Processor Performance

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

Studying instr streams, both experimentally and via simulations, to figure out ways to speed up execution is central to computer architecture.

➪ What counts

Physical run time of real programs

Synthetic workloads are often used to simulate specific execution scenarios.

➪ No one magic workload

➪ Program run time

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

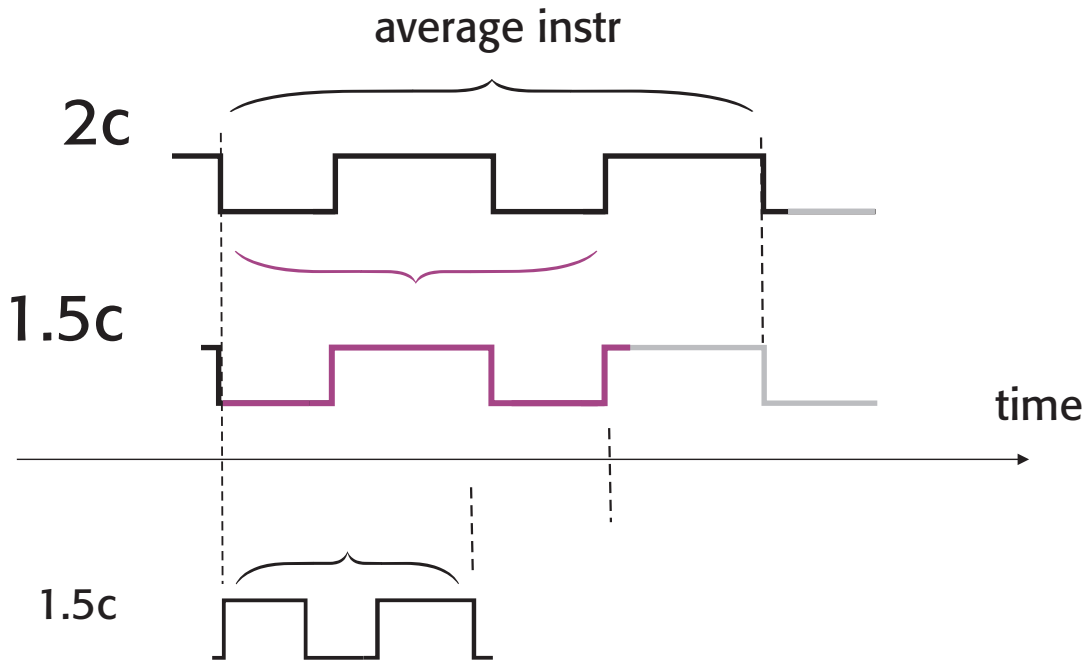
➪ Performance metric

CPU time component of runtime (s)

= CPU cycles × cycle time

Processor Performance
Cycles/Instruction (CPI)

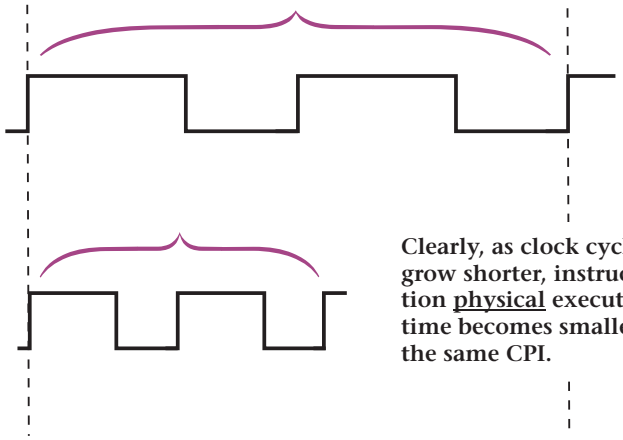
Quiz
How many cycles would it take to execute 10 instr sequence in each case?



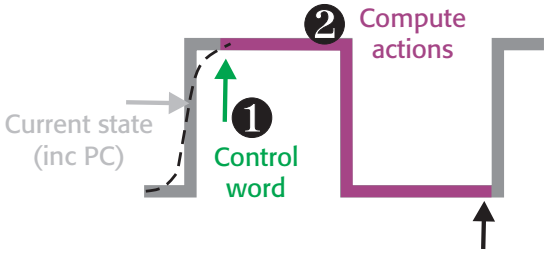
Instruction execution characteristics clearly affect program runtime but that's not the only "clock factor" involved.

Processor Performance
Cycle Length

instr exec 2c on average



Clearly, as clock cycles grow shorter, instruction physical execution time becomes smaller at the same CPI.



