Multiple-Issue Processors: Motivation

- Ideal processor: CPI of 1
  - no hazards, 1-cycle memory latency
- Realistic processor: CPI ~1
  - Dynamic scheduling – avoids stalls on WAR & WAW dependencies
  - Branch prediction – avoids stalls on control flow dependencies
  - Caches – minimize AMAT
- Question: can we do better than that???

Multiple-Issue Processors

- Answer: Yes!
  - Use more transistors: replicate the pipeline!
  - start more than one instruction in the same clock cycle
  - CPI < 1 (or IPC > 1, Instructions per Cycle)

- Two approaches:
  - Superscalar: instructions are chosen dynamically by the hardware
  - VLIW (Very Long Instruction Word): instructions are chosen statically by the compiler (and assembled in a single long "instruction")

Superscalar Processors

- Hardware attempts to issue up to *n* instructions on every cycle, where *n* is the issue width of the processor and the processor is said to have *n* issue slots and to be a *n-wide* processor
- Instructions issued must respect data dependences
- In some cycles not all issue slots can be used
- Extra hardware is needed to detect more combinations of dependences and hazards and to provide more bypasses
- Branches?
  - With branch prediction, we can predict branches and fetch instructions
  - Can we execute such predicted instructions?

Speculative Execution

- Speculative execution – *execute* control-dependent instructions even when we are not sure if they should be executed
- Hardware undo, in case of a misprediction
  - Software recovery too costly, performance-wise
- Key Idea: Execute out-of-order but commit in order
  - Commit: the results and side-effects (e.g., flags, exceptions) of an instruction are made visible to the rest of the system
- Tomasulo + multi-issue + speculation
  - Foundation for today’s high-performance processors

Extending Tomasulo to Support Speculation

- Approach: buffer result until instruction ready to commit (i.e., known to be non-speculative)
  - Use buffered result for forwarding to dependent instructions
  - Discard buffered result if the instruction is on a mis-speculated execution path
  - At commit, write buffered result to register or memory
- Decouples forwarding (potentially speculative) from update of architecturally-visible state (non-speculative)
  - Architecturally visible state: registers (R0-Rn, F0-Fn, memory)

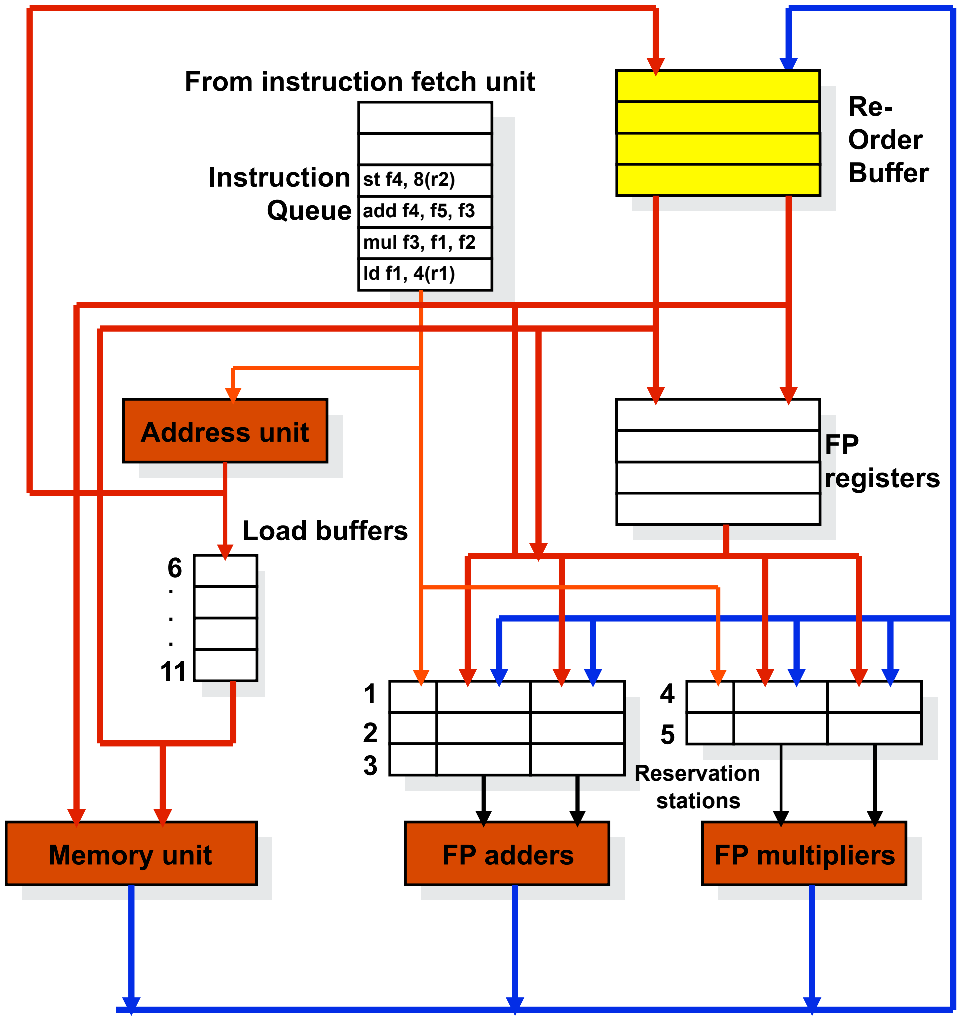
Enabling Speculation with the Reorder Buffer

