UNIT 2 CLASSIFICATION OF PARALLEL COMPUTERS

Structure **Page Nos.**

2.0 INTRODUCTION

Parallel computers are those that emphasize the parallel processing between the operations in some way. In the previous unit, all the basic terms of parallel processing and computation have been defined. Parallel computers can be characterized based on the data and instruction streams forming various types of computer organisations. They can also be classified based on the computer structure, e.g. multiple processors having separate memory or one shared global memory. Parallel processing levels can also be defined based on the size of instructions in a program called grain size. Thus, parallel computers can be classified based on various criteria. This unit discusses all types of classification of parallel computers based on the above mentioned criteria.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- explain the various criteria on which classification of parallel computers are based;
- discuss the Flynn's classification based on instruction and data streams;
- describe the Structural classification based on different computer organisations;
- explain the Handler's classification based on three distinct levels of computer: Processor control unit (PCU), Arithmetic logic unit (ALU), Bit-level circuit (BLC), and
- describe the sub-tasks or instructions of a program that can be executed in parallel based on the grain size.

2.2 TYPES OF CLASSIFICATION

The following classification of parallel computers have been identified:

1) Classification based on the instruction and data streams
2) Classification based on the structure of computers
3) Classification based on how the memory is accessed
4) Classification based on grain size

All these classification schemes are discussed in subsequent sections.

2.3 FLYNN'S CLASSIFICATION

This classification was first studied and proposed by Michael Flynn in 1972. Flynn did not consider the machine architecture for classification of parallel computers; he introduced the concept of *instruction* and *data* streams for categorizing of computers. All the computers classified by Flynn are not parallel computers, but to grasp the concept of parallel computers, it is necessary to understand all types of Flynn's classification. Since, this classification is based on instruction and data streams, first we need to understand how the instruction cycle works.

2.3.1 Instruction Cycle

The instruction cycle consists of a sequence of steps needed for the execution of an instruction in a program. A typical instruction in a program is composed of two parts: Opcode and Operand. The Operand part specifies the data on which the specified operation is to be done. (See *Figure 1*). The Operand part is divided into two parts: addressing mode and the Operand. The addressing mode specifies the method of determining the addresses of the actual data on which the operation is to be performed and the operand part is used as an argument by the method in determining the actual address.

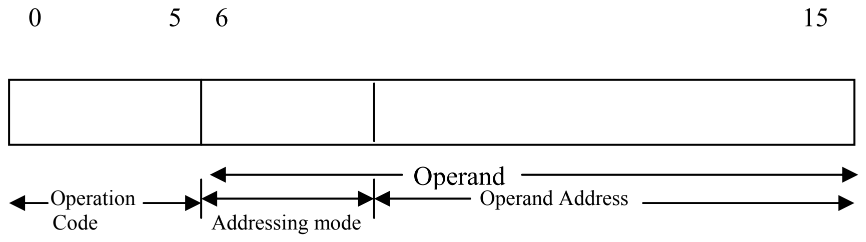

Figure 1: Opcode and Operand

The control unit of the CPU of the computer fetches instructions in the program, one at a time. The fetched Instruction is then decoded by the decoder which is a part of the control unit and the processor executes the decoded instructions. The result of execution is temporarily stored in Memory Buffer Register (MBR) (also called Memory Data Register). The normal execution steps are shown in *Figure 2*.

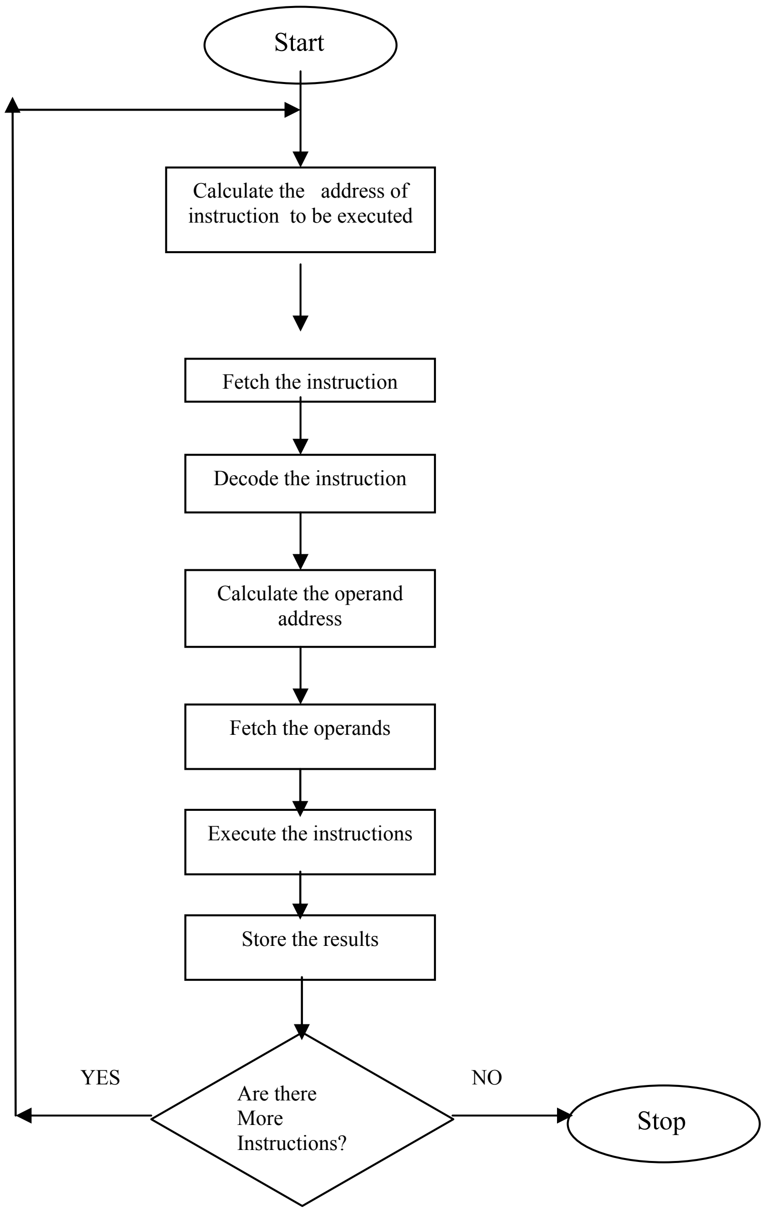

Figure 2: Instruction execution steps

2.3.2 Instruction Stream and Data Stream

The term 'stream' refers to a sequence or flow of either instructions or data operated on by the computer. In the complete cycle of instruction execution, a flow of instructions from main memory to the CPU is established. This flow of instructions is called ***instruction stream.*** Similarly, there is a flow of operands between processor and memory bi-directionally. This flow of operands is called ***data stream***. These two types of streams are shown in *Figure 3*.

