Parallel Processing and Multiprocessors

In this chapter, we will study

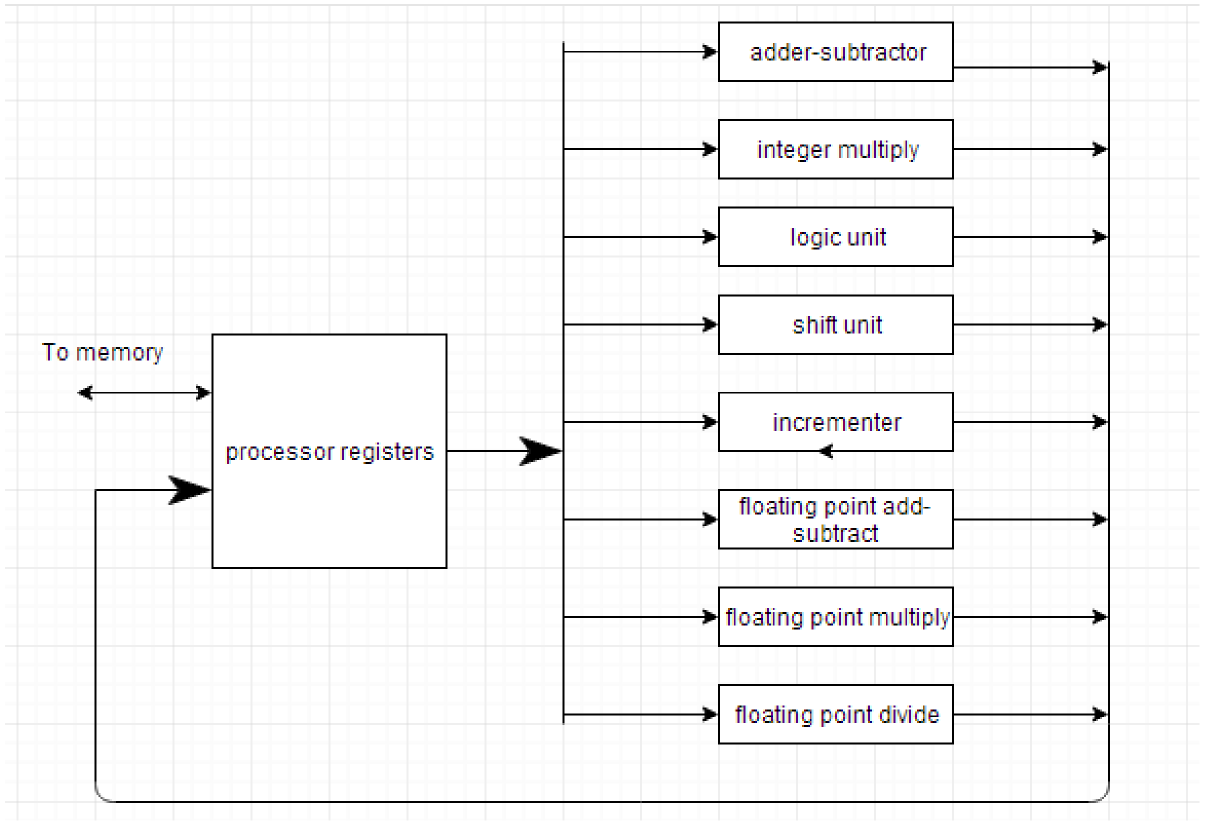
i. Parallel Processing

ii. Pipelining

Parallel Processing

A term used to denote a large class of techniques that are used to provide **simultaneous data-processing tasks** to **increase the computational speed of a computer system**.

Example of a Processor With Multiple Functional Units



Classification of Parallel Processing

Parallel processing can be classified based on:

>> the internal organization of the processors

>> interconnection structure between processors

>> flow of information through the system

Flynn’s Classification of Parallel Processing

Flynn’s classification divides computers into four major groups:

>> 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)

Instruction stream: Sequence of instructions read from memory

Data stream: Operations performed on the data in the processor

Pipelining

Pipelining

Pipelining is a technique of decomposing a sequential process into sub-operations, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments.

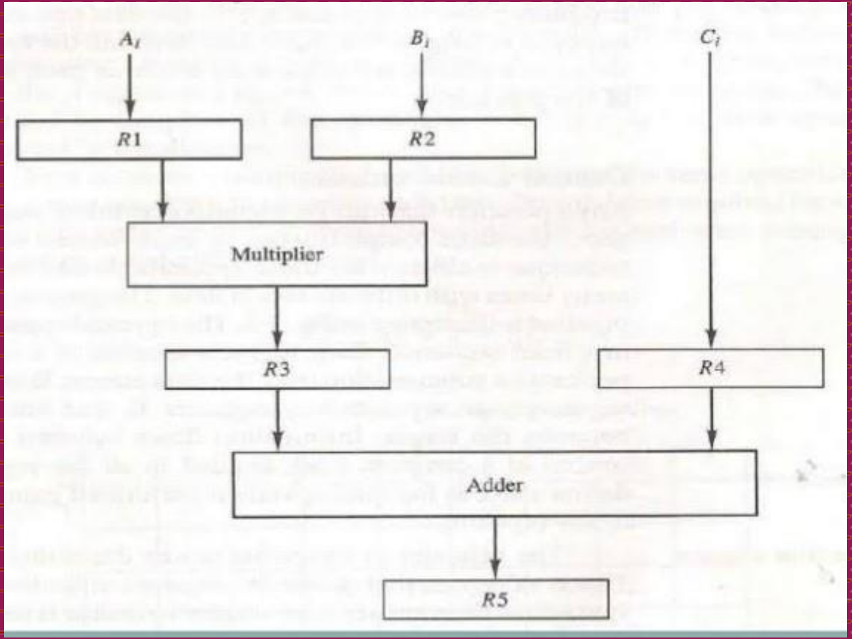
Example of Pipeline Organization

Say we want to perform combined multiplication and addition operation with a stream of numbers…

