I/O Organization

The Input / output organization of computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the computer, provides an efficient mode of communication between the central system and the outside environment

The most common input output devices are:

i) Monitor

ii) Keyboard

iii) Mouse

iv) Printer

v) Magnetic tapes

The devices that are under the direct control of the computer are said to be **connected online.**

Input - Output Interface

Input Output Interface provides a method for transferring information between internal storage and external I/O devices.

Peripherals connected to a computer need **special communication links** for interfacing them with the central processing unit.

The purpose of **communication link** is to **resolve the differences that exist between the central computer and each peripheral.**

The Major Differences are:-

1. Peripherals are electromechnical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed.
2. The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in the peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To Resolve these differences, computer systems include special hardware components between the CPU and Peripherals to **supervises and synchronizes all input and out transfers**

- These components are called **Interface Units** because they interface between the processor bus and the peripheral devices.

I/O BUS and Interface Module

It defines the typical link between the processor and several peripherals.

The I/O Bus consists of **data lines, address lines** and **control lines**.

The I/O bus from the processor is attached to all peripherals interface.

To communicate with a particular device, the processor places a device address on address lines.

Each Interface **decodes** the address and control received from the I/O bus, interprets them for peripherals and provides signals for the **peripheral controller**.

It is also synchronizes the data flow and supervises the transfer between peripheral and processor.

Each peripheral has its own controller.

For example, the printer controller controls the paper motion, the print timing

The control lines are referred as I/O command. The commands are as following:

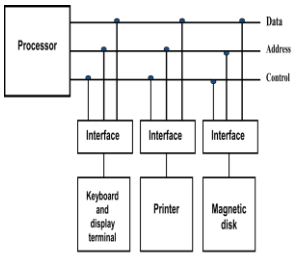
Control command- A control command is issued to activate the peripheral and to inform it what to do.

Status command- A status command is used to test various status conditions in the interface and the peripheral.

Data Output command- A data output command causes the interface to respond by transferring data from the bus into one of its registers.

Data Input command- The data input command is the opposite of the data output.

In this case the interface receives on item of data from the peripheral and places it in its buffer register



Connection of I/O bus to input-output devices

I/O Versus Memory Bus

To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines. There are 3 ways that computer buses can be used to communicate with memory and I/O:

i. Use two Separate buses , one for memory and other for I/O.

ii. Use one common bus for both memory and I/O but separate control lines for each.

iii. Use one common bus for memory and I/O with common control lines.

I/O Processor

In the first method, the computer has independent sets of data, address and control buses one for accessing memory and other for I/O. This is done in computers that provides a separate I/O processor (IOP). The purpose of IOP is to provide an independent pathway for the transfer of information between external device and internal memory.

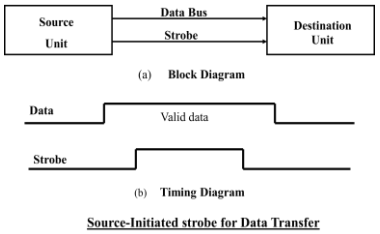
Asynchronous Data Transfer

This Scheme is used when speed of I/O devices do not match with microprocessor, and timing characteristics of I/O devices is not predictable. In this method, process initiates the device and check its status. As a result, CPU has to wait till I/O device is ready to transfer data. When device is ready CPU issues instruction for I/O transfer. In this method two types of techniques are used based on signals before data transfer.i. Strobe Control ii. Handshaking

Strobe Signal

The strobe control method of Asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit.

Data Transfer Initiated by Source Unit

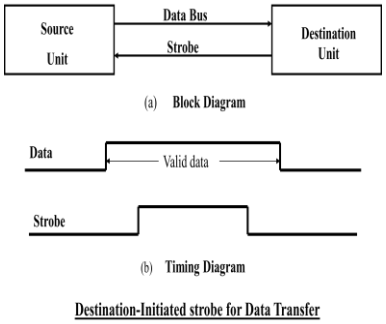


Source-Initiated strobe for Data Transfer

In the block diagram fig. (a), the data bus carries the binary information from source to destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available.

The timing diagram fig. (b) the source unit first places the data on the data bus. The information on the data bus and strobe signal remain in the active state to allow the destination unit to receive the data.

