DICE
DEPARTMENT OF INTERDISCIPLINARY COURSES IN ENGINEERING

CHITKARA UNIVERSITY

Computer Organization and Architecture

Introduction to Parallel Processing , Pipelining , DMA

Parallel Processing

- Parallel processing can be described as a class of techniques which enables the system to achieve simultaneous data-processing tasks to *increase the computational speed of a computer system.*

- The primary purpose of parallel processing is to speed up the *computer processing capability* and increase its throughput, i.e. the amount of processing that can be accomplished during a given interval of time.

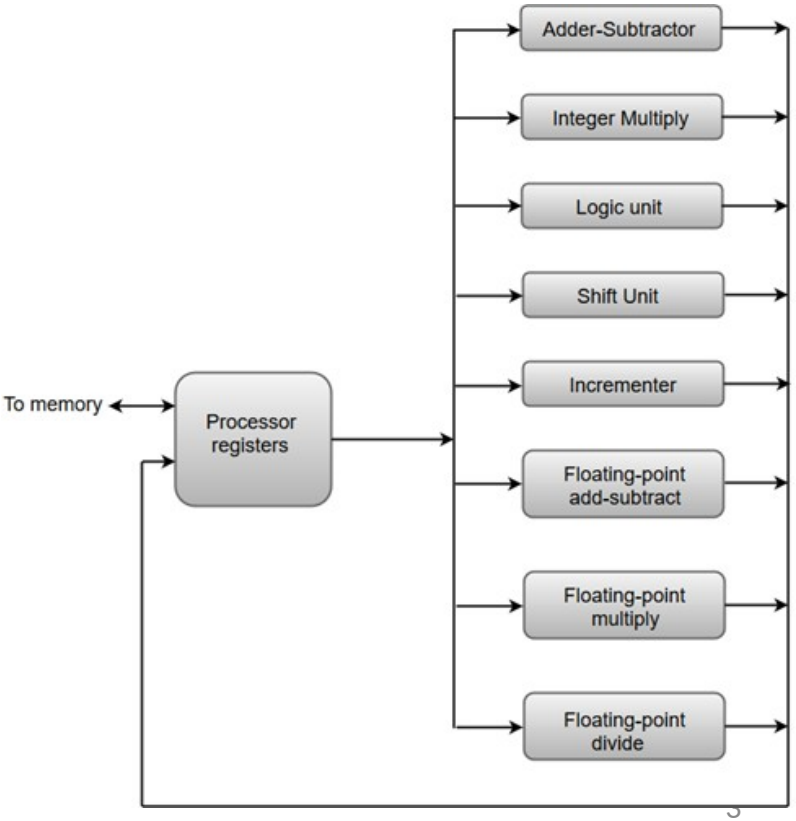
- A parallel processing system is able to perform concurrent data processing to achieve faster execution time. For example, while an instruction is being executed in the ALU, the next instruction can be read from memory.

Parallel Processing

The following diagram shows one possible way of separating the execution unit into eight functional units operating in parallel.