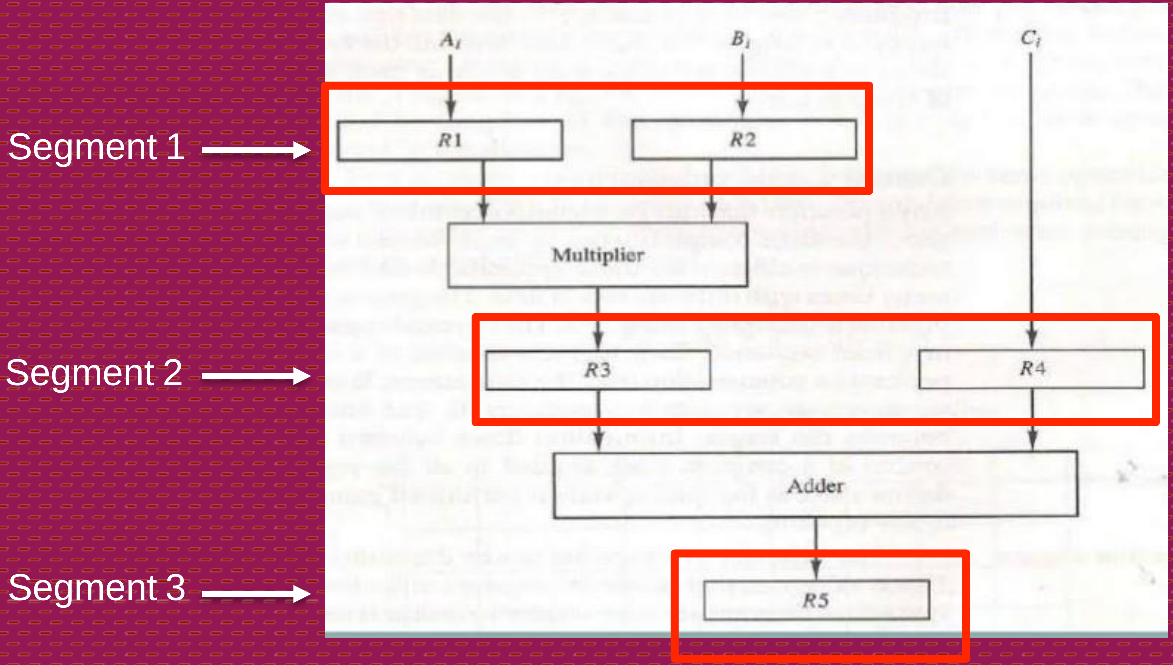
$A_i * B_i + C_i$ for $i = 1, 2, 3, \ldots., 7$

Each sub-operation is to be implemented in a segment within a pipeline; each segment has one or two registers and a combinational circuit.

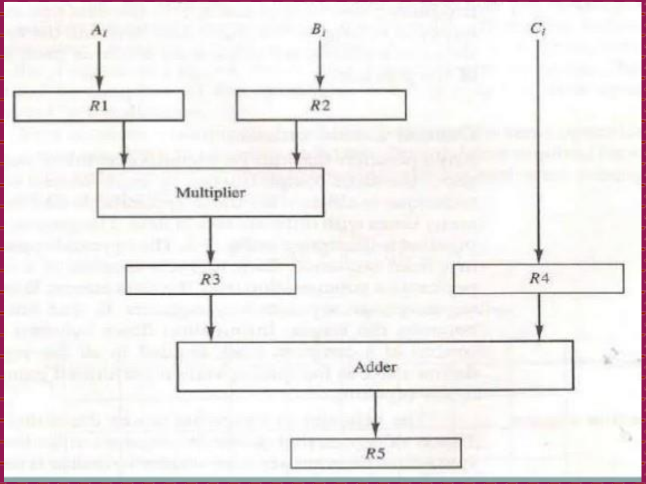
Example of Pipeline Organization



Example of Pipeline Organization



Example of Pipeline Organization



R1 to R5 are registers that receive new data with every clock pulse;
multipler and adder are combinational cicuits.
The sub-operations performed in each segment are as follows:

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

Pipeline Organization: General Considerations

Pipelining is efficient in those applications that need to repeat the same task many times with different sets of data.

Pipeline Organization: General Considerations

Speedup: Consider a case where a *k-segment* pipeline with a clock cycle time t_p is used to execute *n* tasks.

The first task requires time kt_p to complete its operation (since there are k segments in the pipeline).

The remaining $n – 1$ tasks emerge from the pipeine at the rate of one per clock cycle and they will be completed after time equal to $(n – 1)\ t_p$.

Therefore, completion of n tasks in a k-segment pipeline requires k + (n – 1) clock cycles.

Pipeline Organization: General Considerations

Speedup: In our current example, the time required to complete all operations is 3 + (7 – 1) = 9 clock cycles.

Without pipelining, the same operation would have taken 7 * 3 = 21 clock cycles (7 tasks and 3 clock cycles for each task).

Pipeline Organization: General Considerations

Speedup Ratio: The speedup of a pipeline processor over an equivalent non-pipeline processor is defined by the ratio

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

Where nt_n is the total time reqired by a non-pipeline unit to complete n tasks where each task takes time t_n to complete.

Pipeline Organization: General Considerations

Speedup Ratio: As the number of tasks increase, *n* become 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 non-pipeline circuits, we will have $t_n = kt_p$. Speedup then becomes

$$S = k$$

This shows that the maximum theoretical speedup that a pipeline can provide is equal to the number of segements in the pipeline.

Pipeline Organization: General Considerations

In order to duplicate the theoretical speedup advantage of a pipeline process, it is necessary to construct *k identical units* that will operate in parallel.

The implication is that a k-segment pipeline processor can be expected to equal the performance of k copies of an equivalent non-pipeline circuit under equal operating conditions.

Types of Pipelines

Arithmetic Pipelines

Instruction Pipelines

Arithmetic Pipeline

>> *Found in very high speed computers.*

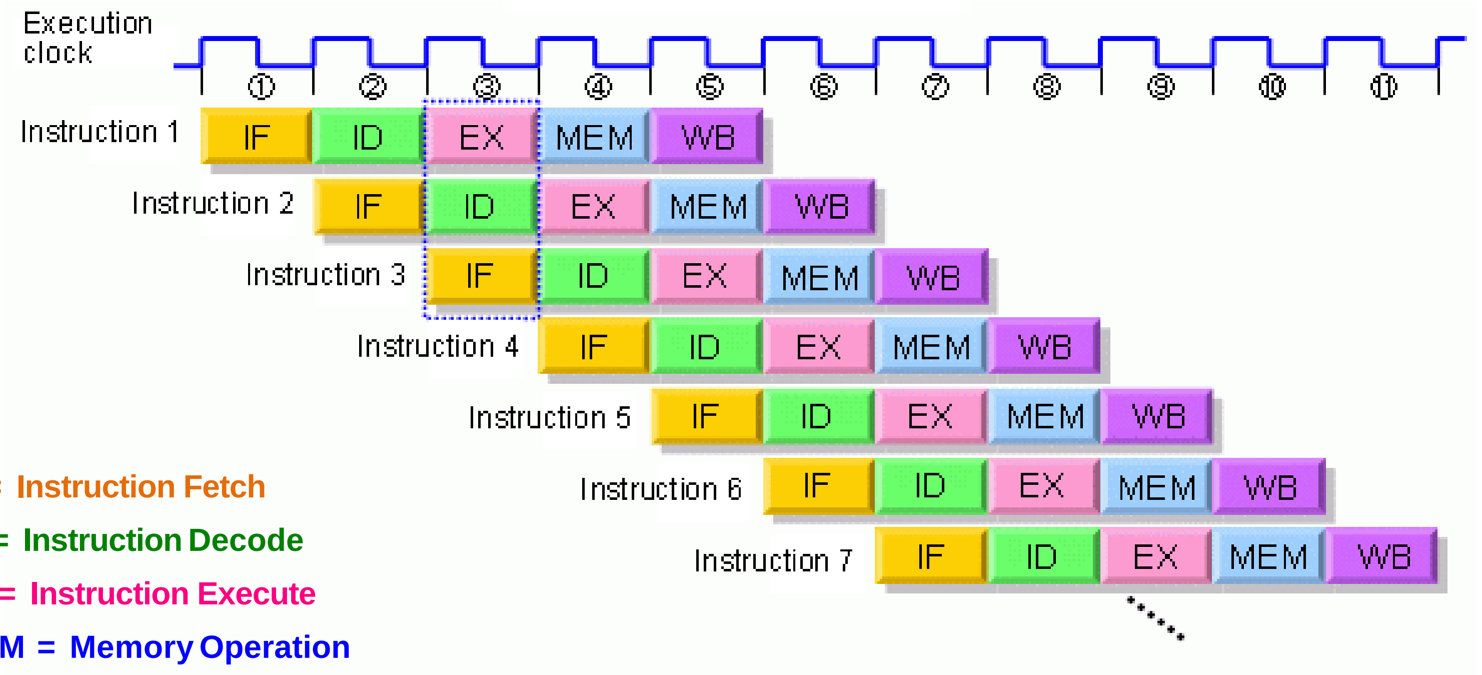
>> *Used to implement floating-point operations, multiplication of fixed-point numbers, and scientific problems.*

