Computer Architecture and Operating Systems
Lecture 10: Processor. Multiple Issue. Exceptions.

Andrei Tatarnikov

atatarnikov@hse.ru
@andrewt0301

Instruction-Level Parallelism (ILP)

- Pipelining: executing multiple instructions in parallel

- To increase ILP
  - Deeper pipeline
    - Less work per stage $\Rightarrow$ shorter clock cycle
  - Multiple issue
    - Replicate pipeline stages $\Rightarrow$ multiple pipelines
    - Start multiple instructions per clock cycle
    - CPI < 1, so use Instructions Per Cycle (IPC)
    - E.g., 4GHz 4-way multiple-issue
      - 16 BIPS, peak CPI = 0.25, peak IPC = 4
    - But dependencies reduce this in practice

Multiple Issue

- Static multiple issue
  - Compiler groups instructions to be issued together
  - Packages them into "issue slots"
  - Compiler detects and avoids hazards
- Dynamic multiple issue
  - CPU examines instruction stream and chooses instructions to issue each cycle
  - Compiler can help by reordering instructions
  - CPU resolves hazards using advanced techniques at runtime

Speculation

- "Guess" what to do with an instruction
  - Start operation as soon as possible
  - Check whether guess was right
    - If so, complete the operation
    - If not, roll-back and do the right thing
- Common to static and dynamic multiple issue
- Examples
  - Speculate on branch outcome
    - Roll back if path taken is different
  - Speculate on load
    - Roll back if location is updated

Compiler/Hardware Speculation

- Compiler can reorder instructions

  - e.g., move load before branch

  - Can include "fix-up" instructions to recover from incorrect guess

- Hardware can look ahead for instructions to execute

  - Buffer results until it determines they are actually needed

  - Flush buffers on incorrect speculation

Static Multiple Issue

- Compiler groups instructions into "issue packets"
  - Group of instructions that can be issued on a single cycle
  - Determined by pipeline resources required
- Think of an issue packet as a very long instruction
  - Specifies multiple concurrent operations
  - $\Rightarrow$ Very Long Instruction Word (VLIW)

Scheduling Static Multiple Issue

- Compiler must remove some/all hazards

  - Reorder instructions into issue packets

  - No dependencies with a packet

  - Possibly some dependencies between packets

    - Varies between ISAs; compiler must know!

  - Pad with nop if necessary

RISC-V with Static Dual Issue

- Two-issue packets
  - One ALU/branch instruction
  - One load/store instruction
  - 64-bit aligned
    - ALU/branch, then load/store
    - Pad an unused instruction with nop

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

Dynamic Multiple Issue

- "Superscalar" processors
- CPU decides whether to issue 0, 1, 2, … each cycle
  - Avoiding structural and data hazards
- Avoids the need for compiler scheduling
  - Though it may still help
  - Code semantics ensured by the CPU

