The following problems are mostly open-ended. Diagrams or charts are not needed in the answers. Therefore it is highly recommended to type the answers. Please keep your answers organised and tidy. Most of the questions are taken from the supervision notes from the lecturer. If you are not sure about the answers, you could look up from the Hennessy and Patterson book or directly from the Internet. If you could not find the answer, please make it clear that you are not sure.

1 Computer Architecture Trends

Q(a) “Computer architecture is a science of trade-offs” was the golden quote from Yale Patt. Please list at least three trade-offs in computer architectures. Extra credits if you list the trade-offs which are still in active debate.

Q(b) Briefly describe the CPU that was installed in your mobile phone. How many transistors and what CMOS manufacturing process it is built on (if known). Do you think the CPU in your phone performs better than an old desktop PC computer? For example: Intel Pentium 4/Single Core/Superscalar/3.0GHz/256KB L2/130nm manufactured in 2000

Justify your answers with respect to technology process, pipelines, systems, cache performance, branch predictions, memory hierarchies and so on.

Q(c) As the CMOS technology process shrinks from 130nm to current 16nm. What changes have brought us with respect to:

- Total chip energy consumption
- Total chip power consumption
- Power density (2011 Paper 7, Question 4)
- Maximum clock frequency
- Clock per instruction (CPI)
- Synchronisation via chip-interconnects
- Wire delays (2013 Paper 8, Question 3(d))
- Cache size, cache performance and memory bandwidth
- (Optional) Instruction set architecture evolution

Q(d) (Optional) How might recent advances in die stacking help to improve microprocessor performance and reduce costs? (from 2015 Paper 7, Question 5)
2 Fundamentals of Computer Design

Read the article from the extra material from the lecture 2, “Amdahl’s Law in the Multicore Era”, Mark D. Hill and Michael R. Marty, IEEE Computer, July 2008 and answer the first question.

Q.(a) According to the Amdahl’s Law, the sequential fraction in an sequential application determines the upper limit of the speedup that could be achieved if the application were properly parallelised.

\[
\text{Speedup}(n) = \frac{1}{B + \frac{(1 - B)}{n}} \tag{1}
\]

where \(B\) is the sequential fraction and \(n\) is the number of cores.

(i) If you were given an repository of the source code of an application, how could you determine the sequential fraction \(B\) of the application?

(ii) In order to make the estimate more accurate, the Amdahl’s Law could be amended. Suppose the cost of forking and joining one thread is \(C\). In the program, \(n\) threads are forked and joined \(N\) times. Show your amended equation and specify what happens as the number of threads \(n\) increases.

(iii) (Optional) On heterogeneous multi-core architectures, each core may have different performance characteristics. Therefore, the Amdahl’s Law for heterogeneous platforms also need to be amended. Suppose there are two types of cores, one \(A\) core and four \(B\) cores. Assume the performance of \(B\) is \(x\) and the performance of \(A\) is \(a \times x\) where \(a > 1\). Show your amended equation and specify what in what circumstances that your final speedup could be greater than 5.

Q.(b) As the transistor budgets available to computer architects grow, does the design of the instruction set become more or less important?

Q.(c) Discuss the pros and cons of architectures with a 64-bit word size versus those with a 32-bit size. What applications are likely to benefit most? (from 2005 Paper 7, Question 1). What hardware extensions are needed for the 64-bit CPU to run a 32-bit application to ensure backward compatibility?

Q.(d) (Optional) If you were a processor architect targeting embedded applications where memory is a scarce resource, how might you design a RISC-like instruction set that will achieve efficient use of memory? (from 2005 Paper 7, Question 1)

3 Advanced Pipelining

Q.(a) Throughout the 1990s mainstream microprocessors were developed with ever deeper pipelines. Since then manufacturers have scaled back to more moderate pipeline depths. (from 2014 Paper 8, Question 3(a))

(i) What determines the optimal pipeline length for a microprocessor?

(ii) What were the benefits from implementing deep pipelines and why were they scaled back?

(iii) (Optional) How do pipelines that support in-order and out-of-order execution differ in their microarchitectural components?
4 Branch predictions

Q. (b) Why do modern processors that execute the IA-32 instruction set (commonly known as x86) first translate the instructions into simpler RISC style operations?

Q. (c) How does the existence of multiple parallel pipelines (as illustrated in slide 21.) make it more difficult for us to provide support for precise exceptions?

4 Branch predictions

Q. (a) Why might a branch target buffer provide a poor prediction of procedure return addresses and what hardware solution may be employed to improve the accuracy of such predictions? (from 2011 Paper 8, Question 3(a))

Q. (b) Why are 2-bit saturating counters often used to predict a branchs direction in preference to single bit schemes?

Q. (c) (Optional) What challenges must be overcome in order to achieve high instruction fetch rates for wide-issue superscalar processors?

Q. (d) Modern high-performance processors incorporate a dynamic branch predictor to avoid stalling when branches are fetched. (from 2014 Paper 8, Question 3(b))

   (i) What is a tournament branch predictor and why might it outperform either a global or local branch predictor alone?

   (ii) You develop a new branch predictor that is significantly more accurate than existing designs. However, its complexity means that it takes several cycles to produce a prediction. How can you make use of this predictor without always introducing a pipeline bubble?

   (iii) (Optional) If you were designing an out-of-order core, why might you decide not to allow predicated execution?

Q. (e) (Optional) Branch prediction and speculative execution are often used to expose greater amounts of instruction-level parallelism in superscalar processors. A reorder buffer or unified register file may be used to help recover after mispredicted branches are detected. (from 2011 Paper 8, Question 3(d))

   (i) How are an instructions operands located when a reorder buffer is used?

   (ii) What actions are taken to recover from a mispredicted branch when a unified register file is used?

End of Problems