EC-252: COMPUTER ARCHITECTURE AND MICROPROCESSORS

Vaskar Raychoudhury

Indian Institute of Technology Roorkee

Computer Organization

- Organization of a Computer is the layout of the following 5 fundamental functional modules
  - Input
  - Output
  - Memory
  - Data path
  - Control

Processor Organization

- CPU mainly consists of the following components
  - Registers
  - Arithmetic and Logic Unit (ALU)
  - Internal buses
  - Control Unit

Registers

- CPU must have some working space (temporary storage)
  - Called registers
- Number and function vary between processor designs
- Top level of memory hierarchy
  - Registers » Cache » Main Memory » Secondary memory
- Two types
  - User visible registers vs. Control and status registers

User Visible Registers

- Visible and used by machine and assembly language programmers
- Usage objective is to minimize main memory references
- Types of user visible registers
  - General Purpose
  - Data
  - Address
  - Condition Codes

General Purpose Registers (1)

- May be true general purpose
  - May be restricted
  - May be used for data or addressing
- Data
  - Accumulator
- Addressing
  - Segment

Segment and Segment Pointers

- Segment Pointers
  - In a CPU with segmented addressing, a register holds the address of the base of a segment.
  - There may be multiple segment registers. E.g., one for the operating system, one for the current process.
- Intel 8086 family has different segments and they are referred to as
  - **code segment:** where the code is written
  - **data segment:** where the data is placed
  - **stack segment:** acts like a stack
  - **extra segment**

General Purpose Registers (2)

- Make them general purpose
  - Increase flexibility and programmer options
  - Increase instruction size & complexity
- Make them specialized
  - Smaller (faster) instructions
  - Less flexibility

How Many GP Registers?

- Between 8 – 32
- Fewer = more memory references
- More does not reduce memory references and takes up processor real estate

How big?

- Large enough to hold full address
- Large enough to hold full word
- Often possible to combine two data registers
    - C programming
    - double int a;
    - long int a;

Condition Code Registers

- Sets of individual bits
  - e.g. result of last operation was zero
- Can be read (implicitly) by programs
  - e.g. Jump if zero
- Can not (usually) be set by programs