Quiz
Suggest at least 3 ways to make the cycle shorter? Answer last slide (Essay Assignment.)

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?

```
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)
srl   $t0,$t0,26
```

```
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)
srl   $t0,$t0,26
```

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

If instructions take the same number of cycles to execute, and the cycle time is the same, then a shorter program is faster.

Processor Performance
CPU Time Components

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

A realistic CPI may be based on ave CPI per family of instr which share exec characteristics (common datapath segments), weighted by frequencies (workload dependent).

⇨ CPU factors (check units)

① Instruction count (IC)

② Cycles per instr (CPI), on average

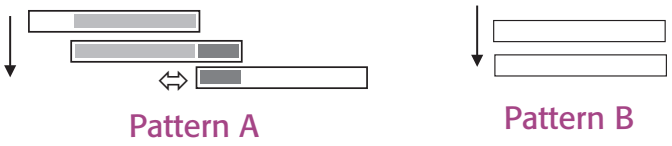
③ Clock cycle time

⇨ CPU performance equation

CPU time (s) = IC × CPI × cycle time

More Performance

➪ Superpipeline



More stages help shorten clock cycle by breaking up work into smaller pieces, but can have a terrible mis-prediction and power consumption costs. Intel basically abandoned this approach in 2006 with its Core architecture

➪ More pipeline stages

Deeper pipelines maximize Pattern A

➪ Multiple issue

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

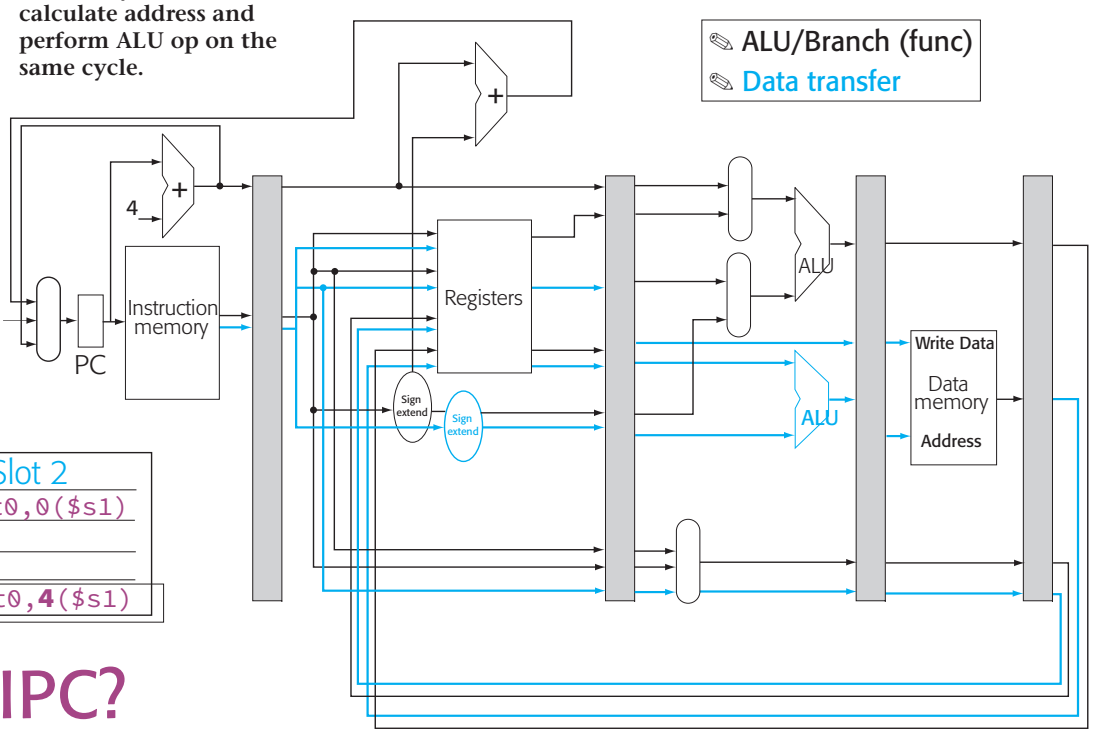
Flipping the Metric

➪ Issue packet

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

Quiz
Name the data hazards (Hint: 3).

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



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

Not quite 2 fold speedup, still significantly better than one op/cycle some of the time.

IPC?

Speeding the Stream

In both cases instr execution is pipelined therefore hazards must be handled.

