# Three Pillars of Computer Architecture-Advance Computer Architecture-Lecture Slides

Advanced Computer Architecture

Gujarat University

34 pag.

# Today's Topics

- Recap
- ISA Taxonomy
- Memory Addressing modes
- Types of operands
- Types of operations
- Summary

# Recap: Lec. 1-3 Chapter 1

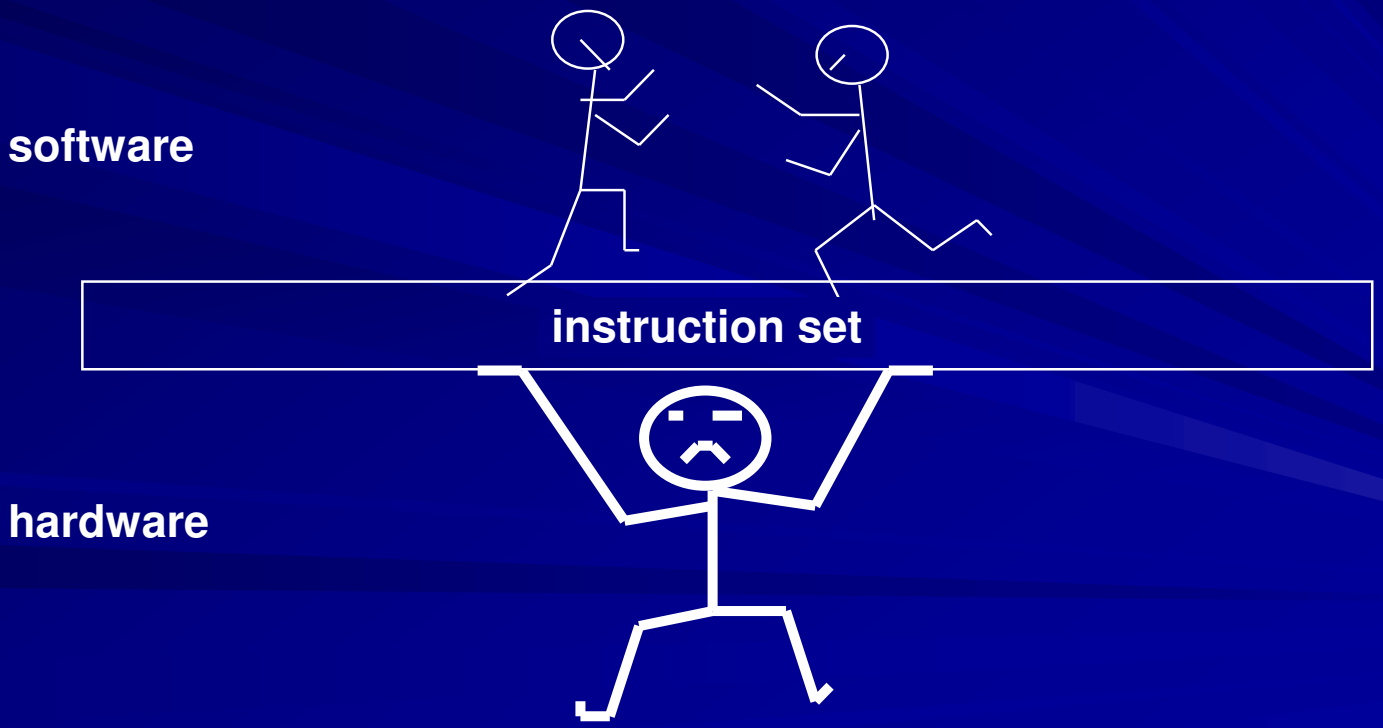
- Computer design cycle
- Performance metrics:

  *Processor and I/O systems*
- Price-Performance design
- Benchmarks: *Performance evaluation*
- Quantitative principles:

  *Performance enhancement*

# Changing Definitions of Computer Architecture

## Three Pillars of Computer Architecture



# Changing Definitions of Computer Architecture …….. Cont'd

- **1950s to 1960s:**

  **The focus of the Computer Architecture Courses has been Computer Arithmetic**

- **1970s to mid 1980s:**

  **The focus of Computer Architecture Course has been Instruction Set Design, the portion of the computer visible to programmer and compiler writer**

