

Multiple Issue Alternatives

- **Superscalar** (hardware detects conflicts)
 - Statically scheduled (in order dispatch and hence execution; cf. Compaq (DEC)Alpha 21164)
 - Dynamically scheduled (in order issue, out of order dispatch and execution; cf. MIPS 10000, IBM Power PC 620, Intel Pentium II, III, and IV), Compaq (DEC)Alpha 21124, Sun UltraSparc etc.)
- **VLIW – EPIC** (Explicitly Parallel Instruction Computing)
 - Compiler generates “bundles” of instructions that can be executed concurrently (cf. Intel Merced – Itanium, lot of DSP’s)

Mult. Issue CSE 471 Autumn 01

1

Multiple Issue for Static/Dynamic Scheduling

- Issue in order
 - Otherwise bookkeeping too complex (the old “data flow” machines could issue any ready instruction in the whole program)
 - Check for structural hazards; if any stall
- Dispatch for static scheduling
 - Check for data dependencies; stall adequately
 - Can take forwarding into account
- Dispatch for dynamic scheduling
 - Dispatch out of order (reservation stations, instruction window)
 - Requires possibility of dispatching concurrently dependent instructions (otherwise little benefit over static. sched.)

Mult. Issue CSE 471 Autumn 01

2

Impact of Multiple Issue on IF

- IF: Need to fetch more than 1 instruction at a time
 - Simpler if instructions are of fixed length
 - In fact need to fetch as many instructions as the issue stage can handle in one cycle
 - Simpler if restricted not to overlap I-cache lines
 - But with branch prediction, this is not realistic hence introduction of (instruction) *fetch buffers* and *trace caches*
 - Always attempt to keep at least as many instructions in the fetch buffer as can be issued in the next cycle (BTB’s help for that)
 - For example, have an 8 wide instruction buffer for a machine that can issue 4 instructions per cycle

Mult. Issue CSE 471 Autumn 01

3

Stalls at the IF Stage

- Instruction cache miss
- Instruction buffer is full
 - Most likely there are stalls in the stages downstream
- Branch misprediction
- Instructions are stored in several I-cache lines
 - In one cycle one I-cache line can be brought into fetch buffer
 - A basic block might start in the middle (or end) of an I-cache line
 - Requires several cache lines to fill the buffer
 - The ID (issue-dispatch) stage will stall if not enough instructions in the fetch buffer

Mult. Issue CSE 471 Autumn 01

4

Sample of Current Micros

- Two instruction issue: Alpha 21064, Sparc 2, Pentium, Cyrix
- Three instruction issue: Pentium II, III (but 5 uops from IF/ID to EX; AMD has 4 uops)
- Four instruction issue: Alpha 21164, Alpha 21264, Power PC 620, Sun UltraSparc, HP PA-8000, MIPS R10000
- Many papers written in 1995-98 predicted 16-way issue by 2000. We are still at 4!

Mult. Issue CSE 471 Autumn 01

5

The Decode Stage (simple case: dual issue and static scheduling)

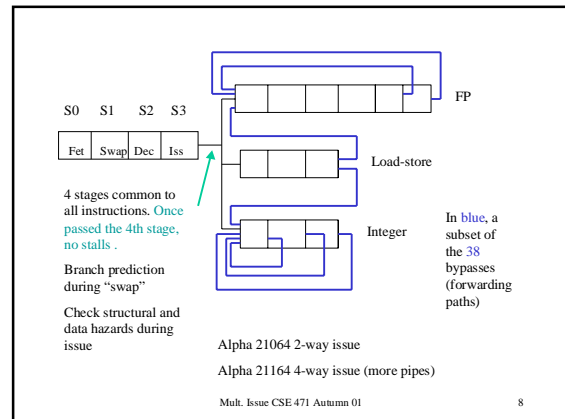
- ID = Issue + Dispatch
- Look for conflicts between the (say) 2 instructions
 - If one integer unit and one f-p unit, only check for structural hazard, i.e. the two instructions need the same f-u (easy to check with opcodes)
 - Slight difficulty for integer ops that are f-p load/store/move (potential multiple accesses to f-p register file; solution: provide additional ports)
 - RAW dependencies resolved as in single pipelines
 - Note that the load delay (assume 1 cycle) can now delay up to 3 instructions, i.e., 3 issue slots are lost

Mult. Issue CSE 471 Autumn 01

6

Decode in Simple Multiple Issue Case

- If instructions i and $i+1$ are fetched together and:
 - Instruction i stalls, instruction $i+1$ will stall
 - Instruction i is dispatched but instruction $i+1$ stalls (e.g., because of structural hazard = need the same f-u), instruction $i+2$ will not advance to the issue stage. It will have to wait till both i and $i+1$ have been dispatched



Alpha 21064

- IF – S0: Access I-cache
 - Prefetcher fetches 2 instructions (8 bytes) at a time
- Swap stage - S1:
 - Prefetcher contains branch prediction logic tested at this stage: 4 entry return stack; 1 bit/instruction in the I-cache + static prediction BTFNT
 - Initial decode yields 0, 1 or 2 instruction potential issue; align instructions depending on the functional unit there are headed for.
- End of decode: S2.
 - Check for WAW and WAR (my guess)

