Module 6

By

Nilanjan Byabarta

Content

Revisit Computer Architecture Design Goals

Key subsystems in a computer system are processors, memories, I/O interfaces and the data paths connecting them

Subsystem within the processors are functional units, registers, cache memories and internal data buses.

Computer Architecture is defined as the arrangement by which the various system building blocks are interconnected and inter-operated to achieve desired *system performance*

System performance is a key benchmark in the study of computer architecture – Parameters: Throughput, Response Time

Parallel Processing - Introduction

Parallel Processing: An efficient form of information processing which emphasizes the exploitation of concurrent events in the computing process. [Hwang, Briggs]

Types of Concurrency:

- Parallelism – during same time interval
- Simultaneity – at the same time instant
- Pipelining – overlapped time spans

Parallel processing is a cost effective means to improve system performance through concurrent activities in the computer and reducing bottlenecks

Levels of Parallelism

Job or Program level

Task or procedure level

Inter-instruction level

Intra-instruction level

Instruction Level Parallelism

Instruction-level parallelism (ILP) of a program—a measure of the average number of instructions in a program that, in theory, a processor might be able to execute at the same time

Mostly determined by the number of true (data) dependencies and procedural (control) dependencies in relation to the number of other instructions

ILP is traditionally “extracting parallelism from a single instruction stream working on a single stream of data”

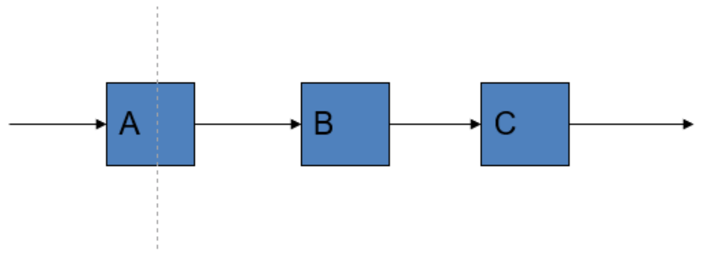
Parallelism in Uni-Processor System

Hardware approaches:

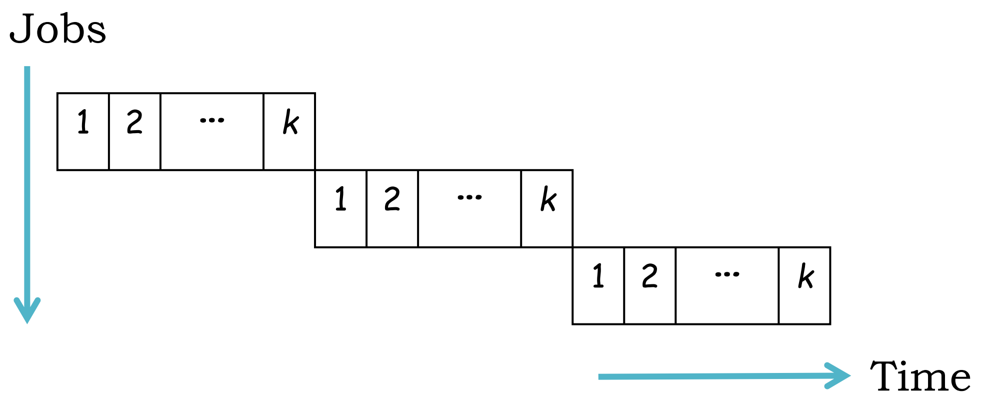
- Parallelism and pipelining within the CPU –
  - *Instruction & Arithmetic Pipelines*
- Multiplicity of functional units –
  - *Many of the functions of the ALU can be distributed to multiple and specialized functional units which can operate in parallel and may be pipelined*
  - *Found in super-scalar architecture*

Pipelining Concept

- Linear Pipeline Structure:
  - Linear cascade of processing stages
  - Performs a fixed function – static pipeline
  - Stream of data flows from one end to another

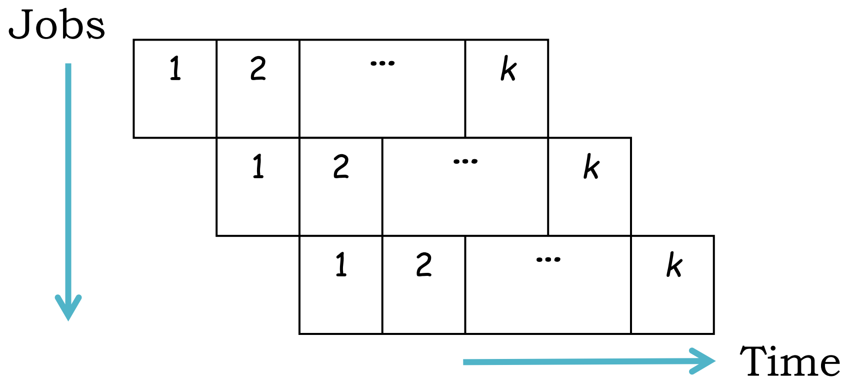


- Serial vs Pipelined Execution:



