Mid-term Exam, 2012 spring, PART II

Two load-store architecture machines, M1 and M2, have the following characteristics:

	M1	M2	
CR	4 GHz	2 GHz	
CPI-*alu*	2	1	ALU operations
CPI-*br*	5	3	Branches
CPI-*mem*	20	7	Loads and Stores

The CPIs are averaged over all instructions in each class. Given an execution trace of any program P, let a be the number of ALU operate instructions in P's trace. Let b be the number of branch instructions and m the number of memory access instructions in the trace. Assume a, b, and m are non-zero.

Q. Determine the characteristics of an execution trace in terms of a, b, and m, such that M2 out performs M1. What sort of a program might P be? That is, what kind of job would have these characteristics? Hint: use the basic processor performance equation and the speed-up formula of M1 versus M2 to find an expression relating a, b, and m.

$$1 < S_{M2\text{-}M1} = \frac{T_{M1}}{T_{M2}} = \frac{n\,(a(2) + b(5) + m(20))(1/4\,GHz)}{n\,(a(1) + b(3) + m(7))(1/2\,GHz)} = \frac{2a + 5b + 20m}{a + 3b + 7m}\left(\frac{1}{2}\right)$$

$$\Rightarrow 2(a + 3b + 7m) < (2a + 5b + 20m)$$

$$\Rightarrow b < 6m$$

the ALU instructions are irrelevant. We can have at most 6 branches for every load or store. Any program w/ $O(n)$ branching wrt to n data reads and writes will not work. Streaming multi-media might work.

Two machines, M1 and M2, implement the same ISA:

Machine	CPI-A	CPI-B	CPI-C	CR
M1	1	2	8	2.0 GHz
M2	1	4	6	3.0 GHz

$$\overline{CPI}_1 = (0.6(1) + 0.3(2) + 0.1(8)) = 2$$

$$\overline{CPI}_2 = (0.6(1) + 0.3(4) + 0.1(6)) = 2.4$$

There are n instructions in trace T: 60% are class A, 30% are class B, and 10% are class C.

Q. What are the MIPS ratings for both machines for T? Recall MIPS = n / (time of execution) expressed in millions of instructions per second. What is the speed-up of M1 w.r.t. M2?

$$T_1 = n\,\overline{CPI}_1\,(1/CR_1) = n(2)(1/2\,G) \qquad MIPS_1 = (n/10^6)/(T_1 = n/10^9) = 10^3\ MIPS$$

$$T_2 = n(2.4)\left(\frac{1}{3G}\right) \qquad MIPS_2 = \frac{n}{10^6}/(n(2.4)(1/3G)) = (3/2.4)\,10^3 = 5/4\ 10^3\ MIPS$$

$$S_{1\text{-}2} = T_2/T_1 = \frac{MIPS_1}{MIPS_2} = \frac{10^3}{5/4\ 10^3} = 4/5$$

Q. Suppose we want to make the slower machine as fast as the faster machine, but we can only improve the execution time of one class of instruction. Find the minimum execution improvement necessary for one class of instruction. That is, for which machine and which class of instruction should we improve performance and by how much, such that the least improvement is needed? What is the new CPI for this class?

We want $S_{1-2} = T_2/T_1 \geq 1$ or $n\,CPI_2(1/CR_2) \geq n\,CPI_1(1/CR_1)$

$\Rightarrow CPI_2\left(\frac{CR_1}{CR_2}\right) \geq CPI_1$ or $(2.4)(2/3) \geq (0.6(1/x) + 0.3(2/y) + 0.1(8/z))$

$\Rightarrow \left(\frac{2\cdot 3\cdot 4\cdot 2}{2\cdot 5\cdot 3}\right) = 8/5 \geq (0.6/x + 0.6/y + 0.8/z)$ where x, y, z are speedups

for classes A, B, C. Biggest impact is z w/ $x = y = 1$. $\Rightarrow 16 \geq 6 + 6 + 8/z$

$\Rightarrow z \geq 2$: double performance of class C. $\Rightarrow CPI_1 = (0.6 + 0.6 + 0.4) = 1.6$

Q. Suppose we can augment the ISA with MMX instructions that can operate on multiple, short, operands in parallel 10 times faster than without the MMX instructions. What sort of instruction mix would result in a speed-up of 2 for M1? For M2? That is, given x% of instructions can be executed as MMX operations, find x that gives a speed-up of 2.

