Processor Performance

Experimental/simulation study of instruction streams to find ways to speed up execution is central to computer architecture.

Computer performance evaluation, however, can be controversial since much is at stake.

Users care about speedy results, a faster computer runs programs in less real time.

⇨ What counts

Physical run time of real programs

A **synthetic workload** is often used to simulate specific execution scenarios.

⇨ No one magic workload

Memory access and other non-CPU event delays also.

⇨ Program runtime components

Power consumption is an important aspect that must also be considered.

In a hyper-connected world, performance must also be balanced against **security** (esp. protecting microarchitectural state).

⇨ Performance metric

CPU time component of runtime (s)
= CPU cycles × cycle time (s)

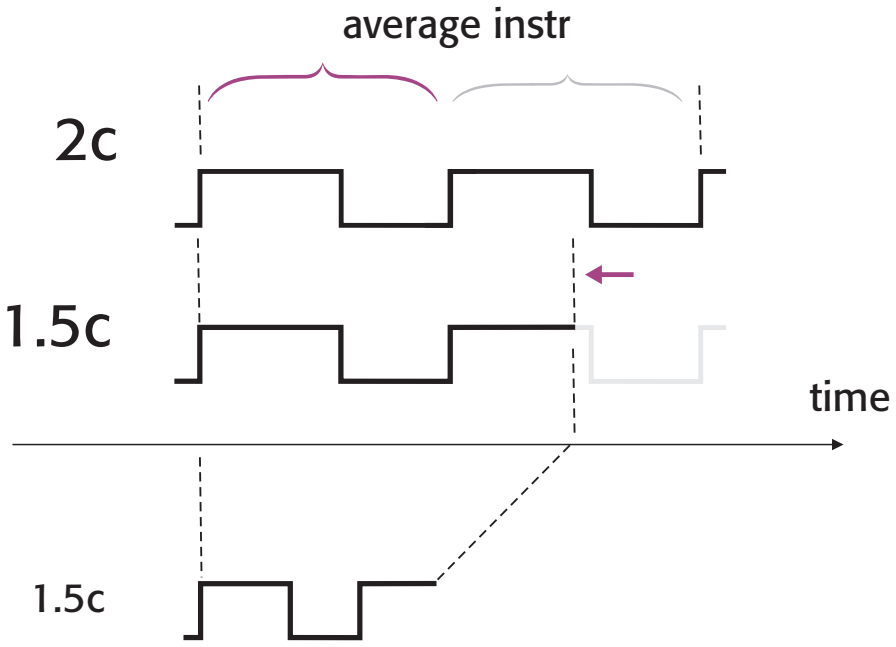
Processor Performance
Cycles/Instruction (CPI)

➪ Instruction latency

Quiz
How many cycles would it take to execute a 10 instruction sequence in each case?

A more efficient datapath, with lower average instr latency, may run the instrs in about 15c, 25% faster on average.

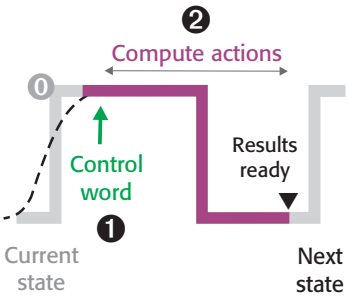
Instruction execution characteristics clearly affect program runtime but that's not the only clock factor.



Processor Performance
Cycle Length

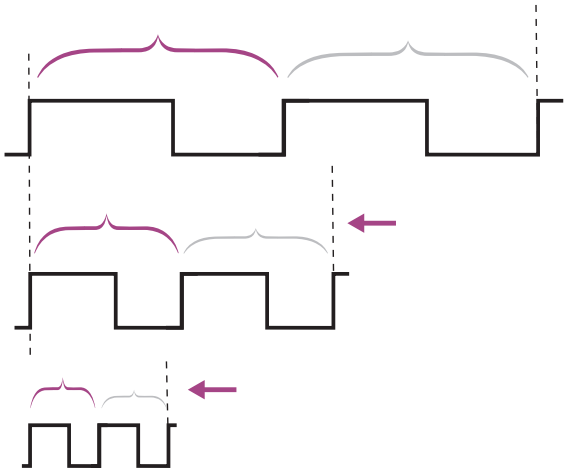
Quiz
How many cycles to execute the 10 instr sequence in each case?