Without Pipelining

One completion every *k* time units



With Pipelining

One completion every 1 time unit

Instruction Pipeline Example

An Example of seven-stage instruction pipeline

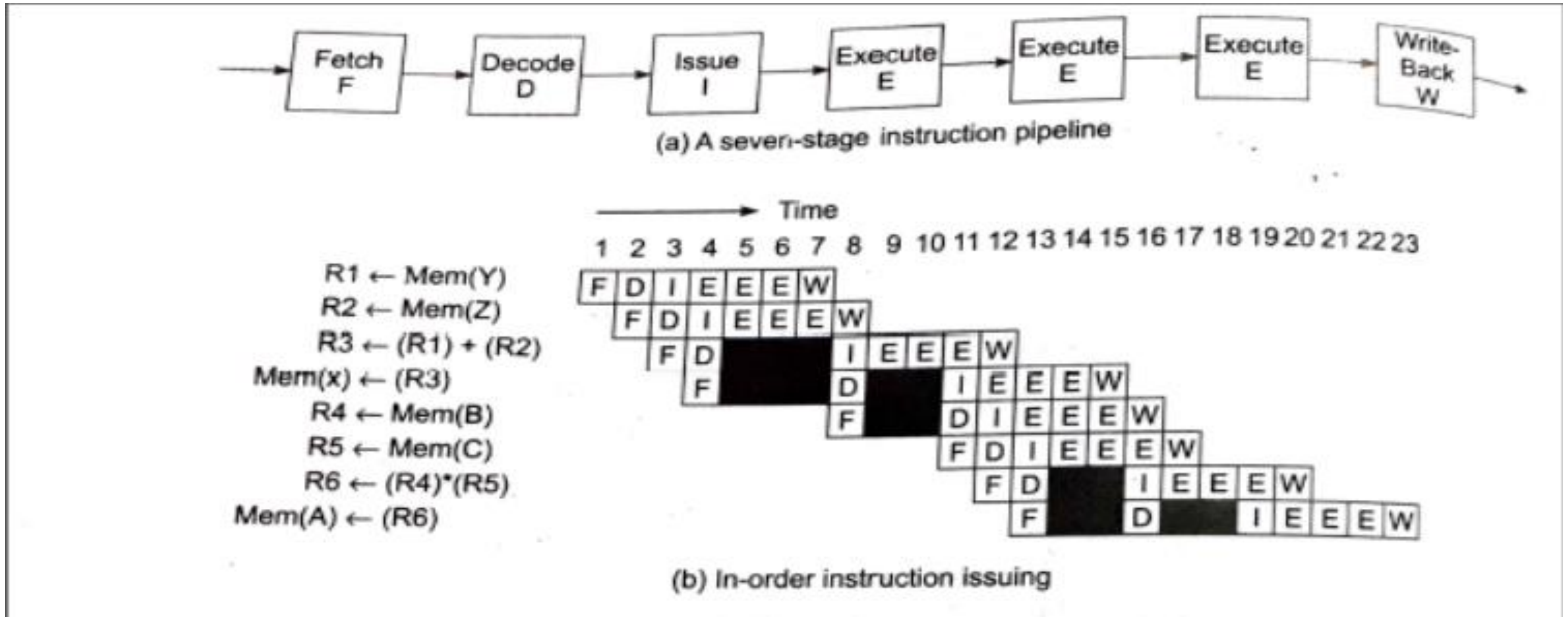


Fig 1. 7-stage instruction pipeline [source: Hwang, Jotwani]

Multiple Issue Processors

Ideally, pipelined execution of an instruction stream in a RISC scalar processor can bring the average CPI (Clock Cycle Per Instruction) down to 1.

But, can we achieve CPI < 1 ?

Techniques invented to get CPI < 1:

- Super-pipelining
- Superscalar
- VLIW (Very Long Instruction Word)

Super-pipelined Architecture

Pipeline stages are sub-divided into multiple sub-stages

Thus, number of instructions supported at any given moment increases and so, speed-up factor is improved

Limitations on the number of stages:

- Number of stages cannot increase indefinitely due to practical constraints on costs, control complexity, circuit implementation and packaging limitations.
- Increasing the number of stages over the optimal number of stages would reduce the pipeline's performance/cost ratio.

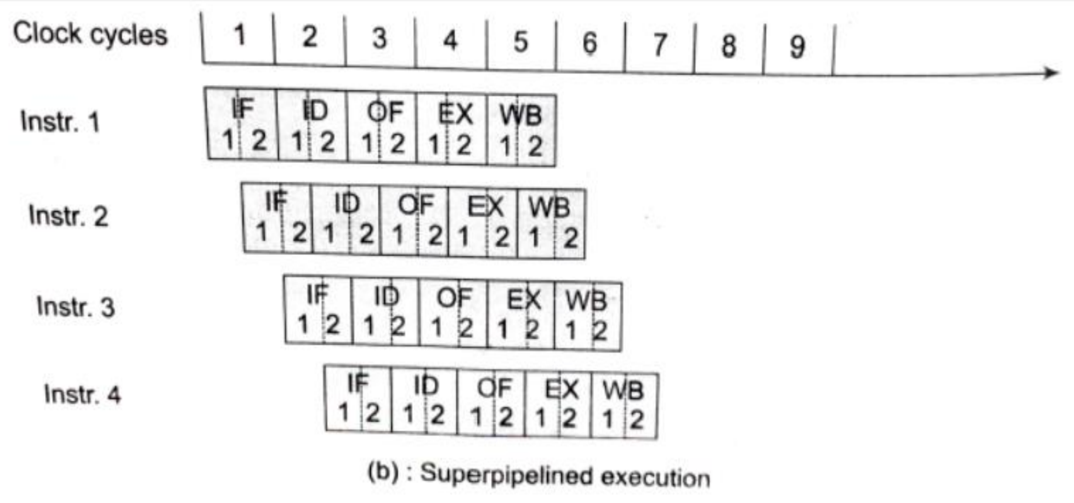


Fig 4. Super-pipeline execution [source: T.K Ghosh]

Superscalar Architecture

Several instructions can be initiated simultaneously and executed independently.

In an m-issue superscalar processor:

- m-instructions can be issued simultaneously per cycle
- The instruction decoding and execution units are increased to form *effectively m pipelines operating concurrently.*
- At some stages, the functional units may be shared by multiple pipelines.

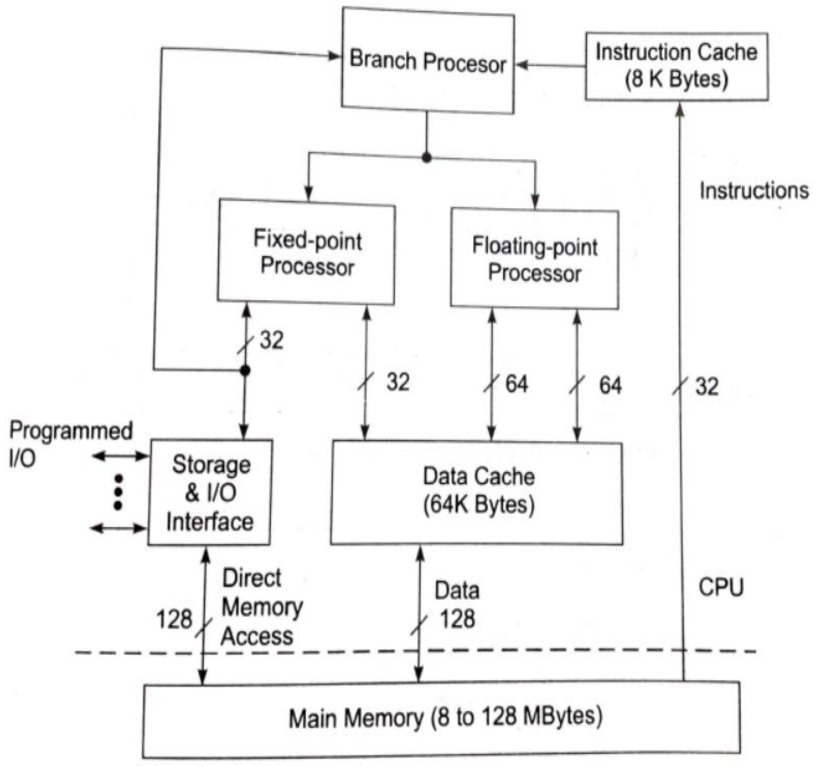


Fig. 4.13 The POWER architecture of the IBM RISC System/6000 superscalar processor (Courtesy of International Business Machines Corporation, 1990)