Data Transfer Initiated by Destination Unit



Destination-Initiated strobe for Data Transfer

In this method, the destination unit activates the strobe pulse, to informing the source to provide the data. The source will respond by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it. When accepted the destination unit then disables the strobe and the source unit removes the data from the bus.

Disadvantage of Strobe Signal

The disadvantage of the strobe method is that, the source unit initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was places in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on bus. The **Handshaking method** solves this problem.

Handshaking

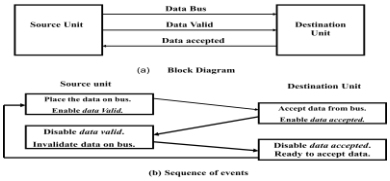
The handshaking method solves the problem of strobe method by introducing a second control signal that provides a reply to the unit that initiates the transfer.

The basic **principle of the two-wire handshaking** method of data transfer is as follow:

One control line is in the same direction as the data flows in the bus from the source to destination. It is used by source unit to inform the destination unit whether there a valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept the data. The sequence of control during the transfer depends on the unit that initiates the transfer.

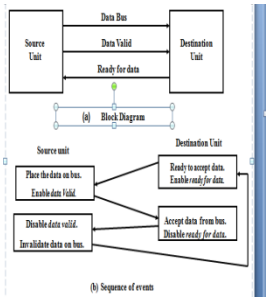
Source Initiated Transfer using Handshaking

The sequence of events shows four possible states that the system can be at any given time. The source unit initiates the transfer by placing the data on the bus and enabling its ***data valid*** signal. The ***data accepted*** signal is activated by the destination unit after it accepts the data from the bus. The source unit then disables its ***data accepted*** signal and the system goes into its initial state.



Destination Initiated Transfer Using Handshaking

The name of the signal generated by the destination unit has been changed to ***ready for data*** to reflects its new meaning. The source unit in this case does not place data on the bus until after it receives the ***ready for data*** signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source initiated case. The only **difference** between the Source Initiated and the Destination Initiated transfer is in their choice of Initial sate.



Advantage of the Handshaking method

- The Handshaking scheme provides degree of flexibility and reliability because the successful completion of data transfer relies on active participation by both units.
- If any of one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a ***Timeout mechanism*** which provides an alarm if the data is not completed within time.

Asynchronous Serial Transmission

The transfer of data between two units is serial or parallel. In parallel data transmission, n bit in the message must be transmitted through n separate conductor path. In serial transmission, each bit in the message is sent in sequence one at a time.

Parallel transmission is faster but it requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive.

In Asynchronous serial transfer, each bit of message is sent a sequence at a time, and binary information is transferred only when it is available. When there is no information to be transferred, line remains idle.

In this technique each character consists of three points :

i. Start bit

ii. Character bit

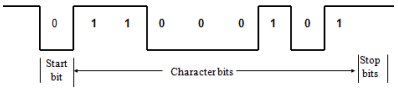
iii. Stop bit

i. **Start Bit-** First bit, called start bit is always zero and used to indicate the beginning character.

ii. **Stop Bit-** Last bit, called stop bit is always one and used to indicate end of characters. Stop bit is always in the 1- state and frame the end of the characters to signify the idle or wait state.

iii. **Character Bit-** Bits in between the start bit and the stop bit are known as character bits. The character bits always follow the start bit.

Asynchronous Serial Transmission



Asynchronous Serial Transmission

Asynchronous Serial Transmission

Serial Transmission of Asynchronous is done by two ways:

a) Asynchronous Communication Interface

b) First In First out Buffer

Asynchronous Communication Interface

It works as both a receiver and a transmitter. Its operation is initialized by CPU by sending a byte to the control register.

The **transmitter register** accepts a data byte from CPU through the data bus and transferred to a shift register for serial transmission.

The **receive portion** receives information into another shift register, and when a complete data byte is received it is transferred to receiver register.

CPU can select the receiver register to read the byte through the data bus. Data in the status register is used for input and output flags.

First In First Out Buffer (FIFO)

A First In First Out (FIFO) Buffer is a memory unit that stores information in such a manner that the first item is in the item first out. A FIFO buffer comes with separate input and output terminals. The important feature of this buffer is that it can input data and output data at two different rates.

