In C, mulit-dimensional arrays are stored in row-major order: elements in a particular row are adjacent in memory. For example, given the declaration for an (n+1)-row by (m+1)-column array, "int A[n+1][m+1]", the array is layed out in memory as follows (lower memory addresses to the left),

A[0][0], A[0][1], ... A[0][m], A[1][0], A[1][1], ..., A[1][m], ..., A[n][0], A[n][1], ..., A[n][m]

That is, "A[k]" is a reference to a linear array of m+1 elements. Here is some C code:

```
int A[8000][8000], B[8000][8000], x, y;

for (x = 0; x < 8000; x++) {
   for (y = 0; y < 8; y++) {
      A[x][y] = B[y][0] + A[y][x];
   }
}
```

Q. Suppose we have 16B cache blocks and INT word size is 32-bit. How many words per cache line?

32b word = 4B word $\Rightarrow (16\,B/\text{line})(\text{word}/4B) = 4\,W/\text{line}$.

Q. How many cache blocks are needed to hold all data items referenced on the LHS in the inner loop?

The LHS is A[x][y]

for (x=0, y=0→7) the references are,

A[0][0], A[0][1], A[0][2], ... A[0][7] 8 contiguous words, 2 blocks

for (x=1, y=0→7) the references are,

A[1][0], A[1][1], A[1][2], ... A[1][7] 8 contiguous words, 2 blocks

... ...

and so on up to x = 8000-1. So, for each x value we reference 2 blocks, and there are 8000 different x values: (8000)(2 blocks) = 16,000 blocks needed for LHS references. Note that there is no block overlap between differing x's.

Q. Show the referenced items for the first term on the RHS. How many blocks are needed to hold these?

The first term on the RHS is B[y][0], y=0→7. The references are

B[0][0], B[1][0], ... B[7][0]

regardless of x. There are 8,000 ints between each reference; so, no overlap in cache blocks. ⇒ 8 blocks for first term.

Q. What items are referenced by the second term on the RHS? How many blocks needed? For a cache to hold all the items referenced, how many blocks would it need to store at once?

Second RHS term is A[y][x]. for x=0, y=0→7, the references are

A[0][0], A[1][0], A[2][0], ... A[7][0]

for x=1, y=0-7, the references are

A[0][1], A[1][1], A[2][1], ... A[7][1]

. . .

for x = 8000-1

A[0][x], A[1][x], A[2][x], ... A[7][x]

Rearranging by columns,

A[0][x] for $x \in [0, 8000)$

A[1][x] for $x \in [0, 8000)$

. . .

A[7][x] for $x \in [0, 8000)$

These are the first 8 entire rows. (But the first 8 words in each row were already counted on the LHS.) Each row is 8,000 contiguous words: 2,000 blocks per row, 8 rows, gives 16,000 blocks. the overlap is 8 words x 8 rows ⇒ 2 blocks x 8 rows ⇒ 16,000 - 16 blocks. Total = 8 + 2x16,000 - 16 blocks

Q. In the above code, which memory references have temporal locality? Which have spatial locality?

The references B[y][0] for y = 0→7, while not spatially local, are accessed every iteration of the outer loop: temporal locality. Variables x and y are also accessed repeatedly, showing temporal locality. For any x, the references A[x][0], .., A[x][7] are spatially local, but being accessed only once, are not temporally local. These are the first 8 columns of A. The first 8 rows of A, A[y][x] have spatial locality only.

Following is a sequence of word-sized memory references (32-bit addresses, 32-bit words, word-addressability). Only the lower 16 bits of each address is shown: assume the upper 16 bits are 0x0040.

0x0001, 0x0134, 0x0212, 0x0001, 0x0135, 0x0213, 0x0162, 0x0161, 0x0002, 0x0044, 0x0041, 0x0221

Suppose we have the choice of either of three direct-mapped cache designs:

(C1) 8 1-word blocks, miss penalty = 25 cycles, hit access time = 2 cycles.
(C2) 4 2-word blocks, miss penalty = 25 cycles, hit access time = 3 cycles.
(C3) 2 4-word blocks, miss penalty = 25 cycles, hit access time = 5 cycles.

Q. For each cache, show which references are cache hits and which are misses for a system using that cache. What is the total number of words transmitted between cache and memory?

We'll assume word-addressability.

C1: 8 1-word blocks
3 bit cache index, 0 offset bits

Addr.	low 3 bits	index	
1	001	1	
134	100	4	
212	010	2	hit
1	001	1	hit
135	101	5	
213	011	3	
162	010	2	
161	001	1	
2	010	2	
44	100	4	
41	001	1	
221	001	1	

Words Transmitted: 11 x 1

C2: 4 2-w blocks
2-bit index, 1 bit offset

index	block	
0	0 1	hit
2	134 135	hit
1	212 213	hit
0	0 1	hit
2	2 3	
1	162 163	
0	160 161	
1	2 3	
2	44 45	
0	40 41	
0	220 221	

9 x 2

C3: 2 4-w blocks
1 bit index, 2-bit offset

index		
0	0 1 2 3	
1	134 135 136 137	hit
0	210 211 212 213	
0	0 1 2 3	
1	134 135 136 137	
0	210 211 212 213	
0	160 161 162 163	
0		hit
0	0 1 2 3	
1	44 45 46 47	
0	40 41 42 43	
0	220 221 222 223	

10 x 4

Q. What is the total access time in cycles for a system using each these caches? Which is better?

C1: (12 hits) x (2 cycles) + (11 miss) x (25 cycles) = 24 + 11x25

C2: (12 hits) x (3 cycles) + (9 miss) x (25 cycles) = 36 + 11x25 - 2x25

C3: (12 hits) x (5 cycles) + (10 miss) x (25 cycles) = 60 + 11x25 - 1x25

C2 is best

Suppose a direct-mapped cache uses its address bits in the following way:

ADDRESS[31:10]	ADDRESS[9:4]	ADDRESS[3:0]
tag	index	offset

The memory is byte-addressable.