We don't know which class MMX applies to, but suppose x% of all can be MMX.

$$S^1_{new\text{-}old} = T_{old}/T_{new} = \frac{W_{mmx}/V + W_{non}/V}{W_{mmx}/10V + W_{non}/V} = \frac{xW + (1-x)W}{xW/10 + (1-x)W} = \frac{1}{x/10 + (1-x)} = 2$$

$\Rightarrow 10 = 2(x + 10(1-x)) \Rightarrow 5 = 10 - 9x \Rightarrow x = 5/9$ Same goes for $S^{12}_{new\text{-}old}$.

We wish to compare the performance of two machines, X, Y. We have 3 benchmark programs with data, b1, b2, b3. We have execution times for each benchmark on each machine and on a reference machine W: T-x-1, T-x-2, T-x-3, T-y-1, T-y-2, T-y-3, T-w-1, T-w-2, T-w-3.

Q. Give an expression for each machine that sums up its performance relative to the reference machine. Show that the ratio of the summary measures of X and Y is a summary measure of the relative speed-ups of the two machines. Suppose W is particularly slow for b2. Would your comparison change if we used reference machine W2 which runs b2 four times faster?

$$m_x = \sqrt[3]{\left(\frac{T\text{-}w\text{-}1}{T\text{-}x\text{-}1}\right)\left(\frac{T\text{-}w\text{-}2}{T\text{-}x\text{-}2}\right)\left(\frac{T\text{-}w\text{-}3}{T\text{-}x\text{-}3}\right)}$$

$$m_y = \sqrt[3]{\left(\frac{T\text{-}w\text{-}1}{T\text{-}y\text{-}1}\right)\left(\frac{T\text{-}w\text{-}2}{T\text{-}y\text{-}2}\right)\left(\frac{T\text{-}w\text{-}3}{T\text{-}y\text{-}3}\right)}$$

$$\frac{m_x}{m_y} = \sqrt[3]{\frac{T\text{-}y\text{-}1 \cdot T\text{-}y\text{-}2\, T\text{-}y\text{-}3}{T\text{-}x\text{-}1 \cdot T\text{-}x\text{-}2 \cdot T\text{-}x\text{-}3}} = \sqrt[3]{S^{b_1}_{x\text{-}y} \cdot S^{b_2}_{x\text{-}y} \cdot S^{b_3}_{x\text{-}y}}$$

Because T-W-2 cancels, it has no effect.

We run program P on two different machines, a 1-core processor and an 8-core processor, with the following results. CPI_*i* is the average cycles per instruction **for a single core** on an *i*-core processor:
CPI_1 = 3/2, CPI_8 = 2

The number of instructions executed **per core** on an *i*-core processor is *n_i*:
*n*_1 = 20, *n*_8 = 3 (in Giga-instructions)

We can adjust the voltages and clock rates:
CR = 4 GHz at v = 1 Volt, or CR = 1 GHz at v = 1/2 Volt

Power consumption is = (4)(v * v) CR (1/GHz) Joules/sec (Watts) **per core**.
(Recall, energy is measured in Joules.)

Q. Find the time to run P and the total energy consumed in all four cases: (1-core @ 4 GHz and 1 GHz; 8-core @ 4 GHz and 1 GHz). Recall, energy = power * time. What is the maximum ratio of energy consumption to get the job done. What is the maximum speed-up of the fastest versus the slowest? Which configuration offers the best trade-off? Hint: use the basic processor performance equation to get the execution time.