When placed between two units, the FIFO can accept data from the source unit at one rate, rate of transfer and deliver the data to the destination unit at another rate.

If the source is faster than the destination, the FIFO is useful for source data arrive in bursts that fills out the buffer. FIFO is useful in some applications when data are transferred asynchronously

Transfer of data is required between CPU and peripherals or memory or sometimes between any two devices or units of your computer system. To transfer a data from one unit to another one should be sure that both units have proper connection and at the time of data transfer the receiving unit is not busy. This data transfer with the computer is **Internal Operation.**

All the internal operations in a digital system are **synchronized** by means of clock pulses supplied by a common **clock pulse Generator.** The data transfer can be

i. Synchronous or

ii. Asynchronous

When both the transmitting and receiving units use same clock pulse then such a data transfer is called **Synchronous process**. On the other hand, if the there is not concept of clock pulses and the sender operates at different moment than the receiver then such a data transfer is called **Asynchronous data transfer**.

Modes of Data Transfer

The data transfer can be handled by various modes. some of the modes use CPU as an intermediate path, others transfer the data directly to and from the memory unit and this can be handled by 3 following ways:

i. Programmed I/O

ii. Interrupt-Initiated I/O

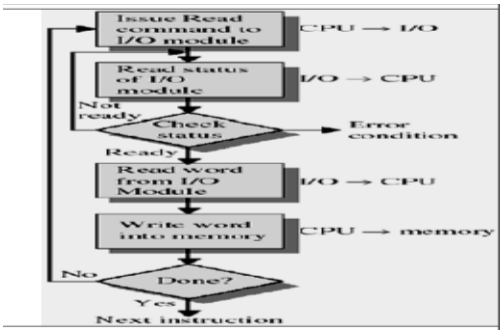
iii. Direct Memory Access (DMA)

Programmed I/O Mode

In this mode of data transfer the operations are the results in I/O instructions which is a part of computer program. Each data transfer is initiated by a instruction in the program. Normally the transfer is from a CPU register to peripheral device or vice-versa.

Once the data is initiated the CPU starts monitoring the interface to see when next transfer can made. The instructions of the program keep close tabs on everything that takes place in the interface unit and the I/O devices.

In this technique CPU is responsible for executing data from the memory for output and storing data in memory for executing of Programmed I/O as shown in Flowchart-:



Drawback of the Programmed I/O

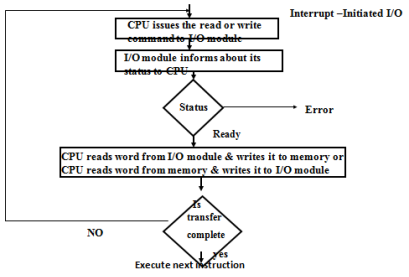
The main drawback of the Program Initiated I/O was that the CPU has to monitor the units all the times when the program is executing. Thus the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process and the CPU time is wasted a lot in keeping an eye to the executing of program. To remove this problem an Interrupt facility and special commands are used.

Interrupt-Initiated I/O

In this method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer. In the meantime the CPU executes other program. When the interface determines that the device is ready for data transfer it generates an **Interrupt Request** and sends it to the computer.

When the CPU receives such an signal, it temporarily stops the execution of the program and branches to a service program to process the I/O transfer and after completing it returns back to task, what it was originally performing.

The Execution process of Interrupt–Initiated I/O is represented in the flowchart:



Direct Memory Access (DMA)

In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called **Direct Memory Access (DMA).**

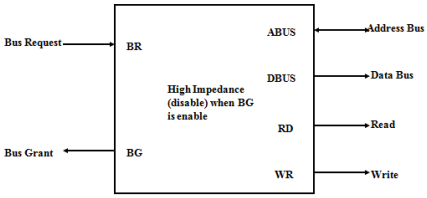
During the **DMA transfer**, the CPU is idle and has no control of the memory buses. A **DMA Controller** takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:

- **Bus Request (BR)**
- **Bus Grant (BG)**

These two control signals in the CPU that facilitates the DMA transfer. The ***Bus Request (BR)*** input is used by the ***DMA controller*** to request the CPU. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus and read write lines into a ***high Impedance state.*** **High Impedance state means that the output is disconnected.**

Direct Memory Access (DMA)



CPU bus Signals for DMA Transfer

The CPU activates the ***Bus Grant (BG)*** output to inform the external DMA that the Bus Request (BR) can now take control of the buses to conduct memory transfer without processor.