Fig 2. Superscalar architecture example [source: Briggs, Jotwani]

Superscalar Architecture

The effective CPI is less than RISC scalar processors as it allows several instructions to be issued and completed per clock cycle

Limitation: Hardware cost and complexity of design

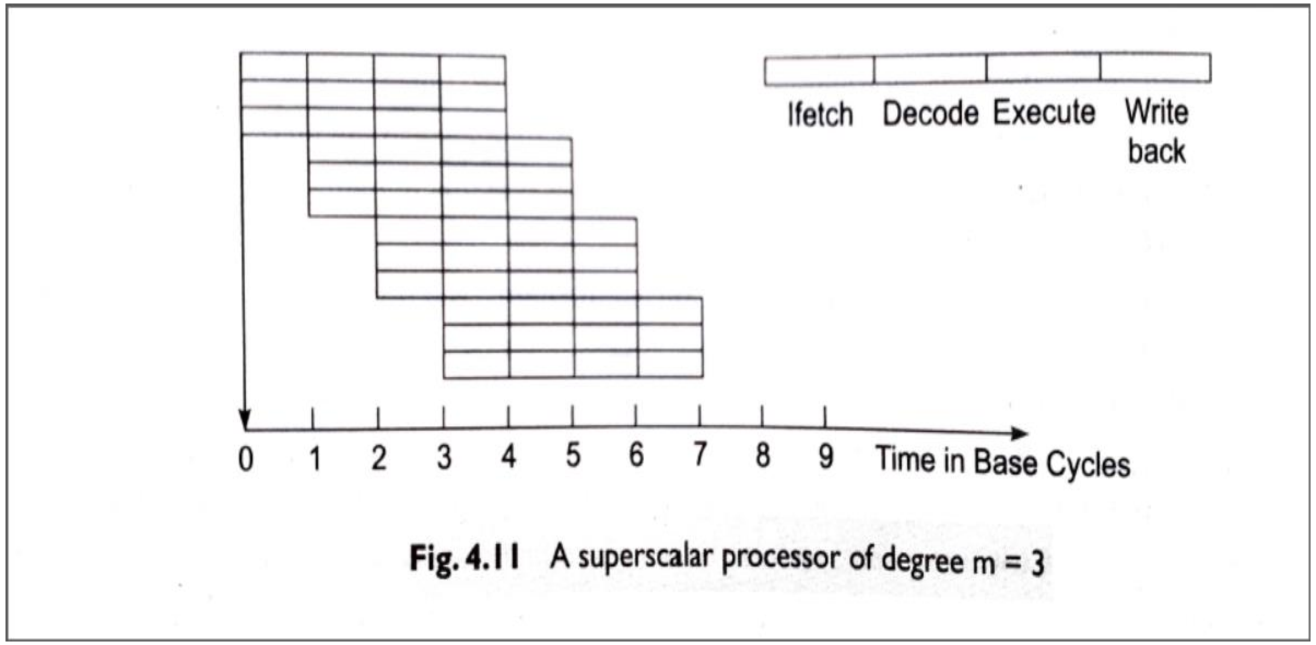


Fig 3. Superscalar processor pipelining [source: Briggs, Jotwani]

VLIW Processor

- The operations to be simultaneously executed are synchronized in a single VLIW instruction
- VLIW (Very Long Instruction Word) Processor rely on compile time detection of parallelism
- VLIW instruction words are hundreds of bits in length
- Multiple functional units run concurrently to perform the simultaneously issued operations
- No hardware needed for run-time detection of parallelism
- Performs well in special purpose computers where program behavior is more predictable

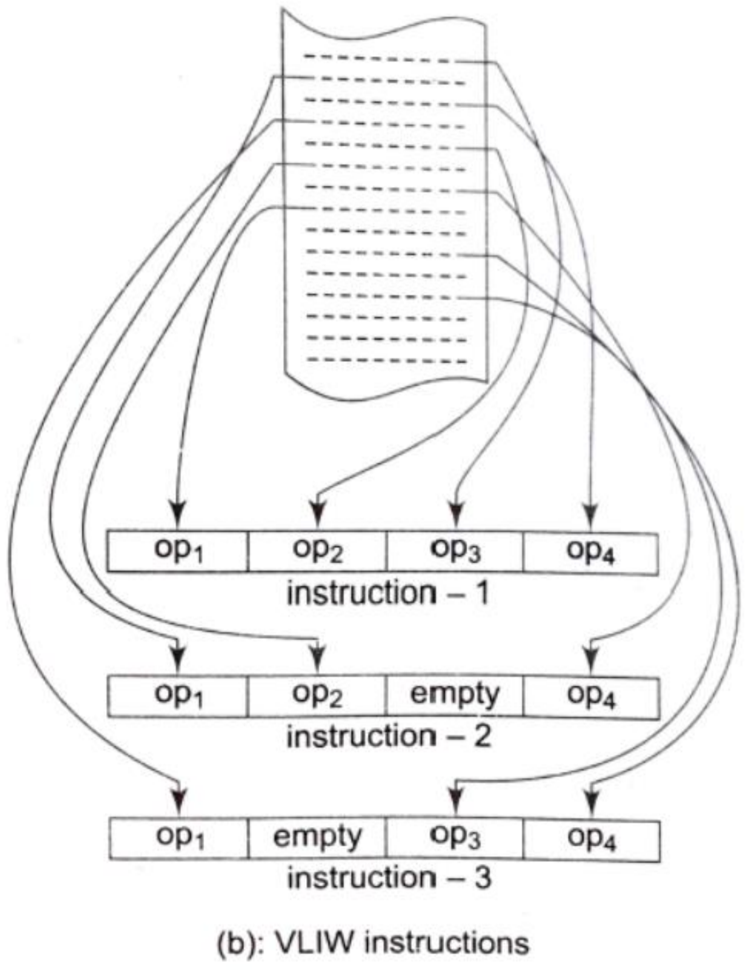


Fig 5. VLIW instruction [source: T. K Ghosh]

VLIW Processor – contd.