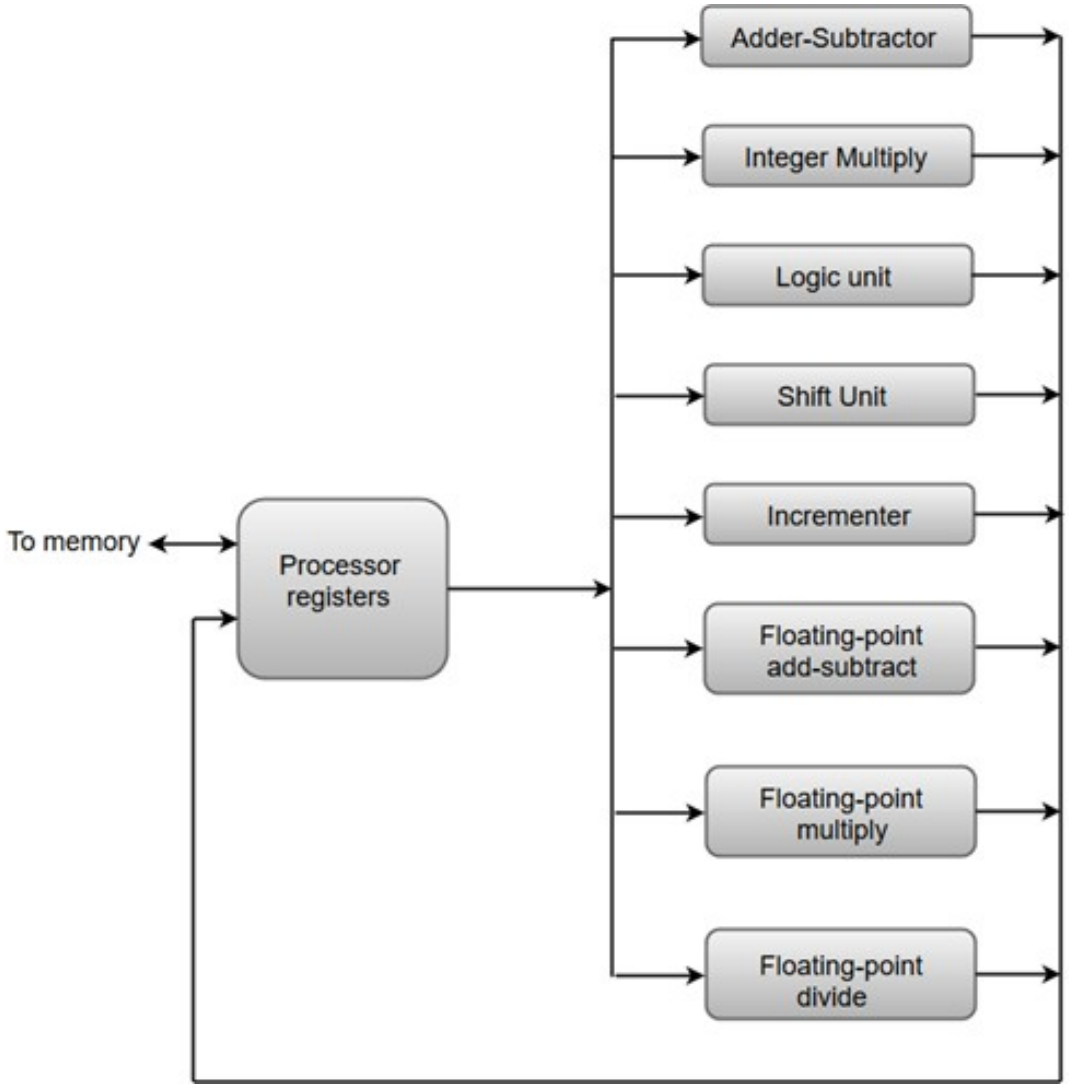
The operation performed in each functional unit is indicated in each block of the diagram:

- The adder and integer multiplier performs the arithmetic operation with integer numbers.
- The floating-point operations are separated into three circuits operating in parallel.
- The logic, shift, and increment operations can be performed concurrently on different data. All units are independent of each other, so one number can be shifted while another number is being incremented.



Parallel Processing

- The adder and integer multiplier performs the arithmetic operation with integer numbers.
- The floating-point operations are separated into three circuits operating in parallel.
- The logic, shift, and increment operations can be performed concurrently on different data. All units are independent of each other, so one number can be shifted while another number is being incremented.



Pipelining

- The term Pipelining refers to a technique of *decomposing a sequential process into sub-operations,* with each sub-operation being executed in a dedicated segment that operates concurrently with all other segments.

- The most important characteristic of a pipeline technique is that several computations can be in progress in distinct segments at the same time.

- 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.

- The structure of a pipeline organization can be represented simply by including an input register for each segment followed by a combinational circuit.

