Name Partial Solution

Computer Architecture and Implementation
EE 7700-4
Practice Midterm Examination

October 1998, 0:00-23:59 CDT

Problem 1	_____	(35 pts)
Problem 2	_____	(30 pts)
Problem 3	_____	(35 pts)
Exam Total	_____	(100 pts)

Alias Partial Solution

To provide a good collection of study questions this practice exam has been made substantially longer than an actual 50-minute exam.

Problem 1: A processor uses the store-set scheme to predict memory dependencies. Up to one instruction per cycle can be issued and they are dynamically scheduled (may start or complete out of order), limited only by true dependencies. Dependency violations are checked as the earlier (store) instruction retires.

The code below is iterated several times, as indicated by the table. Each table row shows the time that an iteration will start (in cycles) and some register values that will be encountered. Store data and load addresses are available immediately but store addresses, registers `r1` and `r2`, are available only after a 7-cycle delay (seven cycles after they are needed by the store instruction). Instructions before the code fragment do not stall.

```
0x100 sw  0(r1), r10
0x104 add r20, r20, r10
0x108 sw  0(r2), r11
0x10c lw  r12, 0(r3)
0x110 lw  r13, 0(r4)
```

Time	r1	r2	r3	r4
500	1000	2000	1000	3000
600	1000	2000	1000	3000
700	1000	1000	3000	1000
800	1000	1000	1000	3000

(*a*) When the code starts the store set ID (SSID) table and last-fetched store table (LFST) are empty.

- Show each change to these tables and the approximate times the changes occur.
- Indicate where and when dependence violations take place.
- Indicate where and when instructions stall (whether or not the stall was necessary).

Use reservation station numbers (make them up) in the LFST.

(*b*) Ignoring cost, why would it not be a good idea to use instruction addresses rather than reservation station numbers in the LFST?

Because an instruction can be executing multiple times (for example, different iterations of a short loop), so an address would be ambiguous.

Problem 2: The program below finds the longest string in a table; a table entry holds either the address of a string or a zero (meaning no string). In this problem the program will encounter a three-entry table, entry zero is `"ab"`, entry one is zero, and entry two is `"c"`.

The program is run on a four-PE trace processor with eight-instruction traces and an infinite trace cache which is initially empty. Trace delineation is at the first indirect control-transfer instruction or the eighth instruction.

```
! r1 = 3, the number of entries in the string table.
! r4 holds base of string table.
! Line numbers in first column.
A0:  add r2, r0, r0  ! Initialize loop index.
A1:  add r7, r0, r0  ! Initialize maximum string length to zero.
LOOP:
B0:  sub r3, r1, r2  ! Check for end of loop. (Executed four times.)
B1:  beqz r3, DONE
B2:  slli r5, r2, #2 ! Offset of table entry.
B3:  add r5, r5, r4  ! Address of table entry.
B4:  lw r20, 0(r5)   ! Load table entry, 0 or address of string.
B5:  beqz r20, NEXT  ! Skip if zero. (Taken on second iteration.)
B6:  jal STRLEN      ! Find string length. (See below.)
B7:  sgt r8, r21, r7 ! Check if larger than max found so far.
B8:  bnez r8, NEXT
B9:  add r7, r21, r0 ! Update maximum.
NEXT:
B10: addi r2, r2, #1 ! Increment loop index.
B11: j LOOP
DONE:
C0:  add r1, r7, r0  ! Move maximum to r1
C1:  add r31, r28, r0 ! Return address.
C2:  jr r31          ! Return from procedure.

STRLEN:
D0:  subi r21, r20, #1
SNEXT:
D1:  addi r21, r21, #1
D2:  lb  r22, 0(r21)
D3:  bneq r22, SNEXT
D4:  sub r21, r21, r20
D5:  jr r31
```

(*a*) When execution starts (at `A0`) the trace cache is empty. Show the instructions included in each trace stored in the trace cache after the return at `C2`. Assume perfect next-trace prediction. Use the sample data provided in the problem statement and comments. Use the line numbers above, for example, Trace 0: A0, A1, ... Trace 1: ....

Trace 0	A0	A1	B0	B1	B2	B3	B4	B5
Trace 1	B6	D0	D1	D2	D3	D1	D2	D3
Trace 2	D1	D2	D3	D4	D5			
Trace 3	B7	B8	B9	B10	B11	B0	B1	B2
Trace 4	B3	B4	B5	B10	B11	B0	B1	B2
Trace 5	B3	B4	B5	B6	D0	D1	D2	D3
Trace 6	B7	B8	B10	B11	B0	B1	C0	C1
Trace 7	C2							

Execution order:

Trace 0, Trace 1, Trace 2, Trace 3, Trace 4, Trace 5, Trace 2 (only hit), Trace 6, Trace 7.

