Computer Architecture and Organization

MCSE503L

MAHESWARI.R

MODULE I

COMPUTER EVOLUTION AND PERFORMANCE

TEXT / REFERENCE BOOKS

Text Book(s)	
1.	William Stallings, Computer Organization and Architecture: Designing for Performance, Pearson, 2022, 11th Edition, Pearson
2	Gerassimos Barlas, Multicore and GPU Programming: An Integrated Approach, 2022, 2nd edition, Morgan Kaufmann
Reference Books	
1.	J.L. Hennessy and D.A. Patterson. Computer Architecture: A Quantitative Approach. 5th Edition, 2012, Morgan Kauffmann Publishers.
2.	Shameem Akhter, Jason Roberts, Multi-core Programming: Increasing Performance Through Software Multi-threading, 2010, Intel Press, BPB Publications

Module I

- **Computer Evolution And Performance**

Defining Computer Architecture and Organization, Overview of Computer Components, Von Neumann architecture, Harvard Architecture CISC & RISC, Flynn's Classification of Computers, Moore's Law, Multi-threading, Comparisons of Single Core, Multi Processors, and Multi-Core architectures, Metrics for Performance Measurement

INTRODUCTION

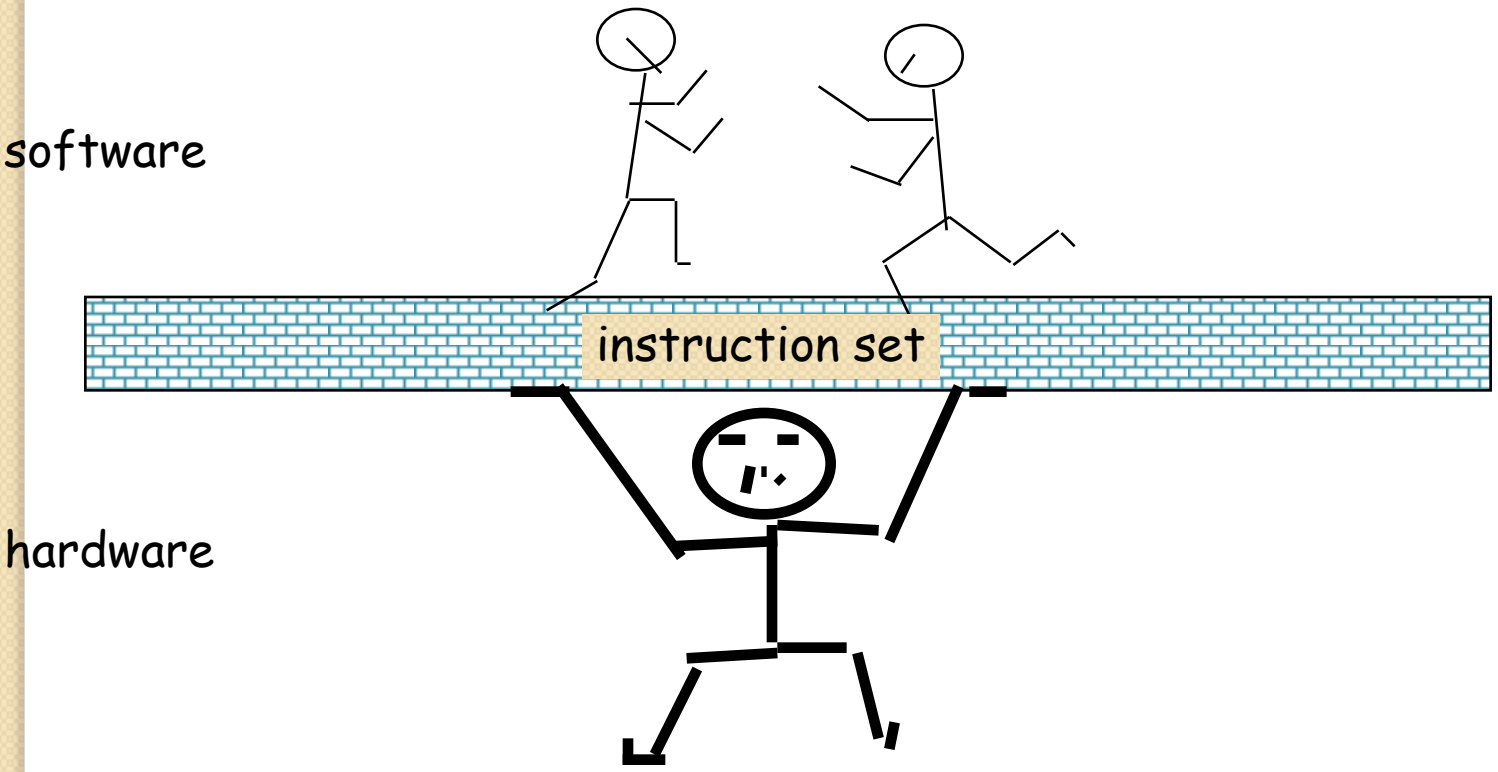
What is *Computer Architecture*

Computer Architecture =

Instruction Set Architecture +

Organization + Hardware + …

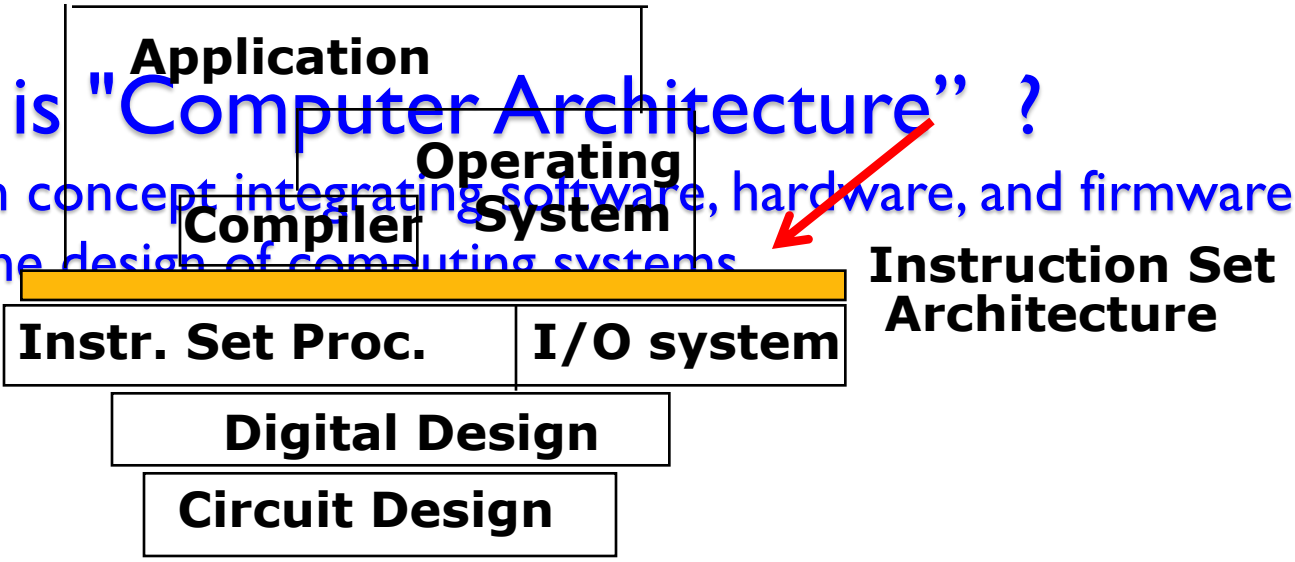
The Instruction Set: a Critical Interface



What is "Computer Architecture" ?

A system concept integrating software, hardware, and firmware to specify the design of computing systems

- Co-ordination of ***levels of abstraction***



Computer Architecture

Computer architecture refers to those attributes of a system visible to a programmer or,

those attributes that have a **direct impact on the logical execution of a program.**

Computer architecture Attributes

- the instruction set,
- the number of bits used to represent various data types (e.g., numbers, characters),
- I/O mechanisms, and techniques
- for addressing memory.

Computer Organization

Computer organization refers to the operational units and their interconnections that realize the architectural specifications.

Computer Organization

Attributes include those hardware details transparent to the programmer:

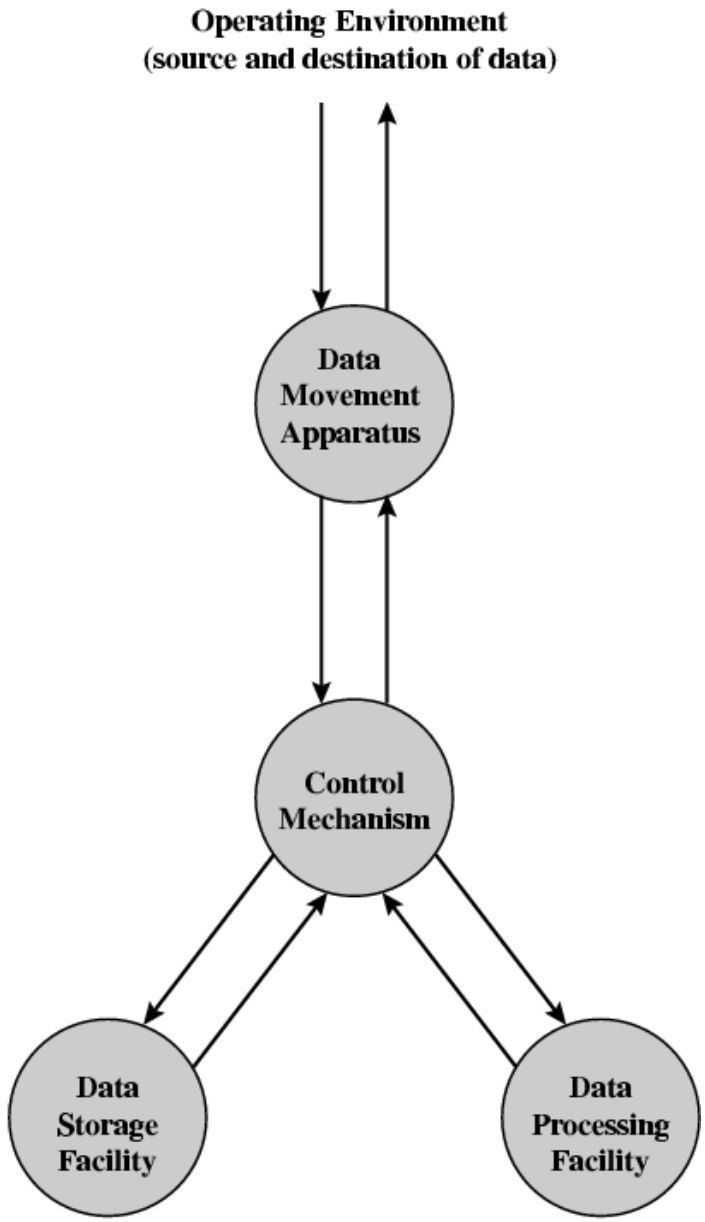
- such as control signals;
- interfaces between the computer
- and peripherals;
- and the memory technology used.

Structure & Function

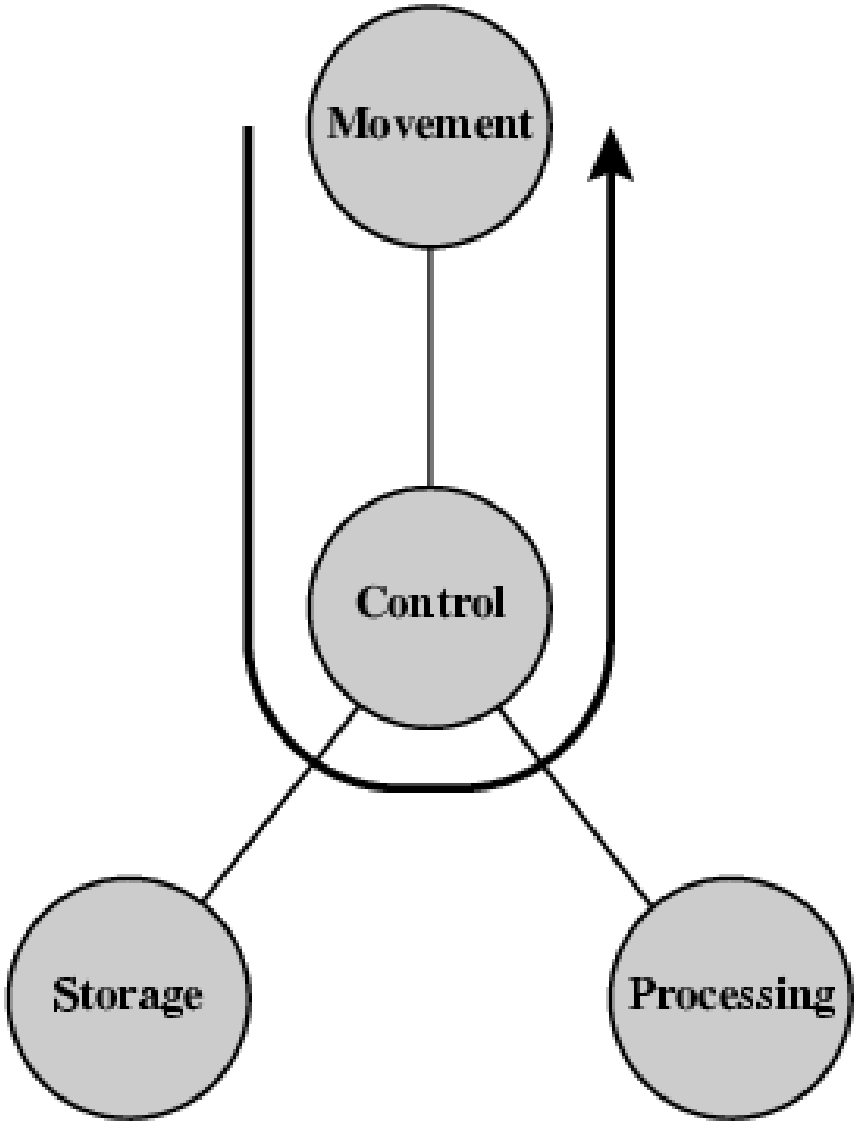
- Structure is the way in which components relate to each other
- Function is the operation of individual components as part of the structure

Function

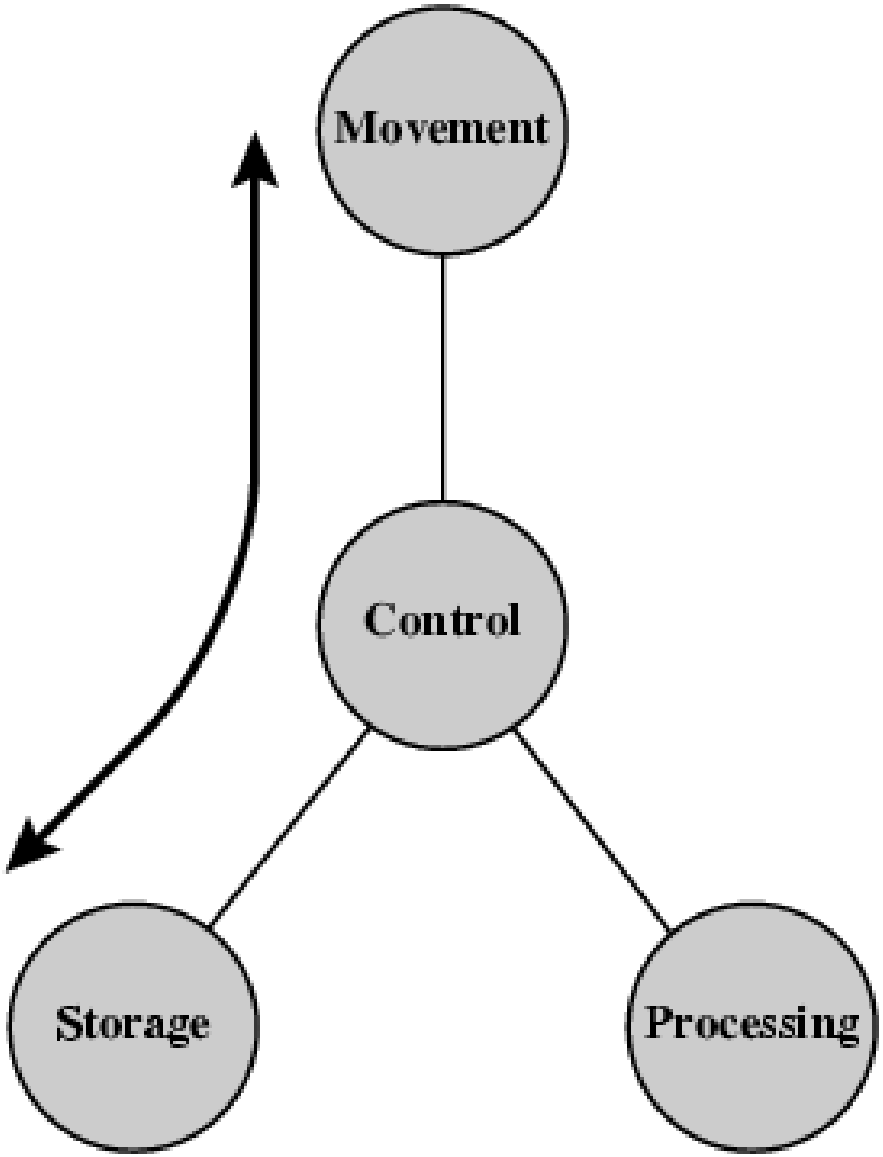
- All computer functions
  - Data processing
  - Data storage
  - Data movement
  - Control



