Computer Architecture

Performance

Virendra Singh

Professor

Computer Architecture and Dependable Systems Lab

Department of Electrical Engineering

Indian Institute of Technology Bombay

http://www.ee.iitb.ac.in/~viren/

E-mail: viren@ee.iitb.ac.in

CS-683: Advanced Computer Architecture

Lecture 2 (02 August 2021)

CADSL

Running Program on Processor

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) (CPI) (cycle time)

Architecture --> Implementation --> Realization

Compiler Designer **Processor Designer** **Chip Designer**

Performance Comparison

- Machine A is n times faster than machine B iff

  perf(A)/perf(B) = time(B)/time(A) = n

- Machine A is x% faster than machine B iff
  - perf(A)/perf(B) = time(B)/time(A) = 1 + x/100
- E.g. time(A) = 10s, time(B) = 15s
  - 15/10 = 1.5 => A is 1.5 times faster than B
  - 15/10 = 1.5 => A is 50% faster than B

Other Metrics

- MIPS and MFLOPS
- MIPS = instruction count/(execution time x 10^6)

  = clock rate/(CPI x 10^6)
- But MIPS has serious shortcomings

Problems with MIPS

- E.g. without FP hardware, an FP op may take 50 single-cycle instructions
- With FP hardware, only one 2-cycle instruction

- **Thus, adding FP hardware:**
  - CPI increases — 50/50 => 2/1
  - Instructions/program decreases — 50 => 1
  - Total execution time decreases — 50 => 2
- BUT, MIPS gets worse! — 50 MIPS => 2 MIPS

Problems with MIPS

- Ignores program
- Usually used to quote peak performance
  - Ideal conditions => guaranteed not to exceed!
- When is MIPS ok?
  - Same compiler, same ISA
  - e.g. same binary running on AMD Phenom, Intel Core i7
  - Why? Instr/program is constant and can be ignored

Other Metrics

- MFLOPS = FP ops in program/(execution time x 10^6)
- Assuming FP ops independent of compiler and ISA
  - Often safe for numeric codes: matrix size determines # of FP ops/program
  - However, not always safe:
    - Missing instructions (e.g. FP divide)
    - Optimizing compilers
- Relative MIPS and normalized MFLOPS
  - Adds to confusion

Rules

- Use ONLY Time
- Beware when reading, especially if details are omitted
- Beware of Peak
  - “Guaranteed not to exceed”

Example

- Machine A: clock 1ns, CPI 2.0, for program x
- Machine B: clock 2ns, CPI 1.2, for program x
- Which is faster and how much?

  Time/Program = instr/program x cycles/instr x sec/cycle

  Time(A) = N x 2.0 x 1 = 2N

  Time(B) = N x 1.2 x 2 = 2.4N

  Compare: Time(B)/Time(A) = 2.4N/2N = 1.2
- So, Machine A is 20% faster than Machine B for this program

Which Programs

- Execution time of what program?
- Best case – your always run the same set of programs
  - Port them and time the whole workload
- In reality, use benchmarks
  - Programs chosen to measure performance
  - Predict performance of actual workload
  - Saves effort and money
  - Representative? Honest? Benchmarketing…

Benchmarks: SPEC2000

- System Performance Evaluation Cooperative
  - Formed in 80s to combat benchmarketing
  - SPEC89, SPEC92, SPEC95, SPEC2000
- 12 integer and 14 floating-point programs
  - Sun Ultra-5 300MHz reference machine has score of 100
  - Report GM of ratios to reference machine

Benchmarks: SPEC CINT2000

Benchmark	Description
164.gzip	Compression
175.vpr	FPGA place and route
176.gcc	C compiler
181.mcf	Combinatorial optimization
186.crafty	Chess
197.parser	Word processing, grammatical analysis
252.eon	Visualization (ray tracing)
253.perlbmk	PERL script execution
254.gap	Group theory interpreter
255.vortex	Object-oriented database
256.bzip2	Compression
300.twolf	Place and route simulator

Benchmarks: SPEC CFP2000

Benchmark	Description
168.wupwise	Physics/Quantum Chromodynamics
171.swim	Shallow water modeling
172.mgrid	Multi-grid solver: 3D potential field
173.applu	Parabolic/elliptic PDE
177.mesa	3-D graphics library
178.galgel	Computational Fluid Dynamics
179.art	Image Recognition/Neural Networks
183.equake	Seismic Wave Propagation Simulation
187.facerec	Image processing: face recognition
188.ammp	Computational chemistry
189.lucas	Number theory/primality testing
191.fma3d	Finite-element Crash Simulation
200.sixtrack	High energy nuclear physics accelerator design
301.apsi	Meteorology: Pollutant distribution

