ($E = 4$) , with 4-byte block size ($B = 4$) and eight sets ($S = 8$)

Consider the following cache state. All addresses, tags, and values are given in hexadecimal format. The Index column contains the set index for each set of four lines. The Tag columns contain the tag value for each line. The V columns contain the valid vit for each line. The Bytes 0-3 columns contain the data for each line, numbered left to right starting with byte 0 on the left.

(Table for this problem is omitted. It's not matter!)

A.

What is the size (C) of this cache in bytes?

Solution

캐시의 사이즈는 set의 개수와 associative, 그리고 블럭 사이즈를 모두 곱해서 구할 수 있다.

$C = S \times B \times E = 8 \times 4 \times 4 = 128$

Answer

128

B.

The box that follows shows the format of an address (1 bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

- CO. The cache block offset

- CI. The cache set index
- CT. the cache tag

Solution

- CI. The cache set index
  - $s = \log_2 S = \log_2 8 = 3$
- CO. The cache block offset
  - $b = \log_2 B = \log_2 4 = 2$
- CT. the cache tag
  - $t = m - (s + b) = 13 - (3 + 2) = 8$

Answer

12	11	10	9	8	7	6	5	4	3	2	1	0
CT.	CT.	CT.	CT.	CT.	CT.	CT.	CT.	CI.	CI.	CI.	CO.	CO.

4. Exercise 6.34

Consider the follwing matrix transpose routine:

```
typedef int array[4][4];

void transpose2(array dst, array src)
{
    int i, j;
    for (i = 0; i < 4; i++){
        for(j = 0; j < 4; j++){
            dst[j][i] = src[i][j];
        }
    }
}
```

Assume this code runs on a machine with the following properties:

- `sizeof(int)` = 4
- The `src` array starts at address 0 and the `dst` array starts at address 64 (decimal).
- There is a single L1 data cache that is direct-mapped, write-through, write-allocate, with a block size of 16 bytes.
- The cache has a total size of 32 data bytes, and the cache is initally empty.
- Accesses to the `src` and `dst` arrays are the only sources of read and write misses, respectively.

A.

For each `row` and `col` , indicate whether the access to `src[row][col]` and `dst[row][col]` is a hit (h) or a miss(m). For example, reading `src[0][0]` is a miss and wrting `dst[0][0]` is also miss.

Solution

Address	Binary	Tag	Index (1)	Offset (4)	Hit?
0	00000000	000	0	0000	No.
64	01000000	010	0	0000	No.
4	00000100	000	0	0100	No.
80	01010000	010	1	0000	No.
8	00001000	000	0	1000	Yes.
96	01100000	011	0	0000	No.
12	00001100	000	0	1100	No.
112	01110000	011	1	0000	No.
16	00010000	000	1	0000	No.
68	01000100	010	0	0100	No.
20	00010100	000	1	0100	Yes.
84	01010100	010	1	0100	No.
24	00011000	000	1	1000	No.
100	01100100	011	0	0100	No.
28	00011100	000	1	1100	Yes.
116	01110100	011	1	0100	No.
32	00100000	001	0	0000	No.
72	01001000	010	0	1000	No.
36	00100100	001	0	0100	No.
88	01011000	010	1	1000	No.
40	00101000	001	0	1000	Yes.
104	01101000	011	0	1000	No.
44	00101100	001	0	1100	No.
120	01111000	011	1	1000	No.

48	00110000	001	1	0000	No.
76	01001100	010	0	1100	No.
52	00110100	001	1	0100	Yes.
92	01011100	010	1	1100	No.
56	00111000	001	1	1000	Yes.
108	01101100	011	0	1100	No.
60	00111100	001	1	1100	Yes.
124	01111100	011	1	1100	No.

Answer

`dst` array

	Col. 0	Col. 1	Col. 2	Col. 3
Row 0	m	m	m	m
Row 1	m	m	m	m
Row 2	m	m	m	m
Row 3	m	m	m	m

`src` array

	Col. 0	Col. 1	Col. 2	Col. 3
Row 0	m	m	h	m
Row 1	m	h	m	h
Row 2	m	m	h	m
Row 3	m	h	h	h

5. Exercise 6.41

You are writing a new 3D game that you hope will earn you fame and fortune. You are currently working on a function to blank the screen buffer before drawing the next frame. The screen you are working with is a 640 × 480 array of pixels. The machine you are working on has a 32 KB direct-mapped cache with 8-byte lines. The C structures you are using are as follows:

```
struct pixel {
    char r;
    char g;
    char b;
    char a;
};

struct pixel buffer[480][640];
int i, j;
char *cptr;
int *iptr;
```

Assume the following:

- `sizeof(char)` = 1 and `sizeof(int)` = 4
- `buffer` begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array `buffer`. Variables `i`, `j`, `cptr`, and `iptr` are stored in registers.