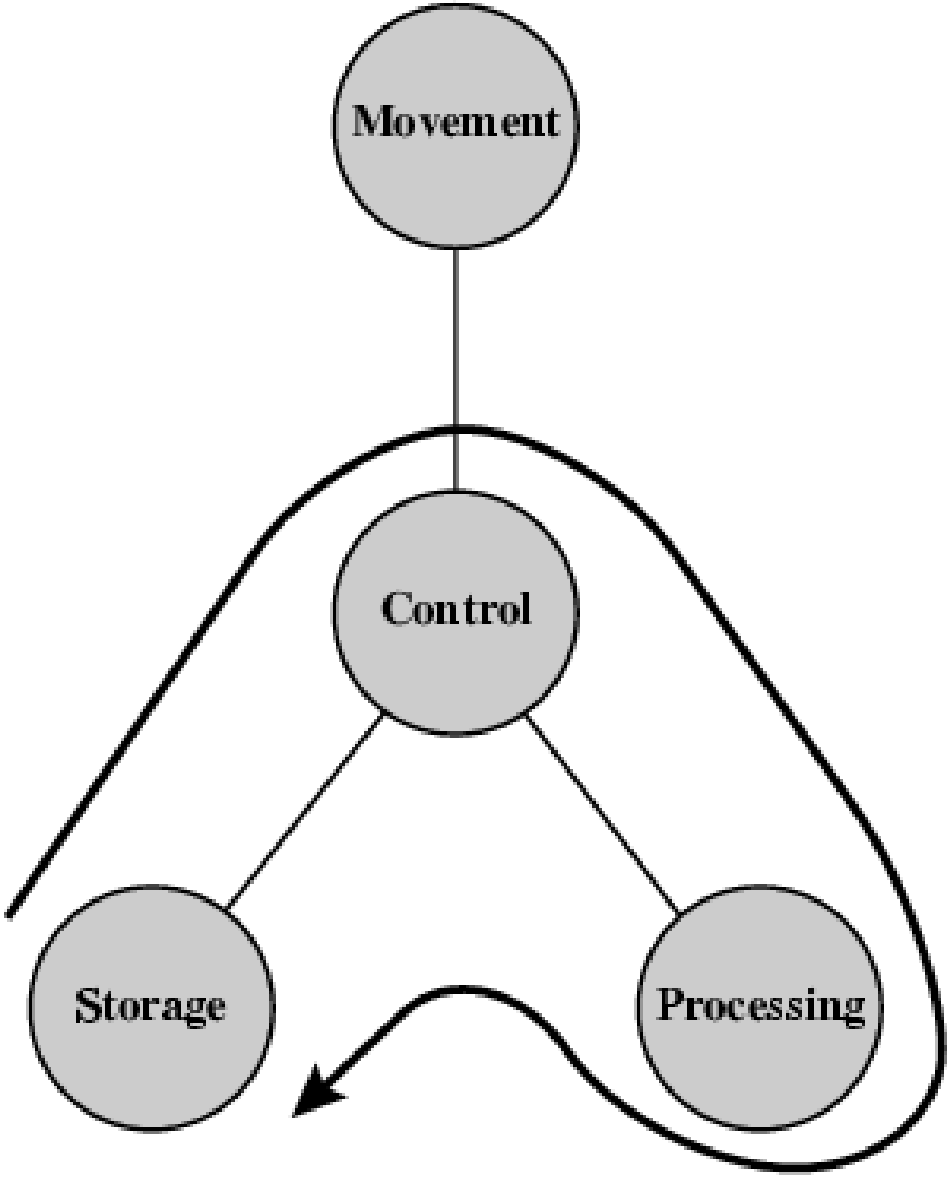
Operations (1) Data movement



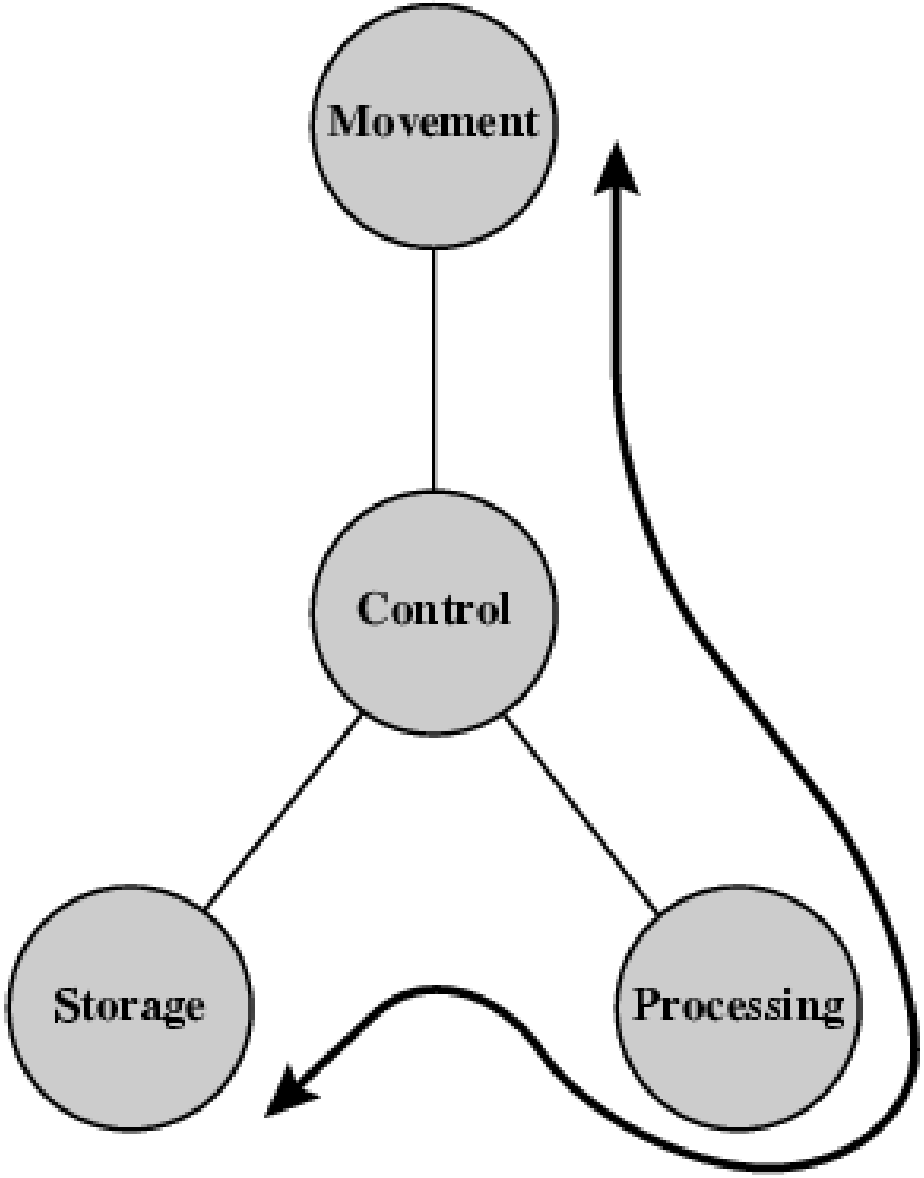
Operations (2) Storage



Operation (3) Processing from/to storage

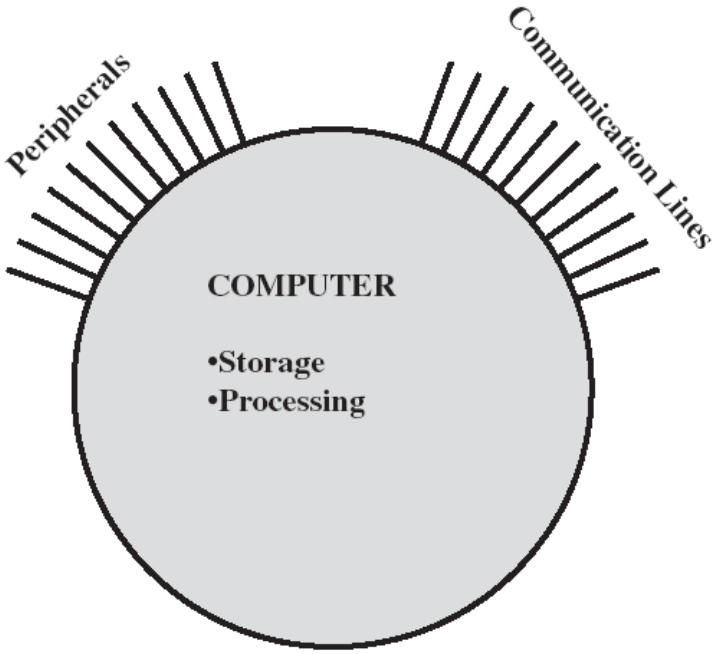


Operation (4)
Processing from storage to I/O

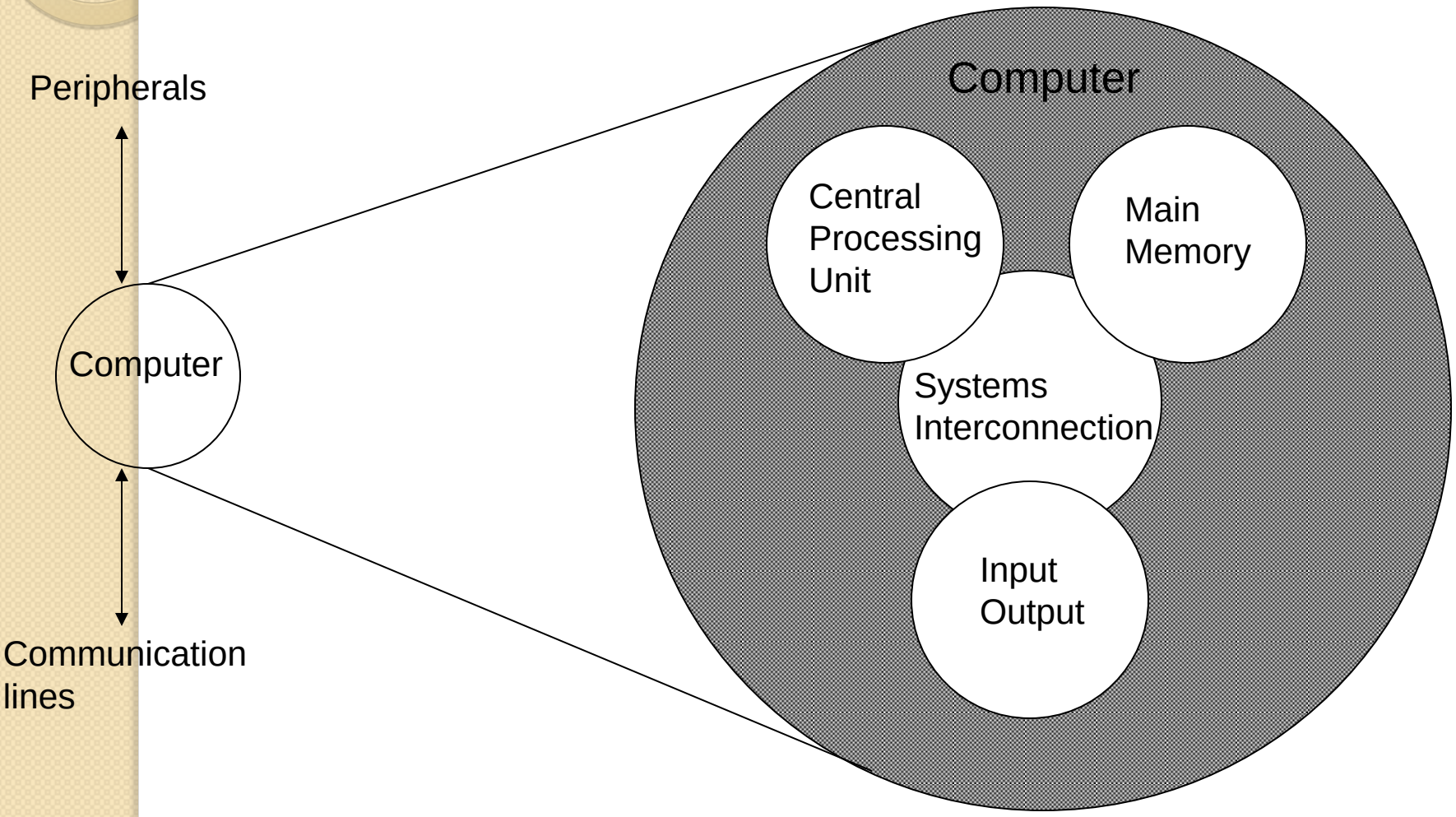


Structure

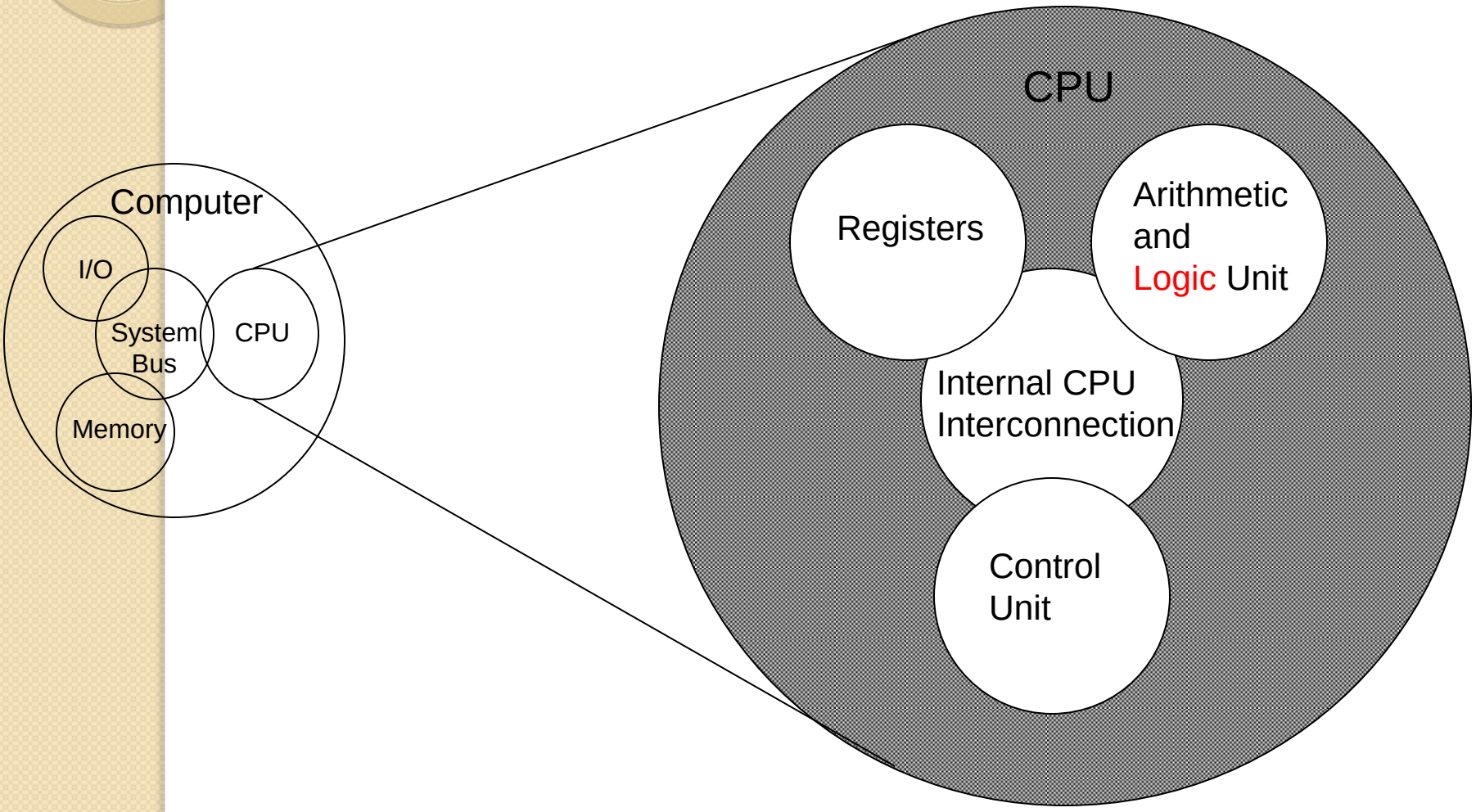
- The Computer
  - CPU
    - Controls the operation of th computer and performs its d processing functions.
  - Main memory
    - Stores data
  - I/O
    - Moves data between the con and its external environment
  - System interconnection
    - Provides for communication among CPU, main memory, and I/O



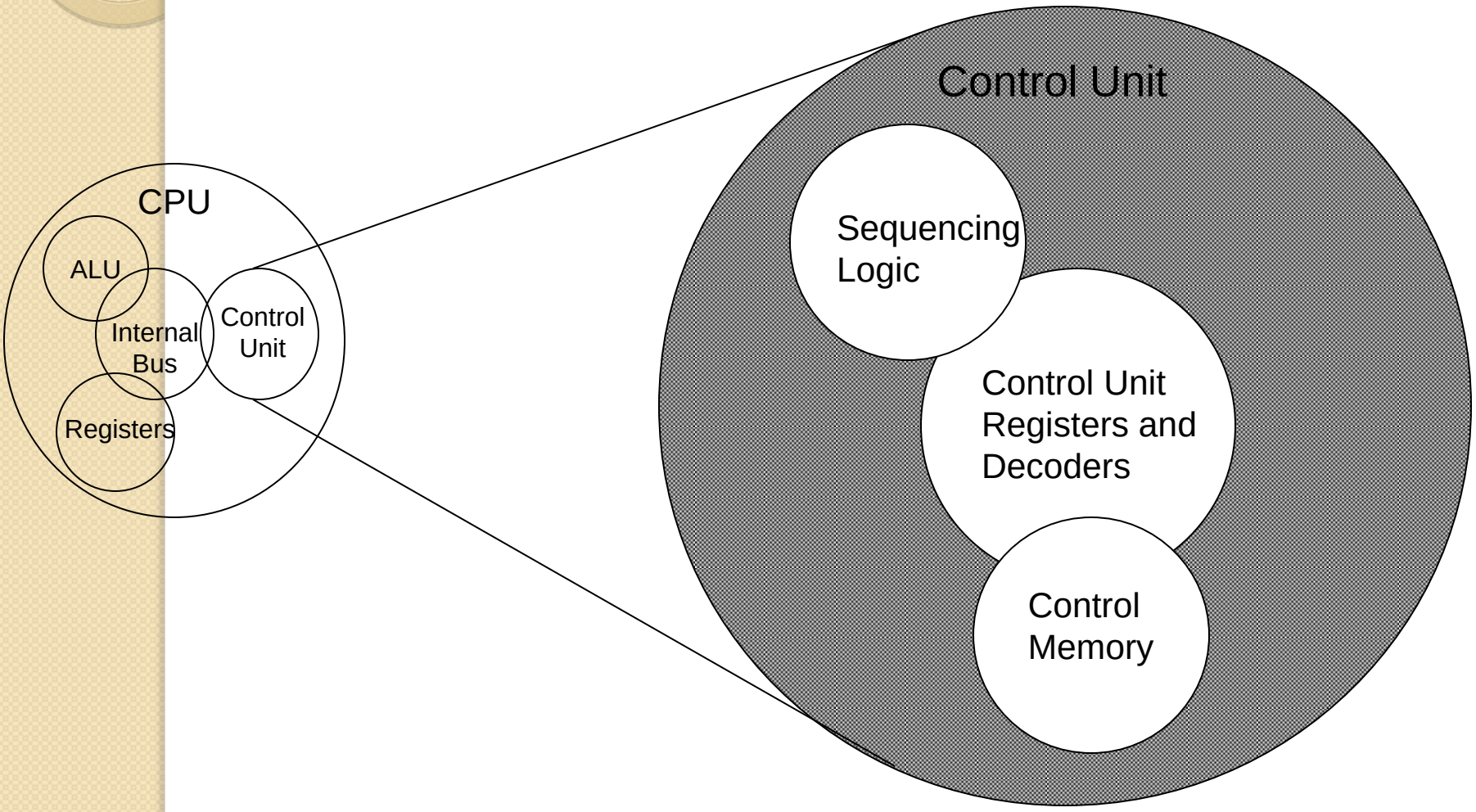
Structure - Top Level



Structure - The CPU



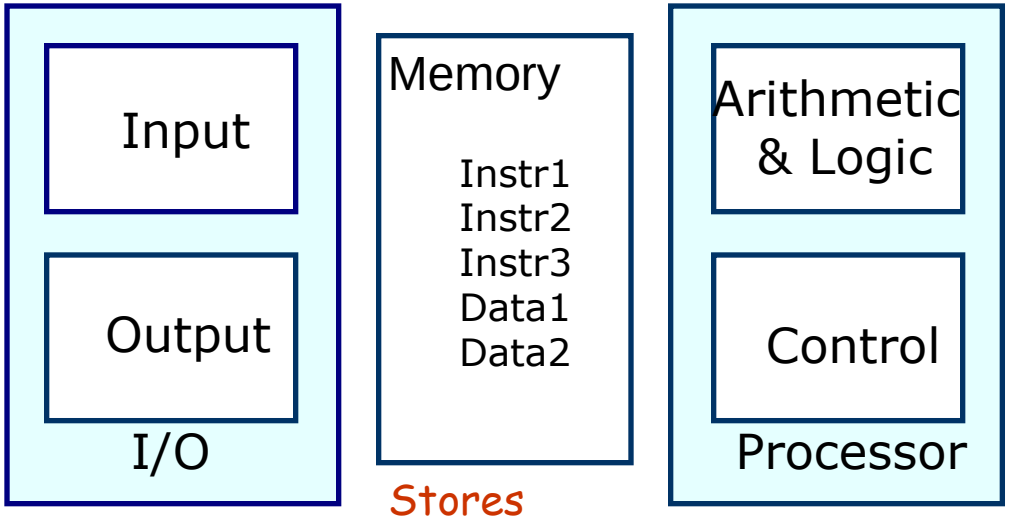
Structure - The Control Unit



Functional units of a computer

Input unit accepts information:
- Human operators,
- Electromechanical devices
- Other computers

Arithmetic and logic unit(ALU):
- Performs the desired operations on the input information as determined by instructions in the memory



Output unit sends results of processing:
- To a monitor display,
- To a printer

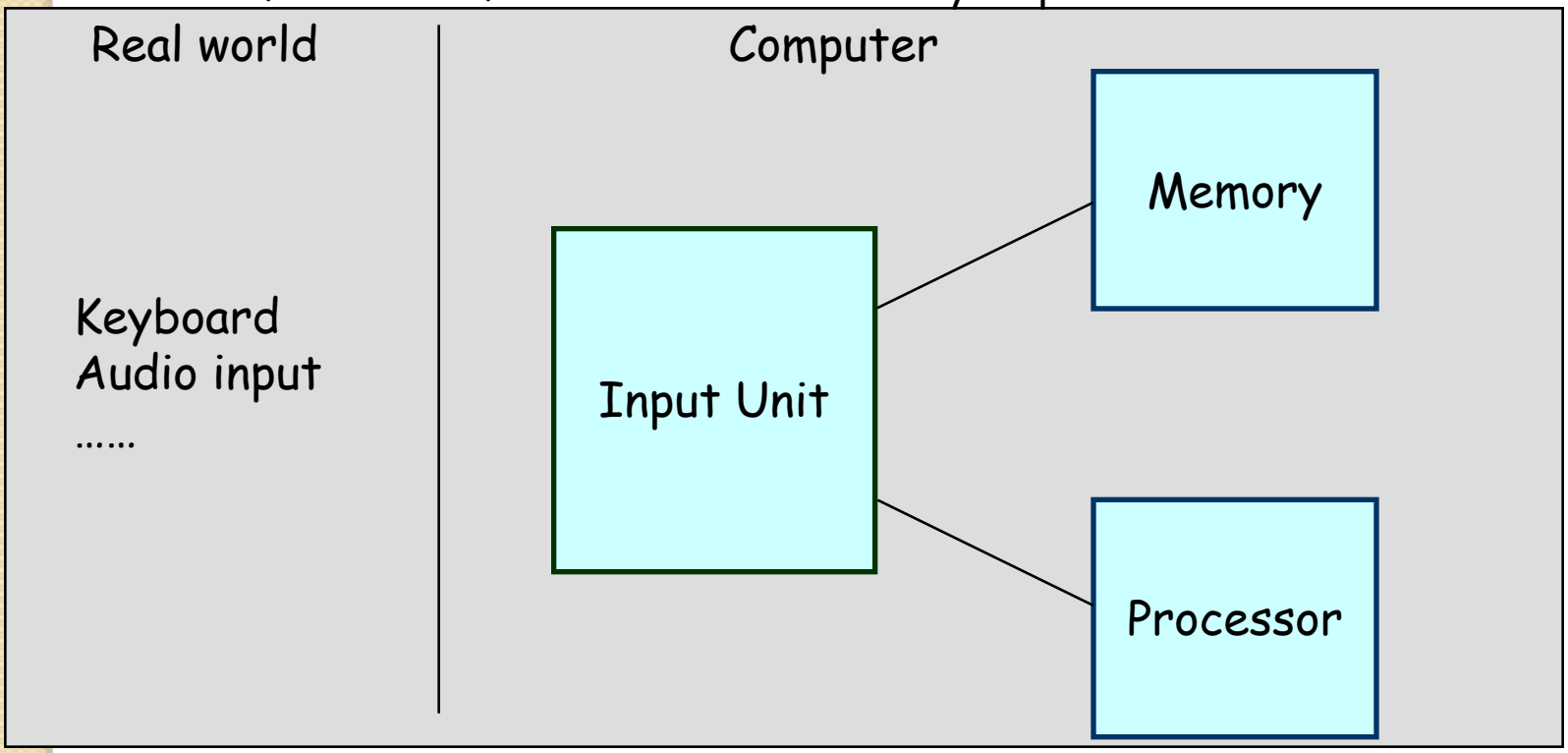
Stores information:
- Instructions,
- Data

Control unit coordinates various actions
- Input,
- Output
- Processing

Input unit

Binary information must be presented to a computer in a specific format. This task is performed by the input unit:

- Interfaces with input devices.
- Accepts binary information from the input devices.
- Presents this binary information in a format expected by the computer.
- Transfers this information to the memory or processor.



Memory unit

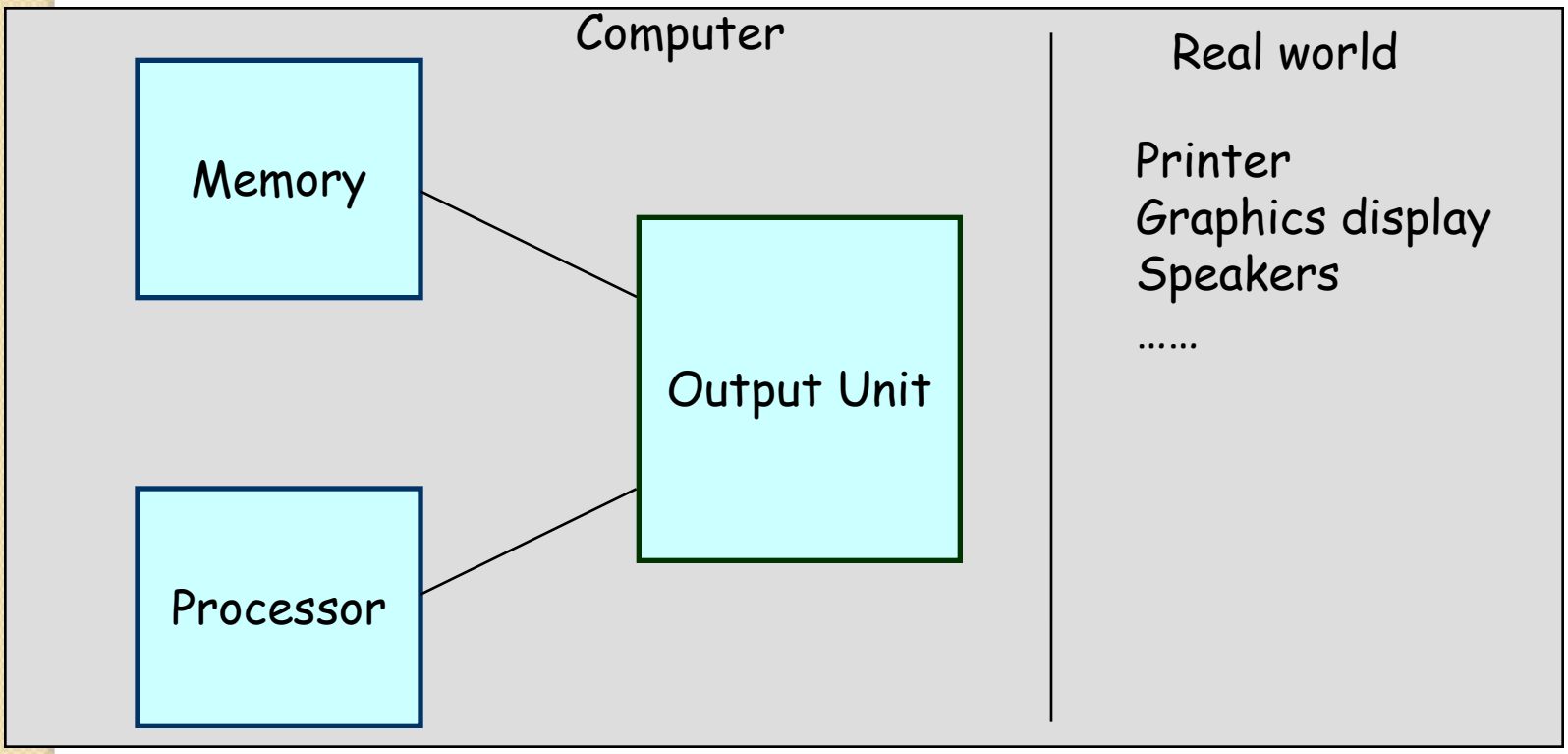
- Memory unit stores instructions and data.
  - Recall, data is represented as a series of bits.
  - To store data, memory unit thus stores bits.
- Processor reads instructions and reads/writes data from/to the memory during the execution of a program.
  - In theory, instructions and data could be fetched one bit at a time.
  - In practice, a group of bits is fetched at a time.
  - Group of bits stored or retrieved at a time is termed as "word"
  - Number of bits in a word is termed as the "word length" of a computer.
- In order to read/write to and from memory, a processor should know where to look:
  - "Address" is associated with each word location.