$T_{1\text{-}1V} = n_1 CPI_1 \left(\frac{1}{4}\text{ GHz}\right) = (20)\left(\frac{3}{2}\right)\left(\frac{1}{4}\right)10^{-9} = \frac{15}{2}$ s $\qquad E = \left(\frac{15}{2}\text{s}\right)\left(4(1)^2(4)\frac{\text{J}}{\text{s}}\right) = 120$ J

$T_{1\text{-}\frac{1}{2}V} = n_1 CPI_1 \left(\frac{1}{(1)}\text{ GHz}\right) = (20)\left(\frac{3}{2}\right)(1) = 30$ s $\qquad E = (30\text{s})\left(4\left(\frac{1}{2}\right)^2(1)\frac{\text{J}}{\text{s}}\right) = 30$ J

$T_{8\text{-}1V} = n_8 CPI_8 \left(\frac{1}{4}\right)10^{-9} = (3)(2)\left(\frac{1}{4}\right) = \frac{3}{2}$ s $\qquad E = \left(\frac{3}{2}\text{s}\right)\left(4(1)^2(4)\frac{\text{J}}{\text{s}}\right)8 = 172$ J

$T_{8\text{-}\frac{1}{2}V} = n_8 CPI_8 (1)10^{-9} = (3)(2)(1) = 6$ s $\qquad E = (6\text{s})\left(4\left(\frac{1}{2}\right)^2(1)\frac{\text{J}}{\text{s}}\right)8 = 48$ J

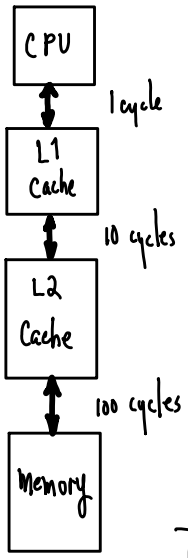
$S_{max} = \frac{T_{1\text{-}\frac{1}{2}V}}{T_{8\text{-}1V}} = \frac{30}{\frac{3}{2}} = 20$

$\frac{E_{max}}{E_{min}} = \frac{E_{8\text{-}1V}}{E_{1\text{-}\frac{1}{2}V}} = \frac{178}{30} \approx 6$

Best trade-off $\Rightarrow$ 8-core, $\frac{1}{2}$V : $S_{(8\text{-}\frac{1}{2}V)\text{-}(1\text{-}\frac{1}{2}V)} = \frac{30\text{s}}{6} = 5$

$\frac{E_{(8\text{-}\frac{1}{2}V)}}{E_{(1\text{-}\frac{1}{2}V)}} = \frac{48}{30} = \frac{4\cdot3\cdot4}{3\cdot2\cdot5} = \frac{24}{15} = 1\frac{9}{15} \approx 67\%$ more

A load-store machine M has a two-level cache heirarchy. On an L1 hit, instruction fetch completes in time for a non-memory instruction (not load/store) to be fetched and complete execution in one cycle. For a load/store instruction two requests are sent to L1 sequentially, one for fetch and one for data access. If both hit in L1, execution takes two cycles. If L1 misses, the first cycle detects the miss, and then 10 cycles are required to detect a hit or miss in L2. If L2 hits, then L1 completes the access in one more cycle. But if L2 misses, memory access takes 200 cycles, then 10 cycles later L2 sends a ready signal to L1, and L1 then completes the access in one more cycle. The CPU stalls until all accesses are complete.



Q. Given clock rate CR how much time is required to execute a non-memory instruction that hits in L1? How much time to execute a memory instruction that hits in L1 for both fetch and data? What is the L1 hit time per access?

T_h = 1 cycles $(1/CR)$ $T_{h/h}$ = 1 + 1 cycles $(1/CR)$

Per access, 1 cycle per hit.

Q. How much time is required for a non-memory instruction to complete execution if it misses in L1 but hits in L2? If it misses in L2? Write each as a sum of individual components.

