<https://www.cs.princeton.edu/courses/archive/spr18/cos217/reading/x86-64-opt.pdf>

Note it’s also attached to the exam paper.

Also note that It’s easy to find updated versions of the doc, but it must be “September 2014”, as things relevant have changed.

1.a)

Did this get taken out the course?

Otherwise, assuming contiguous page allocation. Allocate 4096 \* 128 \* 4 = 2097152 bytes (4KB page size \* 128 entries \* 4 ways), get enough pages to fill the TLB. e.g.

char[2097152];

Then any allocations after would cause associativity conflicts. That’s a heck of a lot of memory so might be missing something no longer taught?

b)

L2 cache is physically indexed. So associativity conflicts depends on frame (physical) addresses distribution. And since frames are often not allocated contiguously (Operating System behaviour) wrt to their pages. They may conflict.

c)

Hit under miss (probably).

Memory banking (probably).

Two loads possible at once (section 2.2.4.1).

Whole cache lines fetched (not just required byte).

d)

i) Idk what a Streamer is, but if it’s anything like a Stream buffer - to prevent pollution of caches higher in memory hierarchy.

ii) If the other processor is using it (frequently/recently). Maybe don’t allow taking from L1.

2.a)

More instructions would be fetched, decoded, executed, under speculation. And if this branch is a mispeculation, we may do work which is pointless. Thereby reducing the (committed) IPC.

b)

BTB will be polluted causing mispredicts.

c)

Branch behaviour is mostly determined by recent history. Since branches are usually determined by local variables, or other very recent branches. Not code from a long while ago, which is likely out of scope.

So if you use too many history bits, it will index predictions from old history, which just isn’t useful or relevant. And so predictions won’t overlap and cement.

d)

e)

i)

Since the same line has multiple targets. And since entries in a BTB are indexed (usually at least mostly) based on the PC, the targets will overlap. Branch predictor will guess targets alternatively and rarely be correct.

ii)

Call the zero case from a seperate PC.

Int n = rand() % 100;

If (!(n & 0x01)) {

n = 0;

Void (\*f)() = handlers[n];

(\*f)();

} else {

Void (\*f)() = handlers[n];

(\*f)();

}

3.a)

i)

The result of many iterations depends on the result of other (previous) iterations. Since after i>=8 the loop uses A[7], which was set when i=7

ii)

Since we do a write after read within each iteration.

A[i] = A[i] + …

So the set value must occur after the expression calculation.

iii)

We can execute half the iterations in parallel. (0 <= i <= 7 in parallel, then 7 < i < 16 in parallel). This has a max potential for 8x speed up (depending on hardware).

b)

Removed from syllabus (I’m pretty sure?).

c)

We need to reuse the same cache line/portion of memory across many processors, which cannot be written to in many caches. Processors may try to invalidate each others copies / locks may prevent each others work.

4. a)

As if it is too small, it will bottleneck the execution time. As instructions will have to stall waiting to gain a spot in the RUU.

b)

Number of execute units.

Load store queue.

Instruction fetch queue.

c)

We can fetch more instructions at once - increasing instantaneous.

We are less likely to encounter capacity caused misses, which use a lot of energy.

d)

Compiler may reduce number of instructions executed in total, but it would be difficult to reduce number of loads, so spend more time stalling (reduces IPC).

If we count mispredicted instructions towards IPC (?), a bad compiler may allow often mis-executed code, which may have few stalls.

e)

Bad cache design can cause load misses. So we won’t have all the data ready always, to fully saturate the integer arithmetic units. So the optimum could then be less.

f)

The clock rate is determined by the slowest pipeline stage. So if we configure the simulator to have any one very complicated stage (e.g. lots of branch prediction hardware). We may find the clock rate should have been affected by our configuration, leading to misleading results.