Arithmetic and logic unit (ALU)

- Operations are executed in the Arithmetic and Logic Unit (ALU).
  - Arithmetic operations such as addition, subtraction.
  - Logic operations such as comparison of numbers.
- In order to execute an instruction, operands need to be brought into the ALU from the memory.
  - Operands are stored in general purpose registers available in the ALU.
  - Access times of general purpose registers are faster than the cache.
- Results of the operations are stored back in the memory or retained in the processor for immediate use.

Output unit

- Computers represent information in a specific binary form. Output units:
  - Interface with output devices.
  - Accept processed results provided by the computer in specific binary form.
  - Convert the information in binary form to a form understood by an output device.



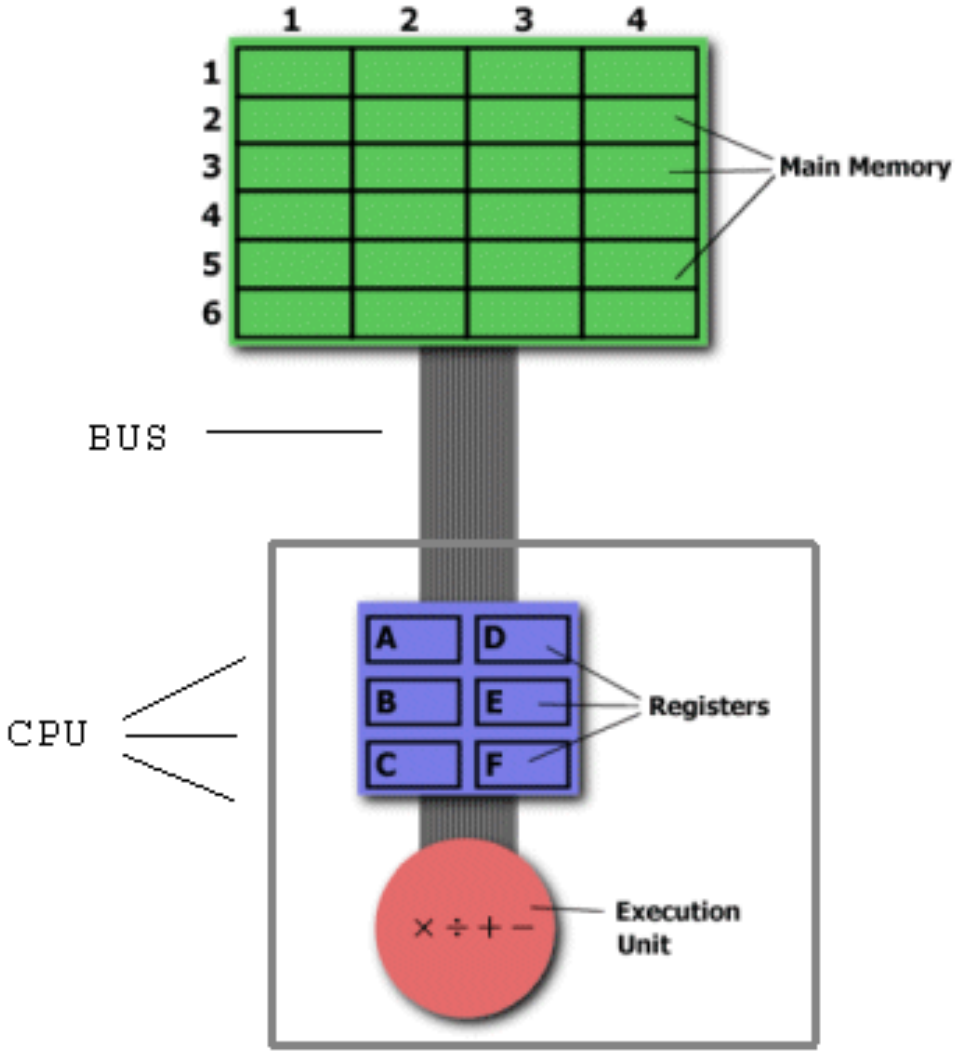
Control unit

- Operation of a computer can be summarized as:
  - Accepts information from the input units (Input unit).
  - Stores the information (Memory).
  - Processes the information (ALU).
  - Provides processed results through the output units (Output unit).
- Operations of Input unit, Memory, ALU and Output unit are coordinated by Control unit.
- Instructions control "what" operations take place (e.g. data transfer, processing).
- Control unit generates timing signals which determines "when" a particular operation takes place.

What are Registers?

- A register is a small amount of storage available on the CPU whose contents can be accessed more quickly than storage available elsewhere (such as main memory)
- They are the fastest way for the CPU to access data, since the CPU doesn't have to utilize the address bus to get data from main memory
- Standard computers today come with processors that use the x86 architecture, using 32-bit registers for data and only a few register files

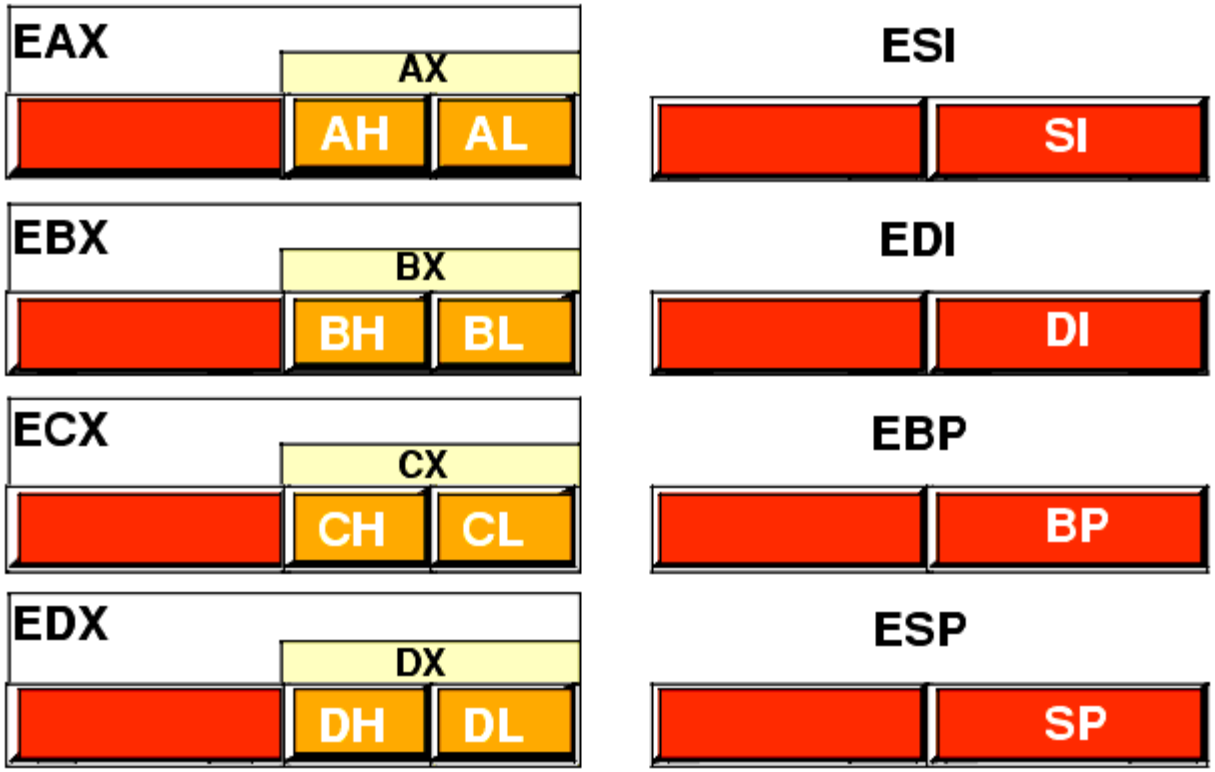
Layout of CPU and Memory



What are Register Files?

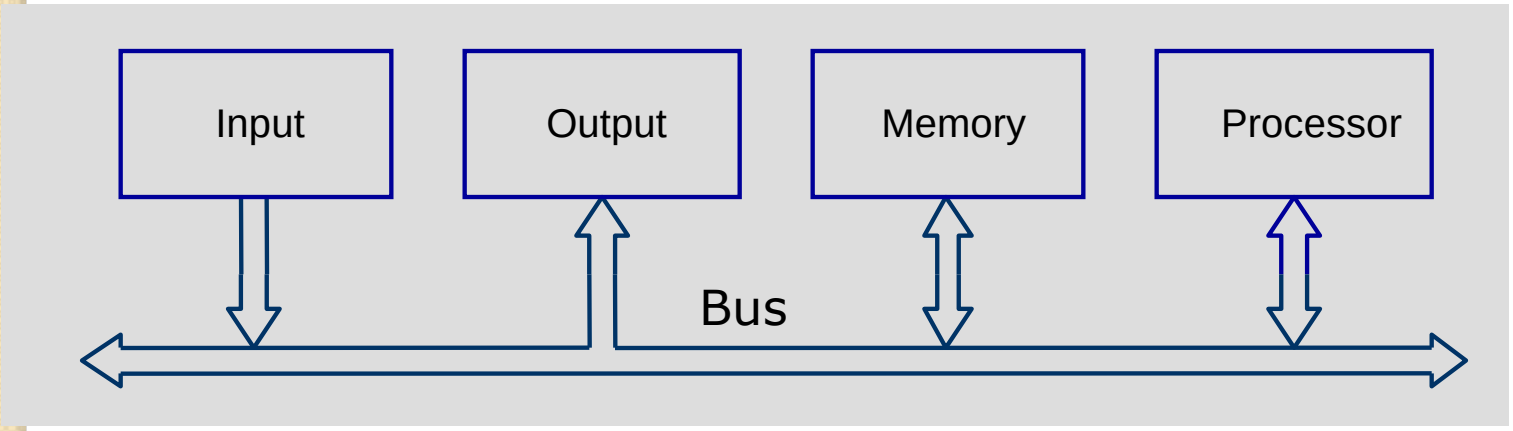
- Register Files are basically arrays of specific registers

- On standard x86 architecture, the 8 general purpose registers (often used in assembly programming) would be one register file.



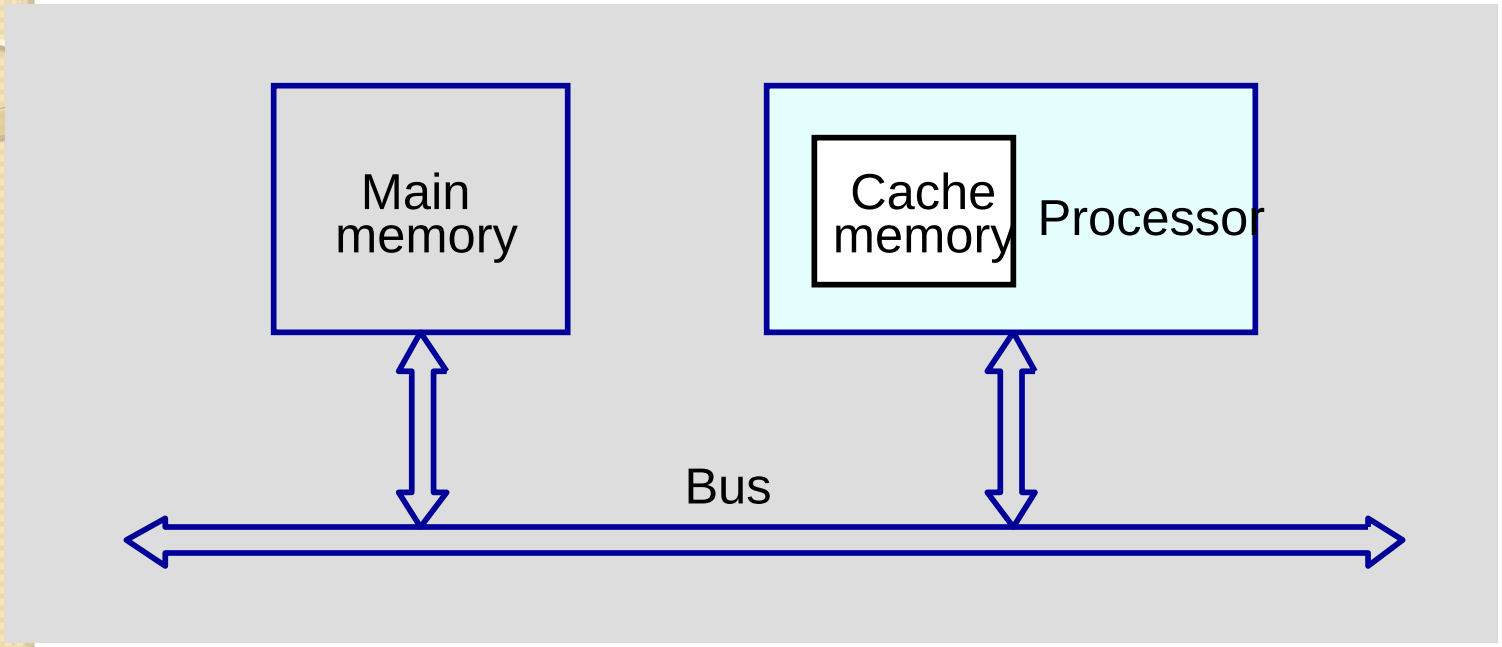
How are the functional units connected?

- For a computer to achieve its operation, the functional units need to communicate with each other.
- In order to communicate, they need to be connected.



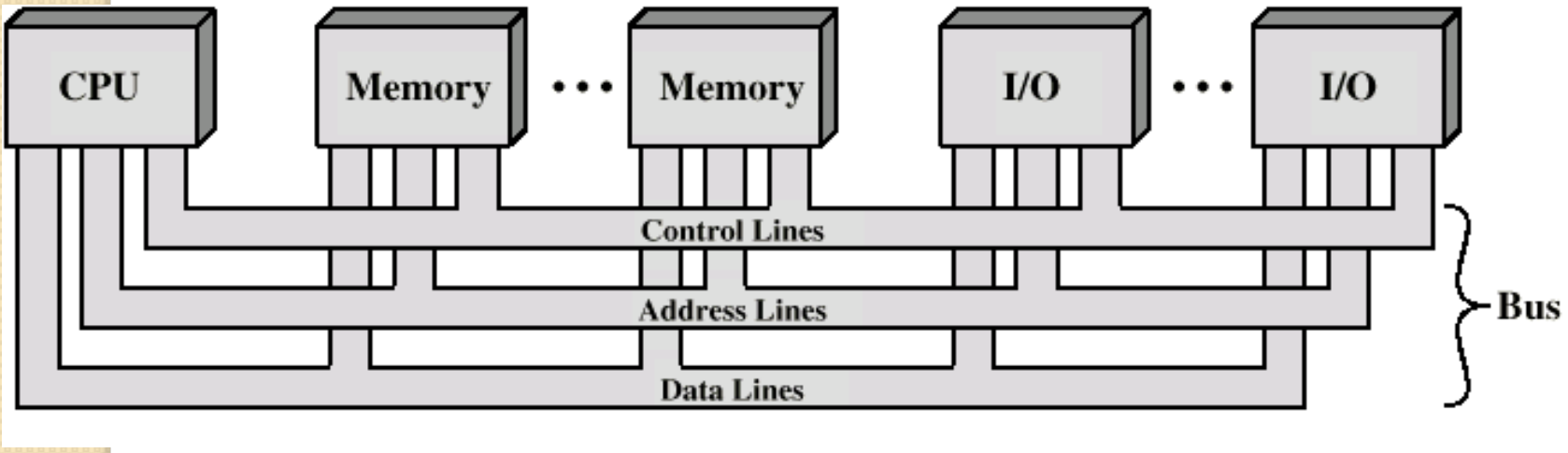
- Functional units may be connected by a **group of parallel wires**.
- The group of parallel wires is called a bus.
- Each wire in a bus can transfer one bit of information.
- The number of parallel wires in a bus is equal to the **word length of a computer**

Organization of cache and main memory



Why is the access time of the cache memory lesser than the access time of the main memory?

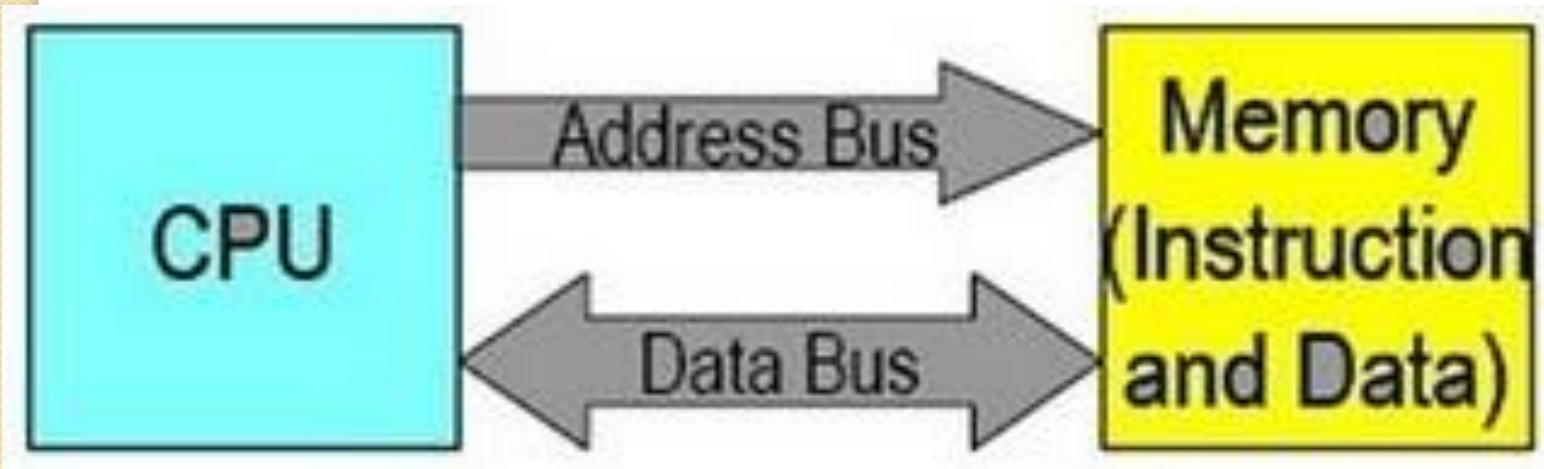
Bus Interconnection



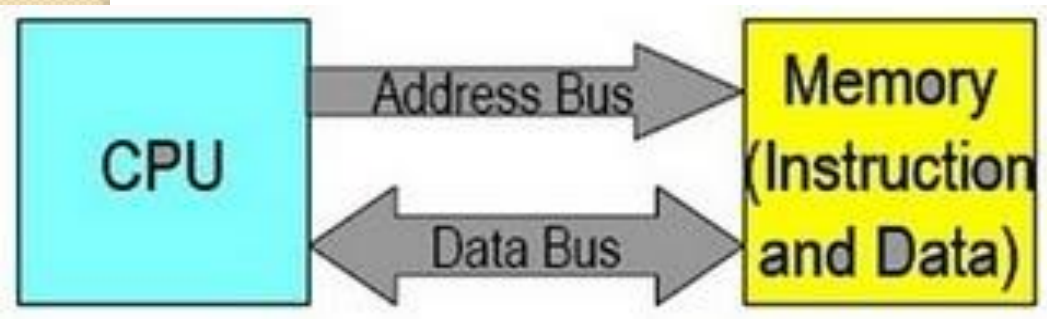
Von Neumann and Harvard architectures

- Von Neumann
  - Allows instructions and data to be mixed and stored in the same memory module
  - More flexible and easier to implement
  - Suitable for most of the general purpose processors

- Harvard:
  - Uses separate memory modules for instructions and for data
  - It is easier to pipeline
  - Higher memory throughput
  - Suitable for DSP (Digital Signal Processors)


CPU
Address Bus
Data Bus
Memory
(Instruction
and Data)

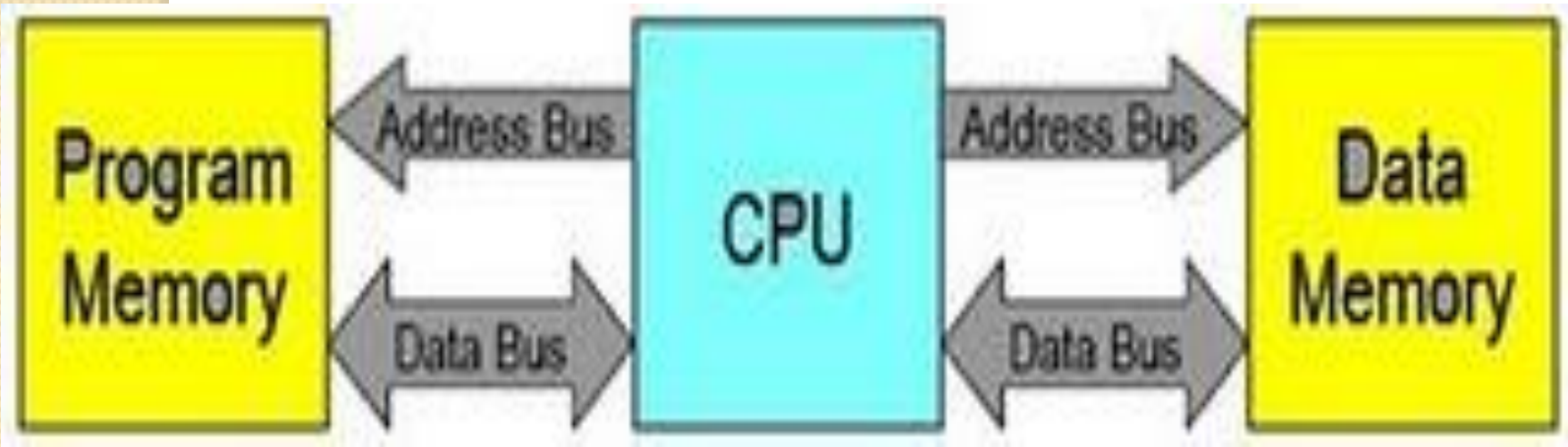
Von Neumann Architecture



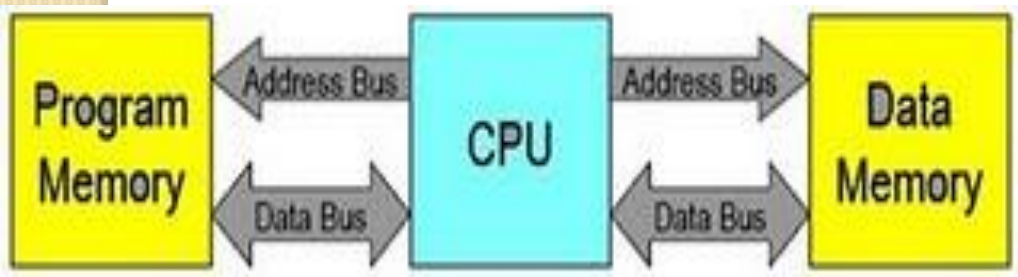
John von Neumann

- **Von Neumann architecture** has a single storage structure to hold both instructions and data. The CPU can be either reading an instruction or reading/writing data from/to the memory because instructions and data use the same bus system.

- The phrase Von Neumann architecture derives name of the mathematician and early computer scientist John von Neumann.


