EE345

Instruction Set Architecture
Measuring Performance

ISA: Instruction Set Architecture

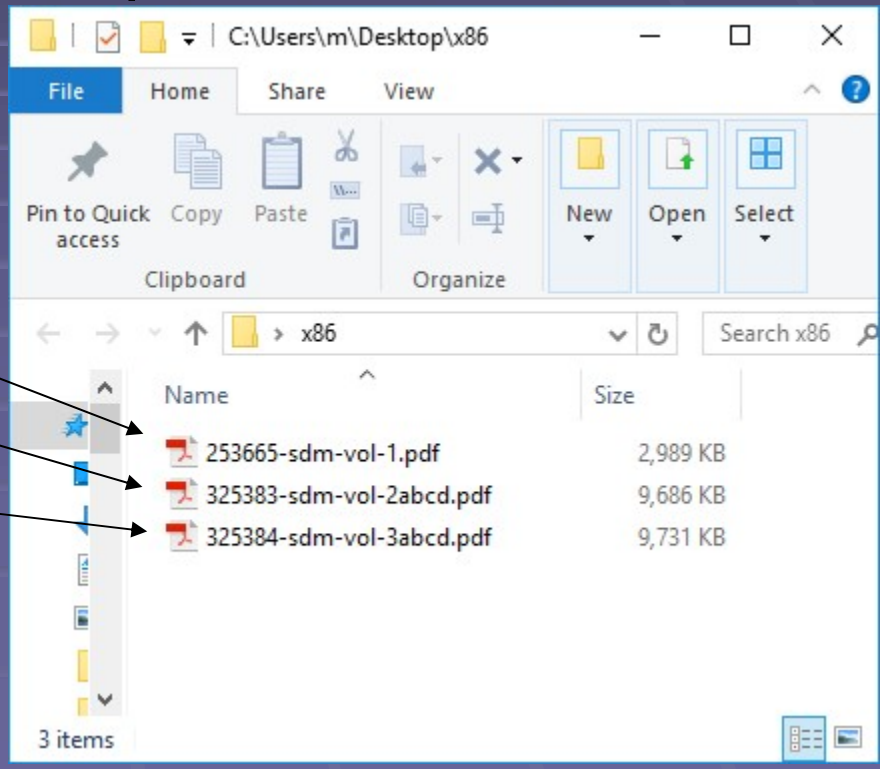
What's ISA?

- Instruction Set Architecture
  - Software's view of the computer

Basic Arch

Instructions

System Arch



- Full specs can be huge
  - Example: x86
  - EE345: we mostly care about basic features

What's in ISA (abridged version)

- Instructions: smallest unit of CPU work programmer can specify
  - Also known as "machine code"
  - Add two values
  - Load from memory
  - Etc.
- Registers: very fast local storage
  - Usually small number: 1-32 registers, depending on architecture
  - Each register can contain a single value
- Memory: slower, larger, storage
  - Dominant technology: SDRAM

More on Instructions

- CPU fetches/consumes instructions
  - Instructions are encoded in binary
  - This encoding is called “machine language” or “machine code”
- What’s in an instruction?
  - Operation
    - like ADD
  - Operand specifiers
    - like R1, or memory address
- We’ll get into much more depth, later

Some Ways to Generate Instructions

- Assembly language
  - Line of assembly ~= one CPU instruction
- "Higher Level Language" like C
  - One line can generate many CPU instructions

```
void main() {
  printf("Hello world!\n");
}
```

C code

```
29 0004 BF000000        movl    $.LC0, %edi
30 0009 E8000000        call    puts
```

Register

Instruction address

Line number

Encoding (machine code)

Assembly language

Performance (Empirical)

Empirical Performance (Measurement)

- Can do this on actual HW
  - CPU(s) in a system
- Can do it on a model
  - RTL in simulator
    - slow – typically 10's of cycle/second
  - Perf model, written in C or similar
    - Faster – maybe 10-100KHz simulation speed
  - Emulation – FPGA's or similar
    - Perhaps 1MHz