How to Average

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

- One answer: for total execution time, how much faster is B? 9.1x

How to Average

- Another: arithmetic mean (same result)
- Arithmetic mean of times:
- AM(A) = 1001/2 = 500.5
- AM(B) = 110/2 = 55
- 500.5/55 = 9.1x

$$\left\{\sum_{i=1}^{n} time(i)\right\} \times \frac{1}{n}$$

- Valid only if programs run equally often, so use weighted arithmetic mean:

$$\left\{\sum_{i=1}^{n} \left(weight(i) \times time(i)\right)\right\} \times \frac{1}{n}$$

Other Averages

- E.g., 30 mph for first 10 miles, then 90 mph for next 10 miles, what is average speed?
- Average speed = (30+90)/2 **WRONG**
- Average speed = total distance / total time

  = (20 / (10/30 + 10/90))

  = 45 mph

Harmonic Mean

- Harmonic mean of rates = $$\frac{n}{\left\{ \sum_{i=1}^{n} \frac{1}{rate(n)} \right\}}$$
- Use HM if forced to start and end with rates (e.g. reporting MIPS or MFLOPS)
- Why?
  - Rate has time in denominator
  - Mean should be proportional to inverse of sums of time (not sum of inverses)
  - See: J.E. Smith, "Characterizing computer performance with a single number," CACM Volume 31 , Issue 10 (October 1988), pp. 1202-1206.

Dealing with Ratios

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

- If we take ratios with respect to machine A

	Machine A	Machine B
Program 1	1	10
Program 2	1	0.1

Dealing with Ratios

- Average for machine A is 1, average for machine B is 5.05
- If we take ratios with respect to machine B

	Machine A	Machine B
Program 1	0.1	1
Program 2	10	1
Average	5.05	1

- Can't both be true!!!
- Don't use arithmetic mean on ratios!

Geometric Mean

- Use geometric mean for ratios
- Geometric mean of ratios =

$$\sqrt[n]{\prod_{i=1}^{n} ratio(i)}$$

- Independent of reference machine
- In the example, GM for machine a is 1, for machine B is also 1
  - Normalized with respect to either machine

But…

- GM of ratios is not proportional to total time
- AM in example says machine B is 9.1 times faster
- GM says they are equal
- If we took total execution time, A and B are equal only if
  - Program 1 is run 100 times more often than program 2
- Generally, GM will mispredict for three or more machines

