Components of CPU Performance and Performance Equation

Why is my computer fast (or slow)?

Would it help to improve ____?

CPU performance equation is one way to start answering these questions.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

02-2

CPU Performance Decomposed into Three Components:

- Clock Frequency (ϕ)
  Determined by technology and influenced by organization.
- Clocks per Instruction (CPI)
  Determined by ISA, microarchitecture, compiler, and program.
- Instruction Count (IC)
  Determined by program, compiler, and ISA.

These combined to form *CPU Performance Equation*

$$t_{\mathrm{T}} = \frac{1}{\phi} \times \mathrm{CPI} \times \mathrm{IC},$$

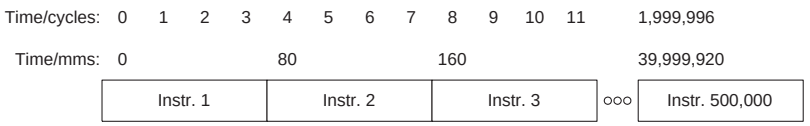
where t_{T} denotes the execution time.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

02-3

CPU Performance: Simple System

Execution in program order . . .
. . . one at a time.



$\mathrm{IC} = 500,000$; $\phi = 50\,\mathrm{kHz}$; $\mathrm{CPI} = 4$.

Execution time: IC × CPI. × clock period.

Here (**and only here)** CPI is number of cycles for each instruction.

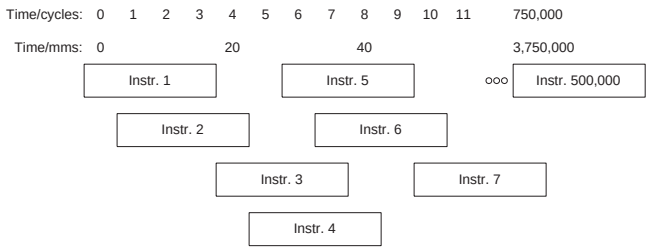
EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

Execution: Pipelined, In Order

02-4

To Run Faster: Overlap Instructions *(Pipelined Execution)*

Result must be same as one-at-a-time execution . . .
. . . not too difficult to achieve.



$\mathrm{IC} = 500,000$; $\phi = 200\,\mathrm{kHz}$; $\mathrm{CPI} = \frac{750000}{500000} = 1.5$.

Execution time at best: IC × clock period . . .
. . . assuming 1 cycle to start each instruction and . . .
. . . instruction can start each cycle. (Slower in illustration.)

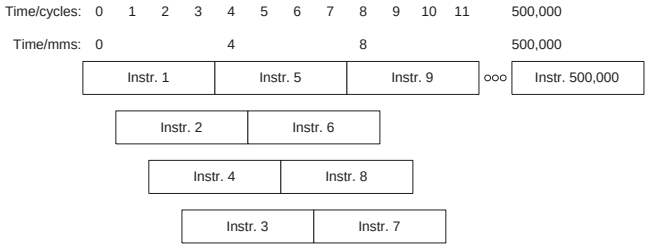
EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

Execution: Pipelined, Ideal Out of Order

02-5

To Run Even Faster: Overlap Instructions and Start Out of Order

Sometimes skip an instruction and execute it later.



IC $= 500,000$; $\phi = 200\,\text{kHz}$; CPI $= 1$.

Execution time at best: IC $\times$ clock period ...
... assuming 1 cycle to start each instruction ...
... instruction can start each cycle.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

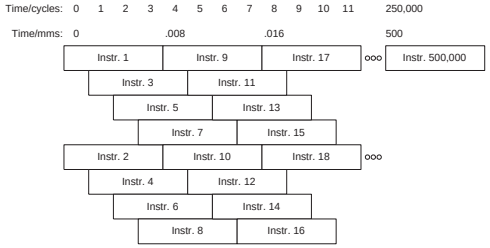
02-5

Execution: Pipelined, Ideal Out of Order, Superscalar

02-6

To Run Fastest[1]: Overlap, Out-of-Order, Start n per Tick (*n-Way Superscalar*).

Requires about n times as much hardware. (Below, $n = 2$.)



IC $= 500,000$; $\phi = 500\,\text{MHz}$; CPI $= \frac{1}{2}$.

Execution time at best: $\frac{1}{n}\times$ IC $\times$ clock period ...
... assuming 1 cycle to start each instruction instruction can start each cycle.

