Name Solution

Computer Architecture
EE 4720
Final Examination

9 December 2008, 17:30–19:30 CST

Problem 1		(10 pts)
Problem 2		(15 pts)
Problem 3		(20 pts)
Problem 4		(15 pts)
Problem 5		(20 pts)
Problem 6		(20 pts)
Exam Total		(100 pts)

Alias ISA were, was I?

Good Luck!

Problem 1: (10 pts) Consider a new floating point instruction `nin.d` which uses a 5-stage computation unit, N1 - N5 (in contrast to FP add's A1-A4). The format and register usage for `nin.d` is the same as the other FP arithmetic instructions. Modify the implementation below so that it can execute `nin.d`. *Hint: This is an easy question,* `nin.d` *is just like the FP add and multiply, except it's 5 stages.*

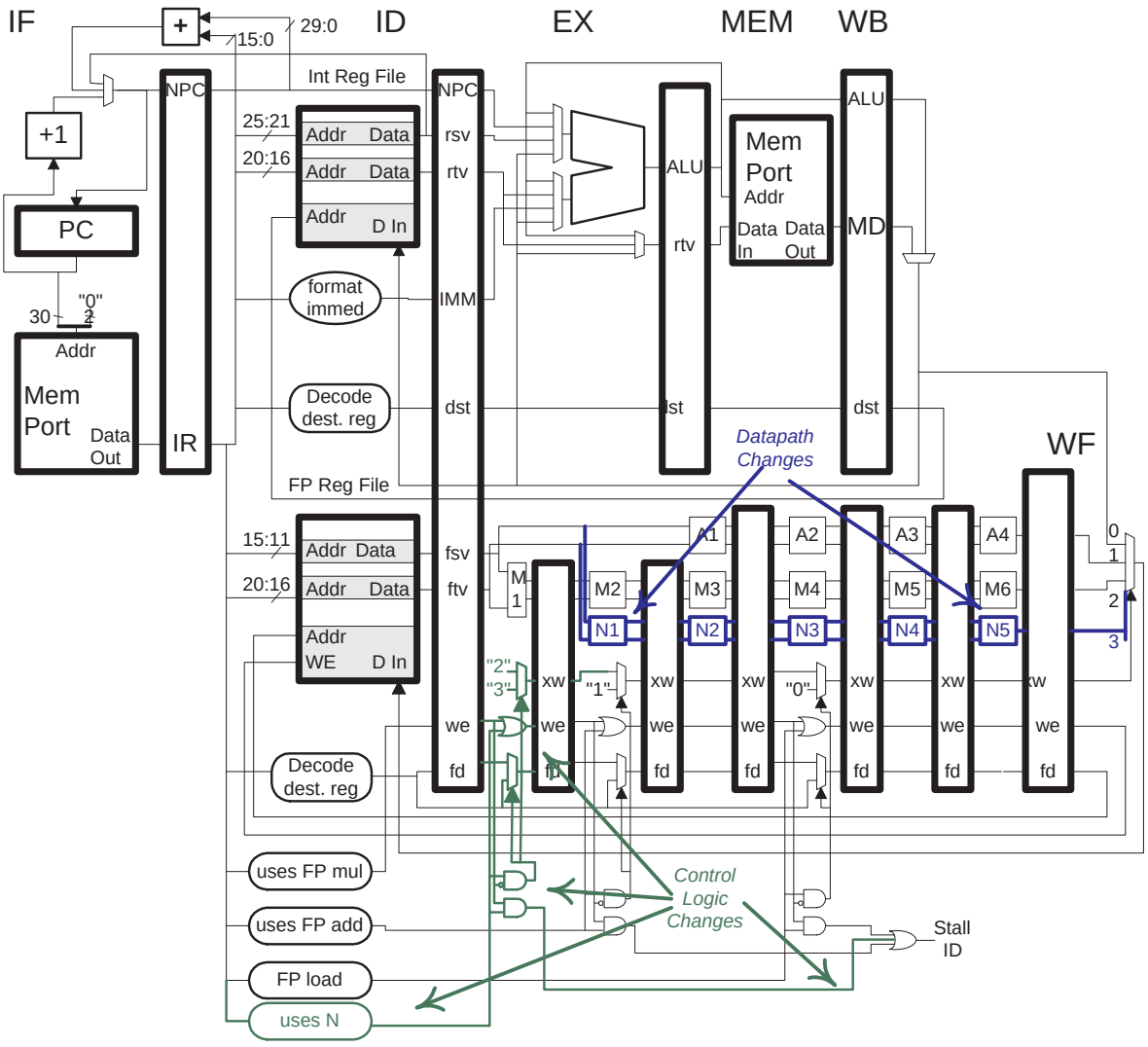
☑ Add the datapath components, including the functional unit stages (N1 to N5).