Clearly, instruction <u>physical</u> execution time, for the same CPI, becomes smaller as cycles get shorter.



Quiz
Suggest at least 3 ways to shorten cycle. ***Answer last slide.***

Assume instrs execute at ~2c on average



Processor Performance
Instruction Count

Logically equivalent, perhaps by different compilers from the same high-level code, these programs are different from datapath viewpoint.

Exercise
What is the IC in each case?

Generally, shorter program should run faster if instructions have same CPI and cycle time.

More instructions increase fetch-decode overheads which include memory access.

Quiz
Is a processor design that produces longer programs necessarily slower? (See next slide). ***Hint: multiply is a higher pipeline latency instruction than add/sub/set/shift.***

```
slt   $t3,$s0,$zero
bne   $t3,$zero,Exit
slt   $t3,$s0,$t2
beq   $t3,$zero,Exit
add   $t1,$s0,$s0
add   $t1,$t1,$t1
add   $t1,$t1,$t4
lw    $t0,0($t1)
jr    $t0
```

```
slt   $t3,$s0,$zero
bne   $t3,$zero,Exit
slt   $t3,$s0,$t2
beq   $t3,$zero,Exit
sll   $t1,$s0,2
add   $t1,$t1,$t4
lw    $t0,0($t1)
jr    $t0
```

```
blt*  $s0,$zero,Exit
bge*  $s0,$t2,Exit
addi  $t1,$zero,4
mul*  $t1,$t1,$s0
add   $t1,$t1,$t4
lw    $t0,0($t1)
jr    $t0
```

* Core RISC-V machine instructions (pseudo-instructions in the original MIPS R2000 provided by the assembler).

Processor Performance
CPU Time Components

Quiz
Which aspect of processor design has a major effect on each factor of performance?

⇨ Performance factors (check units)

① Instruction count (IC)

② Cycles per instr (CPI), on average

③ Clock cycle time

IC due to ISA (not poor compilation); e.g., CISC tends to yield shorter programs with complex execution profiles that may be slower when timed physically.

A detailed CPI is obtained from averaging over families of instrs that share exec profiles (due to shared datapath segments) weighted by their frequencies.

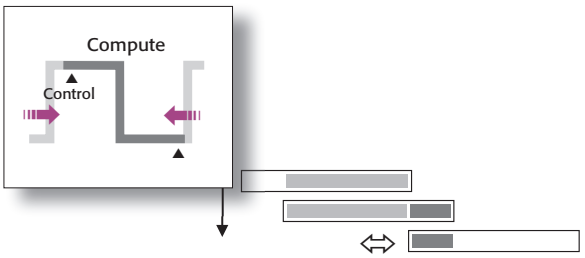
⇨ CPU performance equation

$$\text{CPU time (s)} = \text{IC} \times \text{CPI} \times \text{cycle time}$$

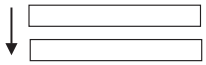
Clock cycles are a poor indicator of runtime alone, valid only when the other two factors are the same (i.e., misleading to compare performance based on cycles when instrs have different execution characteristics).

More Performance

➪ Superpipeline



Pattern A



Pattern B

Make stage smaller or otherwise just shorten cycle (higher clock rates); more stages have higher (ideal) speed-up ceiling but how far can we go?

➪ More pipeline stages

Deeper pipelines maximize Pattern A

*31+ stages in Pentium 4/Rev. E (Prescott, 2004), not counting front-end side fetch-decode stages apparently.

More stages increase performance, too many have terrible misprediction and power costs.

Intel basically abandoned this approach in 2006 with move from Pentium 4 (20 stages*) to the Core architecture.

