

Midterm Exam

Prof. Loh
CS8803AMA - Fall 2005
27 Oct 2005

Overall Stats:

Overall Grade Histogram

Score:	50's	60's	70's	80's	90's	100's	110's
Frequency:	4	0	1	3	0	1	1

	Long Problems	Multiple Choice	Overall
Average Points	58	18.2	76.2
Std-Dev	16.9	11.6	22.2
Average Difficulty*	3.33	3.20	3.27

**Average difficulty based on student evaluation of questions. There were 3-5 votes per problem.*

Long Problems:

	Average Points	Std-Dev	Average Difficulty
Problem 1(a)	7.4	6.08	4.60
Problem 2(a)	14.2	4.14	2.50
Problem 3(a)	16.2	2.25	2.75
Problem 4(a)	9.0	9.94	3.80
Problem 5(a)	11.2	8.16	3.67

Multiple Choice:

	% That answered correctly					Average Difficulty
	(1)	(2)	(3)	(4)	(5)	
Problem 1(b)	70%	90%	40%	30%	20%	3.00
Problem 2(b)	60%	40%	100%	50%	50%	3.00
Problem 3(b)	100%	70%	70%	60%	50%	3.00
Problem 4(b)	30%	60%	70%	70%	40%	3.33
Problem 5(b)	30%	30%	40%	70%	30%	3.67

1. Instruction Fetch

- (a) Given that 25% of instructions are branches, 75% of branches are taken, **what is the average fetch rate** from an instruction *Trace Cache* where each trace can have at most six instructions and two branches. Example traces include (but not limited to): NBNB (4 instructions, N=non-branch, B=branch, trace terminated by second branch), NNNNNB (6 instructions, trace terminated by size-instruction limit). Assume only complete traces where either there are two branches present or six instructions are present (for example, NNNB will not occur because it has fewer than two branches and less than six instructions). [20 points]

The main part of this problem is figuring out the different valid traces, and then computing the probability of each trace happening. Note that in a trace, the direction of the branch does not matter, and so we do not actually care that the branch taken rate is 75%.

There is no way to have a single instruction in a trace because it neither satisfies the branch count limit nor the instruction limit. There is only one possible way to have a two-instruction trace, which is to have two branches. The probability of this happening is:

$$\frac{1}{4} \times \frac{1}{4}$$

For three instructions, you can have NBB or BNB. BBN is not a valid trace because the second branch would have terminated the trace. So the probability of having a 3-instruction trace is:

$$\left(\frac{3}{4} \times \frac{1}{4} \times \frac{1}{4}\right) + \left(\frac{1}{4} \times \frac{3}{4} \times \frac{1}{4}\right) = 2 \times \frac{3}{4} \times \left(\frac{1}{4}\right)^2$$

The four-instruction traces are: NNBB, NBNB, BNNB, which have a probability of:

$$3 \times \left(\frac{3}{4}\right)^2 \times \left(\frac{1}{4}\right)^2$$

The five-instruction traces are: NNNBB, NBNB, NBNNB, and BNNNB, which gives us:

$$4 \times \left(\frac{3}{4}\right)^3 \times \left(\frac{1}{4}\right)^2$$

The six-instruction traces that contain two branches are: NNNNBB, NNNB, NNBNNB, NBNNNB and BNNNNB, which gives us:

$$5 \times \left(\frac{3}{4}\right)^4 \times \left(\frac{1}{4}\right)^2$$

However, there are also several other valid six-instruction traces that only contain one branch: NNNNNB, NNNNBN, NNNBNN, NNBNNN, NBNNNN and BNNNNN, which together have a probability of:

$$6 \times \left(\frac{3}{4}\right)^5 \times \frac{1}{4}$$

And there is a single valid trace with no branches NNNNNN with probability:

$$\left(\frac{3}{4}\right)^6$$

The overall expected fetch rate is equal to the sum of the size of traces times the probability of getting a trace of that length:

$$2 \times \left(\frac{1}{4}\right)^2 + 3 \times \left(2 \times \frac{3}{4} \times \left(\frac{1}{4}\right)^2\right) + 4 \times \left(3 \times \left(\frac{3}{4}\right)^2 \times \left(\frac{1}{4}\right)^2\right) +$$
$$5 \times \left(4 \times \left(\frac{3}{4}\right)^3 \times \left(\frac{1}{4}\right)^2\right) + 6 \times \left(5 \times \left(\frac{3}{4}\right)^4 \times \left(\frac{1}{4}\right)^2 + 6 \times \left(\frac{3}{4}\right)^5 \times \frac{1}{4} + \left(\frac{3}{4}\right)^6\right)$$

If you do all of the arithmetic, this adds up to 5.15 instructions per fetch.