Performance: Empirical Method

- Computations per unit time
- Often use standard benchmarks
- Bottom line: how long to execute a particular program (or “suite” = collection of programs)
  - Dhrystone, SPEC, PCMark, LINPACK, Sunspider, …
- But…
  - Which tool-chain used? (compiler, etc)
  - Which operating system?
  - System configuration?
- CPU vendors good at gaming benchmarks

For Example… SPEC CPU

- A big deal for general-purpose CPU's
  - Multiple versions: CPU95, …, CPU2006, CPU2017
- See results for examples: listing all sorts of conditions and caveats

Results for one test: 5 pages, majority is conditions/caveats of test-run

http://spec.org/cpu2006/results/res2011q2/cpu2006-20110411-15616.pdf

Hardware	
CPU Name:	Intel Core i7-2600
CPU Characteristics:	Intel Turbo Boost Technology up to 3.8 GHz
CPU MHz:	3400
FPU:	Integrated
CPU(s) enabled:	4 cores, 1 chip, 4 cores/chip, 2 threads/core
CPU(s) orderable:	1 chip
Primary Cache:	32 KB I + 32 KB D on chip per core
Secondary Cache:	256 KB I+D on chip per core
L3 Cache:	8 MB I+D on chip per chip
Other Cache:	None
Memory:	8 GB (2 x 4 GB 2Rx8 PC3-10600U-9)
Disk Subsystem:	Seagate 1 TB SATA, 7200 RPM
Other Hardware:	None

Software	
Operating System:	Windows 7 Ultimate (64-bit)
Compiler:	Intel C++ Compiler XE for Intel64 Version 12.0.3.163 Build 20110217 Microsoft Visual Studio 2008 Professional SP1 (for libraries)
Auto Parallel:	Yes
File System:	NTFS
System State:	Default
Base Pointers:	32/64-bit
Peak Pointers:	32/64-bit
Other Software:	SmartHeap Library Version 9.01 from http://www.microquill.com/

Examples of Maximizing Score

- These aren't exactly cheats, but most consumers of SPEC probably don't think about them
- Auto-parallelization
  - A benchmark is spread across multiple processors by the compiler
- Run some benchmarks in 32b mode, others in 64b mode
- Custom code in compiler "recognizes" benchmark code sequences
- Use non-standard memory allocator library
- Tweak compiler flags for each benchmark
- (And lots of others)

Performance: To Be Really Fair

- To compare processors, ideally…
  - Run exact same benchmark binary
  - Run in exactly the same system
  - Run multiple times, report all
    - to weed out asynchronous events
- Unfortunately, this is pretty hard to do!
  - Usually can't replace just the CPU in a system
  - Easier to control all variables when modeling

Performance (Analytical)

Basic Formula

- executionTime =
  instructions * cycles/instruction * time/cycle

  - instructions = number of instructions that execute while running a particular benchmark
  - cycles/instruction (CPI) = the number of machine cycles needed to execute a single insrtuction
  - time/cycle = how fast the computer executes each cycle

These variables tend to be fuzzy, let's examine each…

Execution Time: Instructions

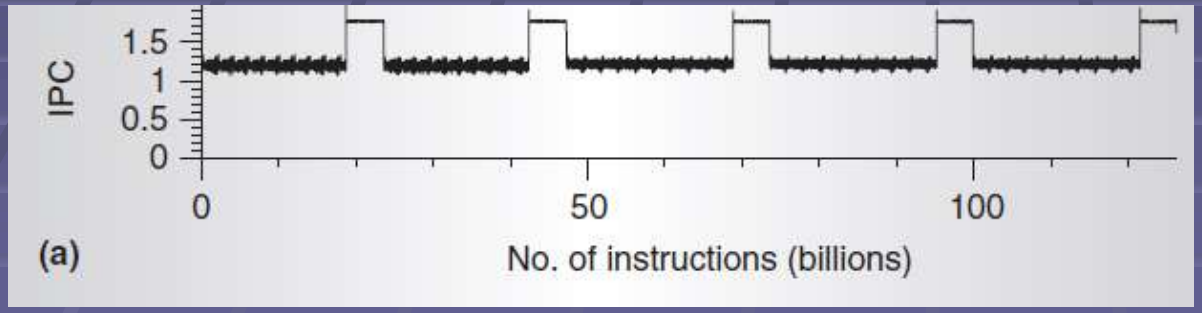
- Some path through the program, which includes a definite number of instructions
- But, code that's not really part of the benchmark may be involved
  - Operating System (OS)
  - Library code
  - Asynchronous events (like network drivers)

