Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.313 Contemporary Computer Design

Problem Set #9 Solutions Fall, 1995

1. a) The average memory latency is the time taken for an L1 hit times the percentage we hit L1, plus the latency for a miss. The miss penalty is the L1 miss rate times the sum of the L2 latency times the L2 hit rate and the memory latency times the L2 miss rate.

$T_{ave} = lat_{L1}(1 - miss_{L1}) + miss_{L1}[lat_{L2}(1 - miss_{L2}) + miss_{L2}lat_{mem}]$

b)

System	L1 cache lat. ns	size	L2 cache lat. ns	size	Memory lat. ns	Average Memory lat. ns
Linux i586@120	8	8K	107	256K	150	11.97
FreeBSD i586@133	7	8K	111	256K	181	11.17
Linux i686@167	12	8K	36	256K	194	12.99
UNIX SV x86at@165	14	8K	34	256K	196	14.83
AIX rs6000-990@71	–	–	13	256K	260	14.24
SunOS-5.5 sun4u@167	6	16K	42	512K	270	7.11
OSF1 alpha@147	12	8K	67	512K	291	14.24
Linux alpha@275	6	8K	46	96K	357	7.72
AIX powerpc@134	6	16K	164	512K?	394	10.77
OSF1 alpha@303	3	8K	66	4M	400	5.55
HP-UX 9000/819@99	–	–	10	1M?	430	11.26
IRIX IP22@250	8	16K	64	2M	1170	9.77
IRIX IP19@250	8	16K	64	4M	1189	9.76
SunOS-5.5 sun4d@50	20	8K	140	1M	1236	24.93

Note that I assumed a miss rate of 0.27% for the 2M cache and 1% for the 96k cache.

c) The easy way to think about this problem is to consider what the system does every 30ms. If the no-page-fault system executes some number of instructions and the page-fault system takes one page fault, system performance has dropped by half in that it has executed half the number of instructions. So we are allowed to have one page fault every 15ms, and we know that the system makes a memory reference every 6ns, so we have made $15ms/6ns = 2.5 \times 10^6$ memory references for every page fault. The page fault rate is the reciprocal of that, or 4×10^{-7}.

The 4ns latency is essentially useless information, although you could have interpreted the problem to mean that every 6ns a request comes along and the

4ns latency is tacked onto that so references really come every 10ns, so we have $15ms/10ns = 1.5 \times 10^6$ memory references for every page fault, for a rate of 6.7×10^{-7}.

Either way, we can see that page faults must be very rare if the system is going to have reasonable performance at all. Disk thrashing (where many pages in a row fault because of an undesirable access pattern) is especially bad and can bring a system to its knees. Remember, moving parts are bad.

2. a) In many machines with a short pipeline, branches must have at least one delay slot since the target address is not known until the "R" stage. To have one delay slot, the machine must have a branch condition evaluation unit (generally a subtracter and comparator) in the R stage. For longer pipelines, the value of the condition may not be known for a few more cycles beyond when the target address is known, so a prediction technique can be used. In general, 90% of backward and 50% of forward branches are taken, so the machine can assume backward branches are taken. The distance of the branch is unlikely to be useful since short branches are often part of "if-then-else" clauses and difficult to predict.

b) Using a "likely" or "unlikely" instruction allows the compiler to put the first (few) instructions of the branch target in the branch delay slot(s). The implementation must then be able to nullify the consequences of those instructions if the branch is not taken. Branch delay slots can only be filled with useful instructions about 50% of the time, so an instruction level hint for branch prediction is very useful.

c) Typical history schemes for dynamic branch prediction store a bit or two in the instruction cache to indicate the branch history of the instruction. Using two bits allows a frequently taken branch which is not taken once to only mispredict once rather than twice (once when the history says it should be taken and then once when the single bit of history says it should not be taken.)

d) Using conditionally executing instructions saves the branch delay slots as well as the branch instruction itself. Taking the absolute value using branches requires two instructions plus the number of branch delay slots (which may be large), whereas a conditionally executing instruction has no delay slots to fill and only requires one instruction.

e) A machine which executes jumps to dynamically computed locations would do well to have a fast "jump-register" instruction. Forwarding from the ALU stage to the register stage for next-address selection helps, as does a table or information in the instruction for the likely target.

f) Increasing bandwidth allows many instructions to be brought into the cache at once. Bringing in a block of many sequential instructions through a high bandwidth, (relatively) long latency channel is perfectly adequate for an instruction cache, and because of the spatial locality, these sequential instructions are the most likely to be needed. The cache can therefore be made smaller and not reduce the hit rate (much).