Alpha 21064 (c'ed)

- Instruction Issue: S3
 - Check for RAW; forwarding etc
- Conditions for 2 instruction issue (S2 and S3)
 - The first instruction must be able to issue (in order execution)
 - Load/store can issue with an operate except stores cannot issue with an operate of different format (share the same result bus)
 - An integer op. can issue with a f-p op.
 - A branch can issue with a load/store/operate (but not with stores of the same format)

The Decode Stage (dynamic scheduling)

- Decode means:
 - Dispatch to either
 - A centralized instruction window common to all functional units (Pentium Pro and Pentium III -- I think)
 - Reservation stations associated with functional units (MIPS 10000, AMD K5, IBM Power PC 620)
 - Rename registers (if supported by architecture)
 - Note the difficulty when renaming in the same cycle
 $R1 < R2 + R3; R4 < R1 + R5$
 - Set up entry at tail of reorder buffer (if supported by architecture)
 - Issue operands, when ready, to functional unit

Stalls in Decode (issue/dispatch) Stage

- There can be several instructions ready to be dispatched in same cycle to same functional unit
- There might not be enough bus/ports to forward values to all the reservation stations that need them in the same cycle

The Execute Stage

- Use of forwarding in the case of static scheduling
- Use of broadcast bus and reservation stations for dynamic scheduling
- We'll talk at length about **memory operations** (load-store) when we study memory hierarchies

Mult. Issue CSE 471 Autumn 01

13

The Commit Step (in-order completion)

- Recall: need of a mechanism (**reorder buffer**) to:
 - “Complete” instructions in order. This **commits** the instruction. Since multiple issue machine, should be able to commit (retire) several instructions per cycle
 - Know when an instruction has completed non-speculatively, i.e., what to do with branches
 - Know whether the result of an instruction is correct, i.e., what to do with exceptions

Mult. Issue CSE 471 Autumn 01

14

Power PC 620 (see figure in book)

- Issue stage. Up to 4 instructions issued/cycle except if structural hazards such as:
 - No reservation station available
 - No rename register available
 - Reorder buffer is full.
 - Two operations (e.g., forwarding operands) for the same unit. Only one write port/set of reservation stations for each unit.
 - Miscellaneous structural hazards, e.g., too many concurrent reads to the register file
- First 3 due to “the program”, last 2 to the implementation

Mult. Issue CSE 471 Autumn 01

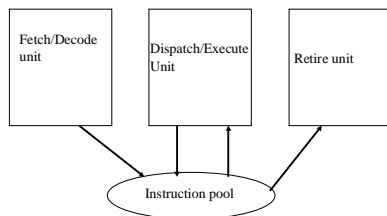
15

Pentium Pro

- Fetch-Decode unit
 - Transforms instructions into micro-operations (uops) and stores them in a global reservation table (instruction window). Does register renaming (RAT = register alias table)
- Dispatch (aka issue)-execution unit
 - Issues uops to functional units that execute them and temporarily store the results (the reservation table is 5-ported, hence 5 uops can be issued concurrently)
- Retire unit
 - Commits the instructions in order (up to 3 commits/cycle)

Mult. Issue CSE 471 Autumn 01

16



The 3 units of the Pentium Pro are “independent” and communicate through the instruction pool

Mult. Issue CSE 471 Autumn 01

17

Impact on Branch Prediction and Completion

- When a conditional branch is decoded:
 - Save the **current physical-logical mapping**
 - Predict and proceed
- When branch is ready to commit (head of buffer)
 - If prediction correct, discard the saved mapping
 - If prediction incorrect
 - Flush all instructions following mispredicted branch in reorder buffer
 - Restore the mapping as it was before the branch as per the saved map
- Note that there have been proposals to execute both sides of a branch using **register shadows**
 - limited to one extra set of registers

Mult. Issue CSE 471 Autumn 01

18

Exceptions

- Instructions carry their exception status
- When instruction is ready to commit
 - No exception: proceed normally
 - Exception
 - Flush (as in mispredicted branch)
 - Restore mapping (more difficult than with branches because the mapping is not saved at every instruction; this method can also be used for branches)

Limits to Hardware-based ILP

- Inherent lack of parallelism in programs
 - Partial remedy: loop unrolling and other compiler optimizations;
 - Branch prediction to allow earlier issue and dispatch
- Complexity in hardware
 - Needs large bandwidth for instruction fetch (might need to fetch from more than one I-cache line in one cycle)
 - Requires large register bandwidth (multiported register files)
 - Forwarding/broadcast requires "long wires" (long wires are slow) as soon as there are many units.

Limits to Hardware-based ILP (c'ed)

- Difficulties specific to the implementation
 - More possibilities of structural hazards (need to encode some priorities in case of conflict in resource allocations)
 - Parallel search in reservation stations, reorder buffer etc.
 - Additional state savings for branches (mappings), more complex updating of BPT's and BTB's.
 - Keeping precise exceptions is more complex