Unit-I

Basic computer Organization

D.Venkata Vara Prasad
Professor
Dept. of Computer Science & Engg.
SSNCE

Objectivies

- What is CA?
- Why CA ?
- The Big Picture
- Performance

CA…

- **What is a computer?**

A *Computer* is a machine which solves problems for people written as programs.

It is a Electronic data processing machine which

takes data as input
processes it,
performs necessary calculations ,
stores the results
and outputs the results as when required.

- **What is a Program?**

A *Program* is a sequence of statements/steps stating how to perform a task. For each step an arithmetic and logical operation is done. For each operation a different set of control signals is needed – i.e. an instruction.
An *instruction* can be machine language instructions
or assembly language instructions or even high level language instructions.

What is Hardware?

- A *hardware* is something that is tangible.
- For e.g. CPU, Memory, I/O devices, Bus etc.

What is Software?

- **A *software* is a collection of programs.**

- **S/W – system S/w & Appl s/w**
- **System s/w-**
  - **Ser of pgm designed to control the operation and extend the capability of computer system**
  - **compilers, linkers, loaders, assemblers**
- **Appl s/w –**
  - **It is a set of pgm designed to solve a specific problem**
  - **Banking s/w, payroll s/w**

What is firmware?

- **A *firmware* is software embedded in hardware during manufacture.**
  E.g. home appliances, etc

What is CO?

- **It deals with how features are implemented.**
- **Hidden from Programmer**
- **Information flow between components**

  Example:

  1. **Control signals**
  2. **Memory technology**
  3. **Interfaces**
- **It basically discuss how h/w components operates and how they are connected to form computer system.**

Need for Computers

- Automatic
- Speed
- Accuracy
- Diligence
- Versatility

What is “Computer Architecture”

Computer Architecture =

Instruction Set Architecture + Machine Organization

What is Computer Architecture…

- **is concerned with the structure and behavior of the computer as seen by the user/programmer. It includes attributes such as**

– Instruction Formats

– Addressing Modes

– Instruction Sets

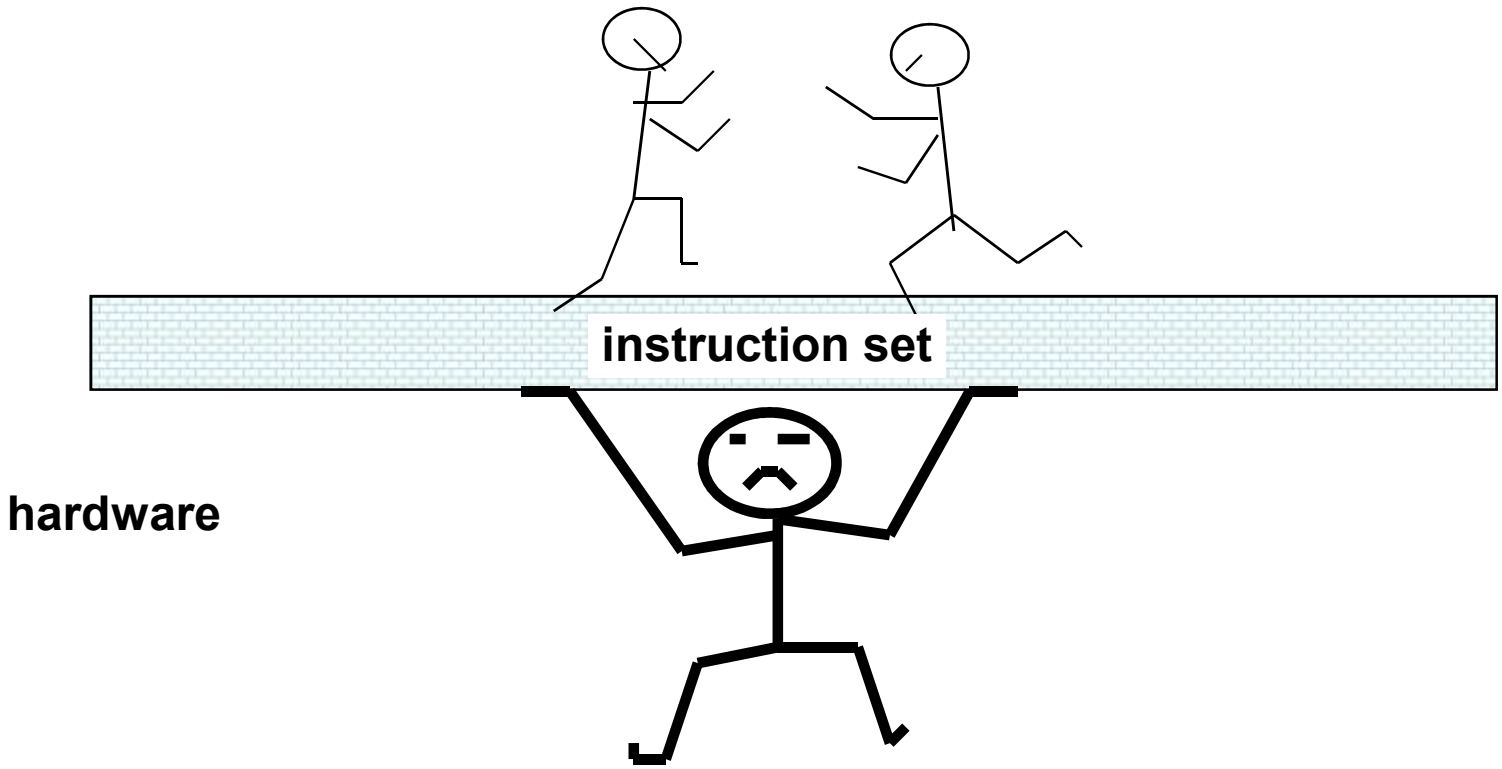
– I/O Mechanisms

-- The number of bits used

-- Techniques for Memory management

For example, It is an architectural design issue whether the computer will have multiply instruction

The Instruction Set: a Critical Interface



Instruction Set Architectures

- Digital Alpha(v1, v3) 1992-97
- HP PA-RISC (v1.1, v2.0) 1986-96
- Sun Sparc (v8, v9) 1987-95
- SGI MIPS (MIPS I, II, III, IV, V) 1986-96
- Intel(8086,80286,80386, 80486,Pentium, MMX, Pentium pro, Itanium...) 1978-200x

Organization

- Capabilities & Performance
  - Characteristics of Principal Functional Units
  - (e.g., Registers, ALU, Shifters, Logic Units, ...)