Q. What is the cache block size in 32b words? How many entries does the cache have (that is, how many cache blocks does the cache hold)?

4-bit offset ⇒ 16 B cache block $\left(\frac{32b}{4B}\frac{\text{word}}{}\right)$ = 4 word block

6-bit index ⇒ 2^6 entries = 64 entries

Q. In total, how many data bits does the cache hold (assuming all entries are valid)? In total, how many bits of storage does the cache require? What is the VLSI area overhead with respect to the storage area for data bits?

Total data bits = 64 entries $\left(\frac{1\text{ block}}{\text{entry}}\right)\left(\frac{4\text{ W}}{\text{block}}\right) = 2^8\text{ W}\left(\frac{2^5\text{ b}}{\text{W}}\right) = 2^{13}\text{ b} = 2^3 \times 2^{10}\text{ b} = 8\text{k b}$

bits per line = (4 data W) + (22 tag bits) + (1 valid bit) = 128 + 23 = 151

Total bits = 64 × 151 = 9664 bits. $\frac{\text{overhead}}{\text{data}} = \frac{64 \cdot 23\text{ bits}}{8\text{k bits}} = \frac{1472}{8\text{k}} \approx 18\%$

Q. Suppose we alter the cache to be 8-way set associative without changing its overall size or the size of its cache block. Show the usage of address bits.

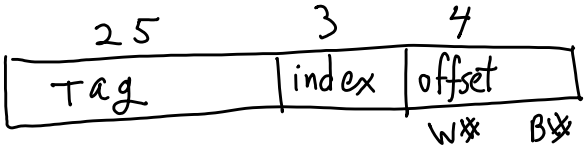
8-way ⇒ 8 blocks per set ⇒ $\left(\frac{8\text{ blocks}}{\text{set}}\right)\left(\frac{4\text{ W}}{\text{block}}\right)\left(\frac{4\text{ B}}{\text{W}}\right) = 128\text{ B/set}$

$\left(\frac{8\text{k bit data}}{\text{cache}}\right)\left(\frac{1\text{ B}}{8\text{ bits}}\right) = 1\text{kB data}$ ⇒ $1\text{kB data}\left(\frac{1\text{ set}}{128\text{ B data}}\right) = \frac{2^{10}}{2^7}\text{ sets} = 2^3\text{ sets}$

⇒ 3-bit index

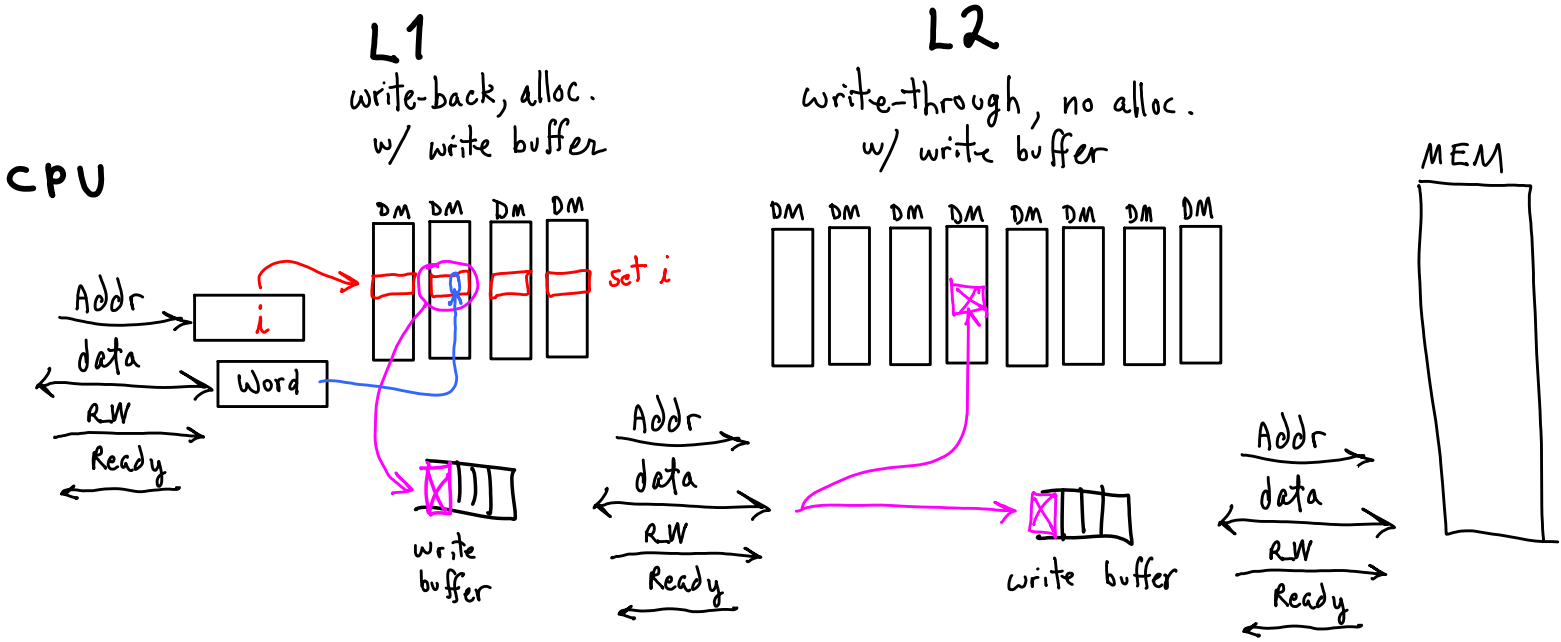
$4\text{ W/block} = 16\text{ B/block}$ ⇒ 4-bit offset