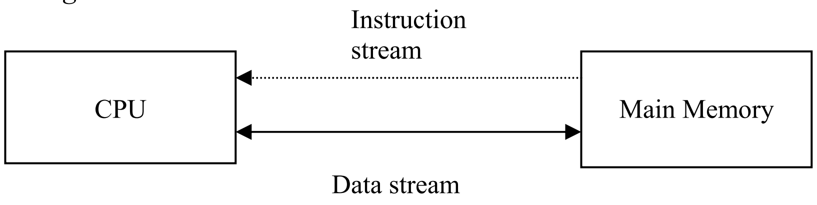

Figure 3: Instruction and data stream

Thus, it can be said that the sequence of instructions executed by CPU forms the Instruction streams and sequence of data (operands) required for execution of instructions form the Data streams.

2.3.3 Flynn's Classification

Flynn's classification is based on multiplicity of instruction streams and data streams observed by the CPU during program execution. Let I_s and D_s are minimum number of streams flowing at any point in the execution, then the computer organisation can be categorized as follows:

1) **Single Instruction and Single Data stream (SISD)**

In this organisation, sequential execution of instructions is performed by one CPU containing a single processing element (PE), i.e., ALU under one control unit as shown in *Figure 4*. Therefore, SISD machines are conventional serial computers that process only one stream of instructions and one stream of data. This type of computer organisation is depicted in the diagram:

$I_s = D_s = 1$

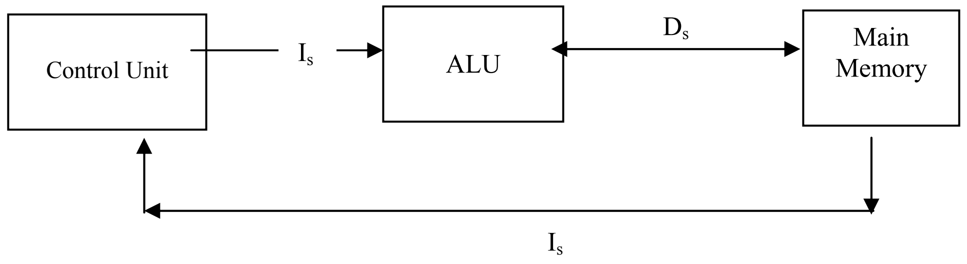

Figure 4: SISD Organisation

Examples of SISD machines include:

- CDC 6600 which is unpipelined but has multiple functional units.
- CDC 7600 which has a pipelined arithmetic unit.
- Amdhal 470/6 which has pipelined instruction processing.
- Cray-1 which supports vector processing.

2) **Single Instruction and Multiple Data stream (SIMD)**

In this organisation, multiple processing elements work under the control of a single control unit. It has one instruction and multiple data stream. All the processing elements of this organization receive the same instruction broadcast from the CU. Main memory can also be divided into modules for generating multiple data streams acting as a *distributed memory* as shown in *Figure 5*. Therefore, all the processing elements simultaneously execute the same instruction and are said to be 'lock-stepped' together. Each processor takes the data from its own memory and hence it has on distinct data streams. (Some systems also provide a shared global memory for communications.) Every processor must be allowed to complete its instruction before the next instruction is taken for execution. Thus, the execution of instructions is synchronous. Examples of SIMD organisation are ILLIAC-IV, PEPE, BSP, STARAN, MPP, DAP and the Connection Machine (CM-1).

This type of computer organisation is denoted as:

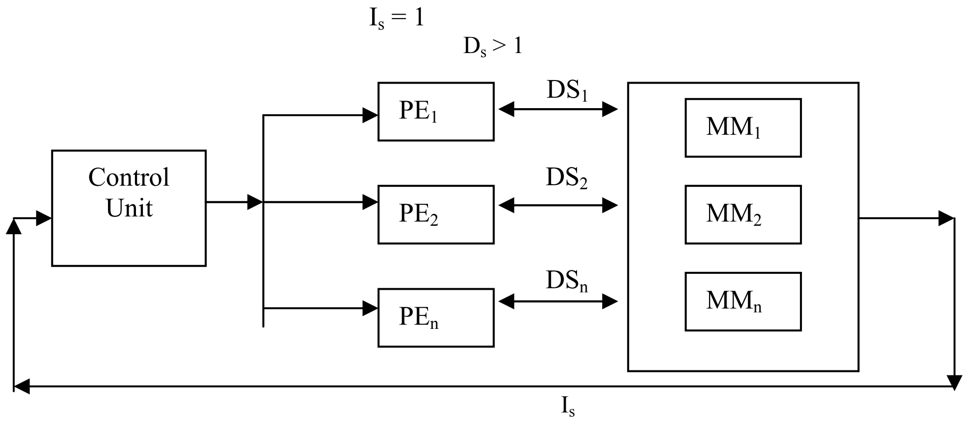

Figure 5: SIMD Organisation

3) **Multiple Instruction and Single Data stream (MISD)**

In this organization, multiple processing elements are organised under the control of multiple control units. Each control unit is handling one instruction stream and processed through its corresponding processing element. But each processing element is processing only a single data stream at a time. Therefore, for handling multiple instruction streams and single data stream, multiple control units and multiple processing elements are organised in this classification. All processing elements are interacting with the common shared memory for the organisation of single data stream as shown in *Figure 6*. The only known example of a computer capable of MISD operation is the C.mmp built by Carnegie-Mellon University.
This type of computer organisation is denoted as:

$I_s > 1$
$D_s = 1$

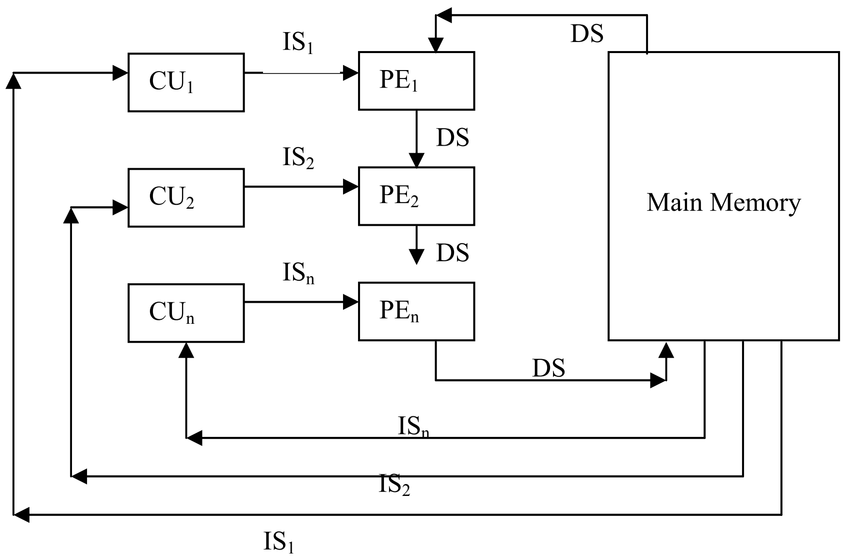

Figure 6: MISD Organisation

This classification is not popular in commercial machines as the concept of single data streams executing on multiple processors is rarely applied. But for the specialized applications, MISD organisation can be very helpful. For example, Real time computers need to be fault tolerant where several processors execute the same data for producing the redundant data. This is also known as N- version programming. All these redundant data are compared as results which should be same; otherwise faulty unit is replaced. Thus MISD machines can be applied to fault tolerant real time computers.

4) **Multiple Instruction and Multiple Data stream (MIMD)**

In this organization, multiple processing elements and multiple control units are organized as in MISD. But the difference is that now in this organization multiple instruction streams operate on multiple data streams . Therefore, for handling multiple instruction streams, multiple control units and multiple processing elements are organized such that multiple processing elements are handling multiple data streams from the Main memory as shown in *Figure 7*. The processors work on their own data with their own instructions. Tasks executed by different processors can start or finish at different times. They are not lock-stepped, as in SIMD computers, but run asynchronously. This classification actually recognizes the parallel computer. That means in the real sense MIMD organisation is said to be a Parallel computer. All multiprocessor systems fall under this classification. Examples include; C.mmp, Burroughs D825, Cray-2, S1, Cray X-MP, HEP, Pluribus, IBM 370/168 MP, Univac 1100/80, Tandem/16, IBM 3081/3084, C.m*, BBN Butterfly, Meiko Computing Surface (CS-1), FPS T/40000, iPSC.

This type of computer organisation is denoted as:

$I_s > 1$
$D_s > 1$

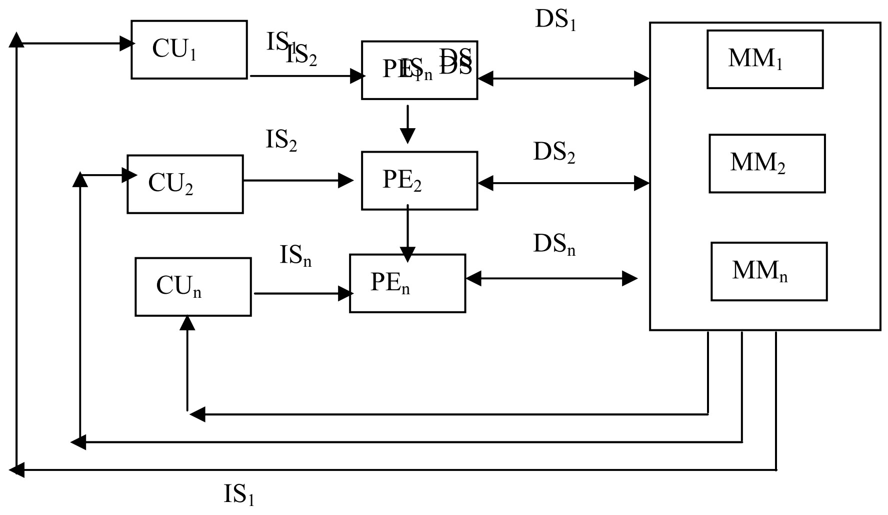

Figure 7: MIMD Organisation

Of the classifications discussed above, MIMD organization is the most popular for a parallel computer. In the real sense, parallel computers execute the instructions in MIMD mode.

Check Your Progress 1

1) What are various criteria for classification of parallel computers?
…………………………………………………………………………………………
…………………………………………………………………………………………
…………………………………………………………………………………………
…………………………………………………………………………………………

2) Define instruction and data streams.
…………………………………………………………………………………………
…………………………………………………………………………………………

…………………………………………………………………………………………
…………………………………………………………………………………………

3) State whether True or False for the following:
   a) SISD computers can be characterized as $I_s > 1$ and $D_s > 1$
   b) SIMD computers can be characterized as $I_s > 1$ and $D_s = 1$
   c) MISD computers can be characterized as $I_s = 1$ and $D_s = 1$
   d) MIMD computers can be characterized as $I_s > 1$ and $D_s > 1$

2.4 HANDLER'S CLASSIFICATION

In 1977, Wolfgang Handler proposed an elaborate notation for expressing the pipelining and parallelism of computers. Handler's classification addresses the computer at three distinct levels:

- Processor control unit (PCU),
- Arithmetic logic unit (ALU),
- Bit-level circuit (BLC).

The PCU corresponds to a processor or CPU, the ALU corresponds to a functional unit or a processing element and the BLC corresponds to the logic circuit needed to perform one-bit operations in the ALU.

Handler's classification uses the following three pairs of integers to describe a computer:

Computer = (p * p', a * a', b * b')

Where p = number of PCUs
Where p'= number of PCUs that can be pipelined
Where a = number of ALUs controlled by each PCU
Where a'= number of ALUs that can be pipelined
Where b = number of bits in ALU or processing element (PE) word
Where b'= number of pipeline segments on all ALUs or in a single PE

The following rules and operators are used to show the relationship between various elements of the computer:

- The '*' operator is used to indicate that the units are pipelined or macro-pipelined with a stream of data running through all the units.
- The '+' operator is used to indicate that the units are not pipelined but work on independent streams of data.
- The 'v' operator is used to indicate that the computer hardware can work in one of several modes.
- The '~' symbol is used to indicate a range of values for any one of the parameters.
- Peripheral processors are shown before the main processor using another three pairs of integers. If the value of the second element of any pair is 1, it may omitted for brevity.

Handler's classification is best explained by showing how the rules and operators are used to classify several machines.