Program Memory
Address Bus
Data Bus
CPU
Address Bus
Data Bus
Data Memory

Harvard Architecture



Howard Hathaway Aiken

- The **Harvard architecture** is a computer architecture with physically separate storage and signal pathways for instructions and data.
- The term originated from the Harvard Mark I relay-based computer, which stored instructions on punched tape (24 bits wide) and data in electro-mechanical counters.

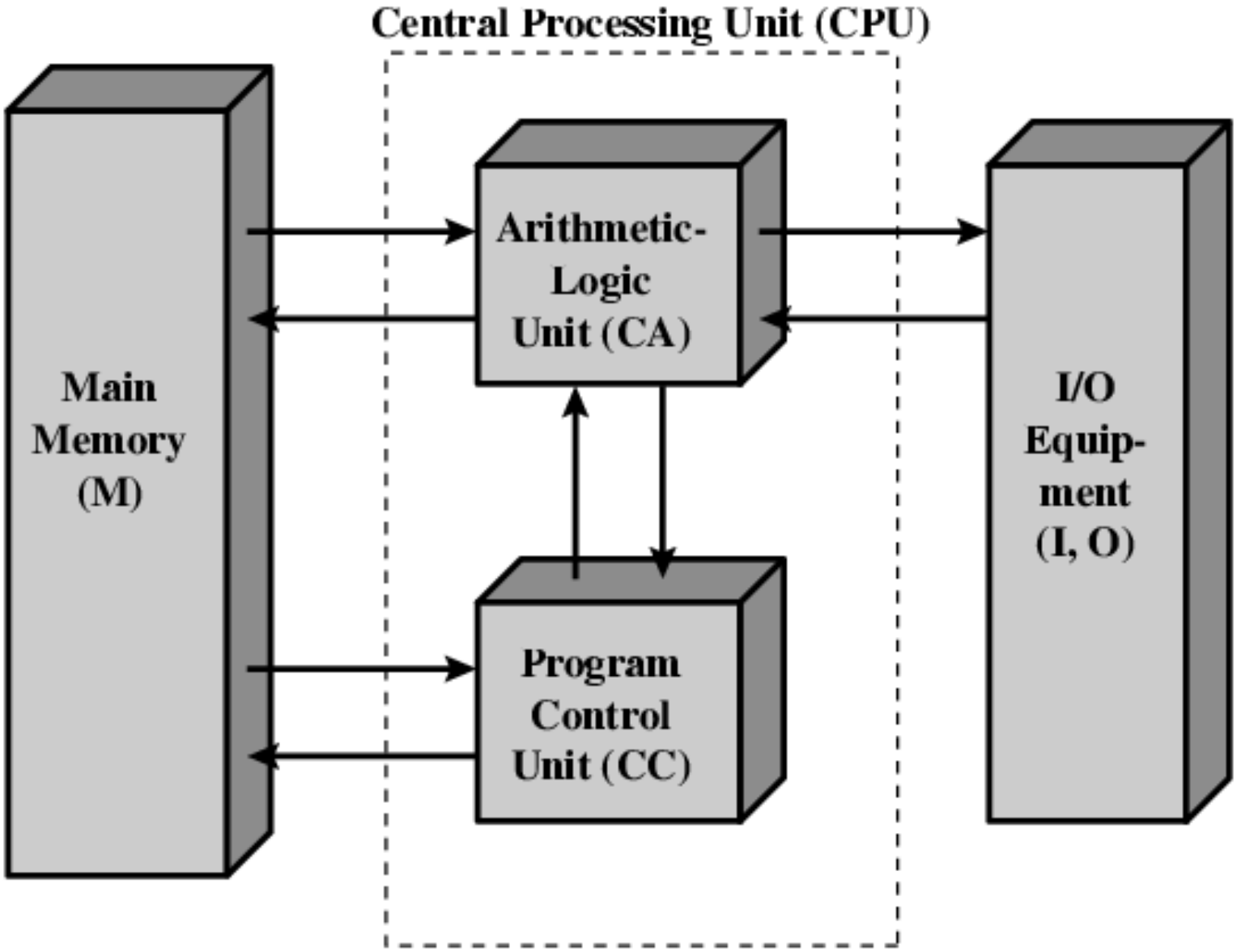
Organization of the von Neumann machine

- John von Neumann in the 1940s

Von Neumann/Turing

- Stored Program concept
- Main memory storing programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit
- Princeton Institute for Advanced Studies
  - IAS
- Completed 1952

Structure of von Neumann machine



Von Neumann Architecture

There are 3 major units in a computer tied together by buses:

1) Memory The unit that stores and retrieves instructions and data.

2) Processor: The unit that houses two separate components:

   The control unit: Repeats the following 3 tasks

   Fetches an instruction from memory

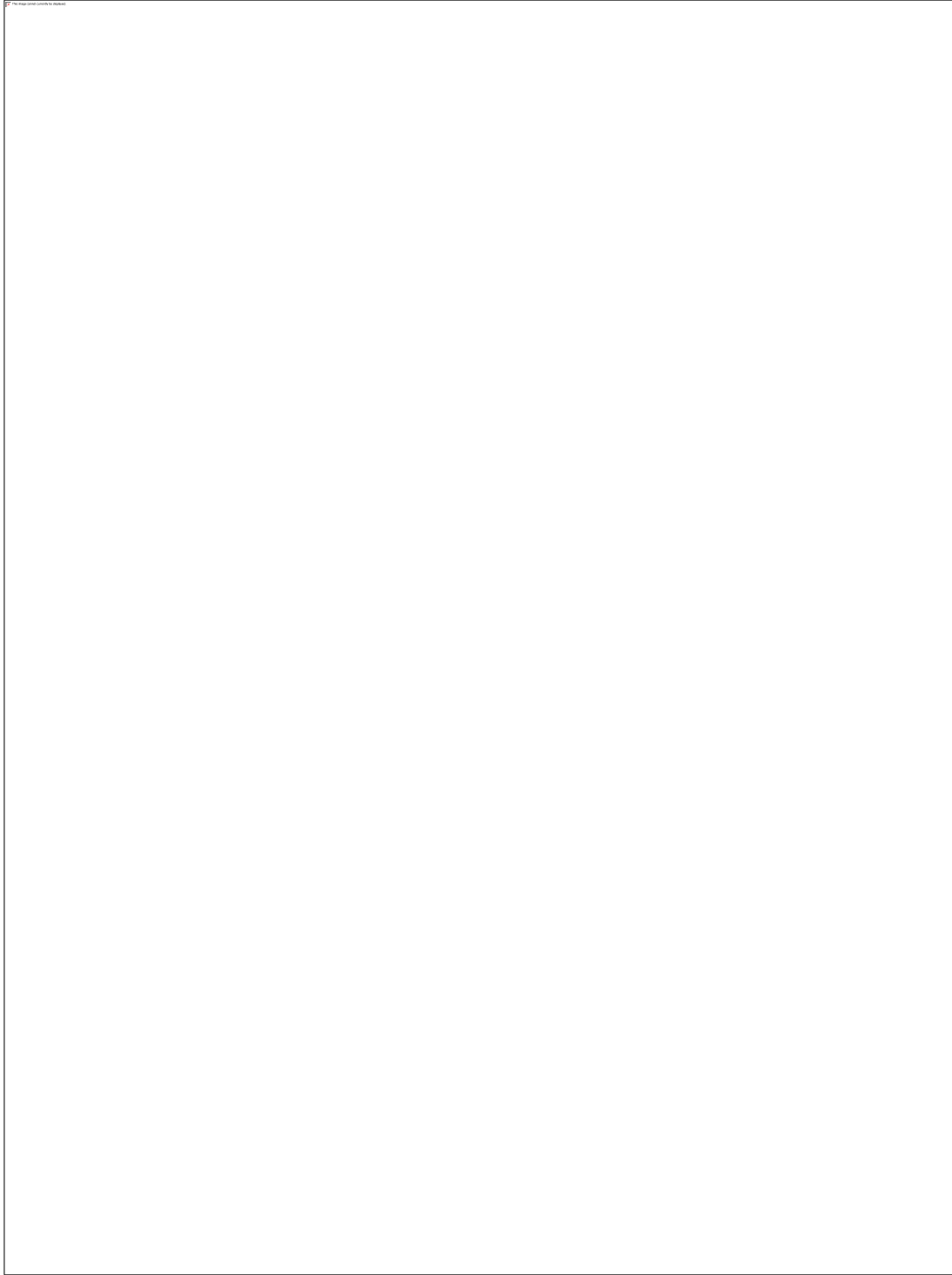
   Decodes the instruction

   Executes the instruction

   The arithmetic/logic unit (ALU): Performs mathematical and logical operations.

3) Input/Output (I/O) Units: Handle communication with the outside world.

Structure of IAS – detail



Registers

Main Memory Unit (Registers) – Accumulator (AC): Stores the results of calculations made by ALU.

Program Counter (PC):
Keeps track of the memory location of the next instructions to be dealt with. The PC then passes this next address to Memory Address Register (MAR).

Memory Address Register (MAR):
It stores the memory locations of instructions that need to be fetched from memory or stored into memory.

Register

Memory Data Register (MDR):
It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.

Current Instruction Register (CIR):
It stores the most recently fetched instructions while it is waiting to be coded and executed.

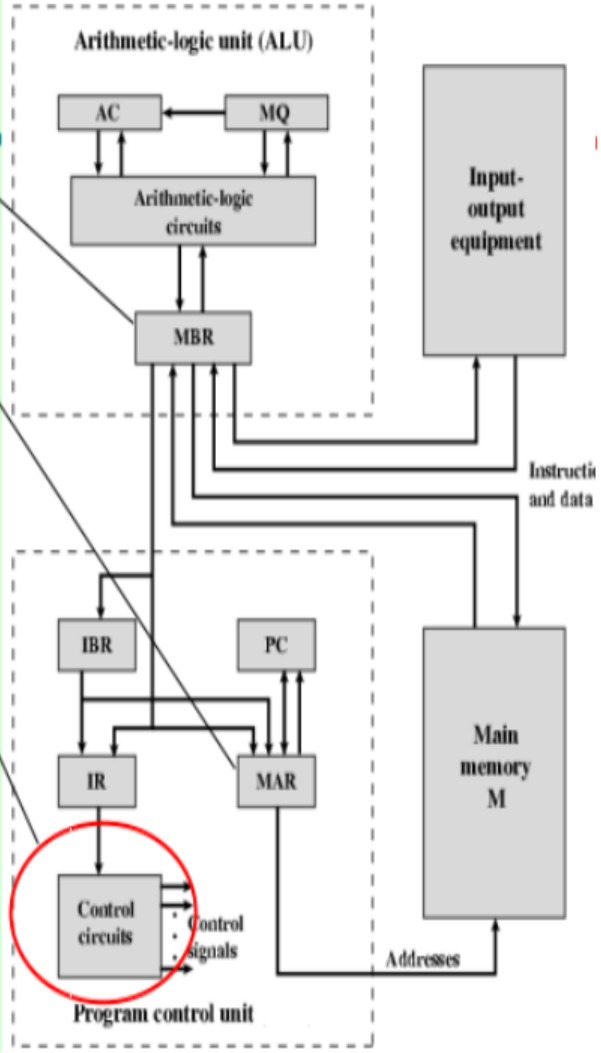
Instruction Buffer Register (IBR):
The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.

IAS organization

Memory Buffer Register either sends data to or receives data from Mem. or I/O

Memory Address Register specifies which Mem. location will be read or written next

Control signals are set by the opcode part of the instruction bits.
Examples:
- Bring a new instruction from Mem. (fetch)
- Perform an addition (execute)



Comparison of von Neumann and Harvard Architecture

Von Neumann Architecture	Harvard Architecture
It is a conceptual design based on stored program architecture	It is a modern computer architecture
Memory is common for both instruction and data	Separate memory is there for instruction and data
Common bus between memory and CPU for both instruction and data	Separate buses for instruction memory and data memory
Two clock cycles required for the processor to execute an instruction	Only one clock cycle is enough for the processor to execute an instruction
Data transfer and instruction fetches cannot be done simultaneously	Data transfer and instruction fetches can be done simultaneously
Control unit design is simple	Control unit for two buses is more complicated which increses the development cost

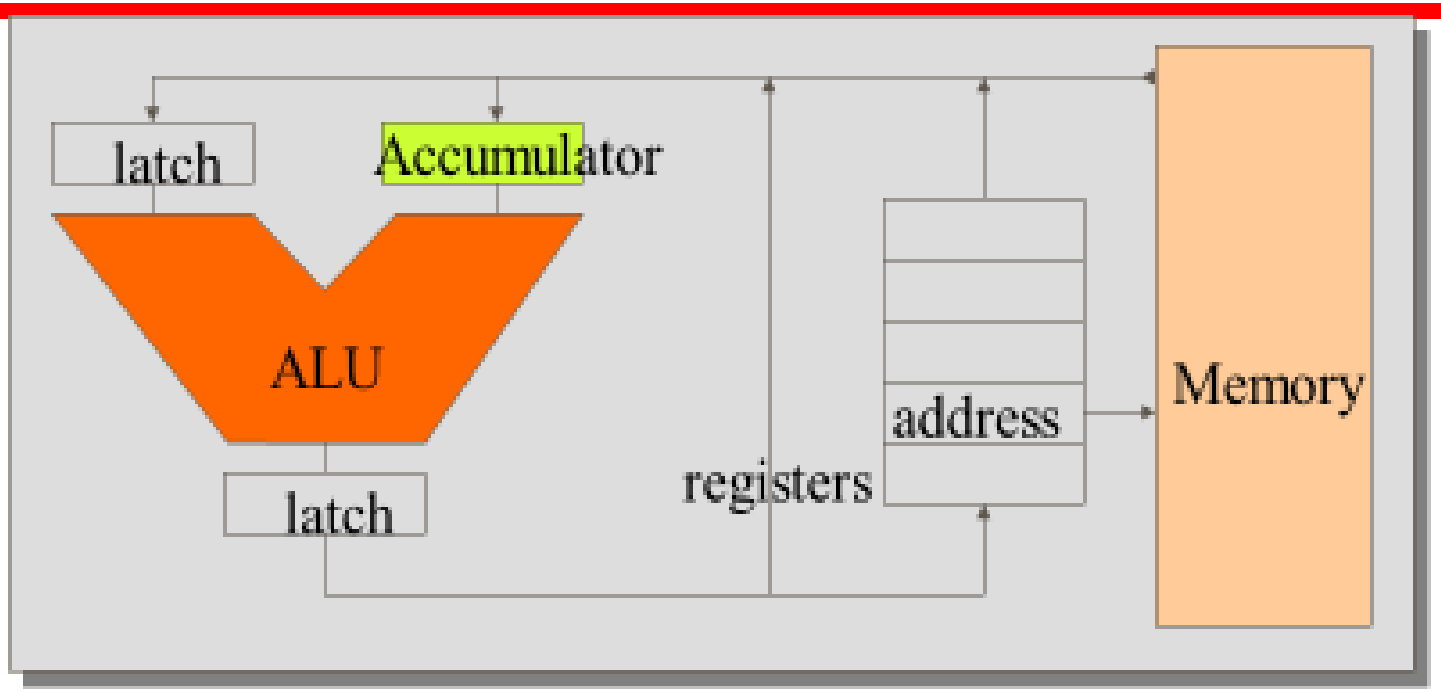
Different Types of Architecture Classification

- Accumulator based
- Stack based
- RISC
- CISC

Instruction Format

(a) Instruction format

Accumulator architecture



Example code: `a = b+c;`

```
load  b;     // accumulator is implicit operand
add   c;
store a;
```

C	6
b	5
a	11

- a= b+ c; b= 5, c=6
- "Value of b to be added with value of c and the result is stored in a"
- Variable in memory locations

- Load b (value of b → accumulator) AC=5 → Data transfer instr
- ADD C (AC= AC+"C") (C=6) (AC= 5 +6) (AC=11) → Arthimetic
- STORE a (a=AC) → Data instructions

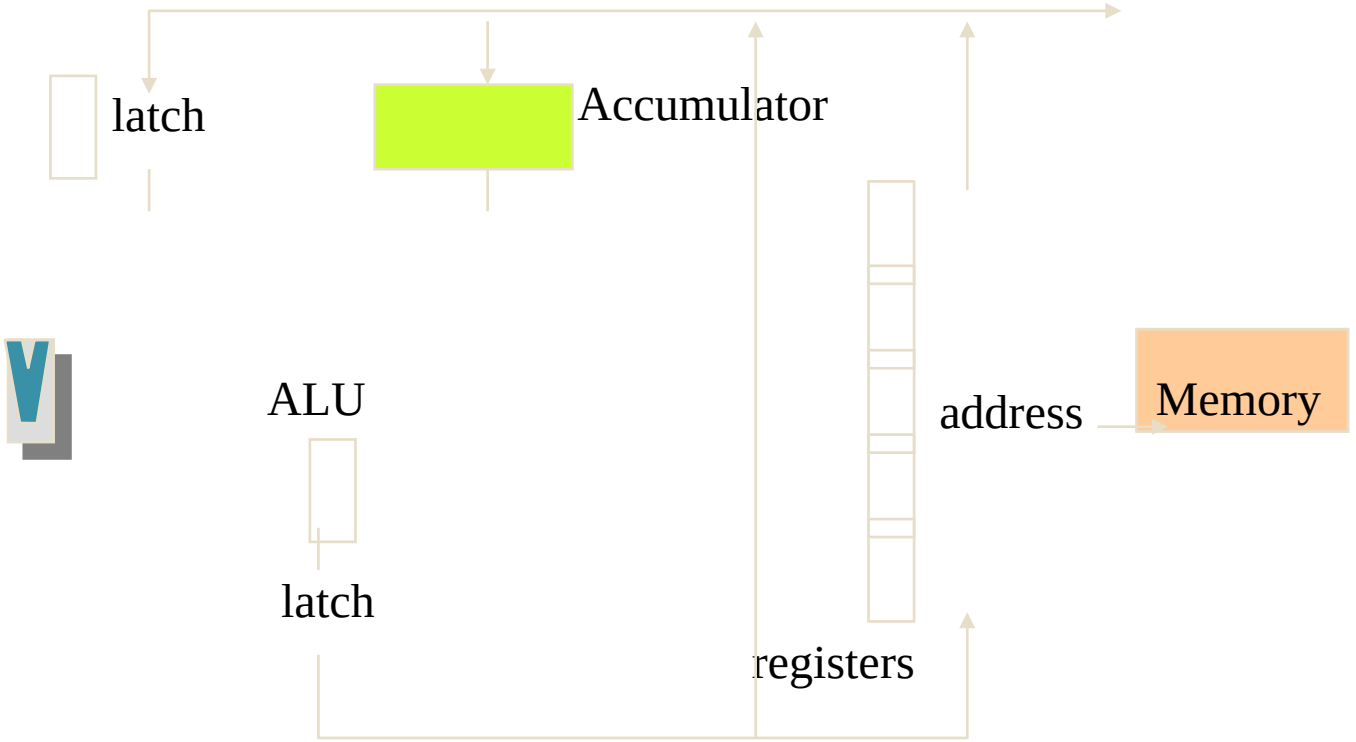
ACCUMULATOR

One Address Instructions –
This use a implied ACCUMULATOR register for data manipulation.

One operand is in accumulator and other is in register or memory location.

Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.

Accumulator architecture

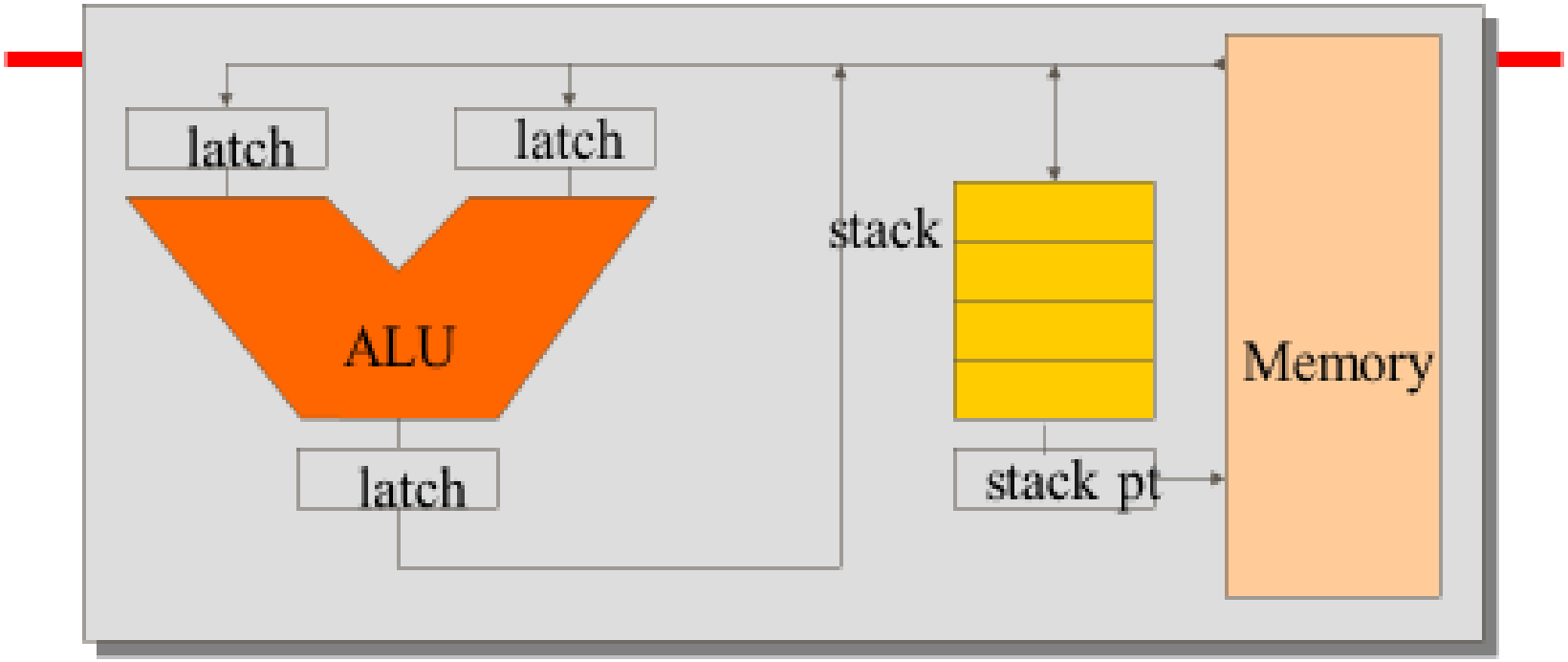


Example code: `a = b+c;`

```
load  b;     // accumulator is implicit operand
add   c;
store a;
```

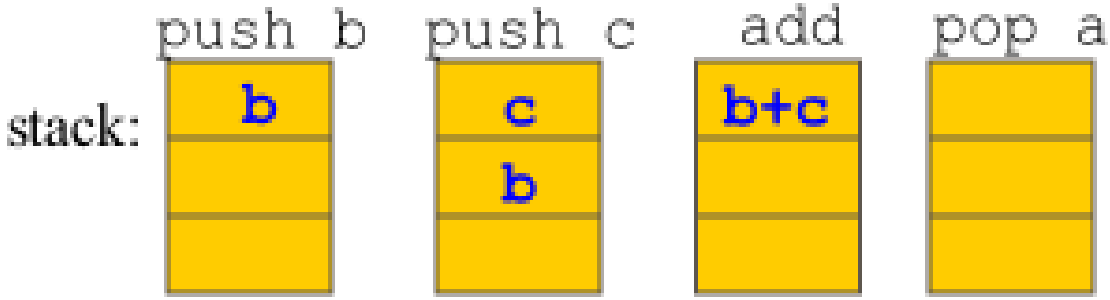
- Write a Accumulator based programming for squaring the number”5” and store the result in y.
- The memory variable x contains 5 ,
- Such as x=5