$\frac{16\text{ B block}}{4\text{ W block}} \rightarrow 4\text{B/W}$ → 2-bit B※ → 2-bit W※



Our L1 cache is (write-back, allocate), our L2 cache is (write-through, no-allocate). Both have their own write buffers. L1's write buffer simply passes its writes on to L2 as writes. Since L2 is write-through with a write buffer, it is alternatively called "write-behind". Both use the same block sizes. L1 is 4-way set associative; L2 is 8-way set associative.

Q. Write a cache-controller algorithm for handling an L1 write miss. A cache controller is a finite-state machine, which can be specified as an algorithm. Both L1 and L2 have separate controllers that communicate via signals (R/W request, Ready, Address, Data, ...) This algorithm for a write miss would be the specification for a combination of the write-miss portion of the L1 controller and part of the L2 controller.



L1: (pick victim from set i ②)

* (Write victim if dirty) Send to L2 via buffer { [Addr] = [TAG, i, W** = 0, B** = 0] = address of victim; [data block of victim] }

** (Send read request to L2) [Addr] = address from CPU; [R_W = Read]

(Send L1.Ready to CPU)

(On L2.Ready)

(get data block from L2.data, put it + tag in victim's entry)

(write word from CPU.data to block, set dirty)

* see next page

** see next page

* **L2:** (write victim block to buffer)
(if hit for victim's Addr, write victim block to cache)

** **L2:** (if hit for Addr, send block to L1)
(else)
(Send [Addr, RW=Read] to Mem)
(Wait Mem.Ready)
(write block to cache)
(Send block to L1)

Suppose program P has the following behavior per 1000 instructions executed: data reads = 180, data writes = 120. Suppose P running on system S has 0.2% instruction cache misses and 2% data cache misses. S executes one instruction per cycle, except for memory stalls. All instruction and data accesses are 32b words, and cache blocks are 16B.

Q. If S has a (write-through, allocate)-cache without write buffering, what minimum memory bandwidth (bytes per cycle) is needed to guarantee S has an overall CPI of 2? If S's clock rate is 2 GHz, what is the required memory bandwidth in B/sec?

Assume we can ignore cache access overhead and only consider memory access stall time. Each write requires w cycles to complete and consists of a single 4B word. Total write cycles per 1000 instructions is,

$$120 \text{ (data writes)} \left(\frac{4\text{B}}{\text{write}}\right)\left(\frac{w \text{ cycles}}{\text{B}}\right)$$

Since write-through caches do not need to write evicted blocks, that's all for writes. For reads, every miss requires 16B block read. Instruction miss rate is 0.2%, or 2 per 1000 instructions. Data read misses are 2% of 180, or 3.6 per 1000 instructions. Data write misses require a block read each, 2% of 120 writes miss, or 2.4 per 1000 instructions. Total read cycles per 1000 instructions,

$$\left[\, 2 \text{ (instr misses)} + 3.6 \text{ (data read misses)} + 2.4 \text{ (data write miss)} \,\right]\left(\frac{16\text{B}}{\text{miss}}\right)\left(\frac{r \text{ cycles}}{\text{B}}\right)$$

Let's assume $r = w$. Total memory requirement is then,

$480 \text{ B write}\left(\frac{w \text{ cycle}}{\text{B}}\right) + 128 \text{ B read}\left(\frac{r \text{ cycle}}{\text{B}}\right) = 608 \text{ B}\left(\frac{w \text{ cycle}}{\text{B}}\right)$. We have

$(1000 \text{ instr. exec.})\left(1 \text{ cycle}/\text{exec}\right)$ cycles for instruction execution, and

$608 \text{ B}\left(\frac{w \text{ cycle}}{\text{B}}\right)$ cycles for cache-miss stalls, or

$$\overline{CPI} = \frac{1000 \text{ instr. exec}\left(\frac{1 \text{ cycle}}{\text{exec}}\right) + 608\text{B}\left(\frac{w \text{ cycle}}{\text{B}}\right)}{1000 \text{ instr}} = 2\left(\frac{\text{cycle}}{\text{INSTR}}\right).$$

Solving for w, $w = \frac{1000}{608}$ (cycles/B)

memory bandwidth, or about $3/5$ B/cycle. For a clock rate of 2 GHz, this is

$\left(2\frac{\text{G cycles}}{\text{sec}}\right)\left(\frac{3/5\,\text{B}}{\text{cycle}}\right) = 1.2 \text{ GB/sec}$ memory bandwidth required.