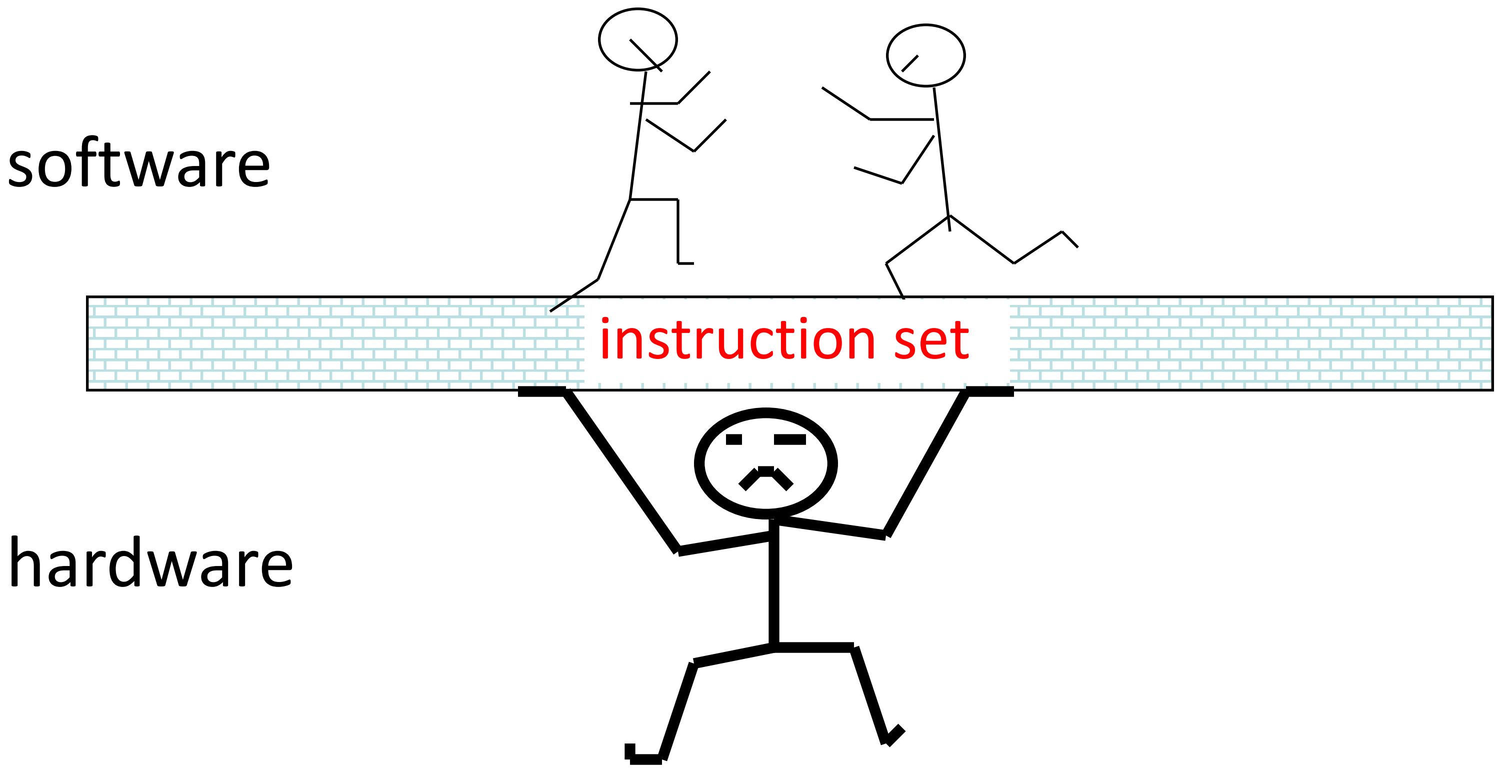
INSTRUCTION SET ARCHITECTURE

Instruction Set Architecture

- Instruction set architecture is the structure of a computer that a machine language programmer must understand to write a correct (timing independent) program for that machine.

- The instruction set architecture is also the machine description that a hardware designer must understand to design a correct implementation of the computer.

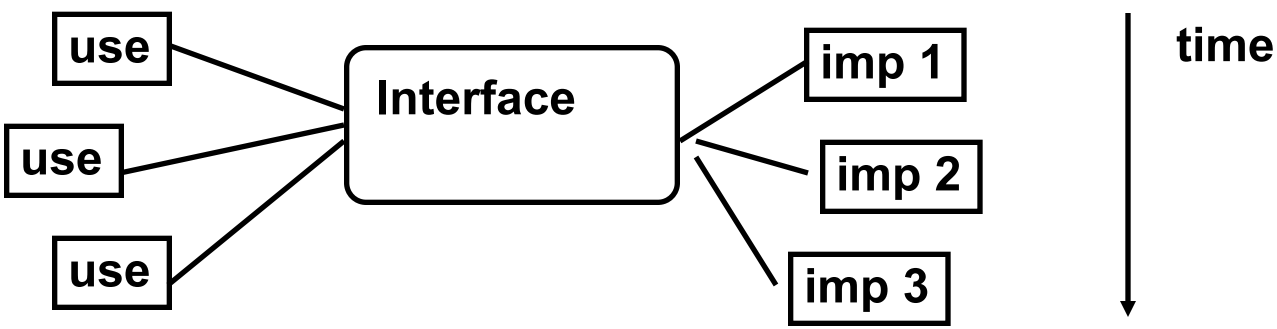
Instruction Set Architecture (ISA)