Solution appears below in blue.

☑ Add control logic for proper write-float and structural hazard detection (as is already present for add and multiply). Be sure not to break existing instructions.

Solution appears below in green.

☑ The changes must fit in efficiently with what is already present.



Problem 2: (15 pts) With the MIPS implementation illustrated below floating-point add and multiply instructions do not raise precise exceptions. If a programmer needs a precise exception for a particular FP instruction he or she can follow it with a `test` (test for exception) instruction. In the code below `test` is used to provide a precise exception for `mul.d`.

```
mul.d  f2, f2, f6
test
sub.d  f16, f14, f20
```

The `test` instruction will stall in ID for **the minimum number of cycles necessary** to ensure a precise exception and will suppress a WF if necessary.

(*a*) Add hardware to implement the `test` instruction. The output of is test is `1` if a `test` instruction is in `ID`. Assume there are A4 and M6 outputs that indicate whether an exception was raised. These can be checked in the middle of the cycle.

☑ Add control logic to stall the pipeline using a new input to the *Stall ID* or gate. *Hint: Very little logic is needed.*

Changes shown in blue on the next page. When a `test` instruction is in ID it will stall the pipeline until there is no FP instruction more than three cycles from `WF`. That is achieved by simply ORing the `we` bits from stages that are more than three cycles from `WF`. If an instruction three or less cycles from WF raises an exception there will be enough time to squash any instruction after the test. (See part b.)

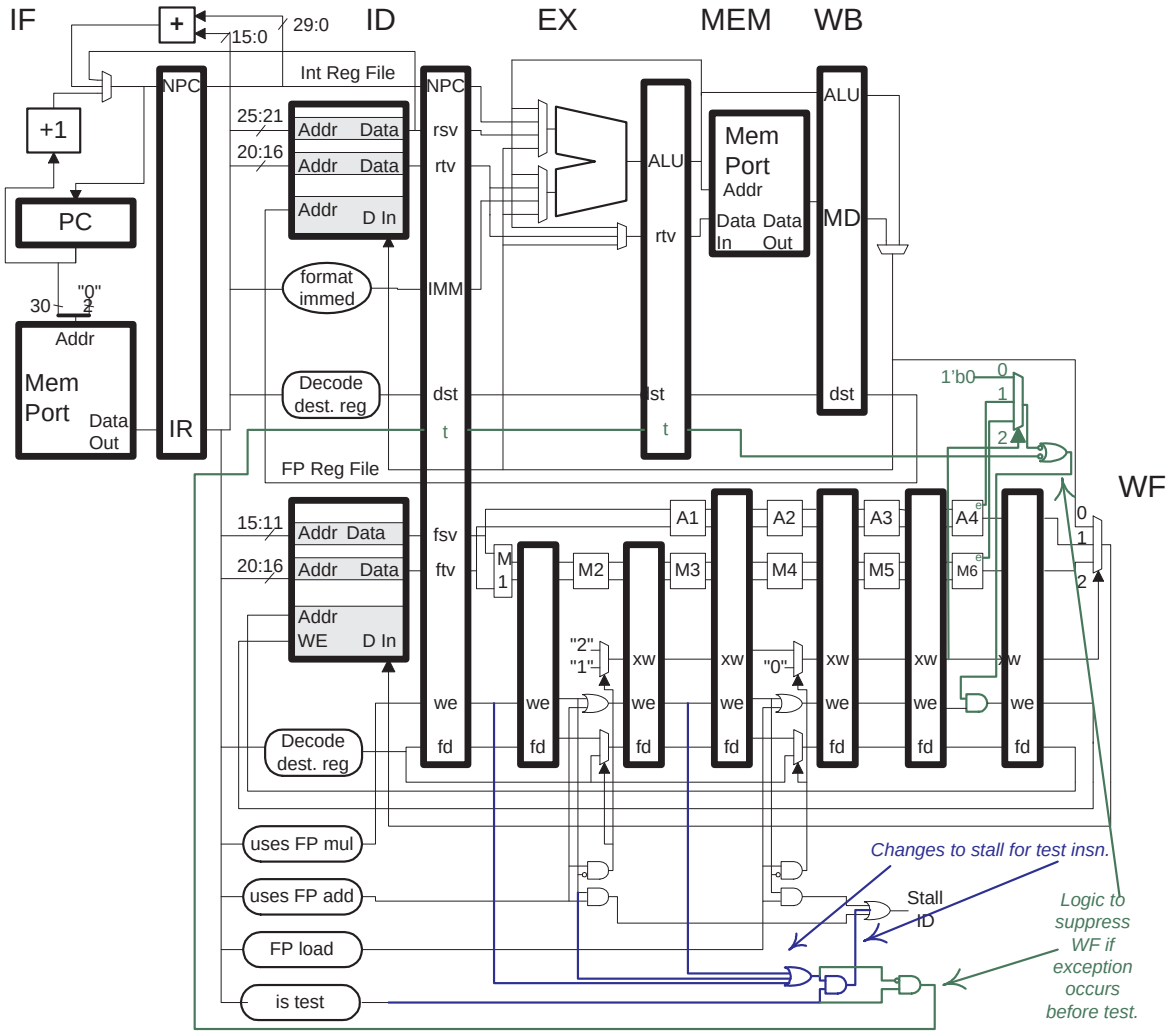
☑ Add logic to suppress WF when necessary.

