LSU EE 4720 **Homework 7** Solution **Due: 5 May 2004**

The PPC 970 which was the subject of a question in Homework 6 is very similar to the POWER4 chip, the main differences being that the POWER4 lacks the packed-operand instructions and POWER4 includes two processors on a single chip.

Answer the following questions about the POWER4 based on information in "POWER4 System Microarchitecture," by Tendler et al, available via `http://www.ece.lsu.edu/ee4720/doc/power4.pdf`. *The questions can be answered without reading the entire paper. In particular, there is no need to read past page 17.*

Problem 1: Translate the following terms, as used in class, to their nearest equivalent in the paper.

- Integer Instruction → Fixed-point instruction.
- Instruction Queue → Issue queue.
- Reorder Buffer → Group completion table.
- Physical Register → Rename register.

Problem 2: The pipeline execution diagram below shows MIPS code on the dynamically scheduled system described in the study guide.

(*a*) Re-draw the diagram using the stages from POWER4. (Do not translate the instructions into the POWER assembly language.) Just show one iteration and assume that the four instructions are formed into one group. Also assume that the branch does not have a delay slot. Use stages `F1`, `F2`, and `F3` for the multiply.

See diagram. Note that in POWER4 there is a mandatory one-cycle gap between two dependent instructions. Yuk!

(*b*) In your diagram identify the *fetch* and *execute* pipelines, as defined in class.

See diagram. Instructions enter the fetch pipeline in IF and exit the fetch pipeline in MP where they are put in (dispatched to) issue queues. They wait in the issue queues until the scheduler chooses them, at which time they enter the execute pipeline.

```
 # Solution
 # Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22
LOOP:
 ldc1 f0, 0(t1)    IF IC D0 D1 D2 D3 XF GD MP IS RF EA DC FM WB XF CP
 mul  f2, f2, f0   IF IC D0 D1 D2 D3 XF GD MP             IS RF F1 F2 F3 WB XF CP
 addi t1, t1, 8    IF IC D0 D1 D2 D3 XF GD MP IS RF EX WB XF                        CP
 bneq t1, t2 LOOP  IF IC DB D1 D2 D3 XF GD MP       IS RF EX WB XF                 CP

 # Note: DB is an abbreviation for BP and D0 (branch uses both.)
 # XF -> Xfer;  FM -> Fmt;  IS -> ISS

 # Fetch Pipeline Stages: IF IC D0 D1 D2 D3 XF GD MP
 # Execute Pipeline Stages: IS RF EX,EA,DC,FM,F1-F6 WB XF
```

Problem 3: The POWER4 uses what is commonly called a *hybrid predictor* in which each branch is predicted by two different predictors and a third predictor predicts the prediction to use. One

predictor is something like the bimodal predictor discussed in class and the other is something like the gshare predictor discussed in class.

(*a*) Provide a code example in which the bimodal predictor described in class will do better than the POWER4's almost equivalent predictor. (Ignore the selector.)

What the PPC paper calls a local predictor is called a bimodal predictor in class, except that the PPC local predictor uses only a 1-bit entry in the BHT (rather than 2). Consider the loop below:

```
# Three-iteration loop.
LOOP:
 ...
 bneq r1, r2, LOOP   T T N  T T N  T T N
 nop
```

The predictor used in class would have an accuracy of 66.7%, misspredicting the not-taken executions of the branch. The local predictor would have an accuracy of 33.3%, because it would only correctly predict the second consecutive taken branch.

(*b*) How might the POWER4 designers justify the differences with the bimodal predictor given the lower performance in the example above?

For a given amount of storage, the PPC local predictor can have twice as many entries. Compared to one using a two-bit counter, the PPC would make more mispredictions due to using just one bit, but it would make fewer mispredictions due to collisions and so overall it would perform better (if the bimodal had many mispredictions due to collisions).

(*c*) Provide a code example in which the gshare predictor described in class outperforms the POWER4's almost equivalent predictor. (Ignore the selector.)

One difference between the PPC's global predictor and gshare is that in PPC the global history register has one bit for each fetch group (not the same as a dispatch group), whether or not it includes a branch. Fetch groups can be as large as eight instructions, dispatch groups can be as large as five instructions. When updating the GHR a fetch group without a CTI is treated like one containing a not-taken branch.

Consider the loop below:

```
# Five-iteration loop.
LOOP:
 # ...
 # ... seventeen instructions, none of them are CTIs ...
 #
 bneq r1, r2, LOOP   T T T T N T T T T N T T T T N ...
 nop
```

Each iteration would span at least three groups (*at least three*, because cache line boundaries might force contiguous instructions to be in separate fetch groups). Assume that the loop can be fetched in exactly three groups. For each iteration three bits would be shifted into the GHR, one for each group. A possible GHR value used for predicting the loop branch would be **TnnTnnTnnTn**, where **n** is the value inserted for groups not holding a branch (it would actually be a zero, for not taken). Since the GHR is eleven bits it can see three and part of a fourth iteration. Therefore the predictor would not be able to tell whether it was in the fourth iteration (where the branch would be taken) or the fifth iteration (where the branch would not be taken), in both cases the GHR would be **nnTnnTnnTnn**.

In a conventional gshare predictor the GHR would only include branch outcomes, and so for the code above it could easily distinguish the fourth and fifth iterations.

(*d*) How might the POWER4 designers justify the differences with the gshare predictor given the lower performance in the example above?

The logic to update and recover the GHR might be simpler.