2 points were awarded if you did not use the taken branch rate. 2 points were given to properly handling of traces of length 2, 3, 4, 5, 6 (with two branches), 6 (with one branch) and 6 (with no branches). 4 more points were given for the final computation.

(b) Multiple Choice [2 points each, total 10 points]

1. Choose the correct statement:
 - A. A 1024-instruction trace cache always provides a higher fetch bandwidth than a 1024-instruction I\$.
 - B. A fully-associative I\$ always provides a higher fetch bandwidth than a direct-mapped I\$.
 - C. **A 1024-instruction trace cache is physically larger than a 1024-instruction I\$.**

2. Fetch fragmentation can be reduced by:
 - A. **Increasing the I\$ line size.**
 - B. Increasing the set-associativity of the I\$.
 - C. All of the above.

3. Compiler branch prediction hints:
 - A. Are not useful because different input sets may result in different branch directions.
 - B. Are not useful because the hint is not known until the decode stage.
 - C. **None of the above.**

4. The branch target buffer:
 - A. Always provides the correct target for PC-relative branches.
 - B. Must have as many entries as the branch direction prediction tables.
 - C. **Does not have to store the branch instruction address (tag) along with the predicted target.**

5. Speculative branch history update:
 - A. May cause a saturating 2-bit counter to be trained in the wrong direction.
 - B. **Is not required for correct program execution.**
 - C. Requires recovering the branch history register after a misprediction/pipeline flush.

2. Register Renaming and Decode

- (a) Consider a 4-wide superscalar processor's rename logic. Using the four example instructions A-D shown below, demonstrate how all the superscalar register renamer performs register renaming on all four instructions *in parallel*. Fill in each blank corresponding to the steps during the rename (RAT lookup and intra-group dependency analysis) as well as the blanks corresponding to the state after the rename logic has processed instructions A-D. For the intra-group dependency analysis, place an X in any blank that corresponds to a register that cannot be directly renamed using only intra-group dependencies. [20 points]

Instructions to be Renamed

A: $R1 = R1 + R3$
 B: $R2 = R1 + R4$
 C: $R1 = R3 - R4$
 D: $R1 = R1 + R2$

Free List

Allocate this → T4
 one first T5
 T13
 T11
 T16

Fill in all _____'s

Renamed sources using the RAT only		RAT State after renaming A-D	
	src_L	src_R	RAT
A:	<u>T19</u>	<u>T8</u>	R1: <u>T11</u>
B:	<u>T19</u>	<u>T17</u>	R2: <u>T5</u>
C:	<u>T8</u>	<u>T17</u>	R3: <u>T8</u>
D:	<u>T19</u>	<u>T21</u>	R4: <u>T17</u>

Renamed sources using intra-group Dependencies only		Final Renamed Instructions			
Dest	src_L	src_R	Dest	src_L	src_R
<u>T4</u>	<u>X</u>	<u>X</u>	<u>T4</u>	<u>T19</u>	<u>T8</u>
<u>T5</u>	<u>T4</u>	<u>X</u>	<u>T5</u>	<u>T4</u>	<u>T17</u>
<u>T13</u>	<u>X</u>	<u>X</u>	<u>T13</u>	<u>T8</u>	<u>T17</u>
<u>T11</u>	<u>T13</u>	<u>T5</u>	<u>T11</u>	<u>T13</u>	<u>T5</u>

$\frac{1}{2}$ point was given for each correct "space" filled in, with a 2-point bonus for complete correctness.

(b) Multiple Choice [2 points each, total 10 points]

1. Register renaming:
 - A. **Only removes some data dependencies.**
 - B. Removes all register data dependencies.
 - C. Removes return-address stack control dependencies.

2. The speculative RAT for a unified physical register file:
 - A. **Has the same number of entries as a RAT for a split ARF/PRF organization.**
 - B. Requires RAT checkpointing to recover from pipeline flushes.
 - C. Always points to the most recently committed version of each architected register.