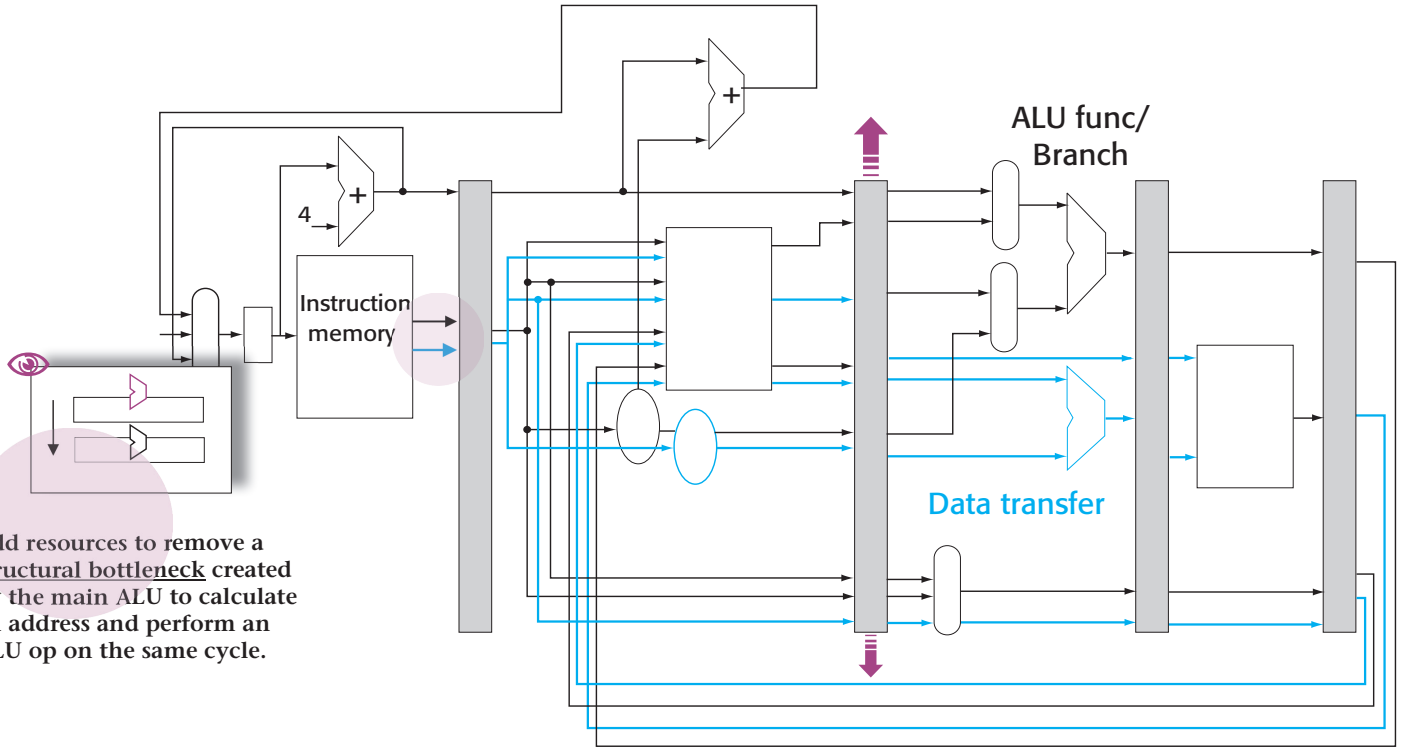
➪ More ILP: multiple issue

Start multiple instructions every cycle (hopefully) to exploit Pattern B

CPI<1 (IPC>1) *superscalar performance* (where *scalar* CPI ≥ 1); it implies more than one pipeline or execution pathway.

More Performance
Multiple Issue

➪ Issue slots



Add resources to remove a structural bottleneck created by the main ALU to calculate an address and perform an ALU op on the same cycle.