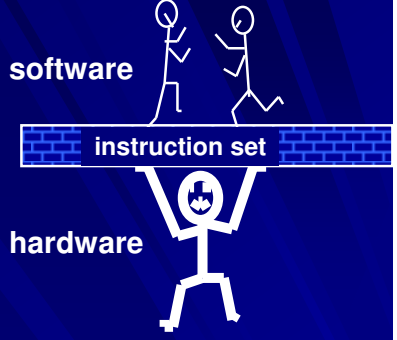
**….. Cont'd**

## Changing Definitions of Computer Architecture … Cont'd

- **1990s to date:**

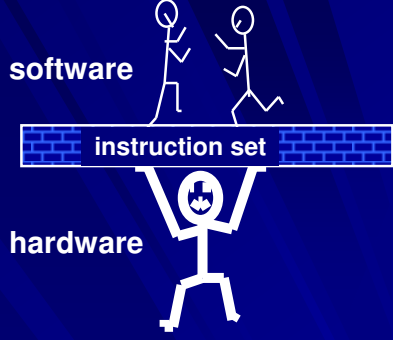
  **The focus of the Computer Architecture Course is the Design of CPU, memory system, I/O system, Multiprocessors based on the quantitative principles to have price - performance design; i.e., maximum performance at minimum price**

# Instruction Set Architecture – ISA



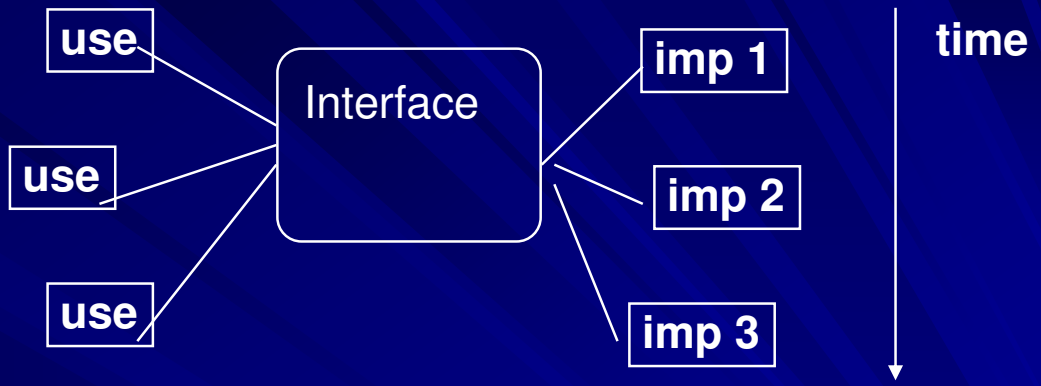

- **Our focus** in couple of lectures will be the **Instruction Set Architecture – ISA** which is the **interface** between the **hardware-software**
- It plays a vital role in understanding the computer architecture from any of the above mentioned perspectives

# Instruction Set Architecture – ISA




- The design of hardware and software can't be initiated without defining ISA
- It describes the instruction word format and identifies the memory addressing for data manipulation and control operations

# What is an 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

# Taxonomy of Instruction Set

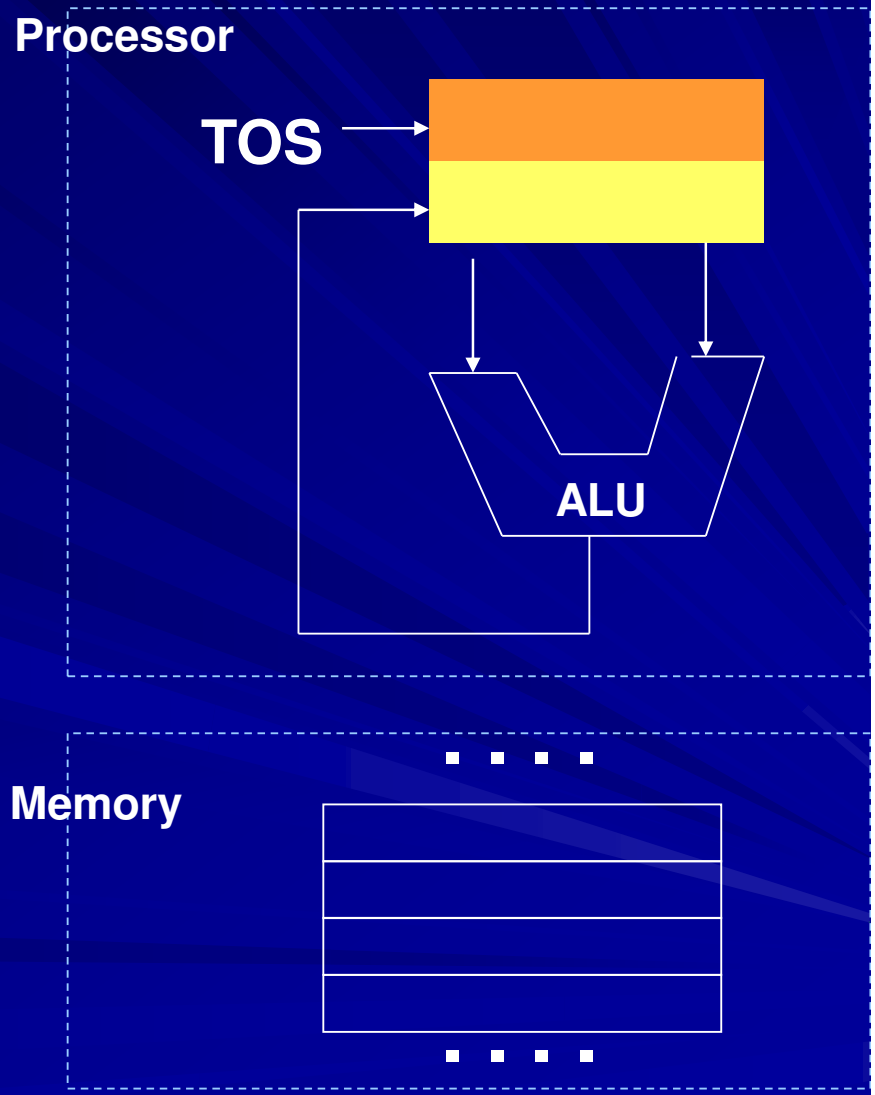
- **Major advances** in computer architecture are typically associated with landmark instruction set designs – **stack, accumulator, general purpose register etc.**
- **Design decisions** must take into account:
  - technology
  - machine organization
  - programming languages
  - compiler technology
  - operating systems

