Unit-6

Pipelining:-

Parallel Processing:- Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing computational speed of a computer system. Instead of processing a single instruction at a time, parallel processing system is able to process multiple instructions at a time. The purpose of parallel processing is to speed up the computer processing capability and increase its throughput (i.e the amount of processing can be accomplished during a given interval of time). The amount of hardware increases with parallel processing so, the cost of system increases.

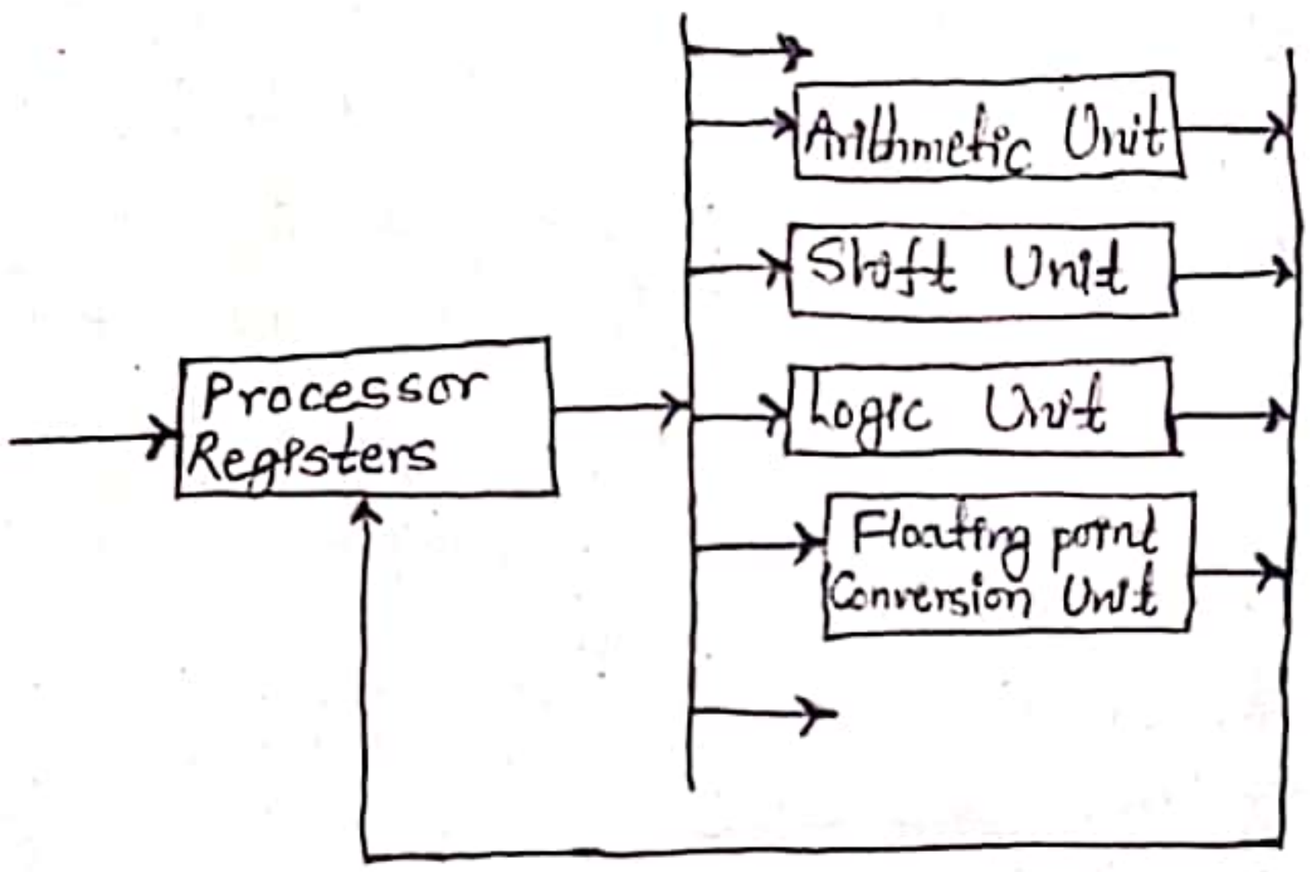


fig. Concept of parallel processing

In the above figure we can see that the data stored in the processor registers is being sent to seperate devices based on operation to be performed on data. If the data inside processor register is requesting for arithmetic operation, then the data will be sent to arithmetic unit. Similarly if it is requesting for logical operation, then the data will be sent to logic Unit. Now, in the same time, both arithmetic operations and logical operations are executing in parallel. This is called parallel processing.