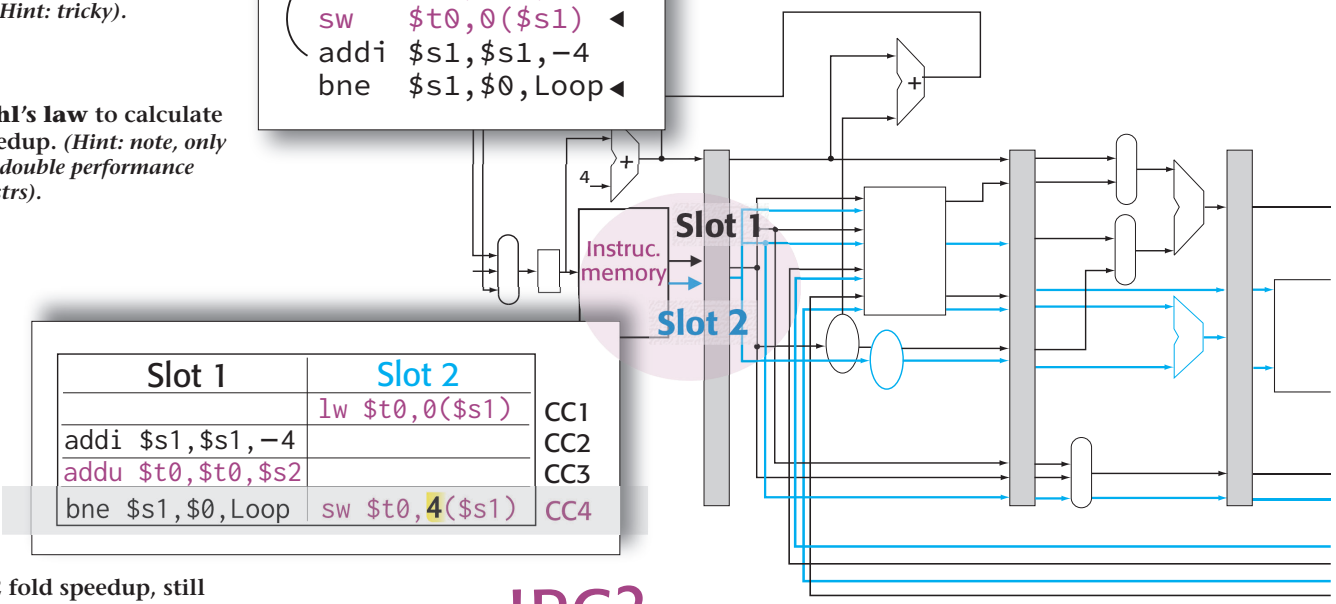
Flipping the Metric

➪ Issue packet

Quiz
Identify data hazards (Hint: 3). Compare CPI to a hazard-free execution on the **scalar** original MIPS. *(Hint: tricky)*.

Exercise
Use **Amdahl's law** to calculate overall speedup. *(Hint: note, only managed to double performance for 2 of 5 instrs)*.

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



Slot 1	Slot 2	
	lw $t0,0($s1)	CC1
addi $s1,$s1,-4		CC2
addu $t0,$t0,$s2		CC3
bne $s1,$0,Loop	sw $t0,4($s1)	CC4

Not quite 2 fold speedup, still significantly better than one op/cycle *most* of the time (which is what close to 1 CPI means).

IPC?

Compare to near ideal scalar

Speeding the Stream

⇨ Superscalar

In all cases hazards must be handled to pipeline execution effectively.

⇨ **Multiple issue** **more than 1 slot to fill**

Multi-instr/opcode scheduling

Dependance analysis performed by compiler on static code.

⇨ **Static multiple issue**

Issue long instruction words

Dependance analysis performed by hardware at run time.

⇨ **Dynamic multiple issue**

Superscalar (term) also refers to older form of dynamic issue with out-of-order execution; also class of architectures that avoid recompiling older program binaries.

Superscalar is more powerful when combined with other execution tricks.

- ✎ Dynamic scheduling (more later)
- ✎ Hardware-based issue packet
- ✎ Out-of-order issue

Static Multiple Issue

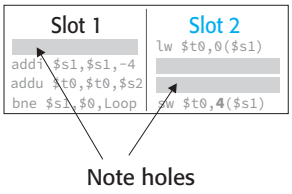
➪ **VLIW**

Exercise
Discuss effects of instr dependencies on VLIW coding deficiency; suggest some mitigation strategies.