Pipelining

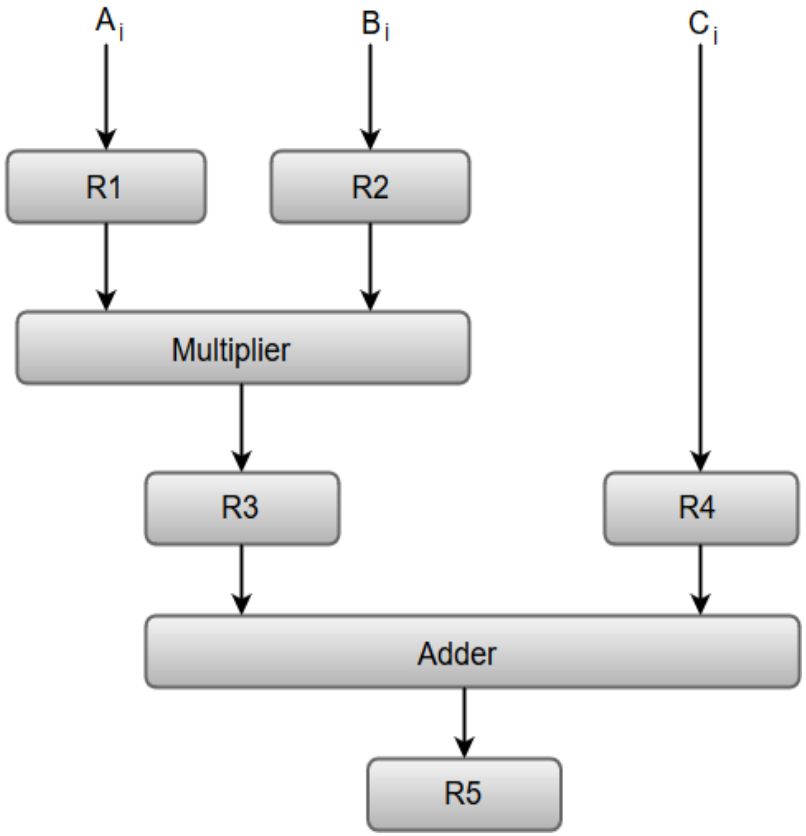
Let us consider an example of combined multiplication and addition operation to get a better understanding of the pipeline organization.

The combined multiplication and addition operation is done with a stream of numbers such as:

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

The operation to be performed on the numbers is decomposed into sub-operations with each sub-operation to be implemented in a segment within a pipeline.

Pipeline Processing:



Pipelining

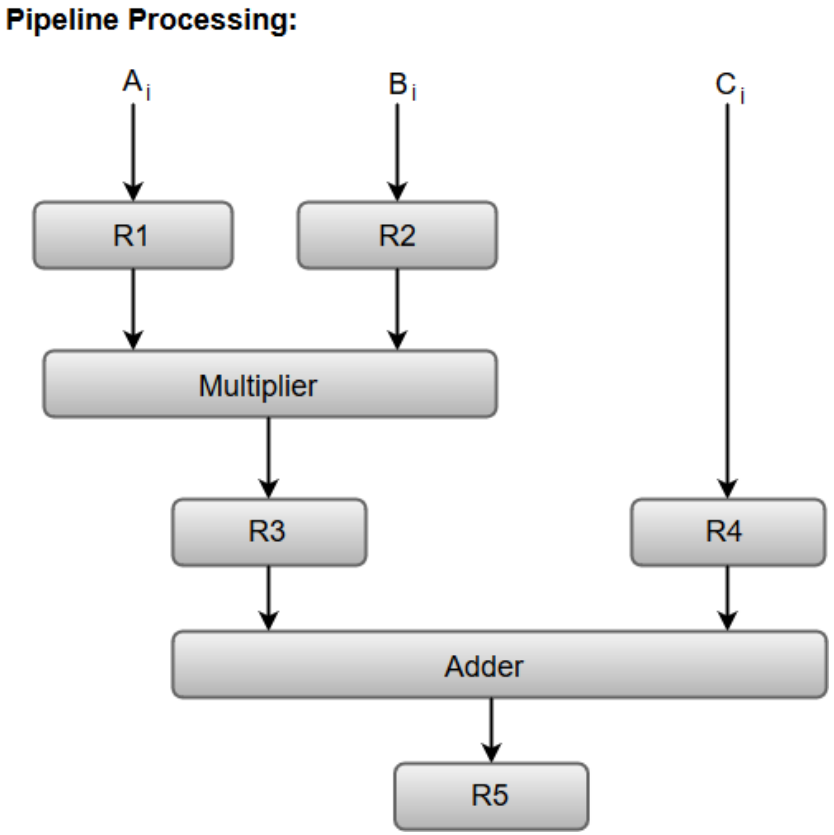
The sub-operations performed in each segment of the pipeline are defined as:

$R1 \leftarrow A_i$, $R2 \leftarrow B_i$ Input A_i, and 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 following block diagram represents the combined as well as the sub-operations performed in each segment of the pipeline.

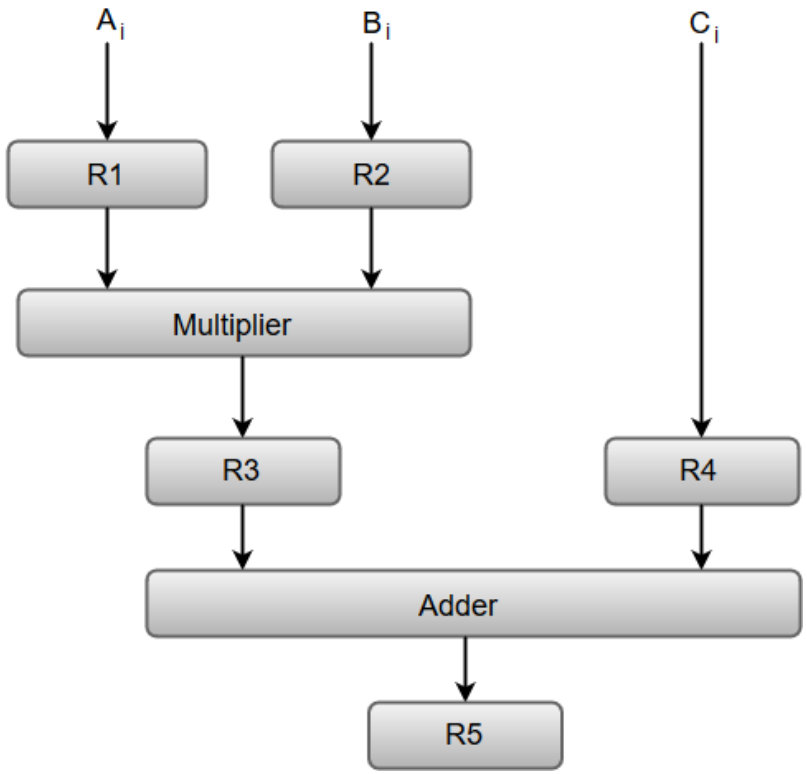


Pipelining

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

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	$R1$	$R2$	$R3$	$R4$	$R5$
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	—	—	—	—	$A_7 * B_7 + C_7$

Pipeline Processing:



Pipelining

Instruction Pipeline

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- Most of the digital computers with complex instructions require instruction pipeline to carry out operations like fetch, decode and execute instructions.

In general, the computer needs to process each instruction with the following sequence of steps.

1. Fetch instruction from memory.
2. Decode the instruction.
3. Calculate the effective address.
4. Fetch the operands from memory.
5. Execute the instruction.
6. Store the result in the proper place.

Pipelining

Instruction Pipeline

Each step is executed in a particular segment, and there are times when different segments may take different times to operate on the incoming information.

Moreover, there are times when two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

The organization of an instruction pipeline will be more efficient if the instruction cycle is divided into segments of equal duration. One of the most common examples of this type of organization is a **Four-segment instruction pipeline.**

Pipelining

Instruction Pipeline

- A ***four-segment instruction pipeline*** combines two or more different segments and makes it as a single one.

- For instance, the decoding of the instruction can be combined with the calculation of the effective address into one segment.

The following block diagram (next slide) shows a typical example of a four-segment instruction pipeline. The instruction cycle is completed in four segments.