Changes shown in green on the next page. The WF needs to be suppressed for a FP instruction that raises an exception and is followed by a `test` instruction.

New "e" outputs were added to A4 and M6, these are logic 1 if there is an exception. The added multiplexer will select the e value for the unit in use (or a 0). The instruction in A4/M6 will not be allowed to write back (its `we` bit will not be be propagated) if it raises an exception and it is right before a test instruction. Note that an excepting instruction that doesn't precede a test instruction is allowed to write back, it would write back a NaN (not a number) or some other special value.

A much simpler solution would just AND the we with the negated e signal. That would suppress writeback for all instructions that raised an exception, which is not the desired behavior.

☑ The hardware should work correctly for floating-point multiplies and adds.


IF
ID
EX
MEM
WB
WF
+
+1
PC
NPC
IR
Mem Port
Addr
Data Out
30
"0"
2
15:0
29:0
Int Reg File
25:21
20:16
Addr Data
Addr D In
format immed
Decode dest. reg
FP Reg File
15:11
Addr WE D In
rsv
rtv
IMM
dst
t
fsv
ftv
we
fd
ALU
Mem Port
Data In
Data Out
MD
1'b0
A1
A2
A3
A4
M1
M2
M3
M4
M5
M6
"2"
"1"
"0"
xw
uses FP mul
uses FP add
FP load
is test
Changes to stall for test insn.
Stall ID
Logic to suppress WF if exception occurs before test.

Problem 2, continued:

(*b*) Show an example in which, in a faulty implementation, the test instruction stalls one less than the minimum number of cycles and as a result an exception is not precise.

☑ Example code and pipeline execution diagram, including fetch of first handler instruction.

```
# SOLUTION
#
# Cycle           0  1  2  3  4  5  6  7  8  9
mul.d  f0,f2,f4   IF ID M1 M2 M3 M4 M5*M6*x
test                 IF ID ----> EX ME WBx
sw r6, 0(r7)            IF ----> ID EX MEx
add                              IF ID EXx
or                                  IF IDx
..
HANDLER:
xor                                    IF ID ..
```

☑ Explain why the exception is not precise.

See the pipeline diagram above. The `mul.d` raises an exception in cycle 7, that's too late to stop the `sw` from writing memory. (The memory write started in the beginning of the cycle but the exception was detected later, in the middle of the cycle.)

As a result the handler, which starts fetch in cycle 7, will "see" the change the `sw` made to memory and that violates a requirement for a precise exception.

If the `test` instruction had stalled one more cycle the `sw` could be squashed in time and the exception would have been precise.

Problem 3: (20 pts) The code below has two branches, branch `B1` implements a 3-iteration loop with outcome pattern `T T N T T N`$\cdots$. The outcome pattern for branch `B2` is shown below. **Note that `B1` executes three times for each execution of `B2`.**

The code runs on three systems, the branch predictor on all systems uses a 2^{14}-entry BHT. One system has a bimodal predictor, one system uses a local history predictor with a 10-outcome local history, and one system uses a global predictor with a 10-outcome global history.

```
BIGLOOP:    Note: B1 and B2 are the only branches in this code.
T1:
B1:  bne r3, r4 T1   .. 3 iteration loop ..

# .. no CTIs ..
B2:  beq r1, r2 T2   N N N T N N T N N N T N N T N N N T N N T N N N T N N T ...
T2:

j BIGLOOP
```

☑ What is the accuracy of the bimodal predictor on branch `B2`?