3. Out-of-order, superscalar processors are impossible to implement for:
 - A. A CISC instruction set architecture.
 - B. A RISC instruction set architecture.
 - C. **None of the above.**

4. A pre-decoded instruction cache:
 - A. Completely eliminates the need for instruction decoding logic.
 - B. **Reduces the branch misprediction penalty.**
 - C. None of the above.

5. Variable-length instruction set architectures:
 - A. Require translation to micro-ops.
 - B. Require pre-decode bits to mark instruction boundaries.
 - C. **None of the above.**

3. Instruction Scheduling

- (a) Consider a two-cluster microarchitecture where the instruction scheduler has been partitioned into two halves. Dependent instructions within the same cluster may schedule in back-to-back cycles; however communication between clusters requires an extra cycle of delay. Compute the IPC of the following instructions assuming the cluster assignments as indicated. [10 points]

Instruction	Cluster	Cluster 0	Inter-Cluster Bypass Delay	Cluster 1
A: R1 = R0 + R10	1			
B: R2 = R9 + R11	0	B		A
C: R3 = R2 + R12	1		(B→), (A←)	
D: R4 = R1 + R13	0	D,E		C
E: R5 = R2 + R1	0		(D→), (C←)	
F: R6 = R2 + R4	1			F,G
G: R7 = R4 + R14	1		(G←)	
H: R8 = R7 + R3	0	H		

IPC: $\frac{8}{7}$

Getting the critical path correct was 5 points, doing the rest correctly was 5 points.

- (b) Change the cluster assignments such that half of the instructions are assigned to Cluster-0 and half are assigned to Cluster-1, and the IPC is equal to that achievable if the cluster-to-cluster communication penalty was zero cycles. [10 points]

Cluster 0	Inter-Cluster Bypass Delay	Cluster 1
A		B
D	(A→)	C
G	(D→), (C←)	E
H		F

IPC: 2

Getting the graph height correct was 5 points, getting the $\frac{1}{2}/\frac{1}{2}$ split was 5 points, but you could lose 5 points if the analysis was done incorrectly.

(c) Multiple Choice [2 points each, total 10 points]

1. An out-of-order processor:
 - A. Must be able to schedule multiple instructions per cycle.
 - B. Requires branch prediction.
 - C. **None of the above.**

2. A data-capture scheduler:
 - A. **Reads ready operands from the register file(s) when the instructions are dispatched to the scheduler.**
 - B. Must use separate architected and physical register files (non-unified RF).
 - C. All of the above.

3. The select logic:
 - A. Provides optimal performance when instructions are scheduled oldest first.
 - B. **Cannot guarantee optimal performance.**
 - C. Can only select a single instruction to issue.

4. The payload RAM contains (among other things):
 - A. **Instruction opcodes.**
 - B. Operand ready bits.
 - C. The select logic.

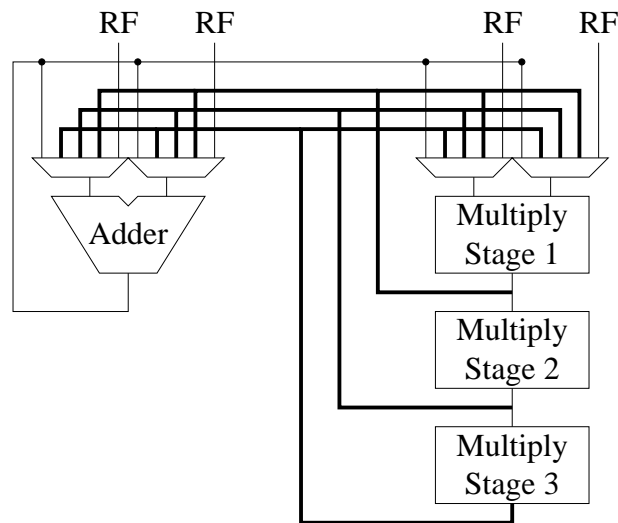
5. Scheduler replays:
 - A. Are required for pipelined scheduling.
 - B. Do not affect performance, but greatly increase power consumption due to (possibly multiple) reschedulings of each instruction.
 - C. **Enable early deallocation of scheduler entries.**

4. Bypass Networks

- (a) Consider the task of multiplying two integers. If the upper bits of one of the operands is all zeros, then the operation can be finished early. Suppose we have a three-stage pipelined integer multiplier as shown below, and the outputs from each stage are MUXed together to support this “early out” result generation. What’s wrong with the bypass network shown below? [10 points]

If a multiply instruction requiring 3 full cycles (all three stages) is followed by a multiply instruction that only requires two cycles (early out after the second stage), then the outputs from the two different stages will collide since the bypass network can only take one output from the multiplier pipeline per cycle. Even worse, these two multiples could be followed by a third multiply that only required a single cycle, which would then make three outputs collide.

