Computer System

Instruction Set Architecture

Virendra Singh

Associate Professor

Computer **A**rchitecture and **D**ependable **S**ystems **L**ab

Department of Electrical Engineering

Indian Institute of Technology Bombay

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

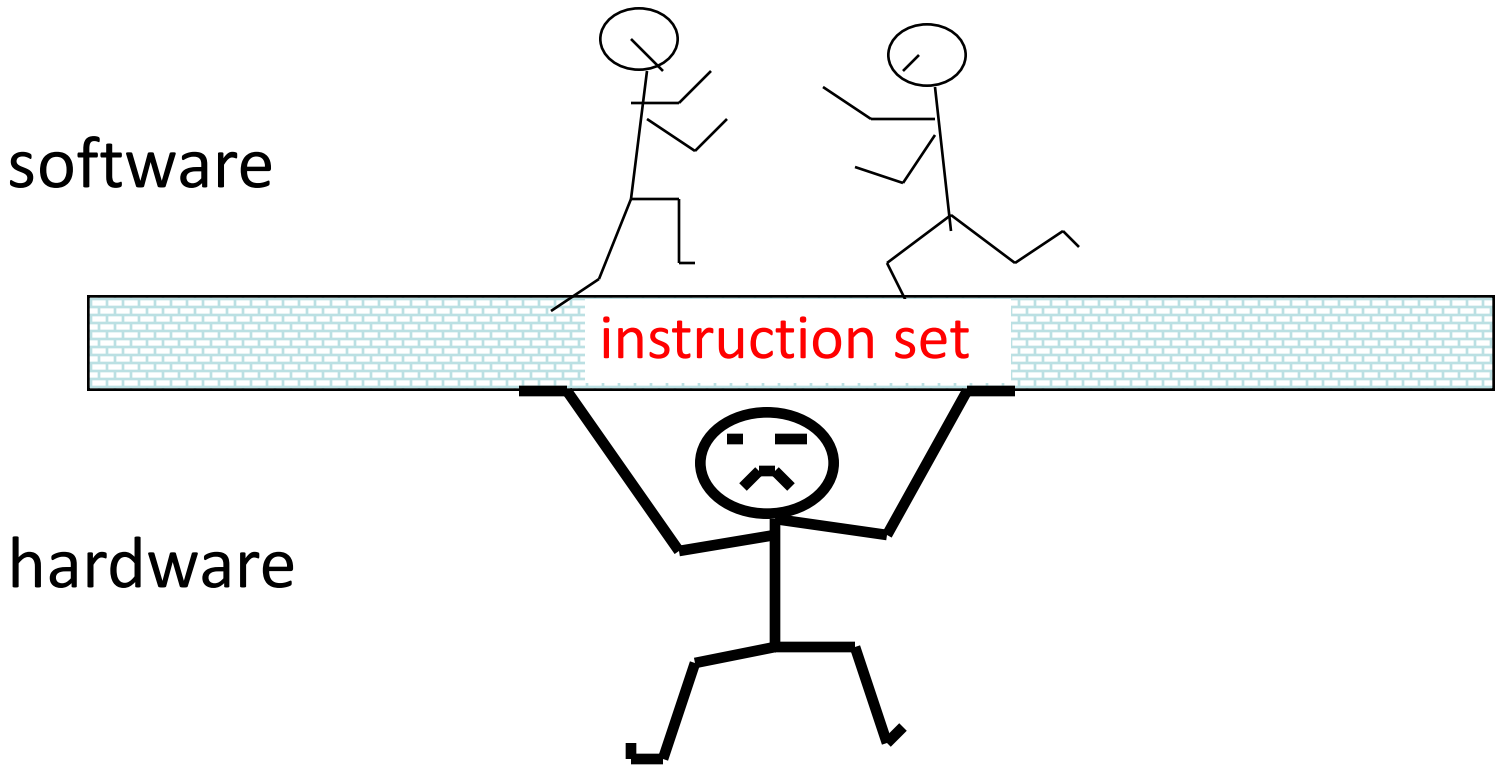
E-mail: viren@ee.iitb.ac.in

CP-226:Computer Architecture

Lecture 5 (07 Feb 2013)

