Module 1

1 List the performance factors and system attributes. Explain how performance factors are influenced by system attributes.

Clock Rate and CPI

• The CPU (or simply the processor) of today's digital computer is driven by a clock with a constant cycle time (τ in nanoseconds).

• The inverse of the cycle time is the clock rate (/ = 1/ τ in megahertz). The size of a program is determined by its instruction count (Ic), in terms of the number of machine instructions to be executed in the program.

• Different machine instructions may require different numbers of clock cycles to execute. Therefore, the cycles per instruction (CPI) becomes an important parameter for measuring the time needed to execute each instruction.

• For a given instruction set, we can calculate an average CPI over all instruction types, provided we know their frequencies of appearance in the program.

• An accurate estimate of the average CPI requires a large amount of program code to be traced over a long period of time.

• Unless specifically focusing on a single instruction type, we simply use the term CPI to mean the average value with respect to a given instruction set and a given program mix.

Performance Factors

• Let Ic be the number of instructions in a given program, or the instruction count.

• The CPU time (T in seconds/program) needed to execute the program is estimated by finding the product of three contributing factors:

$T = I_c \times CPI \times \tau$ (1.1)

• The execution of an instruction requires going through a cycle of events involving the instruction fetch, decode, operand(s) fetch, execution, and store results.

• In this cycle, only the instruction decodes and execution phases are carried out in the CPU.

• The remaining three operations may be required to access the memory. We define a memory cycle as the time needed to complete one memory reference.

• Usually, a memory cycle is k times the processor cycle τ.

• The value of k depends on the speed of the memory technology and processor-memory interconnection scheme used.

• The CPI of an instruction type can be divided into two component terms corresponding to the total processor cycles and memory cycles needed to complete the execution of the instruction.

• Depending on the instruction type, the complete instruction cycle may involve one to four memory references (one for instruction fetch, two for operand fetch, and one for store results). Therefore, we can rewrite Eq. 1.1 as follows;

$T = I_c \times (p + m \times k) \times \tau$ (1.2)

where p is the number of processor cycles needed for the instruction decode and execution, m is the number of memory references needed,

k is the ratio between memory cycle and processor cycle, Ic is the instruction count, r is the processor cycle time.

Equation 1.2 can be further refined once the CPi components (p,m,k) are weighted over the entire instruction set.

System Attributes

• The above five performance factors (Ic, p, m, k, τ) are influenced by four system attributes: instruction-set architecture, compiler technology, CPU implementation and control, and cache and memory hierarchy, as specified in Table 1.2.

• The instruction-set architecture affects the program length (Ic) and processor cycle needed (p). The compiler technology affects the values of Ic, p and the memory reference count (m).

• The CPU implementation and control determine the total processor time (p. τ) needed.

• Finally, the memory technology and hierarchy design affect the memory access latency (k. τ). The above CPU time can be used as a basis in estimating the execution rate of a processor.

Table 1.2 Performance Factors Versus System Attributes

System Attributes	Performance Factors				
	Instr. Count,	Average Cycles per Instruction, CP1			Processor Cycle Time, T
		Processor Cycles per Instruction, p	Memory References per Instruction, *m*	Memory-Access Latency, *k*	
Instruction-set Architecture	X	X			
Compiler Technology	X	X	X		
Processor Implementation and Control		X			X
Cache and Memory Hierarchy				X	X

<u>MIPS Rate</u>

• Let C be the total number of clock cycles needed to execute a given program.

• Then the CPU time in Eq. 1.2 can be estimated as T = C x τ = C/f.

• Furthermore, CPI = C/Ic and T = Ic x CPI x τ = Ic x CPI/f. The processor speed is often measured in terms of million instructions per second (MIPS).

• We simply call it the MIPS rate of a given processor. It should be emphasized that the MIPS rate varies with respect to a number of factors, including the clock rate (f), the instruction count (Ic), and the CPI of a given machine, as defined below:

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{\text{CPI} \times 10^6} = \frac{f \times I_c}{C \times 10^6} \qquad (1.3)$$

• Based on Eq. 1.3, the CPU time in Eq. 1.2 can also be written as T = Ic X 10-6 / MIPS.

• Based on the system attributes identified in Table 1.2 and the above derived expressions, we conclude by indicating the fact that the MIPS rate of a given computer is directly proportional to the clock rate and inversely proportional to the CPI.

• All four system attributes, instruction set, compiler, processor, and memory technologies, affect the MIPS rate, which varies also from program to program.

Throughput Rate

• Number of programs a system can execute per unit time, called the system throughput Ws (in programs/second).

• In a multiprogrammed system, the system throughput is often lower than the CPU throughput Wp defined by:

$$W_p = \frac{f}{I_c \times \text{CPI}} \tag{1.4}$$

• Note that Wp = (MIPS) X 106/Ic from Eq. 1.3- The unit for Wp is programs/second.

• The CPU throughput is a measure of how many programs can be executed per second, based on the MIPS rate and average program length (Ic).

• The reason why Ws < Wp is due to the additional system overheads caused by the I/O, compiler, and OS when multiple programs are interleaved for CPU execution by multiprogramming or timesharing operations.

• If the CPU is kept busy in a perfect program-interleaving fashion, then Ws = Wp. This will probably never happen, since the system overhead often causes an extra delay and the CPU may be left idle for some cycles.