Execution Time: CPI

- You'll also see IPC quoted
  - Especially on superscalar CPU's, where more than one instruction can execute each cycle
- Assume: using average CPI is a good proxy for what's happening
  - But IPC varies throughout run
  - Different instructions require different # cycles
  - IPC depends on rest of system
    - E.g., may need to read memory that's "far away"
    - E.g., OS may block benchmark while doing something
- Still, the concept of average CPI is useful and widely used

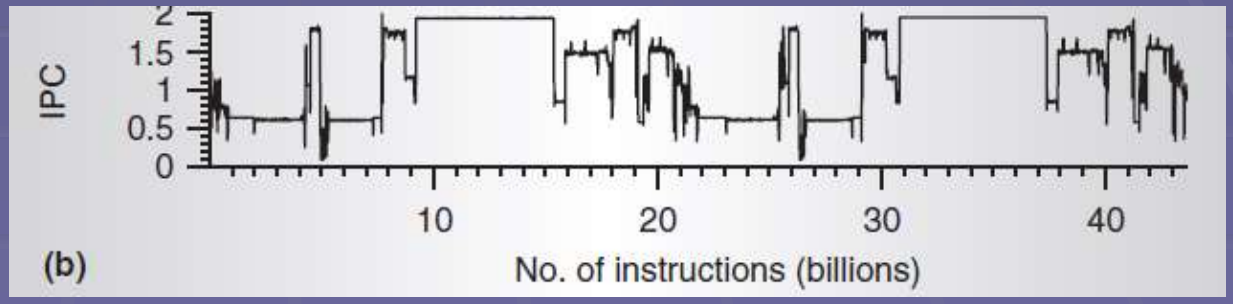
IPC can vary by time (and program phase)

- From *Discovering and Exploiting Program Phases* (Sherwood et al)



Gzip (compression)

Gcc (C compiler)