When the DMA terminates the transfer, it disables the ***Bus Request (BR)*** line. The CPU disables the ***Bus Grant (BG)***, takes control of the buses and return to its normal operation.

The transfer can be made in several ways that are:

i. DMA Burst

ii. Cycle Stealing

i) DMA Burst :- In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.

ii) Cycle Stealing :- Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must returns control of the buses to the CPU.

iii) The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. The DMA controller has three registers:

i. Address Register

ii. Word Count Register

iii. Control Register

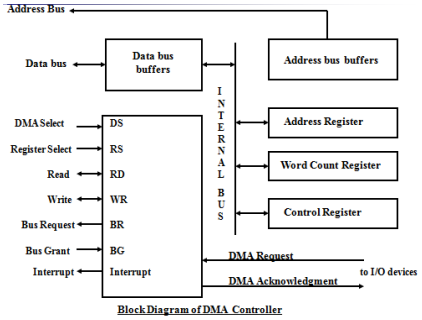
i. Address Register :- Address Register contains an address to specify the desired location in memory.

ii. Word Count Register :- WC holds the number of words to be transferred. The register is incre/decre by one after each word transfer and internally tested for zero.

iii. Control Register :- Control Register specifies the mode of transfer.

The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the **DS (DMA select)** and **RS (Register select)** inputs. The **RD (read)** and **WR (write)** inputs are bidirectional.

When the **BG (Bus Grant) input is 0,** the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When **BG =1**, the DMA can communicate directly with the memory by specifying an address in the address bus and activating the **RD or WR** control.

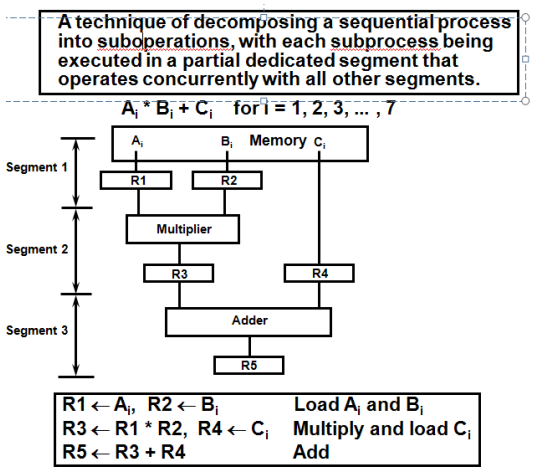


Block Diagram of DMA Controller

The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can transfer between the peripheral and the memory.

When **BG = 0** the **RD and WR are input lines allowing the CPU to communicate** with the internal DMA registers. When **BG=1**, the **RD and WR are output lines from the DMA controller** to the random access memory to specify the read or write operation of data.

PIPELINING



OPERATIONS IN EACH PIPELINE STAGE

Clock Pulse Number	R1 Segment 1	R2 Segment 1	R3 Segment 2	R4 Segment 2	R5 Segment 3
1	A1	B1			
2	A2	B2	A1 * B1	C1	
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8			A7 * B7	C7	A6 * B6 + C6
9					A7 * B7 + C7

General Structure of a 4-Segment Pipeline

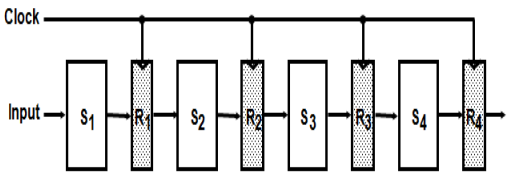


Figure 9-2 Example of pipeline processing.