2 Explain the architecture of vector super computer with neat diagram

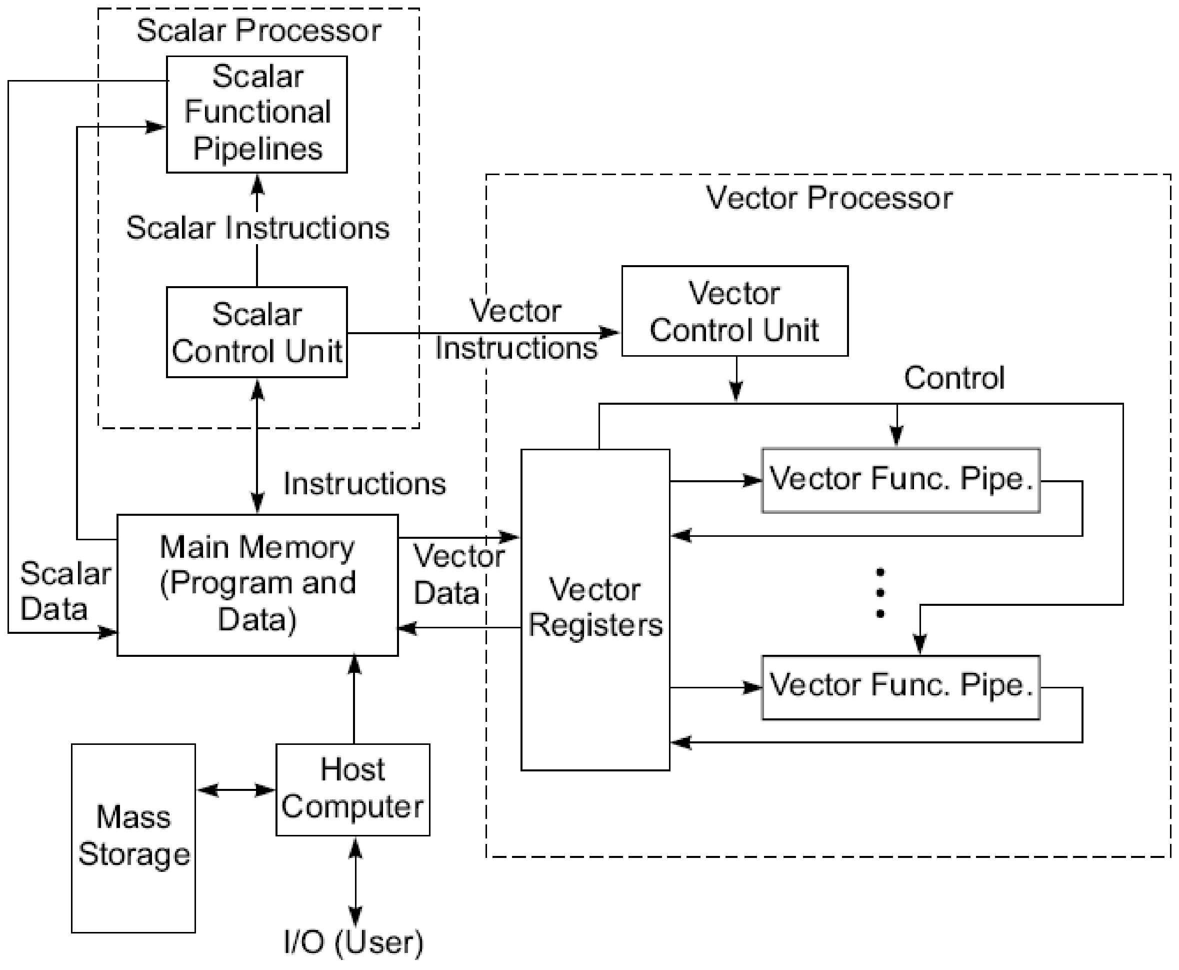


Fig. 1.11 The architecture of a vector supercomputer

A vector computer is often built on top of a scalar processor.

 As shown in Fig. 1.11, the vector processor is attached to the scalar processor as an optional feature.

 Program and data are first loaded into the main memory through a host computer.

 All instructions are first decoded by the scalar control unit.

 If the decoded instruction is a scalar operation or a program control operation, it will be directly executed by the scalar processor using the scalar functional pipelines.

 If the instruction is decoded as a vector operation, it will be sent to the vector control unit.

 This control unit will supervise the flow of vector data between the main memory and vector functional pipelines.

 The vector data flow is coordinated by the control unit. A number of vector functional pipelines may be built into a vector processor.

Vector Processor Models

 Figure l.ll shows a register-to-register architecture.

 Vector registers are used to hold the vector operands, intermediate and final vector results.

 The vector functional pipelines retrieve operands from and put results into the vector registers.

 All vector registers are programmable in user instructions.

 Each vector register is equipped with a component counter which keeps track of the component registers used in successive pipeline cycles.

 The length of each vector register is usually fixed, say, sixty-four 64-bit component registers in a vector register in a Cray Series supercomputer.

 Other machines, like the Fujitsu VP2000 Series, use reconfigurable vector registers to dynamically match the register length with that of the vector operands.

3 Define data dependency. Explain different functions of data dependency with the help of dependency graph.

 The ability to execute several program segments in parallel requires each segment to be independent of the other segments. We use a dependence graph to describe the relations.

 The nodes of a dependence graph correspond to the program statement (instructions), and directed edges with different labels are used to represent the ordered relations among the statements.

 The analysis of dependence graphs shows where opportunity exists for parallelization and vectorization.

