Multiple Instruction Issue

Instruction-level Parallelism (IPL)
the ability to execute multiple instructions together

Pipelining provides IPL by overlapping instruction execution

Superscalar systems replicate hardware to provide IPL
multiple instructions can be sent through separate pipelines
multiple units can execute instructions simultaneously

Multiple issue means starting more than 1 instruction at a time

Static vs. Dynamic Multiple Issue

Static multiple issue is based on:

- compiler deciding which instruction can execute together
- Independent instructions are grouped into packets or bundles
- This packaging occurs prior to execution (compile time)
- Bundles or packets are assigned to issue slots
- These are also called very long instruction words (VLIW)
- Packets are issued within a single clock cycle
- Packed instructions must map to separate resources
  - Instruction mix may be restricted

Static vs. Dynamic Multiple Issue

Dynamic multiple issue is based on:

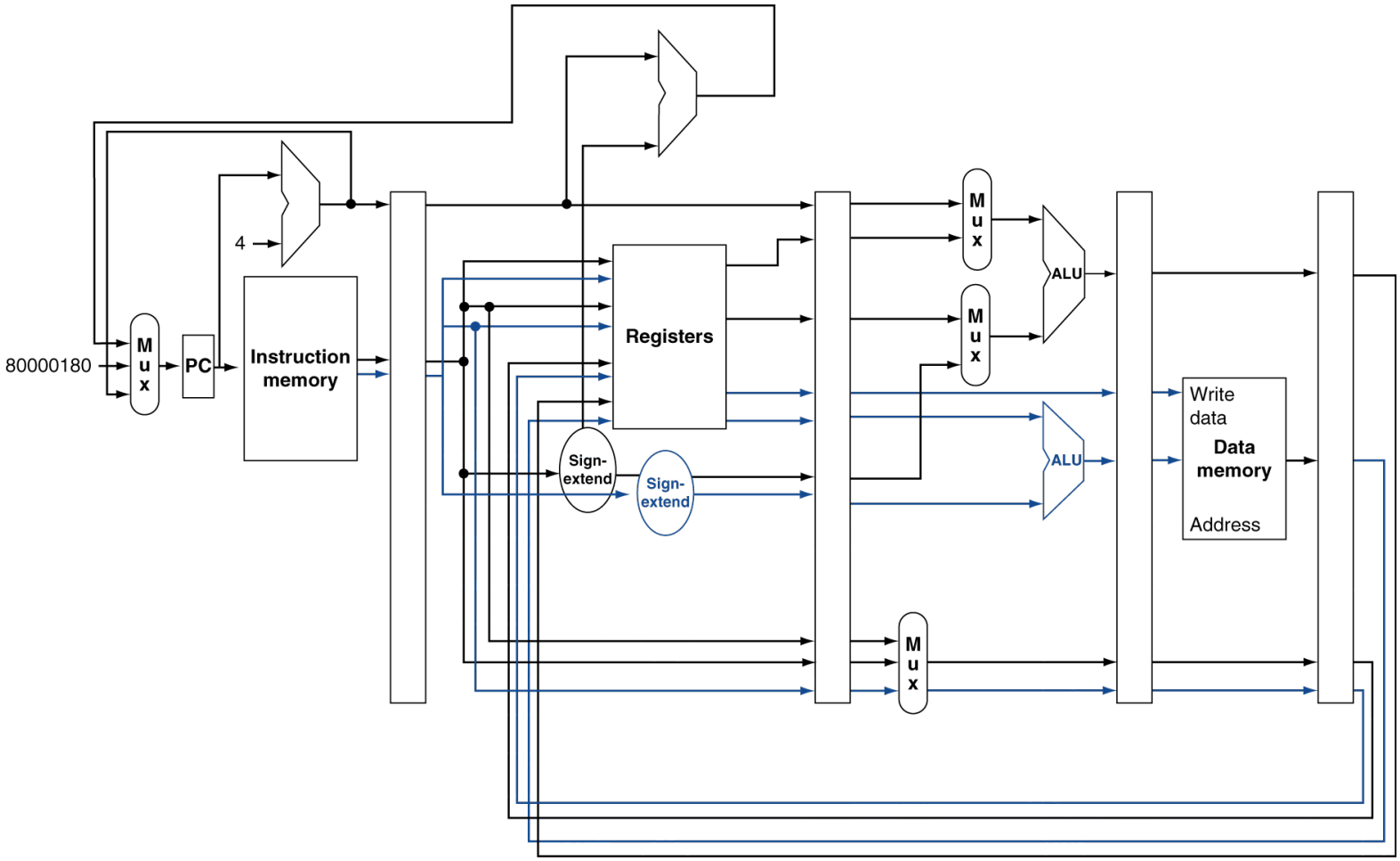
- Control unit deciding which instruction can execute together
- Decisions are made by hardware during execution
- Based on hazards (data, resources, etc.)
- Compiler may assist by reordering machine instructions

MIPS Static Multiple Issue

Consider a 2-issue MIPS processor

- Two instructions are fetched and executed together
- Requires 64-bit CPU-to-memory bus
- One can be an integer or branch instruction
- The other can be a load or store instruction
- Instructions are always issued in pairs
- Nop instruction replaces an unused slot within pair
- Required stalls holdup both instructions in a pair

Dual-issue MIPS



Extra resources: updated register file, unit to compute memory address, 2nd sign extension unit

Hazards in Dual-issue MIPS

- More instructions executing in parallel
- EX data hazard
  - Forwarding avoided stalls with single-issue
  - Now can't use ALU result in load/store in same packet
    - ```
      add  $t0, $s0, $s1
      load $s2, 0($t0)
      ```
    - Split into two packets, effectively a stall
- Load-use hazard
  - Still one cycle use latency, but now two instructions
- More aggressive scheduling required
      ```