# Taxonomy of Instruction Set ….. Cont'd

- **Basic Differentiator:** The type of internal storage of the operand
- **Major Choices of ISA:**
  - Stack Architecture:
  - Accumulator Architecture
  - General Purpose Register Architecture
    - Register – memory
    - Register – Register (load/store)
    - Memory – Memory Architecture (Obsolete)

# Stack Architecture

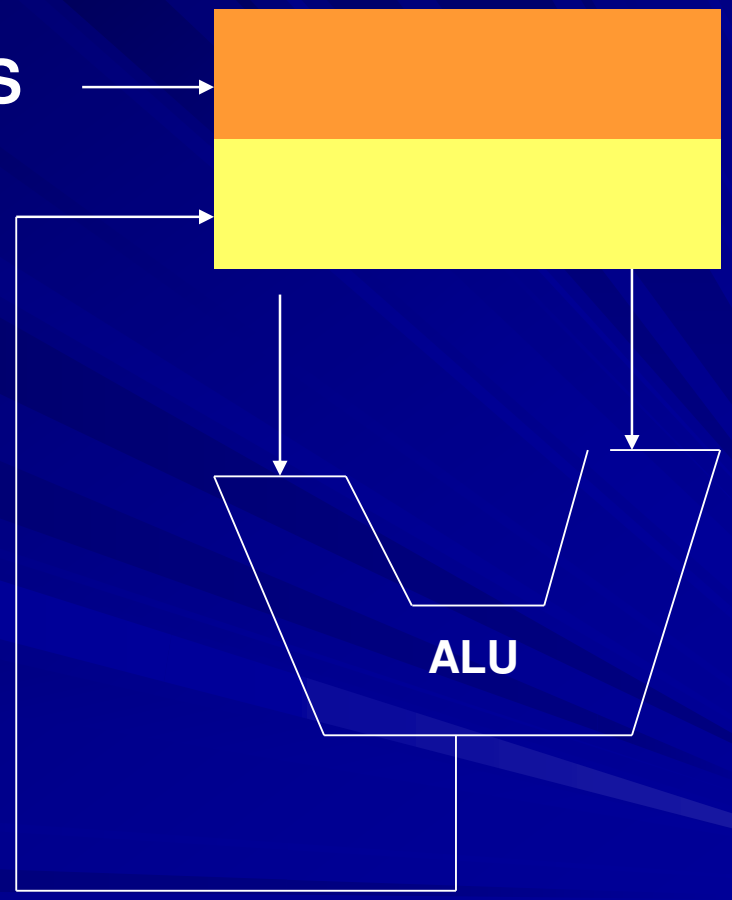
- Both the operands are implicitly on the TOS
- Thus, it is also referred to as Zero-Address machine
- The operand may be either an input (orange shade) or result from the ALU (yellow shade)
- All operands are implicit (implied or inherited)
- The first operand is removed from the stack and the second operand is replaced by the result



