Name Solution ____________________

Computer Architecture
EE 4720
Final Examination

7 May 2008, 10:00–12:00 CDT

Problem 1	______	(10 pts)
Problem 2	______	(30 pts)
Problem 3	______	(20 pts)
Problem 4	______	(15 pts)
Problem 5	______	(15 pts)
Problem 6	______	(10 pts)
Exam Total	______	(100 pts)

Alias ISA were, was I? ____________________

Good Luck!

Problem 1: (10 pts) The tables below show the state of a dynamically scheduled system using method 3 during the execution of a code fragment.

- The code is preceded by many nop instructions.
- Instructions use either a 4-stage FP multiply unit (M1-M4) or a 2-stage FP add unit (A1, A2).
- Assume that there are unlimited RR, WF and execute (A,M) units.

(*a*) Show a program and pipeline execution diagram consistent with these tables.

☑ Show program and all stages used.

☑ On the physical register file show where physical registers are removed from the free list, using a `[`, and where they are put back in the free list, using a `]`.

☑ All destination registers can be determined exactly.

☑ The time for all stages can be determined exactly.

# Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
add.s f20,f0,f1	IF	ID	Q	RR	A1	A2	WF	C							
mul.s f15, f20, f2		IF	ID	Q		RR	M1	M2	M3	M4	WF	C			
add.s f20, f3, f4			IF	ID	Q	RR	A1	A2	WF				C		
add.s f20, f20,f5				IF	ID	Q		RR	A1	A2	WF			C	
# Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

ID Register Map

# Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
f15 9			39												
f20 16		43		82	93										

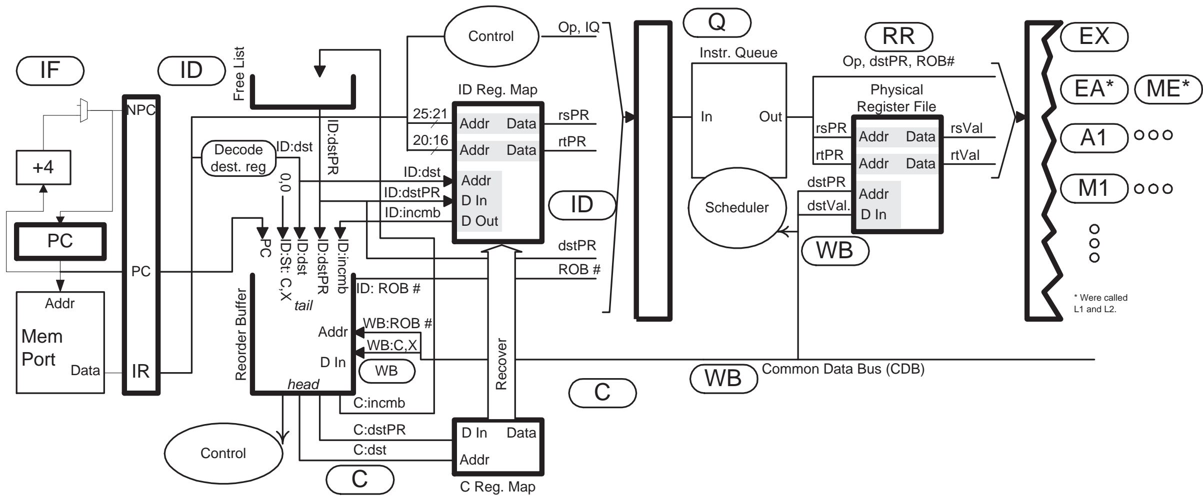
Commit Register Map

# Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
f15 9												39			
f20 16								43					82	93	

Physical Register File

# Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9 11															
16 22															
39											33				
43							44								
82									55						
93											66				
# Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Problem 1, continued: The implementation diagram is provided below for references. Don't forget to answer the question below.



(*b*) In the code fragment on the previous page only two true dependencies are possible.

☑ Show them (by connecting the destination and source with an arrow).

The only true dependencies are through `f20`. That is the second instruction depends on the first, and the fourth instruction depends on the third. One can conclude that the second depends on the first because it waits until the first instruction's result is bypassable, the same for the fourth and third.

☑ Briefly explain why other dependencies are not possible.

The third instruction cannot depend on the second because it starts before the result from the second is available.

Problem 2: (30 pts) Answer the following branch predictor questions.

