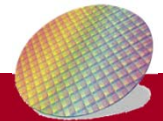# Outline

- A pipelined datapath

- Pipelined control

- Data hazards and forwarding

- Data hazards and stalls

- Branch (control) hazards

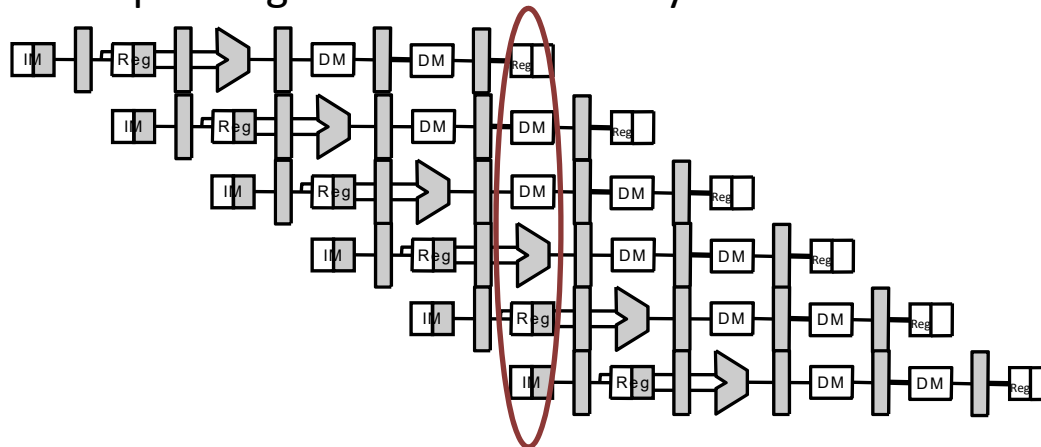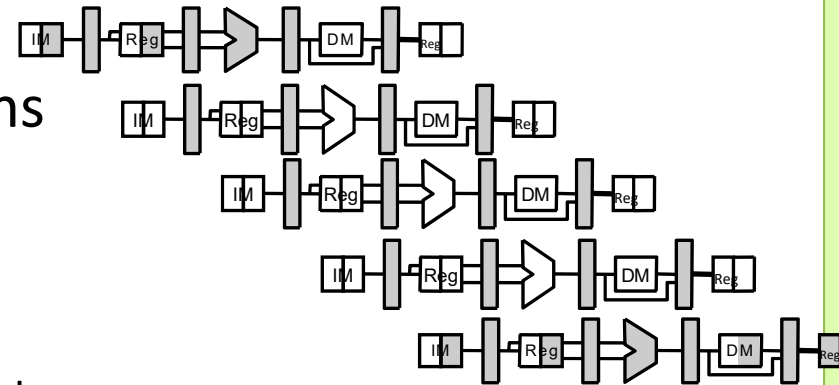- Instruction Level Parallelism (ILP)

Instruction-Level Parallelism (ILP)

- Pipelining: executing multiple instructions in parallel

- Current 5-stage pipelined processor
  - Issue one instruction at one time, max 5 instructions in the pipeline
  - Max speedup is 5

- Instruction-Level Parallelism (ILP):

  => the parallelism among instructions

- To increase ILP
  - Method 1: Deeper pipeline
   - Less work per stage ⇒ shorter clock cycle

6 stage, max 6 inst. In the pipeline

Instruction-Level Parallelism (ILP)

- Method 2: Multiple issue
  - Replicate pipeline stages ⇒ multiple pipelines
  - Start multiple instructions per clock cycle
  - CPI < 1, so use Instructions Per Cycle (IPC) not CPI
  - E.g., 4GHz 4-way multiple-issue
    - peak CPI = 0.25, peak IPC = 4
  - But dependencies reduce IPC

- Two types of multiple issue CPU
 - Compiler-based (static)
  - Hardware-based (dynamic)
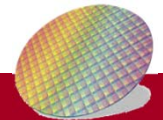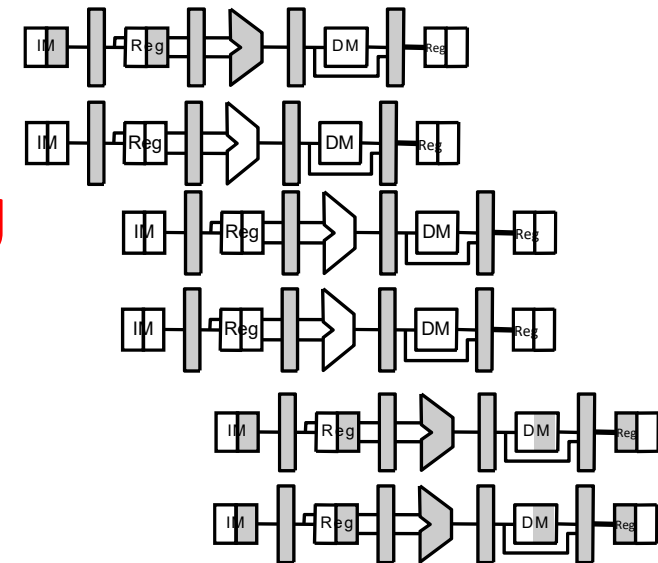
- ## Static multiple issue   (See next slide)

