CS105 – Computer Systems Spring 2021

Problem Session 8: Caching (cont'd)

SOLUTION

March 24, 2021

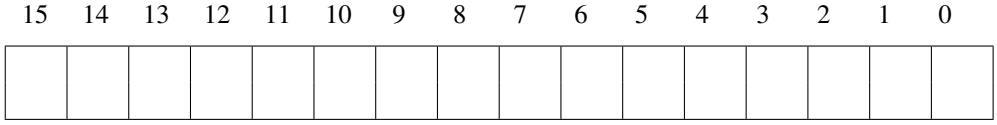
1. **Set-Associative Caches.** The following table depicts a 4-way set associate cache, with a 2 byte block size and 32 total lines:

4-way Set Associative Cache																
Index	Tag	Valid	Data		Tag	Valid	Data		Tag	Valid	Data		Tag	Valid	Data	
0	029	0	34	29	787	0	39	AE	C7D	1	68	F2	88B	1	64	38
1	AF3	1	0D	8F	C3D	1	0C	3A	D4A	1	A4	DB	6D9	1	A5	3C
2	2A7	1	E2	04	FAB	1	D2	04	BE3	0	3C	A4	D01	1	EE	05
3	23B	0	AC	1F	1E0	0	B5	70	B3B	1	66	95	E37	1	49	F3
4	780	1	60	35	02B	0	19	57	549	1	8D	0E	100	0	70	AB
5	EEA	1	B4	17	0CC	1	67	DB	08A	0	DE	AA	118	1	2C	D3
6	11C	0	3F	A4	D01	0	3A	C1	9F0	0	20	13	E7F	1	DF	05
7	D0F	0	00	FF	2AF	1	B1	5F	099	0	AC	96	C3A	1	22	79

You should assume:

- Memory is byte addressable, and all memory accesses read/write 1 byte.
- Memory addresses are 16 bits.
- The cache uses a least-recently used (LRU) eviction policy.
- The cache is write-back, write-allocate.
- No writes occur prior to the beginning of this problem.

(a) The box below depicts a 16-bit memory address. Indicate (by labeling the diagram) the fields that would be used to determine (1) the tag, (2) the index, and (3) the offset.



CT: [15-4] CI: [3-1] CO: [0]

(b) Consider the following sequence of accesses (yes, they occur sequentially). For each access, indicate whether that access would correspond to a cache hit or a cache miss, which byte is read (for reads), and whether or not a memory write will occur. Use the notation `MEM[addr]` for reads are cache misses.

	Operation	Tag	Index	Offset	Hit?	Byte read	Mem write?
i.	`Write $0x01, 0x0CCB`	`0x0CC`	5	1	Hit	n/a	N
ii.	`Read 0xEEAA`	`0xEEA`	5	0	Hit	B4	N
iii.	`Read 0x118B`	`0x118`	5	1	H	D3	N
iv.	`Write $0x02, 0x047B`	`0x047`	5	1	Miss	n/a	N
v.	`Read 0x119B`	`0x119`	5	1	Miss	Mem[0x119B]	Y
vi.	`Read 0x047A`	`0x047`	5	0	Hit	Mem[0x047A]	N

2. **Optimization with Caches.**

After graduation, you start working for a company that wants to make Post-Its by printing yellow squares on white pieces of paper. As part of the printing process, they need to set the CMYK (cyan, magenta, yellow, black) value for every point in the square. They ask you to determine the efficiency of the following algorithms on a machine with a 2048-byte direct-mapped data cache with 32 byte blocks.

You are given the following definitions:

```
struct point_color {
    int c;
    int m;
    int y;
    int k;
};

struct point_color square[16][16];
register int i, j;
```

Assume:

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

(a) What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].y = 1;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: `12.5` %

(b) What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[j][i].c = 0;
        square[j][i].m = 0;
        square[j][i].y = 1;
        square[j][i].k = 0;
    }
}
```

Miss rate for writes to `square`: 25 %

(c) What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].y = 1;
    }
}
for (i=0; i<16; i++) {
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: 25 %

(d) Which algorithm would you recommend they use?

I'd go with (a). Fewer cache misses than (b) or (c), so we would expect it to have better performance.