Stack architecture



Example code: `a = b+c;`

```
push b;
push c;
add;
pop  a;
```

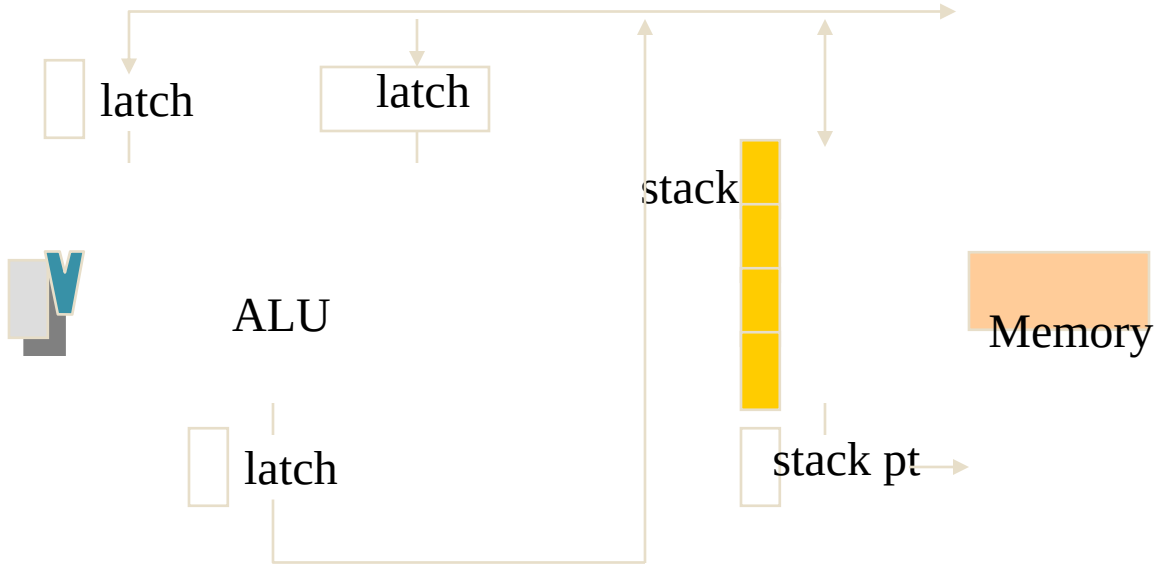


Stack Based – Zero Address

- A stack based computer do not use address field in instruction.
- To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

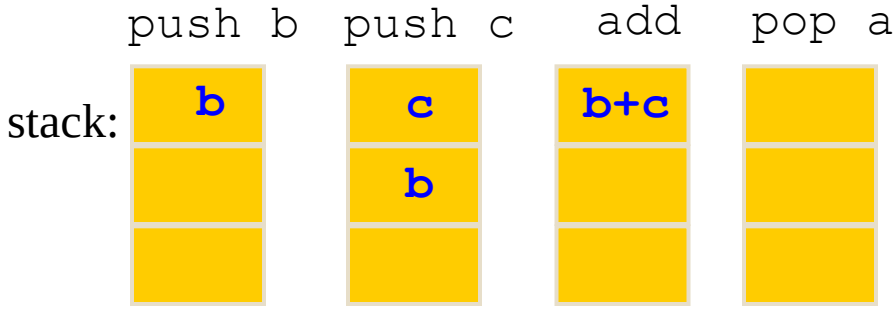
```
Expression: X = (A+B)*(C+D)
Postfixed : X = AB+CD+*
TOP means top of stack
M[X] is any memory location
```

Stack architecture



Example code: `a = b+c;`

```
push b;
push c;
add;
pop  a;
```



X = (A+B)*(C+D)

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

- Expression: X = (A+B)*(C+D)
- AC is accumulator
- M[] is any memory location
- M[T] is temporary location

X = (A+B)*(C+D)

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]
ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

Two Address Instructions –

This is common in commercial computers.
Here two address can be specified in the instruction.
Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

- Expression: X = (A+B)*(C+D)
- R1, R2 are registers
- M[] is any memory location

X = (A+B)*(C+D)

MOV	R1,A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

X = (A+B)*(C+D)

- MOV R1,A
- MOV R2, B
- ADD R1,R2
- MOV R2, C R2=M[C]
- ADD R2,D R2= R2 +M[D]

R1=A+B, R2 =C+D

MUL R1,R2 R1= R1*R2

MOV X,R1

Three Address Instructions –

- This has three address field to specify a register or a memory location.
- Program created are much short in size but number of bits per instruction increase.
- These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle

Try X=(A+B)*(C+D) Using three address instruction

- ADD R1,A,B R1= A+B
- ADD R2, C,D R2 = C+D
- MUL X, R1,R2 X= R1*R2

- Expression: X = (A+B)*(C+D)
- R1, R2 are registers
- M[] is any memory location

ADD	R1, A, B	R1 = M[A] + M[B]
ADD	R2, C, D	R2 = M[C] + M[D]
MUL	X, R1, R2	M[X] = R1 * R2

Summary

C = A + B

Stack Architecture	Accumulator Architecture	Register-Memory	Memory-Memory
`Push A`	`Load A`	`Load r1,A`	`Add C,B,A`
`Push B`	`Add B`	`Add r1,B`	
`Add`	`Store C`	`Store C,r1`	
`Pop C`			

CISC Feature

- Complex instruction set computer
- Large number of instructions (~200-300 instructions)
- Specialized complex instructions
- Many different addressing modes
- Variable length instruction format
- Memory to memory instruction
- For Example : 68000, 80x86

- x=y-z

- Accumulator
- Load y (AC=Y)
- Sub z (AC = Y-Z)
- Store x (X=AC)

- x=y-z

- Stack
- Push y (top of stack=Y)
- Push z (top of stack=z)
- Sub (Out (top of stack) Y-Z)
- pop x (X=> top of stack)

RISC Feature

- Reduced instruction set computer
- Relatively few number of instructions (~50)
- Basic instructions
- Relatively few different addressing modes
- Fixed length instruction format
- Only load/store instructions can access memory
- Large number of registers
- Hardwired rather than micro-program control
- For Example : MIPS, Alpha, ARM etc.

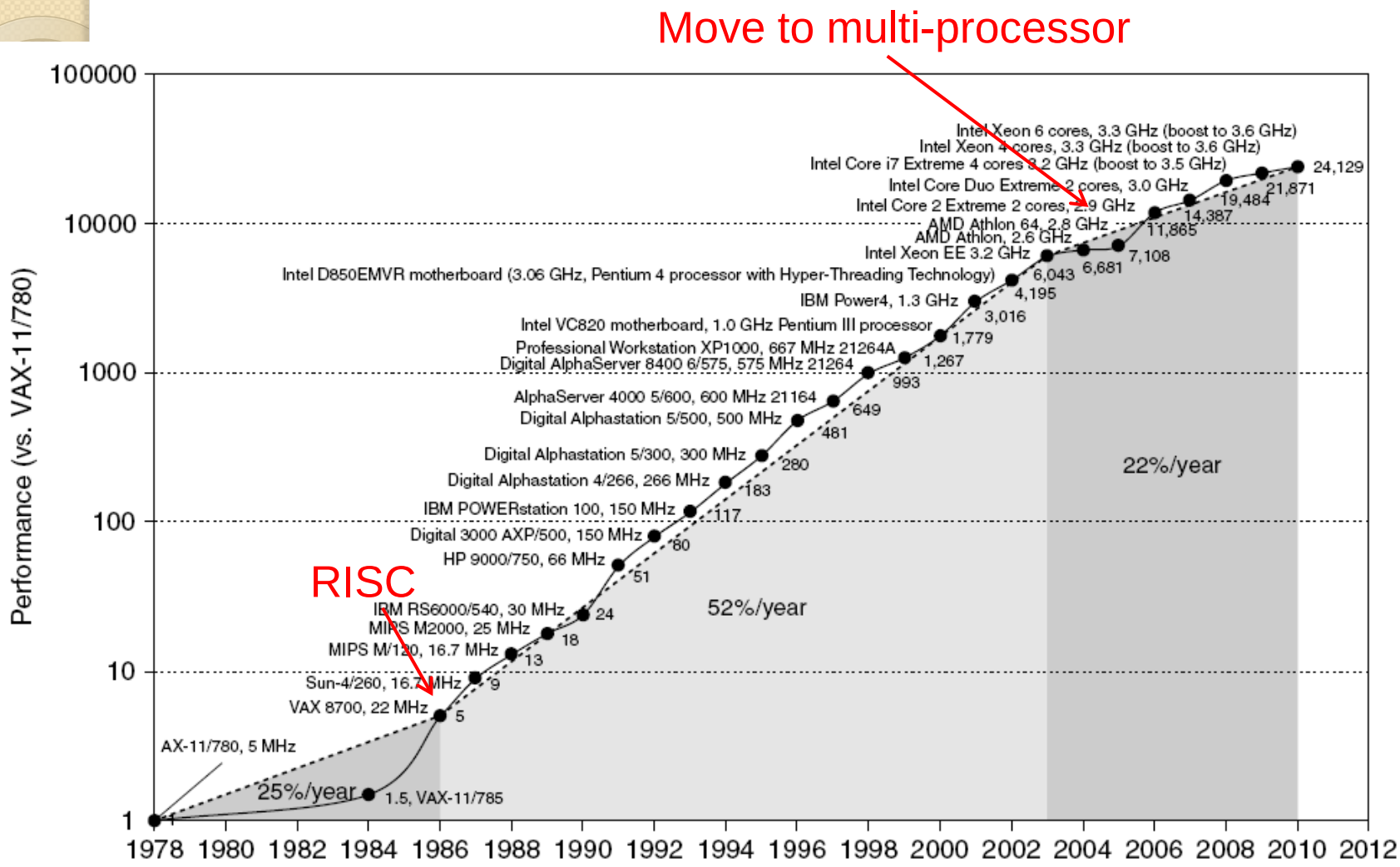
Example of CPU Architectures

- Intel: 80x86
- Motorola: 680x0
- Sun : Sparc
- Silicon Graphics : MIPS
- HP : PA-RISC
- IBM: Power PC
- Compaq: Alpha

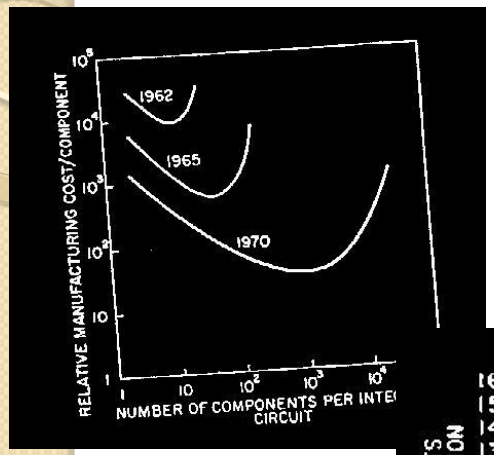
CISC

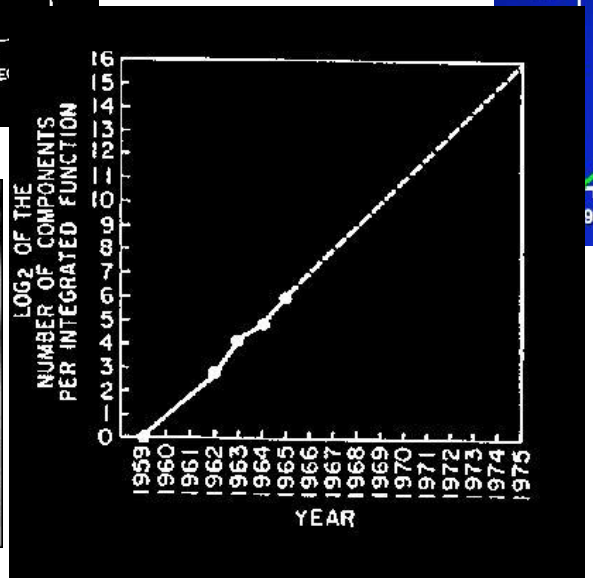
RISC

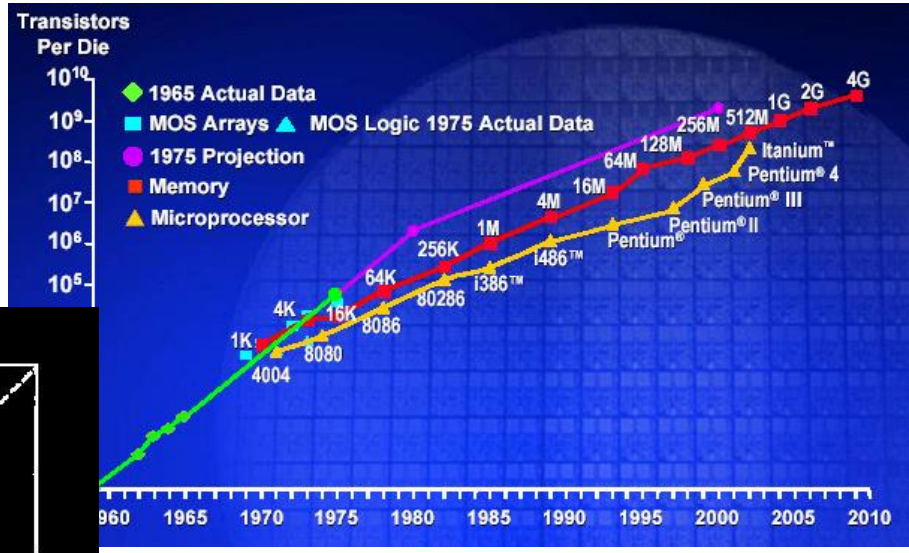
Single Processor Performance

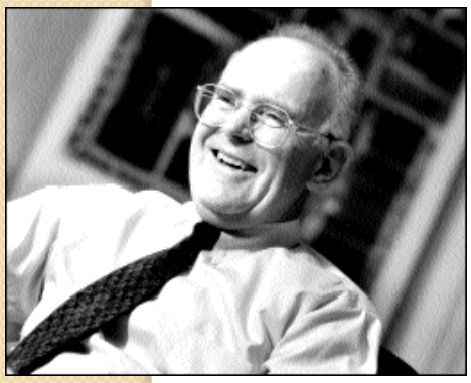


Moore's Law: 2X transistors / "year"









- "Cramming More Components onto Integrated Circuits"
  - Gordon Moore, Electronics, 1965
- # on transistors / cost-effective integrated circuit double every N months ($12 \leq N \leq 24$)

What is CISC….?

- A complex instruction set computer (CISC, pronounced like "*sisk*") is a microprocessor instruction set architecture (ISA) in which each instruction can execute several low-level operations, such as a load from memory, an arithmetic operation, and a memory store, all in a single instruction.

Main Idea of CISC

- The primary goal of the CISC is to complete a task in few lines of assembly instruction as possible.

RISC

- Reduced Instruction Set Computer
  - Small number of instructions
  - instruction size constant
  - bans the indirect addressing mode
  - retains only those instructions that can be overlapped and made to execute in one machine cycle or less.

RISC Architecture

- Small, highly optimized set of instructions
- Uses a load-store architecture
- Short execution time
- Pipelining
- Many registers

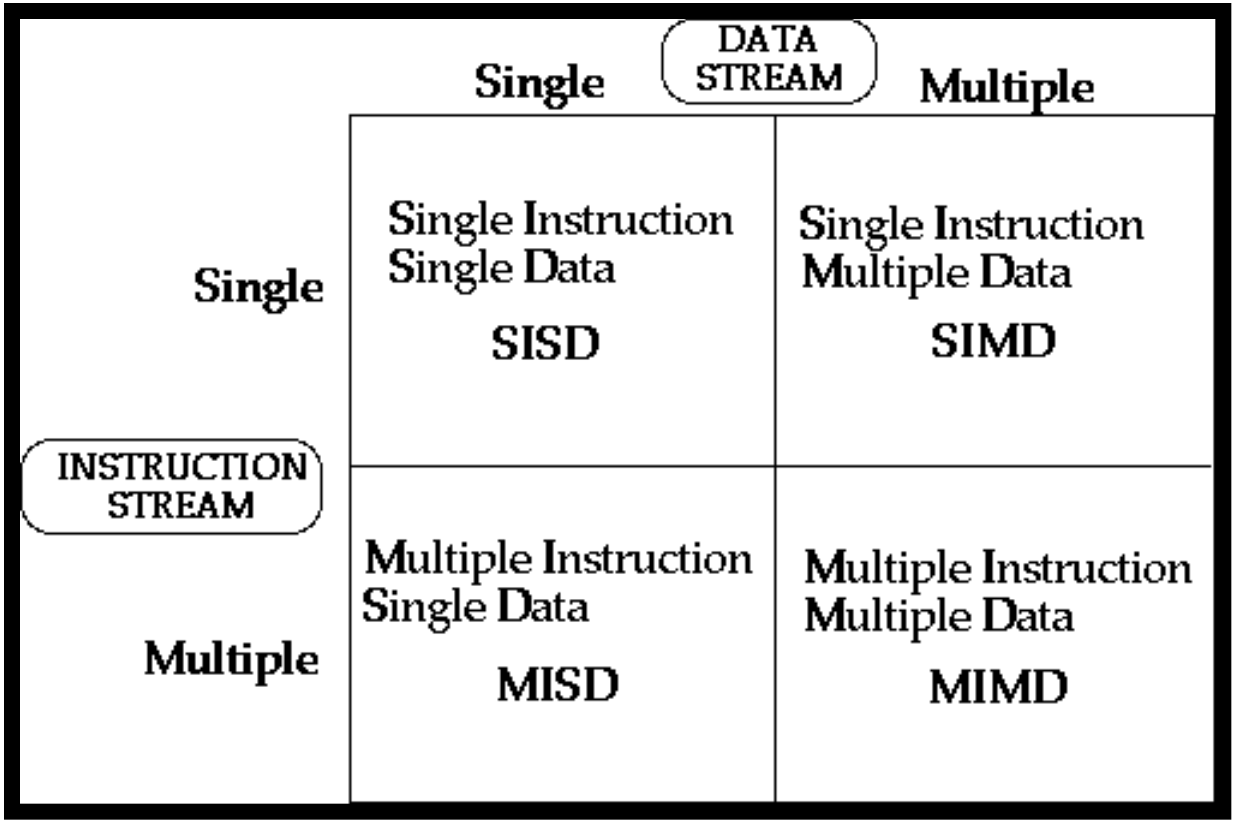
RISC Examples

- Apple iPods (custom ARM7TDMI SoC)
- Apple iPhone (Samsung ARM1176JZF)
- Palm and PocketPC PDAs and smartphones (Intel XScale family, Samsung SC32442 - ARM9)
- Nintendo Game Boy Advance (ARM7)
- Nintendo DS (ARM7, ARM9)
- Sony Network Walkman (Sony in-house ARM based chip)
- Some Nokia and Sony Ericsson mobile phones

Classification of models

Taxonomy of Computer Architectures

Simple Diagrammatic Representation



4 categories of Flynn's classification of multiprocessor systems by their instruction and data streams

Computing…

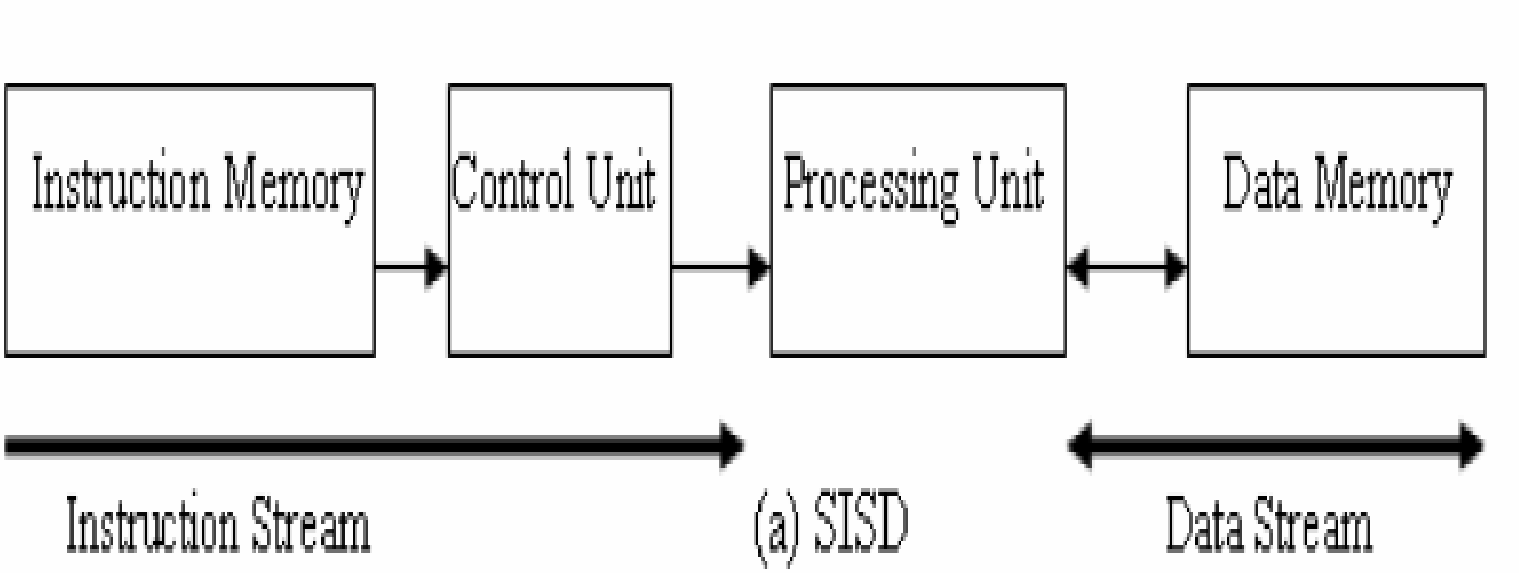
- Instruction that operate on data
- Based on Instruction Stream & Data Stream, Classification done by FLYNN's
  - SISD
  - SIMD
  - MISD
  - MIMD

Single Instruction, Single Data (SISD)

- SISD machines executes a single instruction on individual data values using a single processor.
- Based on traditional Von Neumann uniprocessor architecture, instructions are executed sequentially or serially, one step after the next.
- Until most recently, most computers are of SISD type.