[1] Using a conventional serial instruction set architecture.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

02-6

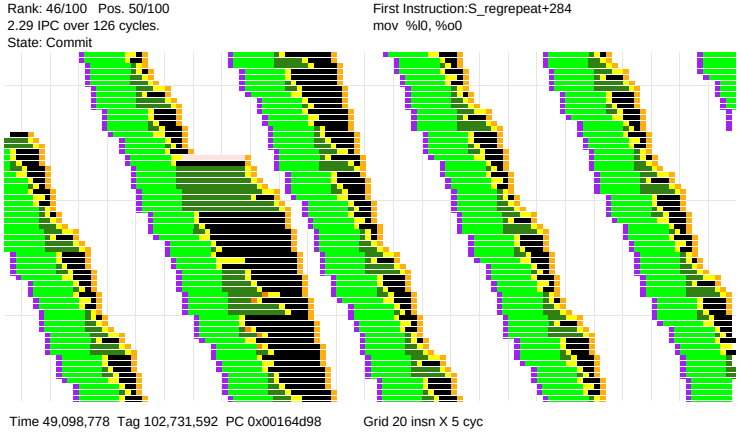
Execution: Pipelined, Out of Order, Superscalar

02-7

Data from a real program, perl. CPI is 0.44.

Processor can start four instructions per cycle.

Colors show the steps in processing an instruction, yellow is execution.



EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

02-7

Component of CPU Performance: Instruction Count

02-8

Given a program there are two ways instructions could be tallied:

Static Instruction Count:
The number of instructions making up the program.

Dynamic Instruction Count (IC):
The number of instructions executed in a run of the program.

For estimating performance, dynamic instruction count is used.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

02-8

Instruction Counts

02-9

Example, assembler program that computes $a = \sum_{i=0}^{9} i$.

Written in Simplescalar assembler.

```
IC
 1      move    r5, r0      ! r0 is always zero.
 1      move    r3, r0
       L23:                ! Branch label.
10      addu    r5, r5, r3  ! Add unsigned.
10      addu    r3, r3, 1
10      slt     r2, r3, 10  ! r2 = r3 < 10
10      bne     r2, r0, L23 ! Branch to L23 if r2 not equal 0.
```

Static count: 6 (number of instructions).

Dynamic count: 42.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

02-9

Component of CPU Performance: Clock Frequency

02-10

CPUs implemented using synchronous clocked logic.

Typical Clock Cycle

- When clock switches from low to high work starts.
- While clock is high work proceeds.
- When clock goes from high to low work should be complete.

Clock frequency determined by *critical path*.

Critical Path:
Logic doing most time consuming work (in a cycle).

If clock frequency is too high work will not be completed ...
... and so system will not perform properly.

For high clock frequencies, keep critical paths short.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

02-10

Component of CPU Performance: CPI

02-11

Cycles (clocks) per Instruction (CPI)

Oversimplified definition: *CPI:*
Average number of cycles needed to execute an instruction.

Better definition: *CPI:*
Number of cycles to execute some code divided by number of instructions.

Difference:

Interested in rate at which instructions executed in program ...

... not time time for any one instruction.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

02-11

Review of CPU Performance Equation

02-12

$$\boxed{t_{\mathrm{T}} = \frac{1}{\phi} \times \mathrm{CPI} \times \mathrm{IC}},$$

where t_{T} denotes the execution time.

- Clock Frequency (ϕ)
  Determined by technology and influenced by organization.
- Clocks per Instruction (CPI)
  Determined by organization and instruction mix.
- Instruction Count (IC)
  Determined by program and ISA.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

02-12

Interaction of Execution Time Components

Tradeoffs between Clock Frequency, CPI, and Instruction Count

Increasing Clock Frequency ...

... reduces the work that can be done in a clock cycle ...
... and possibly limiting instruction overlap.

Reducing IC (by adding "powerful" instructions to ISA) ...

... may force implementors to increase CPI or lower clock frequency.

Balancing these is an important skill in computer design.

Since the ISA is usually fixed, IC is less of a factor.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

Example: Trading off Execution Time Components

Company X is considering two clock frequencies for its next processor, 500 MHz *or* 400 MHz. *A* 500 MHz *implementation would execute instructions at 1.7 CPI, the 400 MHz part at 1.1 CPI. Which would be faster?.*

Find time to execute 1 instruction.

500 MHz execution time: $\frac{1}{500\times10^6} \times 1.7 \times 1 = 3.4\,\mu$s

400 MHz execution time: $\frac{1}{400\times10^6} \times 1.1 \times 1 = 2.75\,\mu$s.

The lower clock rate would nevertheless take less time.

Perhaps because at 500 MHz too much work had to be split into multiple cycles.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

IC v. CPI Tradeoffs

Assumption

IC is based on output of a good compiler.

Compiler is tuned for a particular implementation.

Two Cases

1. Same ISA, different implementation.
2. Different ISA, (and of course) different implementation.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

IC v. CPI Tradeoffs, continued.

Case 1: Same ISA, different implementation.

Newer implementation may have lower CPI on existing code ...
... but even better performance attainable by recompiling ...
... which may *increase* CPI.

Compiler writer selects instructions based on performance of implementation.

EE 4720 Lecture Transparency. Formatted 13:04, 24 January 2003 from lsli02.

Consider two implementations:

Implementation A: `add` CPI 1 cycle, `mul` CPI 5 cycles.