T_{mh} = 1 + 10 + 1 cycles $(1/CR)$ T_{mm} = 1 + 10 + 200 + 10 + 1 cycles $(1/CR)$

Q. What is the miss penalty for an access (instruction fetch or data) that misses in L1 and hits in L2 (aka, L2 hit time)? What is the miss penalty for an access to L2 that misses?

$T_{1,penalty}$ = 11 cycles $T_{2,penalty}$ = 210 cycles

Penalty time is the extra time we have to spend to access the next lower level. Every instruction has to hit L1. If we miss L1 and hit in L2 we pay an extra 10+1 cycles. An L2 miss incurs that expense plus another 200+10 cycles.

Q. Give an expression for average instruction execution time in terms of the quantities above and miss rates for L1 and L2.

There are several ways to look at this. An intuitive approach is as follows:

n instructions, T total execution time $\Rightarrow$ $\bar{T} = T/n$ is average time per instruction.

Some are load/store, the rest are not: $n = (\%LS)n + (1-\%LS)n$, $\%LS$ = fraction that are load/store. $T = \sum_{LS} T_i + \sum_{non\text{-}LS} T_i$. Let's tackle the second term: $m = (\%LS)n$

$e = (1-\%LS)n$ are the number of LS instructions and non-LS instructions.

$$\sum_{\text{non-LS}} T_i = \underbrace{e(HR_1)}_{\text{number that hit in L1}}(\underbrace{T_1}_{\text{L1 hit time}}) + \underbrace{e(MR_1)(HR_2)}_{\text{miss in L1 and hit in L2}}(\underbrace{T_1+T_2}_{\text{L1 miss + L2 hit}}+T_1) + \underbrace{e\,MR_1MR_2}_{\text{miss both}}(\overbrace{T_1}^{\text{L1 miss}}+\overbrace{T_2}^{\text{L2 miss}}+\overbrace{T_m}^{\text{mem}}+\overbrace{T_2}^{\text{L2 hit}}+\overbrace{T_1}^{\text{L1 hit}})$$

$$\overline{T}_{\text{non-LS}} = \frac{1}{e}\sum_{\text{non-LS}} T_i = (1-MR_1)(T_1) + MR_1(1-MR_2)(2T_1+T_2) + MR_1MR_2(2T_1+2T_2+T_m)$$

$$= T_1 + (-MR_1T_1) + 2MR_1T_1 + MR_1T_2 - MR_1MR_2 2T_1 - MR_1MR_2T_2$$
$$+ 2MR_1MR_2T_1 + 2MR_1MR_2T_2 + MR_1MR_2T_m$$

$$= T_1 + MR_1T_1 + MR_1T_2 + MR_1MR_2T_2 + MR_1MR_2T_m$$

$$= T_1 + MR_1(T_1+T_2 + MR_2(T_2+T_m))$$

$$= \underbrace{T_1}_{\text{every instruction has to hit L1}} + MR_1(\underbrace{T_{1,\text{penalty}}}_{\text{added time for hitting L2}} + MR_2(\underbrace{T_{2,\text{penalty}}}_{\text{added time for mem access}}))$$

We've just derived the cache performance equation. We can use it to deal with the LS instructions. Because there are two accesses per instruction, we multiply by 2.

$$\overline{T}_{LS} = \frac{1}{m}\sum_{LS} T_i = \frac{1}{m}\sum_{LS}\left(T_{\text{fetch+exec}_i} + T_{\text{data-access}_i}\right) = \frac{1}{m}\sum_{LS} T_{\text{fetch+exec}_i} + \frac{1}{m}\sum_{LS} T_{\text{data-access}_i}$$

$$= \overline{T}_{\text{fetch+exec}} + \overline{T}_{\text{data-access}} = 2(\overline{T}_{\text{non-LS}})$$

$$= 2\left(T_1 + MR_1(T_{1,\text{penalty}} + MR_2(T_{2\,\text{penalty}}))\right)$$