# Stack Architecture

**To execute: C=A+B**

**ADD instruction has implicit operands for the stack – operands are written in the stack using PUSH instruction**




**PUSH A**

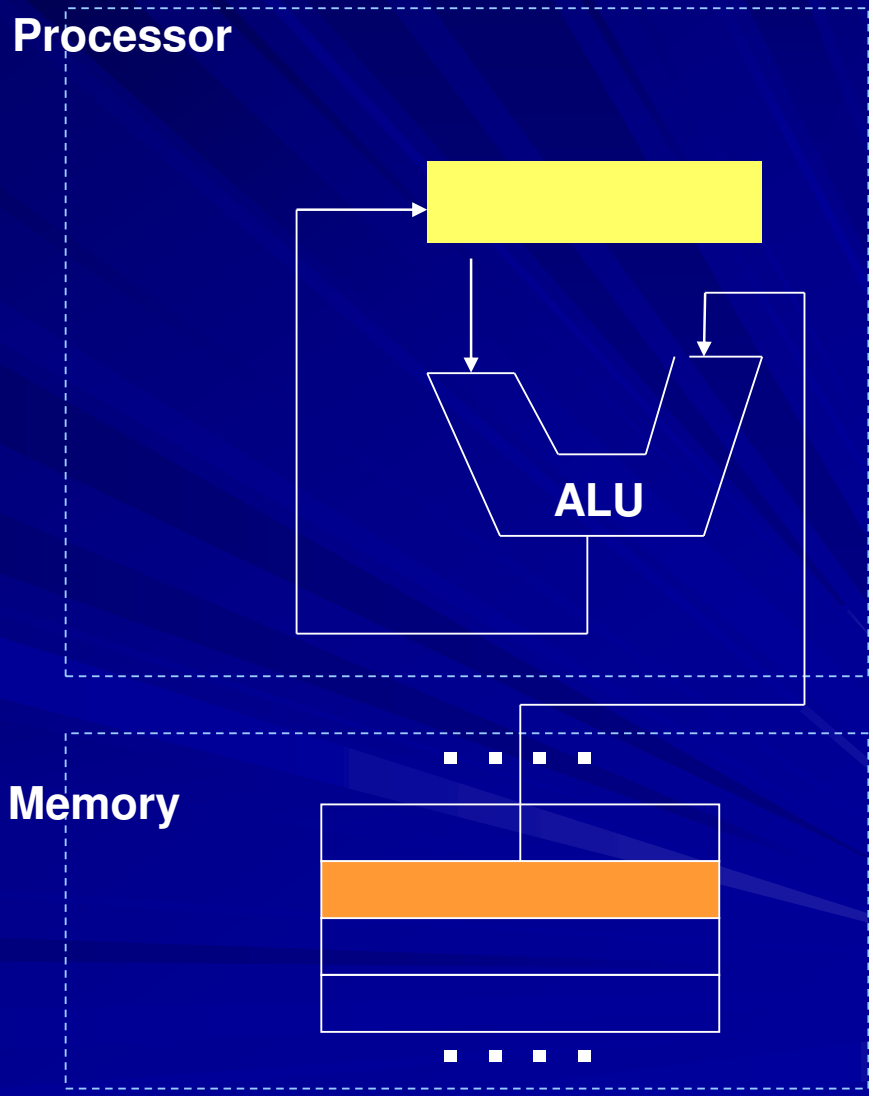
**PUSH B**

**ADD**

**POP C**

# Accumulator Architecture

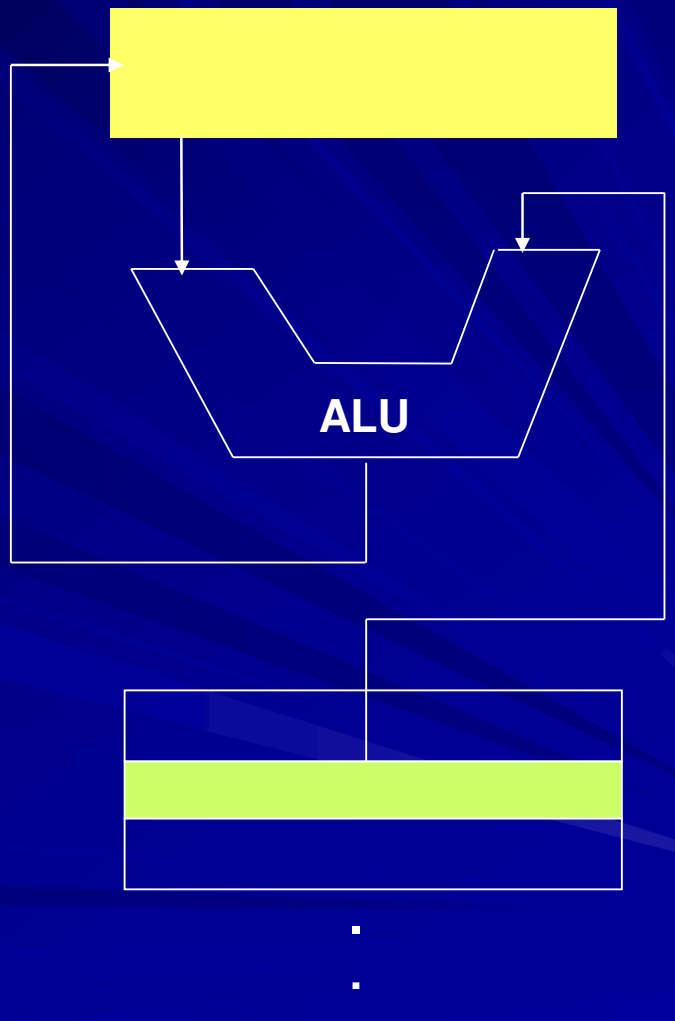
- An *accumulator* is a special register within the CPU that serves both as both the as the implicit source of one operand and as the result destination for arithmetic and logic operations.
- Thus, it accumulates or collect data and doesn't serve as an address register at any time
- Limited number of accumulators - usually only one – are used
- The second operand is in the memory, thus accumulator based machines are also called 1-address machines
- They are useful when memory is expensive or when a limited number of addressing modes is to be used