(b) What is the trace cache hit ratio in the execution of the code above as described in the previous subproblem? Show how a better trace delineation technique would improve the trace cache hit ratio executing the example above (assuming instantaneous trace cache updating).

Out of seven traces only one is used a second time (T2), so the hit ratio is 0.125.

Ending a trace at a backward branch would reduce the number of different traces cached for a loop ($e.g.$, one trace starting on the first instruction, one trace starting on the second instruction, etc.), increasing the chance that the trace needed is found. With such a delineation, the D1-D3 loop would be found several times.

(c) Show how the use of predicated instructions might improve the hit ratio in the trace cache on this example (perhaps using different input data, for example a longer table).

If the instruction at B9 were replaced by a predicated instruction, the branch at B8 would no longer be necessary. With the branch omitted, the same trace could be used for cases where the maximum is and is not updated.

Problem 3: Answer each question below.

(a) Write two programs, one which performs better on a system using a PAg branch predictor and one performing better on a system using a PAp branch predictor. C or assembler may be used. Both predictors use 16 branch outcomes. Assume that there is no aliasing.

(b) Why couldn't Shade be used to simulate eager execution?

Shade follows actual execution and so would not fetch instructions on the wrong side of the branch. Such instructions are needed to simulate eager execution.

(c) Assuming instantaneous table updates, what would be the prediction accuracy of a dual-delta data predictor on the address of the array element, `c + i`, in the code below. State any assumptions made about the initial state of the table(s) used by the predictor. Indicate the number of predictions made.

```
char c[20];
for(i=0; i<20; i+=2)
{
 c[i] = 'x';
}
```

Let the prediction table entry for the address be initialized to all zeros and the address of the first array element be 100. The table below shows the value prediction table entry corresponding to the store before each prediction is made, along with the actual value, prediction, and outcome.

Actual	Previous Value	Δ_1	Δ_2	Prediction	Outcome
100	0	0	0	0	Wrong
102	100	100	0	100	Wrong
104	102	2	0	102	Wrong
106	104	2	2	106	Correct
108	106	2	2	108	Correct

⋮

Out of 10 attempts 7 are correct for a prediction accuracy of 0.7.

(d) Should memory dependence prediction hardware be conservative (predict many dependencies where there may be none)?

No, because dependency violations between instructions are relatively rare. The unnecessary stalling would degrade performance.

(*e*) When issuing past a predicted branch, some systems make a backup copy of the register re-map table. How would misprediction recovery be different on a system that didn't backup the re-map table? Illustrate with an example; be sure to show how making a backup saves time.

Without a backup of the re-map table the processor would be forced to wait until a mispredicted branch instruction reached the head of the reorder buffer. At that time all instructions preceding the branch will have written their register values and so the re-map table would not be needed for following instructions. Instructions on the wrong path would be squashed, the re-map table would be reset, and issuing would resume on the correct side of the branch.

With a backup, instructions from the correct direction can be issued soon after a misprediction is detected because the physical register numbers for issued instructions preceding the branch are in the backup table. There is no need to wait for the branch to reach the head of the reorder buffer.

(*f*) Why is confidence estimation important in multiple-path (poly-path) eager execution?

Without confidence estimation the hardware would be forced to execute both sides of every predicted branch, possibly filling the processor with many paths. Each of the paths executes slowly (since resources are shared); only one of the paths is useful. With confidence estimation execution proceeds on only one side of a predicted branch when the confidence in the prediction is high, so there are fewer paths in the processor. With fewer paths, each path executes faster.

(*g*) Why might it be a good idea to omit direct (non-register) jumps in the traces stored in a trace cache? Why would it be a bad idea to omit the branches from the traces stored in a trace cache?

Since traces have a maximum size in most proposed trace cache implementations, omitting one instruction leaves room for including another, increasing the efficiency of the trace cache.

Unlike jumps which are always taken, branch instructions must be executed so their outcomes can be verified.

(*h*) A trace processor has 8 PE's each containing 2 of each type of functional unit; traces hold up to 16 instructions. Dispatch and retirement are as in the trace processor of Rotenberg *et al.* A 16-way superscalar processor has 16 copies of each type of functional unit, is fully bypassed, up to 16 instructions can issue per cycle; an instruction can issue if its operands are ready and a functional unit is free.

Show an ILP v. window size graph that could be used to argue that a trace processor would be nearly as fast as the superscalar even without data prediction. (Make up the data.) Show another graph that could be used to argue that the trace processor would be substantially slower. (Make up this data too.)

For the trace processor to be effective the available ILP would have to increase by two for each window size increase of 16.

For the trace processor to be inefficient on code that the superscalar processor handles well, the graph would have to show an ILP of 16 for a small window size (say 32), with only very small further increases. In this case, only the first few PEs in the trace processor would have instructions to execute.