Data dependence:

The ordering relationship between statements is indicated by the data dependence. Five type of data dependence are defined below:

1. **Flow dependence**: A statement S2 is flow dependent on S1 if an execution path exists from s1 to S2 and if at least one output (variables assigned) of S1feeds in as input (operands to be used) to S2 also called RAW $S_1 \rightarrow S_2$ hazard and denoted as

2. **Antidependence**: Statement S2 is antidependent on the statement S1 if S2 follows S1 in the program order and if the output of S2 overlaps the input to S1 also called RAW hazard and denoted as $S_1 \nrightarrow S_2$

3. **Output dependence:** Two statements are output dependent if they produce (write) the same output variable. Also called WAW hazard and denoted as $S_1 \circ\!\!\!\rightarrow S_2$

4**. I/O dependence:** Read and write are I/O statements. I/O dependence occurs not because the same variable is involved but because the same file referenced by both I/O statement.

5. **Unknown dependence:** The dependence relation between two statements cannot be determined in the following situations:

• The subscript of a variable is itself subscribed (indirect addressing)

• The subscript does not contain the loop index variable.

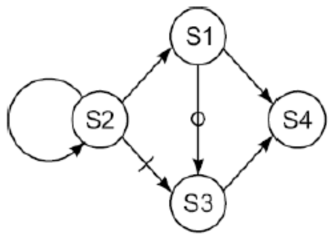
• A variable appears more than once with subscripts having different coefficients of the loop variable.

• The subscript is nonlinear in the loop index variable.

Parallel execution of program segments which do not have total data independence can produce non- deterministic results.

Example: Consider the following fragment of a program:

S1:	**Load**	**R1, A**	**/R1 ← Memory(A) /**
S2:	**Add**	**R2, R1**	**/R2 ← (R1) + (R2)/**
S3:	**Move**	**R1, R3**	**/R1 ← (R3)/**
S4:	**Store**	**B, R1**	**/Memory(B) ← (R1)/**



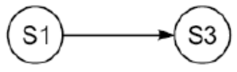
(a) Dependence graph

• Here the Flow dependency from **S1 to S2**, **S3 to S4**, **S2 to S2**

• Anti-dependency from **S2 to S3**

• Output dependency **S1 toS3**

S1: Read (4), A(I)	/Read array A from file 4/
S2: Rewind (4)	/Process data/
S3: Write (4), B(I)	/Write array B into file 4/
S4: Rewind (4)	/Close file 4/



(b) I/O dependence caused by accessing the same file by the read and write statements

The read/write statements S1 and S2 are I/O dependent on each other because they both access the same file.

4 A 40 MHz processor was used to execute a benchmark program with the following instruction mix and clock cycle counts.

Instruction type	Instruction count	Cycles/instruction
Integer arithmetic	45000	1
Data transfer	32000	2
Floating point	15000	2
Control transfer	8000	2

Determine the effective CPI, MIPS rate and execution time for this program.

Step 1 of 6

Given data:

Clock speed of the Processor = 40 MHz

Number of instructions the executed program consists = 10,000

Instruction Type	**Instruction Count**	**Cycles per Instruction**	**cycles**
Integer arithmetic	45,000	1	45000
Data transfer	32,000	2	64000
Floating point	15,000	2	30000
Control transfer	8000	2	16000

Step 2 of 6

Calculating the CPI:

The formula to calculate CPI is,

$$\text{CPI} = \frac{\text{Instruction count} \times \text{Cycles per second}}{\text{Number of instructions the executed program consists}} \quad \text{...... (1)}$$

Substitute the values for "instruction count" and "Cycles per second" from the above table in Equation (1),

$$\begin{aligned}\text{CPI} &= \frac{(45000 \times 1)+(32000 \times 2)+(15000 \times 2)+(8000 \times 2)}{100000} \\ &= \frac{45000+64000+30000+16000}{100000} \\ &= \frac{155000}{100000} \\ &= 1.55\end{aligned}$$

Therefore, the CPI for this program is $\boxed{1.55}$.

Step 3 of 6

Calculating MIPS:

The *Million Instructions per Second* (MIPS) rate can be calculated with the following constraints.

Processor Time, $T = I_c \times CPI \times \tau$

Where ,

- τ represents constant cycle time

τ can be calculated as $\frac{1}{f}$

Here f indicates constant frequency

Step 4 of 6

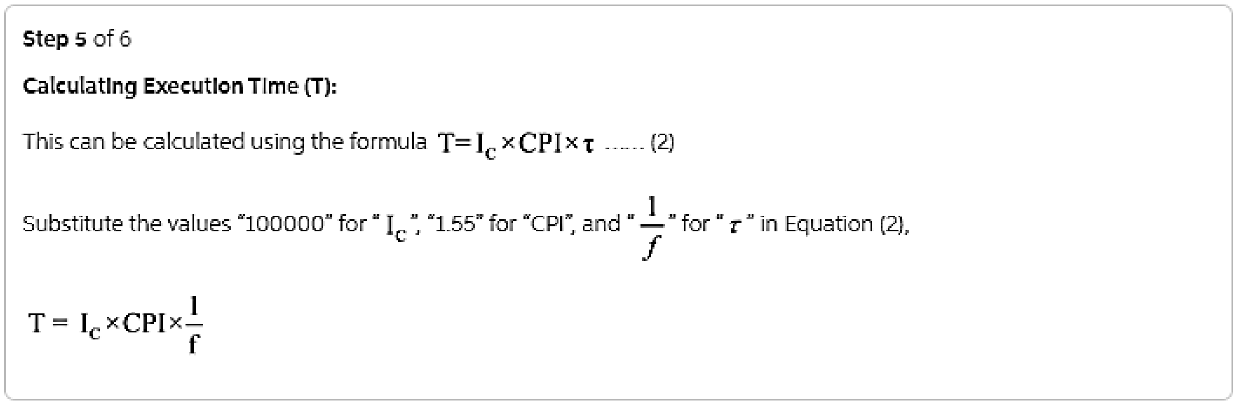
So, based on the above data, MIPS can be calculated as