Segment 1: The instruction fetch segment can be implemented using first in, first out (FIFO) buffer.
Segment 2: The instruction fetched from memory is decoded in the second segment, and eventually, the effective address is calculated in a separate arithmetic circuit.
Segment 3: An operand from memory is fetched in the third segment.
Segment 4: The instructions are finally executed in the last segment of the pipeline organization.

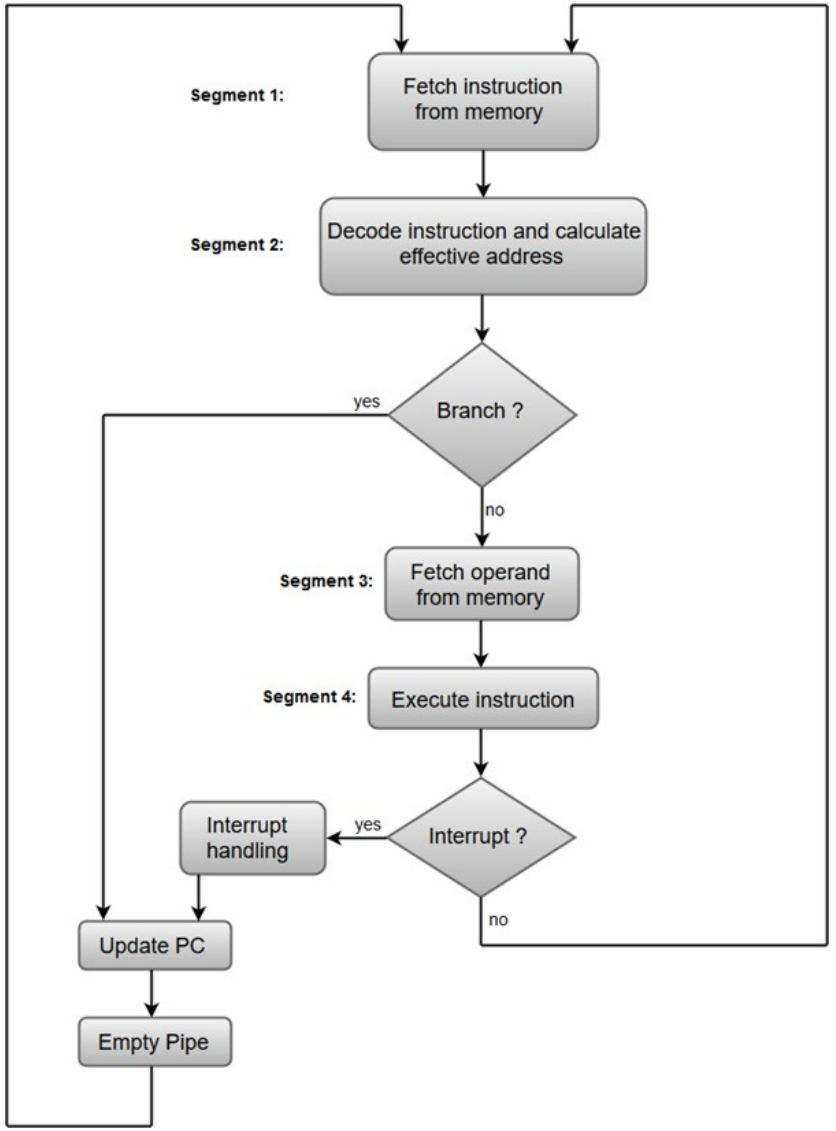
Pipelining

Instruction Pipeline

1. FI is the segment that fetches an instruction.
2. DA is the segment that decodes the instruction and calculates the effective address.
3. FO is the segment that fetches the operand.
4. EX is the segment that execute the instruction.

Step	1	2	3	4	5	6	7	8	9
1	FI	DA	FO	EX					
2		FI	DA	FO	EX				
3			FI	DA	FO	EX			
4				FI	DA	FO	EX		
5					FI	DA	FO	EX	
6						FI	DA	FO	EX

Fig: timing diagram for 4-segment instruction pipeline



Pipelining Hazard

- Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycles.
- Any condition that causes a stall in the pipeline operations can be called a hazard.