$$T = \sum_{\text{non-LS}} T_i + \sum_{LS} T_i = (e+2m)\left(T_1 + MR_1(T_{1,\text{penalty}} + MR_2(T_{2\,\text{penalty}}))\right)$$

$$(e+2m) = n((1-\%LS) + 2(\%LS)) = n(1+\%LS)$$

$$\overline{T} = T/n = (1+\%LS)\left[1 + MR_1(11 + MR_2(210))\right]$$

Machine M has a split L1 cache and unified L2. Running program P, L1's miss rate is MR_1 = 1/8 (combined) and L2's miss rate is MR_2 = 1/256. Loads and stores account for 1/3 of instructions executed. All misses (loads, stores, and instruction fetches) cause CPU stalls. If both fetch and data access hit in L1, instruction execution time is L1's hit time. L1's fetch and data accesses run in parallel, but L2 processes requests from both serially. The number of instructions executed is *n*, M's clock rate is *CR*, L1's hit time is T_1, L2's is T_2 and memory's access time is T*m*.

Q. In total, how many memory accesses occur (fetches and data R/W)? State the result in terms of the parameters given. Of these, how many hit in L1? How many miss L1?

n instructions $\Rightarrow$ n fetches, $n/3$ load/stores $\Rightarrow$ $4/3\,n$ memory accesses

L1 hits $= (4/3\,n)(7/8)$ L1 misses $= (4/3\,n)(1/8)$

Q. If all miss penalties are 0, what is the total execution time? How many clock cycles?

n instructions $\Rightarrow$ $n \cdot T_1$ cycles $= n \cdot T_1 \cdot (CR)$

Q. In total, many memory accesses (fetches and data R/W) miss in L2? How much main memory access stall time results? How many stall cycles?

$(\text{L1 misses} = (4/3\,n)(1/8))(1/256) = (n/3)(1/2^9) = \text{L2 misses}$

$(\text{L2 miss penalty}) = (\text{L2 misses})(T_m) = (n/3)(1/2^9)\,T_m$ L2 stall time.

$\Rightarrow (n/3)(1/2^9)\,T_m(CR)$ stall cycles.

Q. For an L1 miss that hits in L2, how much stall time is attributable to the L2 access? How many cycles?

all L1 misses must pay an L2 access stall equal to L2's hit time, T_2.

$(\text{L2 access time}) = (\text{L1 misses})\,T_2 = (4/3\,n)(1/8)\,T_2$

$(\text{L2 access cycles}) = (4/3\,n)(1/8)\,T_2(CR)$

Q. What is P's total running time? How many cycles? What is M's average CPI?

$T = T_{exec} + (T_{stalls} = T_{L2\text{-}access} + T_{mem\text{-}access})$

$= n \cdot T_1 + (4/3\,n)(1/8)\,T_2 + (n/3)(1/2^9)\,T_m = n\left(T_1 + \frac{1}{3}\{(1/2)T_2 + (1/2^9)T_m\}\right)$

$\text{avg. CPI} = \frac{\#\text{cycles}}{\#\text{instructions}} = \frac{T(CR)}{n} = \left[T_1 + \frac{1}{3}\{(1/2)T_2 + (1/2^9)T_m\}\right]CR$

Q. Suppose memory access time does not improve while processor improvements double the clock rate. Assume L1 and L2 access times are also halved. Show an expression for the new CPI in terms of the old clock rate. What effect does this have on average CPI?

$$CPI_{new} = \left[\frac{T_1}{2} + \frac{1}{3}\left\{(1/2)\frac{T_2}{2} + (1/2^9)T_m\right\}\right](2CR)$$

$$= \left[T_1 + \frac{1}{3}\left\{(1/2)T_2 + (1/2^9)2T_m\right\}\right](CR) = CPI_{old} + (\tfrac{1}{3})(1/2^9)\,T_m\,(CR)$$

Because 2nd term is $(\frac{1}{3})(\frac{1}{2^9})\,T_m$ ($\times 10^9$), $\times$ a small integer (3?), T_m is the order of 128 ns, and CPI is small integer (2?), overall effect is $\frac{(1/4)}{2} = 1/8$ increase in CPI.