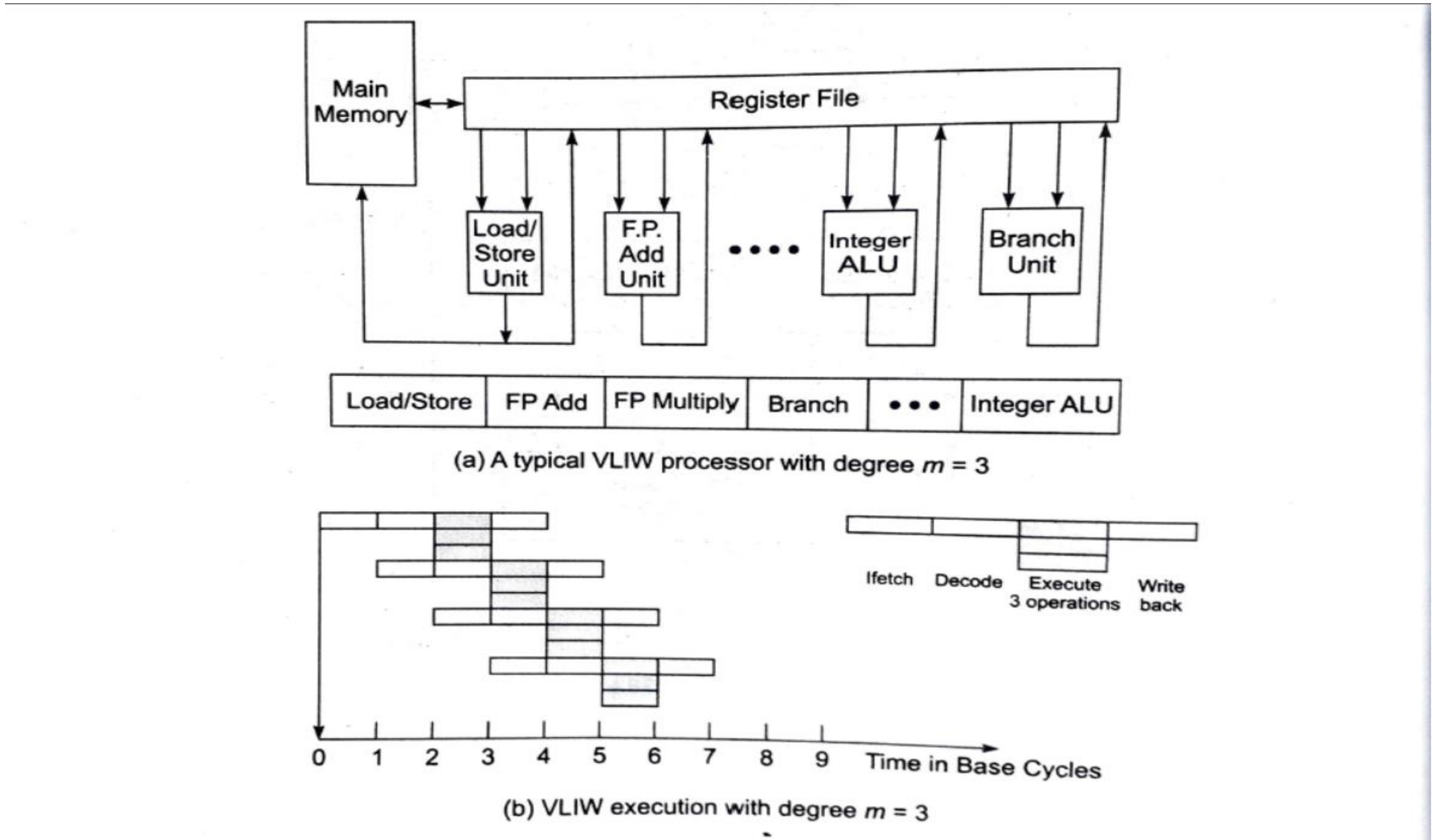


Fig. 4.14 The architecture of a very long instruction word (VLIW) processor and its pipeline operations (Courtesy of Multiflow Computer, Inc., 1987)

Fig 6. VLIW processor architecture [source: Briggs, Jotwani]

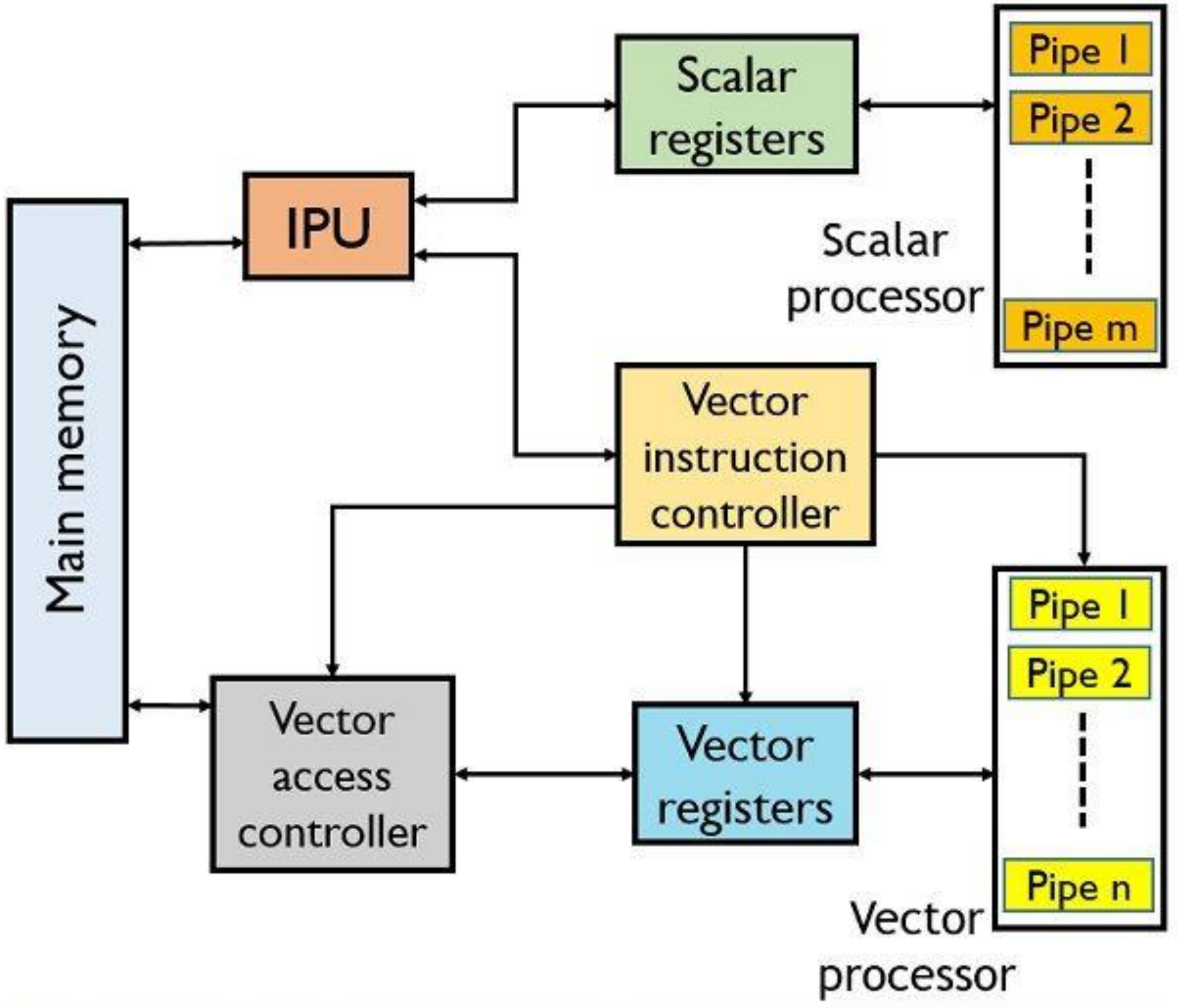
Vector Processor

A vector processor is specially designed to perform vector computations

What is a *vector*?