$$\begin{aligned} MIPS &= \frac{I_C}{T \times 10^6} \\ &= \frac{I_C}{I_C \times CPI \times \tau \times 10^6} \\ &= \frac{1}{CPI \times \frac{1}{f} \times 10^6} \\ &= \frac{f}{CPI \times 10^6} \end{aligned}$$

Frequency is given as 40MHz, substitute “40” for “f” in the above formula,

$$\begin{aligned} MIPS &= \frac{40 \times 10^6}{1.55 \times 10^6} \\ &= \frac{40}{1.55} \\ &= 25.8 \end{aligned}$$

Therefore, the MIPS for this program is $\boxed{25.8}$



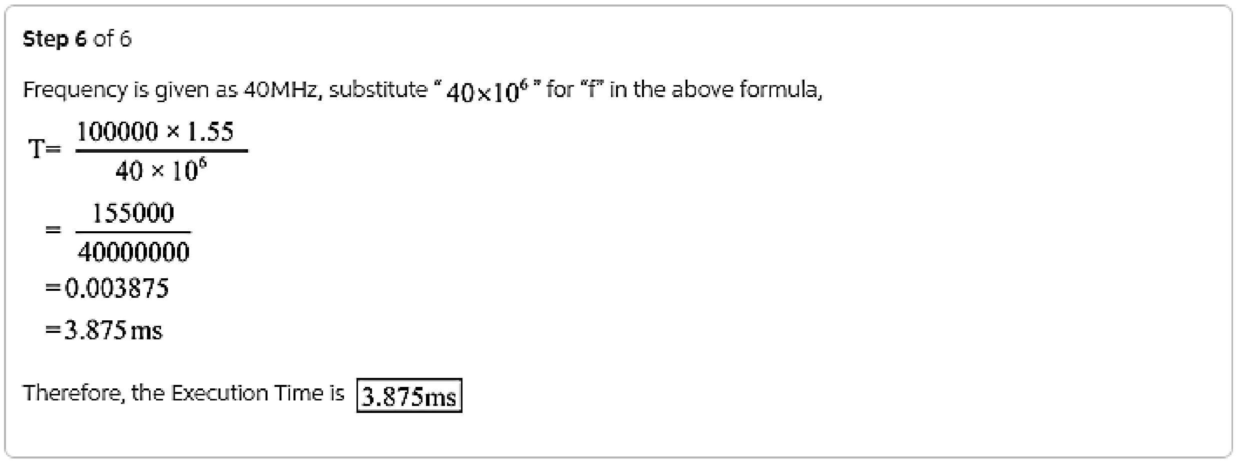
Step 5 of 6

Calculating Execution Time (T):

This can be calculated using the formula $T = I_c \times CPI \times \tau$ ...... (2)

Substitute the values "100000" for "I_c", "1.55" for "CPI", and "$\frac{1}{f}$" for "τ" in Equation (2),

$$T = I_c \times CPI \times \frac{1}{f}$$



Step 6 of 6

Frequency is given as 40MHz, substitute "40×10^6" for "f" in the above formula,

$$T = \frac{100000 \times 1.55}{40 \times 10^6}$$
$$= \frac{155000}{40000000}$$
$$= 0.003875$$
$$= 3.875\,ms$$

Therefore, the Execution Time is $\boxed{3.875ms}$

5 What are the conditions of parallelism? Explain the types of data dependence

6 What are the metrics affecting scalability of a computer system?

Page 128 - text-book

7 What are the important characteristics of parallel algorithms?

Page 115 – text-book

8 Explain the evolution of computer architecture.

 The study of computer architecture involves both hardware organization and programming/software requirements.

 As seen by an assembly language programmer, computer architecture is abstracted by its instruction set, which includes opcode (operation codes), addressing modes, registers, virtual memory, etc.

 From the hardware implementation point of view, the abstract machine is organized with CPUs, caches, buses, microcode, pipelines, physical memory, etc.

 Therefore, the study of architecture covers both instruction-set architectures and machine implementation organizations.