1) **Instruction stream** → The sequence of instructions read from memory is called an instruction stream.

⇒ Data stream → The operations performed on the data in the processor is called as data stream.

⇒ The computers are classified into 4 types based on the Instruction stream and Data stream. They are called as the Flynn's Classification of computers.

⊛. Flynn's Classification of Computers:

Flynn's classification divides computer into four major groups as follows:

Single instruction stream, single data stream (SISD)
Single instruction stream, multiple data stream (SIMD)
Multiple instruction stream, single data stream (MISD).
Multiple instruction stream, multiple data stream (MIMD).

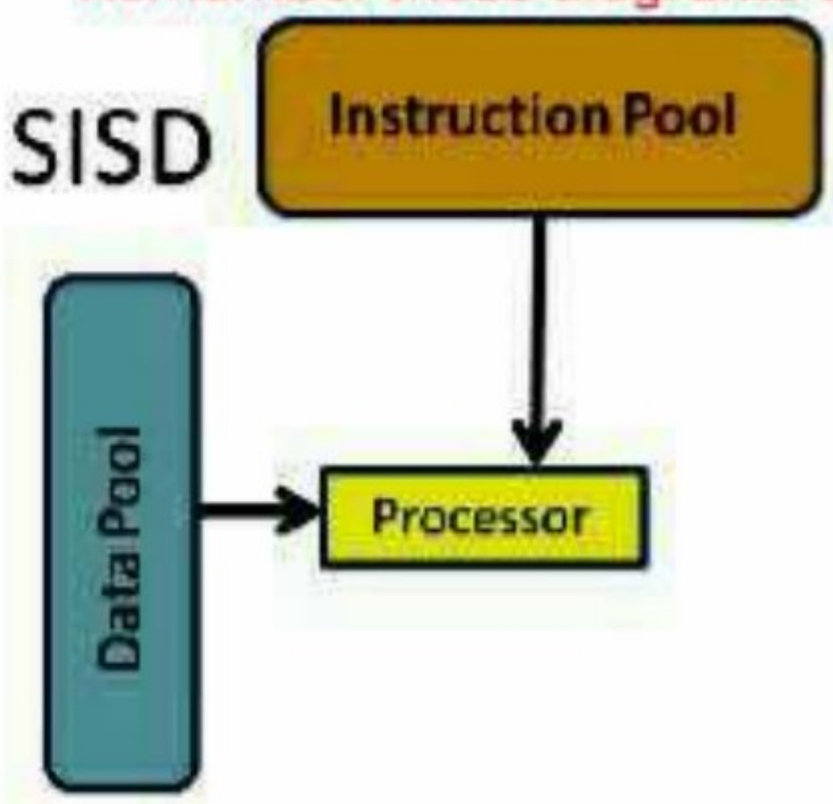
i) SISD → SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit. Instructions are executed sequentially and the system may or may not have internal processing capabilities. Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.