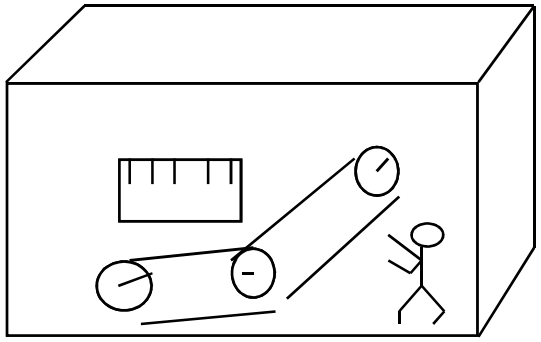
Organization

Logic Designer's View

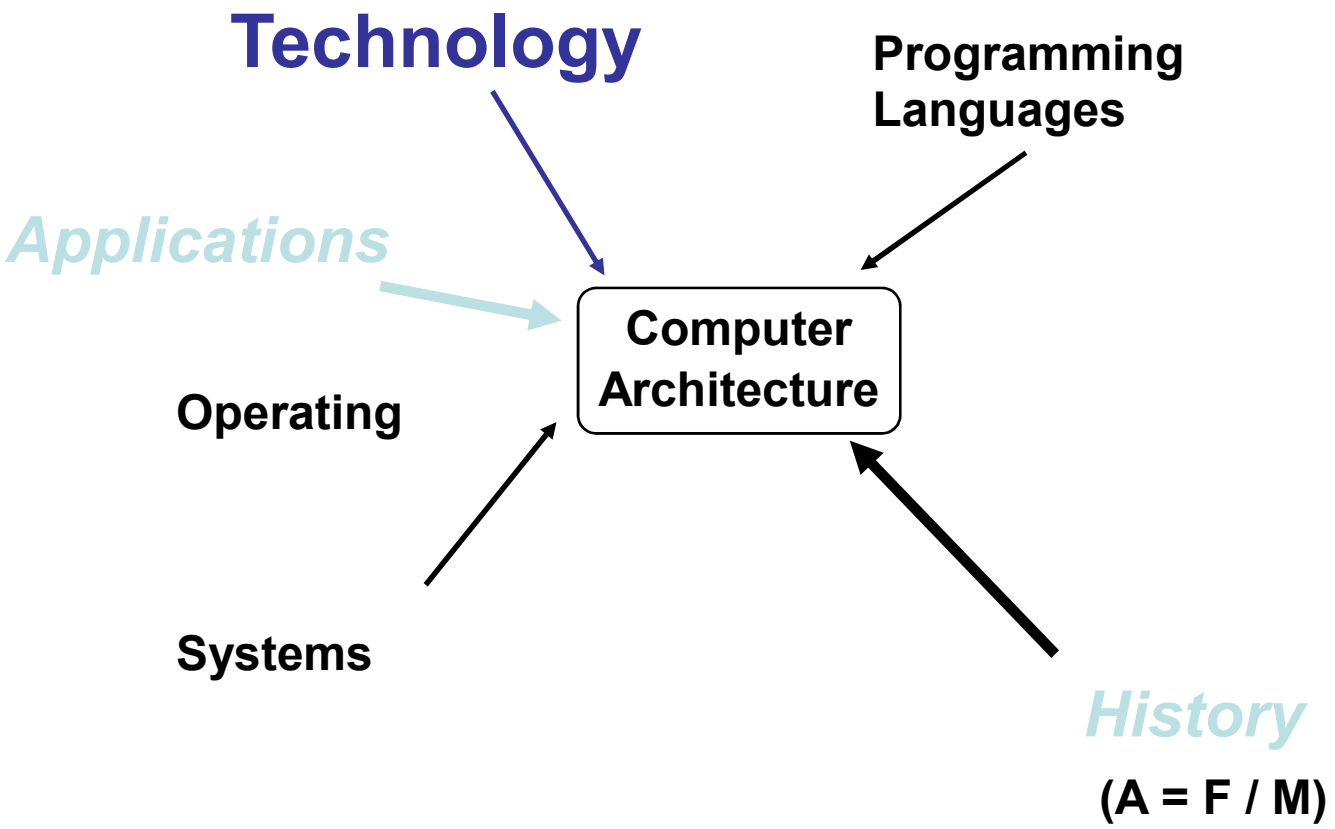
ISA Level

FUs & Interconnect

- Ways in which these components are interconnected
- Information flows between components
- Logic and means by which such information flow is controlled.
- Choreography of FUs to realize the ISA
- Register Transfer Level (RTL) Description



Forces on Computer Architecture



Why do Computer Architecture?

- It's exciting!
- It impacts every other aspect of electrical engineering and computer science

What does a CA Course Contain ?

Computer Architecture and Engineering

Instruction Set Design	Computer Organization
Interfaces	Hardware Components
Compiler/System View	Logic Designer's View
"Building Architect"	"Construction Engineer"

How to Approach CA ?

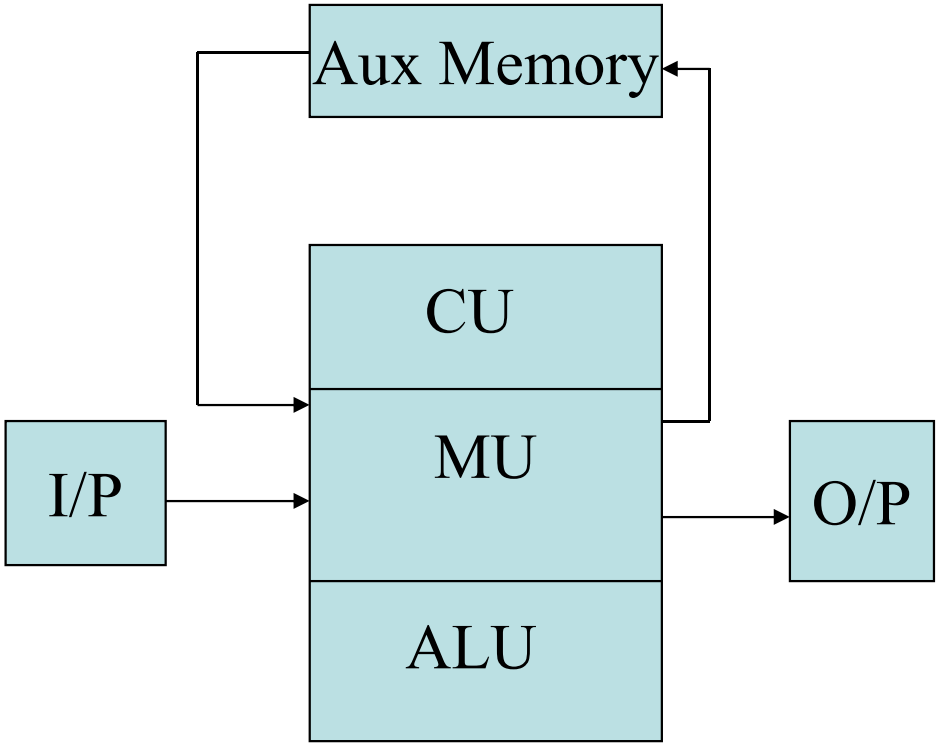
- Components:
  - input (mouse, keyboard)
  - output (display, printer)
  - memory (disk drives, DRAM, SRAM, CD)
  - processor
  - network
- Primary focus: the processor (datapath and control)
  - implemented using millions of transistors
  - Impossible to understand by looking at each transistor