  – Use Compiler groups instructions to be issued together

  – Determined by pipeline resources required

    - E.g. lw or sw instructions can be scheduled with ALU/Branch Inst, but lw and sw can't be scheduled together

  – Compiler detects and avoids hazards during Compiling time

```
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
```

Question: What is the scheduling results for two-issue processor assuming lw/sw can only be scheduled with ALU/Branch instruction? (be sure to use least number of cycles)
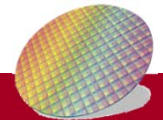
Static Multiple Issue
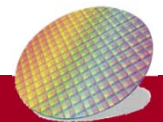
- ## Compiler groups instructions into "issue packets"

  Assume 2-issue processor is used, result of static multiple issues

	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

Note: addi and addu are reordered to reduce nop

Note: lw/sw can only be scheduled with ALU/Branch instruction

- ## Also called Very Long Instruction Word (VLIW)

  – Multiple parallel instructions are packed into an issue packet

Scheduling Static Multiple Issue

- Compiler must remove some/all hazards
  - Reorder instructions into issue packets
  - No dependencies exist within a packet
  - Possibly some dependencies between packets
    - Varies between ISAs; compiler must know!
  - Pad with nop if necessary

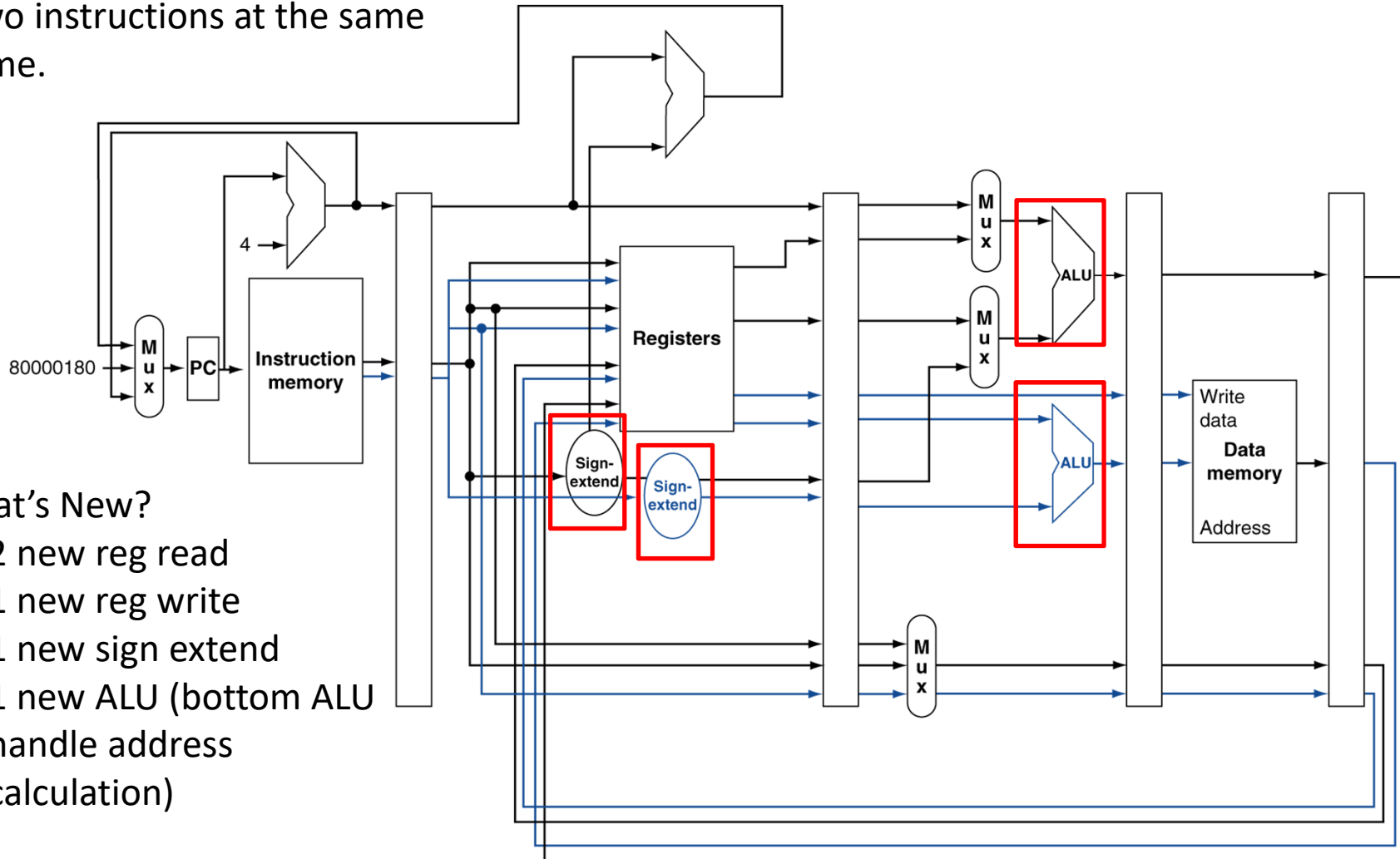|	ALU/branch	Load/store	cycle
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1, −4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