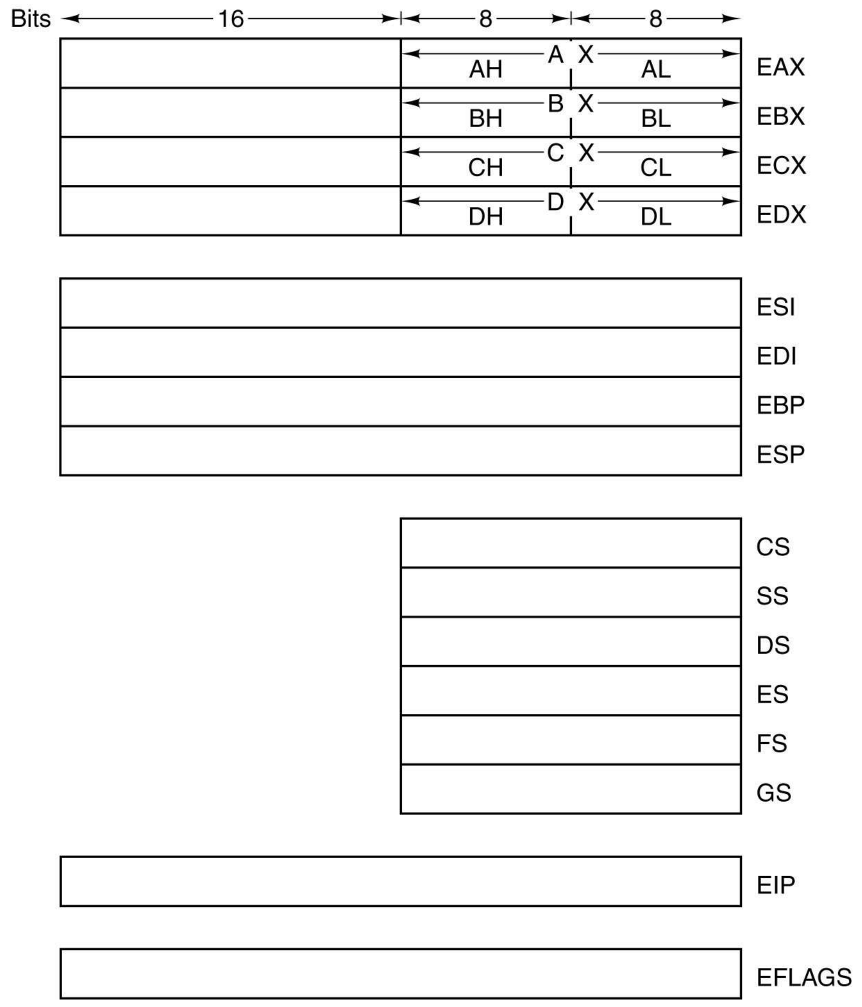
Control & Status Registers

- These registers are employed to control the operation of the CPU.
- Four registers are essential to instruction execution:
  - **Program Counter (PC):** It contains the address of an instruction to be fetched
  - **Instruction Register (IR):** It contains the instruction most recently fetched
  - **Memory Address Register (MAR):** It contains the address of a location in memory
  - **Memory Buffer Register (MBR):** It contains a word of data to be written to memory or the word most recently read