Objective

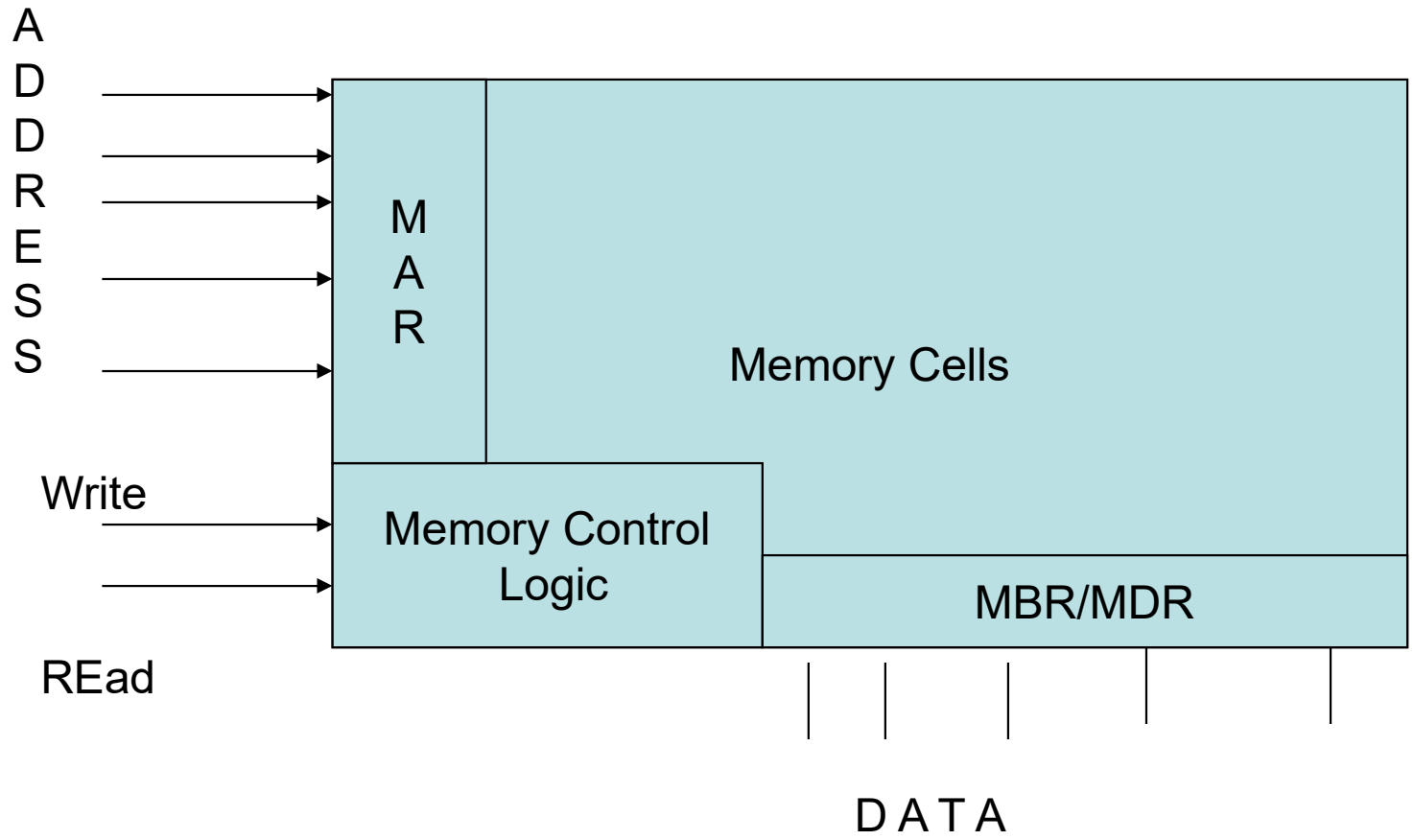
After completing this Session you will be able to understand :

