PRIFYSGOL CYMRU; UNIVERSITY OF WALES

DEGREE EXAMINATIONS JANUARY 2003

SWANSEA

Computer Science

CS 323 High Performance Microprocessors

Attempt 2 questions out of 3

Time allowed: 2 hours

Students are permitted to use the dictionaries provided by the University

Students are NOT permitted to use calculators

CS_323
HIGH PERFORMANCE MICROPROCESSORS

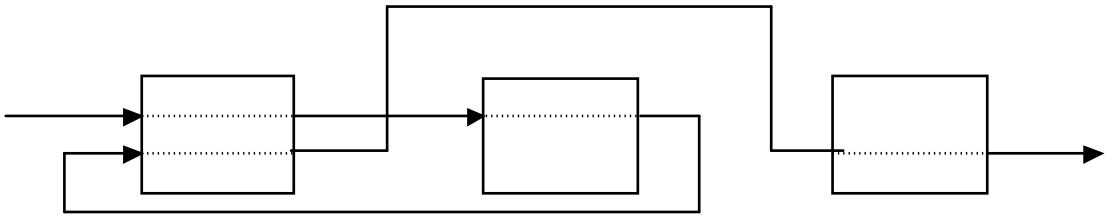
(Attempt 2 questions out of 3)

Question 1

(a) Describe *Read after Write* (RAW) hazards by using a simple example. **[5 marks]**

(b) Describe some *simple* strategies for minimizing the effects of RAW hazards. Also, describe a more advanced strategy that is not yet employed in commercial processors. **[5 marks]**

(c) Consider the following pipeline.



Construct the reservation table, determine the forbidden latencies, and hence derive the collision vector. **[5 marks]**

(d) Describe how to implement a simple ``greedy'' control algorithm for the pipeline in (c) using OR gates and shifters. **[5 marks]**

(e) The algorithm described in (d) is *sometimes* not optimal. Under what circumstances might you still wish to use it if that is the case? **[5 marks]**

Question 2

(a) The P6 is the core implementation used for the Pentium Pro, Pentium II and Pentium III. Describe the P6 pipeline. You should pay particular attention to the mechanism Intel used to overcome issues with the IA32 instruction set that cause problems with pipelined/superscalar implementations. **[10 marks]**

(b) The Pentium IV uses a different implementation called *NetBurst.* Outline how NetBurst differs from P6. Some decisions made for NetBurst are controversial - what are they and why are they controversial **[10 marks]**

(c) Intel have recently introduced a new architecture, called IA-64, which is substantially different to the well-established IA-32, together with the *Itanium* series of processors (Merced, McKinley, Madison), implementing that architecture. Why have Intel introduced IA-64? Describe, in outline, IA-64 and explain how it *radically* differs from IA-32. Why have Intel chosen to make such a radical change? **[5 marks]**

Question 3

(a) Consider a pipelined machine with a reorder buffer, a bank of user registers, a Level 1 data cache, and main memory. Explain the path of the *result* of an instruction *once the execution phase has finished* in the following circumstances.

- An arithmetic operation where the result is to be stored in a register.
- A memory write operation with a write-through cache.
- A memory write operation with a write-back cache.
- An arithmetic operation where the result is to be stored in a register, and used by a closely-following instruction.

[10 marks]

(b) What is a *branch target buffer*, and what information does it minimally contain? What information could you add to speed up instruction execution? What other branch-related information could be usefully stored in a branch target buffer? **[5 marks]**

(c) With simple branch history mechanisms, there is little gain in having more than two history bits. Explain how, and why, extra history bits can be usefully exploited by a *correlating predictor*. You may wish to illustrate your answer with an example of a (1,1) correlating predictor. **[5 marks]**

High Performance Microprocessors 02/03 Marking Scheme

1.(a) Read after Write hazards arise when a pipeline must stall to ensure that a result is available to a subsequent instruction. [2] For example, consider a pipeline with 5 stages: fetch, decode, operand fetch, execute, writeback. If instruction 2 needs the result of instruction 1, then its operand fetch must be delayed until instruction 1's write back has completed. [3].

(b) Simple strategies: get the compiler to try to separate RAW-sensitive instructions [1]; bypass the operand fetch/writeback process by providing a feedback loop from the output of the instruction phase [1]; and extend this idea by providing a generalised forwarding/bypassing mechanism (e.g. a reorder buffer) [1]. A more advanced strategy is to predict the results of arithmetic operations by pattern matching in past values (quite easy for e.g. loop counters) [2].

