Microprocessors

Introduction

Virendra Singh

Associate Professor

Computer Architecture and Dependable Systems Lab

Department of Electrical Engineering

Indian Institute of Technology Bombay

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

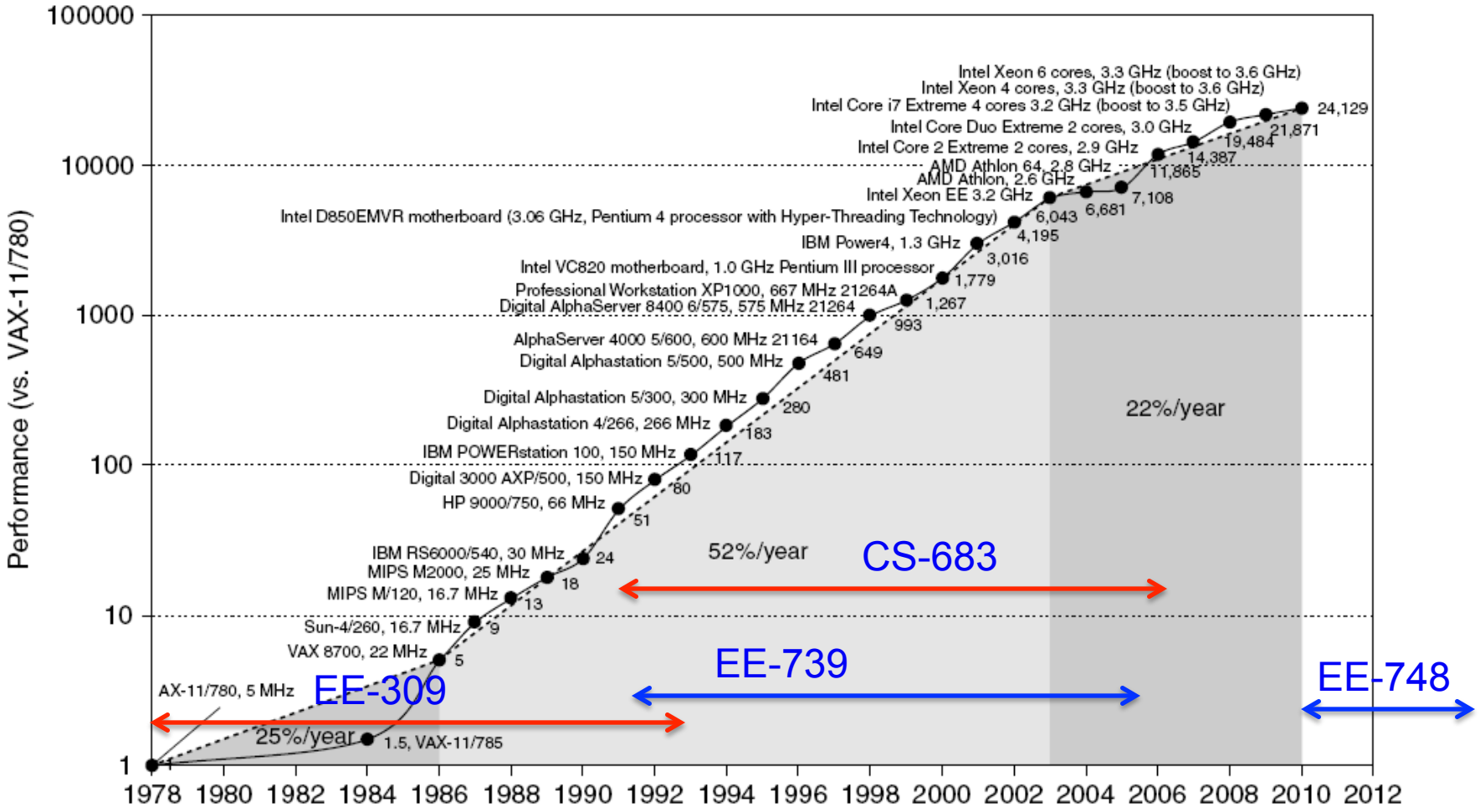
E-mail: viren@ee.iitb.ac.in

EE-309: Microprocessors

Lecture 1 (22 July 2014)