CADSL

Instruction Set Architecture (ISA)

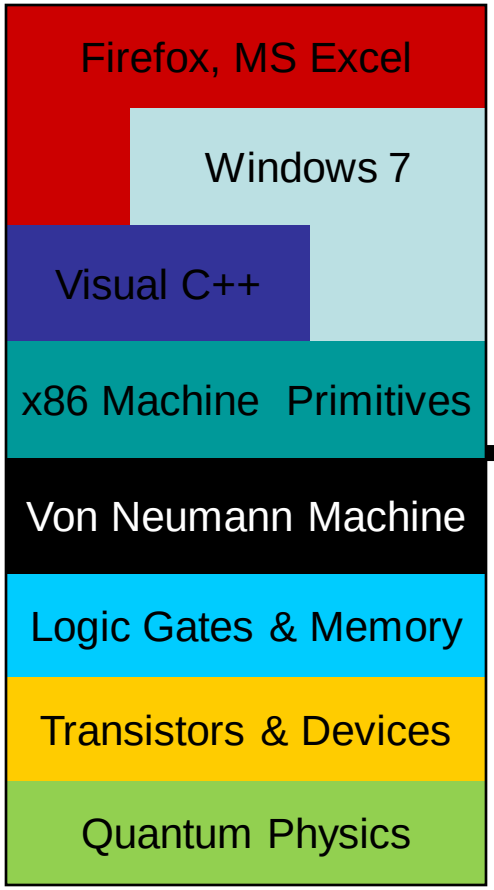


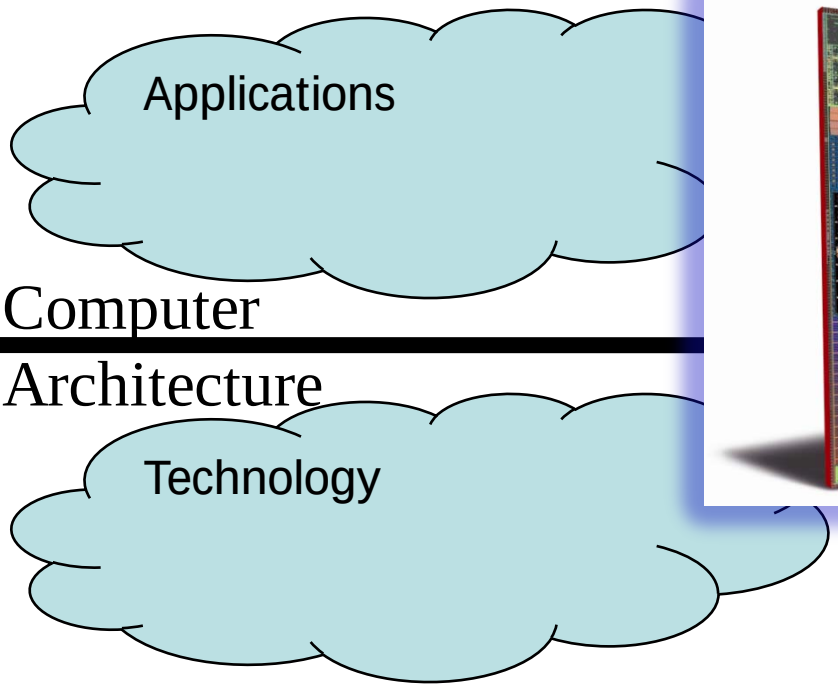
Instruction Set Architecture

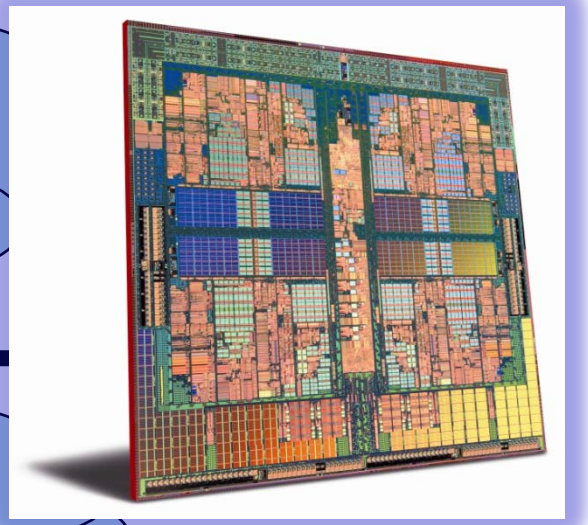
- Instruction set architecture is the structure of a computer that a machine language programmer must understand to write a correct (timing independent) program for that machine.

