Homework #7 — Due 12/7/98

1. How do branches cause problems when wide memories are used?

   If 4 instructions are fetched at once, and instruction 1 is a branch, instructions 2-3 have already been fetched, and will incorrectly be executed. Even worse, if it isn't determined that instruction 1 is a branch instruction until the execute phase, by then instructions 5-8 are also being fetched.

2. How do the level 1 cache, level 2 cache, and main memory compare in size and access time?

   Level 1 cache is small (128KB-512KB) and very fast (1-2 clock cycles access time). Level 2 cache is larger (1MB) and moderately fast (6-8 clock cycles access

Homework #7 — Due 12/7/98 (cont.)

time). Main memory is very large (32MB-64MB) and very slow (50-75 clock cycles access time).

3. Suppose the operations that need to be performed by one stage in a pipeline take longer than those in the other stages. Does this affect the pipeline? Explain.

   Two answers, depending on how you read the question:

   Yes, every stage in the pipeline has to be the same length, so if one stage wants 20ns and every other stage wants 10ns, then every stage will have to be 20ns. (Of course, in this situation, maybe it would be best to break that large stage down into two 10ns stages!)

Homework #7 — Due 12/7/98 (cont.)

   If one stage needs more time (e.g., it's waiting on a memory access), the pipeline can be stalled (no stages do any work) until it is ready.

4. Pipeline stalls and slips are pretty similar. When does each occur, and how is the pipeline affected by each?

   A stall can occur on an instruction fetch; in this case no stage in the pipeline does any work until the stall is resolved.

   A slip can occur using register interlocks, as a solution to write/read data hazards. In this case that one instruction stops, but the previous instructions continue their execution.

Homework #7 — Due 12/7/98 (cont.)

5. Explain the difference between data parallelism and control parallelism.

   In data parallelism, multiple PEs are doing the same thing, just on different pieces of data. When "if" statements are encountered, one set of PEs will process the "then" portion while the other PEs sit idle; then the first set sit idle while the second set process the "else" portion.

   In control parallelism, the PEs can be doing different things on different pieces of data. When "if" statements are encountered, one set of PEs can process the "then" portion while while the second set can process the "else" portion at the same time.