SIMD Processors
(Vector Processing)

P. K. Roy
Asst. Professor
Siliguri Institute of Technology

Problems with conventional approach

Limits to conventional exploitation of ILP:

1) *pipelined clock rate: at some point, each* increase in clock rate has corresponding CPI increase (branches, other hazards)

2) *instruction fetch and decode: at some* point, its hard to fetch and decode more instructions per clock cycle

3) *cache hit rate: some long-running* (scientific) programs have very large data sets accessed with poor locality; others have continuous data streams (multimedia) and hence poor locality

Supercomputers

- Fastest machine in world at given task
- A device to turn a compute-bound problem into an I/O bound problem
- Any machine costing $30M+
- Any machine designed by Seymour Cray

CDC6600 (Cray, 1964) regarded as first supercomputer

Supercomputer Applications

- Typical application areas
  - Military research (nuclear weapons, cryptography)
  - Scientific research
  - Weather forecasting
  - Oil exploration
  - Industrial design (car crash simulation)

- All involve huge computations on large data sets

- *In 70s-80s, Supercomputer ≡ Vector Machine*

Why Vector Processors?

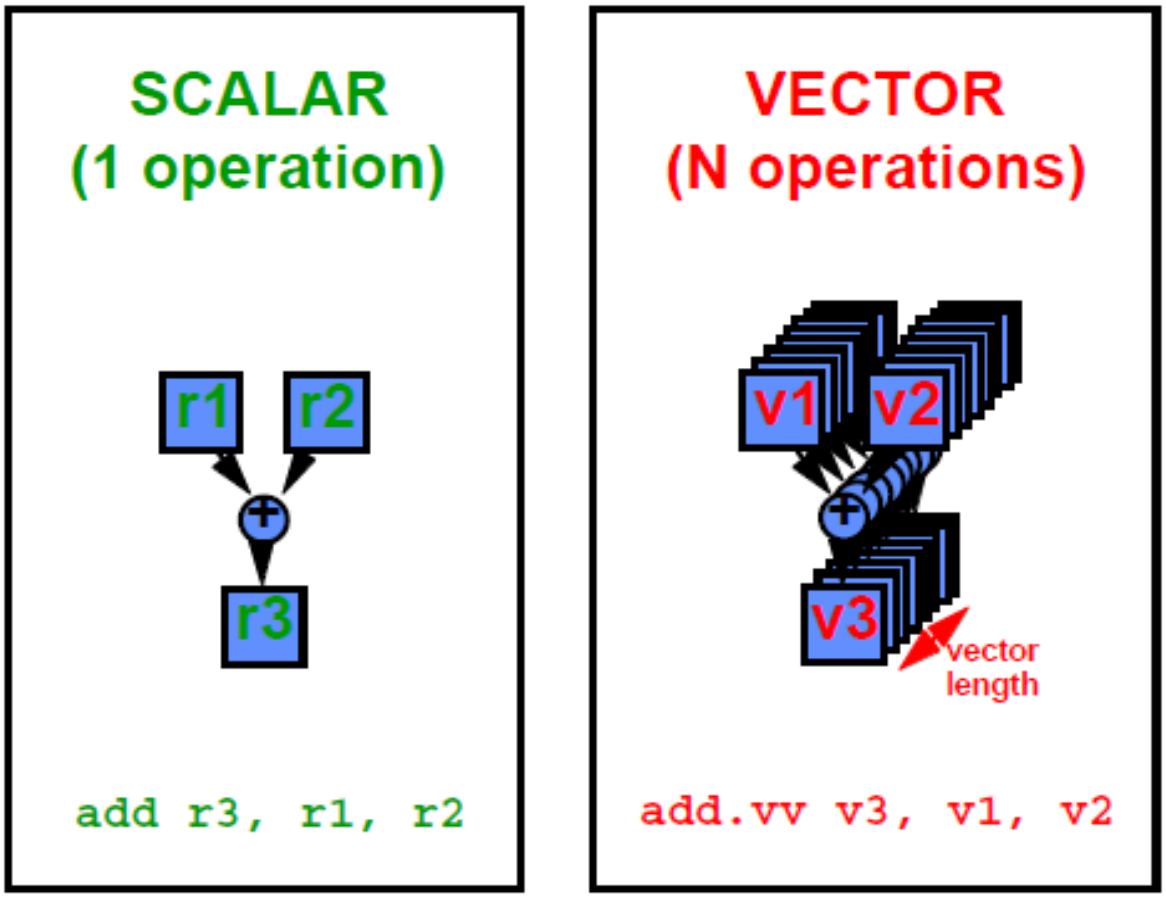
- A single vector instruction specifies a great deal of work—it is equivalent to executing an entire loop.
- The computation of each result in the vector is independent of the computation of other results in the same vector and so hardware does not have to check for data hazards within a vector instruction.
- Hardware need only check for data hazards between two vector instructions once per vector operand, not once for every element within the vectors.
- Vector instructions that access memory have a known access pattern.
- Because an entire loop is replaced by a vector instruction whose behavior is predetermined, control hazards that would normally arise from the loop branch are nonexistent.