# Accumulator Architecture

**To execute: C=A+B**

ADD instruction has **implicit operand A** for the accumulator, written using **LOAD** instruction; and the **second operand B** is in memory at address B




**Load A**

**ADD B**

**Store C**

# General Purpose Register Architecture

- Many general purpose registers are available within CPU
- Generally, CPU registers do not have dedicated functions and can be used for a variety of purposes – address, data and control
- A relatively small number of bits in the instruction is needed to identify the register
- In addition to the GPRs, there are many dedicated or special-purpose registers as well, but many of them are not "visible" to the programmer
- GPR architecture has explicit operands either in register or memory thus there may exist:
  - Register – memory architecture
  - Register – Register (Load/Store) Architecture
  - Memory – Memory Architecture

Processor

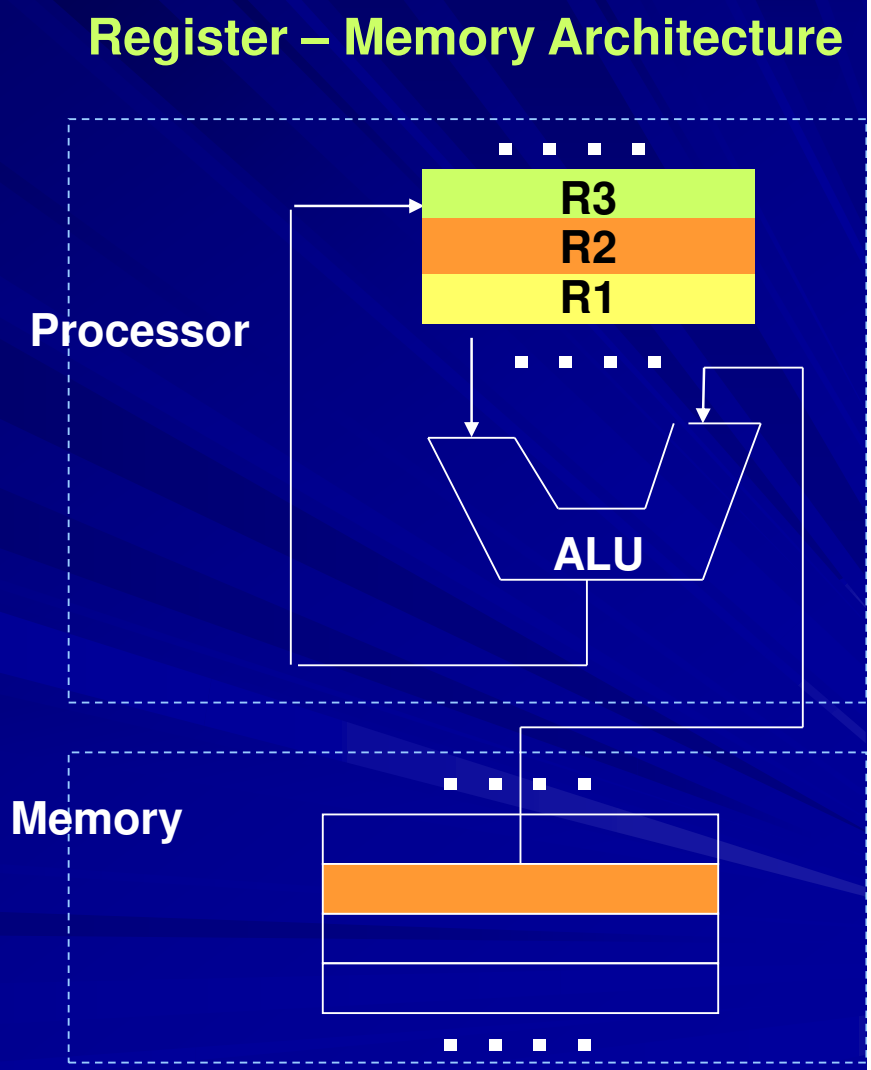
# General Purpose Register Architecture

- One explicit operand is in a register and one in memory and the result goes into the register
- The operand in memory is accessed directly