(c) For reservation table [2] marks. For collision vector (10) [3] marks.

	1	2	3	4
A	*		*	
B		*		
C				*

(d) Store the collision vector in a register that is [conditionaly] bitwise-ORed with the shift register of the same length. The shift register initially contains zero, and is shifted 1-bit right on every cycle. If a zero is shifted out of the register, then it is safe to start a new operation. If a new operation is started, the collision vector is ORed into the shift register (otherwise it is not) [5].

(e) Alternatives are more complex and ``bespoke'' (harder to modify later). If the simple algorithm is not particularly sub-optimal (in a particular case) and/or the pipeline is not expected to be particularly busy, it may be better to use the simple algorithm [3]. In addition, in some circumstances, potentially optimal algorithms involve ``guessing'' when future operations will need to start - if the guesses are wrong, there may be no gains over the simple case [2].

2(a) P6 is a 3-way superscalar, 14-stage pipeline with 5 execution units [2]. For a full discussion of the pipeline and its important stages [3]. The P6 translates variable length/time IA-32 instructions into fixed-length, RISC-like uops internally, using a set of decoders: 2 simple (generating single uop sequences); one more complex (generating 1-4 uop sequences); and one ``worst case'' microprogrammed sequencer (generating up to several hundred uops).

(b) NetBurst has a deeper pipeline (20 stages instead of 14 though two are just *drive* stages that perform no work); more execution units (7 instead of 5), of which the integer units are internally double clocked; a better branch prediction scheme integrated with a trace cache instead of a L1 instruction cache; and a 128-slot register renaming scheme instead of a 40-slot reorder buffer [4]. The decision to go for a deeper pipeline instead of increasing the issue rate (i.e. making it ``more superscalar'') goes against the current trend, which is for shallower pipelines and higher issue rates. There is an argument that Intel's decision was influenced by the need to increase clock rates (easier in deeper pipelines) for marketing reasons. However, tests do show that the decision is technically defensible [6].

(c) IA-64 has been introduced because the limits of 32-bit architectures are being reached (exceeded for high-end users) [1]. IA-64 is a VLIW-derived architecture, with features designed to circumvent the usual VLIW problems (code compatibility/lack of performance gain on upgrade, visibility of hardware organisation etc.) The key features are instruction grouping and bundling. Bundles are groups of three instructions packed into 128-bit words, with a 5-bit template that acts as a partial opcode and indicates the presence of *stops* between instructions. Stops mark the beginning/end of groups, which can span multiple bundles. With a few provisos, all instructions in a group are guaranteed data hazard-free by the compiler and can be executed simultaneously [5]. Other characteristics include: RISC2-like register stacks [1]; and predicated instructions [1]. The choice was made in an attempt to maximize the level of exploitable ILP with modern implementation technicques [2].

3.(a)

- The result would first go to the reorder buffer, where it would wait until all earlier instructions had completed before being moved to its destination register [2].
- The result would first go to the reorder buffer, where it would wait until all earlier instructions had completed before being moved to the L1 cache and [simultaneously] written to memory [2].
- The result would first go to the reorder buffer, where it would wait until all earlier instructions had completed before being moved to the L1 cache. It will be written to memory when the cache block it occupies is discarded [3].
- The result would first go to the reorder buffer. When the following instruction needs the value it would be *copied* to the appropriate execution unit (or reservation station if appropriate). It would wait in the reorder buffer until all earlier instructions had completed before being moved to its destination register [3].

(b) A branch target buffer minimally contains the *pre-computed* target addresses of the last few hundred/thousand branches [1]. By storing a copy of the branch target instruction in the buffer, the cycle lost in fetching the wrong (i.e. non-branch) instruction can be gained bank by replacing it with the instruction in the branch target buffer instead of refetching it [2]. A branch target buffer is the obvious place to store branch history information [2].

(c) Correlating predictors use information about the behaviour of the current and previous branches to make predictions. They work because branches are not typically independent (though some can be an in those cases correlating predictors do not outperform simple ones). This is especially true in conditional statements, where one branch may only [possibly] be taken if a previous one is not (for example) [2]. A (1,1) correlating predictor uses two 1-bit predictors for each branch. The one it uses (and updates) depends on whether the previous branch in the program was taken or not [3].