New structure: Reorder Buffer (ROB)

- Holds completed results until commit time
- Organized as a queue ordered by program (i.e., fetch) order
- Takes over the role of the reservation stations for tracking dependencies and bypassing values
  - Accessed by dependent instructions for forwarding of completed, but not-yet-committed, results
  - Reservation stations still needed to hold issued instructions until they begin execution
- Flushed once mis-speculation is discovered (mispredicted branch commits)
- Enables precise exceptions
  - exception state recorded in ROB
  - flushed if exception occurred on a mis-predicted path

Tomasulo with Hardware Speculation

- Issue:
  - Get instruction from queue
  - Issue if an RS is free and an ROB entry is also free
  - Stall if no RS or no free ROB entry
  - Instructions now tagged with ROB entry number, not RS.Id
- Execute:
  - Same as before: monitor CDB and start instruction when operands are available
- Write Result:
  - CDB broadcasts result with ROB identifier
  - ROB captures result to commit later
  - Store operations also saved in the ROB until store data is available and store instruction is committed
- Commit:
  - If branch, check prediction and squash following instructions if incorrect
  - If store, send data and address to memory unit and perform write action
  - Else, update register with new value and release ROB entry



VLIW Processors

- Compiler chooses and "packs" independent instructions into a single long "instruction" word or "bundle"
- Compiler responsible for avoiding hazards
  - Keeps hardware simple
  - Compiler's schedule must be conservative to guarantee safety
- Not all portions of the long instruction word will be used in every cycle
  - Compiler must be able to expose a lot of parallelism in the schedule to attain good performance
- Example:

MEM op 1	**MEM op 2**	**FP op 1**	**FP op 2**	**INT op**
`ld f18,-32(r1)`	`ld f22,-40(r1)`	`addd f4,f0,f2`	`addd f8,f6,f2`	

VLIW Processors (con'd)

- Key challenge for VLIW processors:
  - find control-independent work to fill each word
  - Cover data-dependent stalls (e.g., F.DIV immediately followed by a use of the result) with independent instructions

- Solutions:
  - Get rid of control flow
    - Predication
    - Loop unrolling
  - Move code around to maximize scheduling opportunities and minimize stalls