- (b) Draw a new bypass network that fixes the problem. [10 points]
The large gaps in the bypass MUXes should have been an obvious hint.



You either got this correct or you didn't.

(c) Multiple Choice [2 points each, total 10 points]

1. Multi-Level Bypass:

- A. Is required to cover the read latency of the physical register file.
- B. **Is required to cover the write latency of the physical register file.**
- C. All of the above.

2. Inter-cluster bypass delays:

- A. **Are not required for a decentralized scheduler.**
- B. **Do not affect ILP.**
- C. Do not affect instructions with multi-cycle latencies.

Option B was supposed to say IPC instead of ILP, so either A or B were accepted as correct answers.

3. ALU Stacks:

- A. **Can reduce bypass wiring requirements.**
- B. Are only useful when there are multiple instances of the same functional unit (e.g., more than one shifter).
- C. All of the above.

4. Mixing integer and floating point bypass networks:

- A. Increases total bypass network area if floating-point values require more bits (ex. 80 bits) than integer values (ex. 64 bits).
- B. Increases the number of inputs per bypass MUX.
- C. **All of the above.**

5. CAM-based bypass control:

- A. Only works with a data-capture scheduler.
- B. Is incompatible with a data-capture scheduler.
- C. **Does not use a centralized control circuit.**

5. Cache/Memory

- (a) Consider the following loop which contains three memory accesses:

```
for(i=0; i<N; i++)
{
    x = a[i];    /* load a[i] */
    *y += x;    /* load y and then store y */
}
```

Assume that the first load ($x=a[i]$) always misses in the L1 and L2 caches, and that the following load and store always hit in the L1 cache. For an L1 cache latency of 3 cycles, and L2 latency of 7 cycles, and a main memory latency of 40 cycles (the processor accesses each in succession, so the latencies are cumulative), what is the average memory latency of all three memory accesses as the number of iterations $N \rightarrow \infty$? [5 points]

-1 point if you computed the total instead of the average.

The first load misses in the L1 (3 cycles), the L2 (7 cycles), and then must wait for main memory (40 cycles), for a total latency of 50 cycles. The following two accesses take 3 cycles each. The average latency for the three instructions is then $(50+3+3)/3 = 56/3 = 18.67$ cycles.

- (b) Assume that for every miss that accesses main memory, the next line is also prefetched in the L2 cache. What is the average memory latency of all three memory accesses as the number of iterations $N \rightarrow \infty$? [15 points]

For iteration k , the first load will miss and so the average latency is the same as in the previous question. In iteration $k+1$, the first load will hit in the L2 cache due to the prefetch from the previous iteration. The average latency of the three memory instructions for iteration $k+1$ is therefore $(10+3+3)/3 = 16/3 = 5.33$ cycles. Since we will alternate between iterations where the first load has to go to main memory and iterations where we'll hit in the L2 cache, the overall average latency is $(50+3+3+10+3+3)/6 = 12$ cycles.

(c) Multiple Choice [2 points each, total 10 points]

1. The load-store queue:
 - A. Is not needed if loads and stores are not permitted to execute out-of-order with respect to each other.
 - B. Forces *all* memory instructions to execute in-order if loads always wait for all earlier store addresses to have been computed.
 - C. **None of the above.**

2. Splitting a store instruction into separate “store address” and “store data” components:
 - A. **Enables earlier detection of ordering violations.**
 - B. Is necessary for speculative load scheduling in the presence of earlier unknown store addresses.
 - C. All of the above.

3. A victim cache:
 - A. Only makes sense for the data cache, but should not be used for the instruction cache.
 - B. **Can decrease *both* conflict and capacity misses.**
 - C. Must be fully associative.

4. Memory-Level Parallelism (MLP):
 - A. Is a property of a program/application (like ILP).
 - B. **Is a property of a program and a microarchitecture.**
 - C. Is not defined for a multi-threaded (CMP or SMT) processor.

5. Load value prediction:
 - A. **Can allow the processor to achieve an IPC rate greater than the program’s ILP.**
 - B. Can improve processor performance, but the IPC rate is still bounded by the program’s ILP.
 - C. Has nothing to do with IPC, ILP nor MLP.