- A vector operand is an ordered set of same type scalar data items, stored in memory
- The number of data elements in a vector determines the length of vector
- The addressing increment between successive elements is fixed – called vector stride
- To access a vector in memory a vector instruction must specify base address, stride and length.

Vector processing – arithmetic or logical operations on vectors



Functional Diagram of Vector Computer

Electronics Desk

The functional units of a vector computer are as follows:

- •IPU or instruction processing unit
- •Vector register
- •Scalar register
- •Scalar processor
- •Vector instruction controller
- •Vector access controller
- •Vector processor

Vector Instruction Types

(1) Vector-Vector instruction

Unary operation: f_1 : V -> V

Binary operation: f_3: V x V -> V

(2) Vector-Scalar instruction

Unary operation: f_2: V -> S

Binary operation: f_4: V x S -> V

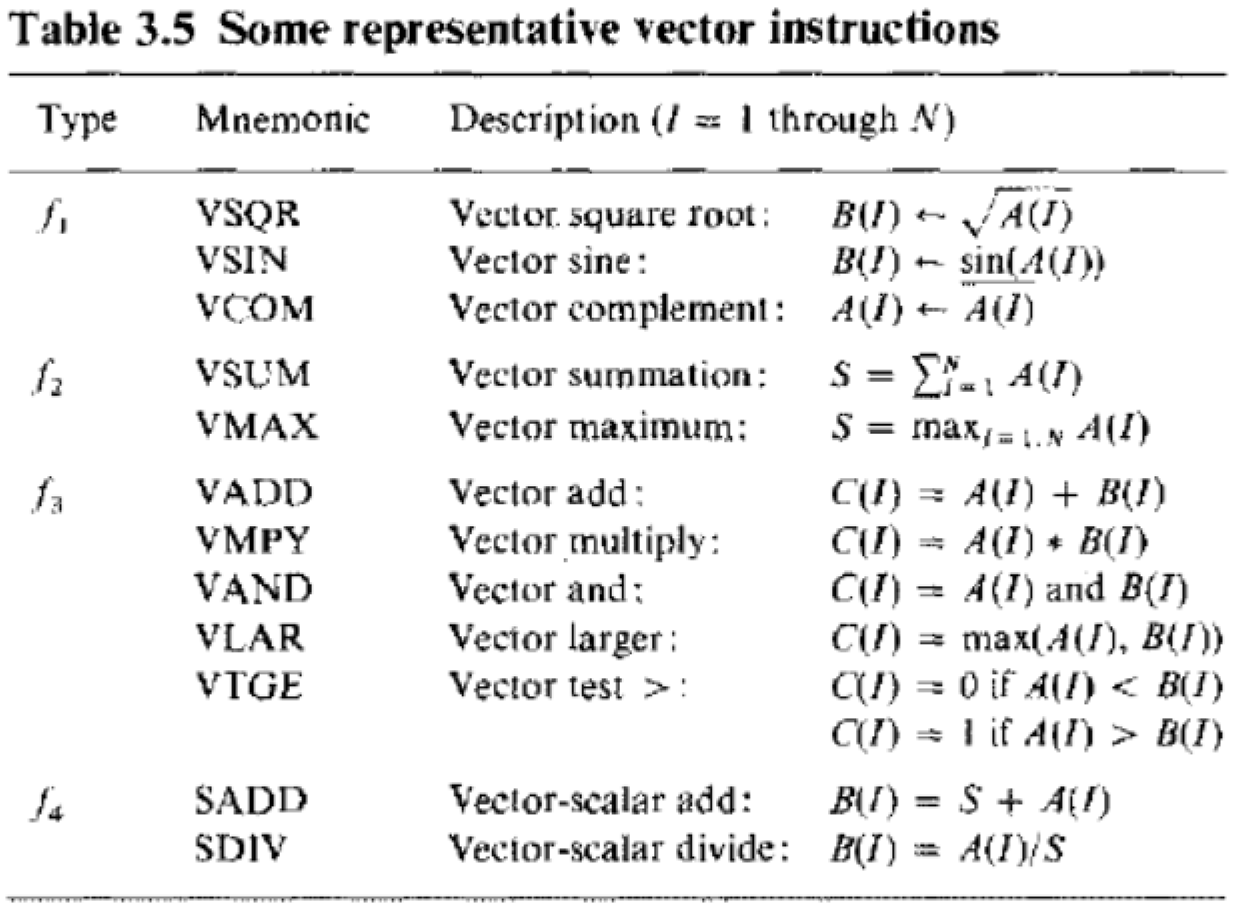
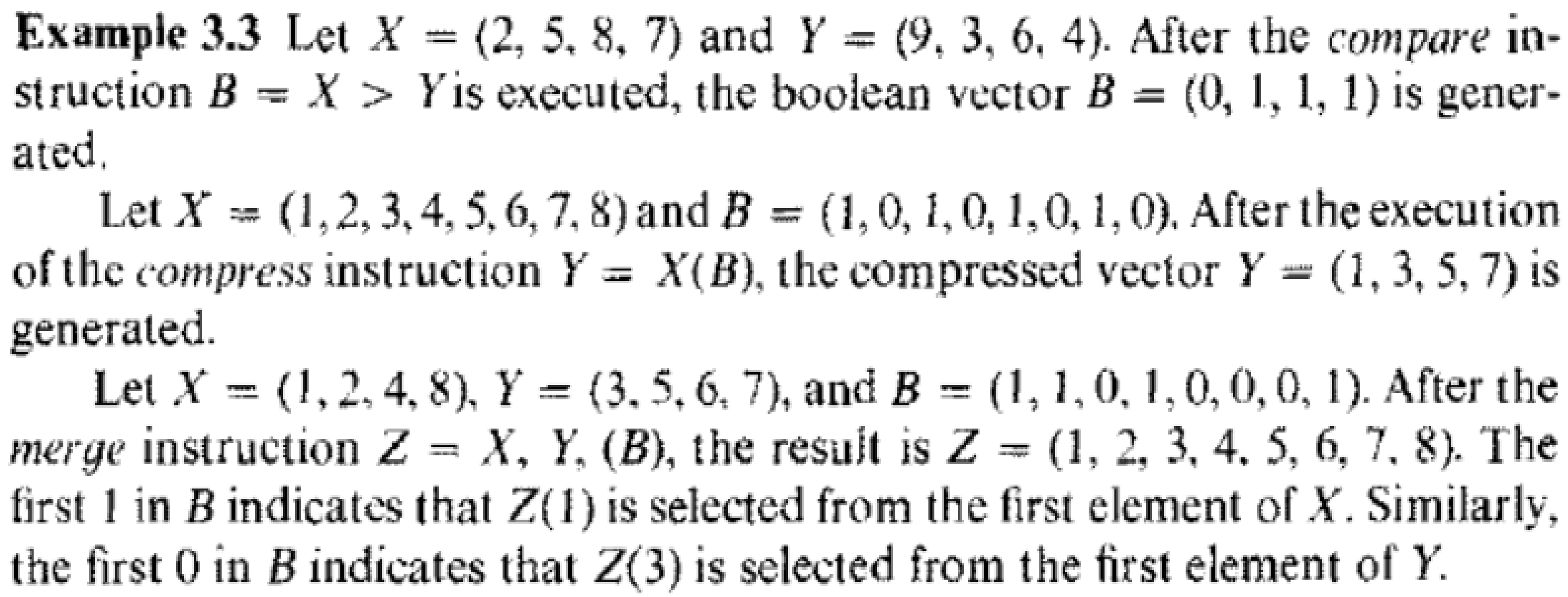