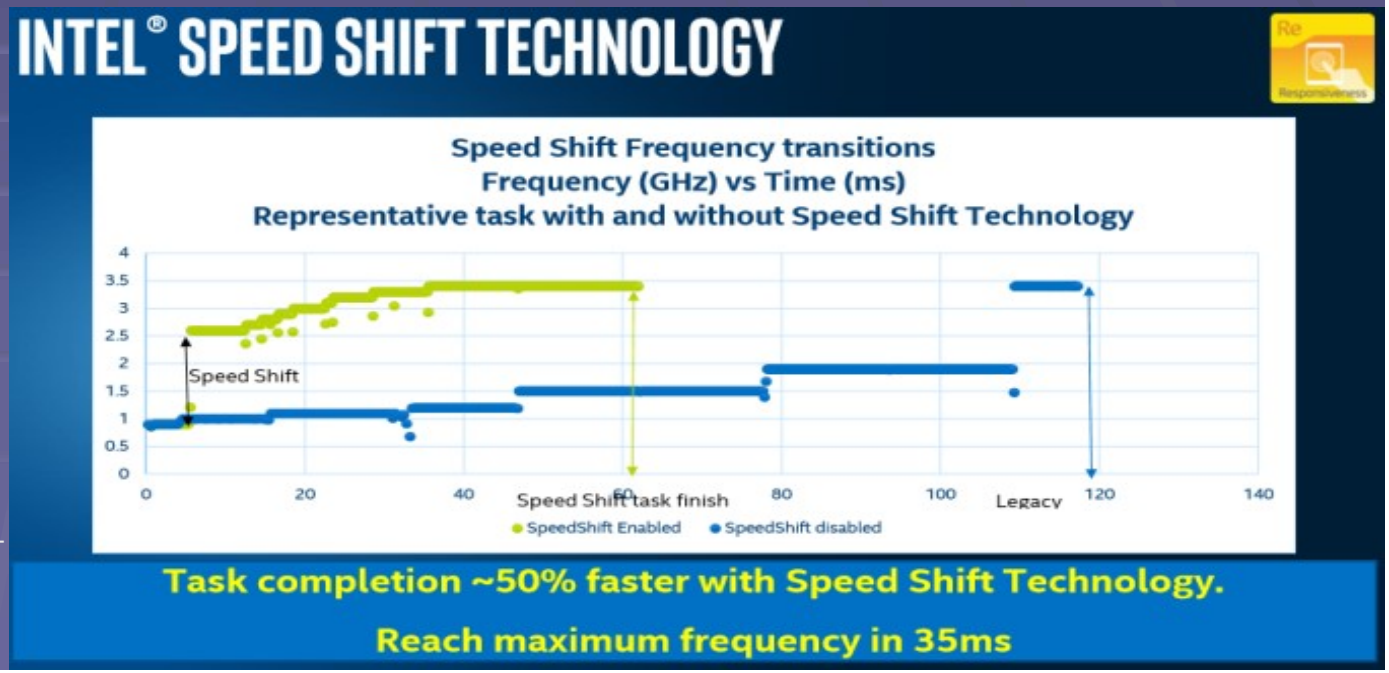
Execution Time: Time/Cycle

- This comes down to CPU clock rate
  - E.g., 1ns/cycle indicates a 1GHz clock
- Modern processors don't have a single clock rate
  - Vary according to load, power, thermals

For measurement, may need to disable dynamic clock speed.

SkyLake CPU information

http://www.anandtech.com/show/9751/examining-intel-skylake-speed-shift-more-responsive-processors



Other Metrics You'll See

- MIPS: Millions of Instructions per Second
  - But… Not all instructions are equal
  - E.g., ADD is faster than DIVIDE
- FLOPS: Floating Point Operations per Second
  - But… Everyone quotes a combined instruction: MUL+ADD (multiply accumulate)
  - FP ADD's might have ½ the FLOPS rating

Which Is Faster?

Compiler	IPC on benchmark	Instruction count	Frequency (Hz)	Time (ms)
A	1.2	3.2e6	1.0E9	2.67
B	1.5	4.4e6	1.0E9	2.93

- Incorrect Answer:
  Compiler B gives faster run, because it yields more instructions per cycle
- Correct Answer:
  IPC doesn't tell us; we need absolute time information
- Precise Answer:
  Compiler A generates fewer instructions, overall time is less

Comparing Performance

- Easy when there's just one benchmark
  - $Perf_1 = 1 / executionTime_1(benchmark)$
  - $Perf_2 = 1 / executionTime_2(benchmark)$
  - $Speedup = Perf_2 / Perf_1$
- What if there are multiple benchmarks?
  - In a suite, like SPEC CPU
  - Need some sort of aggregate score
  - Usually the suite will specify how to aggregate

Comparing Performance (for Suites)

- SPEC CPU uses geometric mean
  - Many detractors
    - James E. Smith. Characterizing Computer Performance with a Single Number CACM 31(10):1202–1206, October 1988
    - Jacob and Mudge. Notes on Calculating Computer Performance, https://pdfs.semanticscholar.org/bf25/5a72d4c0a48d915204cd4866bc8bf7289642.pdf
  - But you have to do it anyway
    - Mandated by SPEC
- Geomean
  - A baseline run is needed. This is a set of execution times for each of the N benchmark components, supplied by SPEC.
  - Multiply the ratio of all the suite components
    - Prod = $P1_{new} / P1_{base} * P2_{new} / P2_{base} * \ldots * Pn_{new} / Pn_{base}$
  - Take the n^{th} root of the product

$$\sqrt[n]{\prod Pi_{new} / Pi_{old}}$$

Comparing Performance (for Suites)

- People currently like a simple arithmetic mean (or inverse = harmonic mean)
- May also want to weight each benchmark
  - According to importance
- Typically, we want to look at result for each component anyway
- But, danger!, often we run 10's of suites, each with a dozen or more components
- So, there's no simple answer!