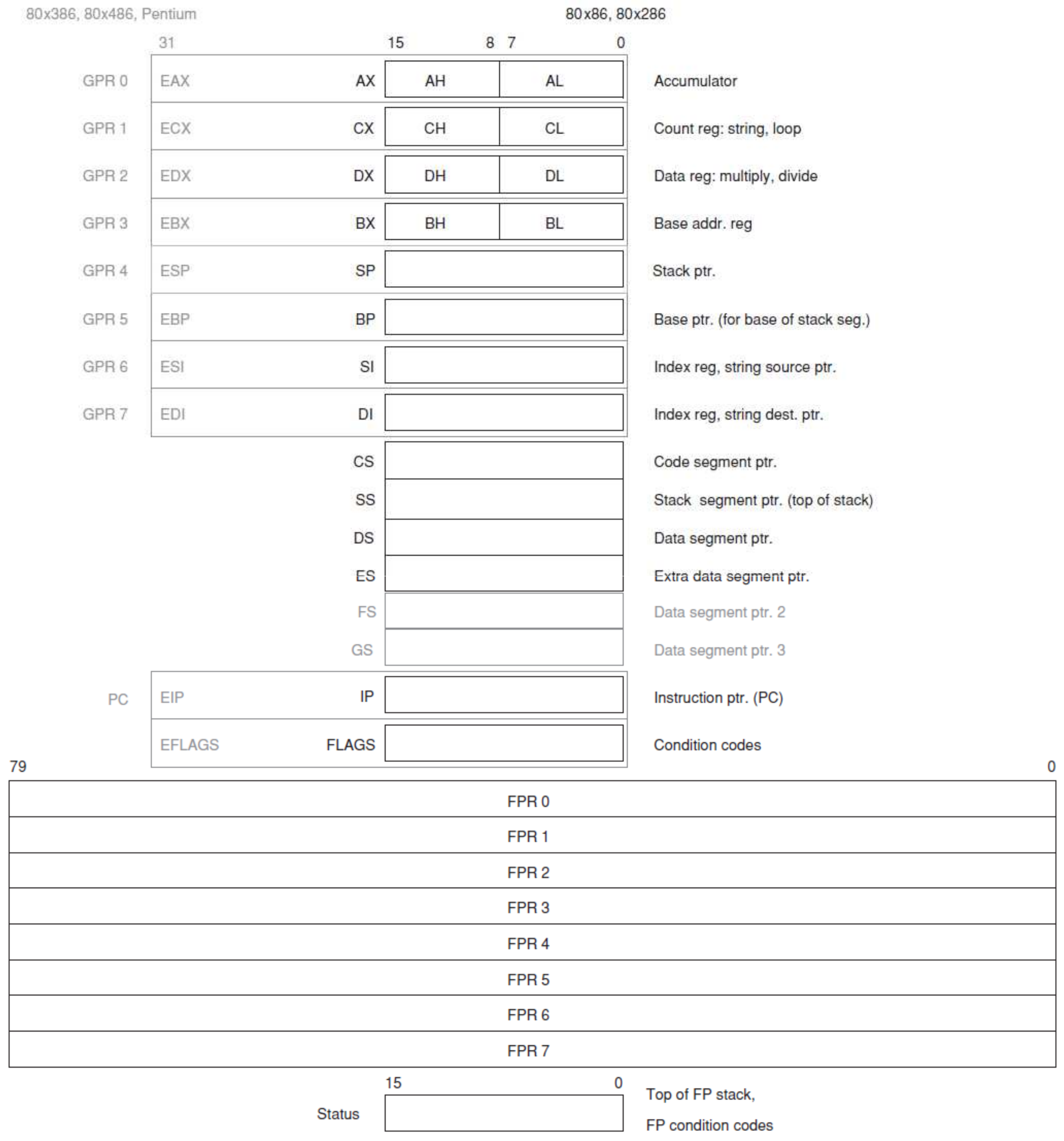
Program Status Word

- It contains status information.
- PSW typically contains condition codes plus other status information. Some of these may be user visible.
- Common flags include:
  - **Sign:** contains the sign bit of the result of the last arithmetic operation.
  - **Zero:** set when the result is 0.
  - **Carry:** set if an operation resulted in a carry (addition) into or borrow (subtraction) out of the high-order bit.
  - **Equal:** set if logical compare result is equality.
  - **Overflow:** used to indicate arithmetic overflow.
  - **Interrupt enable/disable:** used to enable or disable interrupts.
  - **Supervisor:** Indicates whether the CPU is executing in supervisor mode or user mode. Certain privilege instructions can be executed only in supervisor mode (e.g. halt instruction), and certain areas of memory can be accessed only in supervisor mode.

Example Register Organization: Pentium 4



The Pentium 4's primary registers.


80x386, 80x486, Pentium
80x86, 80x286
31
15
8
7
0
GPR 0
EAX
AX
AH
AL
Accumulator
GPR 1
ECX
CX
CH
CL
Count reg: string, loop
GPR 2
EDX
DX
DH
DL
Data reg: multiply, divide
GPR 3
EBX
BX
BH
BL
Base addr. reg
GPR 4
ESP
SP
Stack ptr.
GPR 5
EBP
BP
Base ptr. (for base of stack seg.)
GPR 6
ESI
SI
Index reg, string source ptr.
GPR 7
EDI
DI
Index reg, string dest. ptr.
CS
Code segment ptr.
SS
Stack segment ptr. (top of stack)
DS
Data segment ptr.
ES
Extra data segment ptr.
FS
Data segment ptr. 2
GS
Data segment ptr. 3
PC
EIP
IP
Instruction ptr. (PC)
EFLAGS
FLAGS
Condition codes
79
0
FPR 0
FPR 1
FPR 2
FPR 3
FPR 4
FPR 5
FPR 6
FPR 7
15
0
Status
Top of FP stack,
FP condition codes

What is a Bus?

- A communication pathway connecting two or more devices
- Usually broadcast
- Often grouped
  - A number of channels in one bus
  - e.g. 32 bit data bus is 32 separate single bit channels
- Power lines may not be shown

Data Bus

- Carries data
  - Remember that there is no difference between “data” and “instruction” at this level
- Width is a key determinant of performance
  - 8, 16, 32, 64 bit