The CDC 6600 has a single main processor supported by 10 I/O processors. One control unit coordinates one ALU with a 60-bit word length. The ALU has 10 functional units which can be formed into a pipeline. The 10 peripheral I/O processors may work in parallel with each other and with the CPU. Each I/O processor contains one 12-bit ALU. The description for the 10 I/O processors is:

CDC 6600I/O = (10, 1, 12)

The description for the main processor is:

CDC 6600main = (1, 1 * 10, 60)

The main processor and the I/O processors can be regarded as forming a macro-pipeline so the '*' operator is used to combine the two structures:

CDC 6600 = (I/O processors) * (central processor = (10, 1, 12) * (1, 1 * 10, 60)

Texas Instrument's Advanced Scientific Computer (ASC) has one controller coordinating four arithmetic units. Each arithmetic unit is an eight stage pipeline with 64-bit words. Thus we have:

ASC = (1, 4, 64 * 8)

The Cray-1 is a 64-bit single processor computer whose ALU has twelve functional units, eight of which can be chained together to from a pipeline. Different functional units have from 1 to 14 segments, which can also be pipelined. Handler's description of the Cray-1 is:

Cray-1 = (1, 12 * 8, 64 * (1 ~ 14))

Another sample system is Carnegie-Mellon University's C.mmp multiprocessor. This system was designed to facilitate research into parallel computer architectures and consequently can be extensively reconfigured. The system consists of 16 PDP-11 'minicomputers' (which have a 16-bit word length), interconnected by a crossbar switching network. Normally, the C.mmp operates in MIMD mode for which the description is (16, 1, 16). It can also operate in SIMD mode, where all the processors are coordinated by a single master controller. The SIMD mode description is (1, 16, 16). Finally, the system can be rearranged to operate in MISD mode. Here the processors are arranged in a chain with a single stream of data passing through all of them. The MISD modes description is (1 * 16, 1, 16). The 'v' operator is used to combine descriptions of the same piece of hardware operating in differing modes. Thus, Handler's description for the complete C.mmp is:

C.mmp = (16, 1, 16) v (1, 16, 16) v (1 * 16, 1, 16)

The '*' and '+' operators are used to combine several separate pieces of hardware. The 'v' operator is of a different form to the other two in that it is used to combine the different operating modes of a single piece of hardware.

While Flynn's classification is easy to use, Handler's classification is cumbersome. The direct use of numbers in the nomenclature of Handler's classification's makes it much more abstract and hence difficult. Handler's classification is highly geared towards the description of pipelines and chains. While it is well able to describe the parallelism in a single processor, the variety of parallelism in multiprocessor computers is not addressed well.

2.5 STRUCTURAL CLASSIFICATION

Flynn's classification discusses the behavioural concept and does not take into consideration the computer's structure. Parallel computers can be classified based on their structure also, which is discussed below and shown in *Figure 8*.

As we have seen, a parallel computer (MIMD) can be characterised as a set of multiple processors and shared memory or memory modules communicating via an interconnection network. When multiprocessors communicate through the global shared memory modules then this organisation is called ***Shared memory computer*** or ***Tightly***

coupled systems as shown in *Figure 9.* Similarly when every processor in a multiprocessor system, has its own local memory and the processors communicate via messages transmitted between their local memories, then this organisation is called ***Distributed memory computer*** or ***Loosely coupled system*** as shown in *Figure 10. Figures 9* and *10* show the simplified diagrams of both organisations.

The processors and memory in both organisations are interconnected via an interconnection network. This interconnection network may be in different forms like crossbar switch, multistage network, etc. which will be discussed in the next unit.

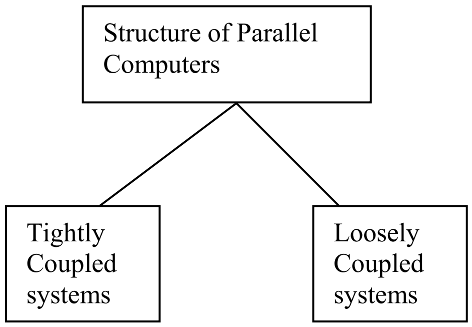

Figure 8: Structural classification

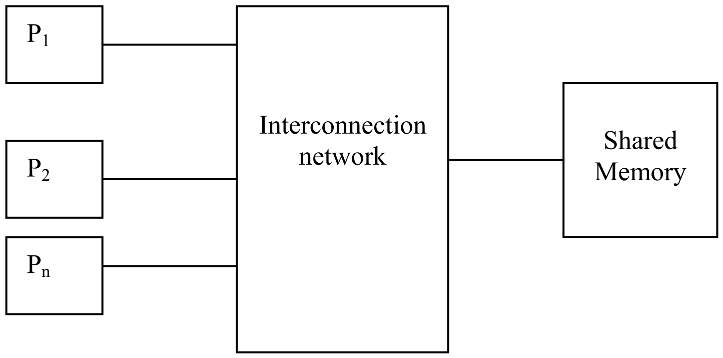

Figure 9: Tightly coupled system

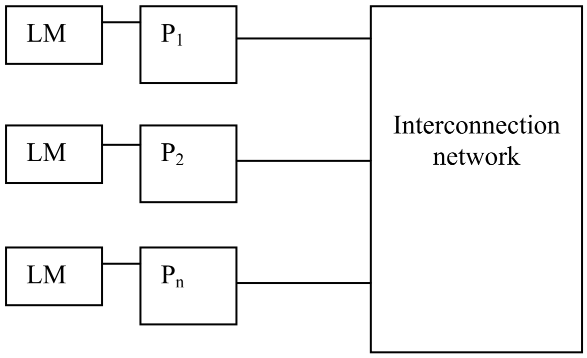

Figure 10 Loosely coupled system

2.5.1 Shared Memory System / Tightly Coupled System

Shared memory multiprocessors have the following characteristics:

- Every processor communicates through a shared global memory.

- For high speed real time processing, these systems are preferable as their throughput is high as compared to loosely coupled systems.

In tightly coupled system organization, multiple processors share a global main memory, which may have many modules as shown in detailed *Figure 11*. The processors have also access to I/O devices. The inter- communication between processors, memory, and other devices are implemented through various interconnection networks, which are discussed below.