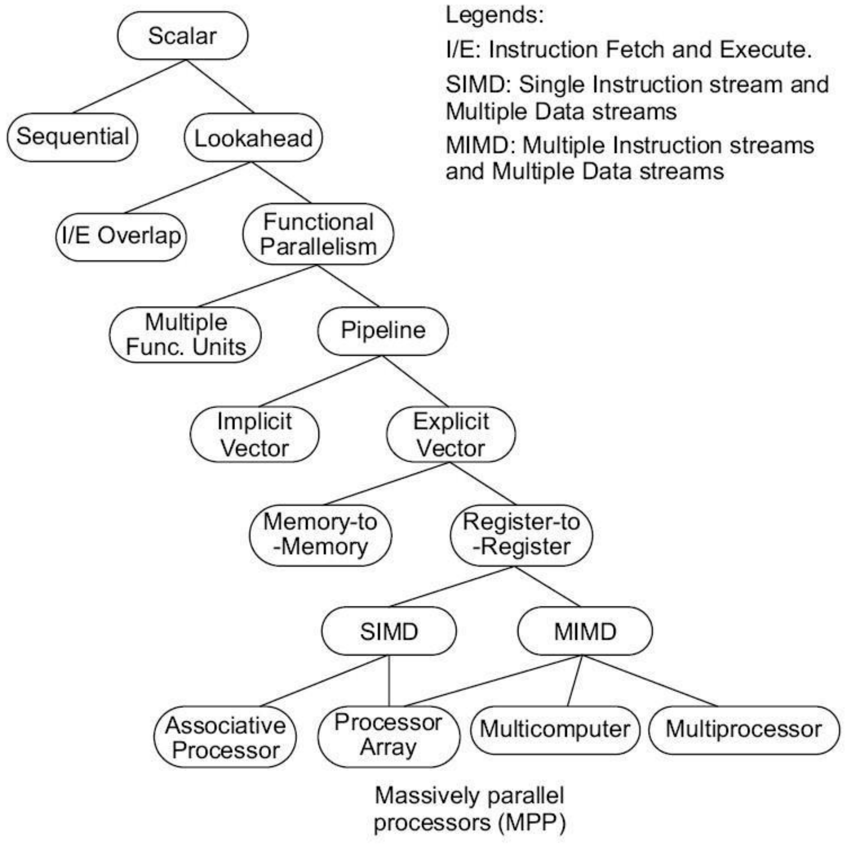


Fig. 1.2 Tree showing architectural evolution from sequential scalar computers to vector processors and parallel computers

<u>Lookahead, Parallelism, and Pipelining</u>

Lookahead techniques were introduced to prefetch instructions in order to overlap I/E (instruction fetch/decode and execution) operations and to enable functional parallelism. Functional parallelism was supported by two approaches:

1. using multiple functional units simultaneously,
2. to practice pipelining at various processing levels.

9 Explain with diagram the operational model of SIMD super computer.

SIMD computers have a single instruction stream over multiple data streams. An operational model of an SIMD computer is specified by a 5-tuple:

M = (N,C, I,M, R)

1. N is the number of processing elements (PEs) in the machine. For example, the Illiac IV had 64 PEs and the Connection Machine CM-2 had 65,536 PEs.

2. C is the set of instructions directly executed by the control unit (CU), including scalar and program flow control instructions.

3. I s the set of instructions broadcast by the CU to all PEs for parallel execution. These include arithmetic, logic, data routing, masking, and other local operations executed by each active PE over data within that PE.

4. M is the set of masking schemes, where each mask partitions the set of PEs into enabled and disabled subsets.

5. R is the set of data-routing functions, specifying various patterns to be set up in the interconnection network for inter-PE communications.

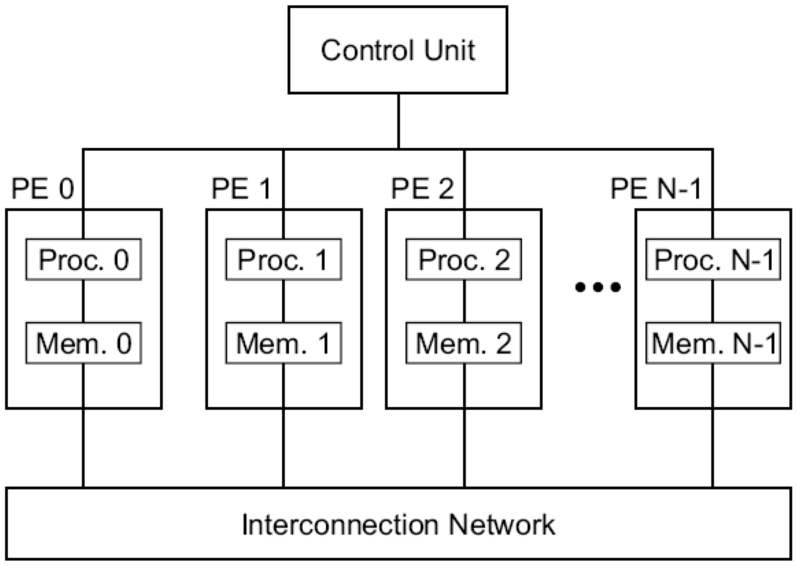


Fig. 1.12 Operational model of SIMD computers

10 Explain the Bernstein's conditions for parallelism. Detect the parallelism in the following code using Bernstein's conditions. (Assume no pipeline).

P1 : C = D * E; P2 : M = G + C; P3 : A = B + C; P4 : C = L + M; P5: G / E

Refer Page – 52 from text-book

11 With a diagram, explain the operation of tagged token data flow computer.

• The Arvind machine (MIT) has N PEs and an N-by-N interconnection network.

• Each PE has a token-matching mechanism that dispatches only instructions with data tokens available.

• Each datum is tagged with

o address of instruction to which it belongs

o context in which the instruction is being executed

• Tagged tokens enter PE through local path (pipelined), and can also be communicated to other PEs through the routing network.

Instruction address(es) effectively replace the program counter in a control flow machine.

• Context identifier effectively replaces the frame base register in a control flow machine.