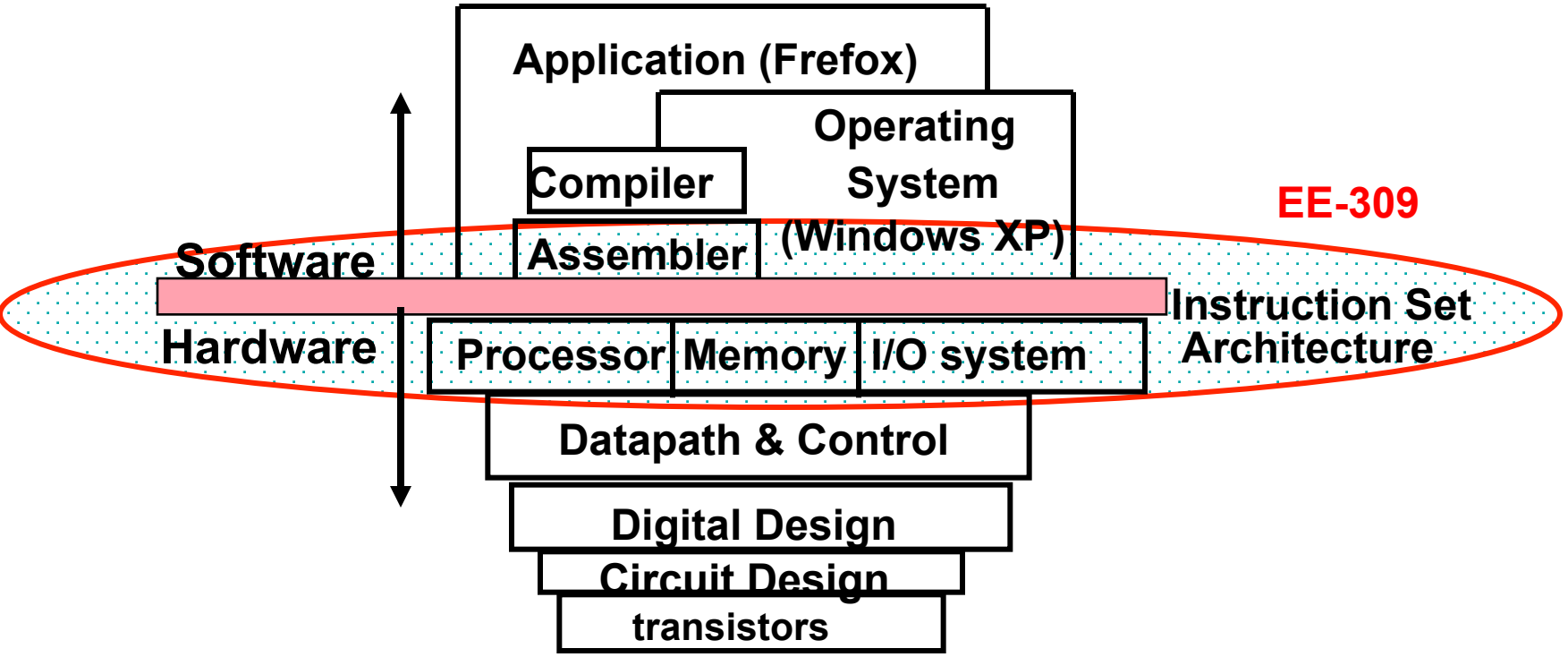
CADSL

Single Processor Performance



CADSL

What is This Course About?



- Coordination of many *levels of abstraction*

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

Performance and Cost

- Which of the following airplanes has the best performance?

Airplane	Passengers	Range (mi)	Speed (mph)
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

- How much faster is the Concorde vs. the 747
- How much bigger is the 747 vs. DC-8?

Performance and Cost

- Which computer is fastest?