Superscalar vs. VLIW Processors

SuperScalars

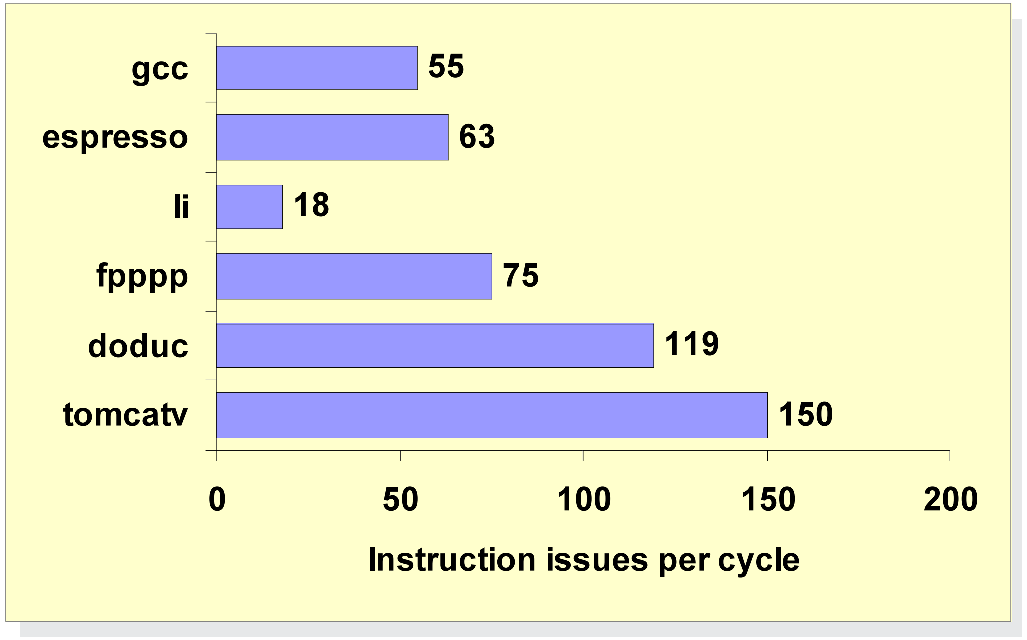
+ Able to handle dynamic events like cache misses, unpredictable memory dependences, branches, etc.
+ Can exploit old binaries from previous implementations
- Complexity limits issue width to 4-8

VLIW

+ Much simpler hardware implementation
+ Implementations can have wider issue than superscalars
- Require more complex compiler support
- Cannot use old binaries when pipeline implementation changes
- Code size increases because of empty issue slots

What are the limitations to ILP?

- Fundamental limit to available ILP in a program
- Limitations on max issue width and ROB size
- Effects of realistic branch prediction
- The effect of limited numbers of rename registers (reservation stations)