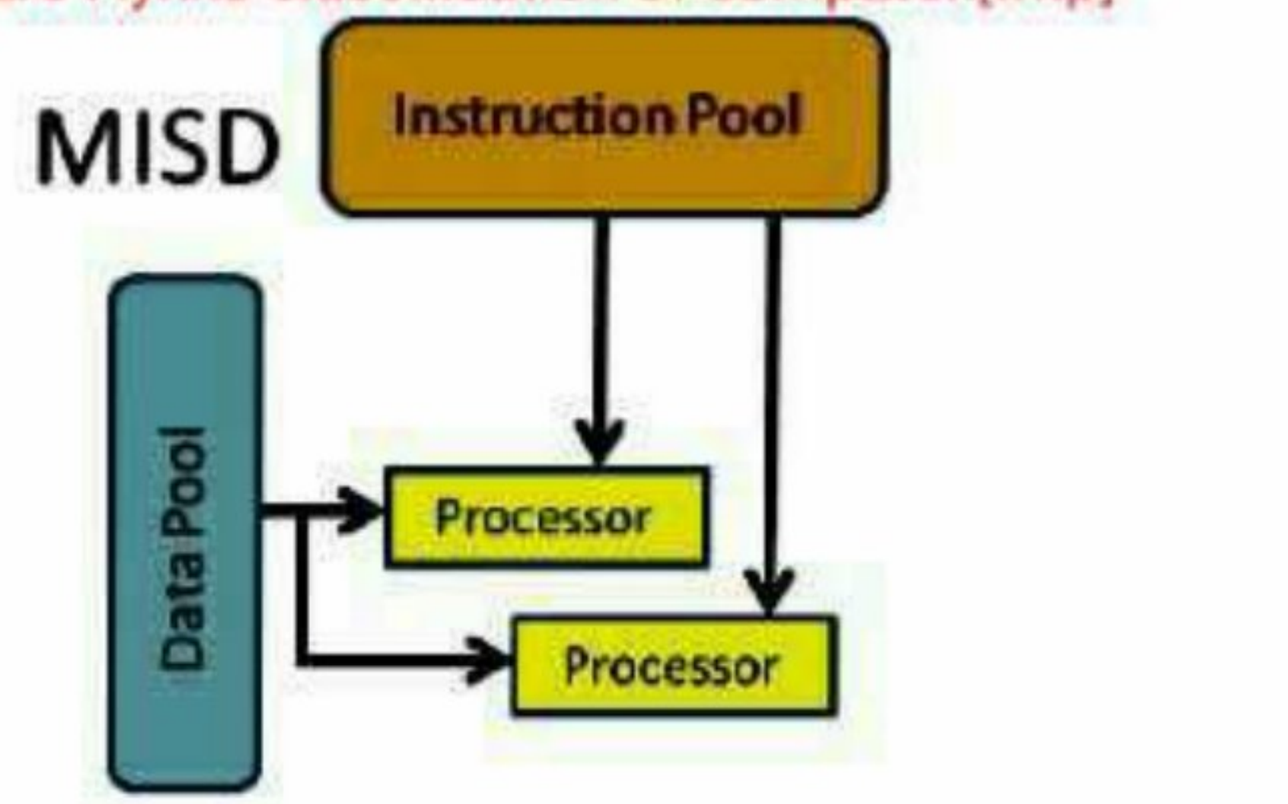
ii) SIMD → SIMD represents an organization that includes many processing units under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate on different items of data.

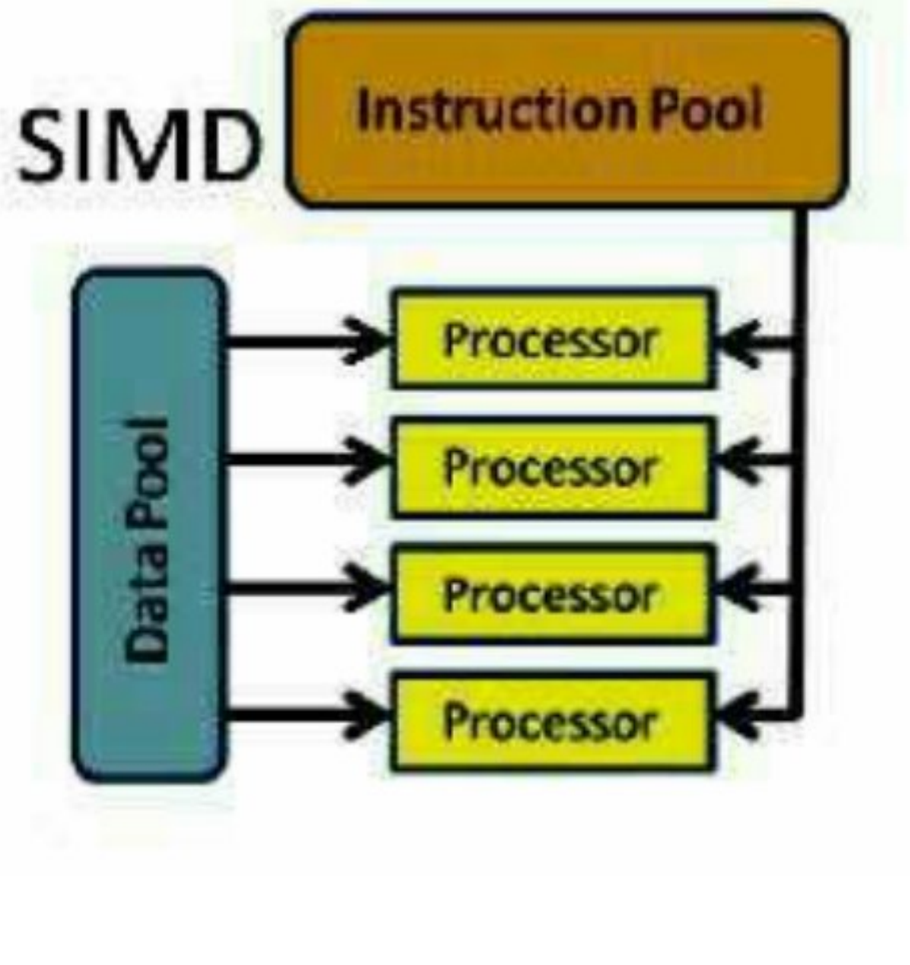
iii) MISD → MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