Dynamic Pipeline Scheduling

- Allow the CPU to execute instructions out of order to avoid stalls
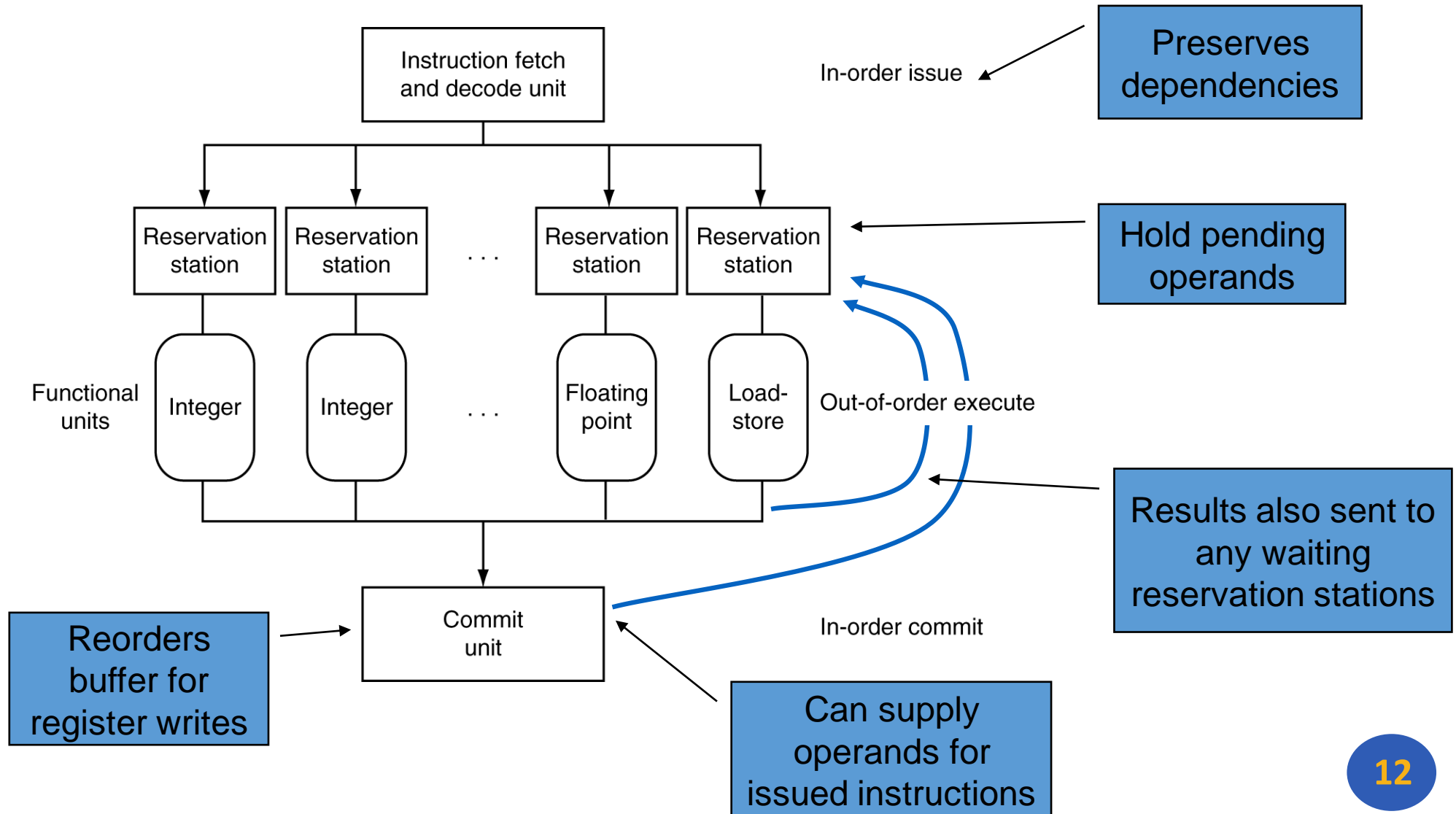 - But commit result to registers in order

- Example
  ```
  ld   x31,20(x21)
  add  x1,x31,x2
  sub  x23,x23,x3
  andi x5,x23,20
  ```
  - Can start sub while add is waiting for ld

Why Do Dynamic Scheduling?

- Why not just let the compiler schedule code?
- Not all stalls are predicable
  - e.g., cache misses
- Can't always schedule around branches
  - Branch outcome is dynamically determined
- Different implementations of an ISA have different latencies and hazards

Dynamically Scheduled CPU

Instruction fetch
and decode unit

In-order issue

Preserves
dependencies

Reservation
station

Reservation
station

. . .

Reservation
station

Reservation
station

Hold pending
operands

Functional
units

Integer

Integer

. . .

Floating
point

Load-
store

Out-of-order execute

Results also sent to
any waiting
reservation stations

Commit
unit

In-order commit

Reorders
buffer for
register writes

Can supply
operands for
issued instructions

Does Multiple Issue Work?

- Yes, but not as much as we'd like
- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
  - e.g., pointer aliasing
- Some parallelism is hard to expose
  - Limited window size during instruction issue
- Memory delays and limited bandwidth
  - Hard to keep pipelines full
- Speculation can help if done well

13

Conclusion

- ISA influences design of datapath and control
- Datapath and control influence design of ISA
- Pipelining improves instruction throughput using parallelism
  - More instructions completed per second
  - Latency for each instruction not reduced
- Hazards: structural, data, control
- Multiple issue and dynamic scheduling (ILP)
  - Dependencies limit achievable parallelism
  - Complexity leads to the power wall

Any Questions?

```
                  .text
    __start:   addi t1, zero, 0x18
               addi t2, zero, 0x21
  cycle:       beq t1, t2, done
               slt t0, t1, t2
               bne t0, zero, if_less
               nop
               sub t1, t1, t2
               j cycle
               nop
  if_less:     sub t2, t2, t1
               j cycle
  done:        add t3, t1, zero
```