Implementation B: `add` CPI 1 cycle, `mul` CPI 2 cycles.

```
! Call original value of r1, x.  Code computes 6x.

! Code For Implementation A
add  r1, r1, r1 ! r1 = 2x
add  r2, r1, r1 ! r2 = 4x
add  r1, r1, r2 ! r1 = 6x

! Code For Implementation B.
mul  r1, r1, 6  ! r1 = 6x.
```

Implementation A: IC = 3, CPI = 1 (Computing CPI will be covered later.)

Implementation B: IC = 1, CPI = 2.

Implementation B is faster despite higher CPI.

Code compiled for B will run slowly on A.

IC v. CPI Tradeoffs, continued.

Case 2: Different ISA, (and of course) different implementation.

Major tradeoffs in complexity and speed.

Consider two implementations:

Implementation A: CPI: `load`, 2; `add` and `store`, 1.

Implementation B: CPI: `add` (doing load and store), 4.

```
! Code for implementation A.
load  r1, [r2]    ! Load r1 with data at address in r2.
add   r3, r1, r4  ! r3 = r1 + r4
store [r2], r3    ! Store r3 at address in r2.

! Code for implementation B.
add [r2], r4, [r2]
```

Execution time same.

Implementation A: IC = 3, CPI = $\frac{4}{3}$.

Implementation B: IC = 1, CPI = 4.

Technological Change

Golden Handcuffs:
The need to maintain compatibility in a successful product line.

Famously, Intel's IA-32. (Popularly referred to as 80x86.)

The ISA is the handcuffs...
... and technological change brings the desire to move your arms.

Technological Change and Computer Designer

Technology determines "raw materials" for designer.

Raw material: number of gates and their speed.

ISA lifetime can be decades.

Raw materials greatly change over this time.

So, design ISA for now and future.

How technological advancement affects processor.

Logic Speed, Clock Rate
No changes to organization or ISA.

Number of Transistors Available for Logic
Changes to organization and possible changes to ISA.

Memory Size
Change ISA to use larger address space.
Can use ISA having larger instruction codings.

Memory Speed Compared to Processor Speed
Include more sophisticated caching in organization.

Benchmarks

Benchmark:
Program used to evaluate performance.

Uses

- Guide computer design.
- Guide purchasing decisions.
- Marketing tool.

Guiding Computer Design

Measure overall performance.

Determine characteristics of programs.
E.g., frequency of floating-point operations.

Determine effect of design options.

Choosing Benchmark Programs

Important: Choice of programs for evaluation.

Optimal but unrealistic:

The exact set of programs customer will run.

Problem: computers used for different applications.

Therefore, must model typical users' workload.

Options:

Real Programs:
Programs chosen using surveys, for example.

+ Measured performance improvements apply to customer.
– Large programs hard to run on simulator. (Before system built.)

Kernels:
Use part of program responsible for most execution time.

+ Easier to study.
– Not all program have small kernels.

Toy Benchmarks:
Program performs simplified version of common task.

+ Easier to study.
– May not be realistic.

Synthetic Benchmarks:
Program "looks like" typical program, but does nothing useful.

+ Easier to study.
– May not be realistic.

Commonly Used Option

Overall performance: real programs

Test specific features: synthetic benchmarks.

Benchmark Suites

Benchmark Suite:
A named set of programs used to evaluate a system.

Typically:

- Developed and managed by a publication or non-profit organization.
  E.g., Standard Performance Evaluation Corp., PC Magazine.
- Tests clearly delineated aspects of system.
  E.g., CPU, graphics, I/O, application.
- Specifies a set of programs and inputs for those programs.
- Specifies reporting requirements for results.

What Suites Might Measure

- Application Performance
  E.g., productivity (office) applications, database programs.
  Usually tests entire system.
- CPU and Memory Performance
  Ignores effect of I/O.
- Graphics Performance

Example, SPEC CPU2000 Suites

Respected measure of CPU performance.

Managed by Standard Performance Evaluation Corporation,...
...a non-profit organization funded by computer companies.

Measures CPU and memory performance on integer and FP code.

Uses common Unix programs such as perl, gcc, gzip.

Requires that results on each program be reported.

Programs compiled with publicly available compilers and libraries.

Programs compiled with and without expert tuning.

02-29

SPEC CPU2000 Suites and Measures

Suite of integer programs run to determine:

- SPECint2000, execution time of tuned code.
- SPECint_base2000, execution time of untuned code.
- SPECint_rate2000, throughput of tuned code.
- SPECint_rate_base2000, throughput of untuned code.

Suite of floating programs run to determine:

- SPECfp2000, execution time of tuned code.
- SPECfp_base2000, execution time of untuned code.
- SPECfp_rate2000, throughput of tuned code.
- SPECfp_rate_rate2000, throughput of untuned code.

02-30

Other Examples

(Fall 2001: This list is out of date.)

BAPCO Suites, measure productivity app. performance on Windows 95.

TPC, measure "transaction processing" system performance.

WinMARK, graphics performance.