Instruction Pipeline

>> *Reads consecutive instructions from memory while previous instructions are being executed in other segments.*

>> *This causes instruction fetch and execute phases to overlap and perform simulataneous operations.*

5-Stage Instruction Pipeline



Execution clock
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪
Instruction 1 IF ID EX MEM WB
Instruction 2 IF ID EX MEM WB
Instruction 3 IF ID EX MEM WB
Instruction 4 IF ID EX MEM WB
Instruction 5 IF ID EX MEM WB
Instruction 6 IF ID EX MEM WB
Instruction 7 IF ID EX MEM WB

IF = Instruction Fetch

ID = Instruction Decode

EX = Instruction Execute

MEM = Memory Operation

WB = Write Back

Instruction Pipeline: Sequence of Steps

1. Fetch instruction from memory
2. Decode instruction
3. Calculate effective address
4. Fetch operand(s) from memory
5. Execute instruction
6. Store result in a proper place

Instruction Pipeline: Difficulties

Factors that prevent instruction pipeline from operating at its maximum rate:

1. **Different segments may take different times** to operate on the incoming information.
2. **Some segments may need to be skipped** for certain operations (for instance, a register mode instruction does not need an effective address computation phase).
3. **Two or more segments may require memory access at the same time**, causing one segment to wait until the other is finished.

Pipeline Hazards

Difficulties that cause the instruction pipeline to deviate from its normal operation:

1. **Resource Conflicts:** Caused by accesses to memory by two segments at the same time (aka STRUCTURAL HAZARDS).

2. **Data Dependency Conflicts**: Arise when an instruction depends on the result of previous instruction (aka DATA HAZARDS).

3. **Branch Difficulties:** May arise from branch and other instructions that change the value of PC (aka CONTROL HAZARDS).

Multiprocessors

Multiprocessor Systems

A multiprocessor system is an interconnection of **two or more CPUs** with **memory** and **input-output equipment**. They are MIMD systems.

The term “processor” in *multiprocessor* can mean either a CPU or an IOP.

A system with a single CPU and and one or more IOPs is usually not included in the definition of a multiprocessor system, unless the IOPs have computational facilities comparable to a CPU.

Multiprocessor Systems Versus Multicomputer Systems

System with Multiple Computers

Computers are interconnected with each other by means of communication lines to form a *computer network*. The network consists of several autonomous computers that **may or may not communicate with one another.**

System with Multiple Processors

Such a system is controlled by one OS that provides interaction between processors, and all the components of the system cooperate in the solution of a problem.

Advantages of Multiprocessor Systems

Multiprocessing improves reliability of the system

so that a failure or error in one part has a limited effect on the rest of the system.

Multiprocessing results in reduced loss of efficiency

in case of failure of a processor, because if one processor fails,

another can be assigned to perform its functions.

Advantages of Multiprocessor Systems (continued)

Multiprocessing improves system performance,

primarily because 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.**

Classification of Multiprocessors

Tightly Coupled aka Shared-Memory Multiprocessors

Multiprocessor systems with common shared memory.

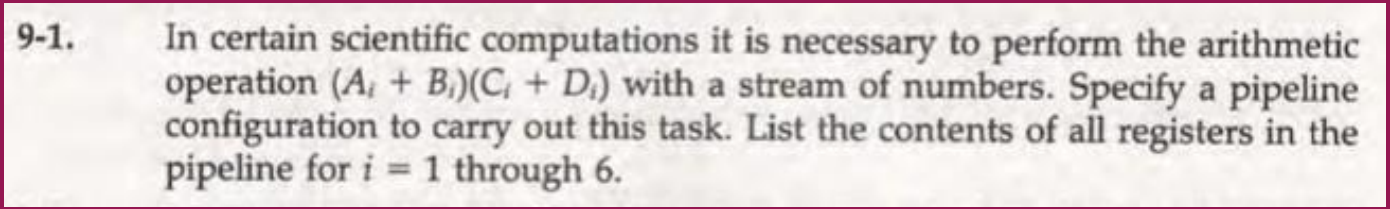
The CPUs still have their own local memories as well.

Loosely Coupled aka Distributed-Memory Multiprocessors

Each processor has its own private local memory.

The processors are tied together by a switching scheme designed to route information from one processor to another through a message-passing scheme.

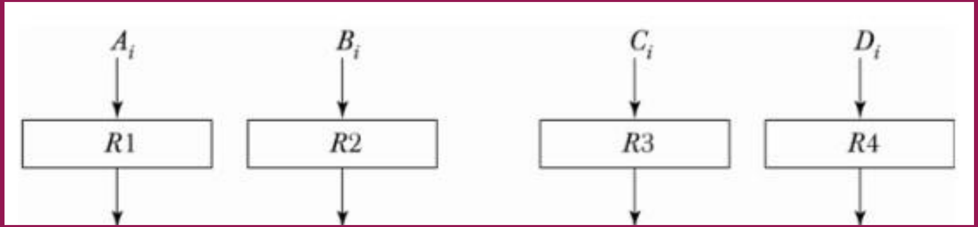
Exercises



9-1. In certain scientific computations it is necessary to perform the arithmetic operation $(A_i + B_i)(C_i + D_i)$ with a stream of numbers. Specify a pipeline configuration to carry out this task. List the contents of all registers in the pipeline for $i = 1$ through 6.