Q. Show the speed-up of the new processor relative to the unimproved version.

$$S = \frac{T_{old}}{T_{new}} = \frac{n\,CPI_{old}\,(1/CR)}{n\,CPI_{new}\,(1/2CR)} = \frac{CPI_{old}\;2}{CPI_{old}\,(9/8)} = 2(\tfrac{8}{9}) = 16/9 = 1\tfrac{7}{9} \approx 75\%\ \text{faster}$$

Q. Assuming T_2 = 10 T_1 and T_m = 10 T_m, what qualitatively is the effect of the improvement? What does Amdahl's Law suggest in regard to which aspect of performance should be improved? What if CR keeps doubling every two years?

$T_2 = 10\,T_1 \quad T_m = 10\,T_2$

$$S = \frac{1 + (1/3)\{(1/2)(10) + (1/2^9)(100)\}}{1/2 + (1/3)\{(1/2)(5) + (1/2^9)(100)\}} \cong \frac{1 + (1/3)\{5 + (1/5)\}}{1/2 + (1/3)\{\frac{5}{2} + (1/5)\}} = \frac{1 + 24/15}{\frac{1}{2} + (\frac{1}{3})(27/10)} = \frac{41/15}{1/2 + 9/10}$$

$$= \frac{41/15}{7/5} = \frac{41}{7}(\tfrac{1}{3}) = \frac{41}{21} = 1\tfrac{20}{21} \approx \text{doubles}$$

$$T \cong 1 + (\tfrac{1}{3})5 + \tfrac{33}{512}$$

↑ L2 performance is biggest % of work: Speed-up L2 has biggest effect

doubling CR n Times:

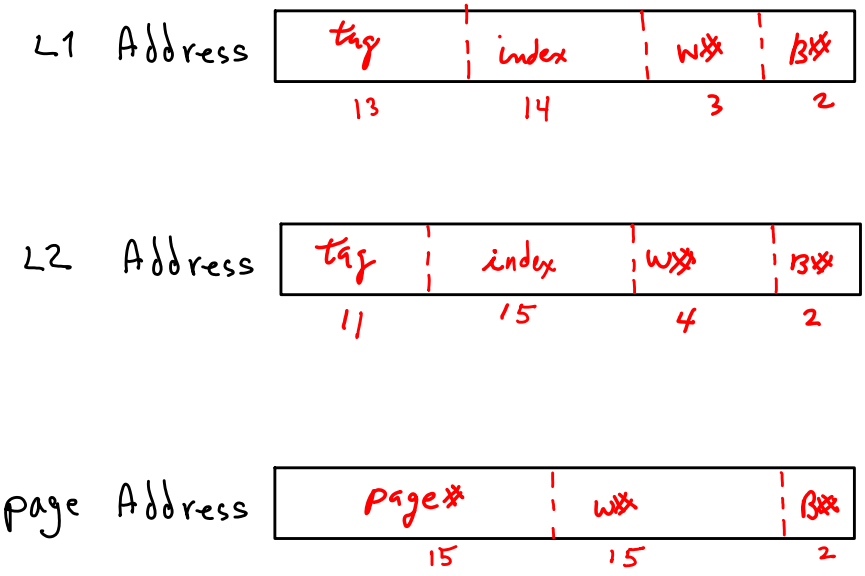
$$T = 1/2^n + \frac{1}{3}\left\{\frac{1}{2}\,10/2^n + 1/2^9\,(100)\right\}$$

$$\cong 1/2^n + 5/3/2^n + 1/15 = 8/3/2^n + 1/15$$ mem becomes most weighty when

$\frac{8/3}{2^n} < \frac{1}{15}$ or $40 < 2^n$ or after about 6 CR doublings.