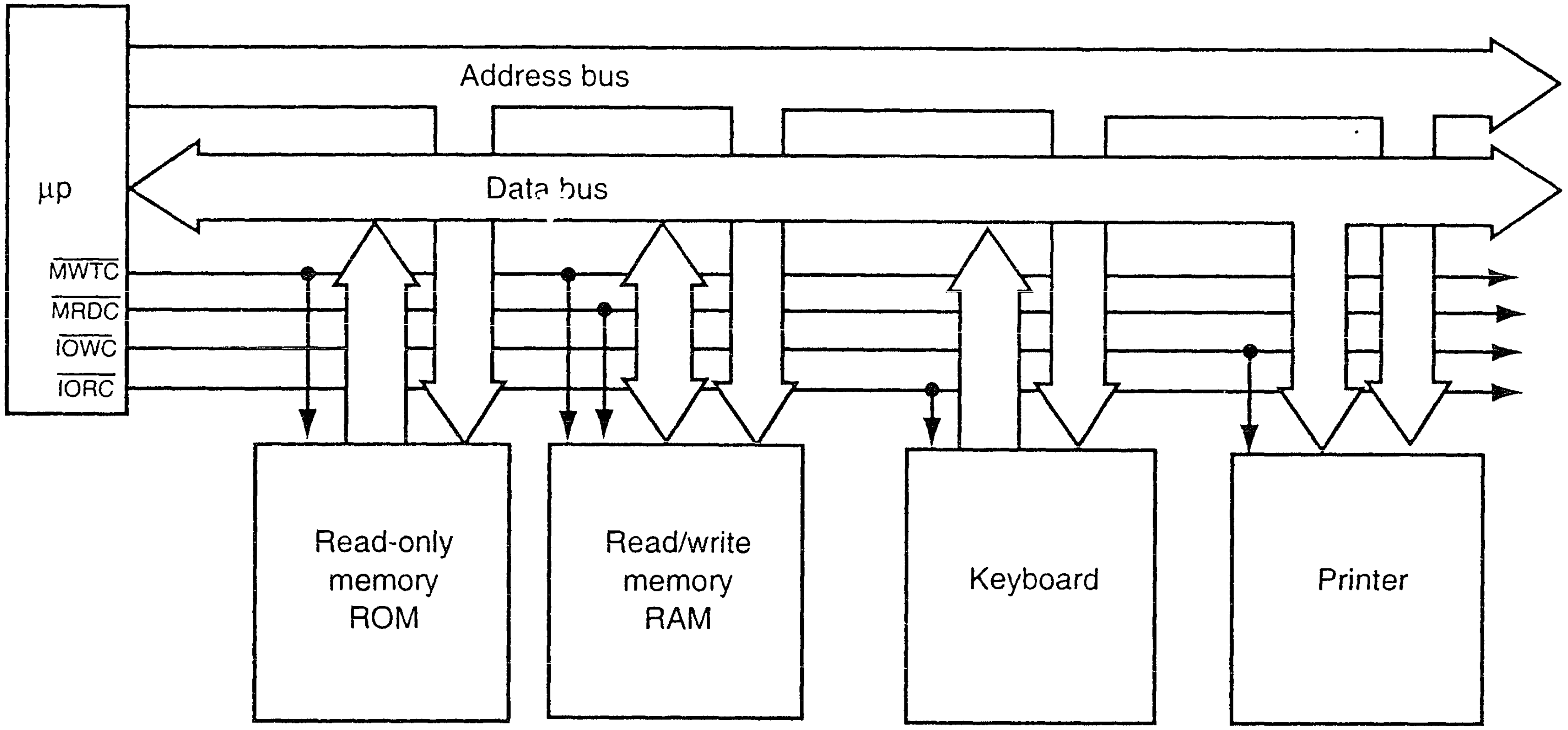
Address bus

- Identify the source or destination of data
- e.g. CPU needs to read an instruction (data) from a given location in memory
- Bus width determines maximum memory capacity of system
  - e.g. 8080 has 16 bit address bus giving 64k address space