**To execute: C=A+B**

ADD instruction has explicit operand A loaded in a register and the operand B is in memory and the result is in register

```
Load  R1, A
ADD   R3, R1, B
Store R3, C
```



# General Purpose Register Architecture

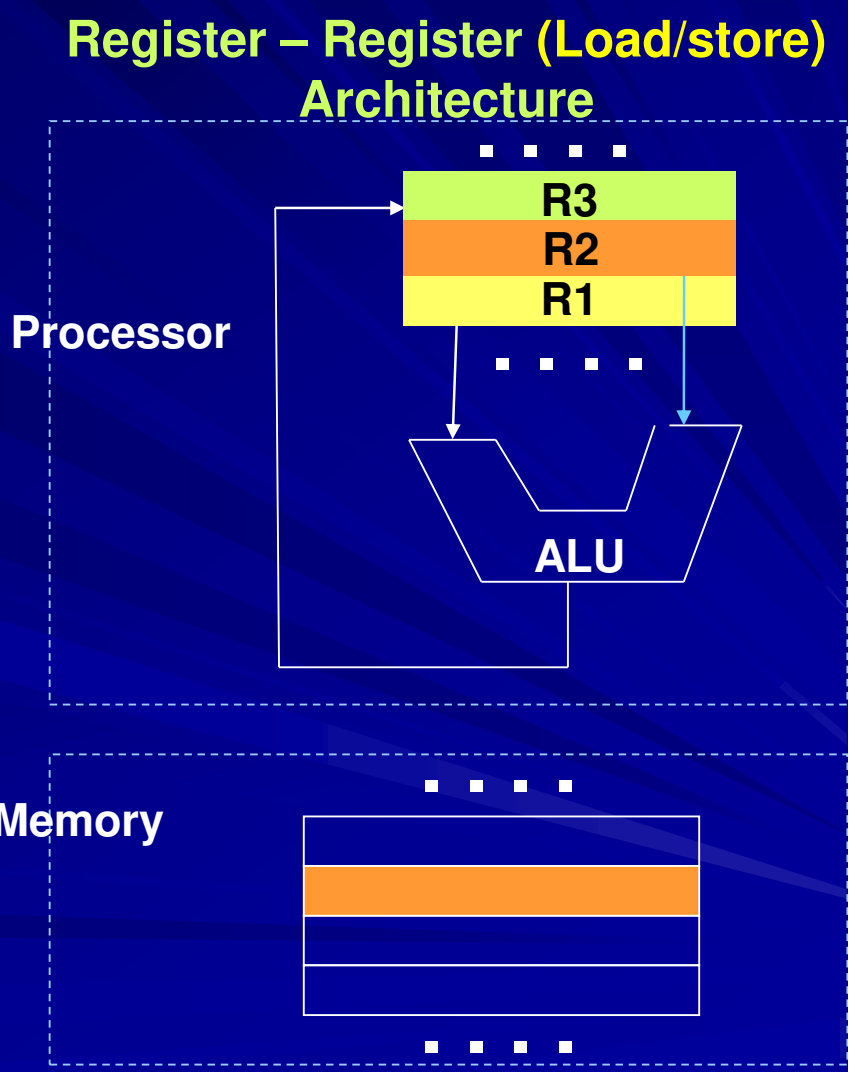
- The explicit operands in memory are first loaded into registers temporarily and
- Are transferred to memory by Store instruction

To execute: C=A+B

ADD instruction has implicit operands A and B loaded in registers

Load R1, A
Load R2, B
ADD R3, R1, R2
Store R3, C

- Both the explicit operands are not accessed from memory directly, i.e., Memory – Memory Architecture is obsolete



# Comparison of three GPR Architectures

## Register-Register

### Advantages

- Simple, fixed-length instruction decoding
- Simple code generation
- Similar number of clock cycles / instruction

### Disadvantages

- Higher Instruction count than memory reference
- Lower instruction density leads to larger programs

# Comparison of three GPR Architectures

- **Register- Memory**

## Advantages

- Data can be accessed without separate Load first
- Instruction format is easy to encode