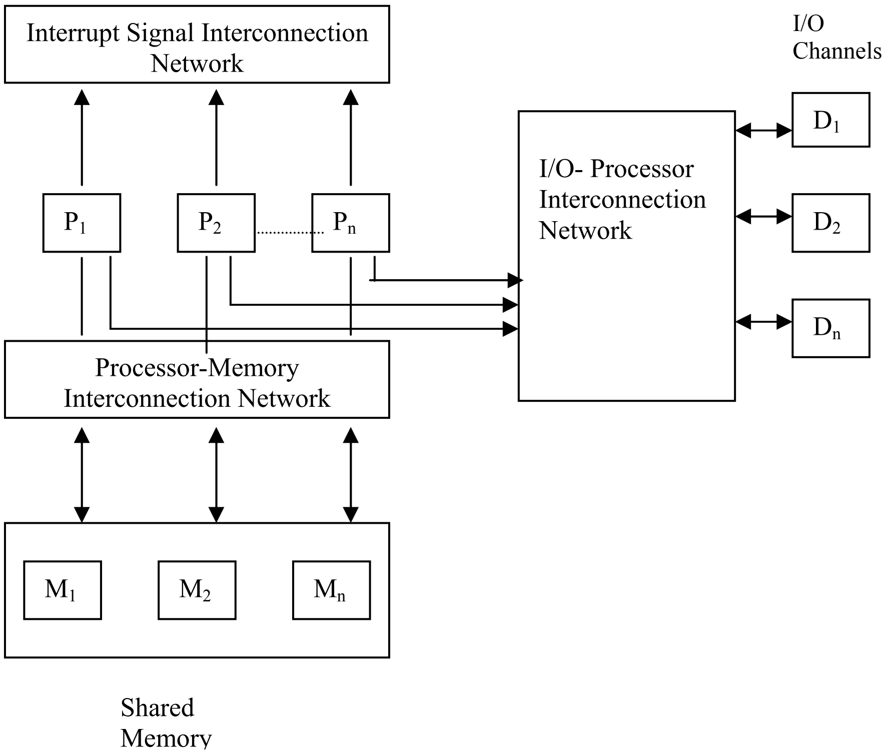
Interrupt Signal Interconnection Network
I/O Channels
P_1
P_2
P_n
I/O- Processor Interconnection Network
D_1
D_2
D_n
Processor-Memory Interconnection Network
M_1
M_2
M_n
Shared Memory

Figure 11: Tightly coupled system organization

i) **Processor-Memory Interconnection Network (PMIN)**

This is a switch that connects various processors to different memory modules. Connecting every processor to every memory module in a single stage while the crossbar switch may become complex. Therefore, multistage network can be adopted. There can be a conflict among processors such that they attempt to access the same memory modules. This conflict is also resolved by PMIN.

ii) **Input-Output-Processor Interconnection Network (IOPIN)**

This interconnection network is used for communication between processors and I/O channels. All processors communicate with an I/O channel to interact with an I/O device with the prior permission of IOPIN.

iii) **Interrupt Signal Interconnection Network (ISIN)**

When a processor wants to send an interruption to another processor, then this interrupt first goes to ISIN, through which it is passed to the destination processor. In this way, synchronisation between processor is implemented by ISIN. Moreover, in case of failure of one processor, ISIN can broadcast the message to other processors about its failure.

Since, every reference to the memory in tightly coupled systems is via interconnection network, there is a delay in executing the instructions. To reduce this delay, every processor may use cache memory for the frequent references made by the processor as shown in *Figure 12*.

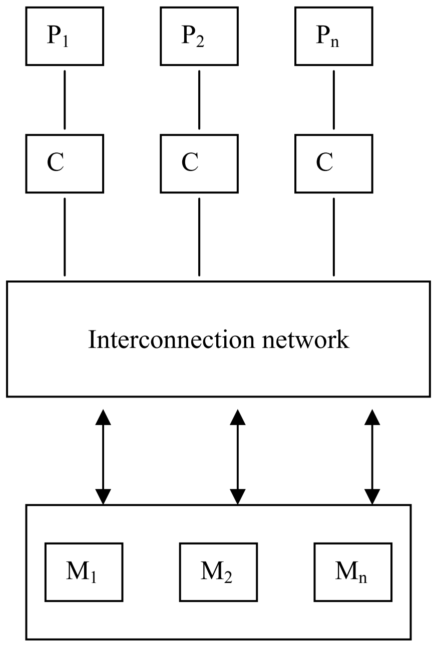

Figure 12: Tightly coupled systems with cache memory

The shared memory multiprocessor systems can further be divided into three modes which are based on the manner in which shared memory is accessed. These modes are shown in *Figure 13* and are discussed below.

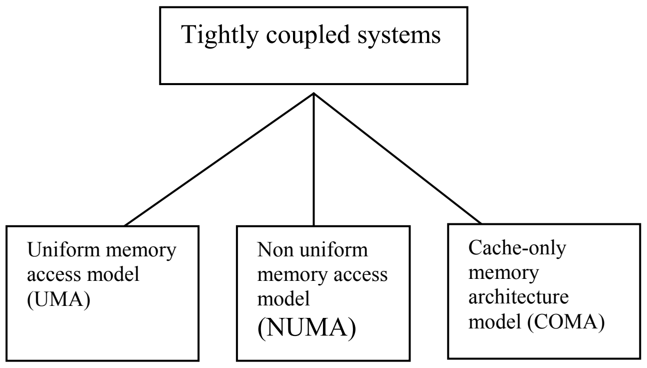
Tightly coupled systems
Uniform memory access model (UMA)
Non uniform memory access model (NUMA)
Cache-only memory architecture model (COMA)

Figure 13: Modes of Tightly coupled systems

2.5.1.1 Uniform Memory Access Model (UMA)

In this model, main memory is uniformly shared by all processors in multiprocessor systems and each processor has equal access time to shared memory. This model is used for time-sharing applications in a multi user environment.

2.5.1.2 Non-Uniform Memory Access Model (NUMA)

In shared memory multiprocessor systems, local memories can be connected with every processor. The collection of all local memories form the global memory being shared. In this way, global memory is distributed to all the processors. In this case, the access to a local memory is uniform for its corresponding processor as it is attached to the local memory. But if one reference is to the local memory of some other remote processor, then the access is not uniform. It depends on the location of the memory. Thus, all memory words are not accessed uniformly.

2.5.1.3 Cache-Only Memory Access Model (COMA)

As we have discussed earlier, shared memory multiprocessor systems may use cache memories with every processor for reducing the execution time of an instruction. Thus in NUMA model, if we use cache memories instead of local memories, then it becomes COMA model. The collection of cache memories form a global memory space. The remote cache access is also non-uniform in this model.

2.5.2 Loosely Coupled Systems

These systems do not share the global memory because shared memory concept gives rise to the problem of memory conflicts, which in turn slows down the execution of instructions. Therefore, to alleviate this problem, each processor in loosely coupled systems is having a large local memory (LM), which is not shared by any other processor. Thus, such systems have multiple processors with their own local memory and a set of I/O devices. This set of processor, memory and I/O devices makes a computer system. Therefore, these systems are also called multi-computer systems. These computer systems are connected together via message passing interconnection network through which processes communicate by passing messages to one another. Since every computer system or node in multicomputer systems has a separate memory, they are called distributed multicomputer systems. These are also called loosely coupled systems, meaning that nodes have little coupling between them as shown in *Figure 14*.