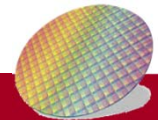What is its IPC?   IPC = 5/4 = 1.25 (c.f. peak IPC = 2)
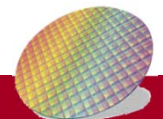
MIPS with Static Dual Issue

Units are duplicated to execute two instructions at the same time.

What's New?
- 2 new reg read
- 1 new reg write
- 1 new sign extend
- 1 new ALU (bottom ALU handle address calculation)

Loop Unrolling

- **Replicate loop body** to expose more parallelism
  - Reduces loop-control overhead (less branches)

- Use different **registers** per replication
  - Called "register renaming"
  - Avoid loop-carried "name-dependencies"
    - Store followed by a load of the same register

```
Loop:   lw    $t0, 0($s1)
        addu  $t0, $t0, $s2
        sw    $t0, 0($s1)
        addi  $s1, $s1, –4
        bne   $s1, $zero, Loop
```

Unrolling 4 times →

```
Loop:
lw   $t0, 0($s1)
addu $t0, $t0, $s2
sw   $t0, 16($s1)
lw   $t1, 12($s1)
addu $t1, $t1, $s2
sw   $t1, 12($s1)
lw   $t2, 8($s1)
addu $t2, $t2, $s2
sw   $t2, 8($s1)
lw   $t3, 4($s1)
addu $t3, $t4, $s2
sw   $t3, 4($s1)
addi $s1, $s1,–16
bne  $s1, $zero, Loop
```
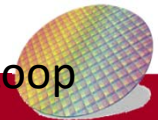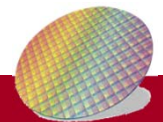
Loop Unrolling Example

|	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

- Dynamic multiple issue (aka. Superscaler processor)
  - CPU examines instruction and chooses instructions to issue each cycle. Hazards are resolved at runtime

- CPU decides whether to issue 0, 1, 2, … each cycle
  - Avoiding structural and data hazards
  - Can be in-order or out-of-order

- Hazard detected and instruction issues by hardware
  - Avoids the need for compiler scheduling