(*a*) The code below runs on two branch predictors, a bimodal predictor with a 2^{10} entry BHT and a local predictor with a 2^{10} entry BHT and a 9-outcome history.

The outcomes of branch B1 form a repeating pattern as shown below.

The outcome pattern for branch B2 can be constructed by tossing a fair coin, adding a `N N N` for heads or a `T T T` for tails, then repeating. More precisely, outcome $3i$ is a Bernoulli random variable with $p = .5$ and outcomes $3i + 1$ and $3i + 2$ match outcome $3i$, for $i = 0, 1, 2, \ldots$. For example, the following is a possible pattern of outcomes for B2: `N N N T T T T T T`. The following is **not possible** for B2: `N N N T T N T T T`, it's not possible because the number of consecutive N's or T's must be a multiple of 3.

```
LOOP:

 B1:       N  N  N  N  T  T  T  N  N  N  N  T  T  T  N  N  N  N  T  T  T...

 B2:        N  N  N  T  T  T  T  T  T  N  N  N  T  T  T  N  N  N  (See text.)

 j LOOP
 nop
```

For the questions below assume all tables are initially zero. All accuracies are after warmup.

☑ What is the accuracy of the bimodal predictor on branch B1?

Accuracy is $\frac{3}{7}$.

☑ What is the accuracy of the local predictor on B1?

The accuracy is close to 100%.

The pattern of B1 outcomes repeats with a period of 7, which is shorter than the 9-element local history. Thee is possible interference with B2, however in part because B2 is random those interfering patterns will rarely occur frequently enough to affect B1's accuracy.

☑ What is the minimum local history size needed to predict B1 with 100% accuracy ignoring branch B2. Briefly explain.

Four outcomes.

At three outcomes pattern `NNN` can appear before an `N` or `T`.

☑ What is the approximate accuracy of the bimodal predictor on branch B2? Explain.

Accuracy is $\approx \frac{4}{6}$.

B2's outcomes come in groups of 3 and after each group the counter will be saturated at either 0 (for `N`) or 3 (for `T`). If the next group of outcomes is the same there will be 3 correct predictions, if the next group differs there will be 1 correct prediction. Since each case is equally likely the accuracy is $\frac{3+1}{6}$.

☑ What is the approximate accuracy of the local predictor on branch B2 ignoring B1? Explain.

Accuracy is $\approx \frac{5}{6}$.

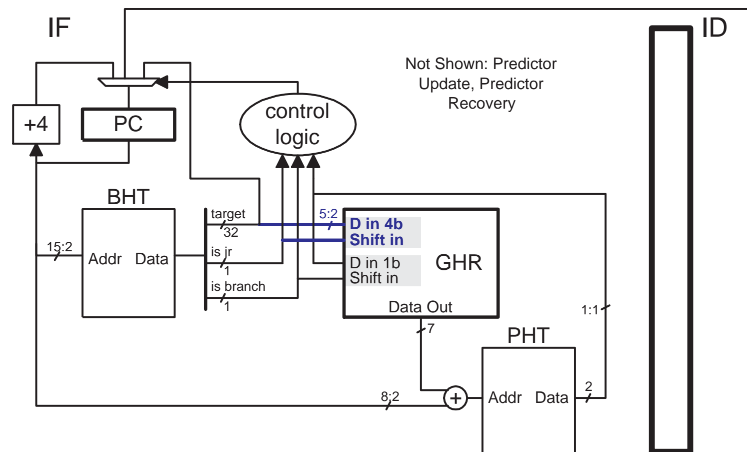
Consider the local history `NNNNNNNNT`, in which the last outcome is taken. Since the next outcome will always be taken the PHT entry, after warmup, will always provide the correct prediction. Lets call this a position 1 local history since the most recent branch is the first in a sequence of 3. All predictions made with position 1 local histories are predicted correctly after warmup, the same for position 2 local histories. For a position 3 local history, say, `NNNNNNTTT`, the accuracy will be 50% because the next outcome is random. Considering position 1, 2, and 3 histories the accuracy is $\frac{1+1+\frac{1}{2}}{3} = \frac{5}{6}$. Finally, consider local histories `NNNNNNNNN` and `TTTTTTTTT`. For these B2 can be at any position, if at 1 or 2 the outcome will be `N` for the first and `T` for the second, if at 3

the outcome is random. Therefore the PHT entries will saturate to 0 or 1, respectively and offer correct predictions $\frac{2}{3}$ of the time. These patterns occur 2^{-3} to 2^{-4} of the time, and so their impact will be small.