Control Bus

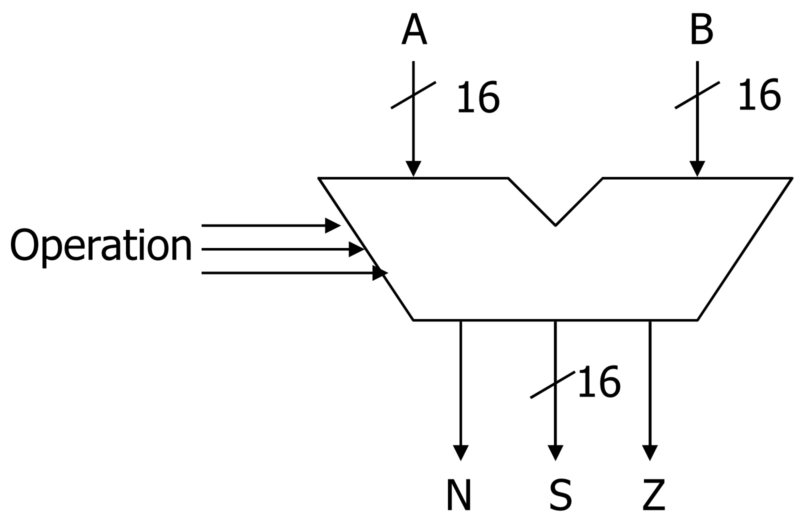
- Control and timing information
  - Memory read/write signal
  - Interrupt request
  - Clock signals

Bus Structure and Interconnection



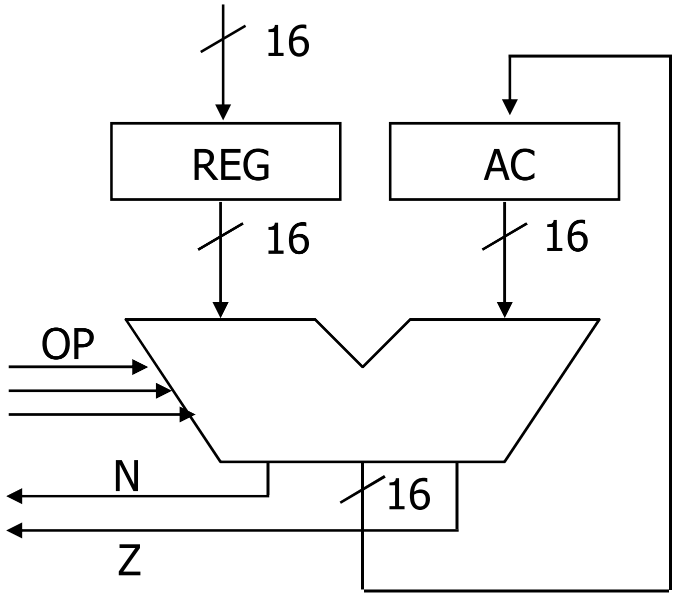
Data Path (ALU)

- ALU Block Diagram
  - Input: data and operation to perform
  - Output: result of operation and status information



Data Path (ALU + Registers)

- Accumulator
  - Special register
  - One of the inputs to ALU
  - Output of ALU stored back in accumulator
- One-address instructions
  - Operation and address of one operand
  - Other operand and destination is accumulator register
  - AC <– AC op Mem[addr]
  - ”Single address instructions” (AC implicit operand)
- Multiple registers
  - Part of instruction used to choose register operands