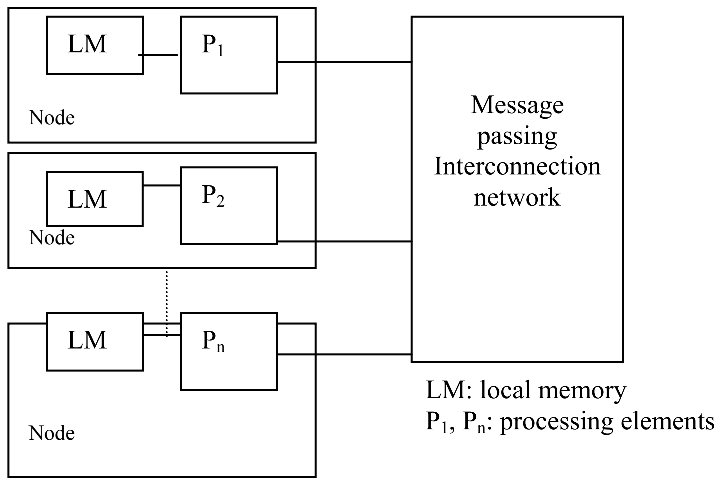
LM
P_1
Node
LM
P_2
Node
LM
P_n
Node
Message passing Interconnection network
LM: local memory
P_1, P_n: processing elements

Figure 14: Loosely coupled system organisation

Since local memories are accessible to the attached processor only, no processor can access remote memory. Therefore, these systems are also known as no-remote memory access (NORMA) systems. Message passing interconnection network provides connection to every node and inter-node communication with message depends on the type of interconnection network. For example, interconnection network for a non-hierarchical system can be shared bus.

Check Your Progress 2

1) What are the various rules and operators used in Handler's classification for various machine types?

……………………………………………………………………………………………..
……………………………………………………………………………………………
……………………………………………………………………………………………
..………………………………………………………………………………………….

2) What is the base for structural classification of parallel computers?
………………………………………………………………………………………..
………………………………………………………………………………………..
………………………………………………………………………………………..
………………………………………………………………………………………..

3) Define loosely coupled systems and tightly coupled systems.
………………………………………………………………………………………
………………………………………………………………………………………
..……………………………………………………………………………………..
…..…………………………………………………………………………………..

4) Differentiate between UMA, NUMA and COMA.
………………………………………………………………………………………..
………………………………………………………………………………………..
………………………………………………………………………………………..
………………………………………………………………………………………..

2.6 CLASSIFICATION BASED ON GRAIN SIZE

This classification is based on recognizing the parallelism in a program to be executed on a multiprocessor system. The idea is to identify the sub-tasks or instructions in a program that can be executed in parallel. For example, there are 3 statements in a program and statements S1 and S2 can be exchanged. That means, these are not sequential as shown in *Figure 15*. Then S1 and S2 can be executed in parallel.

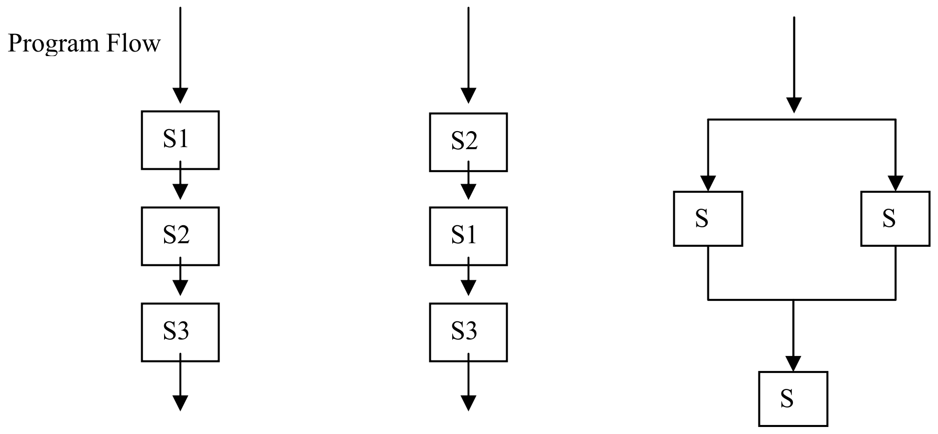

Figure 15: Parallel execution for S1 and S2

But it is not sufficient to check for the parallelism between statements or processes in a program. The decision of parallelism also depends on the following factors:

- Number and types of processors available, i.e., architectural features of host computer
- Memory organisation
- Dependency of data, control and resources

2.6.1 Parallelism Conditions

As discussed above, parallel computing requires that the segments to be executed in parallel must be independent of each other. So, before executing parallelism, all the conditions of parallelism between the segments must be analyzed. In this section, we discuss three types of dependency conditions between the segments
(shown in *Figure 16*).

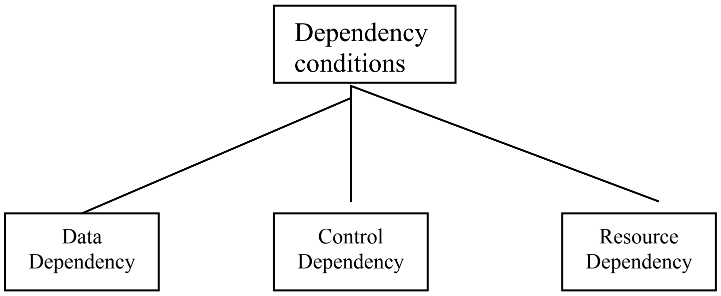

Figure 16: Dependency relations among the segments for parallelism

Data Dependency: It refers to the situation in which two or more instructions share same data. The instructions in a program can be arranged based on the relationship of data dependency; this means how two instructions or segments are data dependent on each other. The following types of data dependencies are recognised:

i) **Flow Dependence :** If instruction I_2 follows I_1 and output of I_1 becomes input of I_2, then I_2 is said to be flow dependent on I_1.

ii) **Antidependence :** When instruction I_2 follows I_1 such that output of I_2 overlaps with the input of I_1 on the same data.

iii) **Output dependence :** When output of the two instructions I_1 and I_2 overlap on the same data, the instructions are said to be output dependent.

iv) **I/O dependence :** When read and write operations by two instructions are invoked on the same file, it is a situation of I/O dependence.

Consider the following program instructions:

I_1: a = b
I_2: c = a + d
I_3: a = c

In this program segment instructions I_1 and I_2 are Flow dependent because variable a is generated by I_1 as output and used by I_2 as input. Instructions I_2 and I_3 are Antidependent because variable a is generated by I_3 but used by I_2 and in sequence I_2 comes first. I_3 is flow dependent on I_2 because of variable c. Instructions I_3 and I_1 are Output dependent because variable a is generated by both instructions.