There are primarily three types of hazards:

i. Data Hazards

ii. Control Hazards or instruction Hazards

iii. Structural Hazards

Pipelining Hazard

i. Data Hazards:

- A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline.

- As a result of which some operation has to be delayed and the pipeline stalls. Whenever there are two instructions one of which depends on the data obtained from the other.

A=3+A
B=A*4
For the above sequence, the second instruction needs the value of 'A' computed in the first instruction.

Thus the second instruction is said to depend on the first.

Pipelining Hazard

i. Data Hazards:

Consider the following set of instructions in a 5-stage pipeline.
Operands are read in ID.
MEM is memory Write for result; RW is Register Write for result

ADD R3, R6, R5	- Results to be written in R3
SUB R4, R3, R5	- R3 has one of the operand
OR R6, R3, R7	- R3 has one of the operand
AND R8, R3, R7	- R3 has one of the operand
XOR R12, R3, R10	- R3 has one of the operand

R3 is accessed in READ mode; expect the result of ADD to be available in R3

But result of ADD written in R3 at t5

	t1	t2	t3	t4	t5	t6	t7	t8	t9
ADD R3, R6, R5	IF	ID	IE	MEM	RW R3	--	--	--	--
SUB R4, R3, R5	--	IF	ID R3	IE	MEM	RW	--	--	--
OR R6, R3, R7	--	--	IF	ID R3	IE	MEM	RW	--	--
AND R8, R3, R9	--	--	--	IF	ID R3	IE	MEM	RW	--
XOR R10, R3, R11	--	--	--	--	IF	ID R3	IE	MEM	RW

Note when each instruction is accessing R3

Solution 1: Introduce three bubbles at SUB instruction IF stage. This will facilitate SUB – ID to function at t6. Subsequently, all the following instructions are also delayed in the pipe.

Pipelining Hazard

ii. Structural Hazards:

- This situation arises mainly when two instructions require a given hardware resource at the same time and hence for one of the instructions the pipeline needs to be delayed.

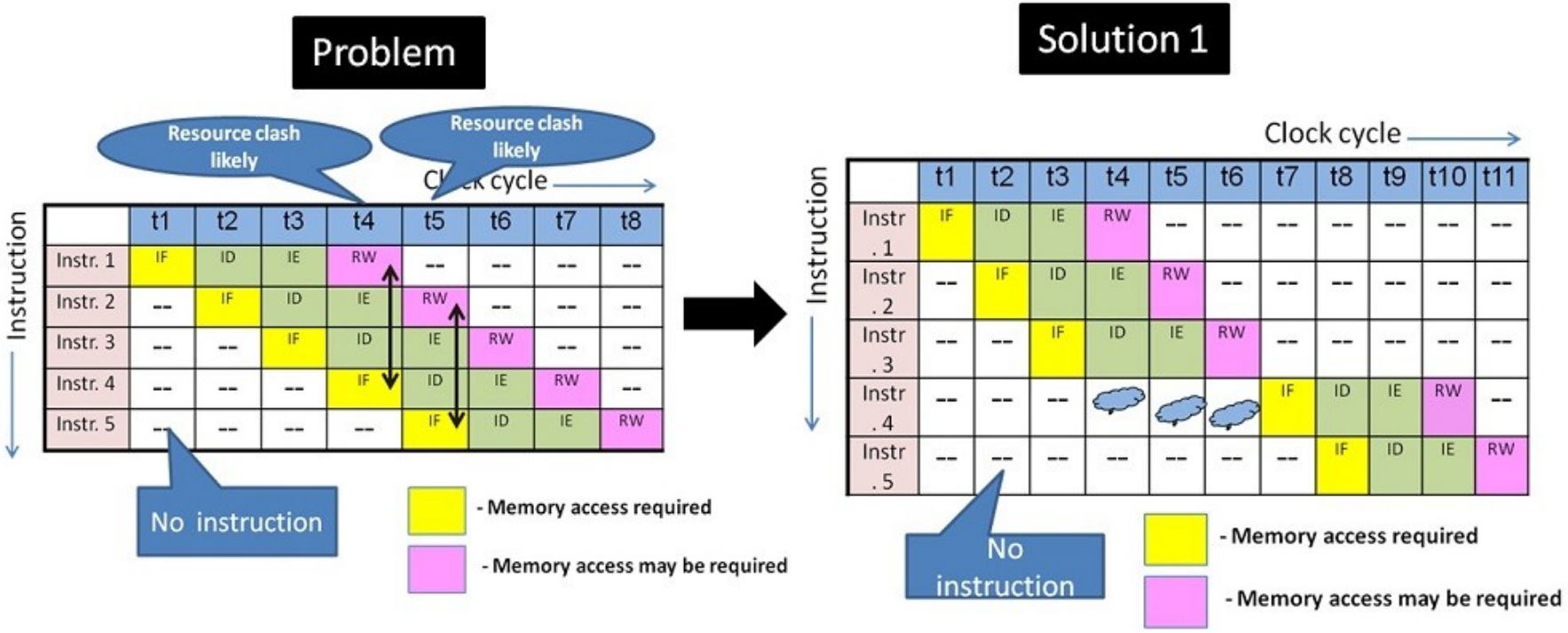
- The most common case is when memory is accessed at the same time by two instructions. One instruction may need to access the memory as part of the Execute or Write back phase while other instruction is being fetched.