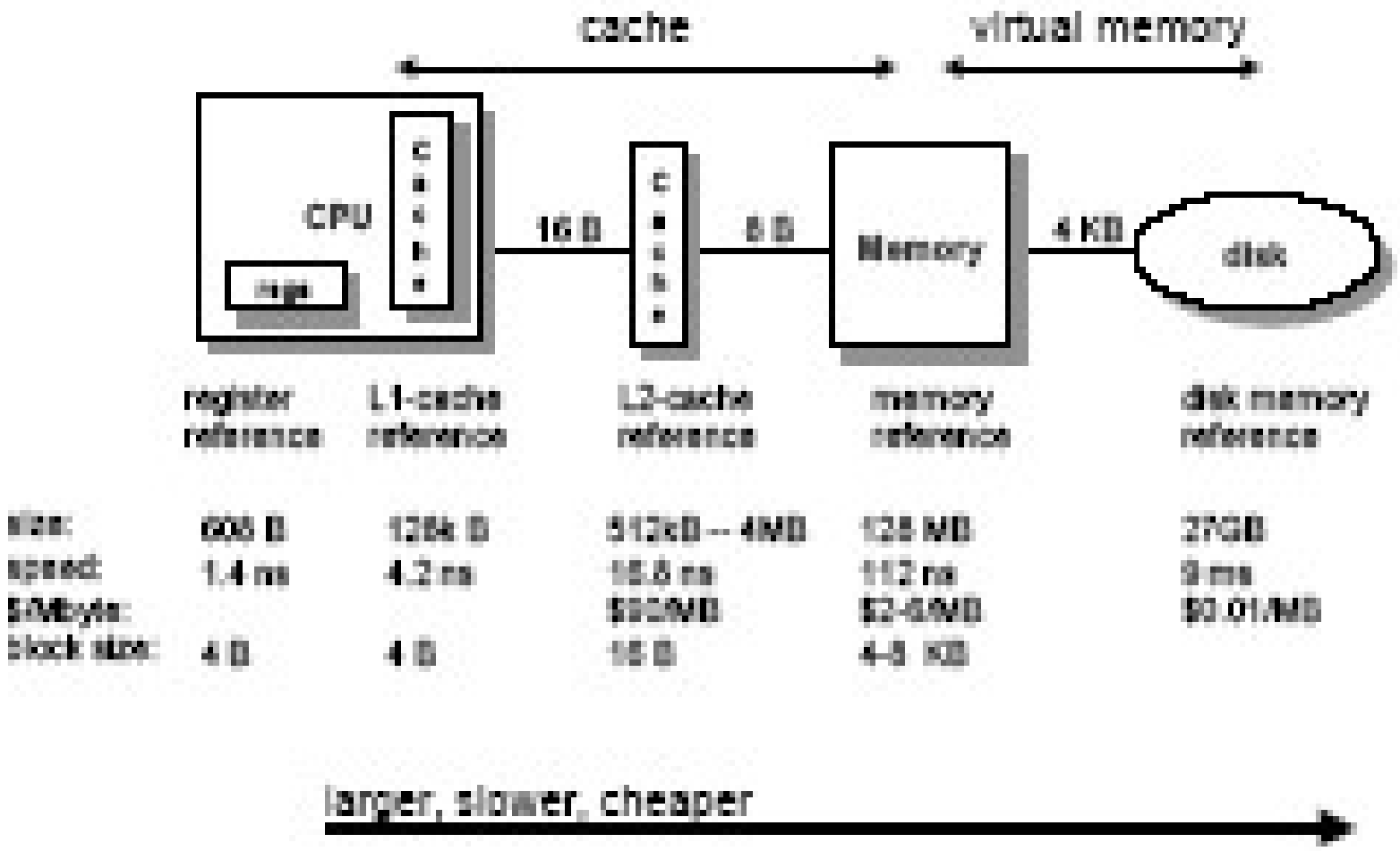
- ALU
- Control unit
- Memory
- I/O

Basic Architecture

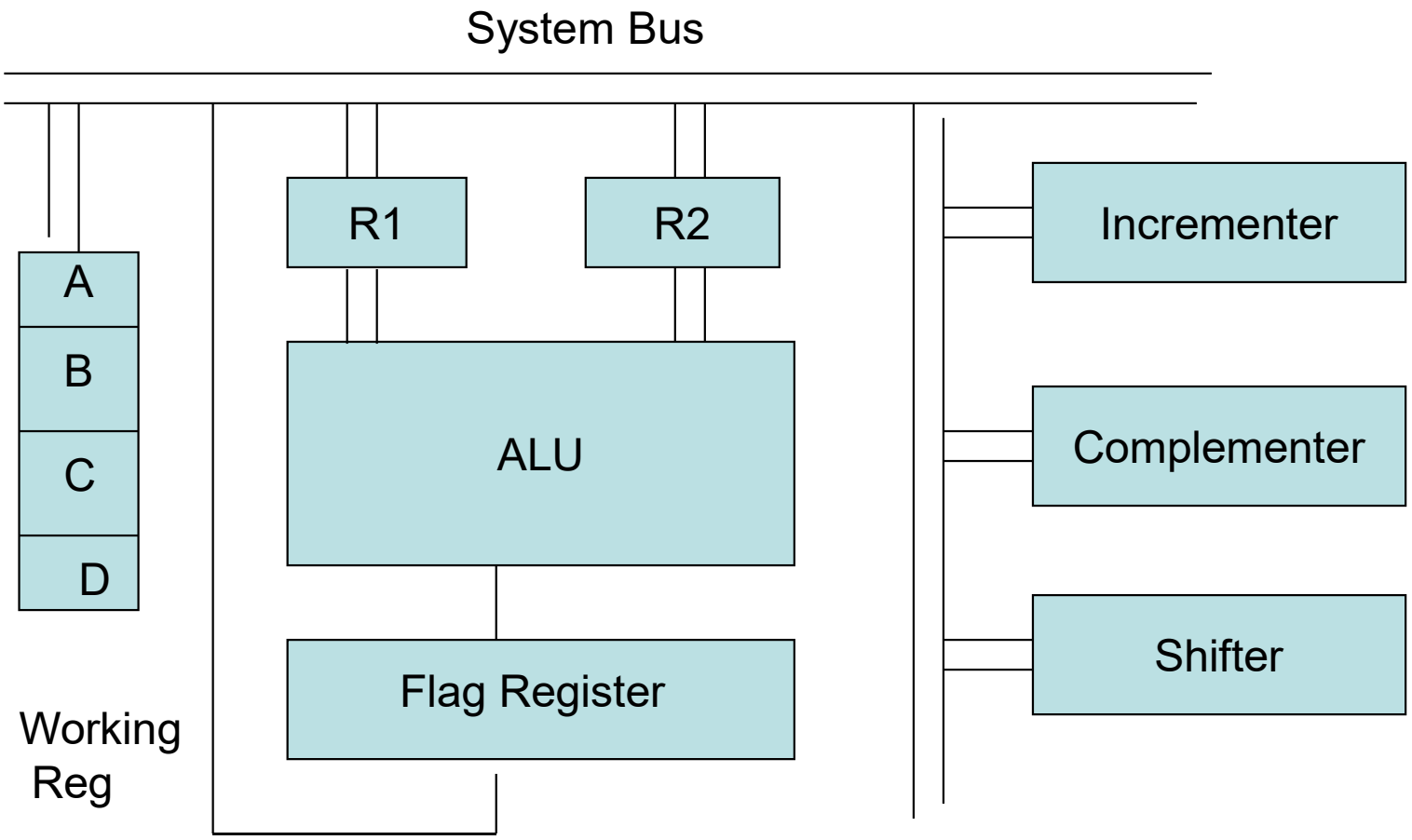


Memory Organization

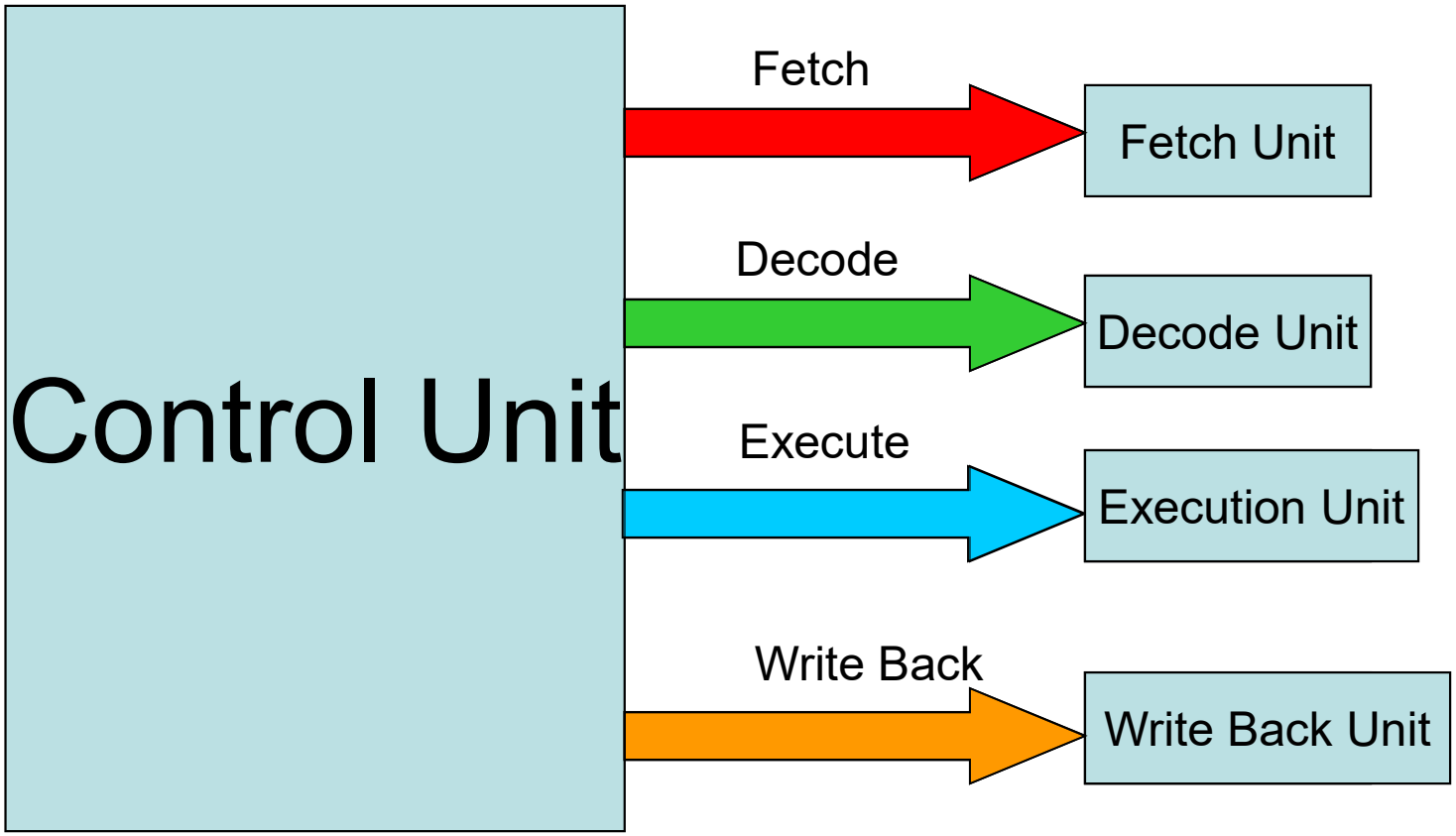




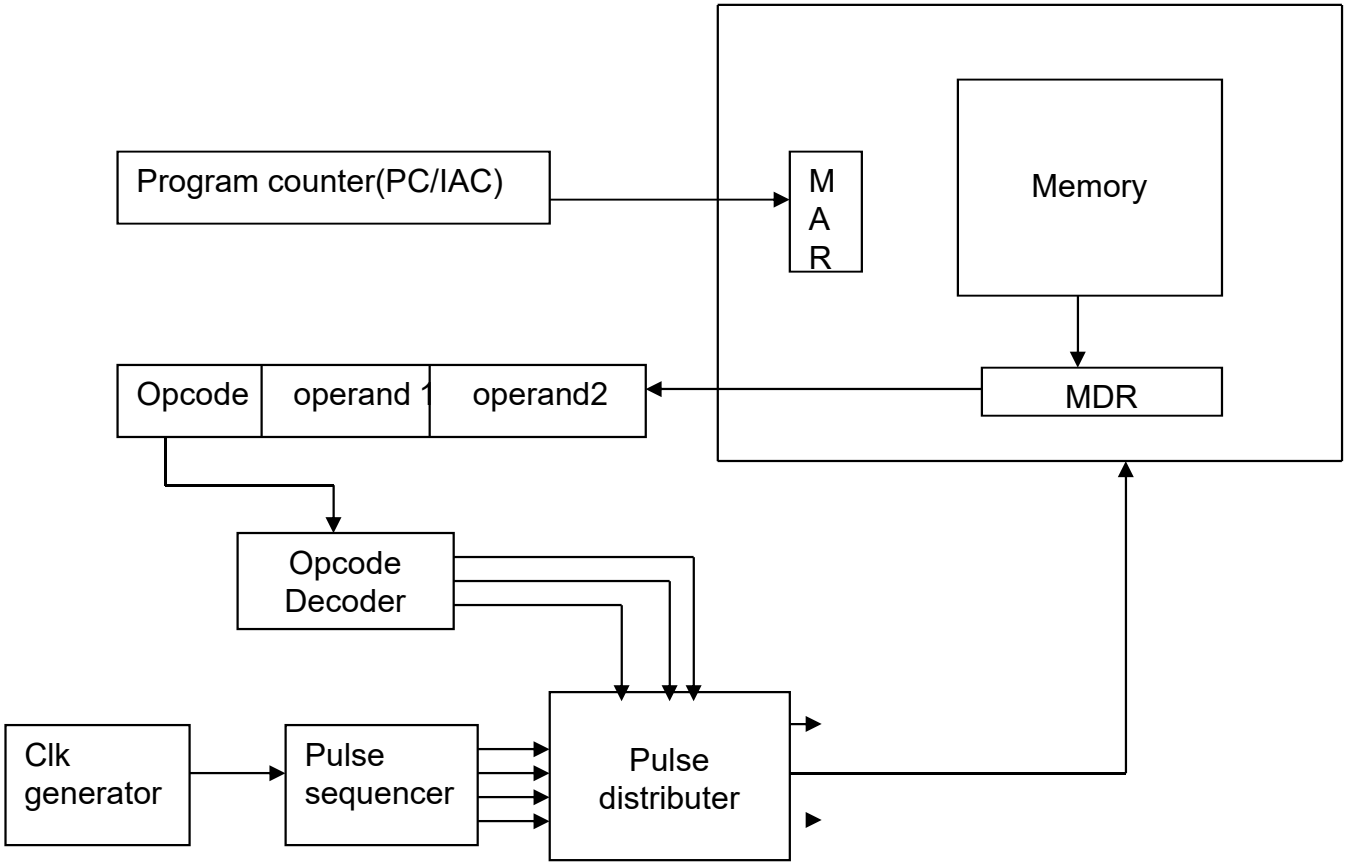
(Numbers are for a 21264 at 700MHz)


System Bus
A
B
C
D
Working Reg
R1
R2
ALU
Flag Register
Incrementer
Complementer
Shifter

A Simplified Control Unit



Internal organization of CU



Generations of Computers

First Generation (1945-54)

- Single CPU
- Accumulator based.
- Fixed point Execution
- All the operations.
- PC
- ALP and MLP
- Ex: ENIAC,Princeton IAS ,IBM 701

II Generation(1955-64)

- Index Register
- Fixed point and Floating point.
- Multiplexed Memory
- Batch Processing
- Subroutine and Libraries.
- IOP
- RTL

II Generation (Contd..)

- HLL such as FORTRAN,COBOL,ALGOL
- EX: IBM7090, UNIVAC LARC

III Generation (1965-74)

- Pipelining
- Cache memory
- Virtual Memory
- Multiprogramming
- Time sharing
- Ex: IBM 360/370, CDC 6600/7600,TI ASC
  DE PDP8

IV Generation (1975-90)

