ECE511 Computer Architecture December 6, 2004

Lecture 24

Lecturer: Matt Frank *Scribe: Spencer Hoke*

1 Dynamic Multithreading I

This lecture covers Dynamic/Implicit Multithreading. We have always assumed that instructions pass through the pipeline in order to the Rename stage, then retire in order from the ROB. Now, we consider renaming out-of-order.

1.1 Problems with Renaming In Order

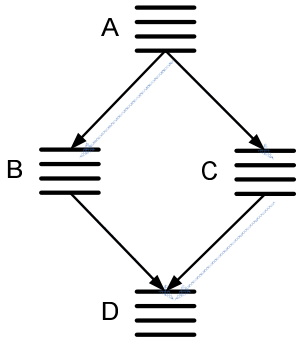

Figure 1: Paths converge after if statements.

In C programming, we know that the paths after an if statement will merge. Suppose the code in Figure 1 executes in order A→B→D, but during execution of D, the processor detects that the branch has been mispredicted. We flush the pipeline, and run path C→D, but D has already been executed. The instructions in block D could have been reused if instructions in D did not read from registers written to in B or C (we will have to check for this-to be discussed in the next lecture).

Another common example is shown in Figure 2. Procedure calls should return to the address following the call, so code in section B could be executed at the same time as the code in the procedure. Figure 3 shows another case where it would be beneficial to fetch instructions ahead of time. If there is little parallelization in the inner loop, then the processor could be fecthing instructions for the next iteration of the outer loop to try to find parallization there.

Big performance gains are possible if instructions in a block are not memory dependent on instructions in previous blocks.

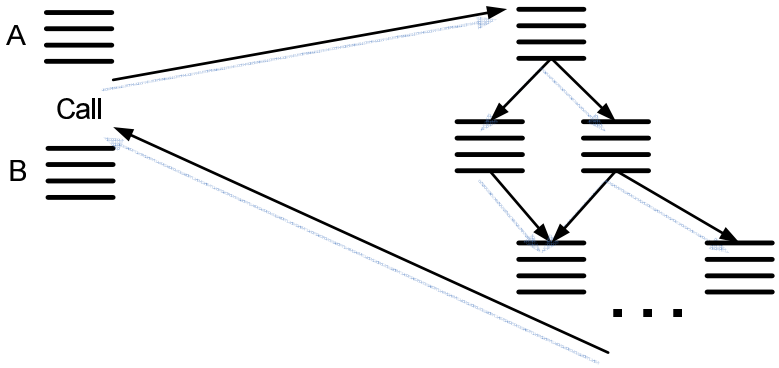

Figure 2: Calling procedures.

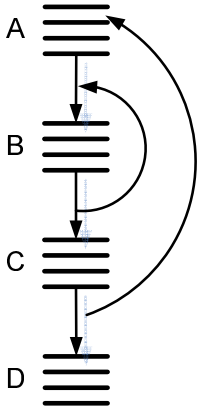

Figure 3: Nested loops.

1.2 Control Dependence

Definition 1 *Instruction X is* control independent *of branch B if all paths from B to the exit go through X.*

Definition 2 *Instruction X is* control dependent *on branch B if it will only execute for a given branch decision.*

See Figures 5 and 5 for examples.

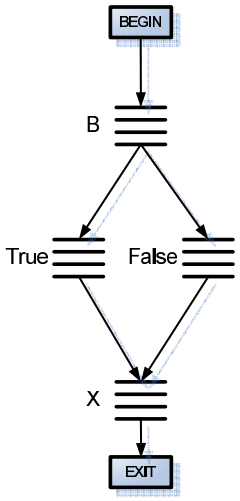

Figure 4: X is *control independent* of B. The True and False blocks are *control dependent* on B.

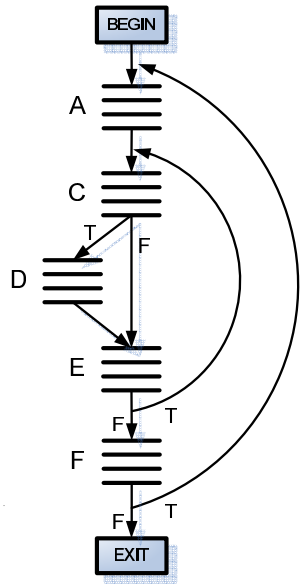

Figure 5: Blocks A, C, E, and F are control dependent on Begin. Block D is control dependent on Branch C. Blocks C and E are control dependent on Branch E. Blocks A, C, E, and F are control dependent on Branch F.

1.3 Control Dependence Analysis

In Figure 4, we want the processor to fetch instructions from X if it has extra time while executing B. Preference should be given to the next instructions in B, so that instructions are fetched from X when the processor would be stalling during normal B execution. Use seperate AGen's, RAT's, ROB's, for each of the threads.

The compiler can't iterate through all possible paths between the start and end of code because the number of paths is infinite. Instead, they can do backwards passes for control analysis (specific rules were not covered).

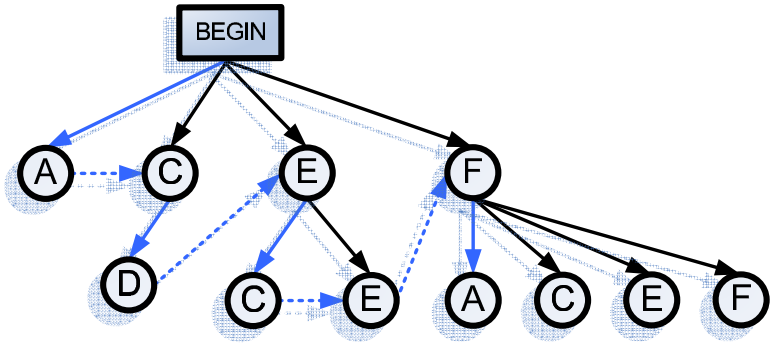

Figure 6: Example execution path through Figure 5. Solid arrow indicate blocks that were started at the same time. Blue lines indicate program flow (A C D E C E F A ...).

Figure 6 shows a possible execution path of the code in Figure 5. Whenever the program starts at Begin, the processor can start threads executing blocks A, C, E, and F, since they were control dependent on Begin. After executing C, if the branch is taken, the processor starts block D. After executing D and E, the branch is not taken and control moves to the already executing F, at which point A, C, E, and F are all started again. This is a tree traversal, which can be implemented like a stack. Anything currently in the tree could be executed because it will all be needed eventually. Deciding what address to fetch next is an open research question. Preference can be given to branches with higher confidence.

In the ROB, instructions should retire in order, but they are not fetched in order and the ROB doesn't know how many instructions could be between those that have been fetched. Linked list structures for each block have to be implemented with pointers to the other lists and the ordering of the lists is determined with feedback from the Address Generator.

We can only use control dependence analysis results as hints for how to generate the next addresses because there is always a possibility of abnormal termination (interrupts, *etc.*).