The prediction accuracy is $\frac{5}{7}$.

☑ What is the accuracy of the local predictor on branch `B2`?

The period-seven pattern can easily be captured in the local predictor's ten-outcome history, and so the prediction accuracy is 100%.

☑ What is the size of the BHT and PHT, in bits, needed to implement the local predictor? Only take into account the storage need for the branch predictor, omit things like CTI type.

The BHT size is $2^{14} \times 10$ bits (each BHT entry holds a local history, which is 10 bits). The PHT size is $2^{10} \times 2$ bits (there is one entry for each possible outcome pattern, each entry is two bits).

☑ What is the minimum local history size needed to achieve 100% accuracy on branch `B2` using the local predictor? Explain.

Six outcomes.

A five-outcome pattern is too short, for example, `NNTNN` can be followed by an `N` or a `T`, but a six-outcome pattern is unambiguous. See the table below in which all seven shifts of the pattern `NNNTNNT` are shown. None of the patterns match in the first six bits. (That's a sufficient, but not a necessary condition for 100% accuracy.)

```
123456 7
NNNTNN TN
NNTNNT NN
NTNNTN NN
TNNTNN NN
NNTNNN NT
NTNNNN TN
TNNNNT NN
NNNNTN NT
```

☑ What is the minimum global history size needed to achieve 100% accuracy on branch `B2` using the global predictor? Explain.

Based on the answer above, the global predictor needs to "see" the last six outcomes of branch B2. However, after each outcome of B2 is a set of 3 outcomes of B1, and so the global history size must be at least $\boxed{4 \times 6 = 24 \text{ outcomes}}$ long to hold six outcomes of B2.

Problem 4: (15 pts) Consider two alternatives for improving the performance of our familiar scalar 5-stage implementation: an n-way superscalar implementation or a $5n$-stage superpipelined implementation.

(a) Both systems can *potentially* reduce execution time by a factor of n. Explain how this is achieved for each system in terms of CPI (cycles per instructions), clock frequency, and other relevant factors. The answers might read, "Because of ... the CPI is x^2 times larger which causes ... but ... **and so overall execution is n times faster**."

☑ Explain how the superscalar system is potentially n times faster.

Because each stage has n copied of most hardware, the superscalar system will execute up to n instructions for each instruction executed by the scalar system. The duplicated hardware might have only a small impact on clock frequency (if n is small), so IPC numbers reflect performance.

☑ Explain how the superpipelined system is potentially n times faster.

Because each stage is carefully divided into n parts, the superpipelined system's clock frequency can be n times faster (a bit lower due to overheads). Though the execution rate expressed in IPC won't go up, there are now n times more cycles per unit time.

(b) Bypass paths add substantially to the cost of sufficiently large superscalar systems. Provide expressions in terms of n for the cost of bypass paths in both systems. Briefly justify your answers, using a diagram if necessary.

☑ Expression for the cost of bypass paths in superscalar systems, with quick diagram and brief description.

The cost of bypass paths will be measured in multiplexor inputs needed. The EX stage has n ALUs, each with 2 inputs, for a total of $2n$ bypass path destinations. The ME and WB stages together hold $2n$ destination values, so the number of bypass multiplexor inputs needed is $4n^2$. One component of $\boxed{\text{cost is thus } 4n^2}$.

☑ Expression for the cost of bypass paths in superpipelined systems, with quick diagram and brief description.

The ALU gets its input values in the EX0 stage (the first part of the EX stage which was divided into n pieces), so the number of bypass path destinations is 2. Each of the stages from ME0 to WBn-1 can hold one bypassable value, for a total of $2n$ values. The total number of multiplexor inputs is thus $\boxed{4n}$.

Note that the superpipelined system has fewer mux inputs because it is doing with time what the superscalar system does with space.

(c) Ideally the $5n$-stage superpipelined system would have a speedup of n (the scalar system would take n times longer than the superpipelined system to run a program). Consider a program that has no nearby dependencies so that the superpipelined system does not have to stall.