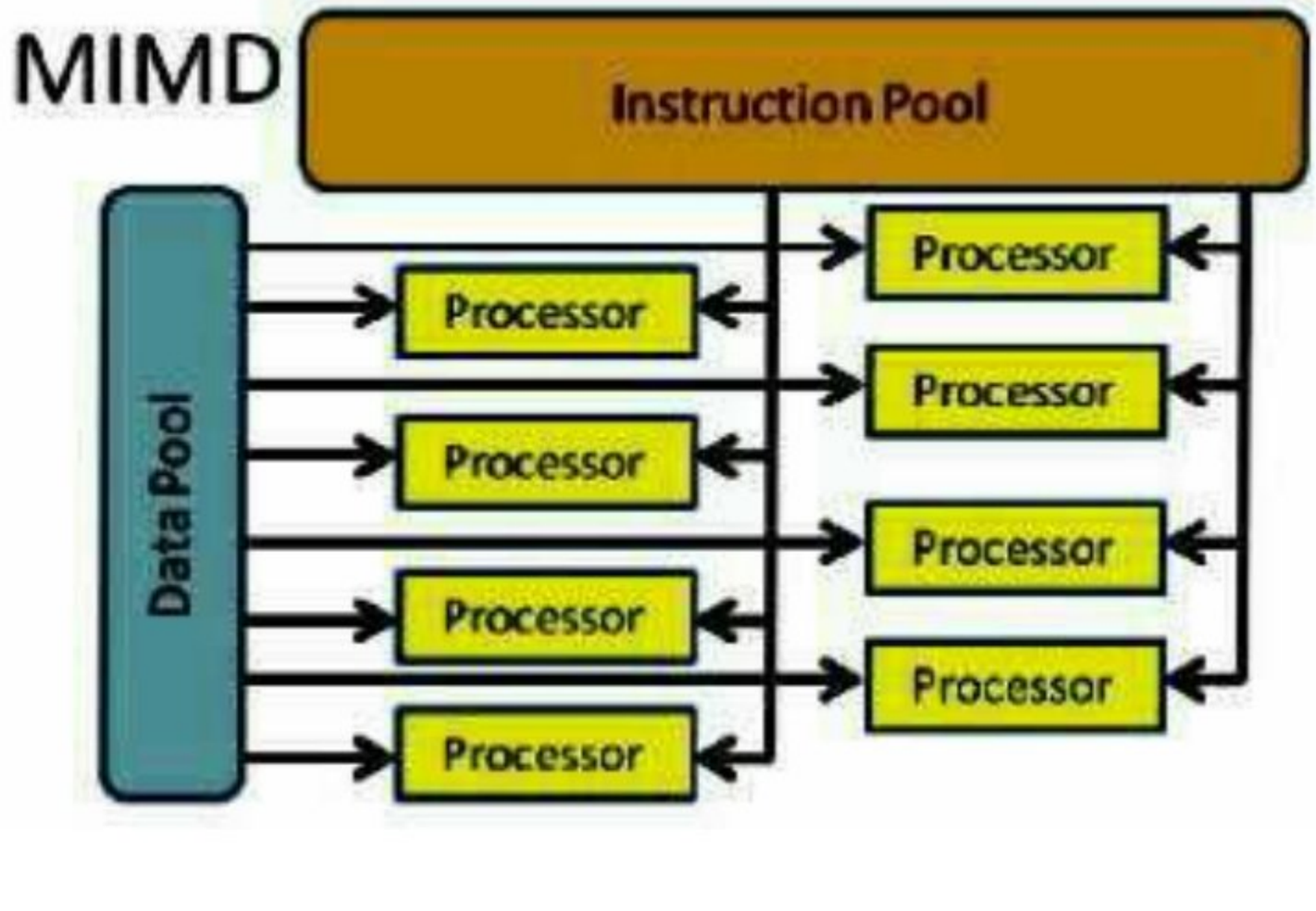
iv) MIMD → MIMD organization refers to a computer system capable of processing several programs at the same time. Most multiprocessor and multi-computer systems can be classified in this category.

Remember these diagrams also which are Flynns classification of computer.[Imp]









④. Pipelining: Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates simultaneously with all other segments. The overlapping of computation is made possible by associating a register with each segment in the pipeline. The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

Consider the operation: Result = (A+B)*C

→ First the A and B values are fetched which is "fetch Operation".
→ The result of the Fetch operations is given as input to the Addition operation, which is "Arithmetic operation."
→ The result of the Arithmetic operation is again is multiplied to data operand C which is fetched from memory which is another "Arithmetic operation".