- **Static multiple issue**
  - Multi-instr/op code scheduling
  - Long instruction word
- **Dynamic multiple issue**
  - Superscalar
  - Dynamic pipeline scheduling
  - Out-of-order execution

Static Multiple Issue

Static = outside run-time, decisions are made by compiler

➪ Compiler-assisted packaging

Schedule sets of ops + handle hazards

Slot 1	Slot 2
	`lw $t0,0($s1)`
`addi $s1,$s1,-4`	
`addu $t0,$t0,$s2`	
`bne $s1,$0,Loop`	`sw $t0,4($s1)`

➪ Issue packet

Predetermined sets of ops per cycle

Exercise
Compare typical CISC, RISC, and VLIW instructions in terms of info content, bit length, and 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 decision stalls.

Common pattern

Update different memory operands using the same register needlessly.

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

Data flow?

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

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

- Data (value) dependence
- Name dependence

Dynamic Multiple Issue

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

➪ Dynamic issue: superscalar

- ✎ Hardware-based issue packet
- ✎ Transparent (hidden) to programs
- ✎ Compiler assist useful, not required

Superscalar is useful but limited on its own, and can be made powerful when combined with other execution tricks.

➪ Dynamic pipeline scheduling

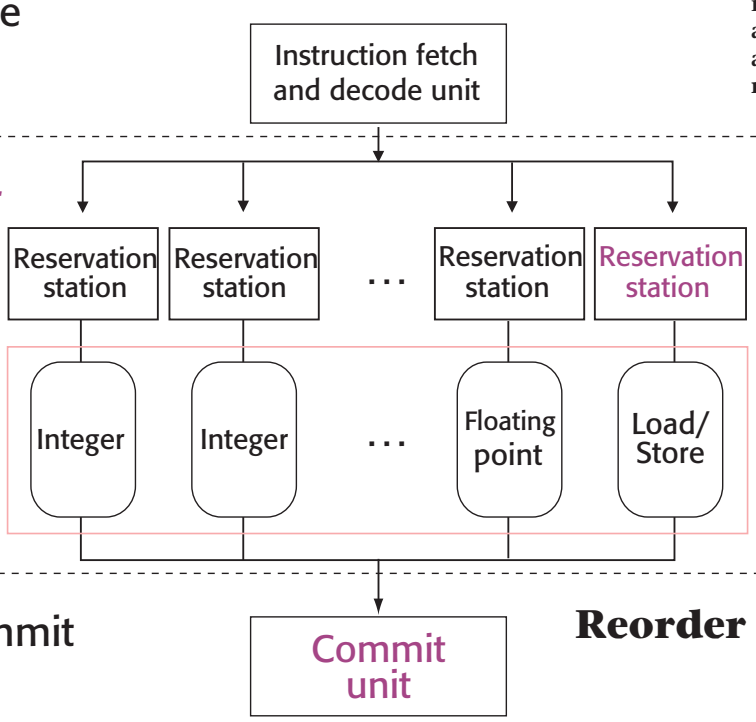
- ✎ Hardware-based reordering
- ✎ On-the-fly = during execution
- ✎ Out-of-order execution

Dynamic Multiple Issue
Dynamic Issue Pipeline

1 In-order issue

2 Out-of-order execute

3 In-order commit



Dynamic issue pipeline chooses which group of instructions to execute in a given cycle, **reordering** and effectively **renaming registers** to avoid stalls.

Dynamic scheduling is transparent to software, it can be improved by upgrading processor-based control algorithms resulting in relatively cheap software performance increase.

Reorder buffer

Dynamic Issue Pipeline Operation

2 main things to know

Operand values can be used before they are committed to their final register and memory destinations (forwarding effectively).

- 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

Reorder buffer provides built-in forwarding.

Processor Design
Consequences

Historically processors were built for speed primarily; soon speed came at significant costs of power (at times too steep to accept).

- ⇨ **Speed-power tradeoff**
- ⇨ **Speculative execution**

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

- ⇨ **The Spectre design "flaw"**
- ⇨ **Security design concerns?**

Conclusions

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

➪ Compiler-hardware interplay

Extra resources are cheap but MIMD parallel processing is not easy + single processor/thread performance is still important to some workloads.

➪ Superscalar advantage

- ✎ Binary compatibility, hide exec details
- ✎ Hardware can develop independently

An understated way of saying don't take these numbers too seriously, especially when workload variability is factored in.

➪ Program performance

Hardware theoretical peak instruction throughput often not sustainable

Essay Assignment

Pick a modern processor

Review architectural features

Expectations

Contribute small essay in discussion group (check topic for specs)

Slide 3 Answer
Reduce actions per stage (simplify instr or break it further), simplify control (o/p word sooner), make devices switch faster/reduce distances (shrink), or make func units response times smaller (better logic).