☑ What is the approximate warmup time for the local predictor on B2? *Note: This problem was alot more difficult to get perfectly correct than originally intended.*

A local history can span 4 groups of 3 outcomes (two complete, the oldest and the newest incomplete) or 3 groups of 3 outcomes. For the first case there are 2^4 possible outcome sets, each in two different positions, say TTTTTNNNT and TTTTNNNTT. Thus there are 2^5 local histories for this case. For the 3 groups of 3 case there are 2^3 possible local histories, for a total of $2^5 + 2^3$ local histories. Each must be seen twice to warm up. The tricky part is determining how long this will take.

(*b*) The predictor to the right is from the solution to last semester's final exam (and this semester's Homework 4). It is similar to gshare except in how the GHR is updated. Like global and gshare, when a branch is predicted the predicted outcome is shifted into the GHR, but when a `jr` is predicted four bits of the target are shifted into the GHR.



The code fragment below is the one used to justify the predictor, except that now the code is in a four-iteration loop. Important assembler code is shown in the comments.

```
for ( iter=0; iter<4; iter++ ) {
  int c = getchar(); // Unpredictable
  switch (c) {                     // jr $t0  # Jump to correct case.
  case 'a': x = 3; j++; break;     // addi $t1, $0, 3;  j ENDSWITCH;  addi $t2, $t2, 1
  case 'b': x = 7; break;
    ...
  case 'z': x = 1; i++; break;
  }                                // ENDSWITCH:
  if ( x < 5 ) foo(); else bar();
}
```

☑ Why might the prediction accuracy of the `for` loop branch be worse with the predictor above than an ordinary gshare?

The `for` loop branch is the one after the test `iter<4` (the branch itself is not shown). The `for` loop iterates four times, and so can be predicted perfectly with just 3 outcomes of the `for` loop branch in the GHRusing a global or gshare predictor. Even with the `if` branch present the GHR should still be large enough (though barely). However, if four bits of the `jr` target are shifted into the GHR there will not be enough room for three outcomes of the `for` branch and so accuracy will suffer.

☑ Considering the assembler code above, why might it make more sense to shift in the PC of the jump (`j`) instructions rather than the target of the `jr`? *Hint: This would only be useful when the case statements had branches.*

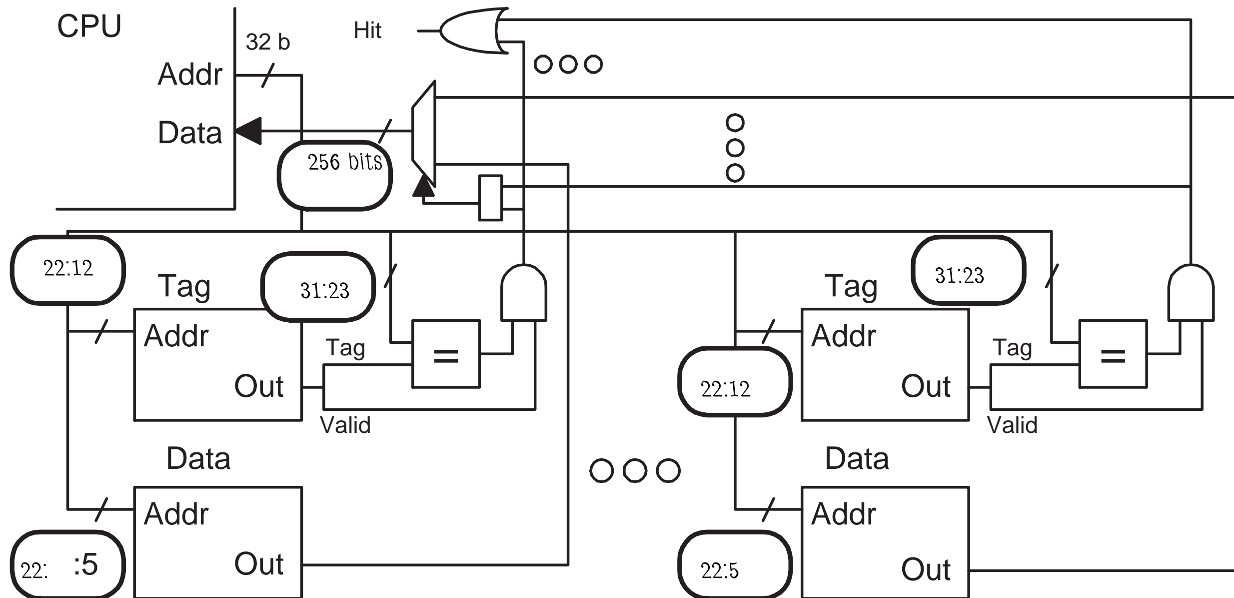
Shifting `jr` bits into the GHR would not help branches in the case statements, and might hurt them. It would not help them because the same bits would be shifted in every time (the address of the start of the case). It would hurt if the `jr` bits shifted in to the GHR pushed out branch outcomes that were needed. For example, suppose a branch in the case statement has the same outcome as the fourth previous branch (encountered before the switch was entered). Shifting in the `jr` bits would push out the outcome of that branch and so accuracy would suffer.