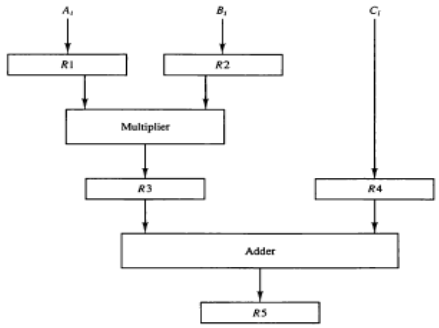
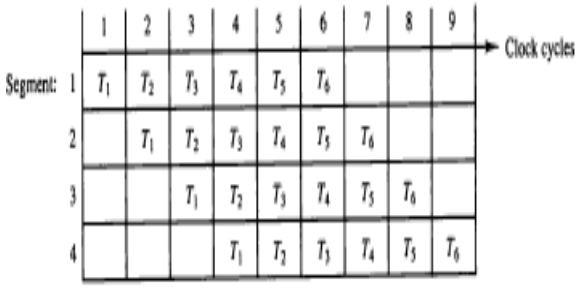


TABLE 9-1 Content of Registers in Pipeline Example

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	—	—	—	—	$A_7 * B_7 + C_7$

Space-Time Diagram



PIPELINE SPEEDUP

n: Number of tasks to be performed

Conventional Machine (Non-Pipelined)

t_n: Clock cycle

t_1: Time required to complete the n tasks

$t_1 = n * t_n$

Pipelined Machine (k stages)

t_p: Clock cycle (time to complete each suboperation)

t_k: Time required to complete the n tasks

$t_k = (k + n - 1) * t_p$

Speedup

S_k: Speedup

$$S_k = n*t_n / (k + n - 1)*t_p$$

$$\lim_{n \to \infty} S_k = \frac{t_n}{t_p} \quad (= k, \text{ if } t_n = k * t_p)$$

There are **two areas of computer design where the pipeline organization is applicable**.

1)Arithmetic pipeline divides the arithmetic operation into sub operations for execution in the pipeline segments.

2)An instruction pipe line operates on a stream of instructions by overlapping the fetch, decode and execute phases of the instruction cycle

ARITHMETIC PIPELINE

Floating-point adder pipeline:

$X = A \times 2^a$

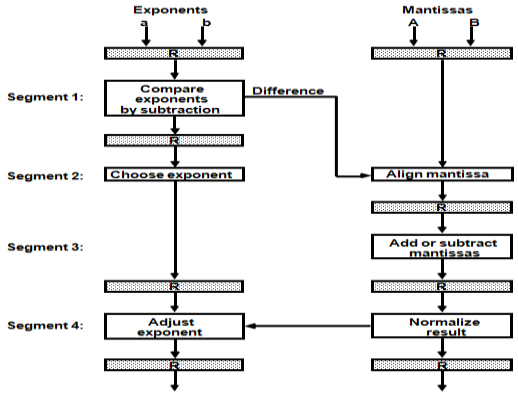
$Y = B \times 2^b$

[1] Compare the exponents

[2] Align the mantissa

[3] Add/sub the mantissa

[4] Normalize the result



INSTRUCTION CYCLE

Six Phases in an Instruction Cycle

1] Fetch an instruction from memory(FI)

[2] Decode the instruction

[3] Calculate the effective address of the operand (DA)

[4] Fetch the operands from memory (FO)

[5] Execute the operation

[6] Store the result in the proper place (EX)

4-Stage Pipeline

[1] FI: Fetch an instruction from memory

[2] DA: Decode the instruction and calculate the effective address of the operand

[3] FO: Fetch the operand

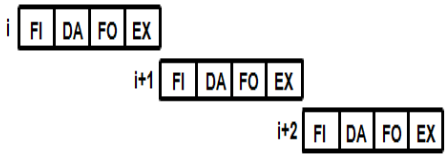
[4] EX: Execute the operation

<u>Execution of Three Instructions in a 4-Stage Pipeline</u>

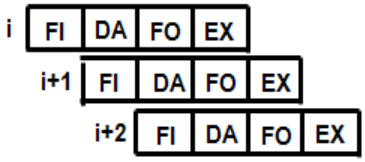
12 Clock pulses without pipeline.