- Not so simple
  - Scientific simulation – FP performance
  - Program development – Integer performance
  - Database workload – Memory, I/O

Performance of Computers

- Want to buy the fastest computer for what you want to do?
  - Workload is all-important
  - Correct measurement and analysis
- Want to design the fastest computer for what the customer wants to pay?
  - Cost is an important criterion

Defining Performance

- What is important to whom?
- Computer system user
  - Minimize elapsed time for program

    = time_end – time_start
  - Called response time
- Computer center manager
  - Maximize completion rate = #jobs/second
  - Called throughput

Other Metrics

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

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

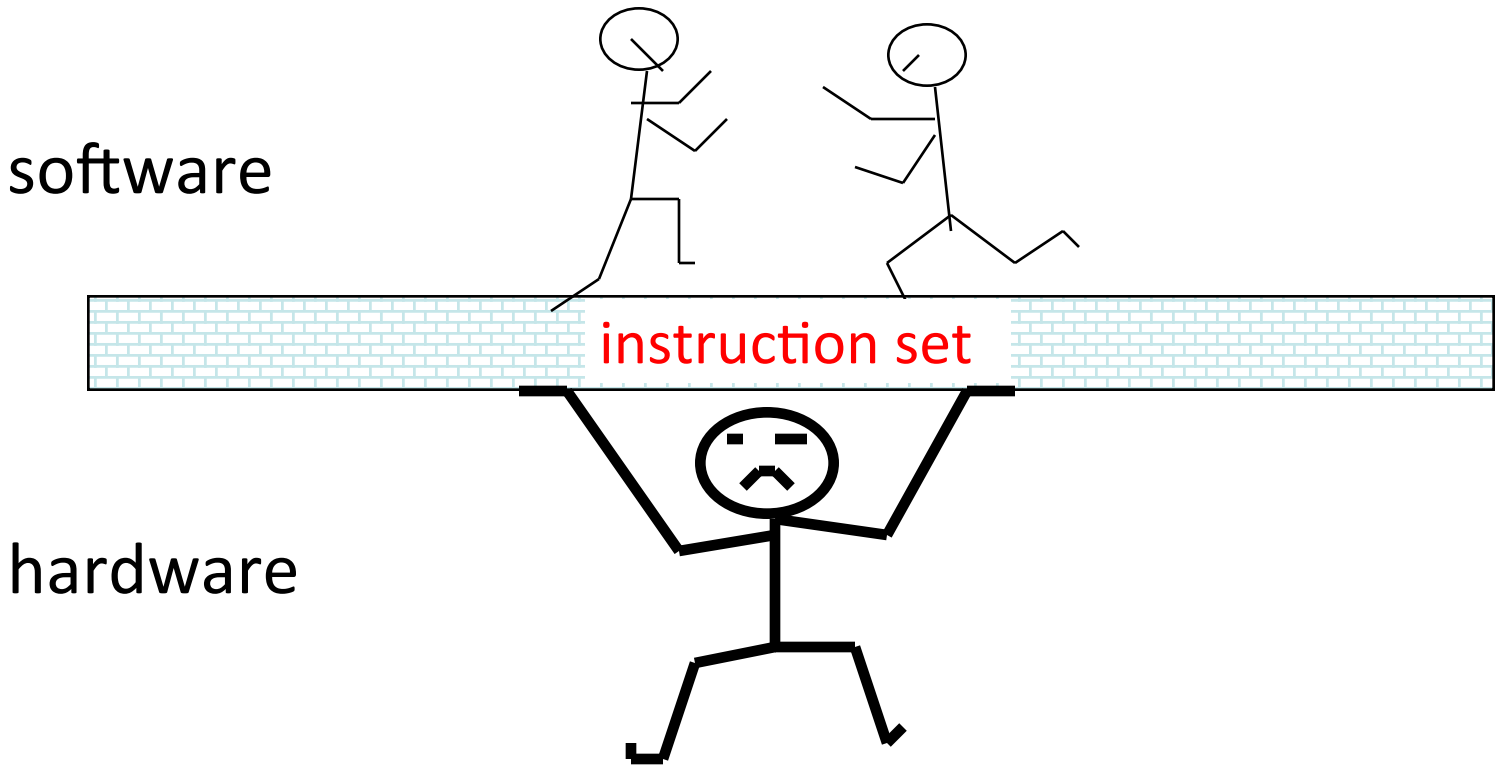
INSTRUCTION SET ARCHITECTURE

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.