Shifting the PC of the jump instructions would help the prediction of the `x < 5` branch. Note that if the branch is reached via case `a` or `z` it is always not taken (assuming `foo` is on the not-taken path) but if the branch is reached via case `b` it is always taken. Suppose the PC of jump instructions (not the jump target) was shifted into the GHR, then when the `x < 5` branch is reached the value of the GHR would depend on the case statement and so for each case statement a different PHT entry would probably be used. The PHT entries for case `a` and `z` would saturate at zero and the entry for `b` would saturate at 3, resulting in perfect predictions.

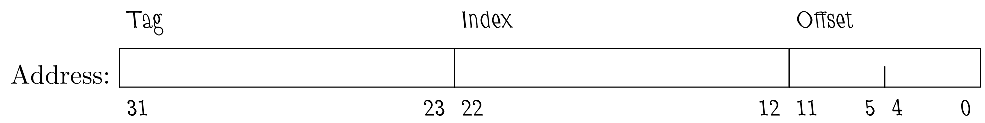
Problem 3: (20 pts) The diagram below is for a 32-MiB (2^{25}-character) 4-way set-associative cache with a line size of 4096 (2^{12}) characters, with the usual 8-bit characters.

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

☑ Fill in the blanks in the diagram.



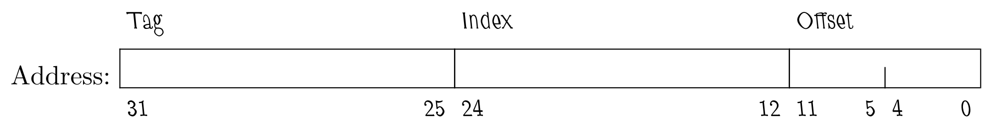
☑ Show the address bit categorization. Label the sections appropriately. (Alignment, Index, Offset, Tag.)



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

It's the cache capacity, 2^{25} characters, plus $4 \times 2^{23-12}\,(32-23+1)$ bits.

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



Problem 3, continued:

(*b*) The code below runs on the same cache as 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;
long *a = 0x2000000;    // sizeof(long) = 8 characters.
int i,j;
int ILIMIT = 1 << 10;    // = 2^10

for (j=0; j<2; j++)
  for (i=0; i<ILIMIT; i++) sum += a[ i ];
```

The line size of 2^{12} characters is given, the size of an array element is $2^3 = 8$ characters, and so there are 2^9 elements per line. The first access, at `i=0`, will miss but bring in a line with 2^9 elements, the next $2^9 - 1$ accesses are to subsequent elements on the line and so will hit, so the hit ratio to the first line is $\frac{2^9-1}{2^9}$, and the same hit ratio will be achieved for all lines accessed with `j=0`. Since `ILIMIT * 8` is much smaller than the cache capacity every access on the second `j` iteration will hit. Therefore the overall hit ratio will be $\frac{1}{2}\frac{2^9-1}{2^9} + \frac{1}{2} = \frac{2^{10}-1}{2^{10}}$.

(*c*) The code below runs on a fully associative cache with 2^7-byte lines, **not the same as the previous cache**. Let h denote the hit ratio of the code below for a cache size of 8 MiB (2^{23} bytes). As always, assume the cache is empty before the code starts.