Table 3.5 Some representative vector instructions

Type	Mnemonic	Description (I = 1 through N)	
f_1	VSQR	Vector square root:	$B(I) \leftarrow \sqrt{A(I)}$
	VSIN	Vector sine:	$B(I) \leftarrow \sin(A(I))$
	VCOM	Vector complement:	$A(I) \leftarrow \overline{A(I)}$
f_2	VSUM	Vector summation:	$S = \sum_{I=1}^{N} A(I)$
	VMAX	Vector maximum:	$S = \max_{I=1,N} A(I)$
f_3	VADD	Vector add:	$C(I) = A(I) + B(I)$
	VMPY	Vector multiply:	$C(I) = A(I) * B(I)$
	VAND	Vector and:	$C(I) = A(I)$ and $B(I)$
	VLAR	Vector larger:	$C(I) = \max(A(I), B(I))$
	VTGE	Vector test >:	$C(I) = 0$ if $A(I) < B(I)$
			$C(I) = 1$ if $A(I) > B(I)$
f_4	SADD	Vector-scalar add:	$B(I) = S + A(I)$
	SDIV	Vector-scalar divide:	$B(I) = A(I)/S$

Fig 7. Representative vector instructions [source: Hwang, Briggs]

Vector Instruction Types

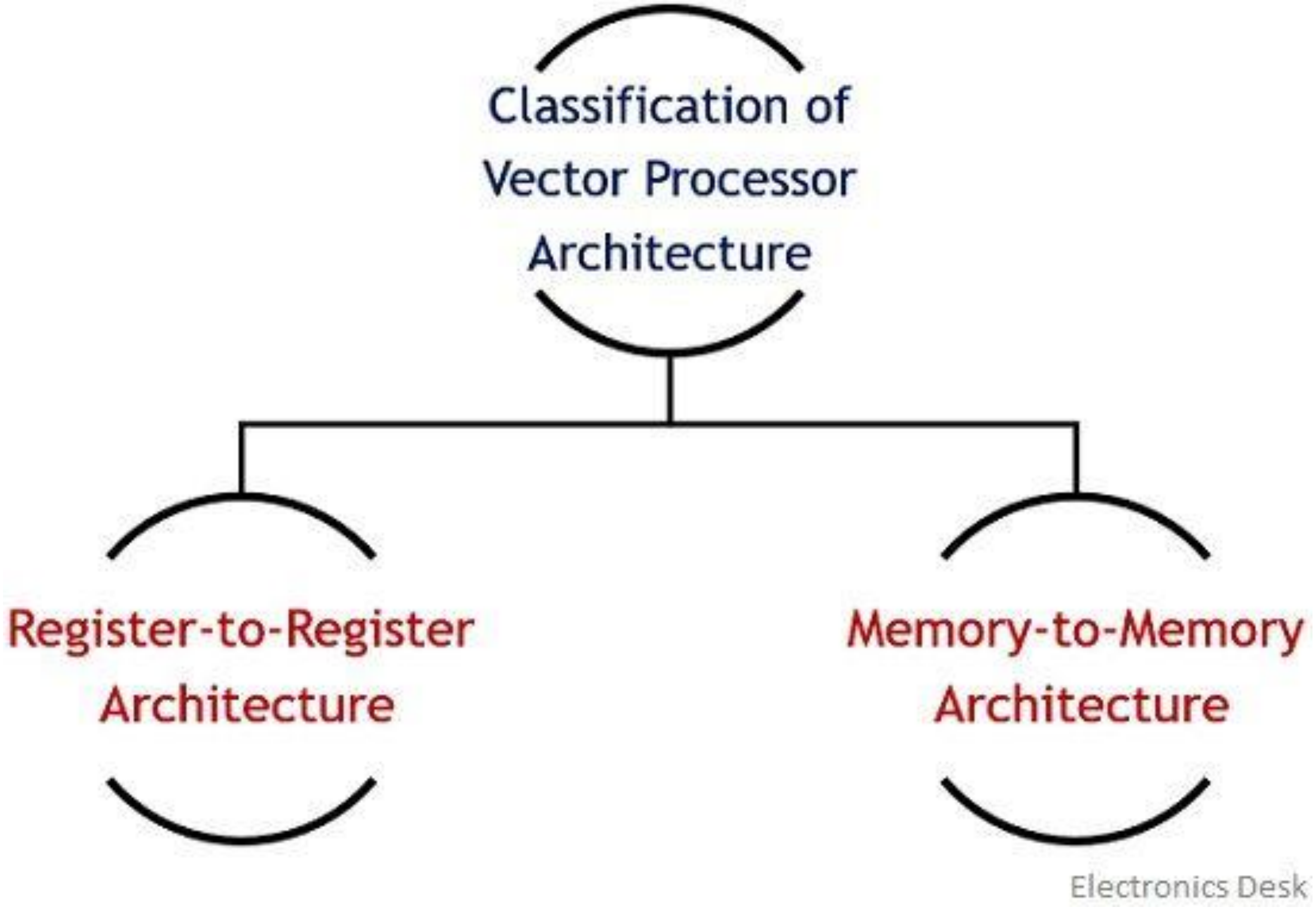


Example 3.3 Let $X = (2, 5, 8, 7)$ and $Y = (9, 3, 6, 4)$. After the *compare* instruction $B = X > Y$ is executed, the boolean vector $B = (0, 1, 1, 1)$ is generated.