Control Dependence: Instructions or segments in a program may contain control structures. Therefore, dependency among the statements can be in control structures also. But the order of execution in control structures is not known before the run time. Thus, control structures dependency among the instructions must be analyzed carefully. For example, the successive iterations in the following control structure are dependent on one another.

```
For ( i= 1; I<= n ; i++)
{
        if (x[i - 1] == 0)
                x[i] =0
        else
                x[i] = 1;
}
```

Resource Dependence : The parallelism between the instructions may also be affected due to the shared resources. If two instructions are using the same shared resource then it is a resource dependency condition. For example, floating point units or registers are shared, and this is known as *ALU dependency.* When memory is being shared, then it is called S*torage dependency.*

2.6.2 Bernstein Conditions for Detection of Parallelism

For execution of instructions or block of instructions in parallel, it should be ensured that the instructions are independent of each other. These instructions can be data dependent / control dependent / resource dependent on each other. Here we consider only data dependency among the statements for taking decisions of parallel execution. **A.J. Bernstein** has elaborated the work of data dependency and derived some conditions based on which we can decide the parallelism of instructions or processes.

Bernstein conditions are based on the following two sets of variables:

i) The Read set or input set R_I that consists of memory locations read by the statement of instruction I_1.
ii) The Write set or output set W_I that consists of memory locations written into by instruction I_1.

The sets R_I and W_I are not disjoint as the same locations are used for reading and writing by S_I.

The following are Bernstein Parallelism conditions which are used to determine whether statements are parallel or not:

1) Locations in R_1 from which S_1 reads and the locations W_2 onto which S_2 writes must be mutually exclusive. That means S_1 does not read from any memory location onto which S_2 writes. It can be denoted as:
$R_1 \cap W_2 = \phi$
2) Similarly, locations in R_2 from which S_2 reads and the locations W_1 onto which S_1 writes must be mutually exclusive. That means S_2 does not read from any memory location onto which S_1 writes. It can be denoted as: $R_2 \cap W_1 = \phi$
3) The memory locations W_1 and W_2 onto which S_1 and S_2 write, should not be read by S_1 and S_2. That means R_1 and R_2 should be independent of W_1 and W_2. It can be denoted as : $W_1 \cap W_2 = \phi$

To show the operation of Bernstein's conditions, consider the following instructions of sequential program:

I1 : $x = (a + b) / (a * b)$
I2 : $y = (b + c) * d$
I3 : $z = x^2 + (a * e)$

Now, the read set and write set of I1, I2 and I3 are as follows:

$R_1 = \{a,b\}$	$W_1 = \{x\}$
$R_2 = \{b,c,d\}$	$W_2 = \{y\}$
$R_3 = \{x,a,e\}$	$W_3 = \{z\}$

Now let us find out whether I_1 and I_2 are parallel or not

$R_1 \cap W_2 = \phi$
$R_2 \cap W_1 = \phi$
$W_1 \cap W_2 = \phi$

That means I_1 and I_2 are independent of each other.
Similarly for $I_1 \parallel I_3$,

$R_1 \cap W_3 = \phi$
$R_3 \cap W_1 \neq \phi$
$W_1 \cap W_3 = \phi$

Hence I_1 and I_3 are not independent of each other.
For $I_2 \parallel I_3$,

$R_2 \cap W_3 = \phi$
$R_3 \cap W_2 = \phi$

$W_3 \cap W_2 = \phi$

Hence, I_2 and I_3 are independent of each other.
Thus, I_1 and I_2, I_2 and I_3 are parallelizable but I_1 and I_3 are not.

2.6.3 Parallelism based on Grain size

Grain size: Grain size or Granularity is a measure which determines how much computation is involved in a process. Grain size is determined by counting the number of instructions in a program segment. The following types of grain sizes have been identified (shown in *Figure 17*):

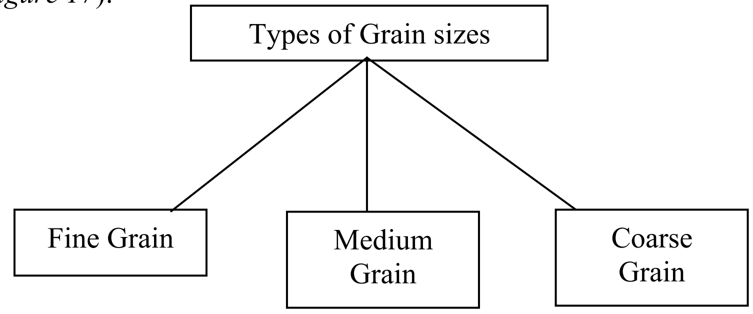

Figure 17: Types of Grain sizes

1) **Fine Grain:** This type contains approximately less than 20 instructions.
2) **Medium Grain:** This type contains approximately less than 500 instructions.
3) **Coarse Grain:** This type contains approximately greater than or equal to one thousand instructions.

Based on these grain sizes, parallelism can be classified at various levels in a program. These parallelism levels form a hierarchy according to which, lower the level, the finer is the granularity of the process. The degree of parallelism decreases with increase in level. Every level according to a grain size demands communication and scheduling overhead. Following are the parallelism levels (shown in *Figure 18*):