In this process we are using up-to 5 pipelines which are:

→ Fetch operation (A), fetch operation (B)
~~→ fetch operation (B).~~
→ Addition of (A & B)
→ fetch operation (C)
→ Multiplication of ((A+B), C)
→ Load ((A+B)*C); Result);

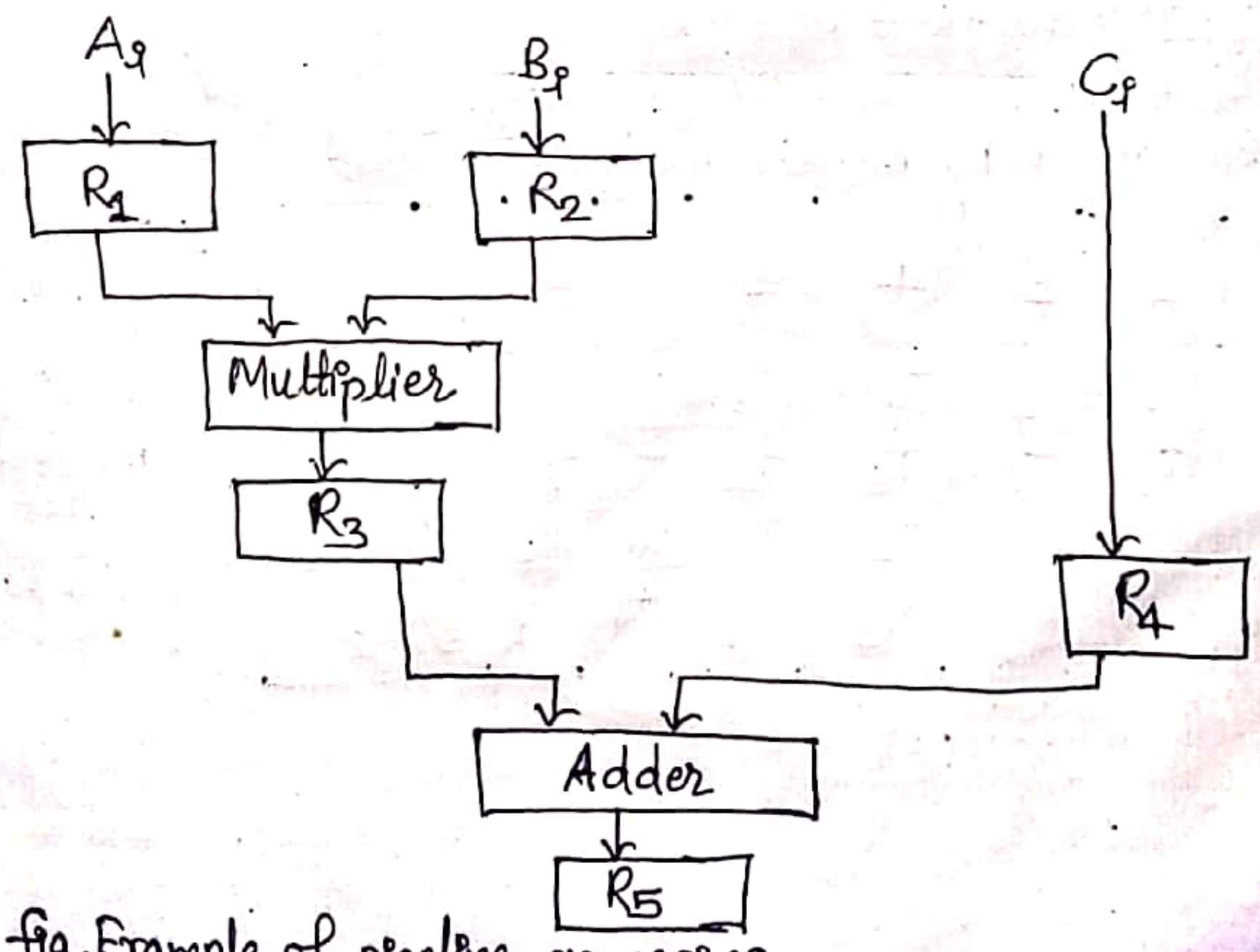


fig. Example of pipeline processing.