- Parallel Computers
- Shared Memory and Distributed Memory.
- Multiprocessing O.S (MACH)
- Ex:IBM3090,BBN TC 2000, VAX 9000
  CRAY X-MP

V-Generation

- MPP
- Scalable
- Latency Tolerant
- Terra flops
- Heterogeneous Processing.
- Ex: KABRU,FUJITSU,PARAGON

Addressing Modes

Addressing modes

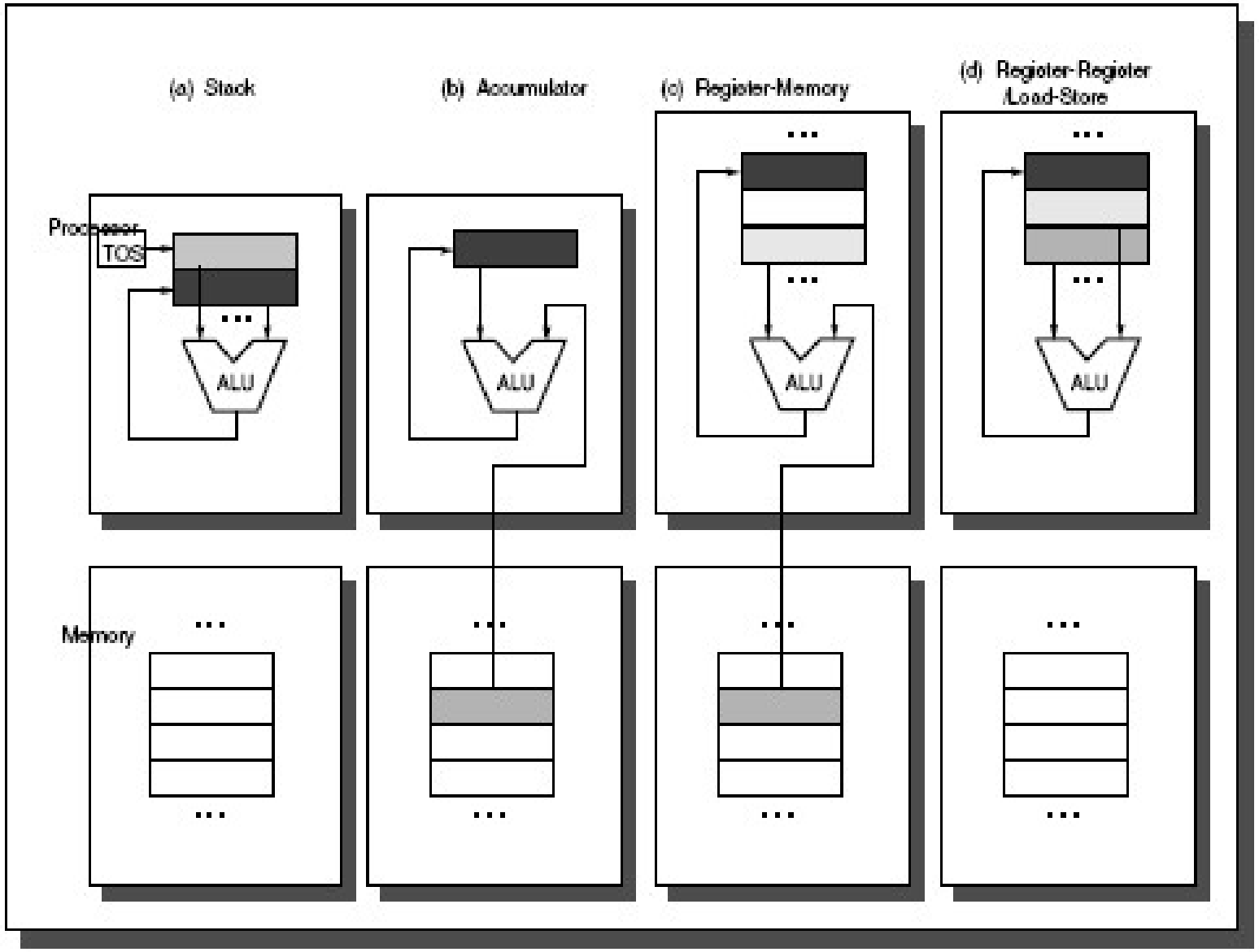
Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a register.
Immediate	Add R4,#3	Regs[R4] ← Regs[R4] +3	For constants.
Displacement	Add R4,100(R1)	Regs[R4] ← Regs[R4] + Mem[100+Regs[R1]]	Accessing local variables (+ simulates register indirect, direct addressing modes)
Register indirect	Add R4,(R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
Indexed	Add R3,(R1 + R2)	Regs[R3] ← Regs[R3] +Mem[Regs[R1] +Regs[R2]]	Sometimes useful in array addressing: R1 = base of array; R2 = index amount.
Direct or absolute	Add R1,(1001)	Regs[R1] ← Regs[R1] + Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect	Add R1,@(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer *p*, then mode yields **p*.