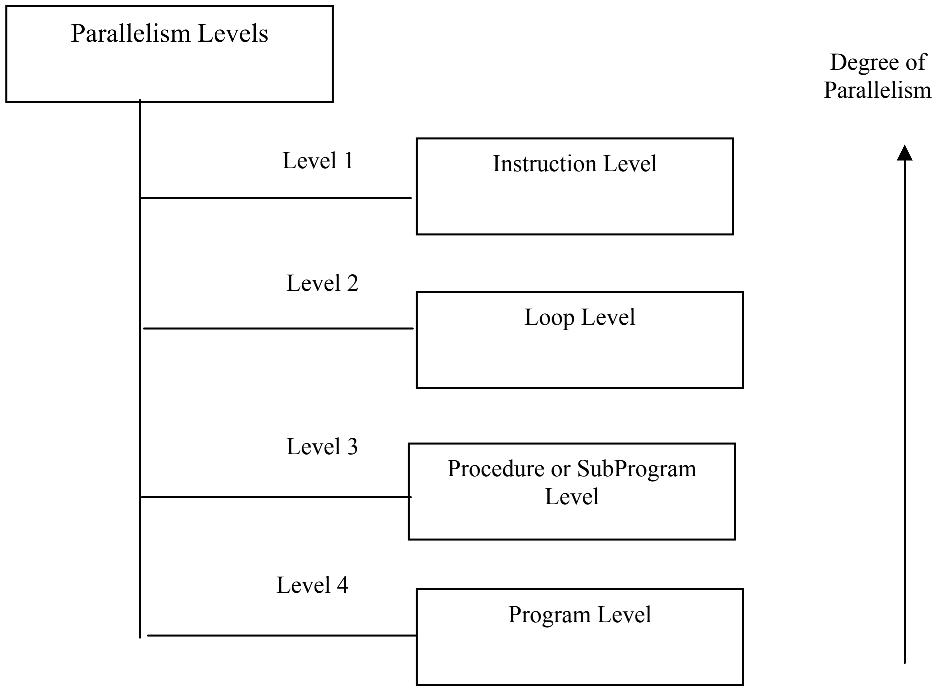

Figure 18: Parallelism Levels

1) **Instruction level:** This is the lowest level and the degree of parallelism is highest at this level. The fine grain size is used at instruction or statement level as only few instructions form the grain size here. The fine grain size may vary according to the type of the program. For example, for scientific applications, the instruction level grain size may be higher. As the higher degree of parallelism can be achieved at this level, the overhead for a programmer will be more.
2) **Loop Level :** This is another level of parallelism where iterative loop instructions can be parallelized. Fine grain size is used at this level also. Simple loops in a program are easy to parallelize whereas the recursive loops are difficult. This type of parallelism can be achieved through the compilers.

3) **Procedure or SubProgram Level:** This level consists of procedures, subroutines or subprograms. Medium grain size is used at this level containing some thousands of instructions in a procedure. Multiprogramming is implemented at this level. Parallelism at this level has been exploited by programmers but not through compilers. Parallelism through compilers has not been achieved at the medium and coarse grain size.

4) **Program Level:** It is the last level consisting of independent programs for parallelism. Coarse grain size is used at this level containing tens of thousands of instructions. Time sharing is achieved at this level of parallelism. Parallelism at this level has been exploited through the operating system.

The relation between grain sizes and parallelism levels has been shown in *Table 1*.

Table 1: Relation between grain sizes and parallelism

Grain Size	Parallelism Level
Fine Grain	Instruction or Loop Level
Medium Grain	Procedure or SubProgram Level
Coarse Grain	Program Level

Coarse grain parallelism is traditionally implemented in tightly coupled or shared memory multiprocessors like the Cray Y-MP. Loosely coupled systems are used to execute medium grain program segments. Fine grain parallelism has been observed in SIMD organization of computers.

Check Your Progress 3

1) Determine the dependency relations among the following instructions:
   I1: a = b+c;
   I2: b = a+d;
   I3: e = a/ f;
   ………………………………………………………………………………………
   ………………………………………………………………………………………
   ………………………………………………………………………………………
   ……………………………………………………………………

2) Use Bernstein's conditions for determining the maximum parallelism between the instructions in the following segment:
   S1: X = Y + Z
   S2: Z = U + V
   S3: R = S + V
   S4: Z = X + R
   S5: Q = M + Z

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
…………………………………………………………………………

3) Discuss instruction level parallelism.

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
…………………………………………………………………………

2.7 SUMMARY

In section 2.3, we discussed Flynn's Classification of computers. This classification scheme was suggested by Michael Flynn in 1972 and is based on the concepts of data stream and instruction stream. Next, we discuss Handler's classification scheme in section 2.4. This classification scheme, suggested by Wolfgang Handler in 1977, addresses the computers at the following three distinct levels:

- Processor Control Unit (PCU)
- Arithmetic Logic Unit (ALU)
- Bit-Level Circuit (BLC)

In section 2.5, in context of structural classification of computers, a number of new concepts are introduced and discussed. The concepts discussed include: Tightly Coupled (or shared memory) systems, loosely coupled (or distributed memory) systems. In the case of distributed memory systems, different types of Processor Interconnection Networks (PIN) are discussed. Another classification scheme based on the concept of grain size is discussed in section 2.6.

2.8 SOLUTIONS / ANSWERS

Check Your Progress 1

1) The following criteria have been identified for classifying parallel computers:
   - Classification based on instruction and data streams
   - Classification based on the structure of computers
   - Classification based on how the memory is accessed
   - Classification based on grain size

2) The flow of instructions from the main memory to the CPU is called instruction stream and a flow of operands between processor and memory bi-directionally is known as data stream.

3) a) F
   b) F
   c) F
   d) T

Check Your Progress 2

1) The following rules and operators are used to show the relationship between various elements of the computer:
   - The '*' operator is used to indicate that the units are pipelined or macro-pipelined with a stream of data running through all the units.

- The '+' operator is used to indicate that the units are not pipelined but work on independent streams of data.
- The 'v' operator is used to indicate that the computer hardware can work in one of several modes.
- The '~' symbol is used to indicate a range of values for any one of the parameters.
- Peripheral processors are shown before the main processor using another three pairs of integers. If the value of the second element of any pair is 1, it may be omitted for brevity.

1) The base for structural classification is multiple processors with memory being globally shared between processors or all the processors have their local copy of the memory.
1) When multiprocessors communicate through the global shared memory modules then this organization
is called shared memory computer or tightly coupled systems . When every processor in a multiprocessor system, has its own local memory and the processors communicate via messages transmitted between their local memories, then this organization is called distributed memory computer or loosely coupled system.
1) In UMA, each processor has equal access time to shared memory. In NUMA, local memories are connected with every processor and one reference to a local memory of the remote processor is not uniform. In COMA, all local memories of NUMA are replaced with cache memories.

Check Your Progress 3

1) Instructions I1 and I2 are both flow dependent and antidependent both. Instruction I2 and I3 are output dependent and instructions I1 and I3 are independent.

2)

$R_1 = \{Y,Z\}$	$W_1 = \{X\}$
$R_2 = \{U,V\}$	$W_2 = \{Z\}$
$R_3 = \{S,V\}$	$W_3 = \{R\}$
$R_4 = \{X,R\}$	$W_4 = \{Z\}$
$R_5 = \{M,Z\}$	$W_5 = \{Q\}$

Thus, S1, S3 and S5 and S2 & S4 are parallelizable.

3) This is the lowest level and the degree of parallelism is highest at this level. The fine grain size is used at instruction or statement level as only few instructions form the grain size here. The fine grain size may vary according to the type of the program. For example, for scientific applications, the instruction level grain size may be higher. The loops As the higher degree of parallelism can be achieved at this level, the overhead for a programmer will be more.