

CSSE7013 - Advanced Computer Architecture Tutorial 1 Partial Solutions

Learning Objectives

These solutions are a guide to answering the rest of the questions.

QUESTION 1—Quantitative Techniques

Use the following data [*moved to the solution with extra space added for calculated numbers, shown in italics*] for the original machine, a load-store architecture, for all parts of the question where calculations are required. In a load-store machine, all memory data references are either copies from memory to a register (loads), or copies from a register to memory (store). All ALU operations are on 2 source registers and 1 destination register. The timings assume no cache misses, and are averages based on the possibility that the pipeline may stall (for reasons we'll see in later chapters). CPI of less than 1 (as here) is usually given as IPC as explained in the notes.

- a. A modification to the design is proposed whereby 90% of branches can be predicted accurately, resulting in a 50% reduction in branch CPI when a branch is predicted correctly. However, when a branch is predicted incorrectly, CPI for a branch is increased by 50%. The modification increases clock cycle time (slows down clock speed) by 15%.

- i. What is the CPI of the original machine with the given instruction mix?

instruction categories	dynamic frequency %	clock cycles (no stalls)	contribution to CPI (<i>original</i>)
ALU	43	0.25	<i>0.1075</i>
loads	21	0.5	<i>0.105</i>
stores	12	0.25	<i>0.03</i>
branches	24	1	<i>0.24</i>
total CPI			<i>0.4825</i>

- ii. What is the CPI of the modified machine with the given instruction mix?

instruction categories	dynamic frequency %	clock cycles (no stalls)	contribution to CPI (<i>new</i>)
ALU	43	0.25	<i>0.1075</i>
loads	21	0.5	<i>0.105</i>
stores	12	0.25	<i>0.03</i>
branches (predict)	21.6%	<i>0.5</i>	<i>0.108</i>
branches (mis-predict)	2.4%	<i>1.5</i>	<i>0.036</i>
total CPI			<i>0.3865</i>

- iii. Which is faster, and by how much?

Here we have to apply the CPU performance equation, and note that the instruction count IC has not changed, so we can see the relative speed change by looking at $CPI_{old} \times t_{CC} = 0.4825 \times$ and $CPI_{new} \times 1.15 \times t_{CC} =$

$$1.15 \times 0.3865 = 0.444475 t_{CC}$$

so we can see that the speedup of the new design is $0.4825/0.444475 = 1.086$. What % faster is the new design as compared with the old?

- iv. Now assume the instruction miss rate is 0.8% for the original and 1% for the modified architecture. In both cases, the data miss rate is 1% and a miss costs 200 extra cycles. Redo the CPI calculations for both machines; which is faster—and by how much?

Useful to attempt this now but go back to it when we do memory systems if you don't manage to do it yet.

- b. Recent designs rely on a very fast on-chip interface to minimize the cost of misses from the L1 cache to the L2 cache. A consequence of this design is that the L2 cache size can be limited, as opposed to earlier designs with off-chip L2 caches.

Useful to attempt this now but go back to it when we do memory systems if you don't manage to do it yet.

- i. Consider a 2-level cache, in which the penalty for references to DRAM is constant at 200 cycles, but the penalty for misses to L2 varies. In the 2 design variations, the following figures apply:

- fraction of references which miss to L2 in both designs: 1%
- fraction of references which miss from L2 in small, slower L2 design: 20% (relative)
- penalty for misses to small, fast L2: 10 cycles
- fraction of references which miss from L2 in bigger, slower L2 design: 10% (relative)
- penalty for misses to bigger, slower L2: 20 cycles

- ii. In the light of (i), comment on the trend towards faster but smaller on-chip caches.

QUESTION 2—Amdahl's Law

- a. You are travelling to Sydney by car. Halfway there, you realize you have been going too slowly: in fact, you are taking twice as long as you should. Explain what if anything you can do to get to Sydney on time. Think about how Amdahl's Law applies.

If you have travelled half the distance at half the speed, you have used up all your time so you will need infinite speedup. Look at the Amdahl's Law speedup formula: you should be able to make it apply to this case.

- b. It is proposed that a new fast train be introduced, linking Brisbane and Gold Coast, over a distance of 50km, with 2 stops on the way. Assume the train averages 150km/h while moving, but needs 10min at each stop to load and unload. What is the speedup versus a train which averages 75km/h but which does not stop on the way?

We need t_{fast} and t_{slow} to work out speedup:

$$t_{fast} = t_{moving} + t_{stopped} = 50/150 \text{ h} + 2 \times 10 \text{ min} = 40 \text{ min}$$

$$t_{slow} = t_{moving} = 50/75 \text{ h} = 40 \text{ min}$$

so speedup = 1 (i.e., not improvement)

QUESTION 3—Performance Measurement

- a. To evaluate new computer architectures, you can use back of the envelope calculations (e.g., question 1), running real-world applications on a simulation of a new design or running traces of memory references from applications from an existing machine through a memory simulator. Trace file entries are *type_of_operation memory_address* where type of operation is usually one of *fetch*, *read* or *write*. Explain why any of the performance measures do or don't apply in these situations:
- You are experimenting with variations on the pipeline timing but otherwise the design is very similar.
Trace files could be used here but aren't very accurate because they don't capture details like which specific instructions executed resulting in a particular memory reference pattern. A more detailed architecture simulation would be better here.
 - The pipeline design is just like an existing machine, but you are experimenting with variations in DRAM speed.
Depending how much accuracy you need, either approach could apply.
 - You are planning significant additions to the instruction set with 2 possibilities:
 - the instructions can be generated by an experimental compiler
You could extend an existing simulator if one existed, which is the best way to see how new instructions change performance – a trace file would not have been generated by the new instructions and since the information about what the instructions were in the run which was recorded is lost, you wouldn't even know when the new instructions could have been used.
 - the new instructions are too different from any existing ISA to create an experimental compiler easily
This kind of case is hard to measure in detail – you may have to revert to back of the envelope studies until you can create a compiler, or hand-write machine code, then run it through a simulator.
- b. Another way to measure where time is being spent in a program is to use a *profiler*. A profiler uses various techniques like interrupting the program at random intervals, or taking measurements at specific control points (e.g., procedure call, on any change of control flow). What kinds of architecture measurements do you think profiling could apply to?
A profiler is a much less precise form of measurement. You can pick up big hot spots e.g. a part of the program where the memory hierarchy is causing big delays. This can help e.g., with picking up program behaviours which are bad for caches or the TLB (see later when we look at memory).

QUESTION 4—More Exercise

- Work through the examples in Chapter 1.
- Work through questions at the end of Chapter 1. Warning: the exercises in the book can sometimes be hard e.g. because details are left out, requiring you to make assumptions.