Autoincrement	Add R1, (R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + *d*	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, *d*.
Autodecrement	Add R1, -(R2)	Regs[R2] ← Regs[R2] - *d* Regs[R1] ← Regs[R1] + Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/increment can also act as push/pop to implement a stack.
Scaled	Add R1, 100(R2)[R3]	Regs[R1] ← Regs[R1] + Mem[100+Regs[R2] + Regs[R3]*d]	Used to index arrays. May be applied to any indexed addressing mode in some computers.

Instruction formats at a glance

- Four address format
- Three address format
- Two address format
- Single address format
- Zero address format

Instruction Set Architectures



Examples

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

Instruction Types

- A computer must have instructions capable of performing *three* types of operations.
- **Data transfers instructions:** Transfers information data from one location to another with out changing the binary information contents

Name	Mnemonic	Meaning
Load	LD	Transfers data from memory to processor register
Store	ST	Transfers data from processor register to Memory
Move	MOV	Transfers data from one register to another Register it has also been used to transfer data between cpu reg and memory (or) between Two memory locations.
Exchange	XCH	Used to swap the data between two registers (or) between reg and memory vice versa.
Input	IN	Data transfers between the processor regis
Output	OUT	and input or output terminal.
PUSH	PUSH	Transfers data between processor register and
POP	POP	memory stack

- **Data Manipulation Instructions:** These are the instructions that perform arithmetic, logical and shift operations.

Arithmetic operations:

Name	Mnemonic	Meaning
Increment	INC	Increment the contents of a register
Decrement	DEC	Decrement the contents of a register
ADD	ADD	Add the contents of two registers (or) Contents of a register and memory.
Subtraction	SUB	Subtract the contents of two registers (or) Contents of a register and memory.
Multiply	MUL	Multiply the contents of two registers (or) Contents of a register and memory.
Divide	Div	Divide the contents of two registers (or) Contents of a register and memory.
Add with carry	ADDC	Add the contents of two registers with carry (or)Contents of a reg and memory with carry
Subtract with carry borrow.	SUBB	Sub the contents of two registers with borrow (or)Contents of a reg and memory with borrow
Negate	NEG	Perform 2' complementation on the given number

Logical Operations

Name	Mnemonic	Meaning
Clear	CLR	Clear the contents of a register or Memory location
Complement	COM	Complement the contents of a register.
AND	AND	Perform logical AND operation
OR	OR	Perform logical OR operation
Exclusive OR	XOR	Perform logical XOR operation
Clear carry	CLRC	Clear the carry flag
Set carry	SETC	Set the carry flag.
Complement Carry	COMC	Complement the carry flag
Enable Interrupt	EI	Set the interrupt flag
Disable Interrupt	DI	Disable interrupt flag

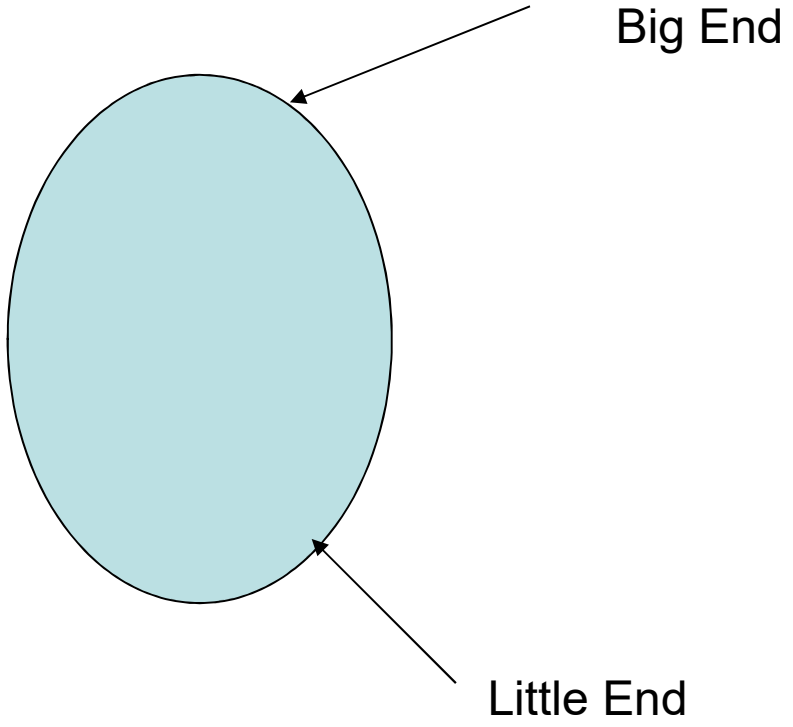
Shift operations

Name	Mnemonics
Logical Shift Right	SHR
Logical Shift Left	SHL
Arithmetic Shift Left	SHLA
Arithmetic Shift Right	SHRA
Rotate Right	ROR
Rotate Left	ROL
Rotate Right Thru Carry	RORC
Rotate Left Thru Carry	ROLC

Program Control Instructions

Name	Mnemonics
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (By Subtraction)	CMP
Test (By AND ing)	TST

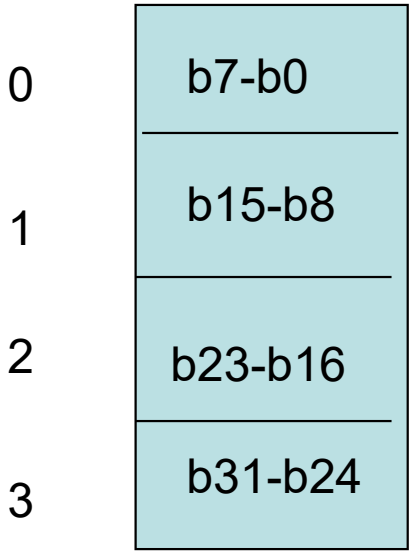
Memory Operand Addressing



Memory Operand Addressing…

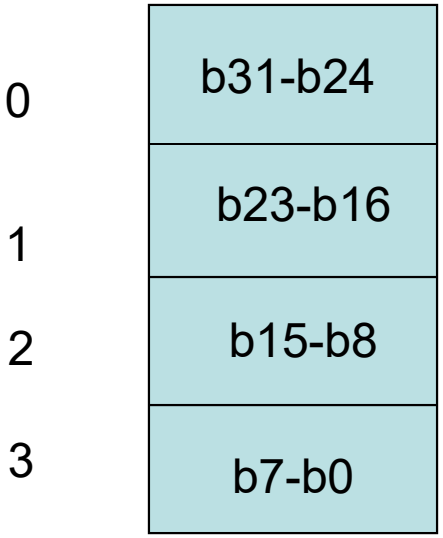
	B31-b24	b23-b16	b15-b8	b7-b0
Big Endian	0	1	2	3
Little Endian	3	2	1	0

- Little Endian



Ex: Intel series

Big Endian



Ex: Motorola series

Buses

- All the Functional units must be connected
- Different type of connection for different type of unit
  - Memory
  - Input/Output
  - CPU

Memory Connection

- Receives and sends data
- Receives addresses (of locations)
- Receives control signals
  - Read
  - Write
  - Timing

Input/Output Connection(1)

- Similar to memory from computer's viewpoint
- Output
  - Receive data from computer
  - Send data to peripheral
- Input
  - Receive data from peripheral
  - Send data to computer

Input/Output Connection(2)

- Receive control signals from computer
- Send control signals to peripherals
  - e.g. spin disk
- Receive addresses from computer
  - e.g. port number to identify peripheral
- Send interrupt signals (control)

CPU Connection

- Reads instruction and data
- Writes out data (after processing)
- Sends control signals to other units
- Receives (& acts on) interrupts

Performance

- Measure, Report, and Summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation

Computer Performance: TIME, TIME, TIME

- **Response Time (latency)**
  - — How long does it take for my job to run?
  - — How long does it take to execute a job?
  - — How long must I wait for the database query?
- **Throughput**
  - — How many jobs can the machine run at once?
  - — What is the average execution rate?
  - — How much work is getting done?