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R_1	R_2	R_3	R_4	R_5
1	A_1	B_1	—	—	—
2	A_2	B_2	A_1*B_1	C_1	—
3	A_3	B_3	A_2*B_2	C_2	$A_1*B_1+C_1$
4	A_4	B_4	A_3*B_3	C_3	$A_2*B_2+C_2$
5	A_5	B_5	A_4*B_4	C_4	$A_3*B_3+C_3$
6	A_6	B_6	A_5*B_5	C_5	$A_4*B_4+C_4$
7	A_7	B_7	A_6*B_6	C_6	$A_5*B_5+C_5$
8	—	—	A_7*B_7	C_7	$A_6*B_6+C_6$
9	—	—	—	—	$A_7*B_7+C_7$

Table: Content of Registers in pipeline example.

⇒ The diagram that shows the segment utilization as a function of time is called space-time diagram. The behaviour of a pipeline can be illustrated with space-time diagram.

Let we take four-segment pipeline which is as below:-

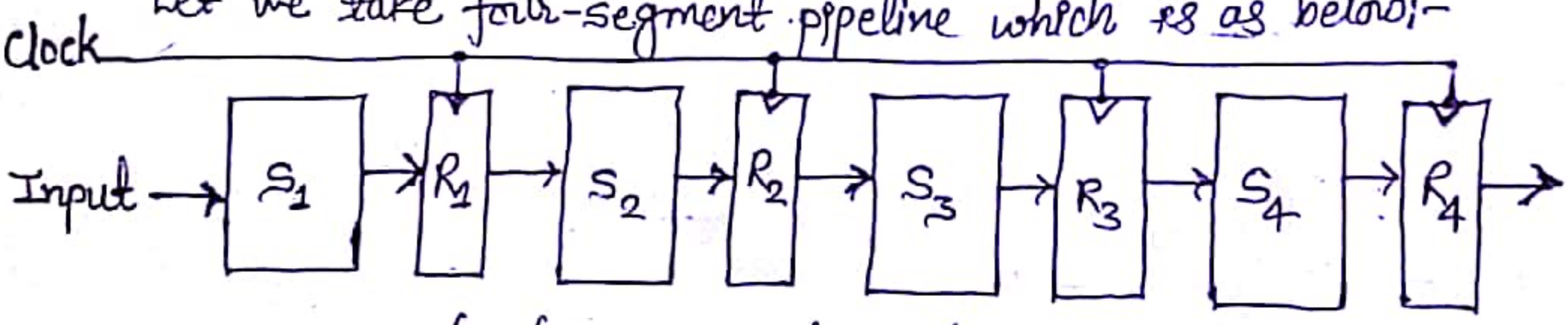


fig. four-segment pipeline

Now the space-time diagram for this four-segmented pipeline will be as below; (let 6-tasks are being executed).

Segment No. \ Clock cycles →	1	2	3	4	5	6	7	8	9
1	T_1	T_2	T_3	T_4	T_5	T_6			
2		T_1	T_2	T_3	T_4	T_5	T_6		
3			T_1	T_2	T_3	T_4	T_5	T_6	
4				T_1	T_2	T_3	T_4	T_5	T_6

fig. Space-diagram for pipeline

Let segment=k
tasks=n
Then, clock cycles = k+(n-1)

Here, k=4
n=6
clock cycles
= 4+(6-1)
=9

⊛. Speed up Equation:-

Consider, there are k-segment pipeline, with a clock cycle time t_p to execute n tasks. The first task T_1 requires a time equal to kt_p to complete its operation since thae are k segments. The remaining n-1 tasks emerge at the rate of one task per clock cycle and they will be completed after $(n-1)t_p$. Therefore to complete n-tasks using a k-segment pipeline requires $k+(n-1)$ clock cycles.

For example:- To complete 6 tasks using a 4 segment timeline requires $4+(6-1)=9$ clock cycles.

⇒ The speedup of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio:

$$S = \frac{nt_n}{(k+n-1)t_p}$$

As the number of tasks increases, n becomes much larger than k-1, and k+n-1 approaches the value of n. Under this condition the speedup becomes

$$S = \frac{t_n}{t_p}$$

If we assume that the time it takes to process a task is the same in the pipeline and nonpipeline circuits, we will have $t_n = k \cdot t_p$. Including this assumption, the speedup reduces to,

$$S = \frac{kt_p}{t_p} = k.$$

⊛. Instruction Level Pipelining: (Instruction Pipeline)

The instruction pipeline execution will be like queue execution (i.e FIFO technique). Therefore when an instruction is first coming, the instruction will be placed in queue and will be executed in the system. Finally result will be passing on to the next instruction in queue. The instruction cycle is as below:

→ Fetch the instruction from the memory.
→ Decode the instruction.
→ calculate the effective address.
→ Fetch the operands from the memory.
→ Execute the instruction.
→ Store the result in proper place.