Machine M has 32-bit virtual and physical addresses, 32-bit words, and memory is byte addressable. Pages are 128 kB. L1 is a 512-kB direct mapped cache and L2 is a 8-MB 4-way set-associative cache. Cache blocks are 8 and 16 words for L1 and L2, respectively.

Q. In the address bit-field diagrams below, show the allocation of address into bit-fields for byte number within a word, word number within a cache block, and tag, showing the number of address bits in each bit-field as it applies to L1, L2, and virtual pages.



$512\ kb\left(\frac{block}{32B}\right) = \frac{2^9 2^{10}}{2^5} = 2^{14}$ blocks

$\Rightarrow$ 14 index bits

8 words = 3 bit W#

4B words $\rightarrow$ 2 bit B#

32 − 19 = 13-bit tag

$8\ MB\left(\frac{1}{4\ ways}\right) = 2MB/way\ (DM)$

$2\ MB\left(\frac{1\ block}{64B}\right) = \frac{2^{21}}{2^6} = 2^{15}$ blocks

$\Rightarrow$ 15 index bits

$\left(\frac{64B}{block}\right)\left(\frac{w}{4B}\right) \Rightarrow 2^4$ word/block

$\rightarrow$ 4-bit W#

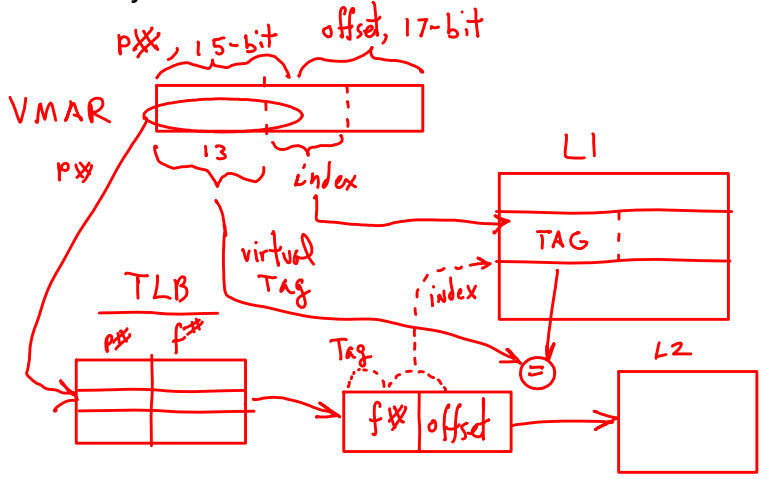
32 − 21 = 11-bit tag

128 kB page $\Rightarrow 2^7 \cdot 2^{10}$ B/page $\Rightarrow$ 17 bits for page offset, 15-bit page#.

$\Rightarrow$ words are 4B $\Rightarrow$ 2 bit B# field

$\Rightarrow$ 15 bits for word#

Q. Comment on the feasibility of using virtual indexing in this system. (Recall, virtual indexing indexes into the caches before address translation.) Is there a performance problem w.r.t. to the TLB in this system?



Because page# overlaps 2 index bits, physical indexing results in an L1 delay until after TLB translation is done. Virtual indexing solves this problem. In that case, if L1 is physically tagged, the entire frame# must be used for the tag as the 2 physical index bits are not the same as the virtual index bits, and a partial frame# would not uniquely identify which block was present. Virtual tagging works and L1 access is fast. L2 can be physically indexed and tagged.

Q. Comment very briefly on the performance tradeoffs of each cache feature.

1. larger cache blocks — More spatial locality => lower miss rate; large block => higher latency replacement

2. more cache levels — lower miss penalty; more complexity + higher miss rates

3. more associativity — lower miss rate; slower access; more complex; more power

4. virtual tagging — faster L1 hit time + parallel translation; synonyms

5. larger total cache size w/ larger blocks

   Lower miss rate in spatially local code + more efficient (pipelined) block reads from memory; loss of efficiency for temporally local code + longer miss penalty + more collisions.