Execution Time

- **Elapsed Time**
  - counts everything *(disk and memory accesses, I/O , etc.)*
  - a useful number, but often not good for comparison purposes
- **CPU time**
  - doesn't count I/O or time spent running other programs
  - can be broken up into **system time**, and **user time**
- Our focus: **user CPU time**
  - time spent executing the lines of code that are "in" our program

Book's Definition of Performance

- For some program running on machine X,
  $\text{Performance}_X = 1 / \text{Execution time}_X$

- "X is n times faster than Y"

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n$$

Problem:

  - machine A runs a program in 20 seconds
  - machine B runs the same program in 25 seconds

Performance …

- Execution time = no. of clock cycles * cycle time
- Execution time = $\frac{\text{no. of clock cycles}}{\text{Clock rate}}$

Performance …

But

no of clock cycles = Instruction count * Average Cycles per Instruction

Execution time = I.C * CPI * Cycle time

$$= \frac{I.C * CPI}{\text{clock rate}}$$

Performance …

- MOV AX,90 → 2 Cycles
- MOV BX,50 → 2 Cycles
- ADD AX,BX → 1 Cycle
- STA AX,4500 → 3 Cycles
- HLT → 1 Cycle

- CPI = 9 /5 =1.8

Performance …

- **Instead of reporting execution time in seconds, we often use cycles**

- Execution time = I.C * CPI * Cycle time

  $$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{instruction}}{\text{Program}} * \frac{\text{cycles}}{\text{instruction}} * \frac{\text{seconds}}{\text{cycles}}$$

- I.C => Instrn count (Dynamic Vs Static)
  - Dynamic I.C useful in performance (timing)
  - Static I.C useful in code size.

Performance …

- Execution Time = I.C * CPI * Cycle Time
  - I.C → depends on ISA and Compiler tech
  - CPI → depends on ISA and organization
  - C.T → depends on organization and technology

Performance …

- CPI – Avg Cycles Per Instruction(CPI) computed from the instruction mix.

	ALU	L/S	Cond Jmp	Uncond Jmp	CPI
Freq	40	25	20	15	
cycles	4	5	3	2	

Performance …

- CPI =40% * 4+ 25% *5 + 20%*3 + 15%*2

  = 0.4*4+0.25 * 5+ 0.5*3 + 0.15*2

  =3.75

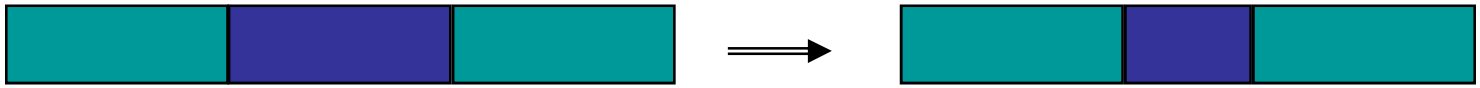
Amdahl's Law

$$\text{Execution Time After Improvement} = \text{Execution Time Unaffected} + \frac{\text{Execution Time Affected}}{\text{Amount of Improvement}}$$

Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



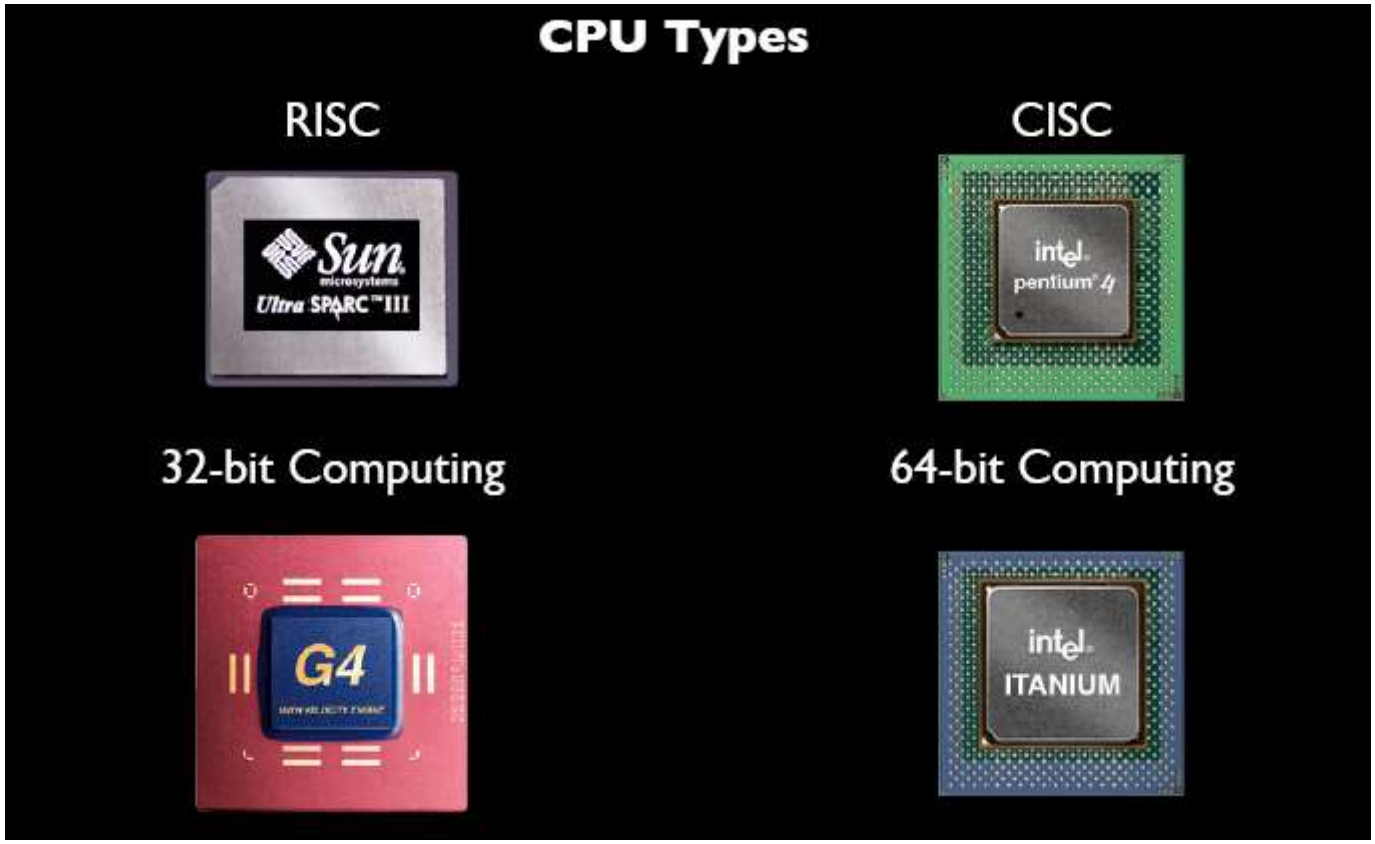
Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected

Amdahl's Law

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times \left[(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right]$$

$$\text{Speedup}_{overall} = \frac{\text{ExTime}_{old}}{\text{ExTime}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

RISC Vs CISC


CPU Types
RISC
CISC
Sun microsystems
Ultra SPARC™III
intel
pentium 4
32-bit Computing
64-bit Computing
G4
intel
ITANIUM