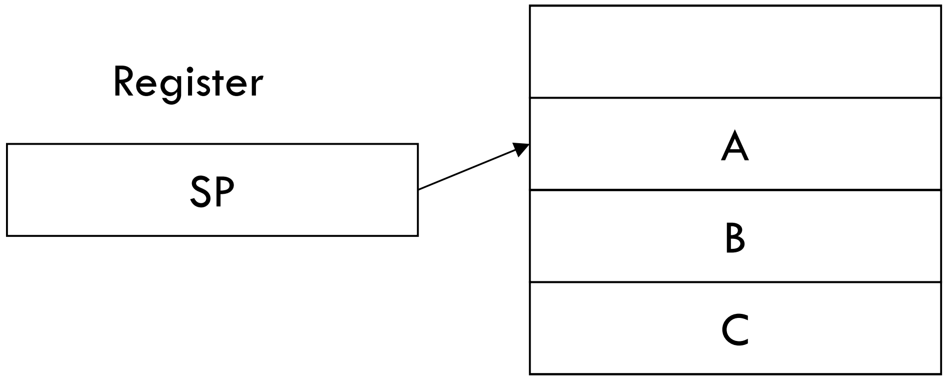


Zero Address Formats

- Operands on a stack

add	M[sp-1] $\leftarrow$ M[sp] + M[sp-1]
load	M[sp] $\leftarrow$ M[M[sp]]

  - Stack can be in registers or in memory (usually top of stack cached in registers)



Variety of Instruction Formats

- *One address formats:* Accumulator machines
  - Accumulator is always other source and destination operand

LOAD	x	$AC \leftarrow M[x]$
STORE	x	$M[x] \leftarrow (AC)$
ADD	x	$AC \leftarrow (AC) + M[x]$

- *Two address formats:* the destination is same as one of the operand sources

(Reg $\times$ Reg) to Reg	$R_I \leftarrow (R_I) + (R_J)$
(Reg $\times$ Mem) to Reg	$R_I \leftarrow (R_I) + M[x]$

  - *x* can be specified directly or via a register
  - effective address calculation for *x* could include indexing, indirection, ...

Variety of Instruction Formats

- *Three address formats:* One destination and up to two operand sources per instruction

(Reg x Reg) to Reg	$R_I \leftarrow (R_J) + (R_K)$
(Reg x Mem) to Reg	$R_I \leftarrow (R_J) + M[x]$

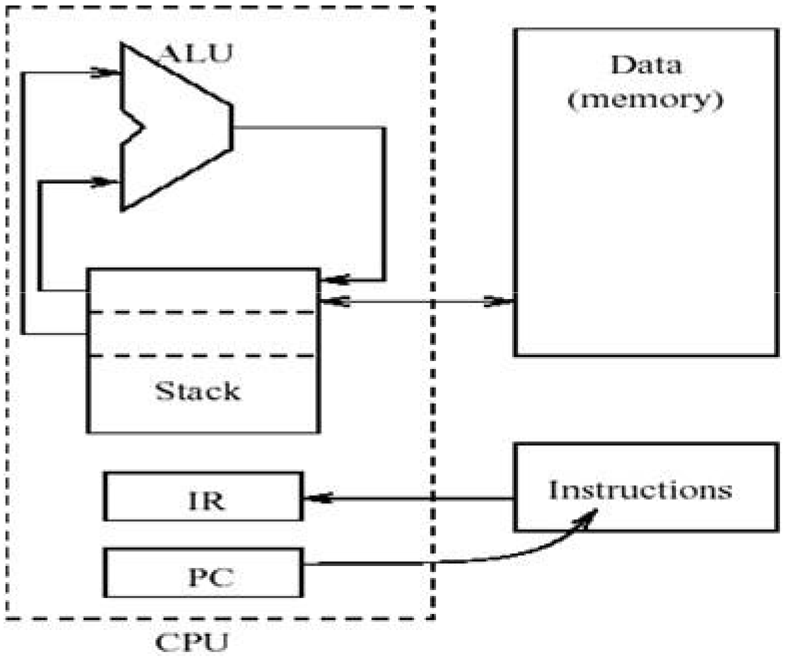
Fundamental Computer Architectures

- Most common Computer Architectures, all of which use *stored program control concept.*
- The three most popular computer architectures are:
  - The Stack Machine
  - The Accumulator Machine
  - The Load / Store Machine

The Stack Machine

- A stack machine implements a stack with registers
- The operands of the ALU are always the top two registers of the stack, and
- the result from the ALU is stored in the top register of the stack
- Examples of the stack machine include Hewlett-Packard RPN calculators and the Java Virtual Machine (JVM)