- Three major unit (See next slide)
  - Instruction fetch & Issue: fetch instruction and put into Reservation station
  - Execution: out of order execution in multiple function unit
  - Commit: write the results
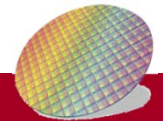
Dynamic Pipeline Scheduling
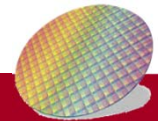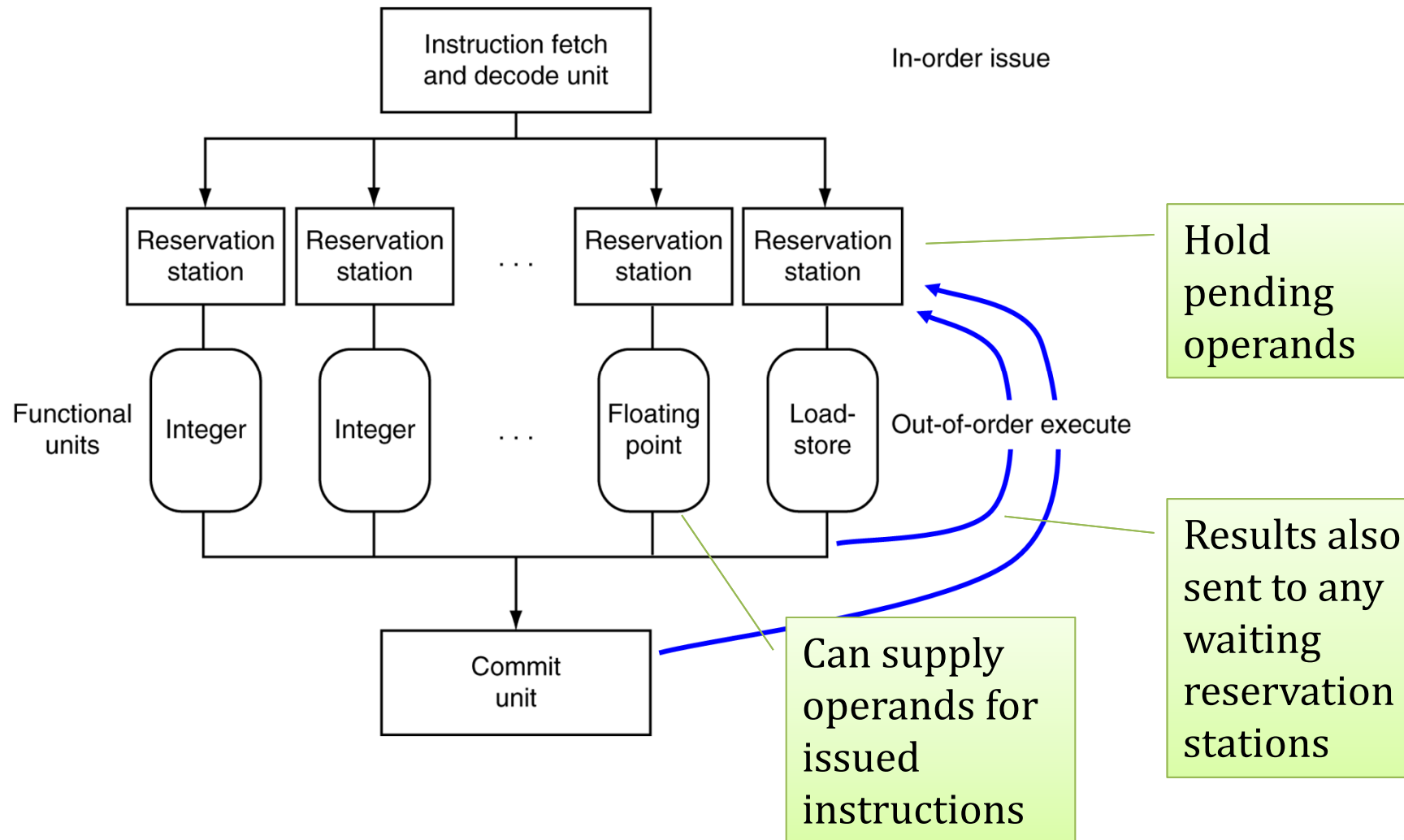
Dynamic Pipeline Scheduling

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

- Out-of-issue Example
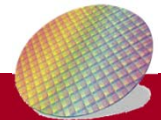
```
lw      $t0,  20($s2)
addu    $t1,  $t0,  $t2
sub     $s4,  $s4,  $t3
slti    $t5,  $s4,  20
```

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

    - Sub inst. is independent
   - Can start sub while addu is waiting for lw

|	Issue	Execution	Commit (Write back result
Dynamic Pipeline Scheduling	In-order	In-order or Out-of-order	Must be in-order

Why Dynamic Scheduling is needed

- Why not just let the compiler schedule code?

- Not all stalls are predictable
    - 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
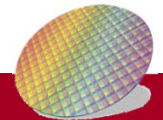
Does Multiple Issue Work?

- 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
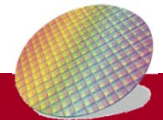
- Speculation can help if done well
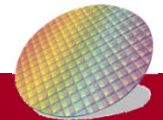
Power Efficiency

- Complexity of dynamic scheduling and speculations requires power

- Multiple simpler cores may be better

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Cores	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W
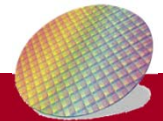
Concluding Remarks

- 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

Backup slides