Static = concerning non running instructions (in static state); not during run time; as loaded in memory.

➪ Compiler-based packaging

Schedule sets of ops + handle hazards



➪ Issue packet

Predetermined sets of ops per cycle

Exercise
Compare typical CISC, RISC, and VLIW instructions in terms of info content, bit length, and fetch-decode overhead.

➪ Very long instruction word

“ One long [not complex] instruction with multiple [MIPS-like] operations ”

Register Renaming

- Antidependence
- Loop unrolling

Regular loops best unrolled to avoid (excessive) decision stalls.

Common pattern

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

Update different memory operands using the same register needlessly.

Quiz
How could the WAR (**write-after-read**) condition on `$t1` affect executing the `sw–lw` sequence?

```
lw  $t0, 0($t0)
add $t0, $t1,$s0
sw  $t0, 0($t0)
lw  $t0, 4($t0)
add $t0, $t1,$s1
sw  $t0, 4($t0)
```

read — write — Data flow?

```
lw  $t0, 0($t0)
add $t0, $t1,$s0
sw  $t0, 0($t0)
lw  $t1, 4($t0)
add $t1, $t2,$s1
sw  $t1, 4($t0)
```

Technique can be implemented in either software (as shown here) or in hardware (next).

- Data (value) dependence
- Name dependence (*anti-dependence*)

Dynamic Multiple Issue

Dynamic means at run-time, decisions are made by control during execution, they vary between runs.

ISA independence.

⇨ Dynamic issue in general

- ✎ Hardware-based reordering
- ✎ Transparent (hidden) to programs
- ✎ Compiler assist useful, not expected

Dynamic multiple issue relies on older dynamic issue and **dynamic pipeline** scheduling techniques.

A dynamic pipeline picks instruction(s) to execute in a given cycle, reordering and effectively renaming registers to avoid stalls.

⇨ Dynamic pipeline

- ✎ [Re]scheduling by hardware
- ✎ On-the-fly = during execution
- ✎ Out-of-order execution

A Basic Dynamic Pipeline

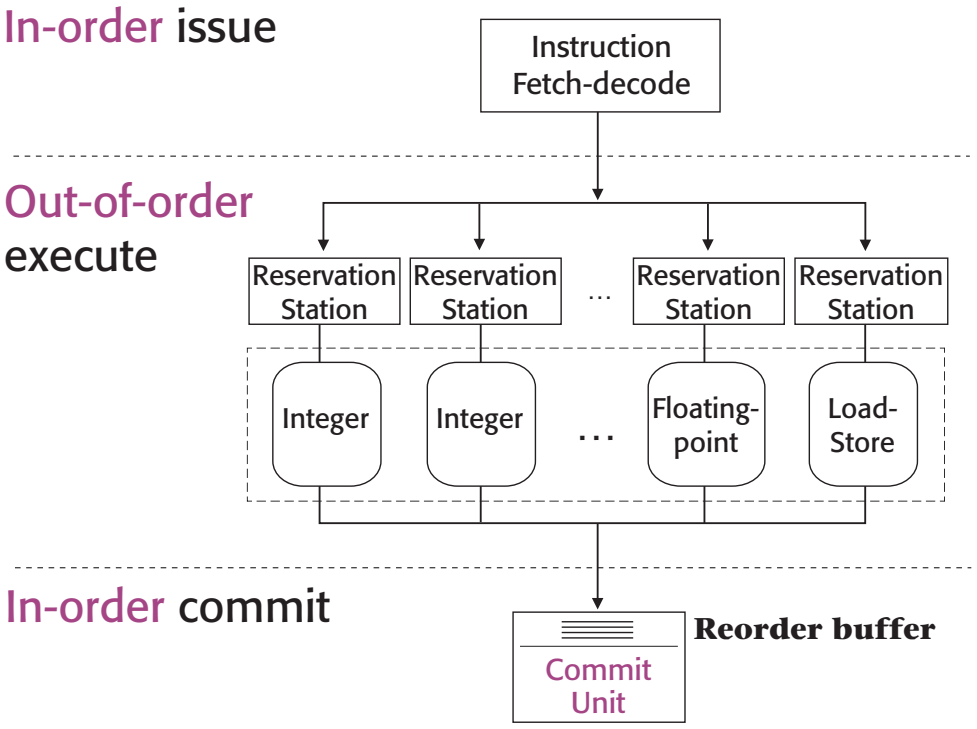
(Basic in-order) An instruction is not issued if a dependence on a previously scheduled one was not eliminated or the forwarding hardware can't bypass it (i.e., issue pause and pipeline stalls, even if a later close instruction was independent).