Let $X = (1, 2, 3, 4, 5, 6, 7, 8)$ and $B = (1, 0, 1, 0, 1, 0, 1, 0)$. After the execution of the *compress* instruction $Y = X(B)$, the compressed vector $Y = (1, 3, 5, 7)$ is generated.

Let $X = (1, 2, 4, 8)$, $Y = (3, 5, 6, 7)$, and $B = (1, 1, 0, 1, 0, 0, 0, 1)$. After the *merge* instruction $Z = X, Y, (B)$, the result is $Z = (1, 2, 3, 4, 5, 6, 7, 8)$. The first 1 in B indicates that $Z(1)$ is selected from the first element of X. Similarly, the first 0 in B indicates that $Z(3)$ is selected from the first element of Y.

Fig 8. Examples of special vector operations [source: Hwang, Briggs]


Classification of Vector Processor Architecture
Register-to-Register Architecture
Memory-to-Memory Architecture
Electronics Desk

Vector Processor Classification

A vector processor is an ensemble of

vector registers, functional pipelines, processing elements and register counters.

Vector processors can be classified as:

- Memory-to-memory architecture – source operands, intermediate and final results are retrieved directly from the main memory

- Register-to-register architecture – operands and results are retrieved indirectly from the main memory through vector register files and/or scalar registers

Vector Processor Architecture

IPU : Fetches and decodes scalar and vector instructions

Vector Instruction Controller: supervises execution of vector instructions

Vector access controller: fetches vector operands from memory

Vector register: Used to close up the speed gap between memory access and vector processor

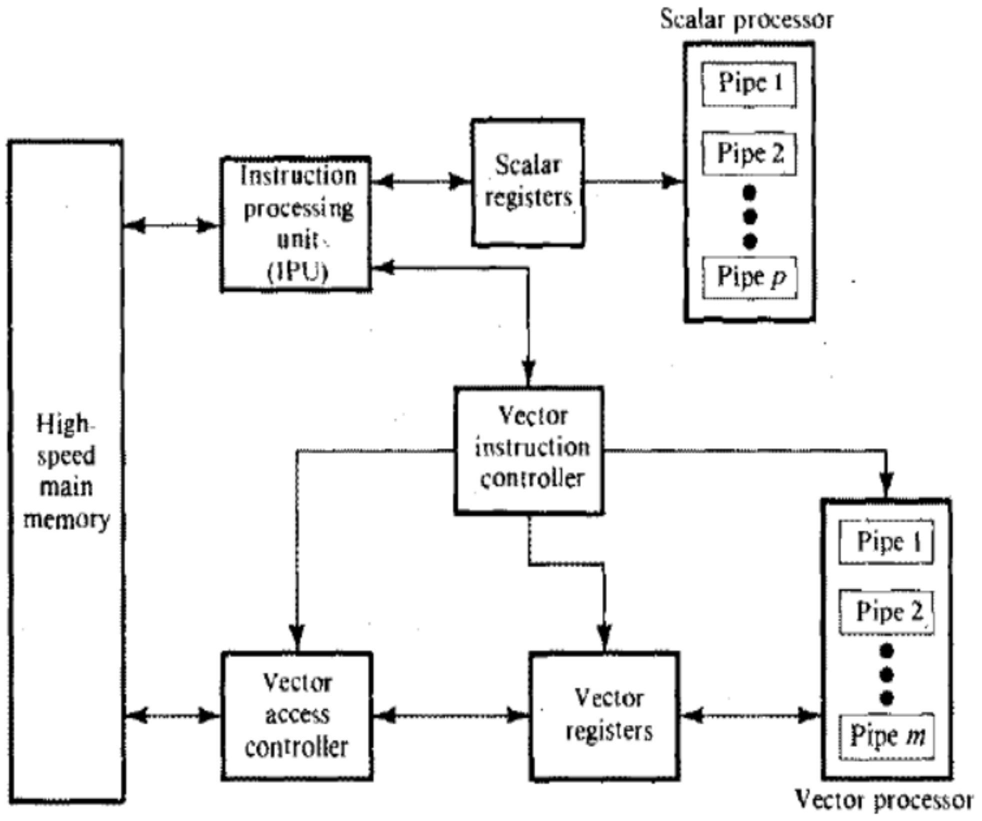


Figure 3.43 The architecture of a typical vector processor with multiple functional pipes.

Fig 9. Vector processor architecture [source: Hwang, Briggs]

Vectorizing Compiler

A vectorizing compiler can take a sequential (scalar) program and create a set of vector instructions by converting language constructs like loop, recurrence computation etc.

The conversion from scalar code to vector code is called vectorization.

E.g. Vectorization of For loop in scalar code:

For i=1 to N do A(i) = B(i) * C(i)

Equivalent Vectorized code: A[1:N] = B[1:N] * C[1:N]

SIMD Array Processors

An array processor is a synchronous array of parallel processors and consists of multiple processing elements (PEs) under the supervision of one control unit (CU).

Can handle single instruction, multiple data (SIMD) streams.

SIMD machines are especially designed to perform vector computations over large matrices or arrays of data.