☑ Explain why the speedup of the superpipelined system might be $\frac{t_0+t_1}{t_0+\frac{t_1}{n}}$, where the clock frequency of our familiar scalar system is $\frac{1}{t_0+t_1}$.

The clock period is shown as a sum of two components, t_0 and t_1. This sum appears in the numerator of the speedup expression, the denominator has the clock period of the superpipelined system. If stages were divide without overhead the superpipelined clock period would be $\frac{1}{n}(t_0+t_1)$ but instead it is $t_0+\frac{t_1}{n}$, indicating that one component of the critical path is not being divided. A likely candidate for that would be the setup time needed for the latches, something which is not split.

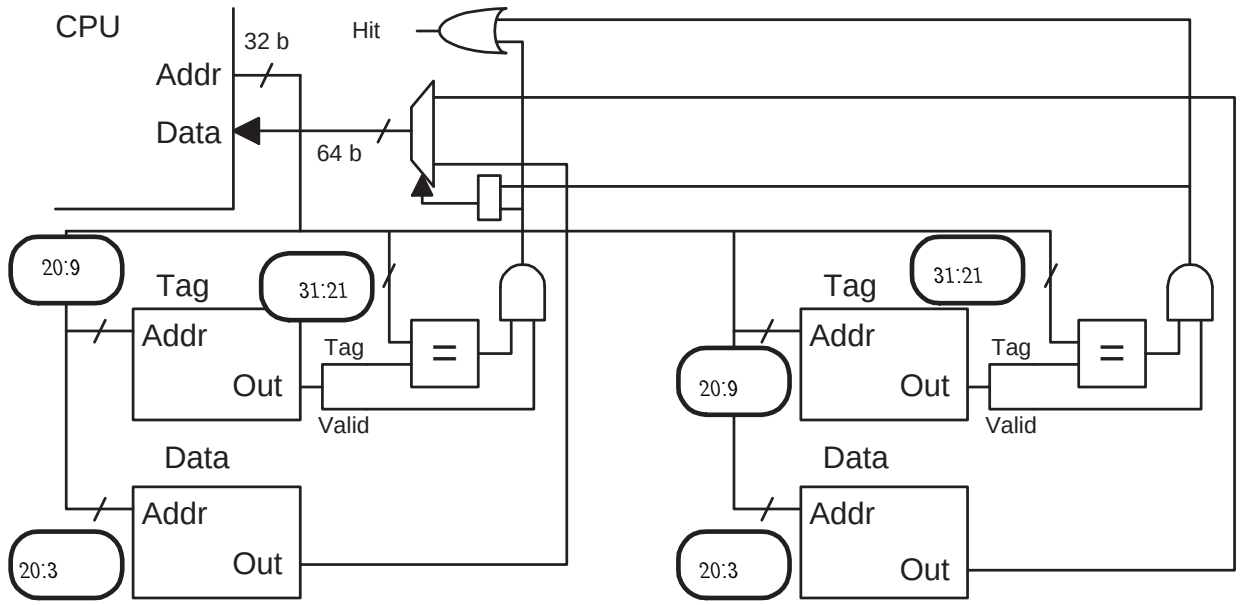
☑ Explain what t_0 is likely to be.

See solution above.

Problem 5: (20 pts) The diagram below is for a 4-MiB (2^{22}-character) set-associative cache with a line size of 512 (2^9) characters, with the usual 8-bit characters.

(*a*) Answer the following, formulæ are fine as long as they consist of grade-time constants.

☑ Fill in the blanks in the diagram.



☑ Show the address bit categorization. Label the sections appropriately. (Alignment, Index, Offset, Tag.)

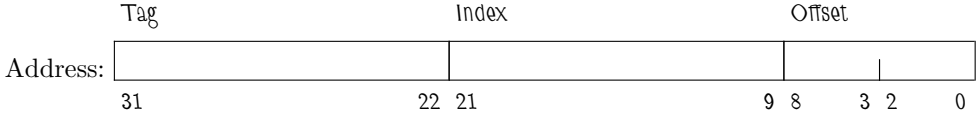
☑ Associativity:

Based on the number of ways in the diagram (there is no ellipses), the associativity is 2.

☑ Memory Needed to Implement (Indicate Unit!!):

It's the cache capacity, 2^{22} characters, plus $2 \times 2^{21-9}\,(32 - 21 + 1)$ bits.

☑ Show the bit categorization for a direct mapped cache with the same capacity and line size.



