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 2 (23 July 2014)

CADSL

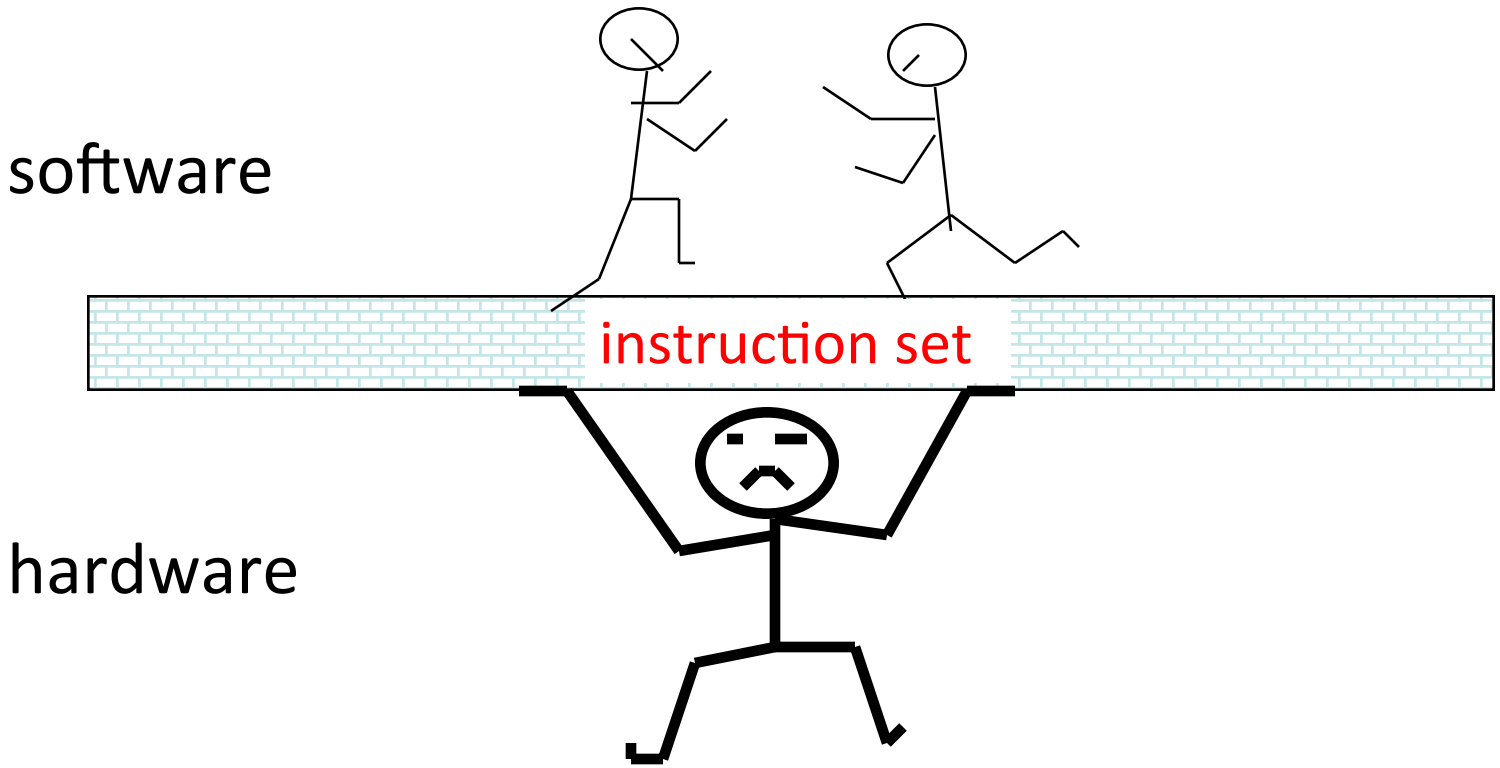
INSTRUCTION SET ARCHITECTURE

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)