## Disadvantages

- Operands are not equivalent since a source operand (in a register) is destroyed in operation
- Encoding a register number and memory address in each instruction may restrict the number of registers
- CPI vary by operand location

# Comparison of three GPR Architectures

- **Memory- Memory**

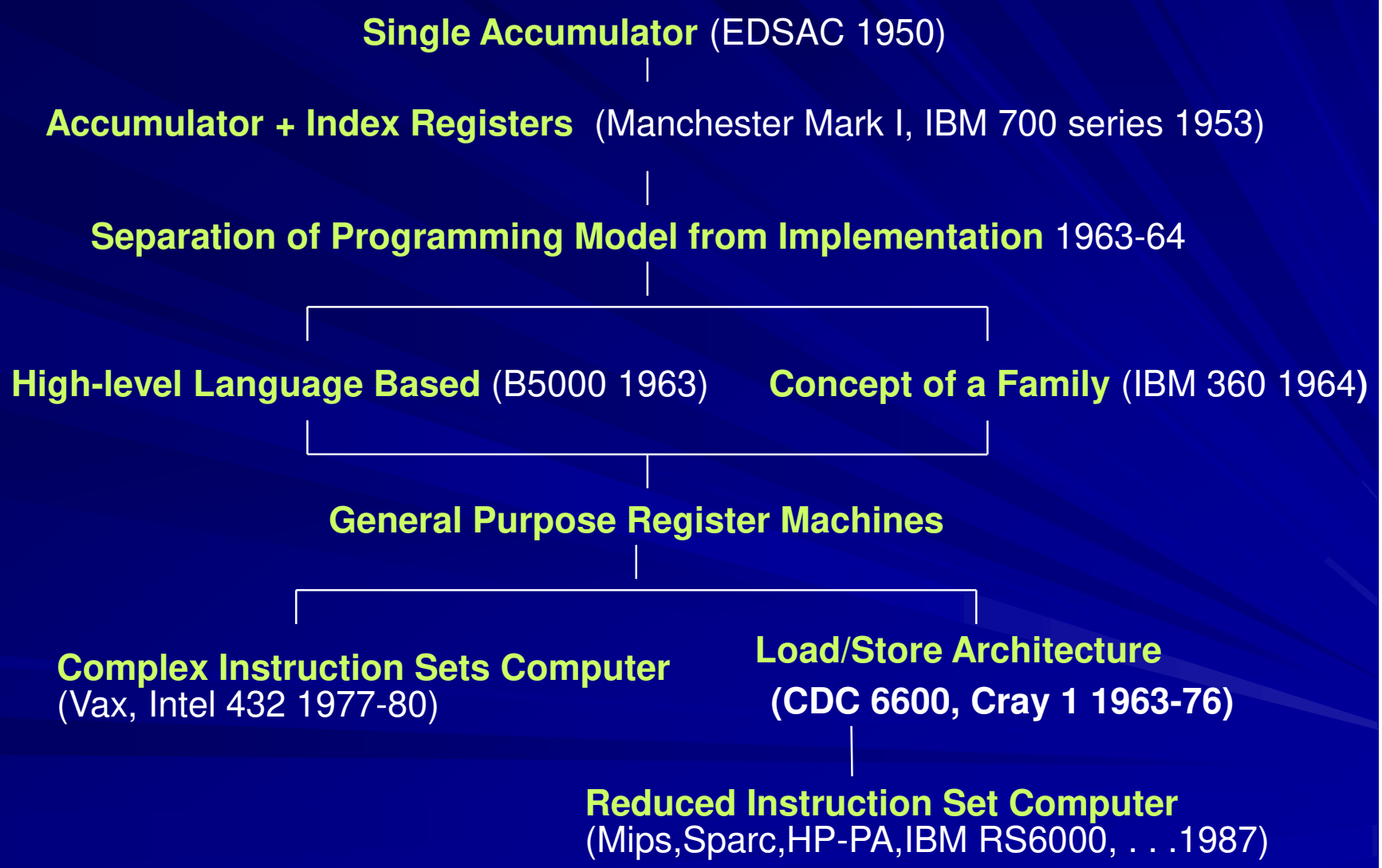
**Advantages**

- **Most compact**
- **Doesn't waste registers for temporary storages**

**Disadvantages**

- **Large variation in instruction size**
- **Large variation in work per instruction**
- **Memory bottleneck by memory access**

# Evolution of Instruction Sets



# Types and Size of Operands

- **Types of an Operand**
  - Integer
  - Single-precision floating point
  - Character

- **Size of Operand**

| | |
|---|---|
| - Character | 8-bit |
| - Half word | 16-bit |
| - Single precision FP or Word | 32-bit |
| - Double precision FP or double word | 64-bit |

