Homework Solutions

Chapter 1

1. Using the resources of the World Wide Web, list the top five reported benchmark results for SPECINT2000, SPECFP2000, and TPC-C.

   Current results are available from http://www.spec.org and http://www.tpc.org

2. Graph SPECINT2000 vs. processor frequency for two different processor families (e.g., AMD Athlon and HP PA-RISC) for as many frequencies as are posted at www.spec.org. Comment on performance scaling with frequency, pointing out any anomalies and suggesting possible explanations for them.

   Current results are available from http://www.spec.org; performance scaling will vary depending on processor family. Look for reasoning based on the following factors that do not scale with processor frequency: memory bandwidth, memory latency, new microarchitectural features, compiler improvements.

3. Explain the differences between architecture, implementation, and realization. Explain how each of these relates to processor performance as expressed in Equation 1-1.

   Architecture specifies the interface between the hardware and the programmer and affects the number of instructions that need to be executed for a particular program or algorithm. Implementation specifies the microarchitectural organization of the design, and determines the instruction execution rate (measured in instructions per cycle) for that particular design. The realization actually maps the implementation to silicon, and determines the cycle time. The product of the three terms determines performance.

4. As silicon technology evolves, implementation constraints and tradeoffs change, which can affect the placement and definition of the dynamic-static interface (DSI). Explain why architecting a branch delay slot (as in the MIPS architecture) was a reasonable thing to do when that architecture was introduced, but is less attractive today.

   When the MIPS ISA was created, a single-chip processor could reasonably contain a straightforward pipelined implementation like the MIPS R2000. There was no room on chip for branch predictors or decoupled fetch pipelines that could be used to mask branch latency. At the same time, processors were already fast enough to tolerate some additional complexity in the compilation process for finding independent instructions to place in the delay slot. Hence, a branch delay slot made sense, since it reduced the penalty due to branch instructions without negatively affecting implementation or realization. However, as silicon technology enabled more advanced pipelines, including superscalar fetch, other techniques for masking the branch latency became available and more attractive. Once this had occurred, the branch delay slot became an archaic special case that only complicated implementation and realization. Hence, more recent ISAs like PowerPC and Alpha omitted the branch delay slot.

5. Many times, implementation issues for a particular generation end up determining tradeoffs in instruction set architecture. Discuss at least one historical implementation constraint that explains why CISC instruction sets were a sensible choice in the 1970s.

   Possible answers to this discussion question include the high cost of memory for program storage, which favored dense CISC encodings. In addition, limited on-chip implementation resources forced various dataflow components to be reused for various purposes. As a result, microcoded control over multicycle execution of each instruction was necessary. Once you assume multicycle microcoded execution, the additional decoding overhead due to CISC encoding becomes less important.

6. A program's run time is determined by the product of instructions per program, cycles per instruction, and clock frequency. Assume the following instruction mix for a MIPS-like RISC instruction set: 15% stores, 25% loads, 15% branches, and 30% integer arithmetic, 5% integer shift, and 5% integer multiply. Given that stores require one cycle, load instructions require two cycles, branches require four cycles, integer ALU instructions require one cycle, and integer multiplies require ten cycles, compute the overall CPI.

TABLE 1 CPI computation

Type	Mix	Cost	CPI
store	15%	1	0.15
load	25%	2	0.50
branch	15%	4	0.60
integer	30%	1	0.30
shift	5%	1	0.05
multiply	5%	10	0.50
Total			**2.10**

7. Given the parameters of Problem 6, consider a strength-reducing optimization that converts multiplies by a compile-time constant into a sequence of shifts and adds. For this instruction mix, 50% of the multiplies can be converted to shift-add sequences with an average length of three instructions. Assuming a fixed frequency, compute the change in instructions per program, cycles per instruction, and overall program speedup.

TABLE 2 CPI computation

Type	**Old Mix**	**New Mix**	**Cost**	**CPI**
store	15%	15%	1	0.15
load	25%	25%	2	0.50
branch	15%	15%	4	0.60
integer & shift	35%	42.5%	1	0.425
multiply	5%	2.5%	10	0.25
Total	**100%**	**105%**		**1.925/105% = 1.83**

There are 5% more instructions per program, the CPI is reduced by 12.7% to 1.83, and overall speedup is 2.1/1.925 = 1.091 or 9.1%.

8. Recent processors like the Pentium 4 processors do not implement single-cycle shifts. Given the scenario of Problem 7, assume that *s* = 50% of the additional integer and shift instructions introduced by strength reduction are shifts, and shifts now take four cycles to execute. Recompute the cycles per instruction and overall program speedup. Is strength reduction still a good optimization?