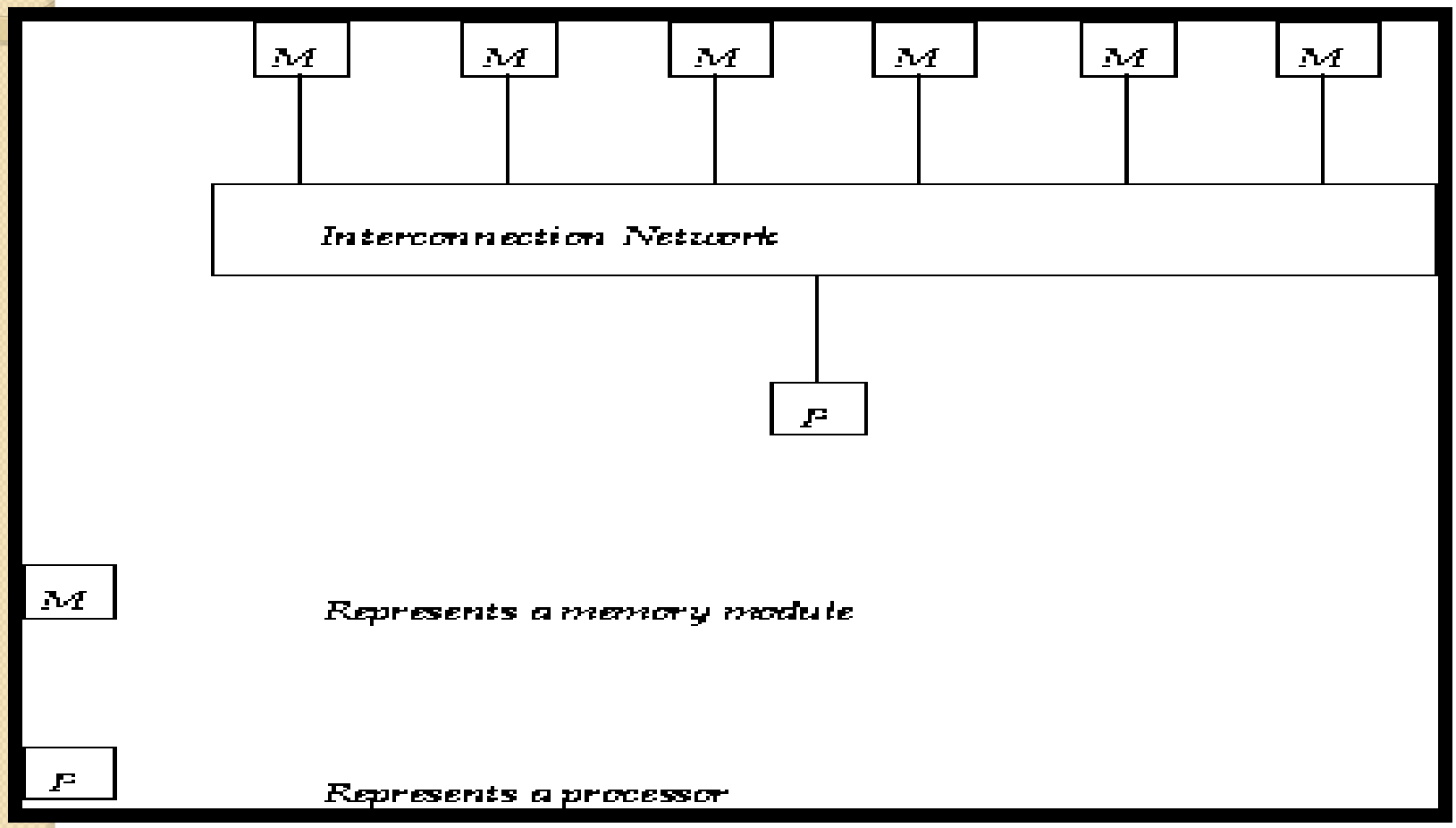
Single Instruction Single Data

- At any time, one Instruction operates on one set of Data



SISD

Simple Diagrammatic Representation

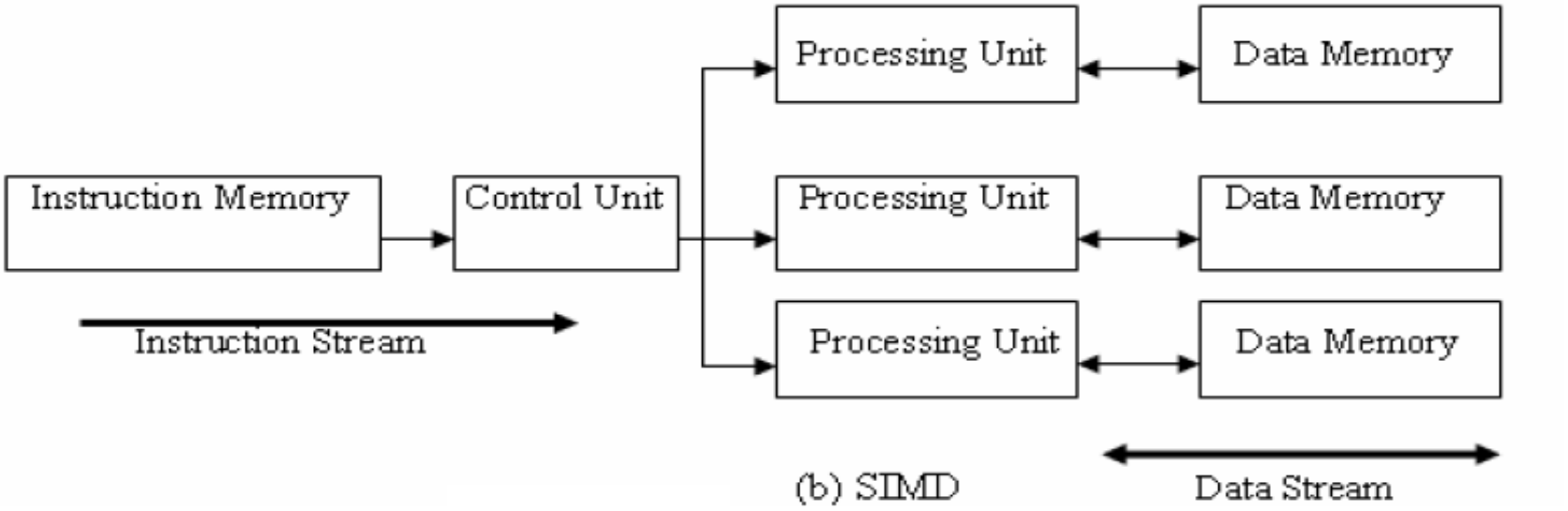


Single Instruction, Multiple Data (SIMD)

- An SIMD machine executes a single instruction on multiple data values simultaneously using many processors.
- Since there is only one instruction, each processor does not have to fetch and decode each instruction. Instead, a single control unit does the fetch and decoding for all processors.
- SIMD architectures include array processors.

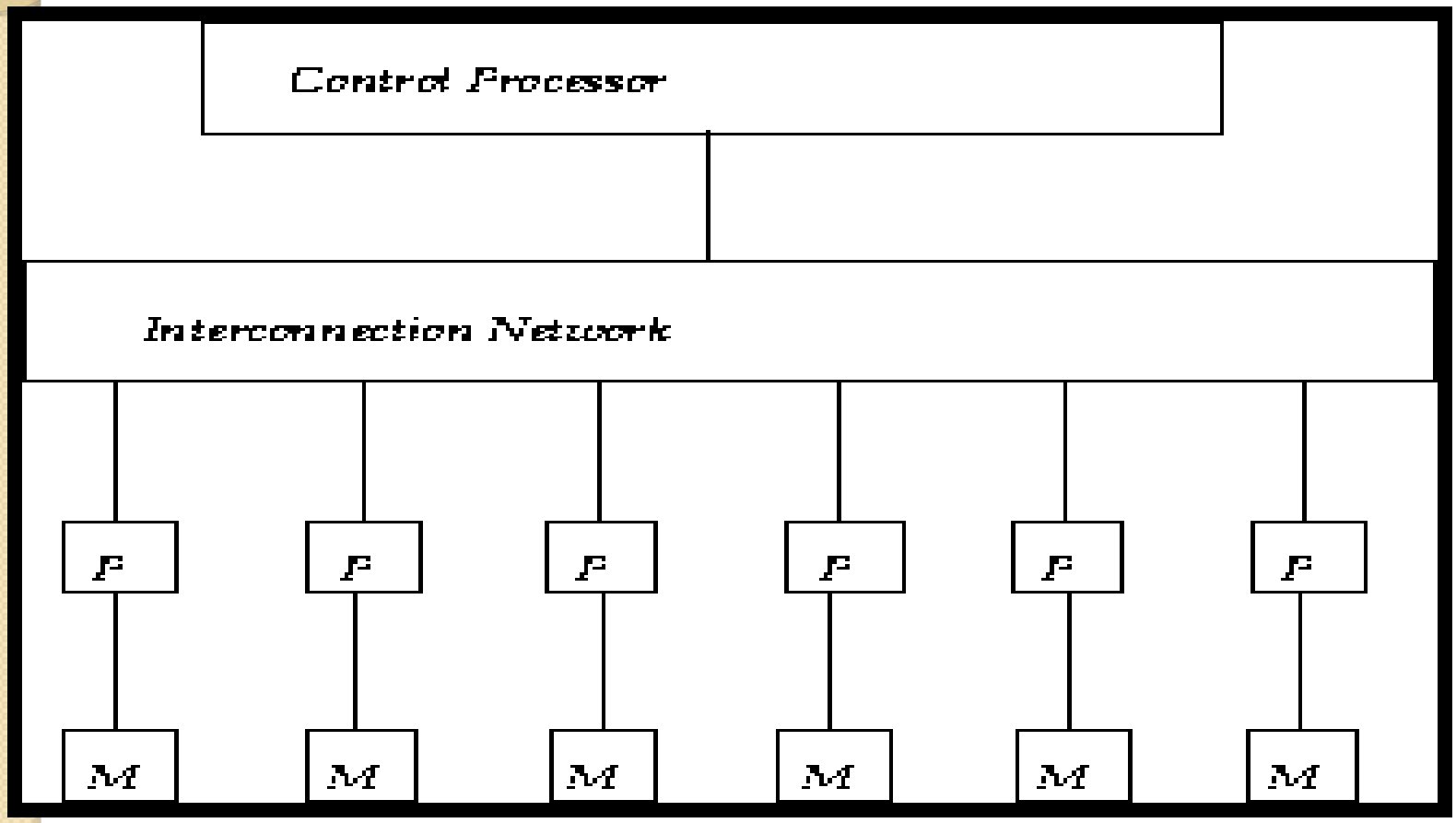
Single Instruction Multiple Data

- At any time, one instruction operates on many data
  - Data parallel architecture
  - Array processors



SIMD

Simple Diagrammatic Representation

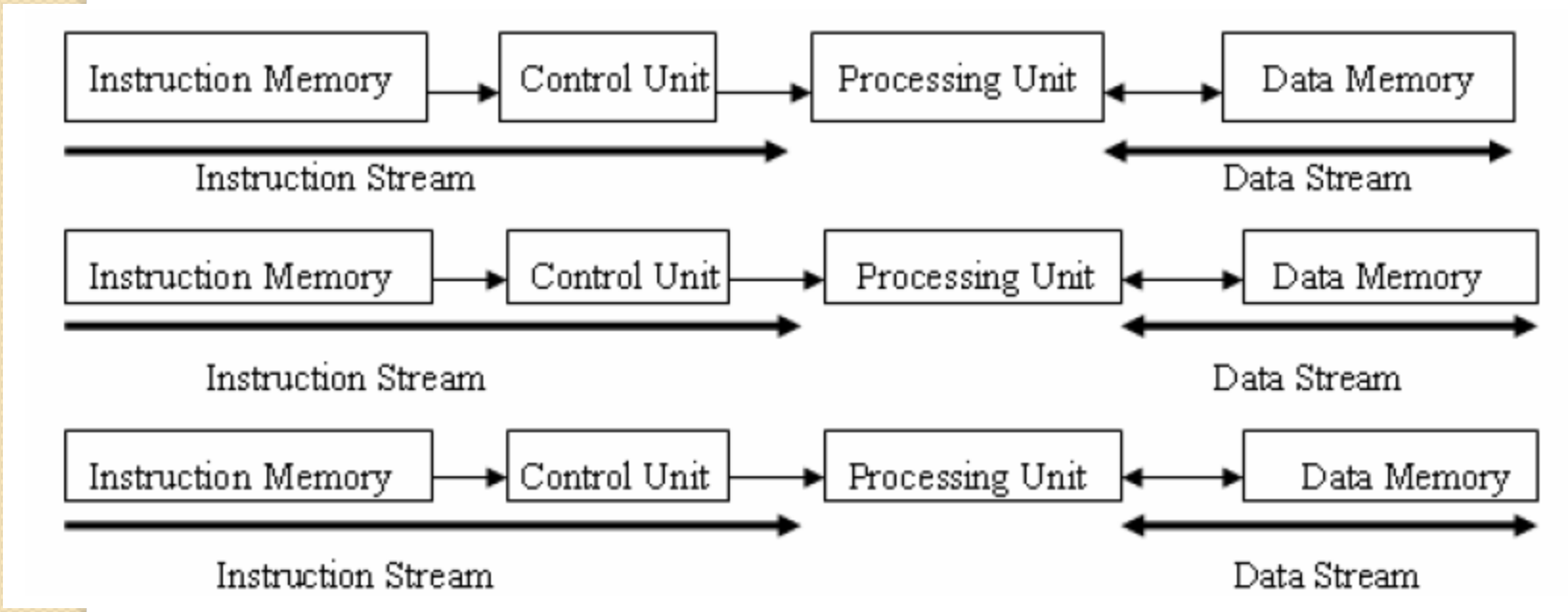


Multiple Instruction, Multiple Data (MIMD)

- MIMD machines are usually referred to as multiprocessors or multicomputers.
- It may execute multiple instructions simultaneously, contrary to SIMD machines.
- Each processor must include its own control unit that will assign to the processors parts of a task or a separate task.
- It has two subclasses: Shared memory and distributed memory

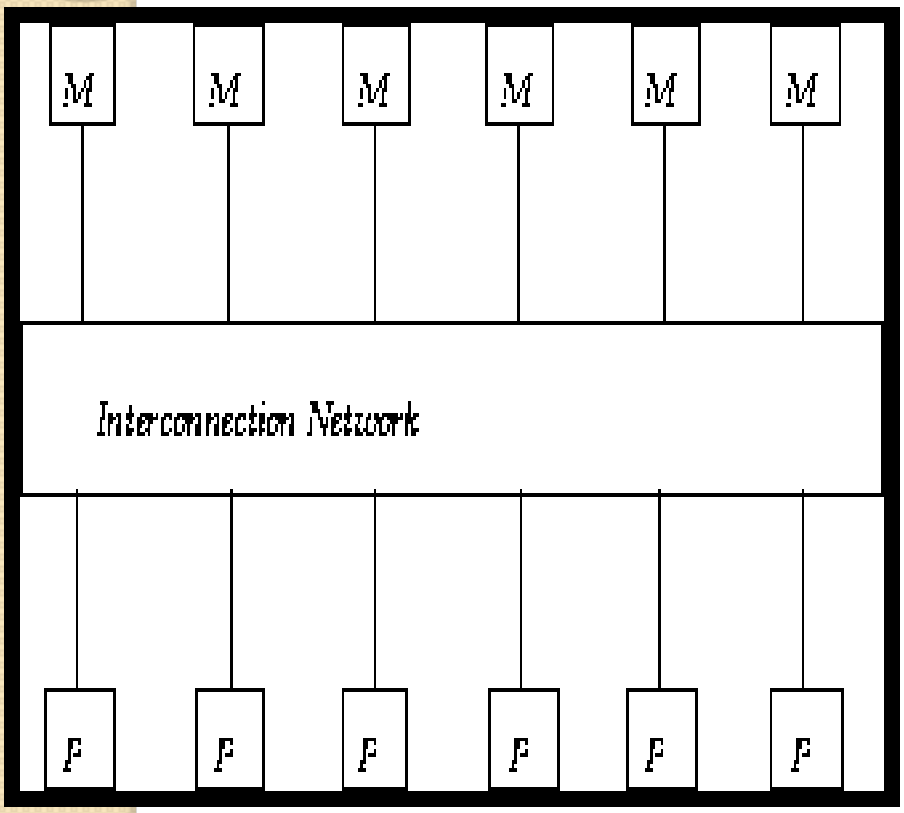
Multiple Instruction Multiple Data

Classical distributed memory or SMP architectures

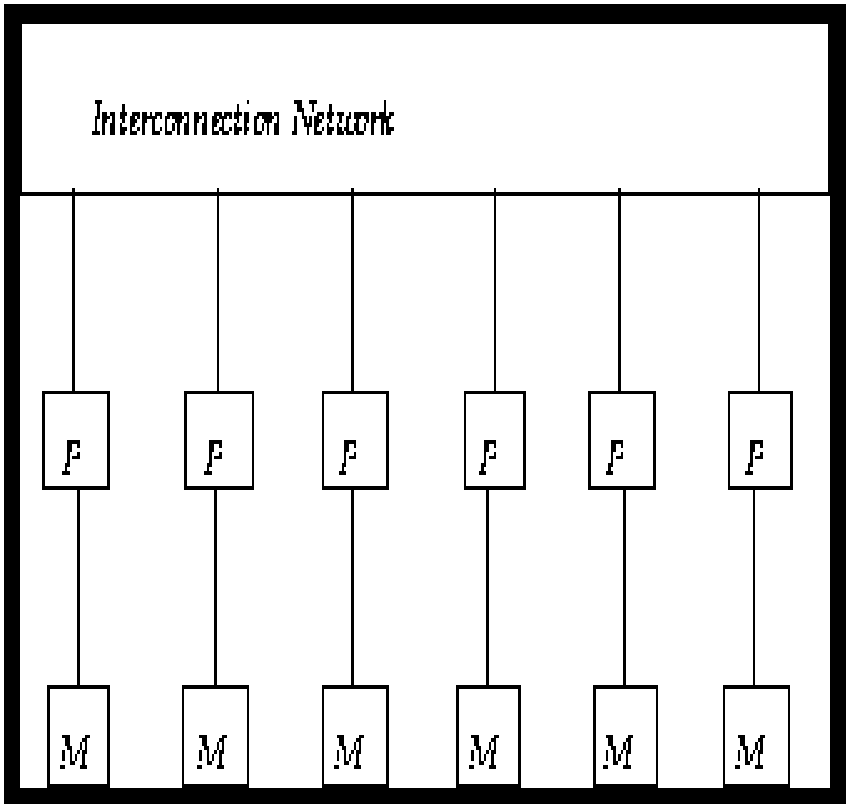


MIMD

Simple Diagrammatic Representation (Shared Memory)

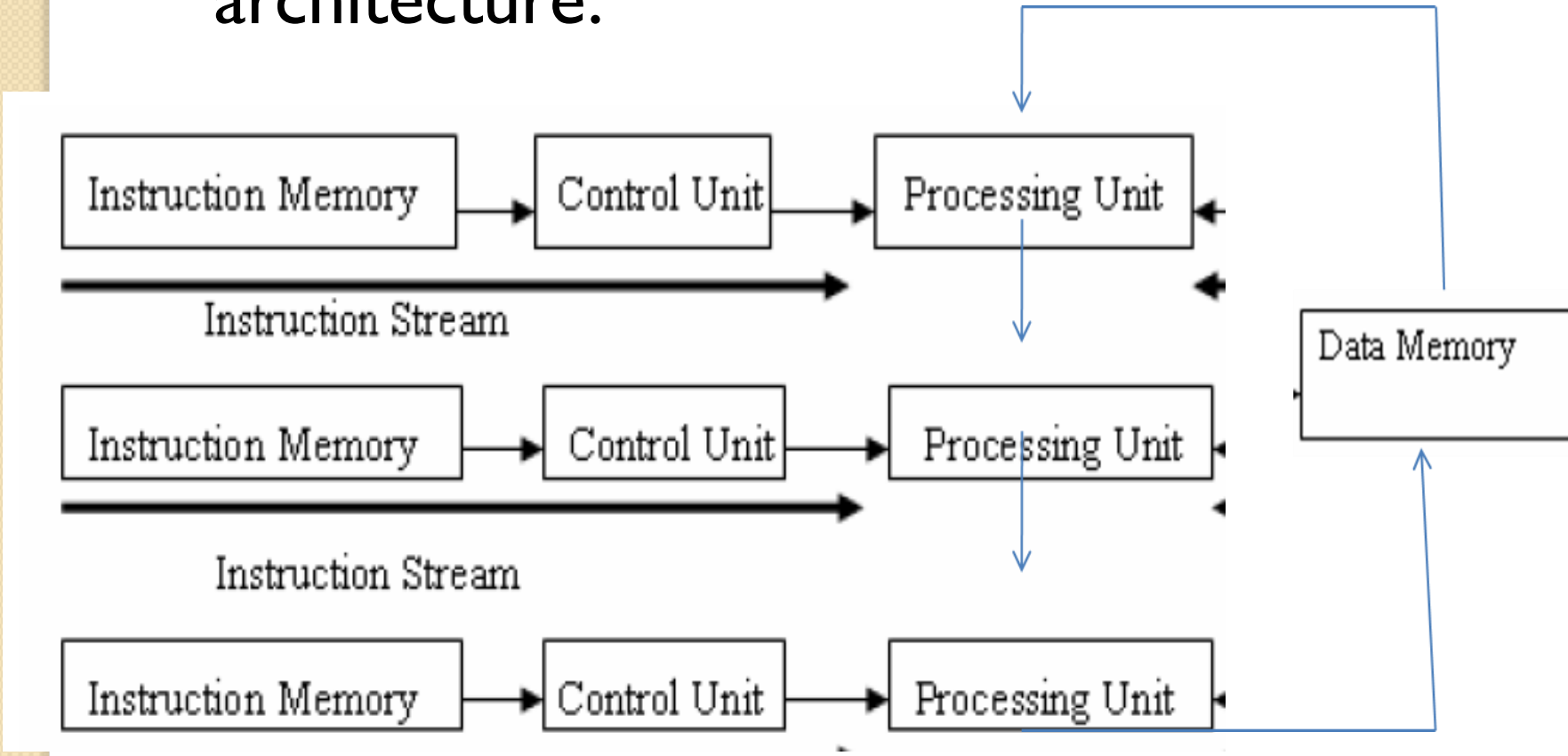


Simple Diagrammatic Representation(DistributedMemory)



Multiple Instruction Single Data

- Systolic array is one example of an MISD architecture.



- Example Z = sin(x)+cos(x)+tan(x)
- The system performs different operations on the same data set.
- Machines built using the MISD model are not useful in most of the application, a few machines are built, but none of them are available commercially.