Structure of Stack Machine



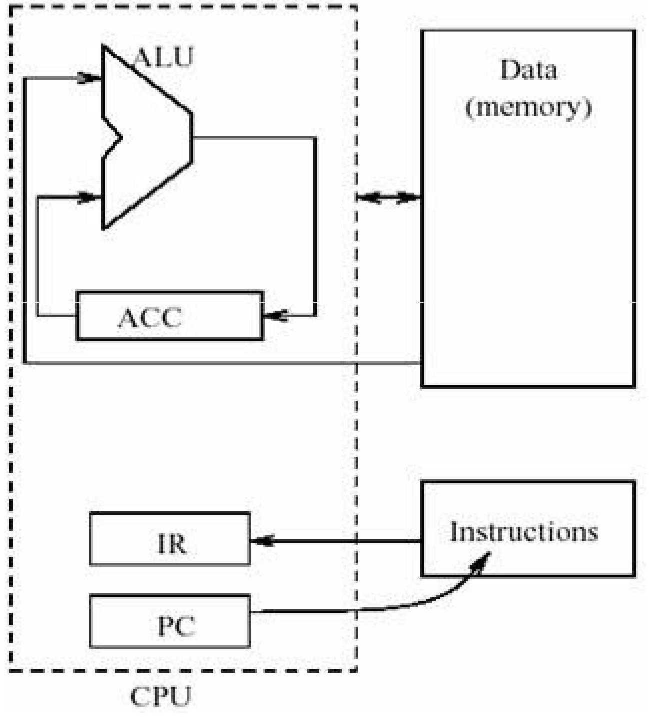
Why to Use Stack Machine?

- The advantage of a stack machine is
  - it can shorten the length of instructions since operands are implicit
- This was important when memory was expensive (20–30 years ago)
- Now, in Java, it is important since we want to ship executables (class files) over the network

The Accumulator Machine

- An accumulator machine has a special register, called an *accumulator*, whose contents are combined with another operand as input to the ALU, with the result of the operation replacing the contents of the accumulator.
- accumulator = accumulator [op] operand

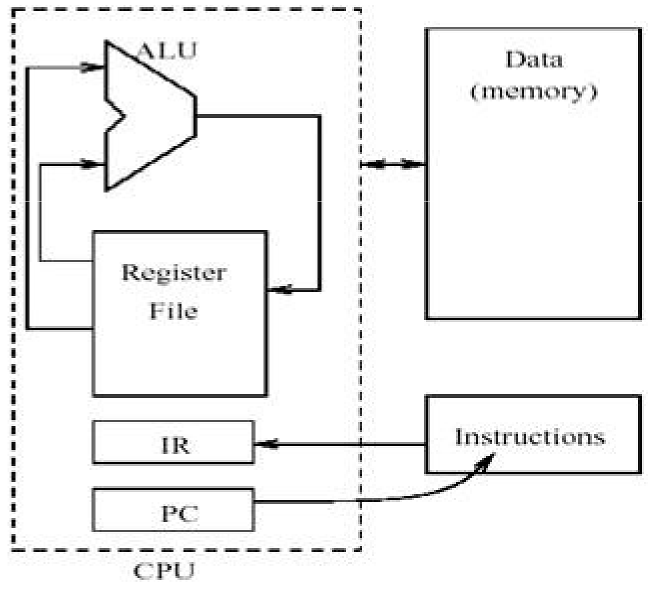
Structure of Accumulator Machine



The Load/Store Machine

- **Registers:** Provide faster access but are expensive
- **Memory:** Provides slower access but is less expensive
- A small amount of high speed memory (expensive), called a *register file*, is provided for frequently accessed variables and a much larger but slower memory (less expensive) is provided for the rest of the program and data. (SPARC: 32 register at any one time)
- This is based on the principle of "locality" – at a given time, a program typically accesses a small number of variables much more frequently than others.

Structure of Load/Store Machine



The Load/Store Machine

- The machine loads and stores the registers from memory. The arithmetic and logic instructions operate with registers, not main memory, for the location of operands.