Sanity check:

$$\begin{pmatrix}\text{Time}\\ \text{for}\\ \text{R/W}\end{pmatrix} = 608\, w \text{ cycles}\left(\frac{\text{sec}}{2\text{G cycles}}\right) = \frac{608\, w}{2\text{G}} \text{ sec}$$

$$\text{Bandwidth} = \frac{608 \text{ B}}{(608\, w/2\text{G})\text{ sec}} = \frac{2\text{G}}{5/3} \text{ B/sec} = 1.2 \text{ G B/sec}$$

Q. Suppose S is modified to have write buffering. Can this change the required minimum memory bandwidth to get an average CPI = 2?

Total traffic to memory does not change. However, total execution time will be affected: memory writes will not cause processor stalls. As we do not stall the processor for writes, there will be more instructions executed per sec. While the total memory traffic is the same, the write traffic can be overlapped with instruction execution time. Provided we can get all the writes done in that time, our only concern is getting the reads done fast enough to get CPI = 2.

Without the 480 B (W cycles/B) write stall cycles, we have

$$\overline{CPI} = \frac{1000 \text{ instr. exec}\left(\frac{1 \text{ cycle}}{\text{exec}}\right) + 128 \text{ B}\left(\frac{W \text{ cycle}}{\text{B}}\right)}{1000 \text{ instr}} = 2\left(\frac{\text{cycle}}{\text{instr.}}\right) \qquad W = \frac{1000}{128}\left(\frac{\text{cycles}}{\text{B}}\right)$$

or $(0.128 \text{ B/cycle}) \Rightarrow$ Bandwidth $= (0.128 \text{ B/cycle})\left(2\text{G} \frac{\text{cycle}}{\text{sec}}\right) = 0.256 \times 10^9 \text{ B/sec}$

$$\approx \frac{1}{4} \text{ GB/sec} = 250 \text{ MB/sec}$$

We just need to check that the writes can be done fast enough. The time we have to do writes is,

$$1000 \text{ instr. exec.}\left(1 \text{ cycle/exec}\right)\left(\frac{1}{2\text{G}} \text{ sec/cycle}\right) = \frac{500}{1\text{G}} \text{ sec}$$

We have to write 480 B in that time,

$$\text{Bandwidth} = (480 \text{ B}) \Big/ \left(\frac{500}{1\text{G}}\right) \text{sec} = \left(\frac{480}{500}\right) \text{G B/sec} \approx 1 \text{ GB/sec}$$

Unfortunately, in this case, the memory bandwidth requirement cannot be relaxed much (~20% decrease). However, the performance increase is,

$$S' = \frac{T_{\text{w/o buffer}}}{T_{\text{w/ buffer}}} = \frac{1000 \text{ (cycle)}\left(\frac{1 \text{ sec}}{2\text{G cycle}}\right) + 608 \text{ B}\left(\frac{1 \text{ sec}}{\text{GB}}\right)}{1000 \text{ (cycle)}\left(\frac{1 \text{ sec}}{2\text{G cycle}}\right) + 128 \text{ B}\left(\frac{1 \text{ sec}}{\text{GB}}\right)}$$

$$= \frac{500 + 608}{500 + 128} = \frac{1108}{628} \Rightarrow \text{about 75\% faster}$$

Q. Suppose system S2 has a (write-back, allocate)-cache with write buffering. Assume 30% of evicted cache blocks are dirty (modified). For the same program P and miss rates, what memory bandwidth is needed to achieve an average CPI of 2?

The total number of write instructions is 120 per 1000 instr. This is the maximum number of cache blocks possibly modified, per 1000 instructions, and misses are 2 (instr. read misses) + 3.6 (data read misses) + 2.4 (data write misses).

How many misses cause evictions? They might all be cold misses, causing no write-backs, best case. Worst case, in a unified cache, all misses cause evictions. Since only evictions cause writes to memory,

30% (2 + 3.6 + 2.4) per 1000 instr. cause memory writes, and all (2 + 3.6 + 2.4) misses cause memory reads.

All are 16 B cache block transactions, unlike write-through.

So, we have $1.3(2+3.6+2.4)\,16\,B = (1.3)(8)(16\,B) = 166.4\,B$ memory accesses per 1000 instr. Write buffering means no stalls for writes (or their alloc. reads), and total execution cycles are

$$1000\,(\text{instr})\left(\frac{1\ \text{cycle}}{\text{instr.}}\right) + (2+3.6)(\text{read misses}) \times \left(\frac{16\,B}{\text{miss}}\right)\left(\frac{r\ \text{cycles}}{B}\right)$$

$$2\left(\frac{\text{cycles}}{\text{instr}}\right) = \overrightarrow{CPI}\left(\frac{\text{cycles}}{\text{instr.}}\right) = \frac{1000\ \text{cycles} + 89.6\,r\ \text{cycles}}{1000\ \text{instr.}}.$$

Solve for r: $\frac{1000}{89.6} \cong \frac{11\ \text{cycles}}{B}$

$$\text{Bandwidth} = \frac{1}{11}\,B/\text{cycle}\ \ 2G\,\frac{\text{cycles}}{\text{sec}} = \frac{2000}{11}\,MB/\text{sec} \approx 200\,MB/\text{sec}$$

Reads due to write-misses and all writes are overlapped w/ execution.