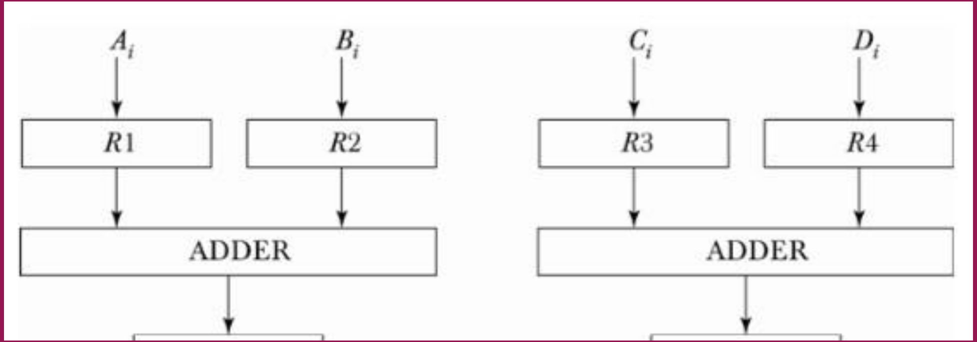
Solution:

$$(A_i + B_i)(C_i + D_i)$$



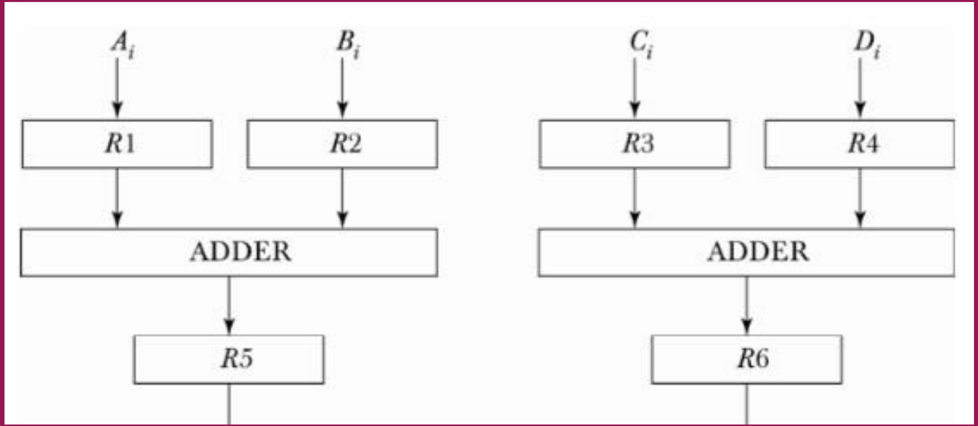
Solution:

$$(A_i + B_i)(C_i + D_i)$$



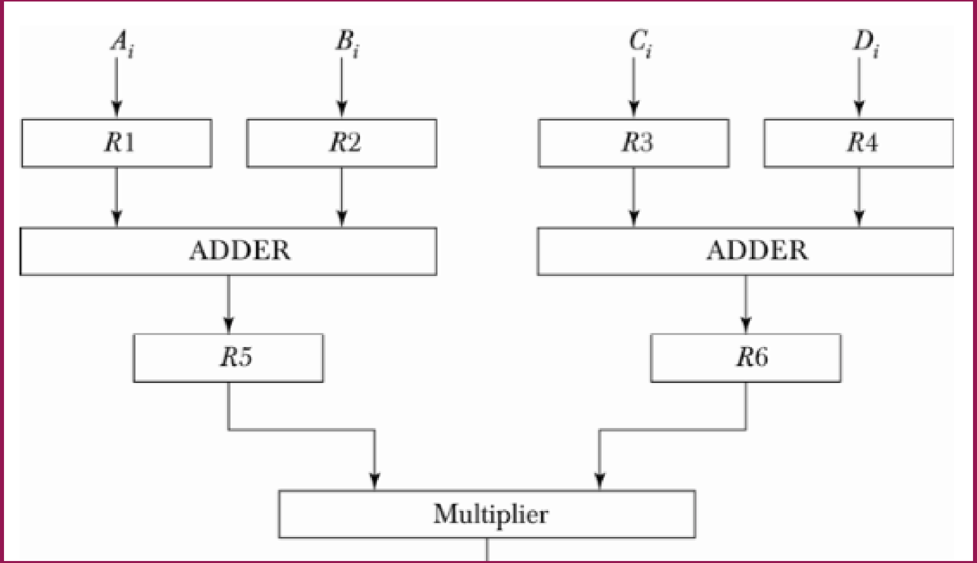
Solution:

$$(A_i + B_i)(C_i + D_i)$$



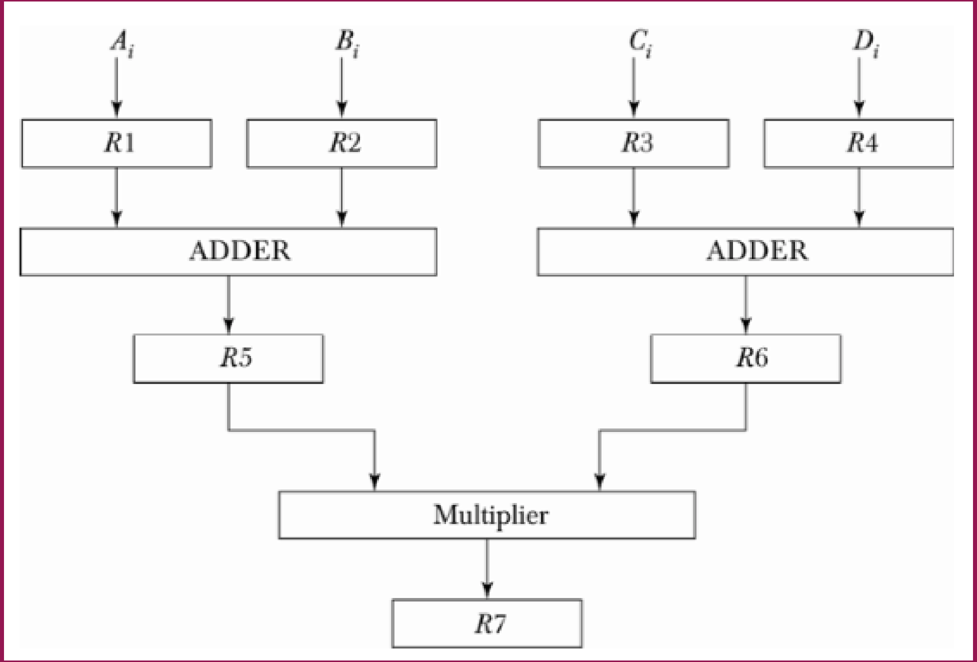
Solution:

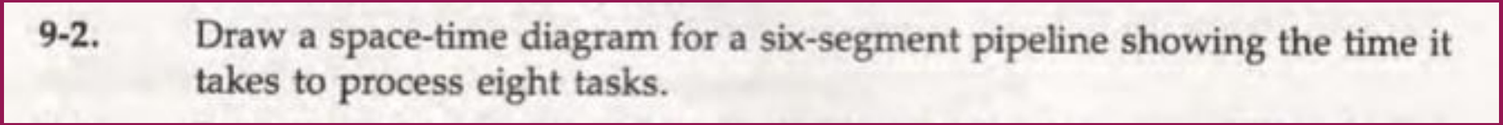
$$(A_i + B_i)(C_i + D_i)$$

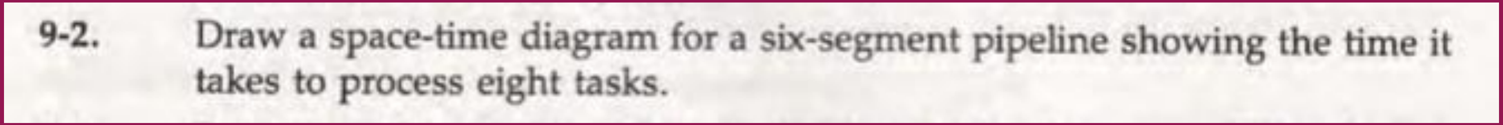


Solution:

$$(A_i + B_i)(C_i + D_i)$$

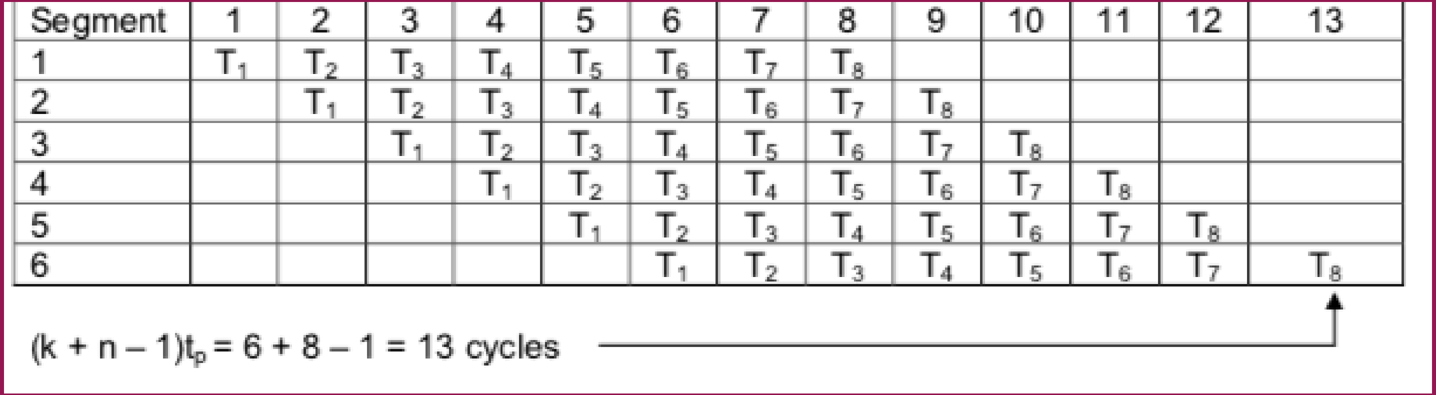



9-2. Draw a space-time diagram for a six-segment pipeline showing the time it takes to process eight tasks.



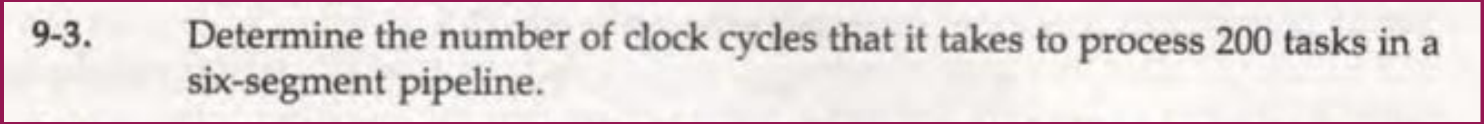
9-2. Draw a space-time diagram for a six-segment pipeline showing the time it takes to process eight tasks.

Solution:

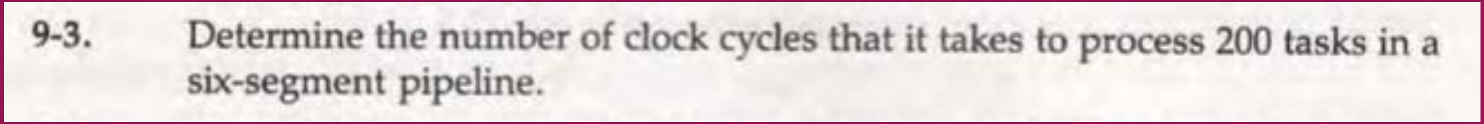


Segment	1	2	3	4	5	6	7	8	9	10	11	12	13
1	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8					
2		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8				
3			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8			
4				T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8		
5					T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	
6						T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8

$(k + n - 1)t_p = 6 + 8 - 1 = 13$ cycles



9-3. Determine the number of clock cycles that it takes to process 200 tasks in a six-segment pipeline.

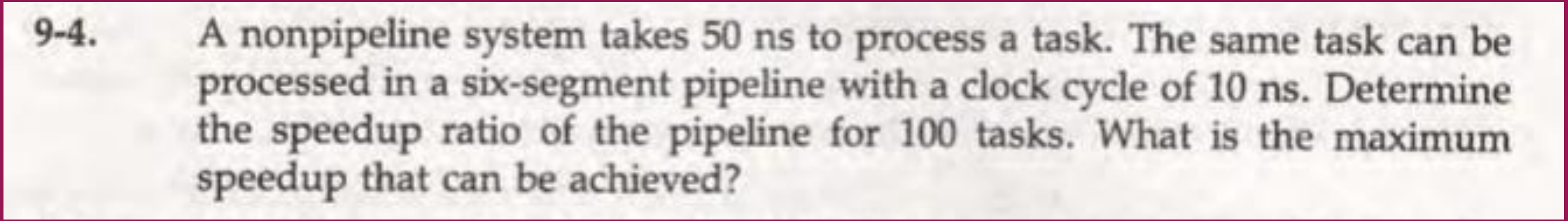

9-3. Determine the number of clock cycles that it takes to process 200 tasks in a six-segment pipeline.

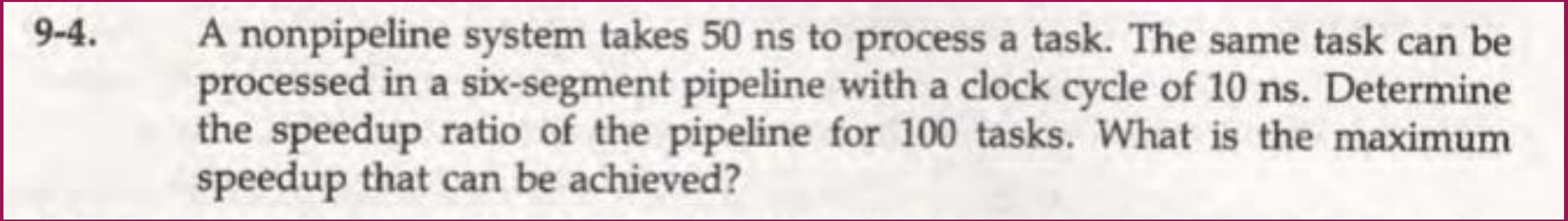
Solution:

Number of segments (k) = 6

Number of tasks (n) = 200

Clocks cycles required = k + n – 1 = 6 + 200 – 1 = 205


9-4. A nonpipeline system takes 50 ns to process a task. The same task can be processed in a six-segment pipeline with a clock cycle of 10 ns. Determine the speedup ratio of the pipeline for 100 tasks. What is the maximum speedup that can be achieved?