# Categories of Instruction Set Operations

All computer provide a full set of following operational instructions for:

- **Arithmetic and Logic**
  - Integer add, sub, and, or, multiply, divide
- **Data Transfer**
  - Load, store and
  - Move instructions with memory addressing
- **Control**
  - Branch, Jump, procedure call and return

# Categories of Instruction Set Operations … Cont'd

**The following support instructions may be provided in computer with different levels**

- **System**
  - operating system call, Virtual Memory Management
- **Floating point**
  - Add, multiply, divide and compare
- **Decimal**
  - BCD add, multiply and Decimal to Character Conversion
- **String**
  - String move, compare and search
- **Graphics**
  - Pixel and vertex operations, compression / de-compression operations

# Operand Addressing Modes

- An **"effective address"** is the binary bit pattern issued by the CPU to specify the location of operands in CPU (register) or the memory

- Addressing modes are the ways of providing access paths to CPU registers and memory locations

- **Commonly used addressing modes are:**
  - **Immediate**
  - **Register**
  - **Direct or Absolute**
  - **Indirect**

# Operand Addressing Modes

- **Immediate** ADD R4, # 24H Reg[R4] ← Reg[R4] + 24 H

- Data for the instruction is part of the instruction itself
- Used to hold source operands only; cannot be used for storing results

- **Register** ADD R4, R3 Reg[R4] ← Reg[R4] + Reg[R3]

- Operand is contained in a CPU register
- No memory access needed , therefore it is fast

- **Direct** (or absolute) ADD R1,(1000) Reg[R1] ← Reg[R1] + Mem[1000]

- The address of the operand is specified as a constant, coded as part of the instruction
- Limited address space ($2^{\text{operand field size}}$) locations

# Commonly used addressing modes ... cont'd

## Indirect Addressing modes

- The address of the memory location where the data is to be found is stored in the instruction as the operand, i.e., the operand is the address of an address
- Large address space ( $2^{\text{memory word size}}$ ) available
- Two or more memory accesses are required

# Commonly used addressing modes … cont'd

## Types of Indirect addressing modes:

- **Register Indirect**
- **Register Indirect Indexed**

- Effective memory address is calculated by adding another register (index register) to the value in a CPU register (usually referred to as the base register)
- Useful for accessing 2-D arrays

- **Register Indirect plus displacement**

- Similarly, "based" refers to the situation when the constant refers to the offset (displacement) of an array element with respect to the first element. The address of the first element is stored in a register

- **Memory Indirect**

# Commonly used addressing modes ... cont'd

## Meanings of Indirect Addressing Modes

- Register Indirect

  ADD R4, (R1)  Reg[R4] ← Reg[R4] + Mem[Reg[R1]]

- Register Indirect Indexed

  ADD R4, (R1+R2)  Reg[R4] ← Reg[R4] + Mem[Reg[R1]+Reg[R2]]

- Register Indirect plus displacement

  ADD R4,100(R1)  Reg[R4] ← Reg[R4] + Mem[100+Reg[R1]]

- Memory Indirect

  ADD R4,@(R1)  Reg[R4] ← Reg[R4] + Mem[Mem[Reg[R1]]

# Special Addressing Modes

**Used for stepping within loops; R2 points to the start of the array; each reference increments / decrements R2 by 'd'; the size of the elements in the array**

- Auto-increment ADD R1, (R2)+

  (i) Reg[R1] ← Reg[R1] + Mem[Reg [R2]]

  (ii) Reg[R2] ← Reg[R2] + d

- Auto-decrement ADD R1, (R2)-

  (i) Reg[R2] ← Reg[R2] - d

  (ii) Reg[R1] ← Reg[R1] + Mem[Reg [R2]]

- Scaled ADD R1, 100(R2)[R3]

  Reg[R1] ← Reg[R1] +

  Mem[100+Reg [R2] + R3]*d]

# Addressing Modes of Control Flow Instructions

- **Branch (conditional)**

  a sort of displacement, in number of instructions, relative to PC

- **Jump (Unconditional)**

  jump to an absolute address, independent of the position of PC

- **Procedure call/return**

  control transfer with some state and return address saving, some times in a special link register or in some GPRs

# Summary

- **ISA Taxonomy**
  - Stack Architecture:
  - Accumulator Architecture
  - General Purpose Register Architecture
    - Register – memory
    - Register – Register (load/store)
    - Memory – Memory Architecture (Obsolete)

# Summary .. Cont'd

- Memory Addressing modes
  - Immediate
  - Register
  - Direct or Absolute
  - Indirect
  - Special
  - Control Flow Instruction

# Thank You
# and
# Allah Hafiz