6. larger total cache size w/ smaller blocks

   Lower miss rate spatially local + shorter miss penalty + less collisions; less efficient memory access

7. write buffering w/ a searchable buffer — shorter write-miss penalty + no delay for reads that hit in buffer; more complexity and energy.

8. write-back — Lower memory bandwidth + more efficient memory access; more complexity

9. no-allocate/write-through — Faster writes + improved miss rate if active read blocks get hits that written block wouldn't get.

10. larger pages — Lower page miss rate + smaller page tables; longer replacement latency + more internal fragmentation.

11. PID fields in caches and TLBs — Less overhead flushing caches and TLB entries on context switch; more complexity and energy for PID matching + more space.

12. word-level valid bits within cache blocks — For write-back, faster write miss (no need to load block)+ less load penalty (on write-back).

13. valid bits in TLB entries — Provides for simple cache flushing.

A system uses an inverted page table on a machine with 64-bit virtual addresses. The inverted page table is hashed into, and if there is a miss (page number not found), the page number is re-hashed with a second hash function and the table probed again. If that fails, another rehash is applied, and so on. If after *k* re-hashes the page number is still not found, the table must be linearly searched. Each entry refers to a specific frame: e.g., page number 10 in entry 3 means that page 10 is in frame 3. A "free" bit in an entry indicates whether the frame contains a valid page.

If the requested page is not in memory, a second table indicating the page number and its corresponding disk address must be read to fetch the page from disk.

Q. How many entries in the inverted table if physical memory is 32 GB and pages are 4MB?

$32\,GB = 2^5 \cdot 2^{30}$

$4\,MB = 2^2 2^{20}$

$32GB\left(\frac{frame}{4\,MB}\right) = \frac{2^{35}}{2^{22}} = 2^{13}$ frames $\Rightarrow 2^{13}$ entries in PT. $= 8k$

Q. How big is the table? (Round the size of an entry up to the nearest integer number of bytes.)

Each entry has a virtual page#. 64 b address $\Rightarrow 2^{64}$ B virtual memory @ 4 MB/page

$\frac{2^{64}}{2^{22}} = 2^{42}$ pages $\Rightarrow$ 42 bit page# (+ 22 bit page offset)

≈ 6 B entrys $\Rightarrow 2^{13}$ PT entries $\left(\frac{6B}{entry}\right) = 6.8kB = 48kB$

Q. How many levels would be required for a multi-level page table scheme (not inverted), assuming each sub-table at any level fits in a single page?

Entries contain frame number $\rightarrow 2^{13}$ frames $\Rightarrow$ 13-bit frame# $\approx 2B$

$\left(\frac{4\,MB}{page}\right)\left(\frac{entry}{2B}\right) = 2M$ entries per page.

n levels $\Rightarrow (2M)^n$ lowest-level entries $= 2^{42}$ pages

$\Rightarrow 2^{21 \cdot n} = 2^{42} \Rightarrow n = 2$

Q. How big is the disk map per process? That is, each process has its own pages, and each has its own map from page numbers to disk addresses for pages that are not in memory but on disk. (NB--Give a maximum, minimum, and some expected size for the table.) The disk is 64 GB, and each addressable unit on the disk contains 4kB.

Total (max) pages per process $= 2^{42} \Rightarrow 2^{42}$ entry disk map.

Suppose disk access unit is 1 kB, disk is 128 GB $\Rightarrow 128\,G/k = 128\,M$ disk addresses,

$\Longrightarrow$ disk address = 27 bits $\approx 4B$/entry $\Rightarrow 2^{42}(4B)$ disk map $= 2^{14}2^{30}B = 16k$ GB

Min pages = 1 $\Rightarrow$ 1-entry map = 4B.

Avg: ? 128 GB? $\Rightarrow 2^7 2^{30}/4\,MB = \frac{2^{37}}{2^{22}} = 2^{15}$ pages $\Rightarrow 32k(4B) = 128k\,B$ disk map.