

HP/Intel EPIC Architecture

- By 2000: able to issue 16 instructions per cycle
- Need architecture support or compilers won't be able to expose sufficient ILP
- EPIC: Explicitly Parallel Instruction Computing
- Running example
 - Processor model: 6-issue processor, can execute one branch per cycle
 - Compiler tries to schedule as many instructions in parallel as possible
 - All operations have latency of one cycle except loads with 2 cycles
- Measures of execution time:
 - longest path length
 - average schedule length assuming branches taken 25% of time
 - Initial schedule for example is in figure 1(b)

Speculation

- Guessing outcome of run-time event at compile time
- **Two kinds of speculation**
- Control speculation (the kind we have been doing)
 - Guess a branch will go a particular direction
 - Compiler moves operations across branches
- Data dependence speculation
 - Breaks data flow dependencies between memory operations
 - First operation writes a value to an address
 - Second operation reads a value from an address and the two addresses MAY conflict
 - Problem: hard for compiler to know whether addresses conflict
 - If it can't tell, must assume they might
 - ambiguous memory dependencies
 - Speculation: guess that the two addresses are independent
 - Data speculation used in EPIC to help compiler scheduler to reorder memory operations

- When applying control and data speculation, get new schedule in figure 1(c)
- Do all the data loads in parallel into registers
- The second and third data loads are **control-speculative**; whether they are executed depends on the outcome of the first branch
- The third load is **memory-dependent** on the store operation (5):
 - statement (5) writes MEM[r2]
 - statement (8) reads MEM[r4]
- We don't know whether r2 and r4 contain the same address
 - If the store changes the value of memory at this address, then the earlier load of MEM[r4] will put the wrong value into register r14
- Compiler can't be sure the addresses don't conflict
- **Guesses that they don't conflict: data speculation**
 - If wrong, then will have to undo this operation

- Due to breaking control dependencies, speculated operations execute more frequently than non-speculated counterparts in original code
- **Exceptions**
- Both speculative and non-speculative instructions can generate exceptions
- Non-speculative instructions should report exceptions immediately
- Speculative instructions are not allowed to initiate exceptions
 - Lose hundreds of cycles to service page faults, tlb misses, slow cache misses
- Add S-bit to each operation that can be speculated
- Add DS-bit to each load that can be data-speculated
- **Delayed exception handling:**
- Add single-bit E-tag to each register
- When set, indicates an exception occurred in generating the value stored in the register
- Propagates these E-tags to instructions that are dependent on the speculated value

- **Delayed exception handling (cont.):**
- Check operation: detects problems in previous speculative execution
 - At same position as original instruction
- When check is reached, check E-tags, handle exceptions, and perform repair if speculation was incorrect

- **Memory Conflicts**
- For data speculation, must repair if actual data dependence between speculated load and one or more stores
- Check operation determines whether dependence actually existed

- Store addresses of data-speculative loads until their independence from intervening stores is established

- **Memory Conflict Buffer (MCB)**
- Temporarily associates destination register number of speculative load with memory address from which it was loaded
- Check destination addresses of subsequent stores against these load addresses
- Intiate recovery if conflict detected

Predication

- Conditional execution of individual operations based on Boolean guards
 - Operation with TRUE predicate executes normally
 - Operation with FALSE predicate is prevented from modifying processor state
- Implemented as predicate register values
- **Allows compiler to restructure program control**
 - Eliminate branches, mispredication penalties
- **Allows separate control flow paths to be overlapped & simultaneously executed in a single thread of control**

- **Predicate define operations (examples)**
- unconditional-true (ut):
 - set to true of condition is true (2) uses `p1ut(r11==0)`
- or-false (of):
 - allows OR-ing of multiple conditions;
 - initial value of predicate is 0;
 - `predicate = (old predicate) OR !(newCondition)`
 - if new condition is false, OR a 1 value and set predicate true: (2) uses `p4of(r11==0)`
- and-true (at):
 - allows AND-ing of multiple conditions;
 - initial value of predicate is 1;
 - `predicate = (old predicate) AND (newCondition)`
 - if new condition is true, AND a 1 value with current predicate: (6') uses `p3at=(r13==1)`
- Predicated code 1(d) removes all branches from original code
- **Increased ILP by overlapping execution of “then” and “else” paths of original code**

Combining Speculation and Predication

- Different means of improving performance
- **Speculation: break control and memory dependencies**
- **Predication: restructure control flow and overlap separate execution paths**

- Techniques can be mutually beneficial
- Figure 2

- Although all branches have been removed, still do control speculation
- **Promotion: Predict outcome of predicates**
 - Remove predicates on second two load operations, making them speculative
 - Allows compiler to hoist operations earlier, before their predicates are computed

- **Misconception: not many opportunities for control speculation**
 - Lots of opportunities to speculate on the predicate outcomes

- **Combined techniques give more speedup than just multiplying the speedups of separate optimizations**