For multiple issue, execution and issue are essentially decoupled.

Out-of-order execution implies out-of-order completion, a queued buffer can guarantee delivery of correct results.

Dynamic scheduling is transparent to software, it can be improved by upgrading processor control algorithms and func units for a relatively cheap software performance increase.

Exercise
Compare: IBM's **Tomasulo algorithm**, CDC-6600 scoreboard, and the basic scenario (slide + fig).



Reproduction (concept P&H 1998-2012)

Dynamic Issue Pipeline Operation

Dynamic pipelines operate outside **architectural state** (programmer/ISA registers and memory).

Reservation stations decouple operand values from architectural registers (register renaming effectively).

Reorder buffer provides built-in forwarding since results may be used before committing to the final register and memory destinations.

2 main things to know

- Op executes as soon as operands and a functional unit are ready
- Operand cases:
  - Ready: available from reg. file or reorder buffer
  - Not ready: not produced yet

More Performance Consequences

Historically, processors were built for speed mainly; old-school speed came at a significant power cost (at times too steep to accept).

➪ **Speed-power tradeoff**

Born originally out of a classic need-for-speed thinking.

➪ **Speculative execution**

Two architectural flaws, named *Spectre* and *Meltdown*, came to attention in 2017 (publicly early 2018).

➪ **The Spectre design "flaw"**

Later a growing family (headache) of variants; they even have cute icons.

➪ **Security design concerns**

➪ **A new tradeoff?**

A new line of attacks emerged in 2022, *Retbleed*, which mitigates the *retpoline* mitigation!

https://arstechnica.com/information-technology/2022/07/intel-and-amd-cpus-vulnerable-to-a-new-speculative-execution-attack/

Conclusions

Involving the decoding component/ burdens of the front-end.

➪ **Compiler-hardware interplay**

A front-facing one associated with an input instruction stream, and another one focused on speed with an internal stream.

➪ **Processor dual personality**

8086/8088 (1978) binaries (.exe) from 1980s may still be run as-is on the latest Intel core processors (except those dependant on obsolete PC parts).

➪ **Superscalar arch. advantage**

Binary compatibility; hide machine details, hardware develops independently

An understated way of saying don't take them numbers too seriously, especially when realistic workloads and run environments are factored in.

➪ **Machine performance**

Theoretical peak instruction throughput in hardware often not sustainable

Conclusions
More Performance

Processor datapaths contribute relatively few cycles to actual average instruction latencies.

Realistic CPI/IPC is not easily predictable and must be measured carefully after we factor in interaction with memory.

⇨ **Historically slow memory**

May add tens to 1000s+ of cycles unpredictably

Best performance trick by far is access to at least *some* memory that can keep up with short and fast processor cycles.

⇨ **Fast and furious (next)**

To consider next: extra resources are cheap but the software side of MIMD parallel processing is not easy + SIMD workloads highly significant + single processor/thread performance still important to some workloads (as of 2022).

⇨ **What about multiprocessing?**

Multicores, processor arrays, later

Essay Assignment

First technical essay reminder

Pick a topic 1–5

Expectations

Contribute a small essay in discussion group (check topic for rules and topic details)

Slide 3 Answer
Reduce work per stage (simplify instr or break it further), simplify control (output control word sooner), make devices switch faster or reduce distances (shrink circuits), or make functional unit response times smaller (perhaps better logic). Some changes are architectural (such as reorganizing or reimplementing function/logic), and others are technological (e.g., better transistors).