- In this case if both the instructions and data reside in the same memory. Both the instructions can't proceed together and one of them needs to be stalled till the other is done with the memory access part.

- Thus in general sufficient hardware resources are needed for avoiding structural hazards.

Pipelining Hazard

ii. **Structural Hazards:**



Pipelining Hazard

iii. Control hazards:

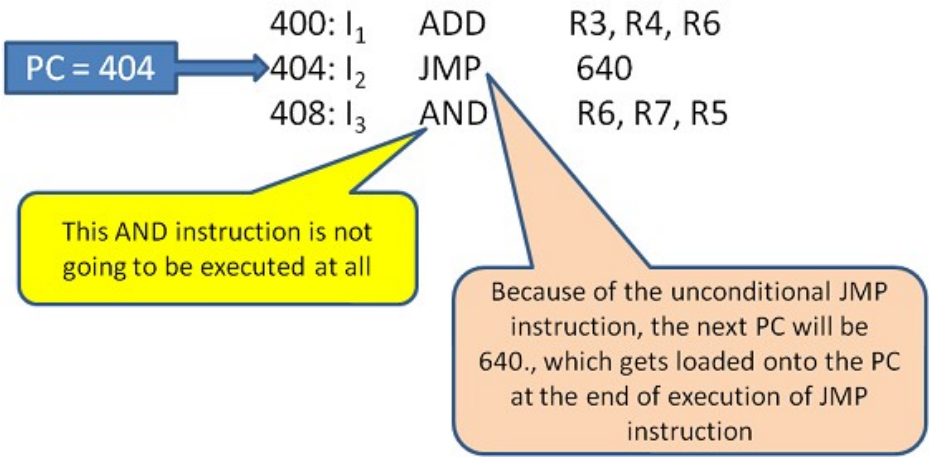
- The instruction fetch unit of the CPU is responsible for providing a stream of instructions to the execution unit. The instructions fetched by the fetch unit are in consecutive memory locations and they are executed.

- However the problem arises when one of the instructions is a branching instruction to some other memory location. Thus all the instruction fetched in the pipeline from consecutive memory locations are invalid now and need to removed(also called flushing of the pipeline).This induces a stall till new instructions are again fetched from the memory address specified in the branch instruction.

- Thus the time lost as a result of this is called a branch penalty. Often dedicated hardware is incorporated in the fetch unit to identify branch instructions and compute branch addresses as soon as possible and reducing the resulting delay as a result.

Pipelining Hazard

iii. **Control hazards:**



Solution: Reordering instructions - Delayed branch i.e. reordering the instructions to position the branch instruction later in the order, such that safe and useful instructions which are not affected by the result of a branch are brought-in earlier in the sequence thus delaying the branch instruction fetch. If no such instructions are available then NOP is introduced. This delayed branch is applied with the help of Compiler.

Thank You