```
void p4(double *a, double *b) {
  // sizeof double = 8 characters.
  double sum = 0;
  int size = 1 << 8;

  for ( int d=0; d<size; d++ )
    for ( int col=0; col<size; col++ )
      sum +=  b[ col * size + d ]  *  a[ d * size + col ];
}
```

☑ What is the minimum cache size for which the hit ratio is h? Explain.

The `a` array is accessed sequentially and each element is accessed once and since this is a fully-associative cache it only need have two lines to keep `a`'s data safe from eviction. The `b` array is accessed at a stride of `size * 8` characters. Call the line brought in on the first miss to `b`, at `d=0` and `col=0`, `b[0]`. Because `size * sizeof(double)` is larger than the line size the next access to `b`, when `col=1`, will be on a different line. The `b[0]` line will not be accessed again until much later, when `d=1` and `col=0`.

With a large cache the `b[0]` line will remain during its long wait, but with a smaller cache it might be evicted. For the cache to be large enough it must be able to hold the `b[0]` line, plus all lines accessed until the `b[0]` is accessed a second time.

Recall that the line size is 2^7 bytes and `size` is 2^8. Since the access to `a` is sequential the number of lines due to `a` is $\frac{\texttt{size}\times\texttt{sizeof(double)}}{2^7} = \frac{2^8 2^3}{2^7} = 2^4$.

The number of lines due to `b` (including `b[0]`) is `size`, so the total number of lines needed is at least $\texttt{size} + 2^4 = 2^8 + 2^4$ which corresponds to a cache size of $2^7(2^8 + 2^4)$ characters.

Grading Note: No one came close to answering this.

Problem 4: (15 pts) Answer each question below.

(a) Describe what's wrong with each execution below.

☑ What's wrong with this execution on our usual bypassed 5-stage MIPS implementation.

```
# Cycle         0  1  2  3  4  5
lw r1, 0(r2)    IF ID EX ME WB
add r3, r1, r4     IF ID EX ME WB
```

Because the value for `r1` created by `lw` is not available until the end of the `ME` stage, in cycle 3, there is no way to bypass its value when needed, by the `EX` stage for the `add`, at the beginning of cycle 3.

☑ What are the two problems below with this execution on our usual scalar statically scheduled MIPS implementation?

```
add.d f0, f1, f2  IF ID A1 A2 A3 A4 ME WF
```

The register numbers for an `add.d` instruction must be even (it uses pairs of registers for 64-bit operands). Also, the `FP` pipeline does not have an `ME` stage.

☑ What's wrong with this execution on a 2-way superscalar dynamically scheduled machine?

```
# Cycle        0  1  2  3  4  5  6  7  8  9
add r1,r2,r3   IF ID Q  RR EX WB C
add r4,r1,r6   IF ID Q     RR EX WB C
add r7,r8,r9      IF ID Q  RR EX WB    C
add r8,r2,r1      IF ID Q     RR EX WB    C
# Cycle        0  1  2  3  4  5  6  7  8  9
```

Because the processor is 2-way superscalar, the commit in cycle 8 should occur in cycle 7, otherwise execution would be limited to 1 instruction per cycle. Note that the last `add` instruction waits one cycle because there are only two execute pipelines (meaning two `RR`, two `EX`, etc.)

(*b*) Alas, it is unlikely that there will soon be 16-way, statically scheduled, superscalar implementations that anyone would want to buy. Three reasons are started below, complete them.

☑ It would be too expensive because ...

The cost of the bypass paths would be much too high, proportional to 16^2. The output of each of the 16 ALUs would have to bypass to each of the 16×2 ALU inputs.

Note: The answer "because 16 copies of most parts are needed" is **wrong** because one should expect to pay 16 times as much as long as there is the potential to get 16 times the performance.

☑ The clock frequency would be too low because ...

To find dependencies, control logic would have to compare an instruction's source registers to up to 15 other instructions in ID, plus 16 instructions per downstream stage (two in a 5-stage design). That would put a strain on critical path.

Having 16 copies of everything plus the necessary bypass to avoid stalls would require substantially more area than, say, a 4-way processor. The increased physical distance would increase the propagation time of signals and so require a lower clock frequency.

☑ A few programs might realize the full potential of the processor, but most won't because ...

True dependencies within 16 instructions of each other could cause a stall, it would be hard for a compiler to avoid those with scheduling.

Problem 5: (15 pts) Answer each question below.

(*a*) In ARM the PC is a general purpose register, `r15`. It could have been defined in ways consistent with ISA design principles, but it wasn't.

☑ Describe how the use of `r15` appears contrary to the usual goals of an ISA.

The value written when storing `r15` or using it in a calculation depends upon the implementation. The point of separating an ISA from an implementation is to avoid making the program dependent upon details of an implementation (thus loosing portability).

(*b*) The two code fragments below are supposed to do the same thing, the first is MIPS the second is SPARC. In both a branch is taken if a less-than comparison is true. The code would work correctly if the called subroutine returned immediately.

☑ Explain why the SPARC code may not execute as intended.

The SPARC branch tests a condition code. It's possible that an instruction in `some_subroutine` has overwritten the condition codes to something else.

☑ Suggest a fix.

Move the `subcc` after the call (or re-execute it).

