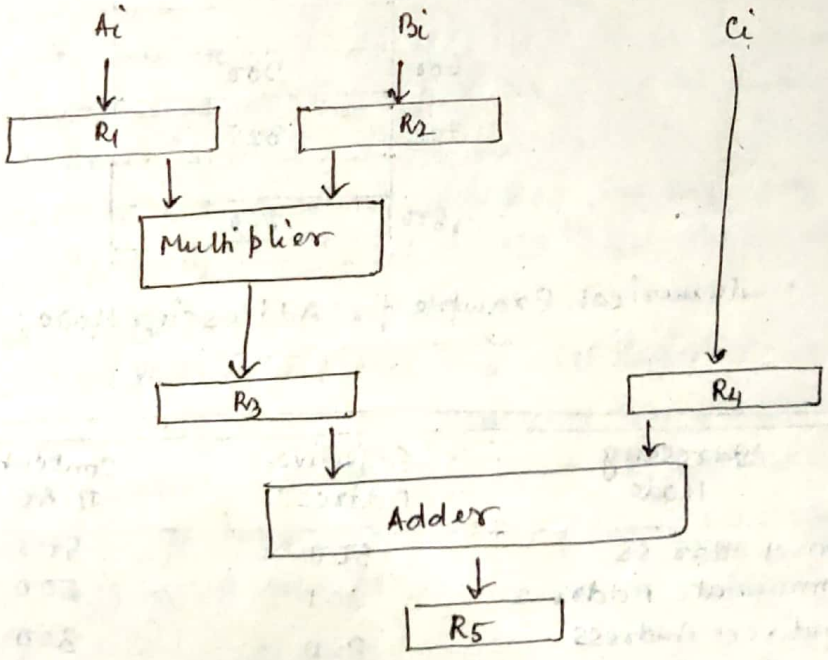
⇒ **Pipelining :-** Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other ~~registers~~ segments. A pipline can be visualized as a collection of processing segments through which binary information flows. The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.

□ **Example :-** The pipeline organization will be demonstrated by means of a simple example. Suppose we want to perform the combined multiply and add operations with a stream of numbers.

$$A_i * B_i + C_i \quad \text{for } i = 1, 2, 3 \cdots, 7.$$

Each suboperation is to be implemented in a segment within a pipeline. Each segment has one or two registers and a combinational circuit as shown in following figure.



R1 through R5 are registers that receive new data with every clock pulse. The multiplier and adder are combination circuits. The suboperations performed in each segment of the pipeline are :-

$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	Input A_i & B_i
$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$	Multiply and input C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

The five registers are loaded with new data every clock pulse. The effect of each clock pulse is shown in following table :-

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.	—	—	—	—	$A_7 * B_7 + C_7$

⇒ General Considerations :-

□ task :- Any operation that can be decomposed into a sequence of suboperations of about the same complexity can be implemented by a pipeline processor. The technique is efficient for those applications that need to repeat the same task many times with different sets of data. We define a task as the total operation performed going through all the segments in the pipeline.

□ space-time diagram :- The behaviour of a pipeline can be illustrated with a space-time diagram.

Initially task T_1 is handled by segment 1. After the first clock, segment 2 is busy with T_1 which segment 1 is busy with T_2. Continuing in this manner, the first task T_1 is completed after the fourth clock cycle. From then on, the pipe completes a task every clock. No matter how many segments are there in the system, once the pipe is full, it takes only one clock period to obtain an o/p.

Segment:	1	2	3	4	5	6	7	8	9	→ Clock cycle
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	

□ speedup :- Consider the case where a k-segment pipeline is used to execute n tasks with a clock-cycle time tp. So the first task T_1 requires a time equal to ktp to complete its operation since there are k-segments in the pipe. The remaining (n-1) tasks emerge from the pipe at the rate of one task per cycle, and they will be completed after a time equal to (n-1)tp. So to complete n tasks using a k-segment pipeline requires a time equal to (k+n-1)tp.

Next, consider a nonpipeline unit that performs the same operation and takes a time tn to complete each task. The total time required for n tasks is ntn. The speedup of a pipeline processing over an equivalent nonpipeline processing is defined by the ratio

$$S = \frac{ntn}{(k+n-1)tp}.$$

note- if $n >> k$ i.e no. of tasks increases n becomes much larger than k-1, i.e $n >> k-1$ so $k+n-1 \approx n$. under this condition

$$S = \frac{ntn}{ntp} = \frac{tn}{tp}$$

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

$$S = \frac{ktp}{tp} = k.$$

This shows the theoritical maximum speedup that a pipeline can provide is k, where k is the number of segments in pip

⇒ There are various reasons why the pipeline cannot operate at its maximum theoritical rate.