Vector Processing

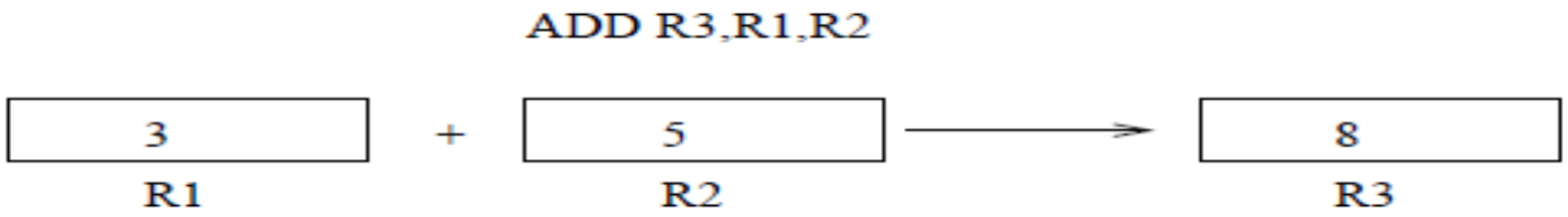
- Manipulation of arrays or vectors is a common operation in scientific and engineering applications.
- Typical operations of array-oriented data include:
  - Processing one or more vectors to produce a scalar result.
  - Combining two vectors to produce a third one.
  - Combining a scalar and a vector to generate a vector.
  - A combination of the above three operations.
- Two architectures suitable for vector processing are:
  - Pipelined **vector processors**
    - Implemented in many supercomputers
  - Parallel **array processors**
- Compiler does some of the difficult work of finding parallelism, so the hardware doesn't have to.
  - Data parallelism.

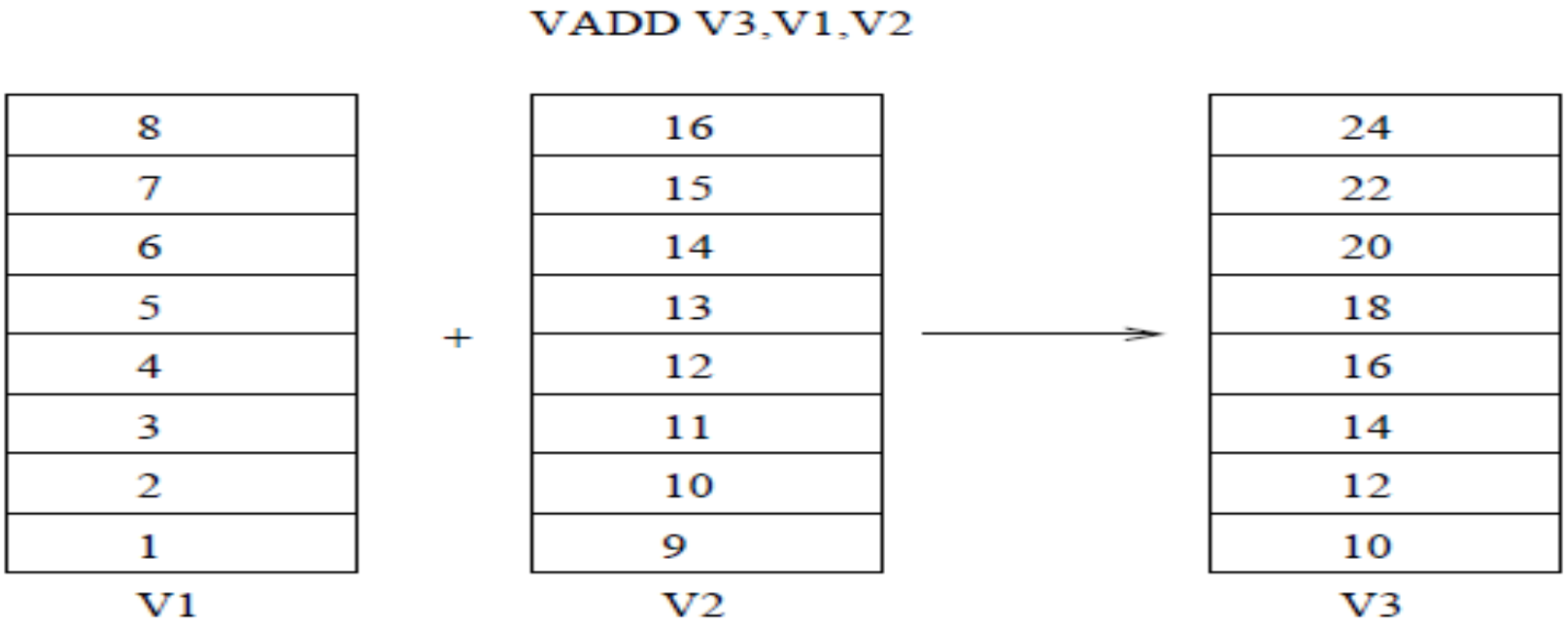
Vector Processor(1)