TABLE 3 CPI computation

Type	**Old Mix**	**New Mix**	**Cost**	**CPI**
store	15%	15%	1	0.15
load	25%	25%	2	0.50
branch	15%	15%	4	0.60
integer	30%	33.75%	1	0.3375
shift	5%	8.75%	4	0.35
multiply	5%	2.5%	10	0.25
Total	**100%**	**105%**		**2.1875/105% = 2.083**

Speedup is now a slowdown: 2.1/2.1875 = 0.96 or 4% slowdown, hence strength reduction is a bad idea.

9. Given the assumptions of Problem 8, solve for the break-even ratio *s* (percentage of additional instructions that are shifts). That is, find the value of *s* (if any) for which program performance is identical to the baseline case without strength reduction (Problem 6).

2.1 = (0.15+0.50+0.60+0.30+0.05x4+0.25 + (1-s)x0.075x1 + sx0.075x4

0.025 = 0.225s => s = 0.111 = 11.1%

10. Given the assumptions of Problem 8, assume you are designing the shift unit on the Pentium 4 processor. You have concluded there are two possible implementation options for the shift unit: 4-cycle shift latency at a frequency of 2 GHz, or 2-cycle shift latency at 1.9 GHz. Assume the rest of the pipeline could run at 2 GHz, and hence the 2-cycle shifter would set the entire processor's frequency to 1.9 GHz. Which option will provide better overall performance?

    4-cycle shifter: Time per program = 1.05 IPP x 2.1875 CPI x 1/2.0GHz = 1.15e-9

    2-cycle shifter: Time per program = 1.05 IPP x (2.1875-0.175) CPI x 1/1.9GHz = 1.11e-9

    Hence, 2-cycle shifter is a better option if strength reduction is applied.

    If there is no strength reduction (back to Problem 6):

    4-cycle shifter: Time per program = 1.00 IPP x 2.25 CPI x 1/2.0GHz = 1.125e-9 s

    2-cycle shifter: Time per program = 1.00 IPP x 2.15 CPI x 1/1.9GHz = 1.132e-9

    Hence, the 4-cycle shifter is a better option. Overall, the best choice is still strength reduction with a 2-cycle shifter @1.9GHz.

11. Using Amdahl's law, compute speedups for a program that is 85% vectorizable for a system with 4, 8, 16, and 32 processors. What would be a reasonable number of processors to build into a system for running such an application?

    $S_{n,p}$ = 1/[p/n + (1-p)]: p = 0.85; n=4, 8, 16:

    $S_{4,0.85}$ = 2.76, $S_{8,0.85}$ = 3.90, $S_{16,0.85}$ = 4.92, $S_{32,0.85}$ = 5.52

    Of these, the 4 processor system achieves the most parallel efficiency (2.76/4), hence that might be a reasonable number.

12. Using Amdahl's law, compute speedups for a program that is 98% vectorizable for a system with 16, 64, 256, and 1024 processors. What would be a reasonable number of processors to build into a system for running such an application?

    $S_{n,p}$ = 1/[p/n + (1-p)]: p = 0.98; n=16, 64, 256, 1024:

    $S_{16,0.98}$ = 12.3, $S_{64,0.98}$ = 28.3, $S_{256,0.98}$ = 42.0, $S_{1024,0.98}$ = 47.7

    Of these, the 16 processor systeam achieves the most parallel efficiency (12.3/16), hence that might be a reasonable number. However, the 256 processor system still gains substantial speedup which is fairly close to the ideal case of 50 (where p is infinite).

13. Replot the graph in Figure 1-8 on page 25 for each of the ILP limits shown in the list of studies in Section 1.4.2. What conclusions can you draw from the graphs you created?

    (graphs not provided).

Conclusions: higher ILP limits make a stronger case for superscalar processors, since the curves for higher ILP limits are less sensitive to the degree of vectorizability (the x-axis).

14. Compare and contrast these two ILP limit studies by reading the relevant papers and explaining why the limits are so different: [Jouppi and Wall 1988] vs. [Wall 1991].

    Discussion question; solution not provided.

15. In 1995, the IBM AS/400 line of computers transitioned from a CISC instruction set to a RISC instruction set. Due to the simpler instruction set, the realizable clock frequency for a given technology generation and the CPI metric improved dramatically. However, for the same reason, the number of instructions per program also increased noticeably. Given the following parameters, compute the total performance improvement that occurred with this transition. Furthermore, compute the break-even clock frequency, break-even cycles per instruction, and break-even code expansion ratios for this transition, assuming the other two factors are held constant.

Performance Factor	AS/400 CISC (IMPI)	AS/400 RISC (PowerPC)	Actual Ratio	Break-even Ratio
	Actual	Actual		
Relative Frequency	50MHz	125MHz	2.5	?
Cycles per instruction	7	3	0.43	?
Relative instructions per program (dynamic count)	1000	3300	3.3	?