```
# MIPS Code
slt $s1, $s2, $s3       # Registers %s0-%s7 preserved.
jal some_subroutine
nop
bneq $s1, $0 SKIP1
...

! SPARC code
subcc %l2, %l3, %l1     ! Registers %l0-%l7 preserved.
call some_subroutine
nop
blt SKIP1
...
```

Hint: The following two questions are really asking about exceptions.

(*c*) A compiler writer is wondering whether dead-code elimination should remove the first assignment to x in the code fragment below (which sure looks dead, right?).

```
x = a / b;
x = a + c;
```

The guy down the hall wrote a program that included these two lines and that program would run differently if dead-code elimination removed the first assignment (the one with the division). The difference has nothing to do with timing or the size of the program.

☑ Why might the program have run differently?

The program might have used a signal handler to be run on a division by zero exception.

☑ The compiler writer wants to DTRT (do the right thing), how does he or she find out what the right thing is?

It's possible to argue that a signal handler can be useful. A stronger argument is that dead-code elimination is even more useful and that most similar needs for a signal handler could be met by tricking the compiler into believing the code isn't dead. (Say, by using it in an if statement.)

However, the compiler must generate code that runs correctly *as defined by the language specification*. The writer then must look at the language specification to see what the correct behavior should be. If the specification says nothing then see what most other compilers do.

Answers which did not mention some kind of a language specification or common behavior were considered wrong.

(*d*) The code fragment below may have come from the code in the previous problem.

```
div.d f0, f2, f4
add.d f0, f2, f6
```

Designers of an implementation are considering whether to avoid WAW hazards like the one above by converting the `div.d` to a nop when the `add.d` reaches `ID`. This will work correctly under ordinary circumstances.

☑ Show a pipeline execution diagram illustrating the WAW hazard.

```
# SOLUTION
# Cycle           0  1  2  3  4  5  6  7  8  9
div.d f0, f2, f4  IF ID DI DI DI DI DI DI DI WF
add.d f0, f2, f6     IF ID A1 A2 A3 A4 WF
```

☑ Under what circumstances will this produce the wrong answer?

If the `add.d` raises an exception. In that case the signal handler won't see the value written by the `div.d` because it was converted to a `nop`.

Problem 6: (10 pts) SPECcpu provides a separate set of training inputs to be used for feedback-directed optimization (FDO) techniques, such as profiling.

☑ Why is a separate training input set needed?

Using the same input set to train and run will produce the best possible results, but it's not something that can be used in real life because input data changes with each run and there is usually no way to determine it in advance.

Profiling using the training set provides a better indication of the benefits of profiling than the inflated numbers obtained by using the same data to profile and run.

☑ Why might an honest and good tester want to substitute a different training set?

There may be newer techniques for choosing good training sets, techniques that anyone can use. The SPEC training sets however used older less-effective training set selection techniques. The tester would like the results to reflect the benefits of his new and perfectly legitimate training selection methods.

The run and reporting rules don't allow such a substitution.

☑ Why do you think the run and reporting rules don't allow a training set substitution when they allow so much flexibility on compilers, optimizations, and libraries?

The rule would have to say that the training set can't be chosen based on an exact knowledge of the reference (data-collection run) inputs. The problem is determining whether a tester's training data violates this rule.