What percentage of writes in the follwing code will hit the cache?

```
for(j = 639; j >= 0; j--) {
    for (i = 479; i >= 0; i--){
        buffer[i][j].r = 0;
        buffer[i][j].g = 0;
        buffer[i][j].b = 0;
        buffer[i][j].a = 0;
    }
}
```

Solution

구조체 `pixel` 의 크기는 4바이트이고, 한 라인의 크기는 8바이트 이므로, 한 라인에 `pixel`이 두 번 들어갈 수 있다. 즉, 두 `pixel` 마다 한 번의 miss와 7번의 hit를 가지므로, miss rate는 12.5%이다.

Answer

87.5% hit, 12.5% miss

6. Exercise 7.6

This problem concerns the `m.o` module from Figure 7.5 and the follwoing version of the `swap.c` function that counts the number of times it has been called:

```
extern int buf[];

int *bufp0 = &buf[0];
static int *bufp1;

static void incr()
{
  static int count = 0;

  count++;
}

void swap()
{
  int temp;

  incr();
  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

For each symbol that is defined and referenced in `swap.io`, indicate if it will have a symbol table entry in `.symtab` section in module `swap.o`. If so, indicate the module that defines the symbol (`swap.o` or `m.o`), the symbol type (local, global, or extern), and the section (`.text`, `.data` or `.bss`) it occupies in that module.

Answer

Symbol	`swap.o` `.symtab` entry?	Symbol type	Module where defined	Section
`buf`	Yes.	extern	main.o	.data
`bufp0`	Yes.	global	swap.o	.data
`bufp1`	Yes.	local	swap.o	.bss
`swap`	Yes.	global	swap.o	.text
`temp`	No.			
`incr`	Yes.	local	swap.o	.text
`count`	Yes.	local	swap.o	.bss

7. Exercise 7.8

Inthis problem, let $\mathrm{REF}(x.i) \rightarrow \mathrm{DEF}(x.k)$denote that the linker will associate an arbitary reference to symbol x in module i to definition of x in module k. For each example below, use this notation to indicate how the linker would resolve references to the multiply-defined symbol in each module. If there is a link-time error(rule 1), wrtie "ERROR". If the linker arbitrarily chooses one of the definitions (rule3), write "UNKNOWN".

A.

```
/* Module 1*/
int main()
{
}
```

```
/* Module 2 */
static int main=1;
int p2()
{
}
```

Solution

모듈 2에서 main을 static을 이용하여 선언하였기 때문에, 모듈 2의 main은 local 변수라고 생각할 수 있다. 즉, 충돌이 일어나지 않으며, 각 모듈에는 유효한 main이 존재한다.

Answer

(a) $\mathrm{REF}(\mathrm{main}.1) \rightarrow \mathrm{DEF}(\mathrm{main}.1)$

(b) $\mathrm{REF}(\mathrm{main}.2) \rightarrow \mathrm{DEF}(\mathrm{main}.2)$

B.

```
/* Module 1 */
int x;
void main()
{
}
```

```
/* Module 2*/
double x;
int p2()
{
}
```

Solution

두 모듈의 `x`에 대해 두 개의 weak definition이 존재하므로, rule 3에 해당하는 경우이다.

Answer

UNKNOWN

C.

```
/* Module 1 */
int x=1;
void main()
{
}
```

```
/* Module 2 */
double x=1.0;
int p2()
{
}
```

Solution

두 모듈의 `x`에 대해 두 개의 strong definition이 존재하므로, rule 1에 해당하는 경우이다.

Answer

ERROR

8. Exercise 7.12

Consider the call to function `swap` in object file `m.o` (Problem 7.6)

```
9:    e8 00 00 00 00    callq e <main+0xe>
```

with the following relocation entry:

```
r.offset = 0xa
r.symbol = swap
r.type   = R_X86_64_PC32
r.addend  = -4
```

A.

Suppose that the linker relocates `.text` in `m.o` to address `0x4004e0` and `swap` to address `0x4004f8`. Then what is the value of the relocated reference to `swap` in the `callq` instruction?

Solution

`swap` + `r.addend` - (`.text` + `r.offset`) = 0x4004f8 - 4 - (0x4004e0 + 0xa) = 0x0a

Answer

0x0a

B.

Suppose that the linker relocates `.text` in `m.o` to address `0x4004d0` and `swap` to address `0x400500`. Then what is the value of the relocated reference to `swap` in the `callq` instruction?

Solution

`swap` + `r.addend` - (`.text` + `r.offset`) = 0x400500 - 4 - (0x4004d0 + 0xa) = 0x22

Answer

0x22