$T_{exec} = 1000/2G$ traffic is $(0.3(8) + 2.4) \times 16B \cong 77\,B$

$$\text{Bandwidth} = 77B/(1000/2G) = \frac{154}{1000}\,G\,B/\text{sec} = 154\,MB/\text{sec}$$

A "streaming" system typically does many sequential data reads with very little reuse of the same memory location. Prefetching brings in data speculatively, based on memory access patterns, before it is referenced. A stream buffer prefetches data that is sequentially adjacent to the most recent cache block referenced. The stream buffer is logically part of the cache; it is accessed in parallel with the cache. If there is a hit in the buffer, the cache block is moved to the cache proper. If a cache block is accessed that is not adjacent to a block in the buffer, that block in the buffer will be overwritten by the next speculative prefetch.

Q. Suppose system S has a 2-block prefetch buffer and that the amount of computation required per cache block takes long enough so that prefetches always complete before the next cache block is requested. Suppose a process running on S accesses a 1/2 MB of contiguous data in a loop that runs for 100 iterations. The sequence of memory addresses accessed has these offsets from the start of the data,

0, 4, 8, 12, 16, 20, 24, ..., 0, 4, 8, 12, ...

and so on. S has a 64kB DM cache, cache blocks are 8 32b words, and the cache is initially cold (all entries are invalid). The memory is byte-addressable. Calculate the miss rate for this job. What would be the miss rate if there were no prefetching?

64kB cache $\left(\frac{1\text{ block}}{8\text{ word}}\right)\left(\frac{1\text{ word}}{4\text{ B}}\right) = \frac{2^{16}}{2^5} = 2^{11}$ blocks in cache $\Rightarrow$ 11 index bits

cache blocks, labeled by word instead of by byte:

$\frac{1}{2}\text{MB}\left(\frac{\text{word}}{4\text{B}}\right) = \frac{1}{2}2^{20}\text{B}\left(\frac{\text{word}}{2^2\text{B}}\right) = 2^{17}$ words

$2^{17}\text{ words}\left(\frac{\text{block}}{8\text{ words}}\right) = 2^{14}$ blocks

As the blocks are sequentially accessed, the cache is filled from top to bottom in 2^{11} block reads. The next 2^{11} block reads clobbers the entire cache. This happens $2^{14}/2^{11} = 8$ times per iteration.

w/o pre-fetch, every 8th word access causes a block miss. There are $2^{14} \times 100$ misses. There are $2^{14} \times 8 \times 100$ data accesses: $\text{MR} = \frac{2^{14} \times 100 \text{ misses}}{2^{14} \times 100 \times 8 \text{ accesses}} = 1/8$.

w/ pre-fetch, the next sequential block arrives, preventing a miss. The first block accessed is a cold miss. After 2^{14} sequential blocks are accessed, the next prefetch is the next block following the 2^{14}-th block of data. This is not a block of data. The next data access will have the same index and will miss. But after that, prefetching will again prevent misses for 2^{14} blocks. So, there is one miss for every iteration through the data, ie., 100 misses: $\text{MR} = \frac{100}{2^{14} \times 100 \times 8} = 1/2^{17}$

Q. Suppose S's miss penalty is BS X 20 cycles, for a cache block size of BS bytes. For instance, if BS = 16, S's miss penalty would be 16 X 20 cycles = 320 cycles. Suppose job J running on S has a memory access pattern that results in the following miss rates, depending on the cache block size (MR_i means the Miss Rate for BS = i bytes):

MR_8 = 8%, MR_16 = 3%, MR_32 = 1.8%, MR_64 = 1.5%, MR_128 = 2%

J runs with an average CPI of 1, except for memory stalls, and has an average memory reference rate of 1.35 (mem refs / instr.), which includes both instruction fetches and data read/writes. Find the block size that gives J its best performance. What is J's average CPI? NB--Miss rates include misses that apply to the combined cache and prefetch buffer.

$$S_{j-i} = \frac{T_{MR_i}}{T_{MR_j}} = \frac{n_{instr.}\left(\frac{1\ cycle}{instr}\right) + (1.35)n_{instr}\,MR_i \times BS_i \times 20}{n_{instr.}\left(\frac{1\ cycle}{instr.}\right) + (1.35)n_{instr}\,MR_j \times BS_j \times 20} = \frac{1 + (1.35)\,MR_i\,BS_i\,20}{1 + (1.35)\,MR_j\,BS_j\,20}$$

Let $R_i = MR_i \times BS_i$

Note: MR applies to instruction fetches also.

BS_i	MR_i	R_i
8	8%	
16	3%	0.48
32	1.8%	0.576
64	1.5%	0.96
128	2%	

by inspection $R_8 > R_{16}$, $R_{128} > R_{64}$ → Least

BS = 16B is best

$\overline{CPI} = 1 + (1.35)\,MR_{16}\,BS_{16}\,20 \approx 1.4$

Systems S1 and S2 have a memory access time of 70 ns. Job J has 36% memory accesses for data. Miss rates for J are MR_1k = 11%, MR_2k = 8%. S1's L1 cache is 1kB and access time is 0.62 ns; S2's L1 cache is 2kB with access time of 0.66 ns.