6 Clock pulses with pipeline

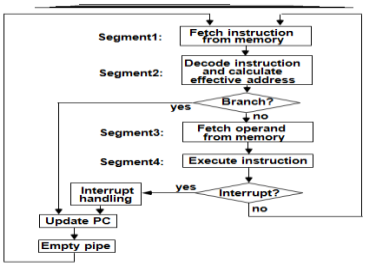
Conventional

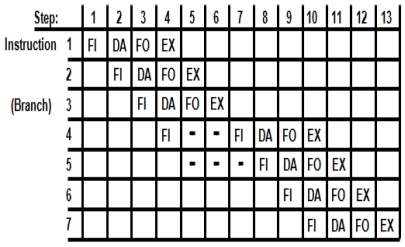


Pipelined



INSTRUCTION EXECUTION IN A 4-STAGE PIPELINE





MAJOR HAZARDS IN PIPELINED EXECUTION

Three major difficulties that cause the instruction pipeline to deviate from its normal operation:

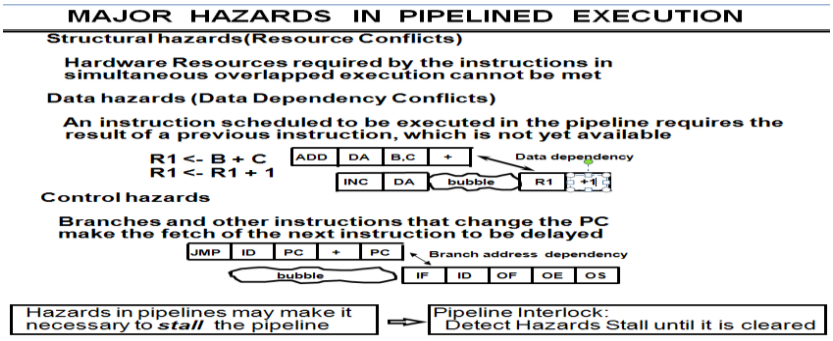
1) <u>Resource conflicts: (Structural hazards)</u> two segments access memory at the same time.

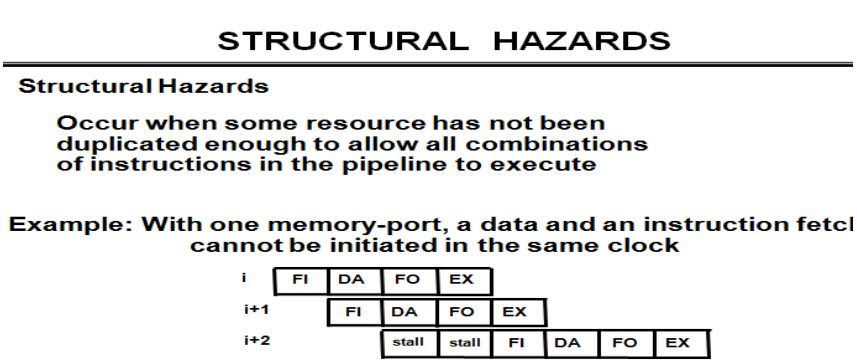
This can be resolved by using separate instruction and data memories.

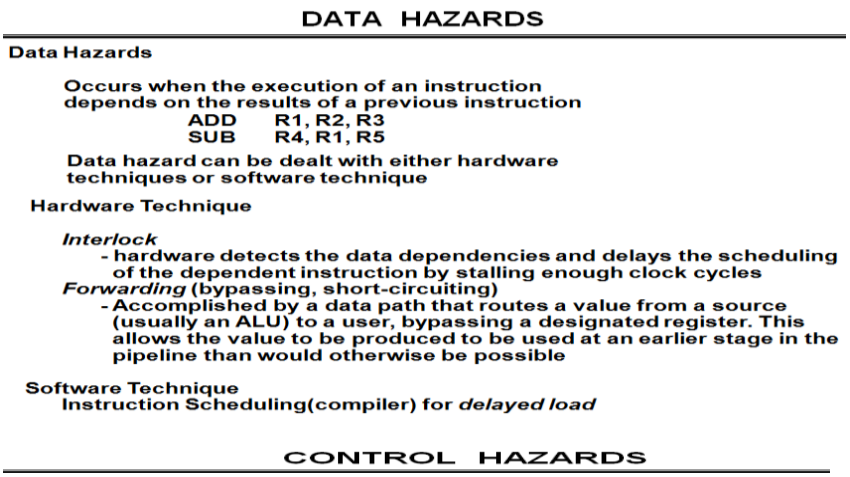
2) Data dependency(Data hazards): **An instruction scheduled to be executed in the pipeline requires the result of a previous instruction, which is not yet available.**