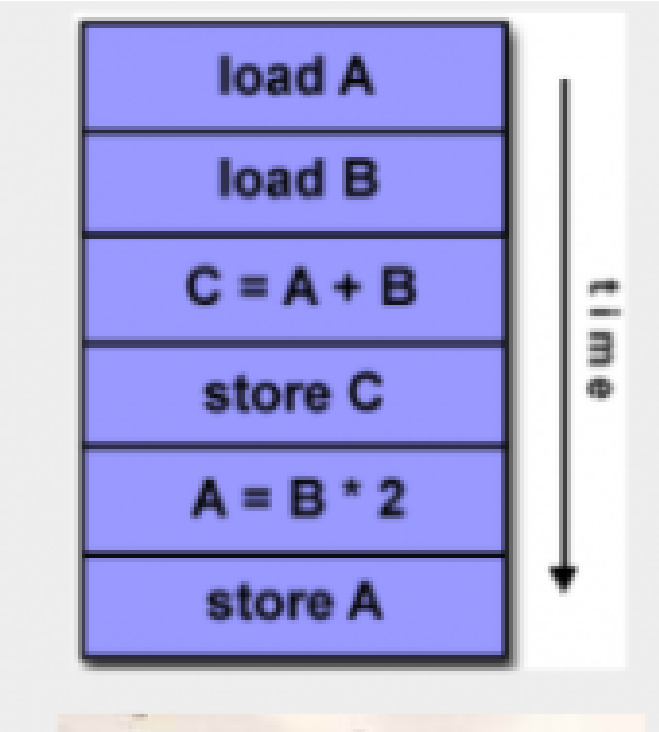
- State whether True or False for the following: a) SISD computers can be characterized as
  - Is > 1 and Ds > 1
- b) SIMD computers can be characterized as

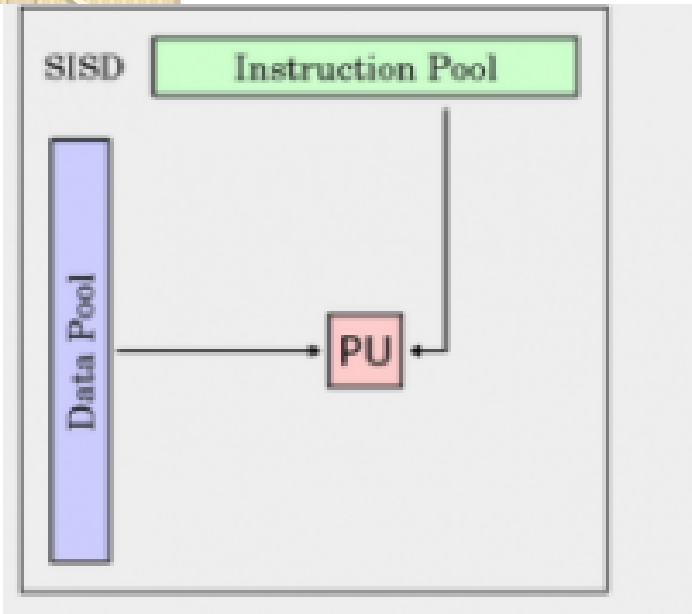
  Is > 1 and Ds = 1

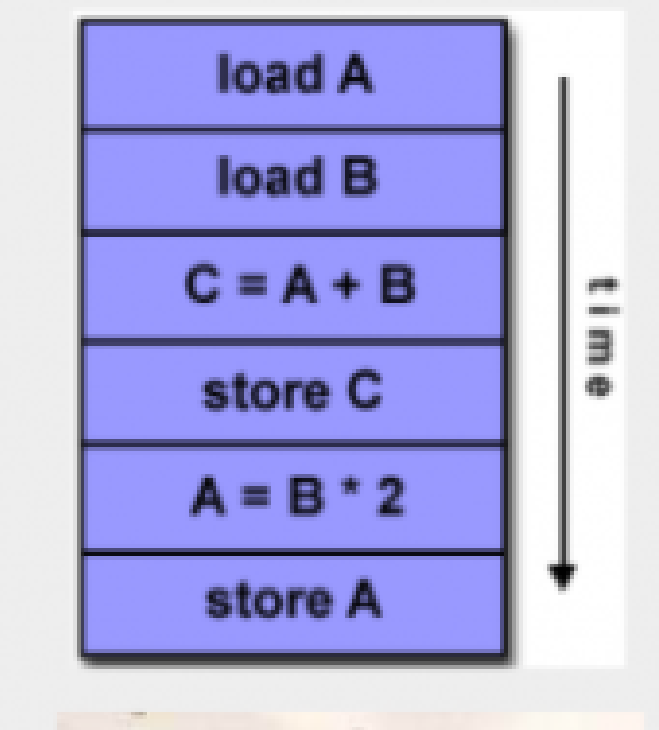
  c) MISD computers can be characterized as

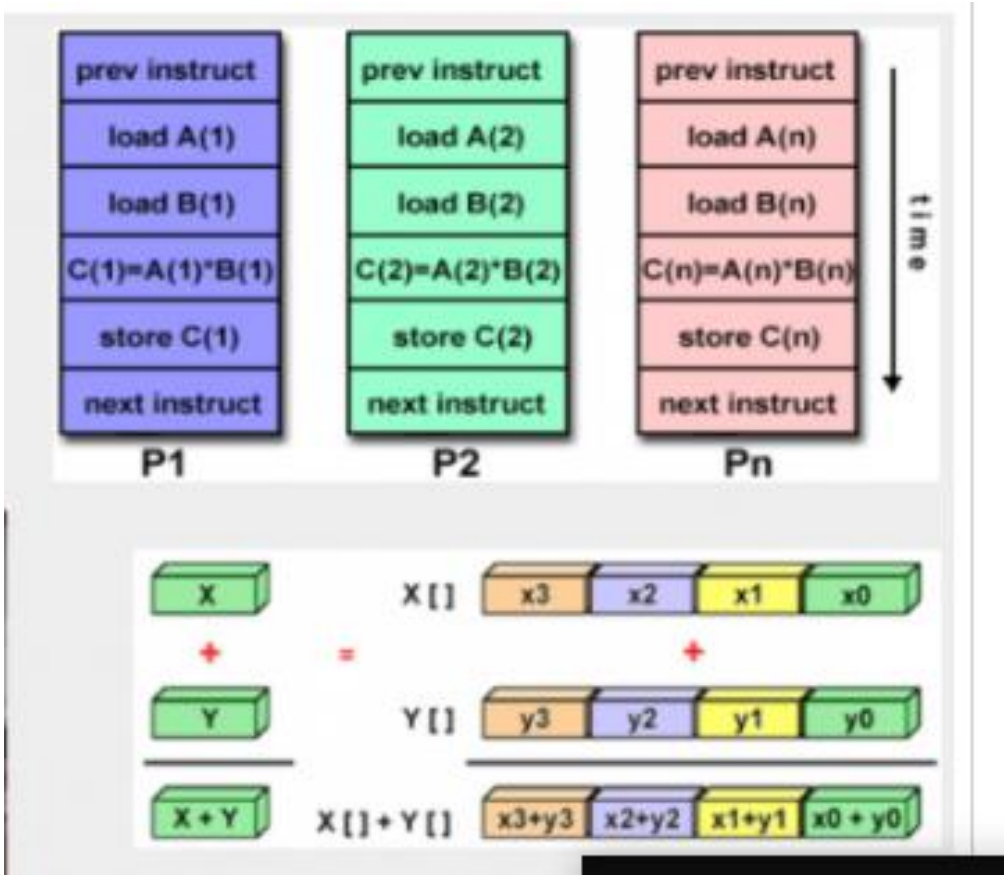
  Is = 1 and Ds = 1

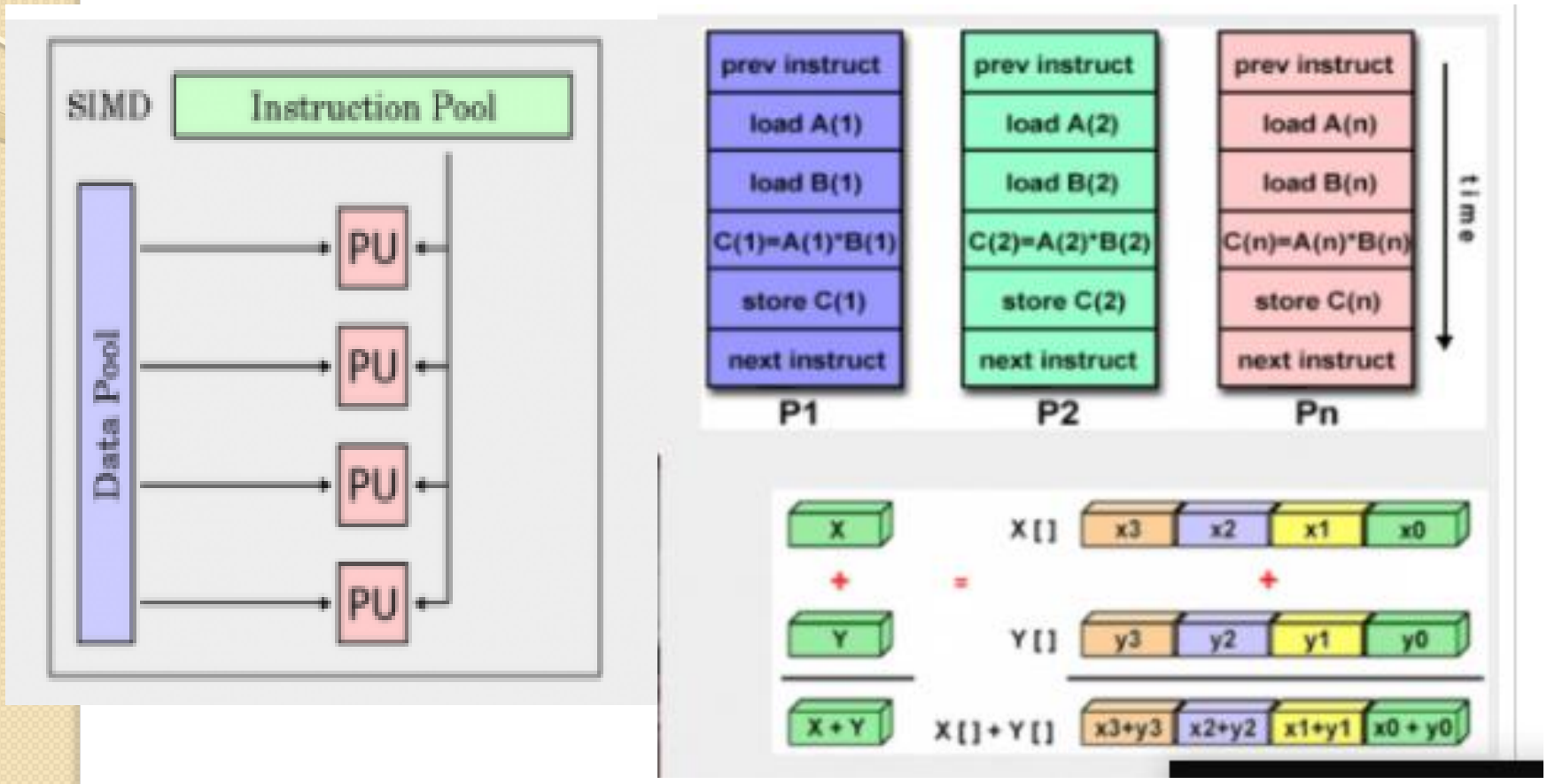
  d) MIMD computers can be characterized as Is > 1 and Ds > 1

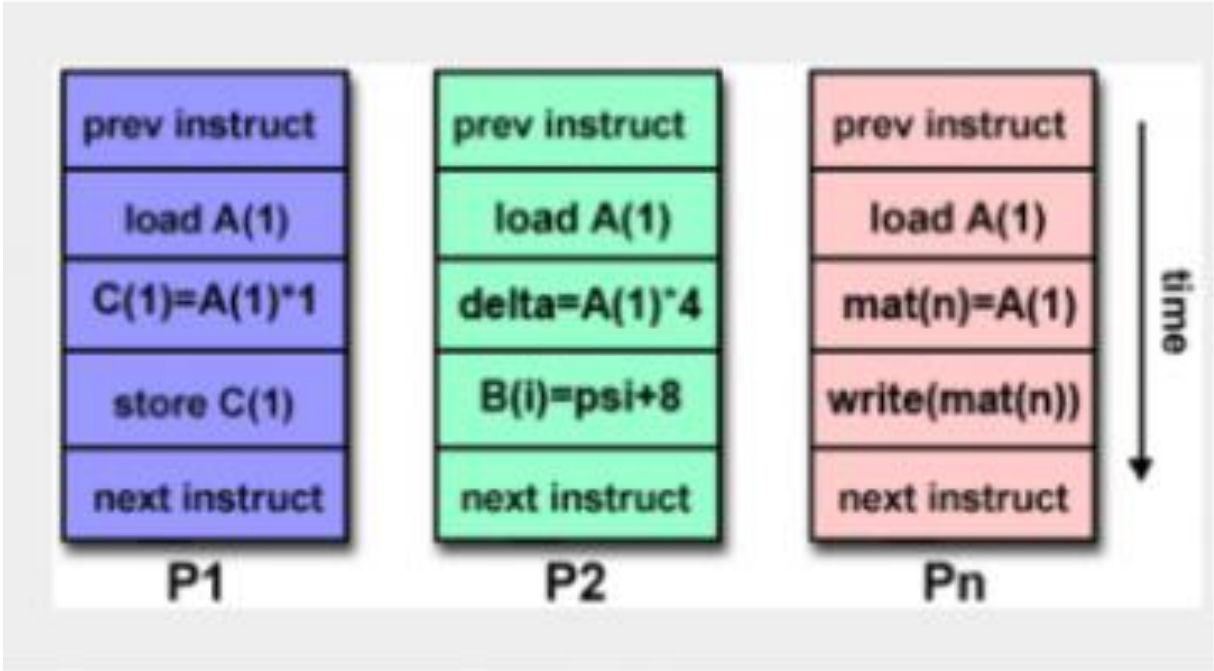

load A
load B
C = A + B
store C
A = B * 2
store A
time

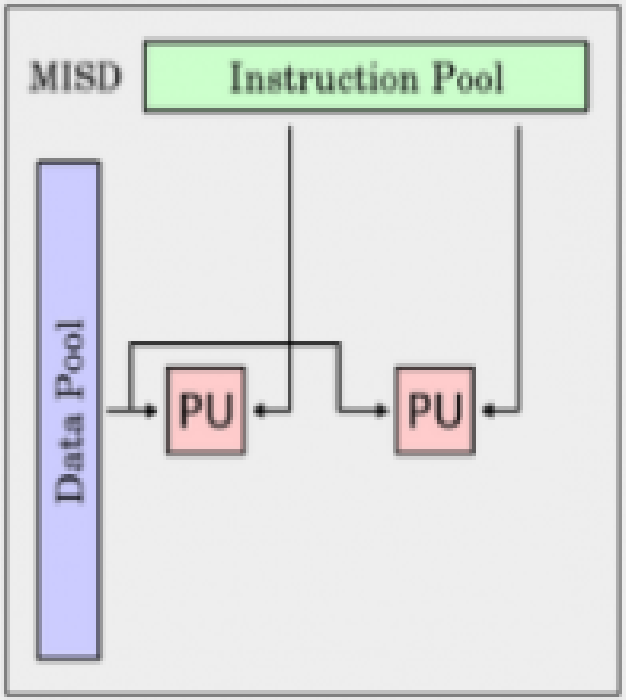

SISD
Instruction Pool
Data Pool
PU

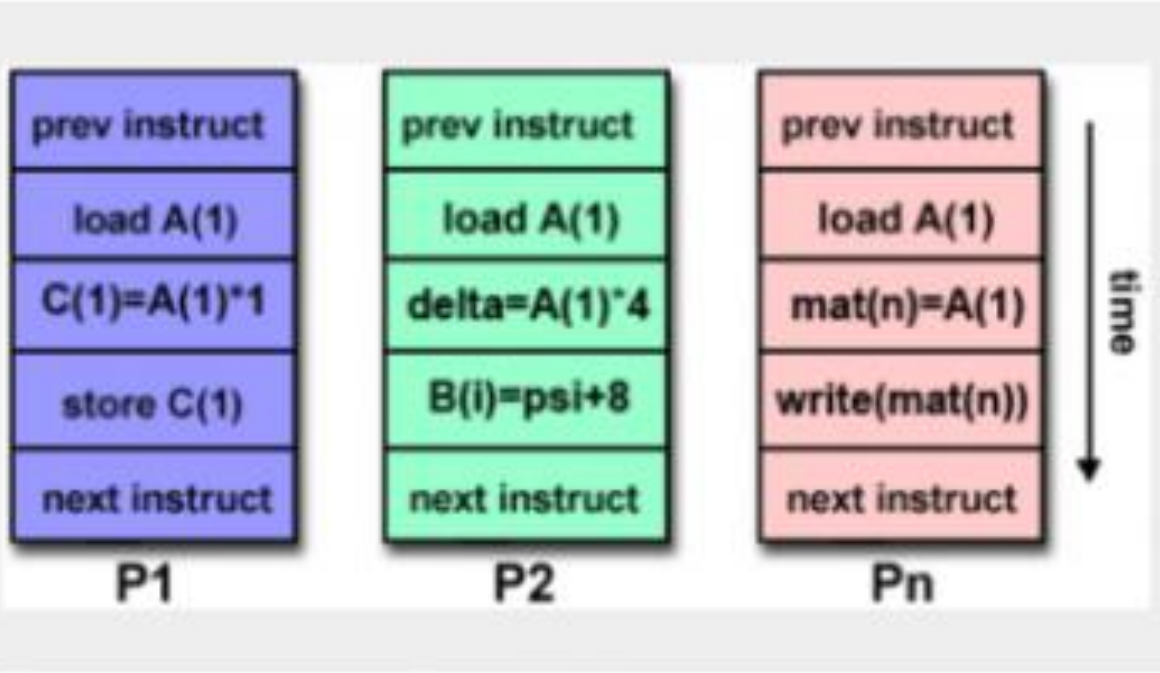

load A
load B
C = A + B
store C
A = B * 2
store A
time

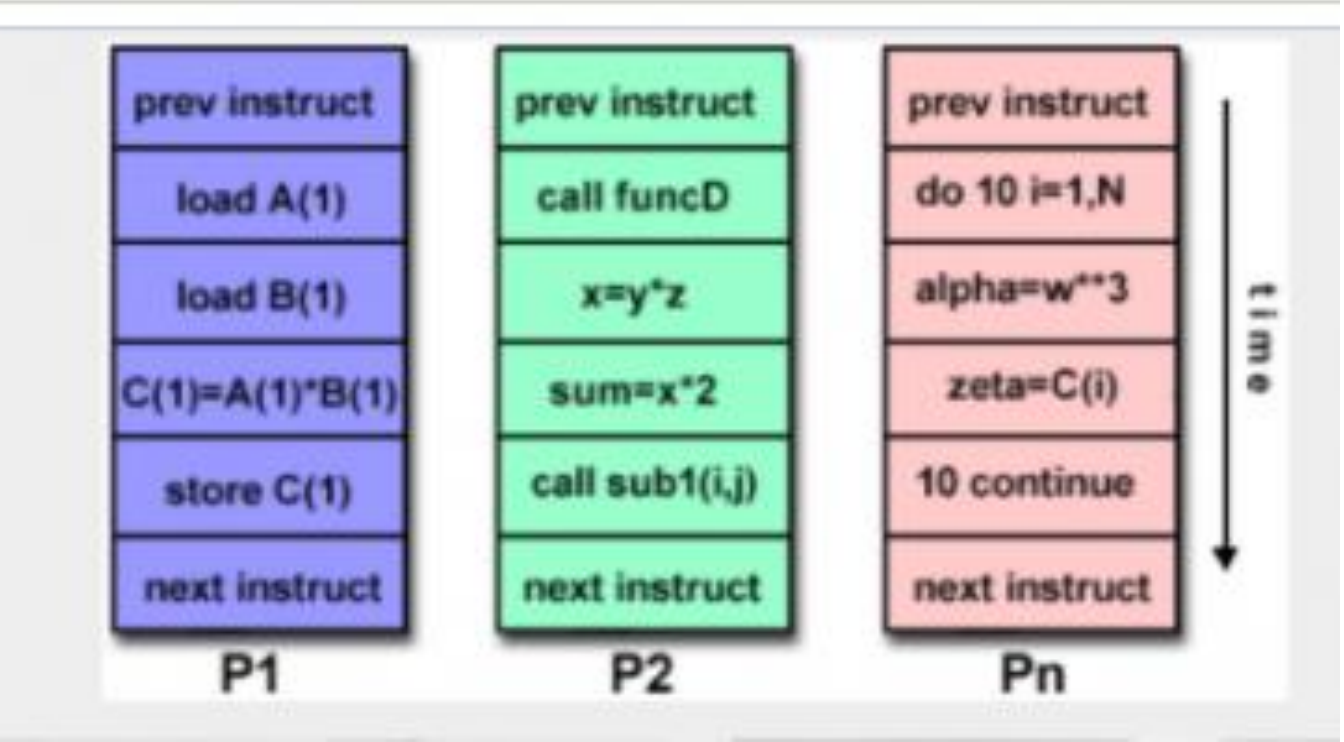

prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
load A(2)
load B(2)
C(2)=A(2)*B(2)
store C(2)
next instruct
P2
prev instruct
load A(n)
load B(n)
C(n)=A(n)*B(n)
store C(n)
next instruct
Pn
time
X
X[]
x3
x2
x1
x0
+
=
+
Y
Y[]
y3
y2
y1
y0
X + Y
X[] + Y[]
x3+y3
x2+y2
x1+y1
x0 + y0

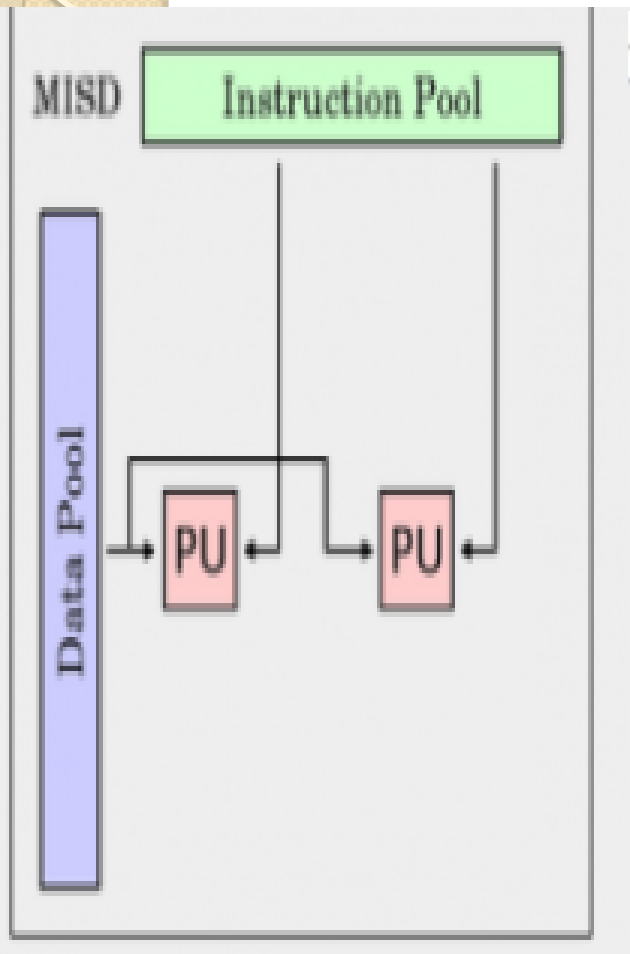

SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
load A(2)
load B(2)
C(2)=A(2)*B(2)
store C(2)
next instruct
P2
prev instruct
load A(n)
load B(n)
C(n)=A(n)*B(n)
store C(n)
next instruct
Pn
time
X
Y
X + Y
X [ ]
Y [ ]
X [ ] + Y [ ]
x3
x2
x1
x0
y3
y2
y1
y0
x3+y3
x2+y2
x1+y1
x0 + y0

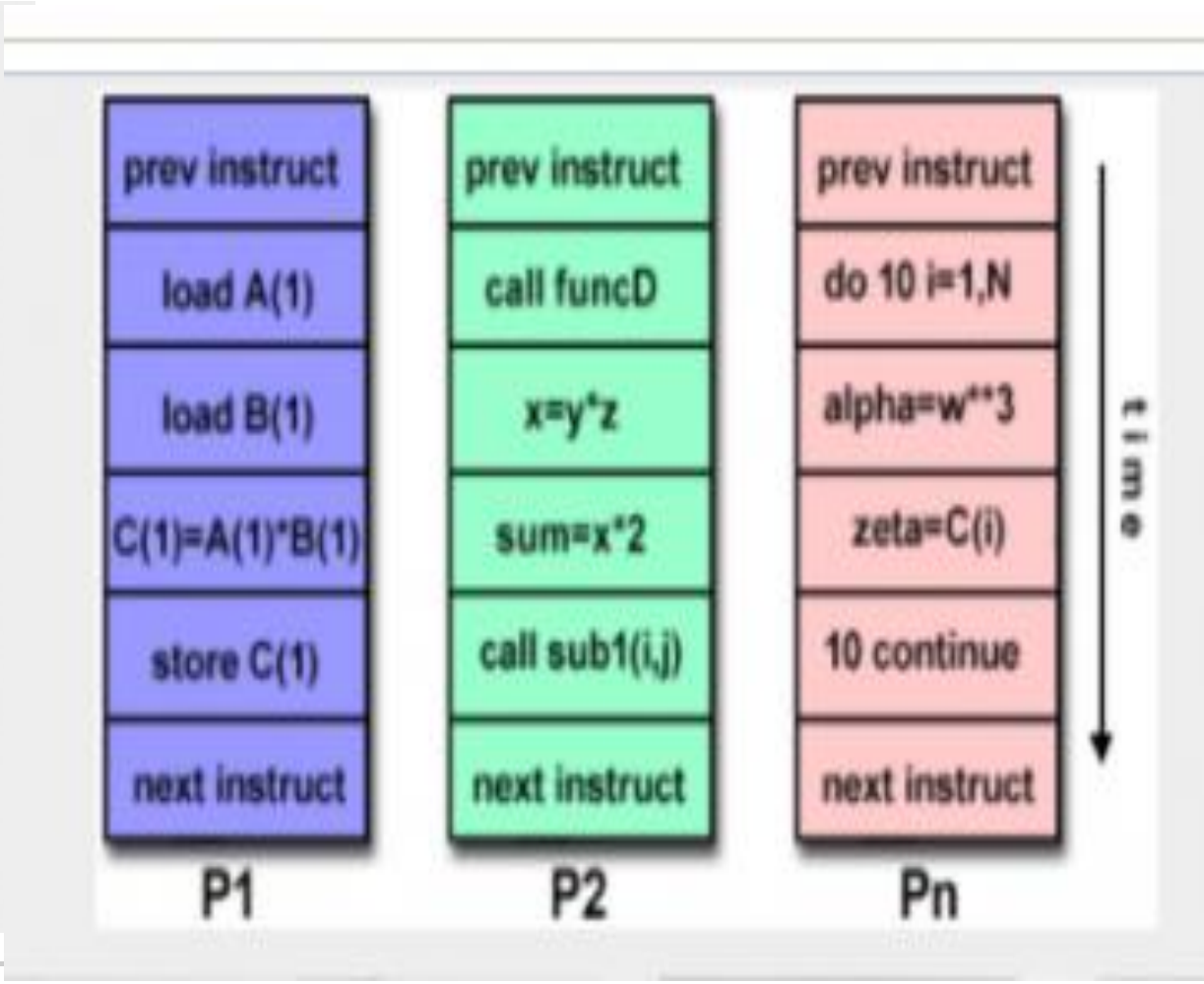

prev instruct
load A(1)
C(1)=A(1)*1
store C(1)
next instruct
P1
prev instruct
load A(1)
delta=A(1)*4
B(i)=psi+8
next instruct
P2
prev instruct
load A(1)
mat(n)=A(1)
write(mat(n))
next instruct
Pn
time


MISD
Instruction Pool
Data Pool
PU
PU


prev instruct
load A(1)
C(1)=A(1)*1
store C(1)
next instruct
P1
prev instruct
load A(1)
delta=A(1)*4
B(i)=psi+8
next instruct
P2
prev instruct
load A(1)
mat(n)=A(1)
write(mat(n))
next instruct
Pn
time


prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
call funcD
x=y*z
sum=x*2
call sub1(i,j)
next instruct
P2
prev instruct
do 10 i=1,N
alpha=w**3
zeta=C(i)
10 continue
next instruct
Pn
time


MISD
Instruction Pool
Data Pool
PU
PU


prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
call funcD
x=y*z
sum=x*2
call sub1(i,j)
next instruct
P2
prev instruct
do 10 i=1,N
alpha=w**3
zeta=C(i)
10 continue
next instruct
Pn
time

- MI MD
- CALCULATION OF REPORT CARD OF STUDENT IN UNVIVERSITY : -- SUM, AVERAGE, CGPA,
- STUDENTS OF CAO 62 STUDENTS → 1 TO 62

- MISD
- A=5, b=10
- C=a+b
- D=a*b
- E=a/b

- IMAGE → CONVOLUTION
- X*Y -> ?