⊛ Example: Four-Segment Instruction Pipeline:

Assume that the decoding of the instruction can be combined with the calculation of the effective address into one segment. Assume further that most of the instructions place the result into a processor register so that the instruction execution and storing of the result can be combined into one segment. This reduces the instruction pipeline into four segments.

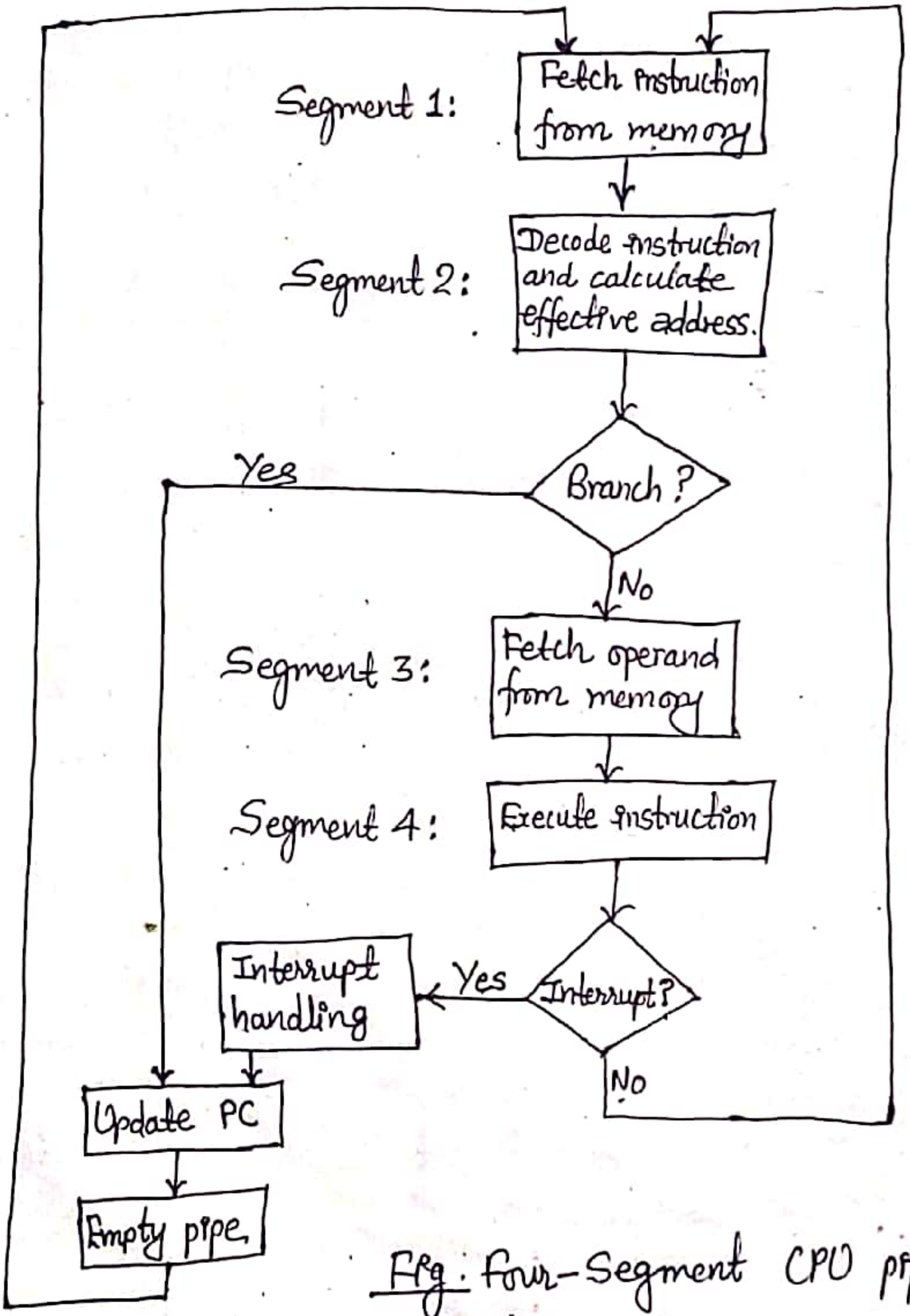


Fig: four-Segment CPU pipeline.