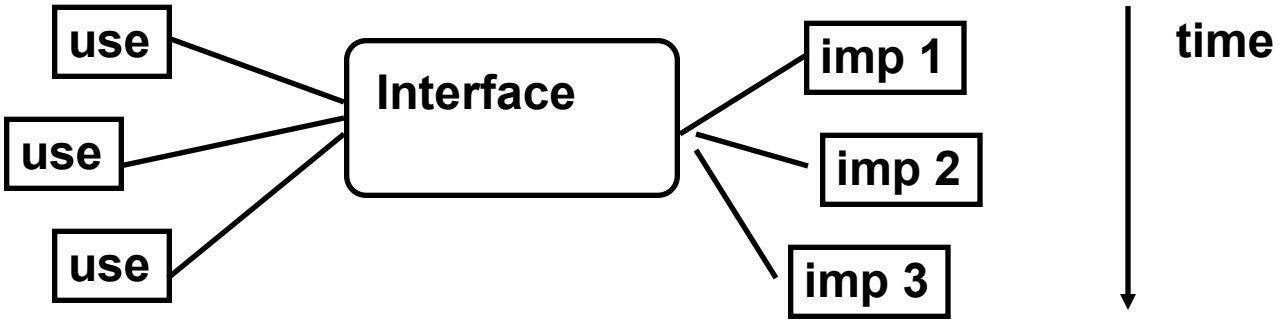
Instruction Set Architecture (ISA)



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

- 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?

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?

- 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?

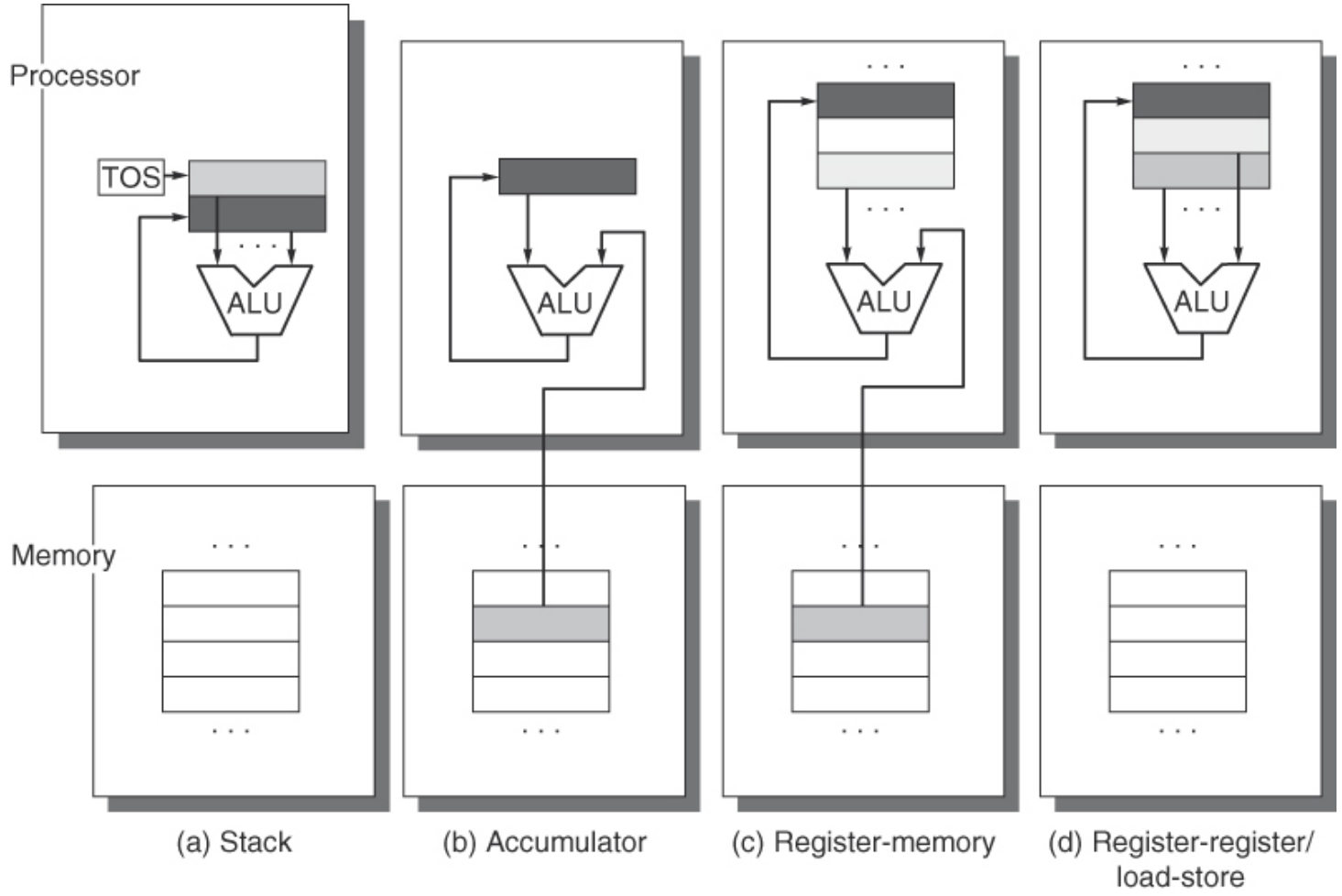
Instructions Can Be Divided into 3 Classes