- IMAGE (COLLECTION OF PIXEL) – (X,Y)
- MULTIPLE DATA
- CONVOULTION → SINGLE → SI MD

- Metrics for Performance Measurement

Processor Performance

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) **(CPI)** **(cycle time)**

- In the 1980's (decade of pipelining):
  - CPI: 5.0 => 1.15
- In the 1990's (decade of superscalar):
  - CPI: 1.15 => 0.5 (best case)
- In the 2000's (decade of multicore):
  - Marginal CPI improvement

Millions of Instructions per Second (MIPS),

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

Example

- Consider the execution of a program which results in the execution of 2 million instructions on a 400-MHz processor. The program consists of four major types of instructions. The instruction mix and the CPI for each instruction type are given below based on the result of a program trace experiment:

Instruction Type	CPI	Instruction Mix
Arithmetic and logic	1	60%
Load/store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	8	10%

- The average CPI when the program is executed on a uniprocessor with the
- above trace results is CPI???

$$CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (8 \times 0.1) = 2.24.$$

MIPS rate is $(400 \times 10^6)/(2.24 \times 10^6) \approx 178.$

Amdahl's law

Amdahl's law states that if ***P* is the proportion of a program that can be made parallel** (i.e., benefit from parallelization), and **(1 – *P*)** is the proportion that **cannot be parallelized** (remains serial), then the maximum speedup that can be achieved by using *N* processors is

$$S(N) = \frac{1}{(1 - P) + \frac{P}{N}}$$

$$\text{Speedup } S(p) = \frac{T_s}{f * T_s + (1 - f) * T_s/p} = \frac{\text{Sequential execution time}}{\text{Exe. time (seq.+ parallel)}}$$

with T_s : Execution time for sequential processing of the whole task

f : Fraction of the execution time for program segments which cannot run in parallel $(f = 0..1)$

p : Number of parallel processing elements (processors)

for $p \rightarrow \infty$: $S(p) = 1 / f$, or for $f \rightarrow 0$: $S(p) = p$

Examples

- If 90% of a calculation can be parallelized (i.e. 10% is sequential) then the maximum speed-up – 5 processor

- which can be achieved on 5 processors is 1/(0.1+(1-0.1)/5) or roughly 3.6 (i.e. the program can theoretically run 3.6 times faster on five processors than on one)

Examples

- If 90% of a calculation can be parallelized then the maximum speed-up on 10 processors is

- 1/(0.1+(1-0.1)/10)
- or 5.3 (i.e. investing twice as much hardware speeds the calculation up by about 50%).

Examples

- If 90% of a calculation can be parallelized then the maximum speed-up on 1000 processors is 1/(0.1+(1-0.1)/1000) or 9.9 (i.e. throwing an absurd amount of hardware at the calculation results in a maximum theoretical (i.e. actual results will be worse) speed-up of 9.9 vs a single processor).

- If 90% of a calculation can be parallelized then the maximum speed-up on 20 processors is 1/(0.1+(1-0.1)/20) or 6.9 (i.e. doubling the hardware again speeds up the calculation by only 30%).

Ex

Suppose that we are considering an enhancement to the processor of a server system used for Web serving. The new CPU is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original CPU is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

Refernce : Patterson 1.9 Quantitative Principles of Computer Design pg: **47**

$$Fraction_{enhanced} = 0.4$$

$$Speedup_{enhanced} = 10$$

$$Speedup_{overall} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

Example

A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics. Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQR hardware and speed up this operation by a factor of 10. The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for half of the execution time for the application. The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root. Compare these two design alternatives.

$$\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1 - 0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{Speedup}_{\text{FP}} = \frac{1}{(1 - 0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

Multithreading on A Chip

- Multithreading – increase the **utilization of resources** on a chip by allowing multiple processes (threads) to share the **functional units of a single processor**

Multithreading

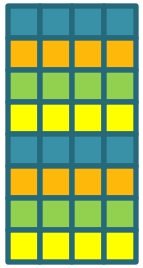
- Processor must duplicate the state hardware for each thread – a separate **register file, PC, instruction buffer, and store buffer** for each thread
- The **caches, buffers** can be shared
- The **memory can be shared** through virtual memory mechanisms
- Hardware must support **efficient thread context switching**

Types of Multithreading

- **Fine-grained**
  - Cycle by cycle
- **Coarse-grained**
  - Switch on event (e.g., cache miss)
  - Switch on quantum/timeout
- **Simultaneous**
  - Instructions from multiple threads executed concurrently in the same cycle

Fine-grain multithreading

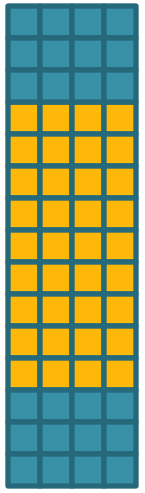
- Switch contexts at fixed fine-grain interval (e.g. every cycle)
- Need enough thread contexts to cover stalls
- Example: Tera MTA,
- Benefits:
- Conceptually simple, high throughput, deterministic behavior



- Drawback:
  - Very poor single-thread performance

Coarse-grain multithreading

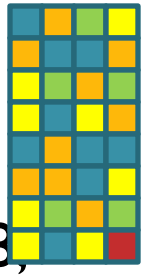
- Switch contexts on long-latency events (e.g. cache misses)
- Need a handful of contexts (2-4) for most benefit



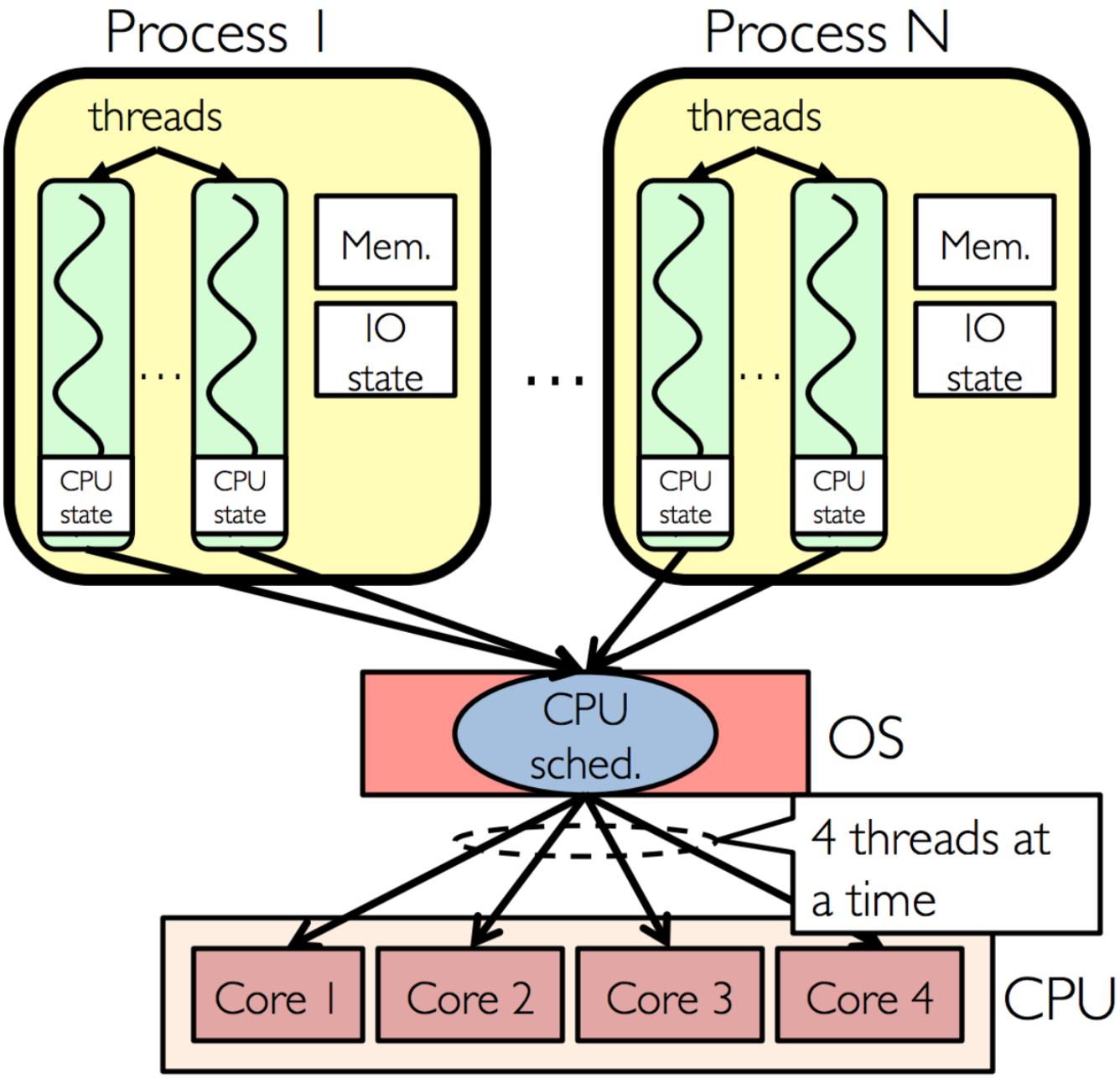
- Example: IBM RS64-IV (Northstar),
- Benefits:
  - Simple, improved throughput (~30%), low cost
  - Thread priorities mostly avoid single-thread slowdown
- Drawback:
  - Nondeterministic, conflicts in shared caches

Simultaneous multithreading

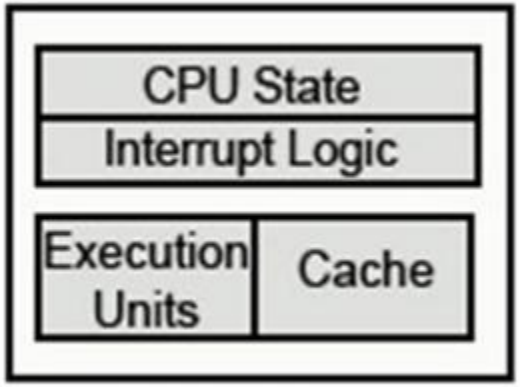
- Multiple concurrent active threads (no notion of thread switching)
- Need a handful of contexts for most benefit (2-8)



- Examples: Intel P4, Intel Nehalem, IBM Power 5/6/7/8, Alpha EV8/21464, AMD Zen
- Benefits:
  - Natural fit for superscalar
  - Improved throughput
  - Low incremental cost
- Drawbacks:
  - Additional complexity over superscalar
  - Cache conflicts


Process 1
Process N
threads
threads
Mem.
IO state
Mem.
IO state
CPU state
CPU state
CPU state
CPU state
CPU sched.
OS
4 threads at a time
Core 1
Core 2
Core 3
Core 4
CPU

Single Core



a) Single Core

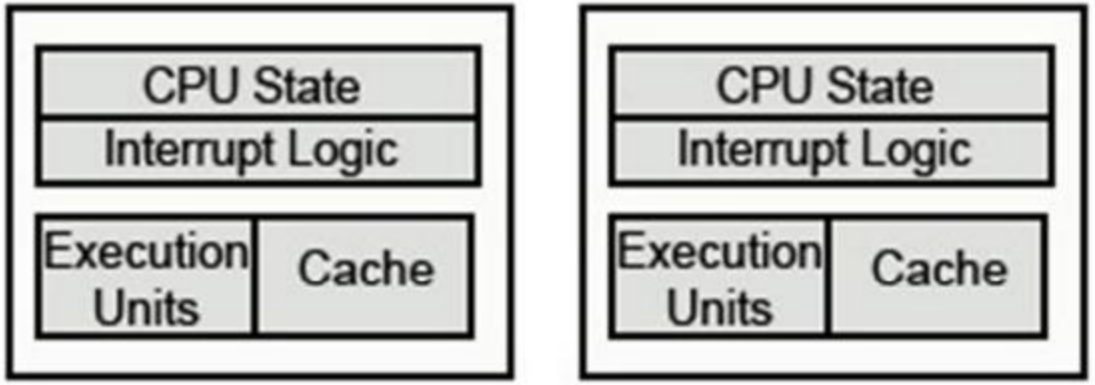
Parameter	Single-Core	Multi-Core
No of cores	One primary core	Two or more separate core
Processing	Sequential	Parallel
SMT	Not Possible	Possible
Power	Low	High
Speed	Slow	Fast
Efficiency	Low	High
Operation	One task at a time	Multitasking

Simultaneous **multithreading** (**SMT**)

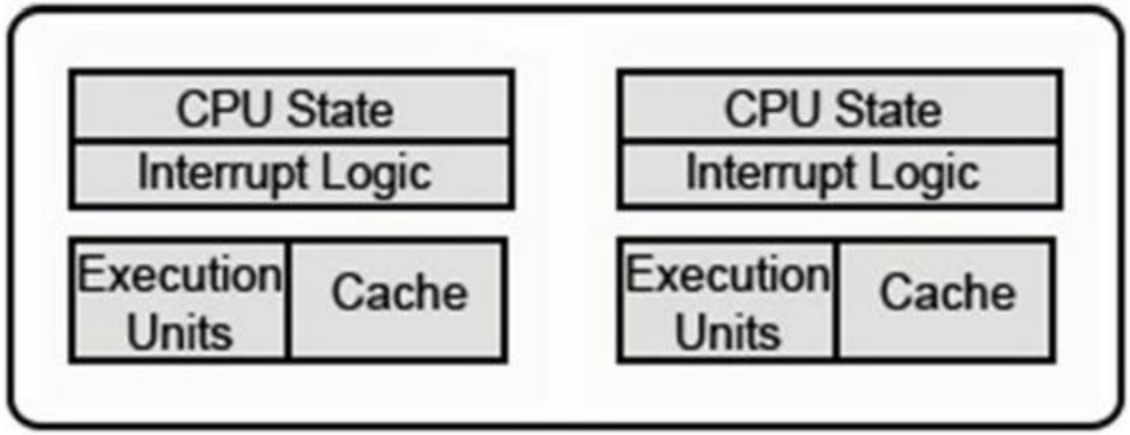
Difference between MultiCore and MultiProcessor System

MultiCore	MultiProcessor
A single CPU or processor with two or more independent processing units called cores that are capable of reading and executing program instructions.	A system with two or more CPU's that allows simultaneous processing of programs.
It executes single program faster.	It executes multiple programs Faster.
Not as reliable as multiprocessor.	More reliable since failure in one CPU will not affect other.

MultiCore	MultiProcessor
It has less traffic.	It has more traffic.
It does not need to be configured.	It needs little complex configuration.
It's very cheaper (single CPU that does not require multiple CPU support system).	It is Expensive (Multiple separate CPU's that require system that supports multiple processors) as compared to MultiCore.



b) Multiprocessor

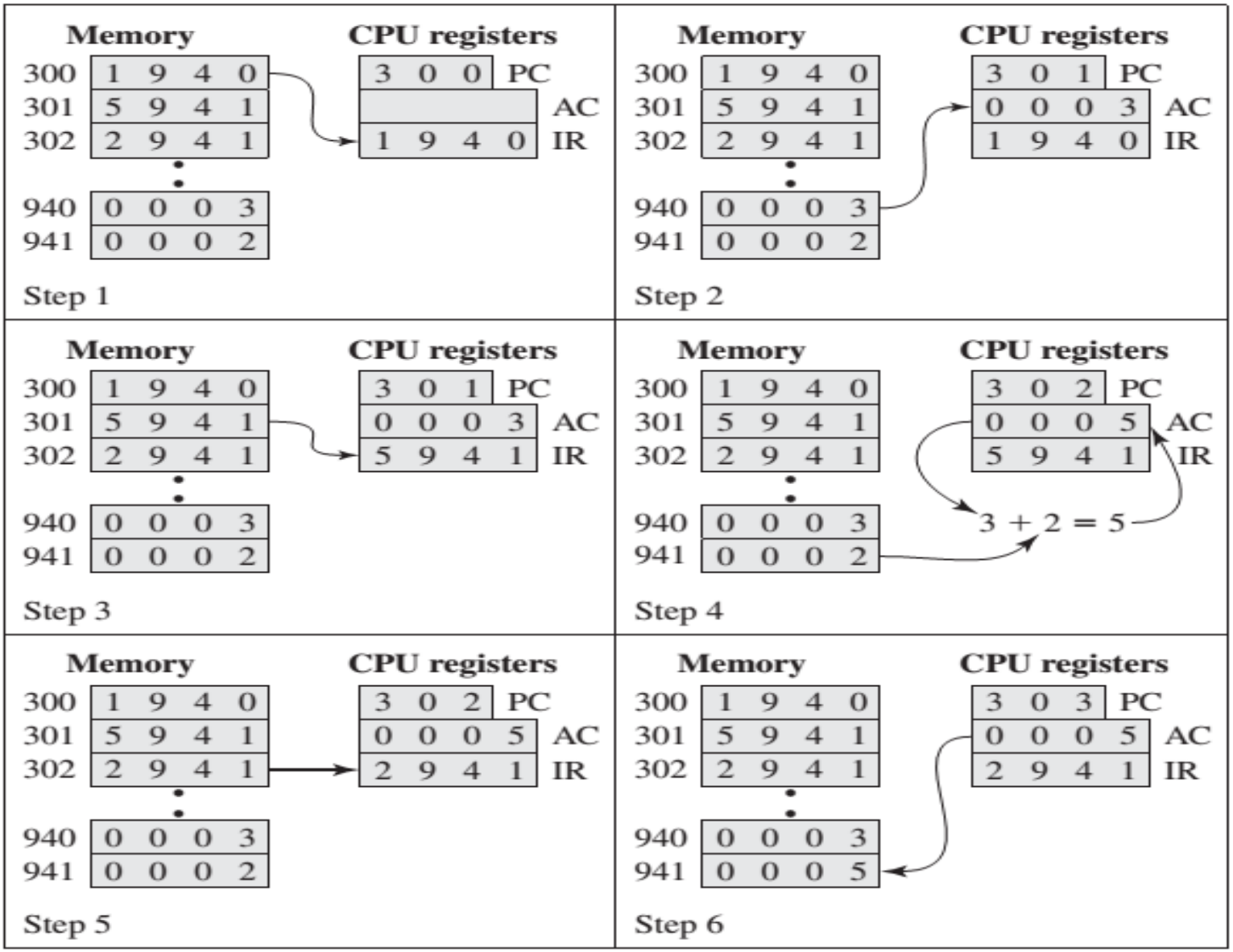


c) Multi-core

Practice Problems

Processor-Typical operational steps

- Program reside in the memory usually get there through the input unit
- Execution of the program starts when PC is set to point to the first instruction of the program
- The content of the PC are transferred to MAR
- A read control signal is sent to the memory
- The addressed word is read out of the memory and loaded into the MDR
- Next, the contents of the MDR are transferred to the IR
- At this point the instruction is ready to be decoded and executed


Memory
CPU registers
300 1 9 4 0
301 5 9 4 1
302 2 9 4 1
940 0 0 0 3
941 0 0 0 2
3 0 0 PC
AC
1 9 4 0 IR
Step 1
3 0 1 PC
0 0 0 3 AC
1 9 4 0 IR
Step 2
3 0 1 PC
0 0 0 3 AC
5 9 4 1 IR
Step 3
3 0 2 PC
0 0 0 5 AC
5 9 4 1 IR
3 + 2 = 5
Step 4
3 0 2 PC
0 0 0 5 AC
2 9 4 1 IR
Step 5
3 0 3 PC
0 0 0 5 AC
2 9 4 1 IR
Step 6

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the instruction register IR and the PC is incremented. Note that this process involves the use of a memory address register (MAR) and a memory buffer register (MBR). For simplicity, these intermediate registers are ignored.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded. The remaining 12 bits (three hexadecimal digits) specify the address (940) from which data are to be loaded.

3. The next instruction (5941) is fetched from location 301 and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
5. The next instruction (2941) is fetched from location 302 and the PC is incremented.
6. The contents of the AC are stored in location 941.

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the instruction register IR and the PC is incremented. Note that this process involves the use of a memory address register (MAR) and a memory buffer register (MBR). For simplicity, these intermediate registers are ignored.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded. The remaining 12 bits (three hexadecimal digits) specify the address (940) from which data are to be loaded.
3. The next instruction (5941) is fetched from location 301 and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
5. The next instruction (2941) is fetched from location 302 and the PC is incremented.
6. The contents of the AC are stored in location 941.