9-4. A nonpipeline system takes 50 ns to process a task. The same task can be processed in a six-segment pipeline with a clock cycle of 10 ns. Determine the speedup ratio of the pipeline for 100 tasks. What is the maximum speedup that can be achieved?

Solution:

$t_n = 50$ ns

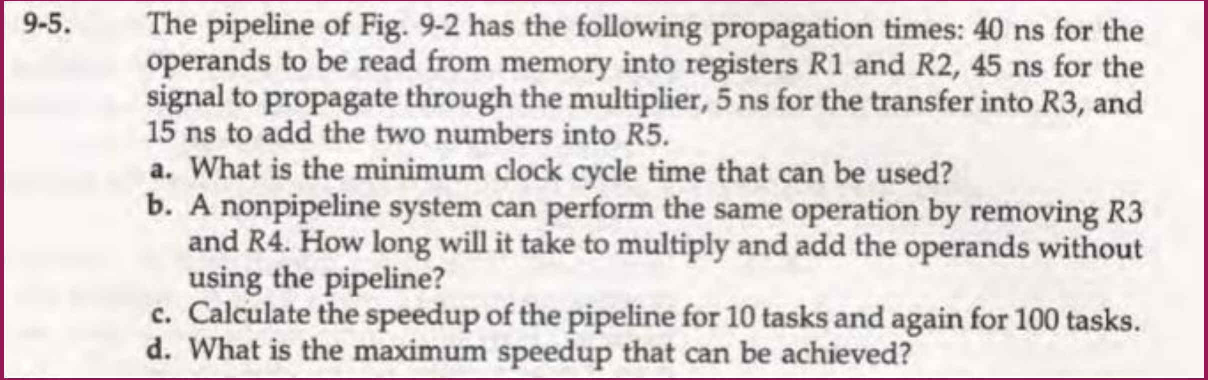
$k = 6$

$t_p = 10$ ns

$n = 100$

$S = n\, t_n / (k + n - 1)\, t_p = (100 \times 50) / [(6 + 100 - 1) \times 100] = 4.76$

$S_{max} = t_n / t_p = 50 / 10 = 5$



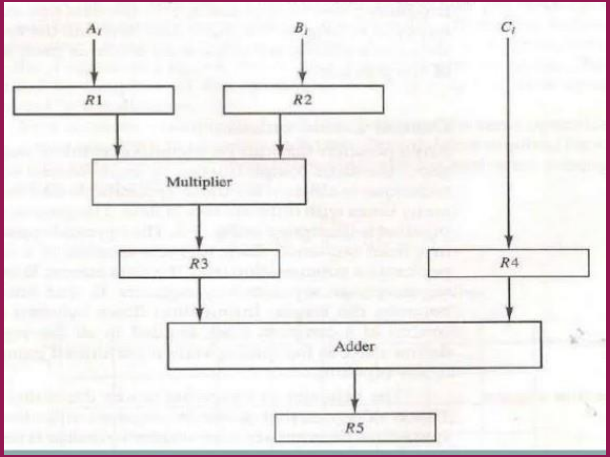
9-5. The pipeline of Fig. 9-2 has the following propagation times: 40 ns for the operands to be read from memory into registers $R1$ and $R2$, 45 ns for the signal to propagate through the multiplier, 5 ns for the transfer into $R3$, and 15 ns to add the two numbers into $R5$.

a. What is the minimum clock cycle time that can be used?

b. A nonpipeline system can perform the same operation by removing $R3$ and $R4$. How long will it take to multiply and add the operands without using the pipeline?

c. Calculate the speedup of the pipeline for 10 tasks and again for 100 tasks.

d. What is the maximum speedup that can be achieved?



Solution:

a. What is the minimum clock cycle time that can be used?

$t_p = 45 + 5 = 50$ ns $\quad (k = 3)$

b. A nonpipeline system can perform the same operation by removing $R3$ and $R4$. How long will it take to multiply and add the operands without using the pipeline?

$t_n = 40 + 45 + 15 = 100$ ns

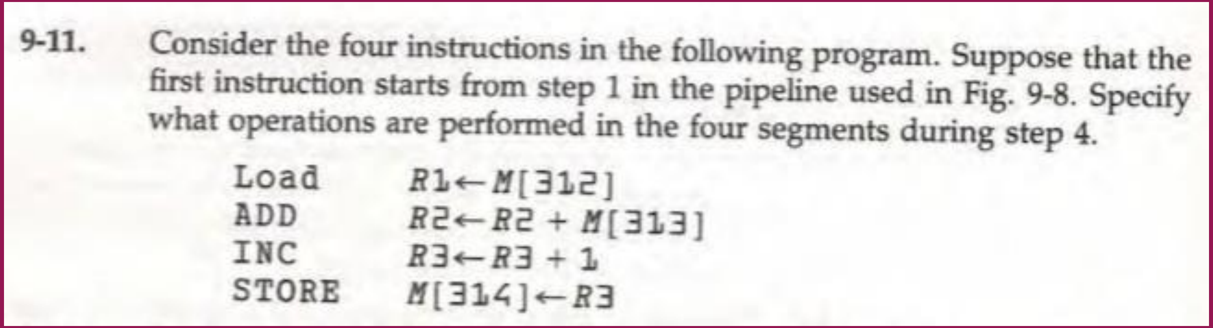
c. Calculate the speedup of the pipeline for 10 tasks and again for 100 tasks.

$S = n\,t_n / (k + n - 1)\,t_p = (10 \times 100) / [(3 + 9) \times 50] = 1.67$ (for $n = 10$)

$S = (100 \times 100) / [(3 + 99) \times 50] = 1.96$ (for $n = 100$)

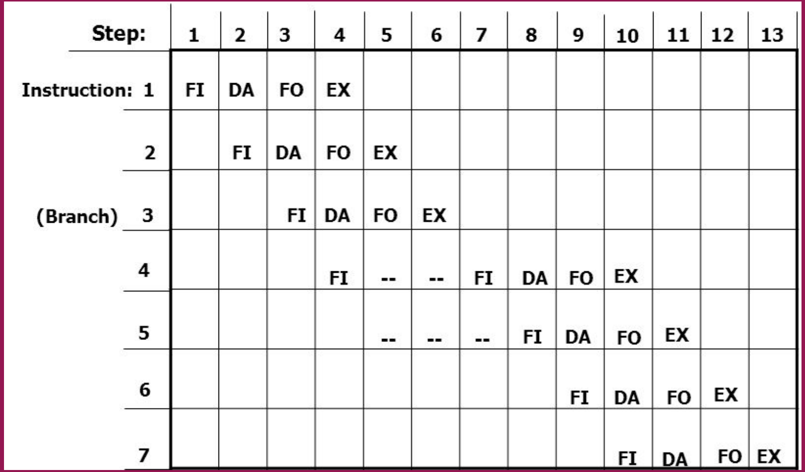
d. What is the maximum speedup that can be achieved?