- 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

ISA Classification

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

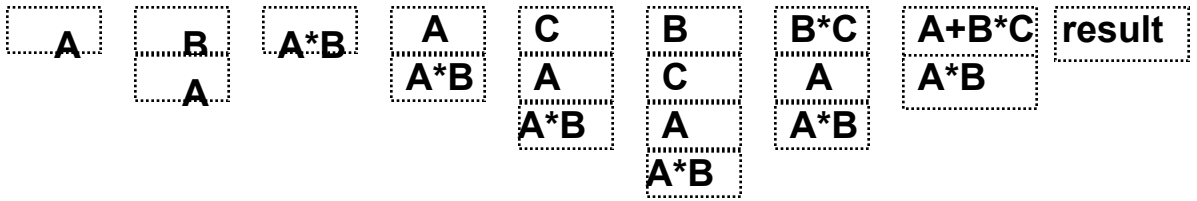
Basic Machine Organizations



Source: CA: A quantitative approach

Stack Architectures

- Instruction set:
  - add, sub, mult, div, . . .
  - push A, pop A
- Example: A*B - (A+C*B)
  - push A
  - push B
  - mul
  - push A
  - push C
  - push B
  - mul
  - add
  - sub



Stacks: Pros and Cons

- Pros
  - Good code density (implicit operand addressing → top of stack)
  - Low hardware requirements
  - Easy to write a simpler compiler for stack architectures
- Cons
  - Stack becomes the bottleneck
  - Little ability for parallelism or pipelining
  - Data is not always at the top of stack when need, so additional instructions like TOP and SWAP are needed
  - Difficult to write an optimizing compiler for stack architectures

Accumulator Architectures

- Instruction set:

  add A, sub A, mult A, div A, . . .

  load A, store A

- Example: A*B - (A+C*B)

  load B

  mul C

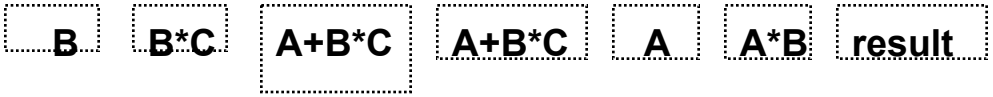
  add A

  store D

  load A

  mul B

  sub D



Thank You