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

Computer Architecture





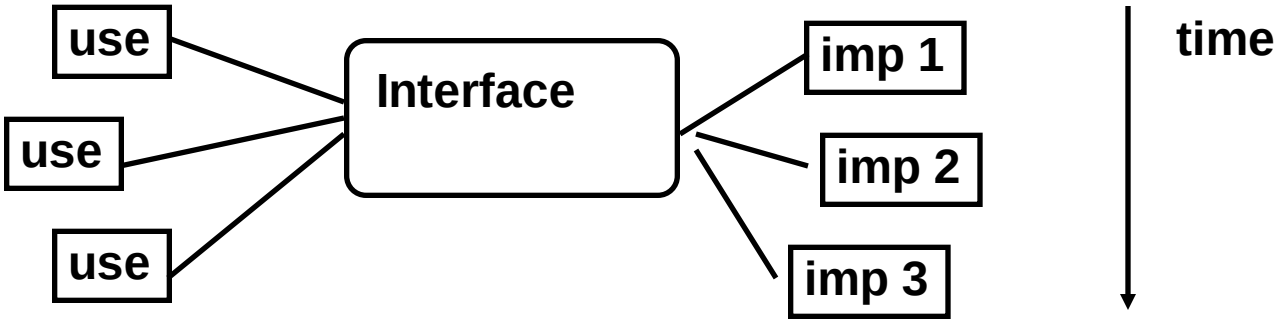


- Rely on *abstraction layers* to manage complexity

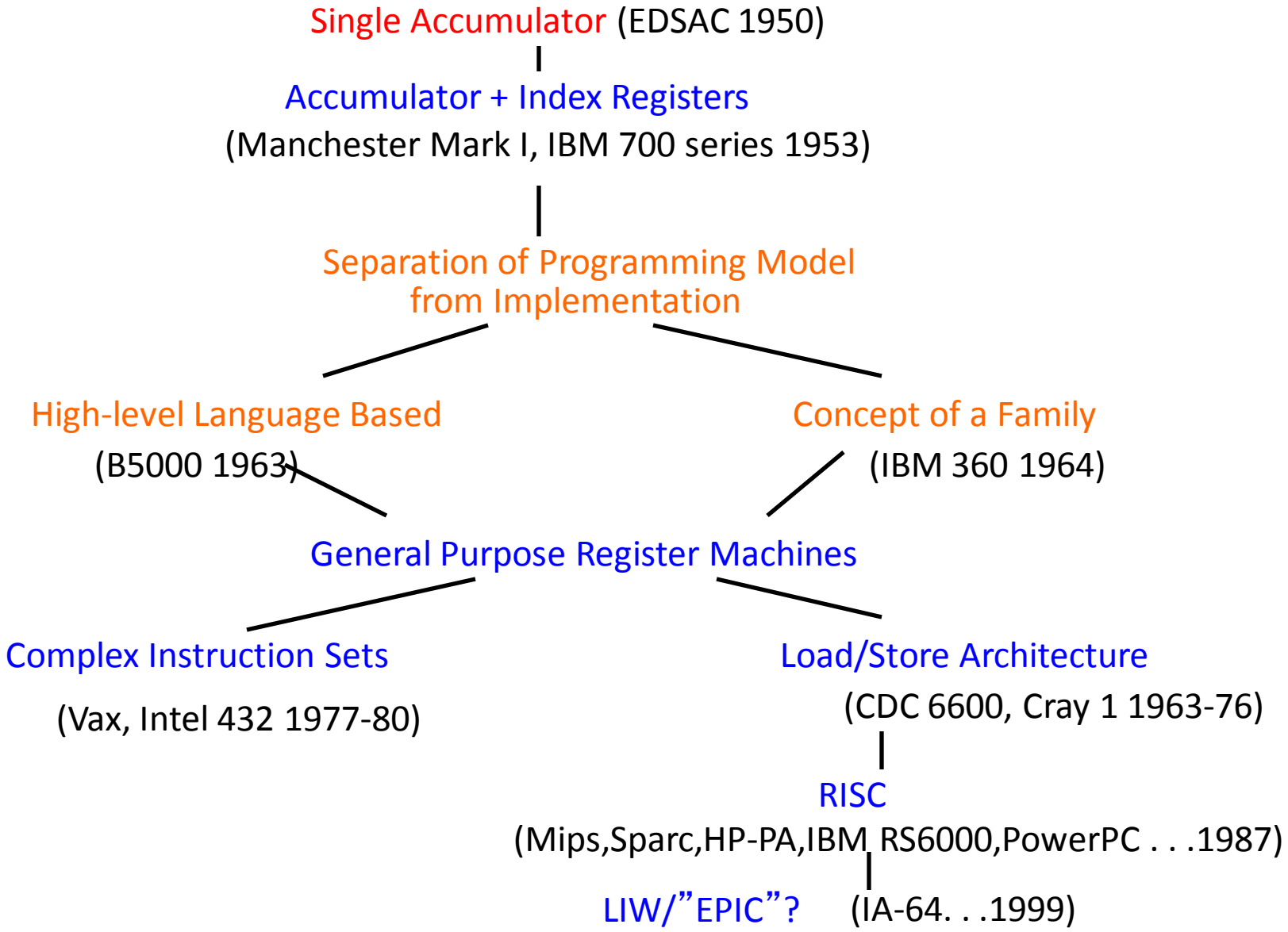
Interface Design

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



Evolution of Instruction Sets



Evolution of Instruction Sets

- Major advances in computer architecture are typically associated with landmark instruction set designs
  - Ex: Stack vs GPR (System 360)
- Design decisions must take into account:
  - technology
  - machine organization
  - programming languages
  - compiler technology
  - operating systems
- And they in turn influence these

What Are the Components of an ISA?

- Sometimes known as *The Programmer's Model* of the machine
- Storage cells
  - General and special purpose registers in the CPU
  - Many general purpose cells of same size in memory
  - Storage associated with I/O devices
- The machine instruction set
  - The instruction set is the entire repertoire of machine operations
  - Makes use of storage cells, formats, and results of the fetch/execute cycle
  - i.e., register transfers

What Are the Components of an ISA?

- The instruction format
  - Size and meaning of fields within the instruction
- The nature of the fetch-execute cycle
  - Things that are done before the operation code is known

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

Use of Computers

- Desktop
  - Performance of program with Floating point, integer data type
  - Little regard to program size
- Servers
  - Data bases, file servers. Etc.
  - Integer and character strings
- Embedded Systems
  - Values cost, energy, code size

ISA Classification

- Type of internal storage in a processor is the most basic differentiator
- Stack Architecture
- Accumulator Architecture
- General Purpose Register Architecture

ISA Classification

# Memory Address	Max. no. of operands allowed	Type of architecture	Examples
0	3	Load-Store	Alpha, ARM, MIPS, PowerPC
1	2	Reg-Mem	IBM360, Intel x86, 68000
2	2	Mem-Mem	VAX
3	3	Mem-Mem	VAX

ISA Classification

Type	Adv	Disadv
Reg-Reg	Simple, fixed length encoding, simple code generation, all instr. Take same no. of cycles	Higher instruction count, lower instruction density
Reg-Mem	Data can be accessed without separate load instruction first, instruction format tend to be easy to encode and yield good density	Encoding register no and memory address in each instruction may restrict the no. of registers.
Mem-Mem	Most compact, doesn't waste registers for temporaries	Large variation in instruction size, large variation in in amount of work (NOT USED TODAY)

Thank You