Total performance improvement:

TPP_{IMPI} = 1000 IPP x 7 CPI x 1/50MHz = 140us

$TPP_{PowerPC}$ = 3300 IPP x 3 CPI x 1/125MHz = 79.2us

Speedup = 140/79.2 = 1.77 or 77%

Frequency breakeven: 7000/50 = 9900/50y => y = 9900/7000 = 1.41

CPI breakeven: 7000/50 = 23100y/125 => y = 17500/23100 = 0.76

IPP breakeven: 7000/50 = 3000y/125 => y = 17500/3000 = 5.83

Therefore, gaining 41% in frequency, or reducing CPI by 24%, or increasing pathlength by no more than 5.83x (all in isolation) would all have provided constant performance across the IMPI-to-PowerPC transition.

16. MIPS (millions of instructions per second) was commonly used to gauge computer system performance up until the 1980s. Explain why it can be a very poor measure of a processor's performance. Are there any circumstances under which it is a valid measure of performance? If so, describe those circumstances.

The iron law of processor performance tells us that program performance is determined by the product of three terms: instructions per program, cycles per instruction, and time per cycle. MIPS only captures the last two terms, and ignores the instructions per program term. Hence, MIPS comparisons are meaningless if the program changes, if the ISA changes, or if the optimization level of the compiler changes, since any of these can affect the first term. Conversely, MIPS is a meaningful measure if comparing the performance of the same program binary on two processors that implement the same instruction set.

17. MFLOPS (millions of floating-point operations per second) was commonly used to gauge computer system performance up until the 1980s. Explain why it can be a very poor measure of a processor's performance. Are there any circumstances under which it is a valid measure of performance? If so, describe those circumstances.

The iron law of processor performance tells us that program performance is determined by the product of three terms: instructions per program, cycles per instruction, and time per cycle. MFLOPS does not fully capture any of these terms, as it only describes the throughput or execution rate of a subset of the instructions executed by a program (specifically, the subset that are floating-point arithmetic operations). Furthermore, there are variations in the types of floating point operations provided by various instruction sets (for example, some ISAs provide FP divide and square root, while others do not). In these cases, reporting MFLOPS masks the fact that one computer may need to implement a square root function with a sequence of hundreds of instructions while the other can implement it with a single instruction; a much higher MFLOPS rate in the former case can actually result in much lower performance, since the single square root operation still operates faster than hundreds of other instructions.

MFLOPS can be a useful measure when discussing the performance of programs that require a deterministic number of equivalent floating-point operations to be performed. A good example of this is floating-point matrix multiply benchmarks. These require a fixed number of FP multiply and FP add operations that is strictly determined by the size of the matrices being multiplied. Hence, a MFLOPS rating for a processor may be able to accurately predict the performance of such a program. However, even in this case, once the combined sizes of the arrays exceed cache capacity, MFLOPS can be misleading, since performance can be limited by memory bandwidth (i.e. fetching the array elements from memory) rather than by MFLOPS throughput. All in all, MFLOPS is a fairly useless measure of performance.

Terms and Buzzwords

These problems are similar to the "Jeopardy Game" on TV. The "answers" are shown below and you are to provide the best correct "questions". For each "answer" there may be more than one appropriate "question"; you need to provide the best one.

18. A: Instruction-level parallelism within a basic block is typically upper bounded by 2.

Q: What is __Flynn's Bottleneck____________________ ?

19. A: It will significantly reduce the machine cycle time, but can increase the branch penalty.

Q: What is __pipelining or deep pipelining______________ ?

20. A: Describes the speedup achievable when some fraction of the program execution is not parallelizable.

Q: What is _Amdahl’s Law________________________ ?

21. A: A widely-used solution to Flynn’s bottleneck.

    Q: What is _branch prediction______________________ ?

22. A: The best way to describe a computer system’s performance.

    Q: What is _execution time (time per program)____________ .

23. A: This specifies the number of registers, available addressing modes, and instruction opcodes.

    Q: What is _the instruction set architecture (ISA)__________ ?

24. A: This determines a processor’s configuration and number of functional units.

    Q: What is _the microarchitecture______________________ ?

25. A: This is a type of processor that relies heavily on the compiler to statically schedule independent instructions.

    Q: What is VLIW or very long instruction word, also EPIC (explicitly parallel instruction set computer) ?

26. A: This is a type of processor where results of instructions are not available until two or more cycles after the instruction begins execution.

    Q: What is _a superpipelined processor__________________________ ?

27. A: This is a type of processor that attempts to execute more than one instruction at the same time.

    Q: What is _a superscalar processor___________________________ ?

28. A: This important study showed that instruction level parallelism was abundant, if only control dependences could somehow be overcome.

    Q: What is _Riseman & Foster’s 1972 paper____________________ ?

29. A: This is a type of processor that executes high-level languages without the aid of a compiler.

    Q: What is _DEL or direct execution language computer_____________ ?

30. A: This approach to processor simulation requires substantial storage space.

    Q: What is _trace-driven simulation________ ?