• Since the dataflow machine matches the data tags from one instruction with successors, synchronized instruction execution is implicit.

• An I-structure in each PE is provided to eliminate excessive copying of data structures.

• Each word of the I-structure has a two-bit tag indicating whether the value is empty, full or has pending read requests.

• This is a retreat from the pure dataflow approach.

• Special compiler technology needed for dataflow machines.

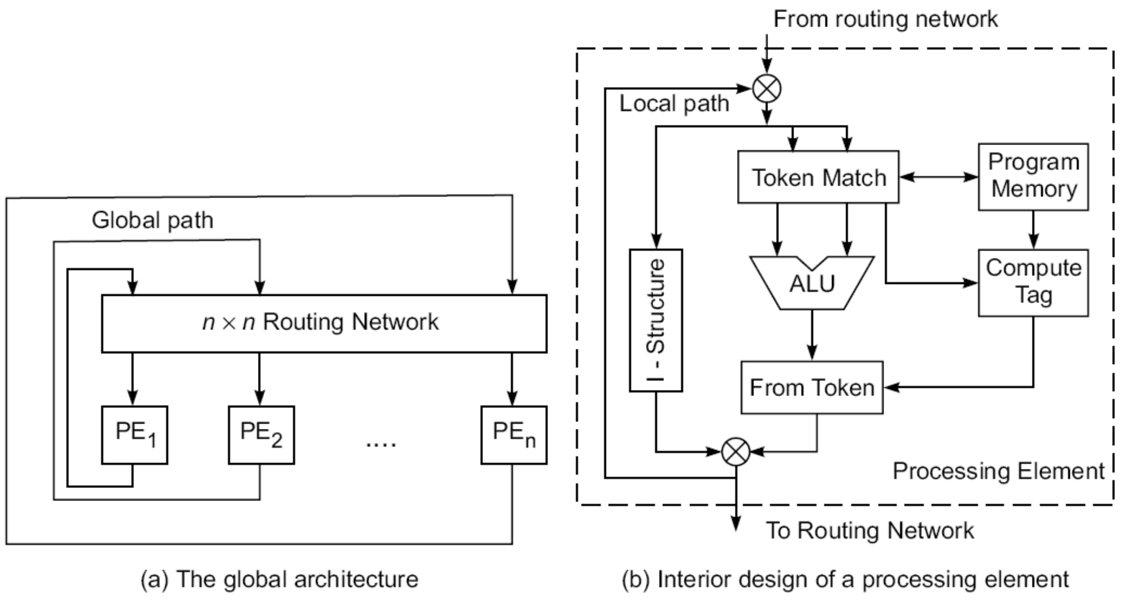


Fig. 2.12 The MIT tagged-token dataflow computer (adapted from Arvind and Iannucci, 1986 with permission)

12 With a neat diagram explain the elements of modern computer system.

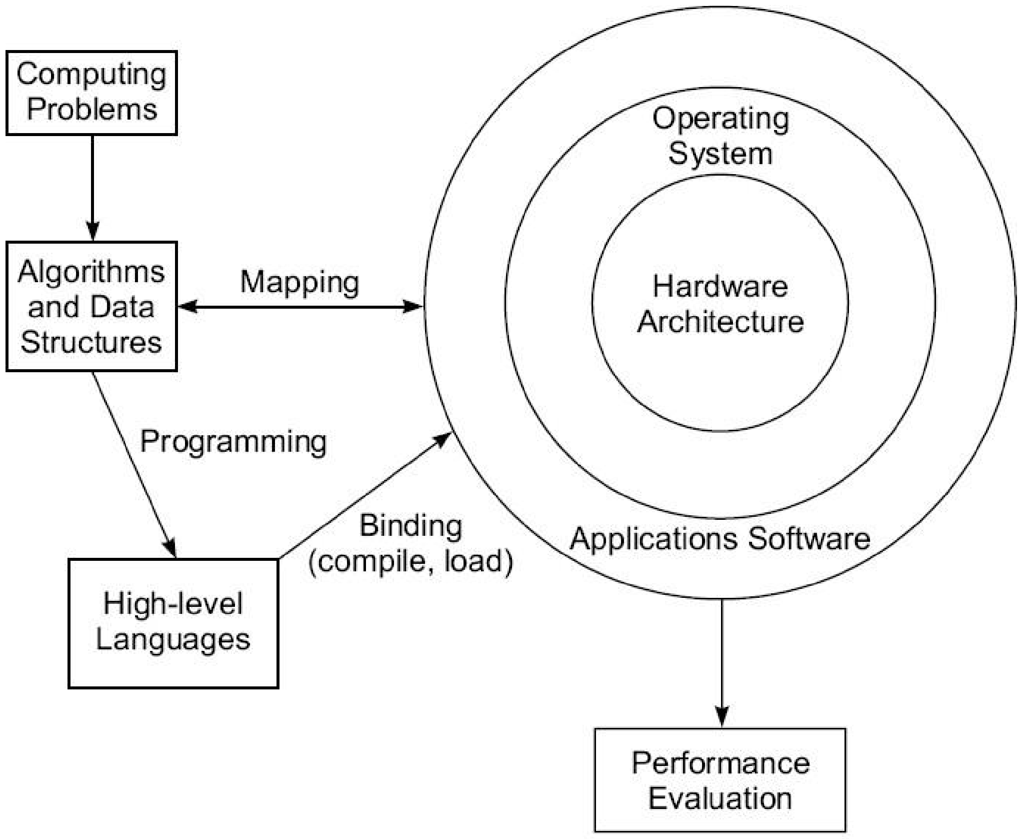


Fig. 1.1 Elements of a modern computer system

Computing Problems

 The use of a computer is driven by real-life problems demanding fast and accurate solutions. Depending on the nature of the problems, the solutions may require different computing resources.