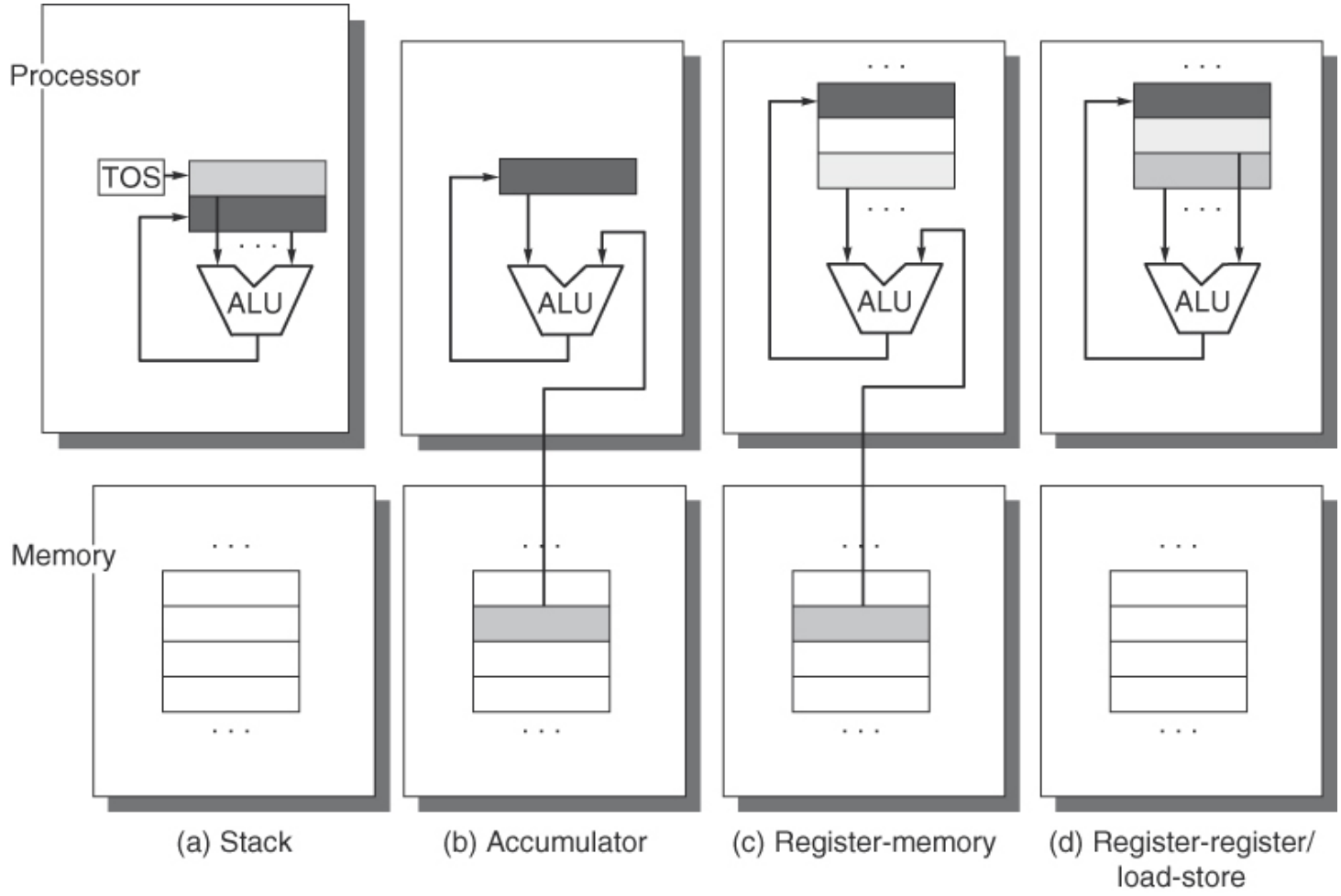
Instructions Can Be Divided into 3 Classes

- 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

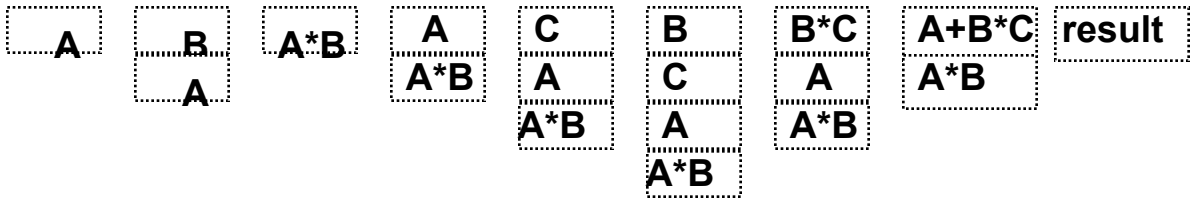
Basic Machine Organizations



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

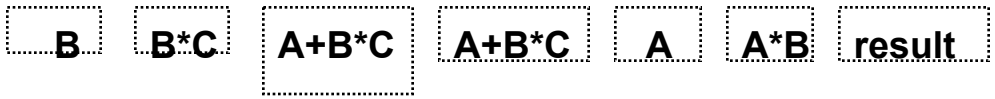


Stacks: Pros and Cons

- 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



Accumulators: Pros and Cons

- Pros
  - Very low hardware requirements
  - Easy to design and understand

- Cons
  - Accumulator becomes the bottleneck
  - Little ability for parallelism or pipelining
  - High memory traffic

Memory-Memory Architectures

- Instruction set:

  (3 operands) add A, B, C

  sub A, B, C mul A, B, C

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

    mul D, A, B

    mul E, C, B

    add E, A, E

    sub E, D, E

Memory-Memory: Pros and Cons

- Pros
  - Requires fewer instructions (especially if 3 operands)
  - Easy to write compilers for (especially if 3 operands)
- Cons
  - Very high memory traffic (especially if 3 operands)
  - Variable number of clocks per instruction (especially if 2 operands)
  - With two operands, more data movements are required

Register-Memory Architectures

- Instruction set:

  add R1, A sub R1, A mul R1, B

  load R1, A store R1, A

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

```
load R1, A
mul R1, B          /*   A*B          */
store R1, D
load R2, C
mul R2, B          /*   C*B          */
add R2, A          /*   A + CB           */
sub R2, D          /*   AB - (A + C*B)   */
```

Memory-Register: Pros and Cons

- Pros
  - Some data can be accessed without loading first
  - Instruction format easy to encode
  - Good code density

- Cons
  - Operands are not equivalent (poor orthogonality)
  - Variable number of clocks per instruction
  - May limit number of registers

Load-Store Architectures

- Instruction set:

  add R1, R2, R3 sub R1, R2, R3 mul R1, R2, R3

  load R1, R4 store R1, R4

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

```
load R4, A
load R5, B
load R6, C
mul R7, R6, R5        /*      C*B          */
add R8, R7, R4        /*    A + C*B        */
mul R9, R4, R5        /*      A*B          */
sub R10, R9, R8       /*  A*B - (A+C*B)    */
```

Load-Store: Pros and Cons

- Pros
  - Simple, fixed length instruction encoding
  - Instructions take similar number of cycles
  - Relatively easy to pipeline

- Cons
  - Higher instruction count
  - Not all instructions need three operands
  - Dependent on good compiler

Thank You