332 Advanced Computer Architecture

Tutorial exercise 8

Exercise 8.1. IA-64 ("EPIC") instruction encoding

Refer to the the two articles you have been given entitled

- *Introducing the IA-64 Architecture* by Huck, Morris et al (IEEE Micro, Sept/Oct 2000) (henceforth "IIA").
- *Itanium Processor Microarchitecture* by Harsh Sharangpani and Ken Arora (IEEE Micro, Sept/Oct 2000) (henceforth "IPM").

The second page (page 13) of IIA explains how IA-64's 41-bit instructions are packaged as 3-instruction bundles each 128 bits long. The remaining 5 bits are called the "template". The role of the template in instruction issue is explained in further detail in IPM, page 32.

1. The authors claim (IPM pg 32) that a conventional superscalar architecture must perform $O(n^2)$ comparisons between source and destination registers to determine the independence of instructions in the same issue packet (where n is the number of instructions to be issued per packet).
   (a) Itanium can issue a maximum of 6 instructions per cycle. Each IA-64 instruction can refer to three register operands (from its 128 general-purpose registers, and 128 floating-point registers) and also one of its 64 predicate registers.

   How many bit comparisons would be needed in a superscalar design to establish that all 6 instructions are independent?

   (b) Suggest how the complexity of this circuitry could be reduced in a superscalar design. Does this analysis have to happen *every* time the instruction packet is issued?

Exercise 8.2. Branch prediction

For this question, refer to IPM pages 29-31.

According to the Itanium Microarchitecture Reference Manual, a conditional branch can lead to zero, 1, 2, 3 or 9 pipeline bubbles (assuming the branch destination is in the L1 instruction cache).

1. How can the branch delay be zero?
2. Why can't all correctly-predicted branches have a delay of zero?
3. What is a resteer?
4. Why is there special branch prediction provision for procedure returns?

Paul Kelly, Imperial College, November 2000

Notes on the solutions to Exercise 8

Exercise 8.1. IA-64 ("EPIC") instruction encoding

Refer to the the two articles you have been given entitled

- *Introducing the IA-64 Architecture* by Huck, Morris et al (IEEE Micro, Sept/Oct 2000) (henceforth "IIA").
- *Itanium Processor Microarchitecture* by Harsh Sharangpani and Ken Arora (IEEE Micro, Sept/Oct 2000) (henceforth "IPM").

The second page (page 13) of IIA explains how IA-64's 41-bit instructions are packaged as 3-instruction bundles each 128 bits long. The remaining 5 bits are called the "template". The role of the template in instruction issue is explained in further detail in IPM, page 32.

1. The authors claim (IPM pg 32) that a conventional superscalar architecture must perform $O(n^2)$ comparisons between source and destination registers to determine the independence of instructions in the same issue packet (where n is the number of instructions to be issued per packet).

   (a) Itanium can issue a maximum of 6 instructions per cycle. Each IA-64 instruction can refer to three register operands (from its 128 general-purpose registers, and 128 floating-point registers) and also one of its 64 predicate registers.

   How many bit comparisons would be needed in a superscalar design to establish that all 6 instructions are independent? **Answer:** The packet assigns to up to $N_{dst} = 6$ registers, and uses up to $N_{src} = 12$ operand registers (2 per instruction).

   - To detect a RAW or WAR dependence we need to compare all 6 with all 12, giving 72 7-bit comparisons. Except that an instruction can overwrite its own operands without causing a dependence - to $(N_{dst} - 1) \times N_{src} = 5 \times 12 = 60$ 7-bit comparisons.
   - To detect WAW we need to check whether any destination registers re-occur. The last register has to be compared with all 5 preceding ones, the 5th with the 4 preceding... $5 + 4 + 3 + 2 = 14$ 7-bit comparisons $(= N_{dst}(N_{dst} - 1)/2 - 1)$.
   - Then we need to find the first instruction to suffer a dependence, which involves a little additional logic.

   It may be possible to make some savings if register renaming will resolve WAR and WAW hazards.

   We also need to check for dependences due to predicate registers. Any or all of the six instructions could be compares, generating predicate register values (and using operand registers). They could also be predicated - so for RAW/WAR we have up to $(N_{dst} - 1) \times N_{src} = 5 \times 6 = 30$ 6-bit comparisons, plus $N_{dst}(N_{dst} - 1)/2 - 1 = 14$ 6-bit comparisons for WAR.

   Total:

Register class	Dependence	comparisons	bits	
Operand registers	RAW and WAR	60	7	420
	WAW	14	7	98
Predicate registers	RAW and WAR	30	6	180
	WAW	14	6	84
			Total:	782

   (b) Suggest how the complexity of this circuitry could be reduced in a superscalar design. Does this analysis have to happen *every* time the instruction packet is issued? **Answer:**

- Idea 1: use the same trick - get the compiler to add templates
- Idea 2: require instructions which use/update FP registers to be at a particular position in the packet, block multiple issue if not. If there is only one such position, there can be no FP register dependences; if there are several, we have at least reduced the size of the problem.
- Idea 3: do the dependence analysis in the instruction fetch unit. Record the results as a "template" associated with the I-cache (or trace cache). This has the happy result that a branch which hits in the I-cache doesn't have to wait for the destination instruction to be re-analysed.

Exercise 8.2. Branch prediction

For this question, refer to IPM pages 29-31.

According to the Itanium Microarchitecture Reference Manual, a conditional branch can lead to zero, 1, 2, 3 or 9 pipeline bubbles (assuming the branch destination is in the L1 instruction cache).

1. How can the branch delay be zero? **Answer:** The only way is if the branch matches a TAR (Target Address Register), which is accessed in the same cycle as the branch is issued.
    - Why can't all correctly-predicted branches have a delay of zero? Because the BPT and MBPT are too large to access in the same clock cycle as the branch is issued.
    - What is a resteer? it's when a slower branch prediction either improves the accuracy of a faster one, or provides a prediction where none was available from the fast one. It's a way of avoiding the full misprediction penalty - which in turn means that the first predictor can be small and very fast.
    - Why is there special branch prediction provision for procedure returns? Because returns are poorly predicted by a target address cache, even if indexed by branch history

(Paul Kelly, Imperial College, November 2000)