Different segments may take different times to complete their suboperation. The clock cycle must be choosen to equal the time delay of the segment with the maximum propagation time. This causes all other segments to waste time while waiting for the next clock. Moreover, it is not always correct to assume that a nonpipe circuit has the same time delay as that of an equival pipeline circuit. Many of the intermediate registers will not b needed in a single-unit circuit, which can usually be construct entirely as a combinational circuit. Nevertheless, the pipeline technique provides a faster operation over a purely serial sequ even though the maximum theoritical speed is never fully achived.

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

Figure 9.8 Timing of instruction pipeline.

Instruction 1 is being executed in step 4, in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FI.

Assume, instruction 3 is branch instruction. As soon as it is decoded in step 4, the transfer from FI to DA of the other instruction being halted until the branch instruction is executed in step 6. If the branch is taken, a new instruction is fetched in step 7. If the branch is not taken the instruction fetched in step 4 can be used. The pipeline then continue until a new branch instruction is encountered.

⇒ Pipeline Hazards :- In general there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

① Resource conflicts caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction & data memories.

② Data Dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.

③ Branch Difficulties arise from branch and other instructions that change the value of PC.

⇒ Data Dependency :- A difficulty that may caused a degradation of performance in an instruction pipeline is due to possible collision of data or address. A collision occurs when an instruction cannot proceed because previous instructions did not complete certain operations. A data dependency occurs when an instruction needs data that are not yet available. Therefore, the operand access to memory must be delayed until the required address is available.

Pipelined computers deal with this conflict in a variety of ways:-

// Hardware Interlocks :- The most straightforward method is Hardware Interlock. An interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in the pipeline. This approach maintains the program sequence by using hardware to insert required delays.

// operand forwarding :- Another technique called operand forwarding uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments. For example, instead of transferring an ALU result into a destination register, the hardware checks the destination operand, and if it is needed as a source in next instruction, it passes the result directly to the ALU input, bypassing the register file.

// Delayed load :- The compiler of some computers is designed to detect a data conflict and reorder the instructions as necessary to delay the loading of the conflicting data by inserting no-operation instructions. This method is referred as delayed load.

⇒ Handling of Branch Instructions :- One of the major problems in operating an instruction pipeline is the occurance of branch instructions. Pipelined computers employ various hardware techniques to minimize the performance degradation caused by instruction branching.

T.O. →→

// prefetch target instruction :- One way of handling a conditional branch is to prefetch the target instruction in addition to the instruction following the branch. Both are saved until the branch is executed. If the branch condition is successful, the pipeline continues from the branch target instruction. An extension of this procedure is to continue fetching instructions from both places until the branch decision is made.

// Branch Target Buffer (BTB) :- Another possibility is to use a branch target buffer or BTB. Each entry of BTB consists the address of previously executed branch instructions and the target instruction of the branch. When a pipeline decodes a branch instruction, it searches the associative memory BTB for the address of the instruction. If it is in the BTB the instruction is directly available, prefetch continues from new path.

// Loop Buffer :- A variation of BTB is loop buffer when a program loop is detected in the program, it is stored in loop buffer including all branches. The program loop can be executed directly without having to access memory.

// Branch Prediction :- Another procedure that some computers use is branch prediction. A pipeline with branch prediction uses some additional logic to guess the outcome of a conditional branch instruction before it is executed. The pipeline then begins prefetching the instruction stream from the predicted path.

// Delayed Branch :- In this procedure, the compiler detects the branch instructions and rearranges the machine languages code sequence by inserting useful instructions that keep the pipeline operating without interruptions. An example of delayed branch is the insertion of a no-operation instruction after a branch instruction. This causes the computer to fetch the target instruction during the execution of the no-operation instructions, allowing a continuous flow of the pipeline.

☐ Complex Instruction set computer (CISC) :-

An important aspect of computer architecture is the design of the instruction set for the processor. Early computers had small and simple instruction sets, forced mainly by the need to minimize the hardware used to implement them. Later, the trend into computer hardware complexity was influenced by various factors, such as upgrading existing models to provide more applications, these computers also employ a variety of data types and a large no. of addressing modes. A computer with a large number of instructions is classified as a Complex Instruction set Computer (CISC).

☐ Reduced Instruction Set Computer (RISC) :-

In early 1980s, a number of computer designers recommended that computers use fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often. This type of computer is classfied as a Reduced Instruction Set Computer (RISC).