Q. What are the clock rates for the two systems? What is the average memory access time for each? Suppose both have average CPIs of 1 ignoring memory stalls, what are their CPIs?

$n_{instr}\left(0.36\ \frac{data\ r/w\ instr}{instr}\right) = n_{r/w}$. Suppose misses are all data misses.

(assuming instructions execute at cache rate so that execution is 1 cycle on hit.) ⇒

$CR_1 = \frac{1\ cycle}{0.62\ ns} \cong 1.6\ GHz$

$CR_2 = 1/0.66\ ns \cong 1.5\ GHz$

Assuming memory access time refers to data only:

$\overline{T}_{r/w} = \left(n_{r/w}\left(\frac{cache}{hit\ time}\right) + n_{r/w}\,MR\,(70\ ns)\right)/n_{r/w} = \left(\frac{cache}{hit\ time}\right) + MR(70\ ns)$

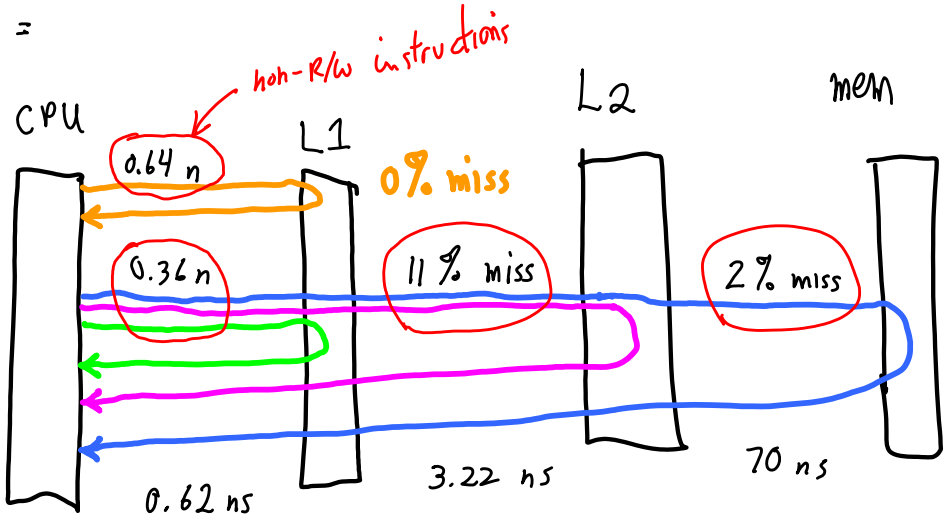
$\overline{T}^{(1)}_{r/w} = (0.62\ ns) + (11\%)(70\ ns) = 8.32\ ns$ $\left(\frac{1.6\ G\ cycle}{sec}\right) \Rightarrow 13.3\ cycles/r/w$

$\overline{T}^{(2)}_{r/w} = (0.66\ ns) + (8\%)(70\ ns) = 6.26\ ns$ $\left(\frac{1.5\ G\ cycle}{sec}\right) \Rightarrow 9.4\ cycles/r/w$

$CPI = \left[(n - n_{r/w})\left(\frac{1\ cycle}{inst}\right) + n_{r/w}(avg\ r/w\ cycles)\right]/n \Rightarrow \begin{cases} 0.64 + (0.36)(13.3) = 5.428\ cycle/instr \\ 0.64 + (0.36)(9.4) = 4.024\ cycle/instr \end{cases}$

Q. Suppose system S1 is given a 512B L2 cache with the following characteristics for J: MR_512k = 2%, access time = 3.22 ns. Which processor is faster?

$\overline{T}_1 =$



$= \left[(0.64 + 0.36)n \text{ @ } 0.62\text{ns} + 0.36n\,(11\%) \text{@ } 3.22\text{ ns} + 0.36n\,(11\%)(2\%) \text{ @ } 70\text{ ns} \right] / n$

$= 0.62 + + 0.128\text{ ns} + 0.055\text{ ns} = 0.803\text{ ns/instr}$

$\overline{T}_2 = \left[(0.64+0.36)n \text{ @ } 0.66\text{ns} + 0.36n\,(8\%) \text{@ } 70\text{ ns} \right] / n = 2.66\text{ ns/instr}$

$0.66\text{ ns} + 2\text{ ns}$

$S_{1-2} = \overline{T}_2 / \overline{T}_1 = \frac{2.66}{0.803} \cong 33$

System S has these characteristics: CPI = 2 (w/o memory stalls), CR = 2 GHz, memory access time = 125 ns, L1 cache MR = 5%.