RISC	CISC
Reduced Instruction Set Computer	Complex Instruction set computer
Less number of instructions (64-100)	More number of instructions. (100-250)
Fixed length instruction formats (32 bits)	Variable length instruction formats (32 bits)
Limited addressing modes (3-5)	More addressing modes (12-24)

RISC	CISC
More no. of general purpose registers (32-192)	Limited no. of general purpose registers (8-20)
It employs H/w control organization.	It employs micro-program control organization. Recent architectures Employ h/w tech
It does not uses control memory	It uses control memory
Memory manipulation instructions are available	No memory manipulation instructions are available. Memory access is limited to simple LOAD/STORE instructions

RISC	CISC
Split caches were used	Unified cache was used. Now a days split caches were used
Clock speed (50-150Mhz in 1992)	Clock speed (33-50Mhz in 1992)
SPARC,DLX,MIPS architectures	Intel 80x86,Motorola MC68000 series

Instruction Types of MIPS

I - type instruction

Encodes: Loads and stores of bytes, words, half-words
All immediates (rd <- rs1 op immediate)

Conditional branch instructions (rs1 is register, rd unused)
Jump register, Jump and link register
(rd = 0, rs = destination, immediate = 0)

Instruction Types of MIPS (Cont..)

R - type instruction

Register - register ALU operations: rd <- rs1 func rs2

Function encodes the data path operation: Add, Sub,...

Read/Write special registers and moves

Instruction Types of MIPS (Cont..)

J - type instruction

Jump and Jump and link

Trap and RFE

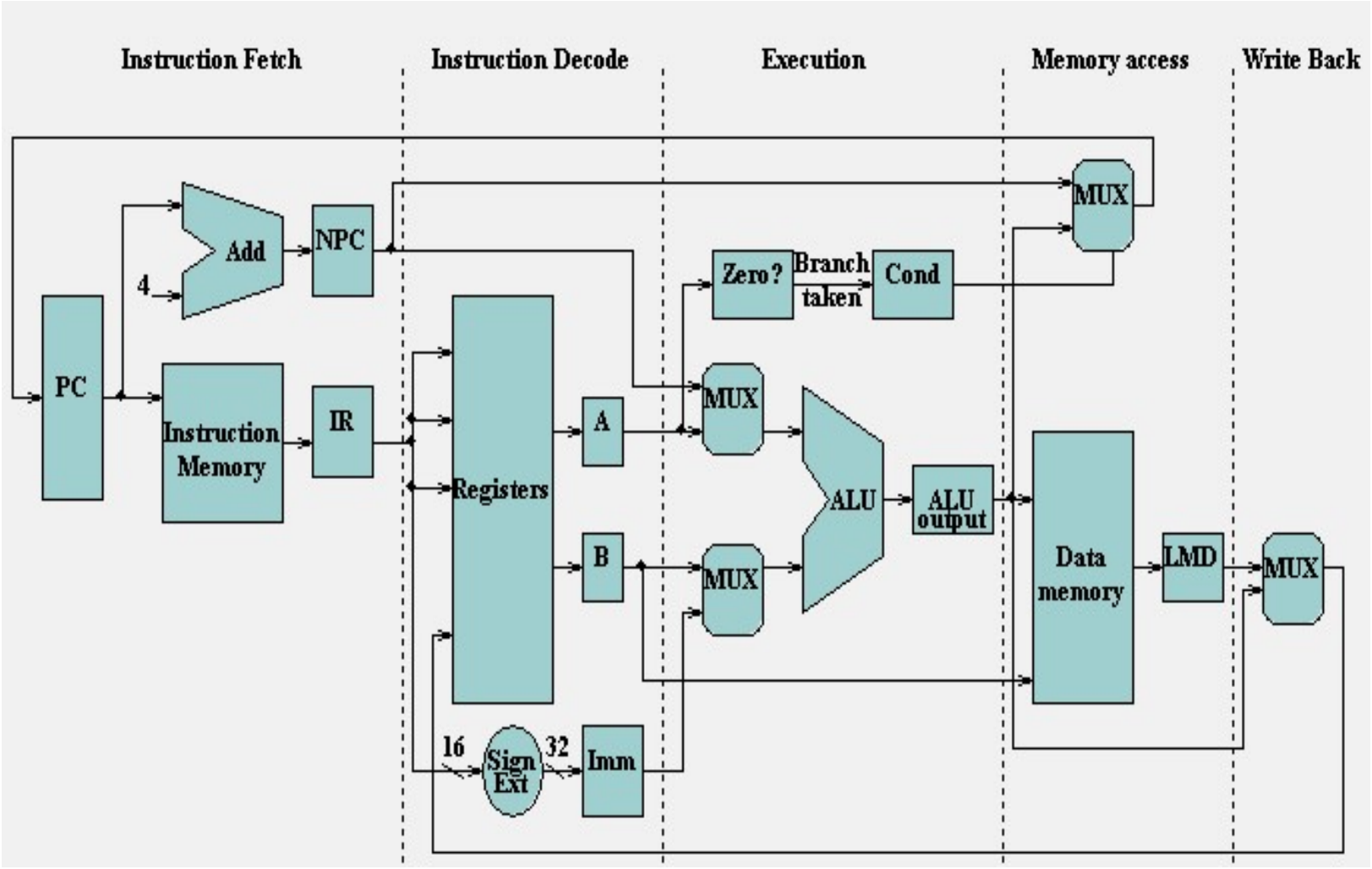
5-segment MIPS pipeline

- Instruction fetch cycle (IF)
- Instruction decode/register fetch (ID)
- Execution/Effective address cycle (EX)
- Memory access/branch completion (MEM)
- Write-back cycle (WB)

Pipeline char diag

Instr Num	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
instr i	IF	ID	EX	MEM	WB				
instr i+1		IF	ID	EX	MEM	WB			
instr i+2			IF	ID	EX	MEM	WB		
instr i+3				IF	ID	EX	MEM	WB	
instr i+4					IF	ID	EX	MEM	WB

MIPS Pipeline processor



Instruction Fetch Cycle

IR <- MEM[PC]

NPC <- PC +4

Instruction decode/register fetch (ID)

A <- Regs[IR6..10]
B <- Regs[IR11..15]
Imm <- ((IR16)16##IR16..31)

Execution/Effective address cycle (EX)

- **Memory reference:**
  ALUOutput <- A +Imm
- **Register-Register ALU instruction:**
  ALUOutput <- A *op* B
- **Register- Immediate ALU instruction:**
  ALUOutput <- A *op* Imm
- **Branch:**
  ALUOutput <- NPC + Imm
  Cond <- (A op 0)

Memory access/branch completion cycle (MEM)

Memory reference:

LMD <- Mem[ALUOutput] or Mem[ALUOutput] <- B

Branch:

if (cond) PC <- ALUOutput
else PC <- NPC

Write-back cycle (WB)

- Register-Register ALU instruction:
  Regs[IR16..20] <- ALUOutput Register
- Immediate ALU instruction:
  Regs[IR11..15] <- ALUOutput
- Load instruction:
  Regs[IR11..15] <- LMD

Questions

- A memory unit with a capacity of 65,536 words of 25 bits each. It is used in conjunction with a general-purpose computer. The instruction code is divided into four parts an indirect mode bit, opcode, two bits that specify a processor register and an address part.
- what is the maximum number of operation that can be in the computer if the instruction is stored in one memory word.
- Draw the instruction word format indicating the number of bits and the function of each part.
- How many processor registers are there in the computer and how many bits in each. How many bits are there in MBR,MAR and PC

Questions

- what is the difference between a direct and indirect addressed instruction? How many reference are needed for each type of instruction if the instruction is in memory and to bring an operand in to a processor register.

Thank you