☐ CISC Characteristics :-

① A large number of instructions - typically from 100 to 250 instructions.

② Some instructions that perform specialized tasks and are used infrequently.

③ A large variety of addressing modes - typically from 5 to 20 different modes.

④ Variable length instruction formats.

⑤ Instructions that manipulate operands in memory.

T.O.

☐ RISC characteristics :- The concept of RISC architecture involves an attempt to reduce execution time by ~~specifying~~ simplyfying the instruction set of the computer. Major characteristics are :-

① Relatively few instructions.
② Relatively few addressing modes.
③ Memory access limited to load and store instruction.
④ All operations done within the registers of CPU.
⑤ Fixed-length easily decoded instruction format.
⑥ Single-cycle instruction execution.
⑦ Hardwired rather than microprogrammed control.
⑧ Relatively large no. of registers in the processor unit.
⑨ Efficient instruction pipeline.

☐ Concept of Multiprocessor :- A multiprocessor system is an interconnection of two or more CPUs with memory and input-output equipment. The term processor in multiprocessor can mean either a central processing unit (CPU) or an ~~i~~ input-output processor (IOP). Multiprocessors are classified as multiple instruction stream, multiple data stream (MIMD) systems.

Multiprocessing improves the reliability of the system so that a failure or error in one part has a limited effect on the rest of the system. If a fault causes one processor to fail, a second processor can be assigned to perform the functions of the disabled processor.

The benifit derived from a multiprocessor org. is an improved system performance. The system derives its high performance from the fact that computations can proceed in parallel in one of two ways.

1. Multiple independent jobs can be made to operate in parallel
2. A single job can be partitioned into multiple parallel tasks

1. RISC pipeline: Reduced instruction set computer (RISC)
, An ability to use an efficient instruction pipeline
/ ability to execute instructions at the rate of
one per clock cycle.

say a three segment instruction pipeline has following phases:

I: Instruction
A: ALU operation
E: Execute Instruction.

Now, say the operation of following four instructions:

1. LOAD: $R1 \leftarrow M[address\ 1]$
2. LOAD: $R2 \leftarrow M[address\ 2]$
3. ADD: $R3 \leftarrow R1 + R2$
4. STORE: $M[address\ 3] \leftarrow R3$

Clock	1	2	3	4	5	6
1. LOAD R1	I	A	E			
2. LOAD R2		I	A	E		
3. ADD R_3			I	A	E	
4. STORE				I	A	E

clock	1	2	3	4	5	6	7
1. LOAD R1	I	A	E				
2. LOAD R2		I	A	E			
3. No-operation			I	A	E		
4. ADD R3				I	A	E	
5. STORE					I	A	E

So here actually after LOAD R2 it is wasting a clock cycle. That is why it is called as pipeline timing with delayed load.

- Delayed Branch: say the following five/six instructions:

Load from memory to R1
increment R2
Add R3 to R4
Subtract R5 from R6
Branch to address X.
Next instruction in X.

On this delayed branch system no-operation instructions are being fetched from the memory and they are executed through the pipeline when the branch instruction is executed. It is upto the compiler to find useful instructions to put after the branch instruction. Failing that the compiler can insert no-op instructions.

Clock	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Next instruction in X.								I	A	E

2. LOAD $R_1 \leftarrow M[312]$
Add $R_2 \leftarrow R_2 + M[313]$
Increment $R_3 \leftarrow R_3 + 1$
STORE $M[314] \leftarrow R_3$

	1	2	3	4
	FI	DA	PO	EX.
		FI	DA	PO
			FI	DA
				FI

so seg EX: transfer memory word to R1
F0: Read M[313]
DA: Decode (increment) instruction
FI: Fetch the instruction from memory.

8. RISC → I LOAD R1 ← Memory[313]
A INCREMENT R1 ← R1+1
E

	Stage 1	2	3	4
instr 1.	I	A	E	
instr 2		I	A	E

Data hazards.

Before the completion of loading into R1 it is not possible to increment.

4. 4 floating-point pipeline processor.
each processor uses a cycle time of 40 ns.
total 400 floating-point operations are there.
so 400 operations will be divided into each of four processors
so processing time: $\frac{400}{4} \times 40 = 4000$ ns.

using a single pipeline, # cycle time is given 10 ns.
So processing time $400 \times 10 = 4000$ ns
so no change.

5. 250 billion floating-point operations so 250×10^9
100 megflop ⇒ 100 million floating point operation 100×10^6
so required time = $\frac{250 \times 10^9}{100 \times 10^6}$ sec.
= 2500 sec = 41.67 minutes.