Q. We are considering two different L2 caches for S: L2a is direct-mapped, has an access time of 15 cycles, and results in a global MR= 3%. L2b is 8-way set associative, has a 25 cycle access time, and results in a 1.8% global MR. (A global MR is the percentage of the total memory references that result in a read or write to main memory.) What are S's CPIs for (1) only an L1 cache, (2) both L1 and L2a, (3) both L1 and L2b?

$$CPI = \left[2\left(\frac{\text{cycle}}{\text{instr}}\right) n_{\text{instr}} + (5\%)\, n \left(125\text{ns}\left(\frac{2\text{G cycle}}{\text{sec}}\right)\right)\right]\frac{1}{n} = (2 + 12.5)\frac{\text{cycles}}{\text{instr.}} = 14.5$$

$$CPI_a = \left[2n + (5\%)n(15\text{ cycles}) + (5\%)(3\%)\left(125\text{ns}\left(2\text{G}\right)\right)\right]\frac{1}{n} = 2 + 0.75 + 0.375 = 3.125$$

$$CPI_b = \left[2 + 5\%(25) + (5\%)(1.8\%)(125 \cdot 2)\right] = 2 + 1.25 + 0.225 \cong 3.5$$

L1 + L2a is best

Q. If within the next few years we can expect memory speeds to double, which configuration is best? What if processor speed also quadruples?

memory speed $\rightarrow T_{mem} \rightarrow \frac{1}{2} T_{mem} \Rightarrow 125_{ns} \rightarrow 62.5_{ns}$

$CPI \rightarrow 2 + \quad 6.25 \rightarrow \quad 8.25$

$CPI_a \rightarrow 2 + 0.75 + 0.1875 \rightarrow 2.94$ L1 + L2a is best

$CPI_b \rightarrow 2 + 1.25 + 0.1125 \rightarrow 3.36$

+ CR → 4 CR (2G → 8G): the combined effect makes memory access ×4 in cycles

a → 2 + 0.75 + (4)(0.375) = 4.25 b → 2 + 1.25 + 4(0.225) = 4.15, b is best

System S has virtual memory with 4kB pages, 4-entry fully-associative TLB, true LRU replacement. Physical memory that holds is single page is called a page "frame".

Q. For the sequence of memory references below, the initial TLB content, and the initial page table content, show the final page table, TLB, and for each reference whether it is a TLB hit, a page table hit (page in memory), or a page fault. TLB entries are [valid, tag, #frame]; PT entries are [1, #frame] or [0, d], depending on whether the page is in a physical frame or not (if not, d is some disk address).

Memory references: 4095, 31272, 15789, 15000, 7193, 8912

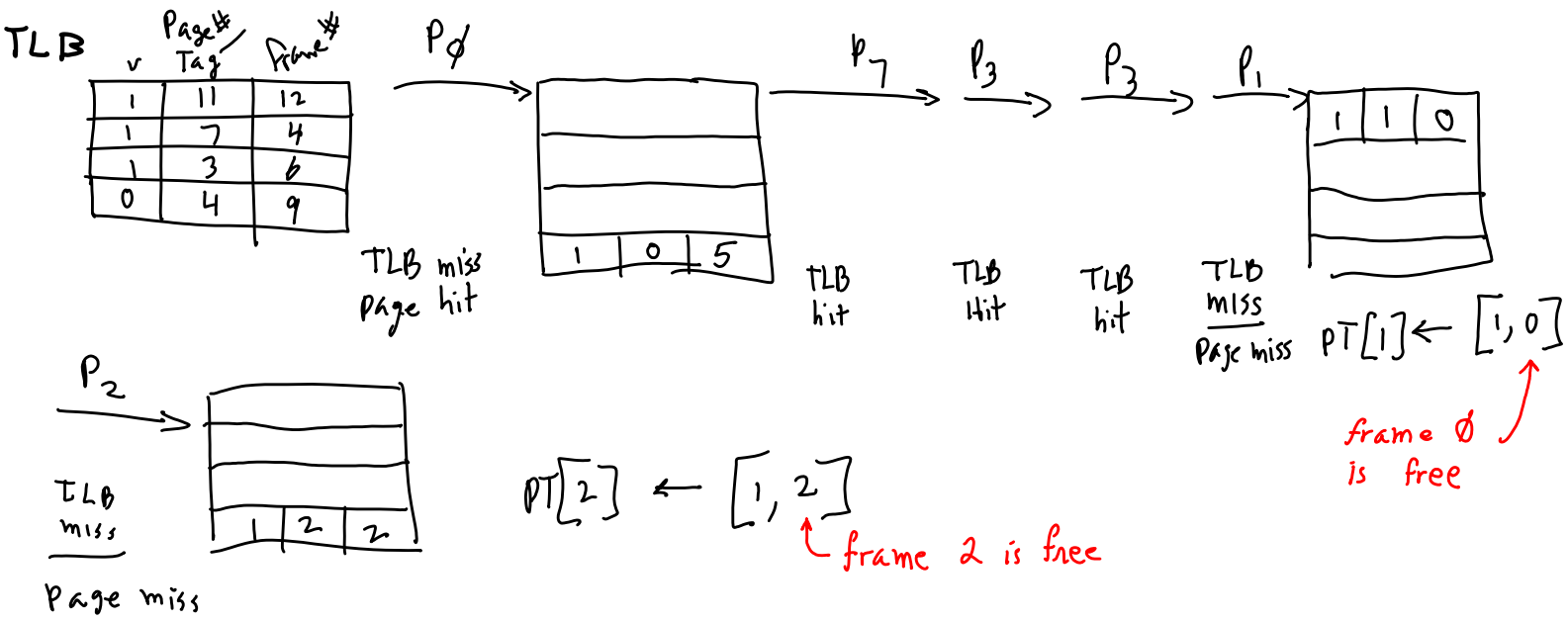
TLB: [1, 11, 12], [1, 7, 4], [1, 3, 6], [0, 4, 9]
PT: [1, 5], [0, d], [0, d], [1, 6], [1, 9], [1, 11], [0, d], [1, 4], [0, d], [0, d], [1, 3], [1, 12]

If a page must be brought in from disk, assume the free frame with the smallest frame number is used.