Problem 5, continued:

(*b*) The code below runs on the cache from the first part of this problem. Initially the cache is empty; consider only accesses to the array.

☑ What is the hit ratio running the code below? Explain

```
double sum = 0.0;
short *a = 0x2000000;    // sizeof(short) = 2 characters.
int i;
int ILIMIT = 1 << 10;    // = 2^10

for (i=0; i<ILIMIT; i++) sum += a[ i ];
```

The line size of 2^9 characters is given, the size of an array element is $2^1 = 2$ characters, and so there are 2^8 elements per line. The first access, at `i=0`, will miss but bring in a line with 2^8 elements, the next $2^8 - 1$ accesses are to subsequent elements on the line and so will hit, so the hit ratio to the first line is $\frac{2^8-1}{2^8}$.

(*c*) The code below runs on a direct mapped (not set-associative) cache with a line size of 512 characters and a capacity of 4 MiB. *Grading note: The cache size was omitted from the original problem.*

☑ Determine an address for `b` that will result in a 0% hit ratio when running the code below.

☑ Briefly explain.

If the index bits of `a[i]` and `b[i]` are identical then `b[i]` will evict the line holding `a[i+1]` (except when `i` is 255, 511, etc.). The index bits are positions 22 to 9, so choose a `b` address in which those bits match `a`, the smallest possibility is `0x400000`.

```
double sum = 0.0;
short *a = 0x2000000;    // sizeof(short) = 2 characters.

short *b =  0x400000;                              // <--- SOLUTION

int i;
int ILIMIT = 1 << 10;    // = 2^10

for (i=0; i<ILIMIT; i++) sum += a[ i ] + b[ i ];
```

Problem 6: (20 pts) Answer each question below.

(*a*) Describe restrictions (or lack of) on instruction addresses and the placement of instructions that distinguish RISC, CISC, and VLIW ISAs.

☑ RISC address and placement restrictions.

☑ Reason for each restriction (if any).

Instruction addresses must be a multiple of four, beyond that there are no placement restrictions. The address restriction simplifies instruction fetch since less shifting (or none at all) is needed to correctly position instructions. Also, the alignment restriction quadruples the reach of CTIs that hold less than a full address.

☑ CISC address and placement restrictions.

☑ Reason for each restriction (if any).

There are no restrictions.

☑ VLIW address and placement restrictions.

☑ Reason for each restriction (if any).

Instruction bundle addresses must be aligned. There might be placement restrictions within a bundle, for example, one might not be allowed to put a load instruction in the first slot.

The reason for the alignment restriction for VLIW is the same as RISC. Restrictions within a bundle can reduce the number of paths to functional units. For example, if the first slot cannot hold a load there is no need to have a connection from the slot-0 position in the memory stage to the memory port.

(*b*) A feature of the SPECcpu benchmarks is that they come with **source code** and it is the tester's responsibility to **compile** (and run) them. Note: "are these" in the questions below refers to source code and compiling.

☑ Are these necessary, important, or irrelevant for testing an implementation of a new ISA? (This is not a yes-or-no question.) Explain.

If source code is not provided then instead executable code must be provided. If the ISA is new then there is not likely to be an executable compiled for it (especially if the new ISA is still being developed!). So, source code is necessary.

☑ Are these necessary, important, or irrelevant for testing a new implementation of an existing ISA? Explain.

Source code is important, but not 100% necessary. Most new implementations will substantially improve the performance of code compiled for an older implementation, which is a good thing for those upgrading their computers. So lack of source code is not necessary. Source code will help show the full potential, and so it is important.

(*c*) A corrupt member of the committee choosing benchmarks for the next SPECcpu suite has successfully bribed the other committee members so that the benchmarks that were selected favor the corrupt member's company.

☑ How might this be discovered? Indicate who would discover the problem and what public information draw their suspicion?

Call the company providing the corrupt member company C, and let V be some other participating company that manufactures computers. Experts from V are aware of characteristics of different benchmarks, especially those which their machines handle well or poorly. They will become suspicious when none of the benchmarks that their machines run well are chosen, benchmarks that they knew were nominated for the suite. Perhaps their suspicions will be raised when their SPEC representative arrives at work up in a new luxury car.

☑ Can we expect those involved to be sufficiently motivated to correct the situation? Explain.

Buying decisions are influenced by benchmarks, so those from company V have a strong financial incentive to take action against their own bribed employee and see to it that the benchmarks are re-chosen.