Scheduling Example

```
Loop: lw   $t0, 0($s1)      # $t0=array element
      addu $t0, $t0, $s2    # add scalar in $s2
      sw   $t0, 0($s1)      # store result
      addi $s1, $s1,-4      # decrement pointer
      bne  $s1, $zero, Loop # branch $s1!=0
```

	ALU/branch	Load/store	cycle
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1,-4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

Loop Unrolling

- Replicate loop body to expose more parallelism
  - Reduces loop-control overhead
- Use different registers per replication
  - Called “register renaming”
  - Avoid loop-carried “anti-dependencies”
    - Store followed by a load of the same register
    - Aka “name dependence”
      - Reuse of a register name

Loop Unrolling Example

	ALU/branch	Load/store	cycle
Loop:	addi $s1, $s1,-16	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t4, $s2	sw $t1, 12($s1)	6
	nop	sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

- IPC = 14/8 = 1.75
  - Closer to 2, but at cost of registers and code size

Dynamic Multiple Issue

- "Superscalar" processors
- CPU selects instructions to execute each cycle
  - Avoiding structural and data hazards
- Avoids the need for compiler scheduling
  - Though compiler may still help
  - Code semantics ensured by the CPU

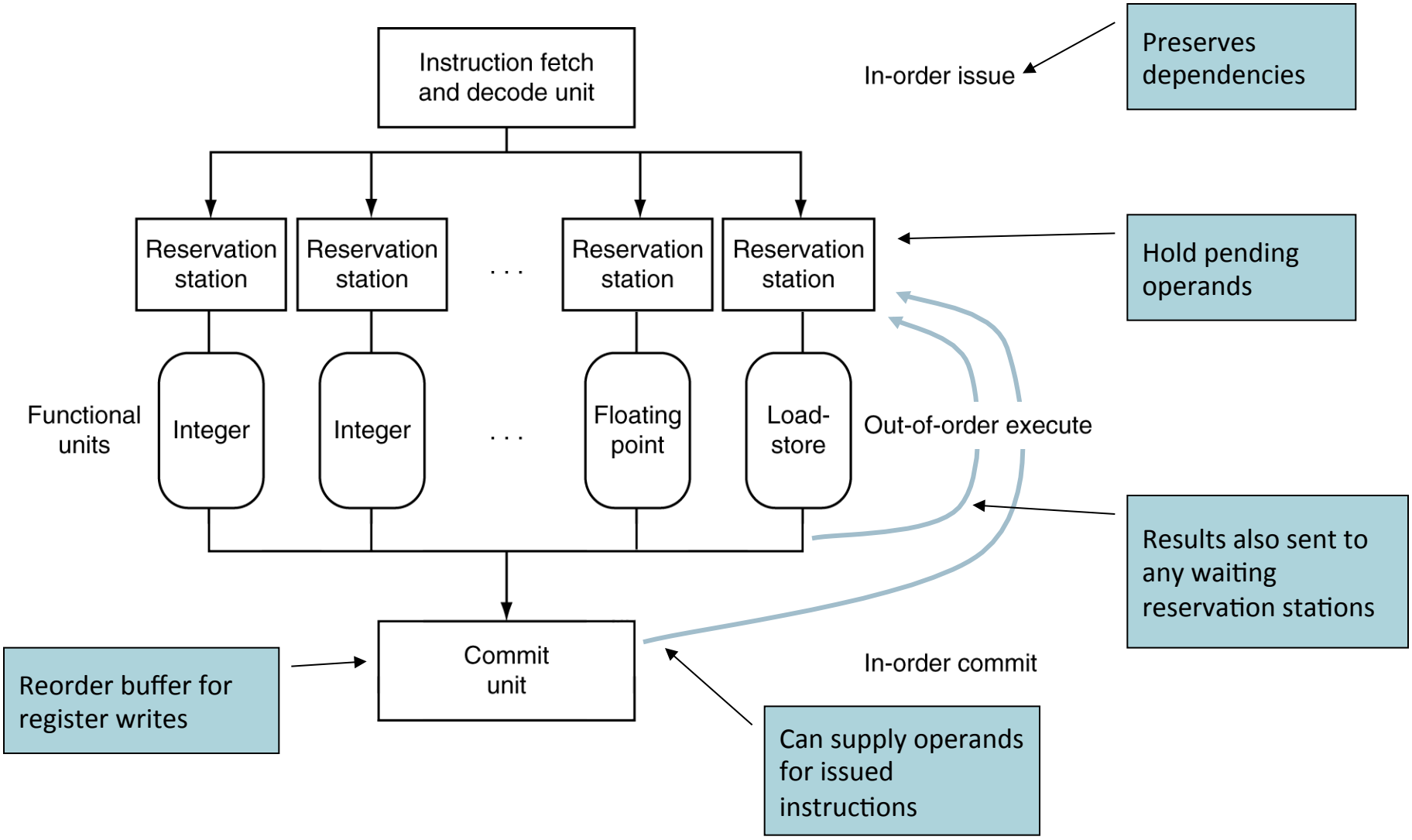
Dynamic Scheduling

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

```
lw    $t0, 20($s2)
addu  $t1, $t0, $t2
sub   $s4, $s4, $t3
slti  $t5, $s4, 20
```

  - Can start sub while addu is waiting for lw

Dynamic Scheduling



Register renaming

- Register renaming via reservation stations (RS) & ROB
  - ROB is reorder buffer
- On instruction issue to reservation station
  - copy available operand to reservation station
  - from register file or from ROB
    - Once done, register can be overwritten
  - When available, operands are provided to RS
    - Come from the function unit producing the result
    - Register update may not be required

Superscalar Operation