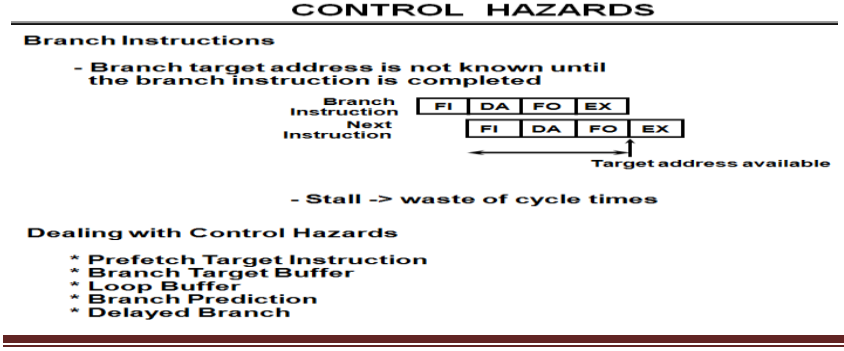
R1 <- B + C R1 <- R1 + 1

3) Branch difficulties(Control hazards): **branch and other instructions change the value of PC.**









CONTROL HAZARDS

Prefetch Target Instruction

- Fetch instructions in both streams, branch not taken and branch taker
- Both are saved until branch branch is executed. Then, select the right instruction stream and discard the wrong stream

Branch Target Buffer(BTB; Associative Memory)

- Entry: Addr of previously executed branches; Target instruction and the next few instructions
- When fetching an instruction, search BTB.
- If found, fetch the instruction stream in BTB;
- If not, new stream is fetched and update BTB

Loop Buffer(High Speed Register file)

- Storage of entire loop that allows to execute a loop without accessing memory

Branch Prediction

- Guessing the branch condition, and fetch an instruction stream based on the guess. Correct guess eliminates the branch penalty

Delayed Branch

- Compiler detects the branch and rearranges the instruction sequence by inserting useful instructions that keep the pipeline busy in the presence of a branch instruction

RISC PIPELINE

RISC

- Machine with a very fast clock cycle that executes at the rate of one instruction per cycle
- Simple Instruction Set
  - Fixed Length Instruction Format
  - Register-to-Register Operations

Instruction Cycles of Three-Stage Instruction Pipeline

Data Manipulation Instructions

I: Instruction Fetch
A: Decode, Read Registers, ALU Operations
E: Write a Register

Load and Store Instructions

I: Instruction Fetch
A: Decode, Evaluate Effective Address
E: Register-to-Memory or Memory-to-Register

Program Control Instructions

I: Instruction Fetch
A: Decode, Evaluate Branch Address
E: Write Register(PC)

DELAYED LOAD

LOAD: R1 ← M[address 1]
LOAD: R2 ← M[address 2]
ADD: R3 ← R1 + R2
STORE: M[address 3] ← R3

Three-segment pipeline timing

Pipeline timing with data conflict

clock cycle	1	2	3	4	5	6
Load R1	I	A	E			
Load R2		I	A	E		
Add R1+R2			I	A	E	
Store R3				I	A	E

Pipeline timing with delayed load

clock cycle	1	2	3	4	5	6	7
Load R1	I	A	E				
Load R2		I	A	E			
NOP			I	A	E		
Add R1+R2				I	A	E	
Store R3					I	A	E

The data dependency is taken care by the compiler rather than the hardware

DELAYED BRANCH

Compiler analyzes the instructions before and after the branch and rearranges the program sequence by inserting useful instructions in the delay steps

Using no-operation instructions

Clock cycles:	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. NOP						I	A	E		
7. NOP							I	A	E	
8. Instr. in X								I	A	E

Rearranging the instructions

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to X			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instr. in X						I	A	E

Reference

1.‘Computer System Architecture’, Morris M. Mano, 3rd edition,Prentice Hall India.

2. Computer Organization and Achitecture, William Stallings ,8th edition,PHI

3. Computer Organization, Carl Hamachar, Vranesic, McGraw Hill.