$S_{max} = t_n / t_p = 100 / 50 = 2$



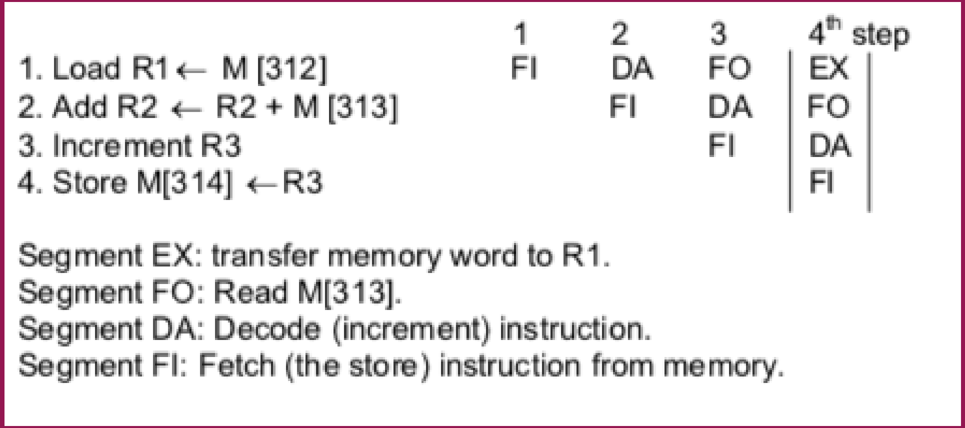
9-11. Consider the four instructions in the following program. Suppose that the first instruction starts from step 1 in the pipeline used in Fig. 9-8. Specify what operations are performed in the four segments during step 4.

```
Load    R1←M[312]
ADD     R2←R2 + M[313]
INC     R3←R3 + 1
STORE   M[314]←R3
```



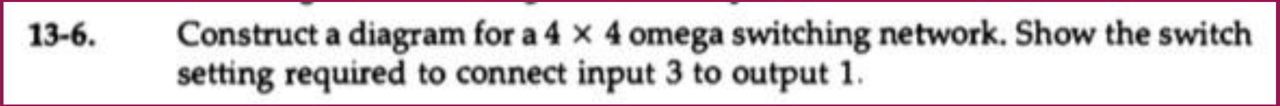
Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction: 1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
(Branch) 3			FI	DA	FO	EX							
4				FI	--	--	FI	DA	FO	EX			
5					--	--	--	FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

Solution:

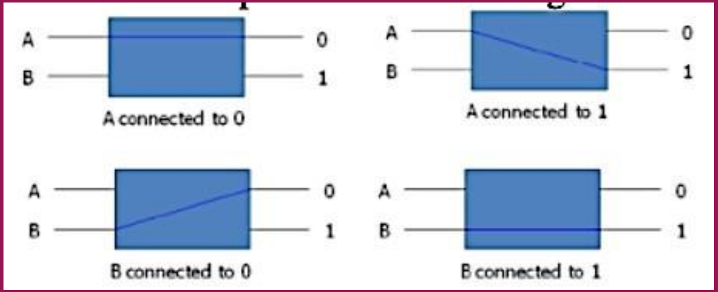


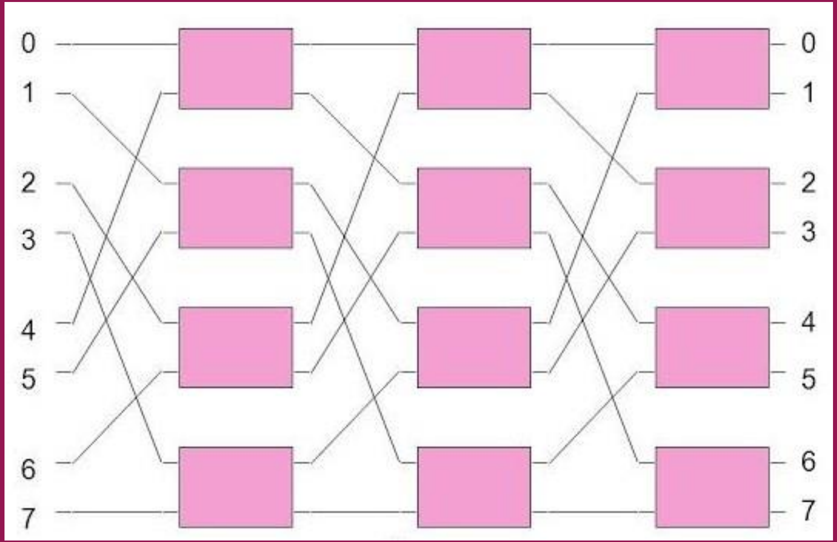
	1	2	3	4th step
1. Load R1 $\leftarrow$ M [312]	FI	DA	FO	EX
2. Add R2 $\leftarrow$ R2 + M [313]		FI	DA	FO
3. Increment R3			FI	DA
4. Store M[314] $\leftarrow$ R3				FI

Segment EX: transfer memory word to R1.
Segment FO: Read M[313].
Segment DA: Decode (increment) instruction.
Segment FI: Fetch (the store) instruction from memory.



13-6. Construct a diagram for a 4 × 4 omega switching network. Show the switch setting required to connect input 3 to output 1.

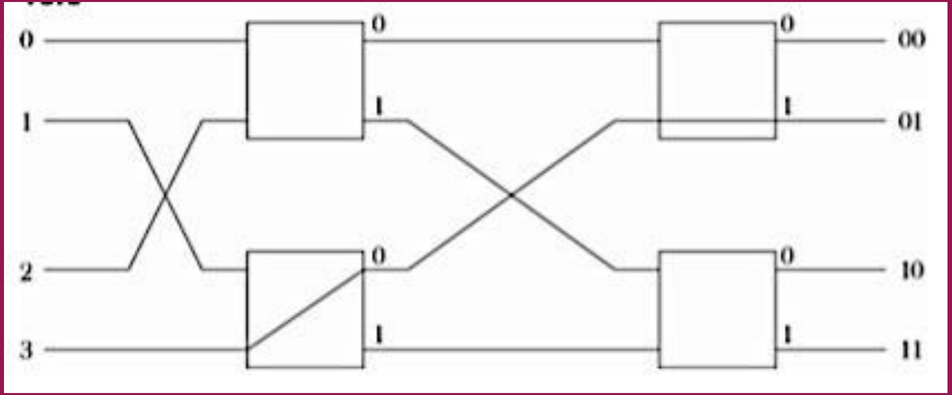


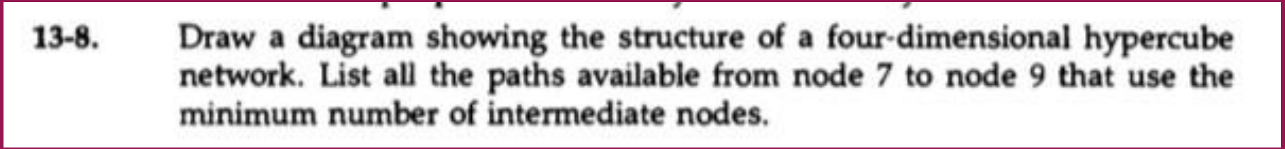


8 x 8 omega switching network

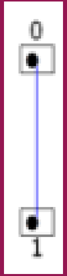
Solution:

Solution:

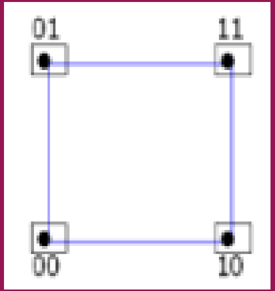




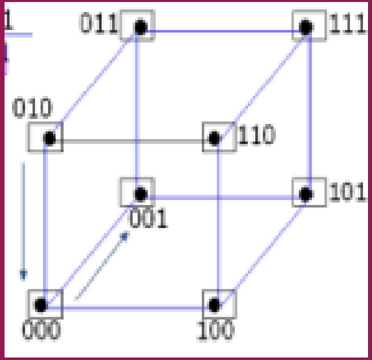
13-8. Draw a diagram showing the structure of a four-dimensional hypercube network. List all the paths available from node 7 to node 9 that use the minimum number of intermediate nodes.



One-Cube

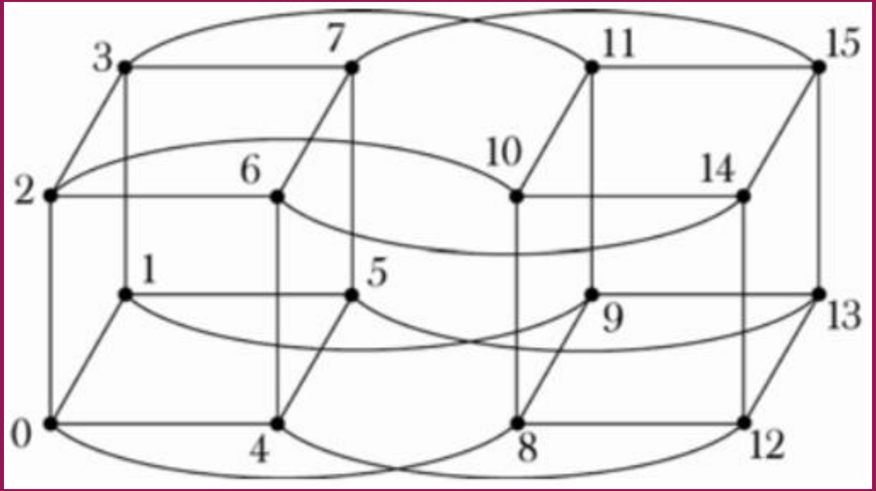


Two-Cube

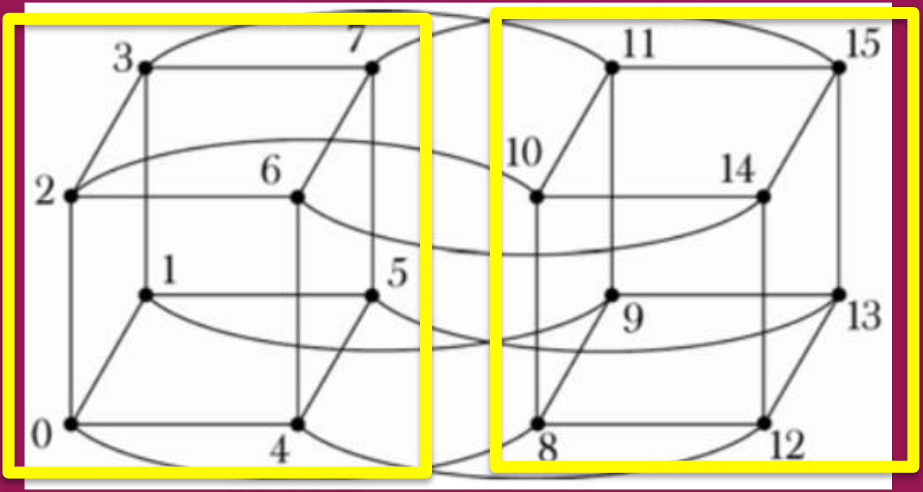


Three-Cube

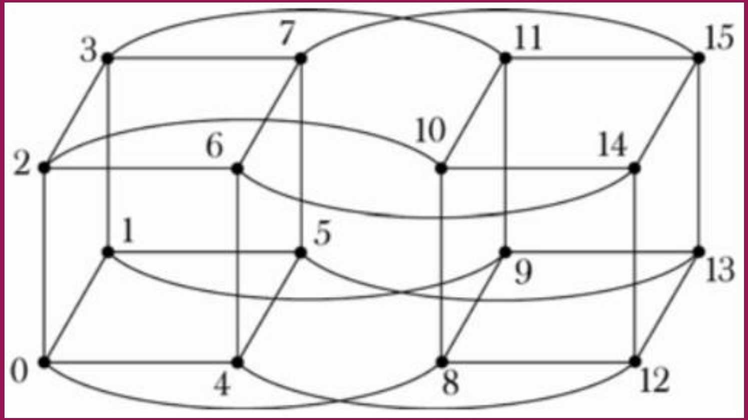
Solution:



Solution:



Solution:



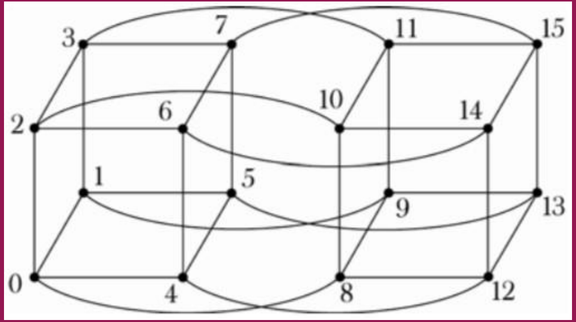
Paths available from node 7 to node 9 i.e., 0111 to 1001:

0111 XOR 1001 = 1110
So, we will have to traverse three axes (via two nodes).

Possible paths:

7 – 15 – 13 – 9
7 – 15 – 11 – 9
7 – 3 – 11 – 9
7 – 3 – 1 – 9
7 – 5 – 13 – 9
7 – 5 – 1 – 9

Solution:



Path 1: 7 – 15 – 13 – 9
(0111 – 1111 – 1101 – 1001)

Path 2: 7 – 15 – 11 – 9
(0111 – 1111 – 1011 – 1001)

Path 3: 7 – 3 – 11 – 9
(0111 – 0011 – 1011 – 1001)

Path 4: 7 – 3 – 1 – 9
(0111 – 0011 – 0001 – 1001)

Path 5: 7 – 5 – 13 – 9
(0111 – 0101 – 1011 – 1001)

Path 6: 7 – 5 – 1 – 9
(0111 – 0101 – 0001 – 1001)