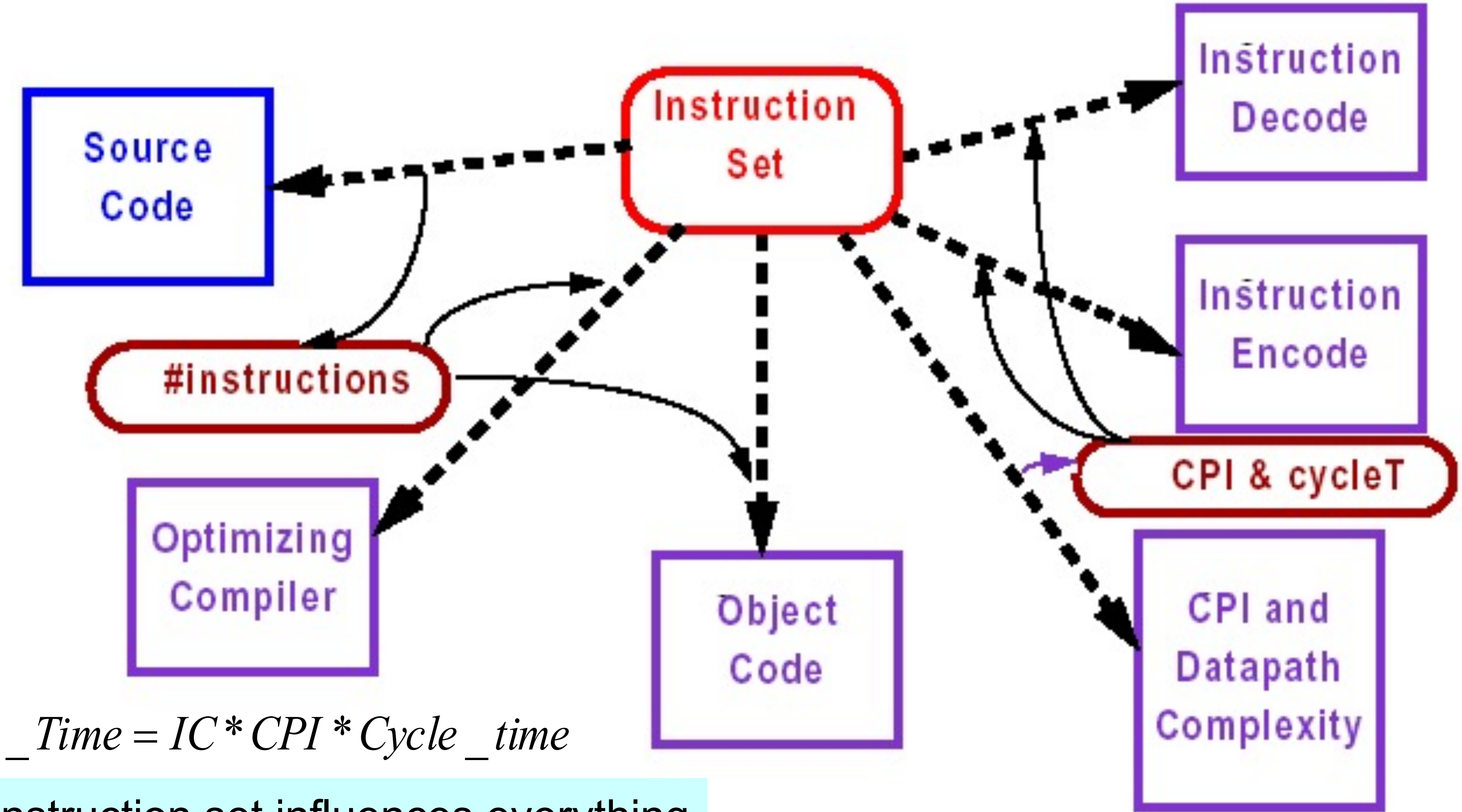
ISA as Interface

A good interface:

- Lasts through many implementations (portability, compatibility)
- Is used in many different ways (generality)
- Provides *convenient* functionality to higher levels
- Permits an *efficient* implementation at lower levels



Instruction Set Design



$$CPU\_Time = IC * CPI * Cycle\_time$$

The instruction set influences everything

Instruction

- C Statement

  f = (g+h) – (i+j)

- Assembly instructions

  add t0, g, h

  add t1, I, j

  sub f, t0, t1

- Opcode/mnemonic, operand , source/destination

Why not Bigger Instructions?

- Why not “f = (g+h) – (i+j)” as one instruction?
- Church’s thesis: A very primitive computer can compute anything that a fancy computer can compute – you need only logical functions, read and write to memory, and data dependent decisions
- Therefore, ISA selection is for practical reasons
  - Performance and cost not computability
- Regularity tends to improve both
  - E.g, H/W to handle arbitrary number of operands is complex and slow, and UNNECESSARY

What Must an Instruction Specify?(I)

Data Flow

←

- Which operation to perform add r0, r1, r3
  - Ans: Op code: add, load, branch, etc.
- Where to find the operands: add r0, r1, r3
  - In CPU registers, memory cells, I/O locations, or part of instruction
- Place to store result add r0, r1, r3
  - Again CPU register or memory cell

What Must an Instruction Specify?(II)

- Location of next instruction

  add r0, r1, r3
  br endloop

  – Almost always memory cell pointed to by program counter—PC

- Sometimes there *is* no operand, or no result, or no next instruction. Can you think of examples?

Instructions Can Be Divided into 3 Classes (I)

- Data movement instructions
  - Move data from a memory location or register to another memory location or register without changing its form
  - *Load*—source is memory and destination is register
  - *Store*—source is register and destination is memory
- Arithmetic and logic (ALU) instructions
  - Change the form of one or more operands to produce a result stored in another location
  - *Add, Sub, Shift*, etc.
- Branch instructions (control flow instructions)
  - Alter the normal flow of control from executing the next instruction in sequence
  - *Br Loc, Brz Loc2*,—unconditional or conditional branches

Types of Operations

✓ Arithmetic and Logic:	AND, ADD
✓ Data Transfer:	MOVE, LOAD, STORE
✓ Control	BRANCH, JUMP, CALL
✓ System	OS CALL, VM
✓ Floating Point	ADDF, MULF, DIVF
✓ Decimal	ADDD, CONVERT
✓ String	MOVE, COMPARE
✓ Graphics	(DE)COMPRESS

Thank You