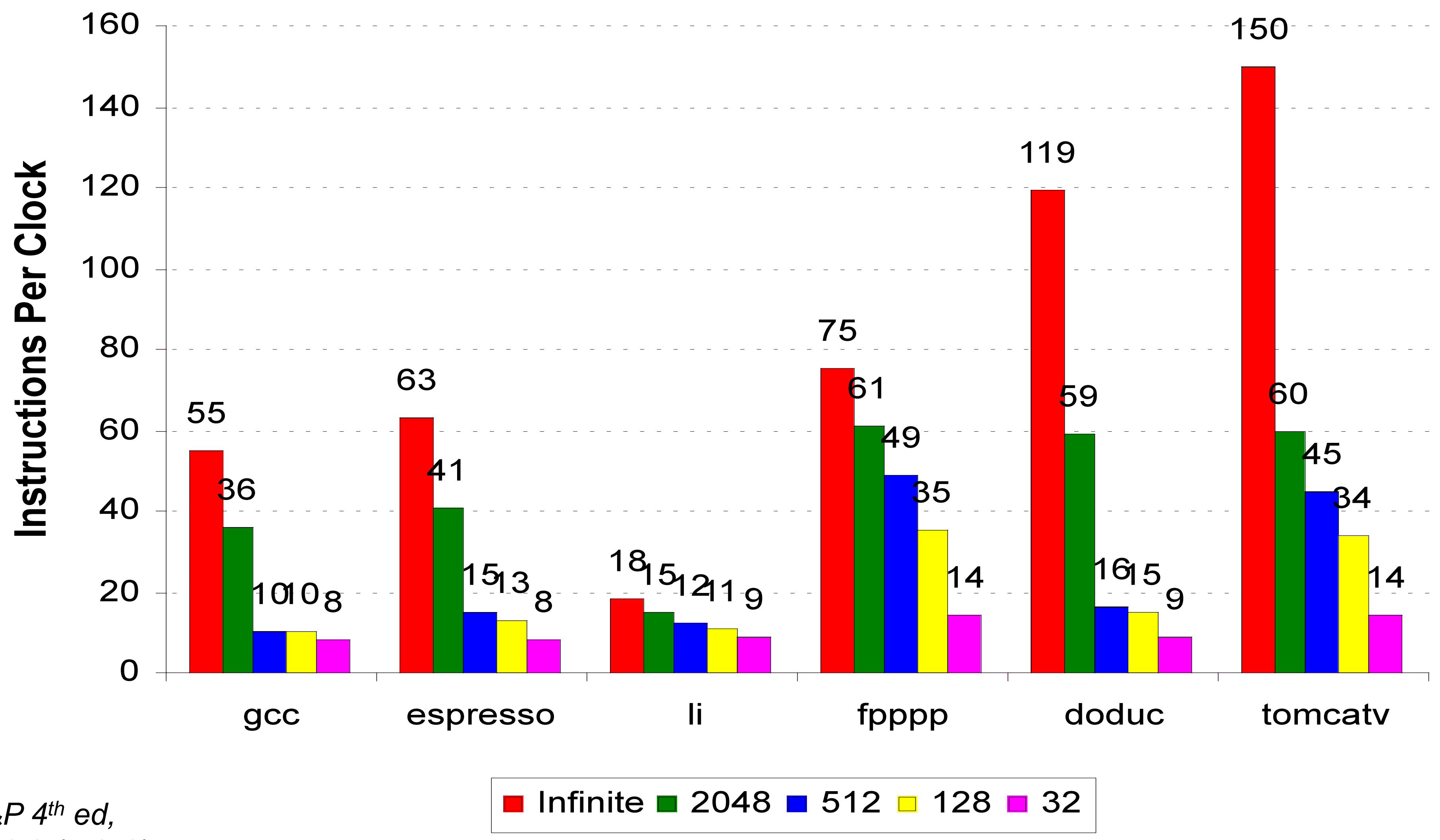
Available ILP in a perfect processor, with none of the above constraints.

- 6 SPEC92 benchmarks
- the first 3 are Int, the last 3 are FP

These levels of ILP are impossible to achieve in practice due to limitations above