4kB pages → 2^{12} B pages → 12 bits offset, 4096 B per page

page # = Addr/4096 : 4095 → P_0, 31272 → P_7, 15789 → P_3, 15000 → P_3

7193 → P_1, 8912 → P_2



System S has 64b virtual addresses, 16 GB physical memory, 8 kB pages, 8B PT entries.

Q. For a single-level page table scheme, how many page table entries in a page table? How much physical memory would the page table require? What would be the minimum page size to make a single-level page table scheme practical?

$\left[(2^{64}\ B/mem)/(2^{13}\ B/page) = 2^{51}\ \text{page table entries} \right] @ 8B = 2^{54}\ B\ \text{table}$

$= 16\ \text{Peta B}$

16 GB physical memory, 2^{64} B virtual memory, $\left(2^{64}/2^{x}\ B/page\right) = 2^{64-x}$ pages

$(8\ B/PT\ entry)(2^{64-x}\ pages/PT) = 2^{64-x+3}\ B < 1\%\ (physical\ memory)$

$\approx \frac{1}{2^7}(2^4 \cdot 2^{30}\ B) = 2^{27}\ B$

Solve for x: $2^{67-x} < 2^{27} \Rightarrow 67-x < 27 \Rightarrow x = 40$

$\Rightarrow 2^{40}\ B/page = \text{Tera B pages} > 1{,}000\ GB$ per page.

Q. If we instead use a multi-level page table, with each an 8kB page directory and 8kB sub-directories and sub-tables. That is, each piece of the multi-level page table data structure fits into an 8kB frame. How many levels would be required?

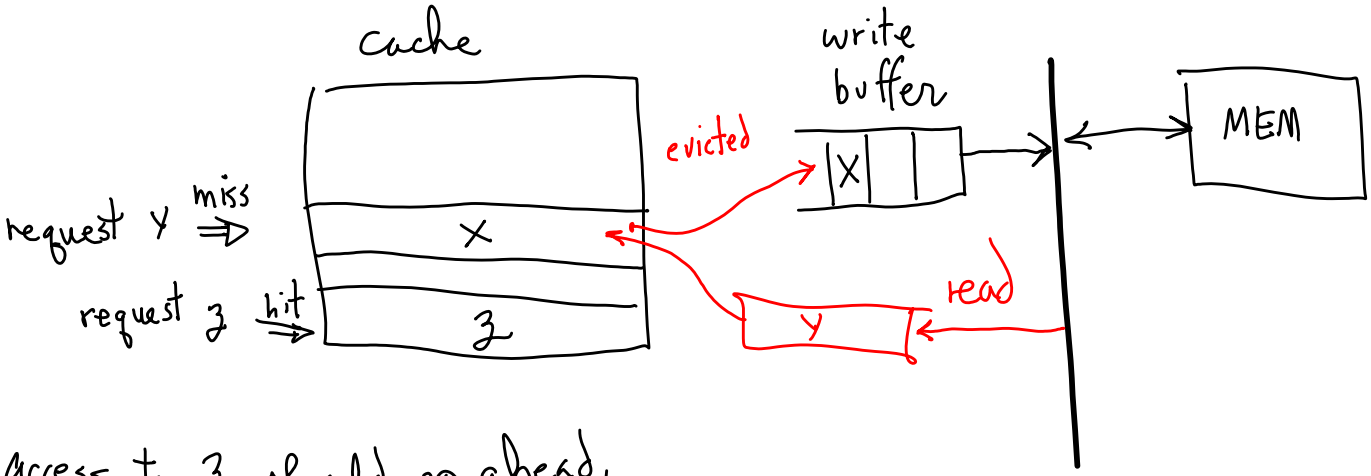
$(8\ kB\ page/8\ B/entry) = 1\ k\ entries/page$

$(2^{10}\ 1^{st}\text{-level entries})(2^{10}\ 2^{nd}\text{-level}) \times (2^{10}\ 3^{rd}\text{-level}) \times (2^{10}\ 4^{th}) \times (2^{10}\ 5^{th}) \times (2^{10}\ 6^{th}) \geq 2^{51}$

$\Rightarrow$ 6-levels

System S has a direct-mapped cache write buffer. The cache is write-back. If there is a cache miss and the evicted cache line is modified, the cache will immediately issue a memory read request for the missed cache block and sends the evicted cache block to the write buffer to be handled by a memory bus interface unit whenever the memory bus is free.

Q. Suppose the evicted cache block is being written from the write buffer when another instruction makes a memory reference that hits in the cache. What action should the cache take? What would happen if an instruction referenced the evicted cache block?



- Access to Z should go ahead, MMU handles write buffer independently

- Access request for X
  - Causes a cache miss
  - find victim to evict
    ⇒ But, that is Y, and Y was just brought in.
    - send victim Y to write buffer, even if not needed (not dirty).
      Put Y in write buffer, even if not written (set a "no write" flag in that case) so it can be found if a request for Y occurs soon.
  - Search write buffer for X, remove X from buffer and bring to cache.