The different instruction cycles are:-

FI → It is a segment that fetches an instruction.
DA → It is a segment that decodes instruction and identifies effective address.
FO → It is a segment that fetches the operand.
EX → It is a segment that executes the instruction with the operand.

Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction: 1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
(Branch) 3			FI	DA	FO	EX							
4				FI	—	—	FI	DA	FO	EX			
5					—	—	—	FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

fig. Timing of instruction pipeline.

⊛. Pipelining Conflicts:-

There are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

i) **Resource conflicts** → There conflicts are caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using seperate instruction and data memories.

ii) **Data dependency conflicts** → These conflicts arise when an instruction depends on the results of previous instruction, but this result is not available yet. The data dependency conflicts can be solved by using following methods.

@ **Hardware interlocks** → An interlock is a circuit that detects instructions whose source operands are destination of instructions farther up in the pipeline. Detection of this situation causes the instruction whose source is not available to be delayed by enough clock cycles to resolve the conflict. This approach maintains the program sequence by using hardware to insert the required delay.

(b) **Operand Forewarding** → This is another technique that uses special hardware to detect a conflict and avoid the conflict path by using a special path to foreward the values between the ppeline segments.

(c) **Delayed load** → It delays the execution starting of the instruction such that all the data that is needed for the instruction can be successfully updated before execution.

9) Branch Conflicts → These difficulties arise from branch and other instructions that change the value of PC. The following are the solutions for solving the branch conflicts that are obtained in the pipelining concept.

ⓐ Pre-fetch target instruction: In this the branch instructions which are to be executed are pre-fetched to detect if any errors are present in the branch before execution.

ⓑ Branch target buffer: It is the associative memory implementation of the branch conditions.

ⓒ Loop buffer: It is a very high memory device, whenever a loop is executed in the computer. The complete loop will be transferred into the loop buffer memory and will be executed as in the cache memory.

ⓓ Branch prediction: In this before a branch is to be executed, the instructions along with the error checking conditions are checked. Therefore we will no be going into any unnecessary branch loops.

ⓔ Delayed Brach: The delayed branch concept is same as the delayed load process in which we are delaying the execution of a branch process, before all the data is fetched by the system for beginning the CPU.

⊛. Vector Processing:-

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems are characterized by the fact that they require a vast number of computations that will take a conventional computer a number of days or even weeks to complete. In many science and engineering applications, the problems can be formulated in terms of vectors and matrices that lend themselves to vector processing.

def^{n} → Vector processing is the process of using vectors to store a large number of variables for high-intensity data processing like weather forecasting, GIS data etc. It is processing of sequences of data in a uniform manner with a common occurance in manipulation of matrices or other arrays of data. The elements of those matrices are vectors.

⊛ Application areas of vector processing:

i) Long-range weather forecasting.
ii) Petroleum explorations.
iii) Seismic data analysis
iv) Medical diagnosis
v) Aerodynamics and space flight simulations
vi) Artifical intelligence and expert systems.
vii) Image Processing.

⊛ Vector Operations:

Many scientific problems require arithmetic operations on large arrays of numbers. These numbers are usually formulated as vectors and matrices of floating-point numbers. A vector is an ordered set of a one-dimensional array ~~floating~~ of data items. A vector V of length n is represented as a row vector by $V=[V_1\ V_2\ V_3 \ldots V_n]$ It maybe represented as a column vector if the data items are listed in a column. A conventional sequential computer is capable of processing operands one at a time.

Operations on vectors must be broken down into single computations with subscripted variables. The element V_i of vector V is written as V(I) and the index I refers to a memory address or register where the number is stored. Consider the following Fortran DO loop:

```
DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

This is a program for adding two vectors A and B of length 100 to produce a vector C.

A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop. It allows operations to be specified with a single vector instruction of the form:

$$C(1:100) = A(1:100) + B(1:100)$$

The vector instruction includes the initial address of the operands, the length of the vectors, and the operation to be performed, all in one composite instruction.

⊛. Matrix Multiplication:-

Matrix multiplication is one of the most computational intensive operations performed in computers with vector processors. The multiplication of two $n \times n$ matrix consists of n^2 inner products or n^3 multiply-add operations. A $n \times m$ matrix constitutes a set of n row vectors or a set of m column vectors. Consider, for example, the multiplication of two 3×3 matrices A and B.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

The product matrix C is a 3×3 matrix whose elements are related to the elements of A and B by inner product:

$$c_{ij} = \sum_{k=1}^{3} a_{ik} \times b_{kj}$$

For example, the number in the first row and first column of matrix C is calculated by letting $i=1, j=1$, to obtain

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length.

fig. Instruction format for vector processor