 For numerical problems in science and technology, the solutions demand complex mathematical formulations and tedious integer or floating-point computations.

 For alpha numerical problems in business and government, the solutions demand accurate transactions, large database management, and information retrieval operations.

 For artificial intelligence (AI) problems, the solutions demand logic inferences and symbolic manipulations.

 These computing problems have been labeled numerical computing, transaction processing, and
logical reasoning.

 Some complex problems may demand a combination of these processing modes.

- Hardware Resources

 A modern computer system demonstrates its power through coordinated efforts by hardware resources, an operating system, and application software.

 Processors, memory, and peripheral devices form the hardware core of a computer system.

 Special hardware interfaces are often built into I/O devices, such as terminals, workstations, optical page scanners, magnetic ink character recognizers, modems, file servers, voice data entry, printers, and plotters.

 These peripherals are connected to mainframe computers directly or through local or wide-area networks.

- Operating System

 An effective operating system manages the allocation and deallocation of resources during the execution of user programs.

 Beyond the OS, application software must be developed to benefit the users.

 Standard benchmark programs are needed for performance evaluation.

 Mapping is a bidirectional process matching algorithmic structure with hardware architecture, and vice versa.

 Efficient mapping will benefit the programmer and produce better source codes.

 The mapping of algorithmic and data structures onto the machine architecture includes processor scheduling, memory maps, interprocessor communications, etc.

 These activities are usually architecture-dependent.

- System Software Support

 Software support is needed for the development of efficient programs in high-level languages. The source code written in a HLL must be first translated into object code by an optimizing compiler.

 The compiler assigns variables to registers or to memory words and reserves functional units for operators.

 An assembler is used to translate the compiled object code into machine code which can be recognized by the machine hardware. A loader is used to initiate the program execution through the OS kernel.

- Compiler Support

There are three compiler upgrade approaches:

 Preprocessor: A preprocessor uses a sequential compiler and a low-level library of the target computer to implement high-level parallel constructs.

 Precompiler: The precompiler approach requires some program flow analysis, dependence checking, and limited optimizations toward parallelism detection.

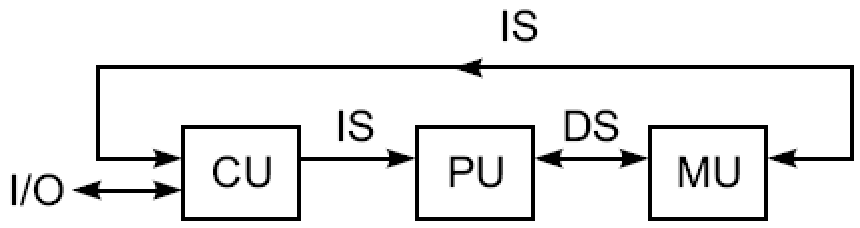
 Parallelizing Compiler: This approach demands a fully developed parallelizing or vectorizing compiler which can automatically detect parallelism in source code and transform sequential codes into parallel constructs.

13 Explain Flynn's classification of computer architecture.

Michael Flynn (1972) introduced a classification of various computer architectures based on notions of instruction and data streams.

1. SISD (Single Instruction stream over a Single Data stream) computers
2. SIMD (Single Instruction stream over Multiple Data streams) machines
3. MIMD (Multiple Instruction streams over Multiple Data streams) machines.
4. MISD (Multiple Instruction streams and a Single Data stream) machines

<u>1. SISD (Single Instruction stream over a Single Data stream) computers</u>



(a) SISD uniprocessor architecture

 Conventional sequential machiunes are called SISD computers.

 They are also called scalar processor i.e., one instruction at a time and each instruction have only one set of operands.

 Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle

 Single data: only one data stream is being used as input during any one clock cycle

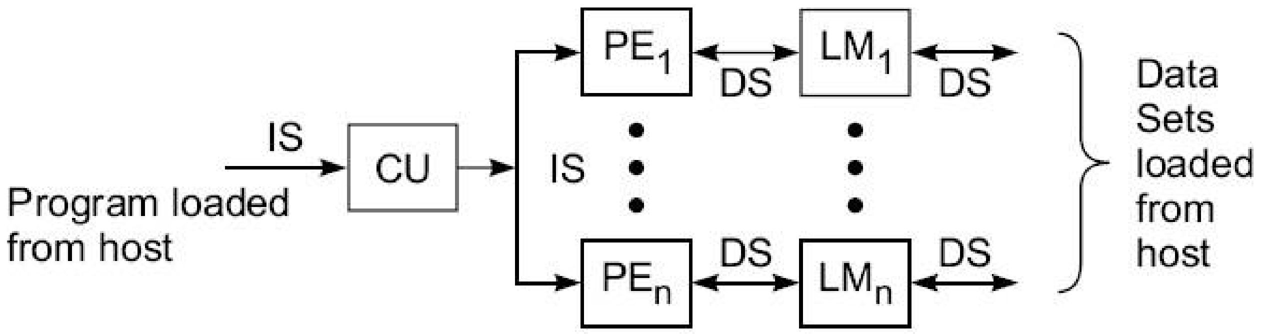
 Deterministic execution

 Instructions are executed sequentially.