Architectural Configuration of Array Processor

Configuration 1: Illiac IV configuration

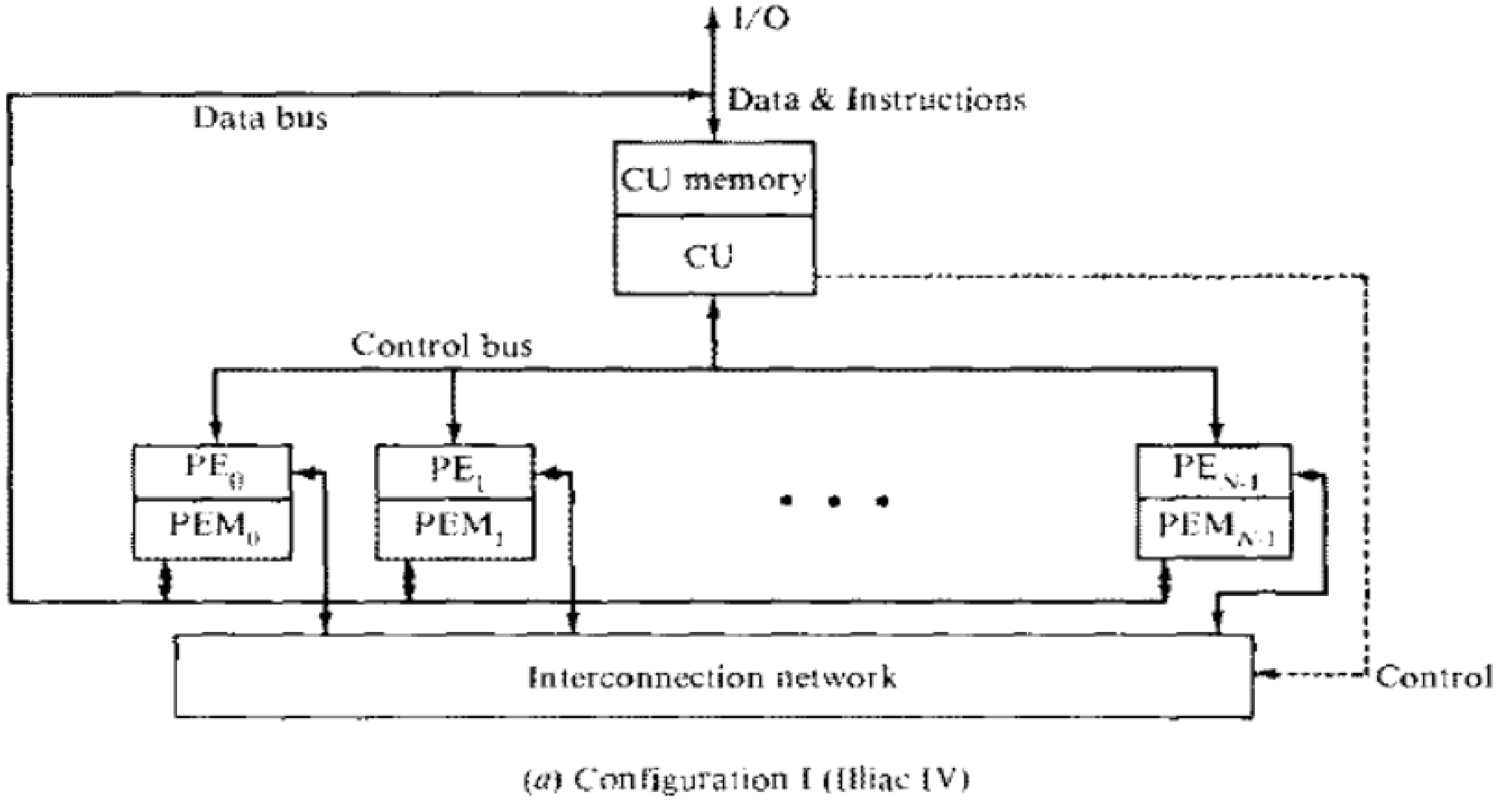
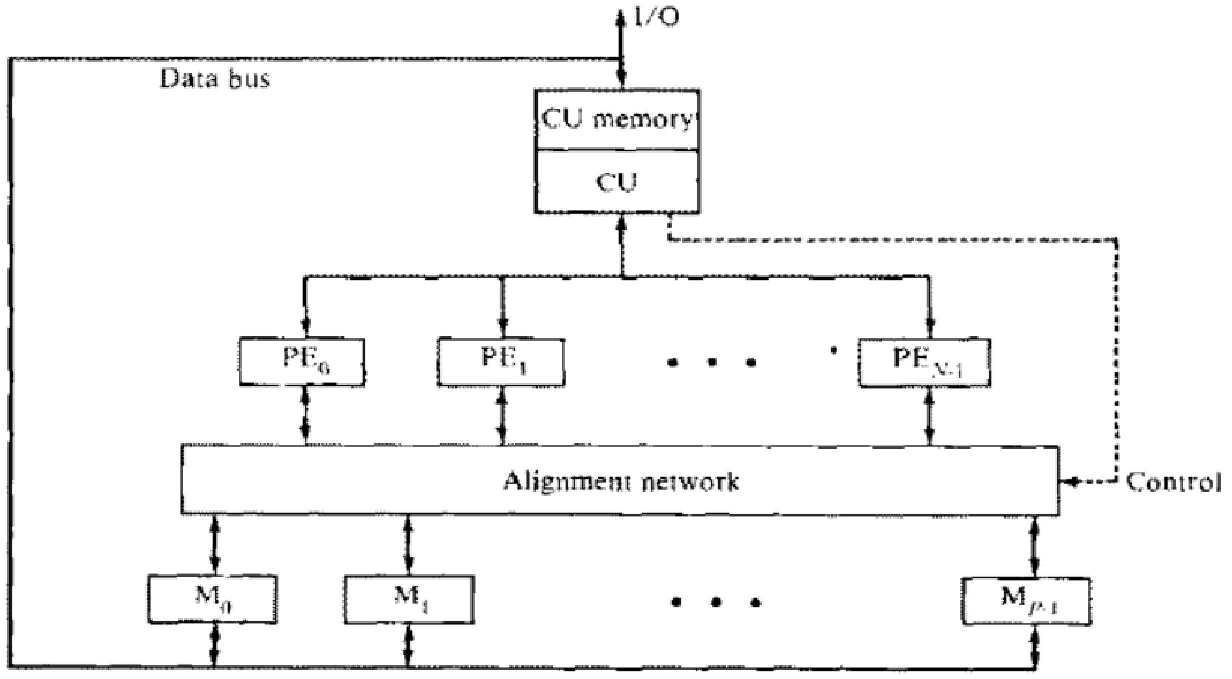


Fig 10. Illiac IV configuration [source: Hwang, Briggs]

Architectural Configuration of Array Processor – contd.



(*b*) Configuration II (BSP)

Figure 5.1 Architectural configurations of SIMD array processors.

Configuration 2: BSP configuration

Fig 11. BSP (Burroughs' Scientific Processor) configuration [source: Hwang, Briggs]

System Design Goals

- Important criteria:
  - Theoretical peak performance of processor
  - System performance under realistic load conditions
  - Scalability
  - Price
  - Usability
  - Reliability
  - Power consumption
  - Physical size
- There should be no performance bottlenecks in the system
  - Performance bottleneck arises when one of the subsystems cannot keep up with the overall throughput of the system

Instruction Pipeline

Instruction execution can be decomposed into a linear, ordered sequence of operations –

- Typically, (i) Fetch, (ii) Decode, (iii) Operand Fetch, (iv) Execute, (v) Write-Back
- Each of these operations can be performed in its specific stage hardware at its designated clock-cycle.
- So, these operations can be performed in an overlapped fashion in a 5-stage pipeline

In an instruction stream, instructions execution is primarily sequential

So, the same sequence of operations are expected to occur repetitively

In an instruction pipeline, a stream of instruction can be executed in an overlapped manner.