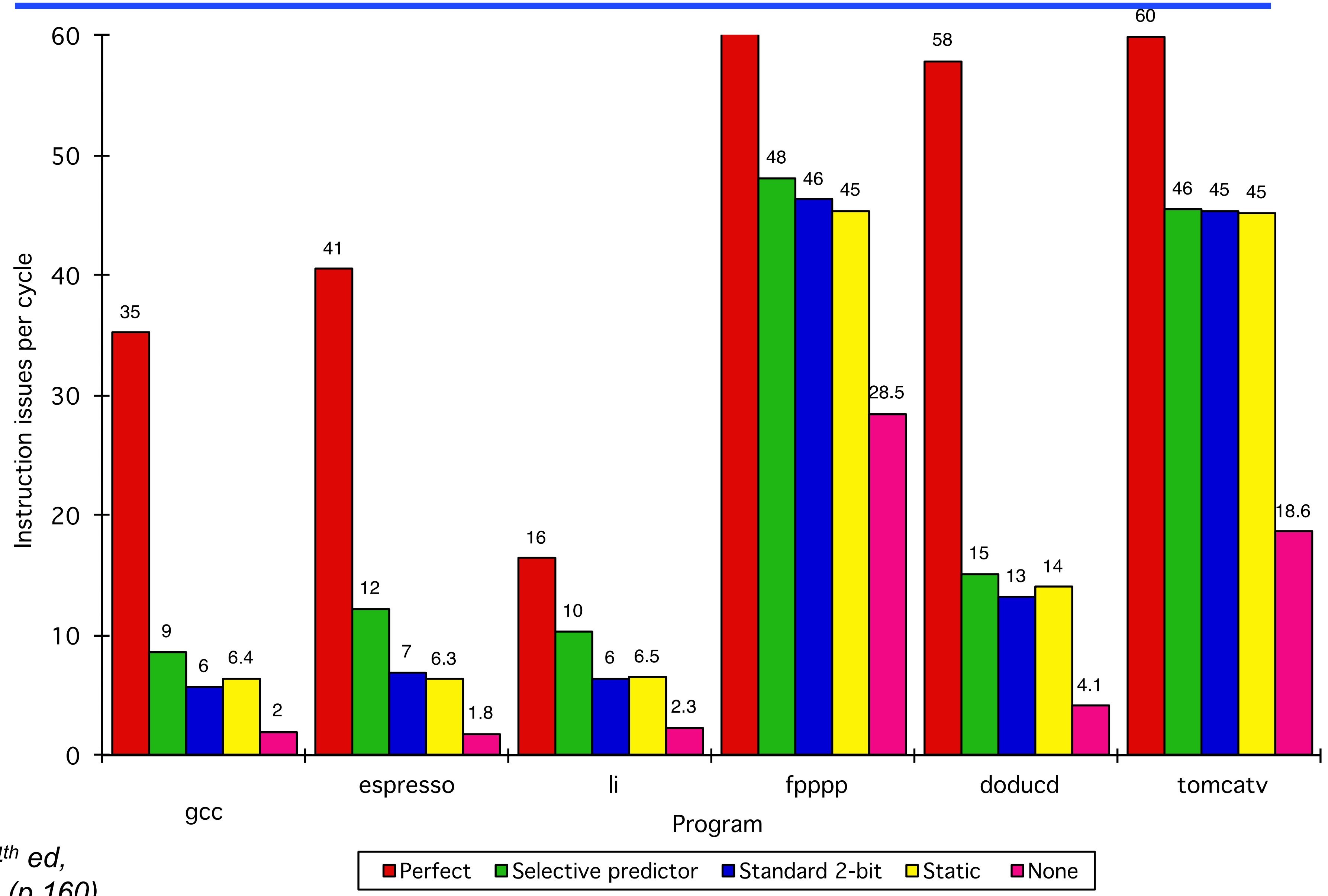
H&P 4th ed, fig 3.1 (p.157)

Effect of Instruction Window (i.e., ROB)



H&P 4th ed, fig 3.2 (p.159)

Effect of Branch prediction (2K ROB)



H&P 4th ed, fig 3.3 (p.160)

Limits to Multiple-issue

- Fundamental limits to ILP in most programs:
  - Need *N* independent instructions to keep a W-issue processor busy, where *N = W * pipeline depth*
  - Data and control dependences significantly limit amount of ILP

- Complexity of the hardware based on issue width:
  - Number of functional units increases linearly → OK
  - Number of ports for register file increases linearly → bad
  - Number of ports for memory increases linearly → bad
  - Number of dependence tests increases quadratically → bad
  - Bypass/forwarding logic and wires increases quadratically → bad

These two tend to ultimately limit the width of practical dynamically-scheduled superscalars

Summary of Factors Limiting ILP in Real Programs

- Compared with an ideal processor
  - Limited instruction window
  - Imperfect branch prediction (pipeline flushes)
  - Limited issue width
  - Instruction fetch delays (cache misses, across-block fetch)

- Implications for future performance growth?
  - Single processor has inherent limits
  - To use future silicon area, need to go to multiple processors