 This is the oldest and until recently, the most prevalent form of computer

 Examples: most PCs, single CPU workstations and mainframes

<u>2. SIMD (Single Instruction stream over Multiple Data streams) machines</u>



(b) SIMD architecture (with distributed memory)

A type of parallel computer

 Single instruction: All processing units execute the same instruction issued by the control unit at any given clock cycle.

 Multiple data: Each processing unit can operate on a different data element. The processors are connected to shared memory or interconnection network providing multiple data to processing unit.

 This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.

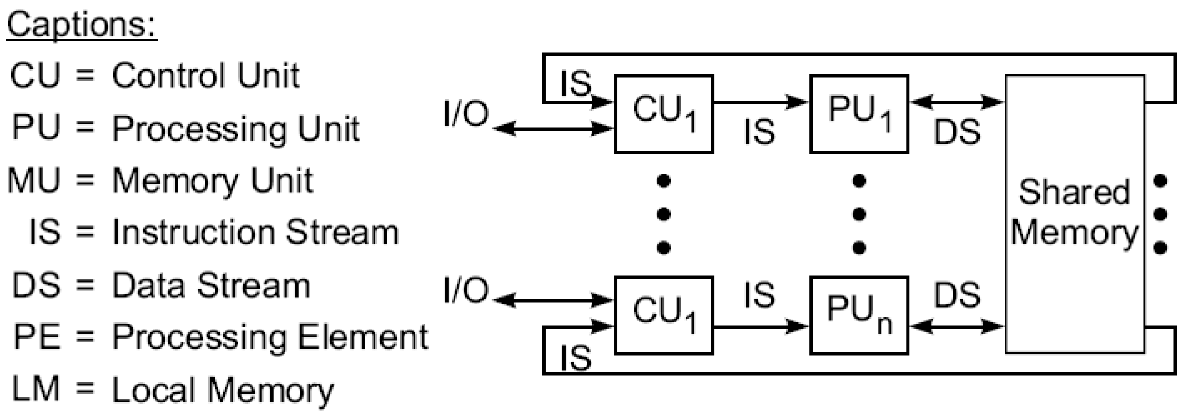
 Thus single instruction is executed by different processing unit on different set of data.

 Best suited for specialized problems characterized by a high degree of regularity, such as image processing and vector computation.

 Synchronous (lockstep) and deterministic execution.

 Two varieties: Processor Arrays e.g., Connection Machine CM-2, Maspar MP-1, MP-2 and Vector Pipelines processor e.g., IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

3. MIMD (Multiple Instruction streams over Multiple Data streams) machines.



(c) MIMD architecture (with shared memory)

 A single data stream is fed into multiple processing units.

 Each processing unit operates on the data independently via independent instruction streams.

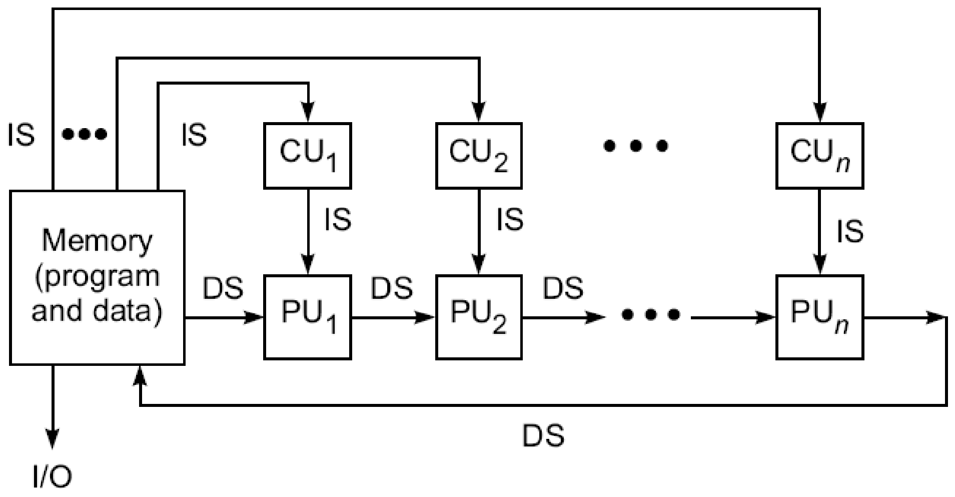
 A single data stream is forwarded to different processing unit which are connected to different control unit and execute instruction given to it by control unit to which it is attached.

 Thus in these computers same data flow through a linear array of processors executing different instruction streams.

 This architecture is also known as Systolic Arrays for pipelined execution of specific instructions. Some conceivable uses might be:

1. multiple frequency filters operating on a single signal stream
2. multiple cryptography algorithms attempting to crack a single coded message.

<u>4. MISD (Multiple Instruction streams and a Single Data stream) machines</u>



(d) MISD architecture (the systolic array)

Fig. 1.3 Flynn's classification of computer architectures (Derived from Michael Flynn,

 Multiple Instructions: Every processor may be executing a different instruction stream

 Multiple Data: Every processor may be working with a different data stream, multiple data stream is provided by shared memory.

 Can be categorized as loosely coupled or tightly coupled depending on sharing of data and control.

 Execution can be synchronous or asynchronous, deterministic or non-deterministic

 There are multiple processors each processing different tasks.

 Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.