- Vector processors have high-level operations that work on linear arrays of numbers: "vectors"



Vector Processor(2)





Difference between scalar and vector add instructions

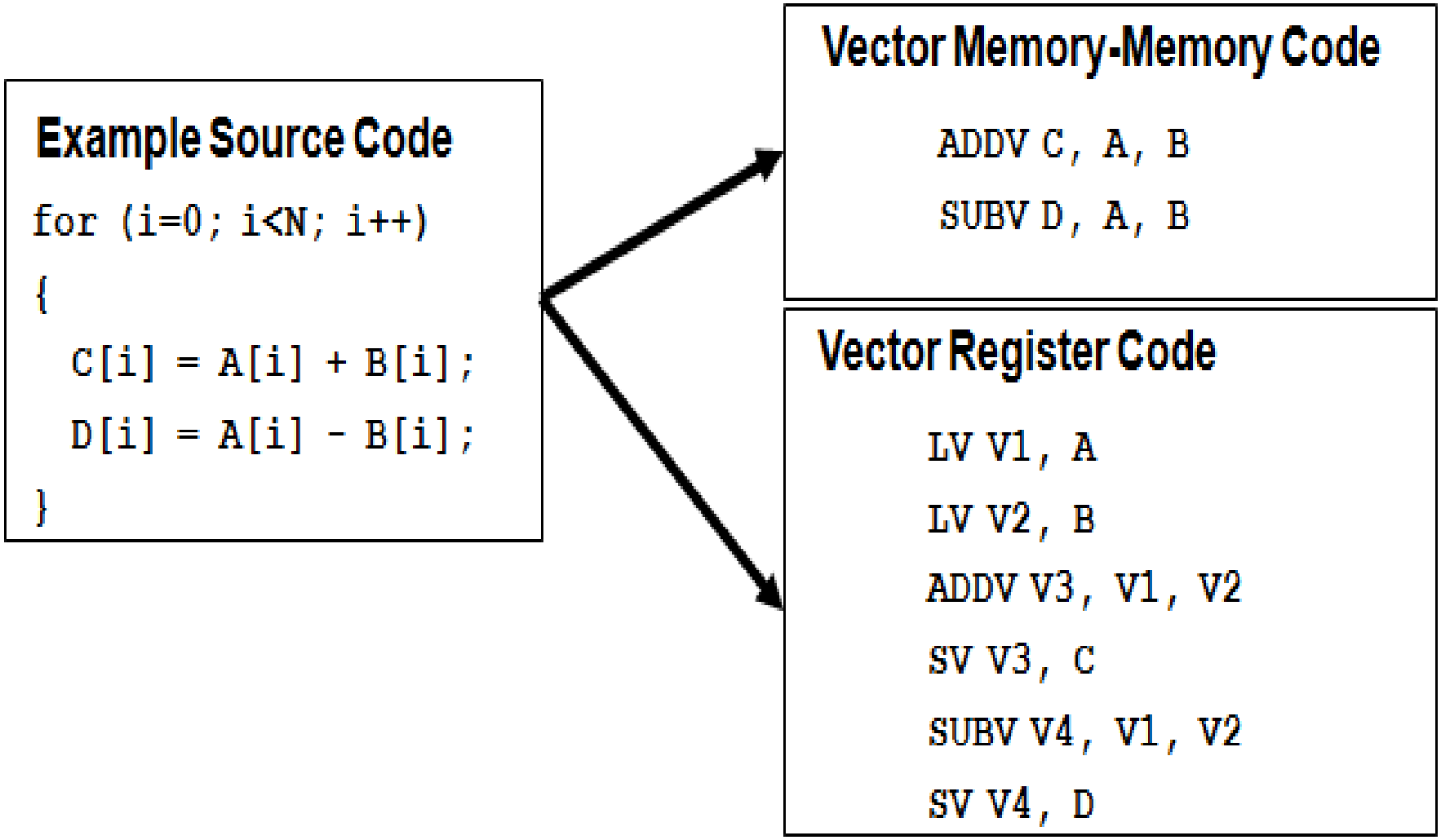
Vector Processor(3)

- Strictly speaking, vector processors are not parallel processors.
  - They only behave like SIMD computers.
- There are not several CPUs in a vector processor, running in parallel.
  - They are SISD processors with vector instructions executed on pipelined functional units.
- Vector computers usually have vector registers which can each store 64 to 128 values.
- Vector instructions examples:
  - Load vector from memory into vector register
  - Store vector into memory
  - Arithmetic and logic operations between vectors
  - Operations between vectors and scalars

Self-Review : Vector instructions from *Kai Hwang*

- The programmers are allowed to use operations on vectors in the programs, and the compiler translates these operations into vector instructions at machine level.

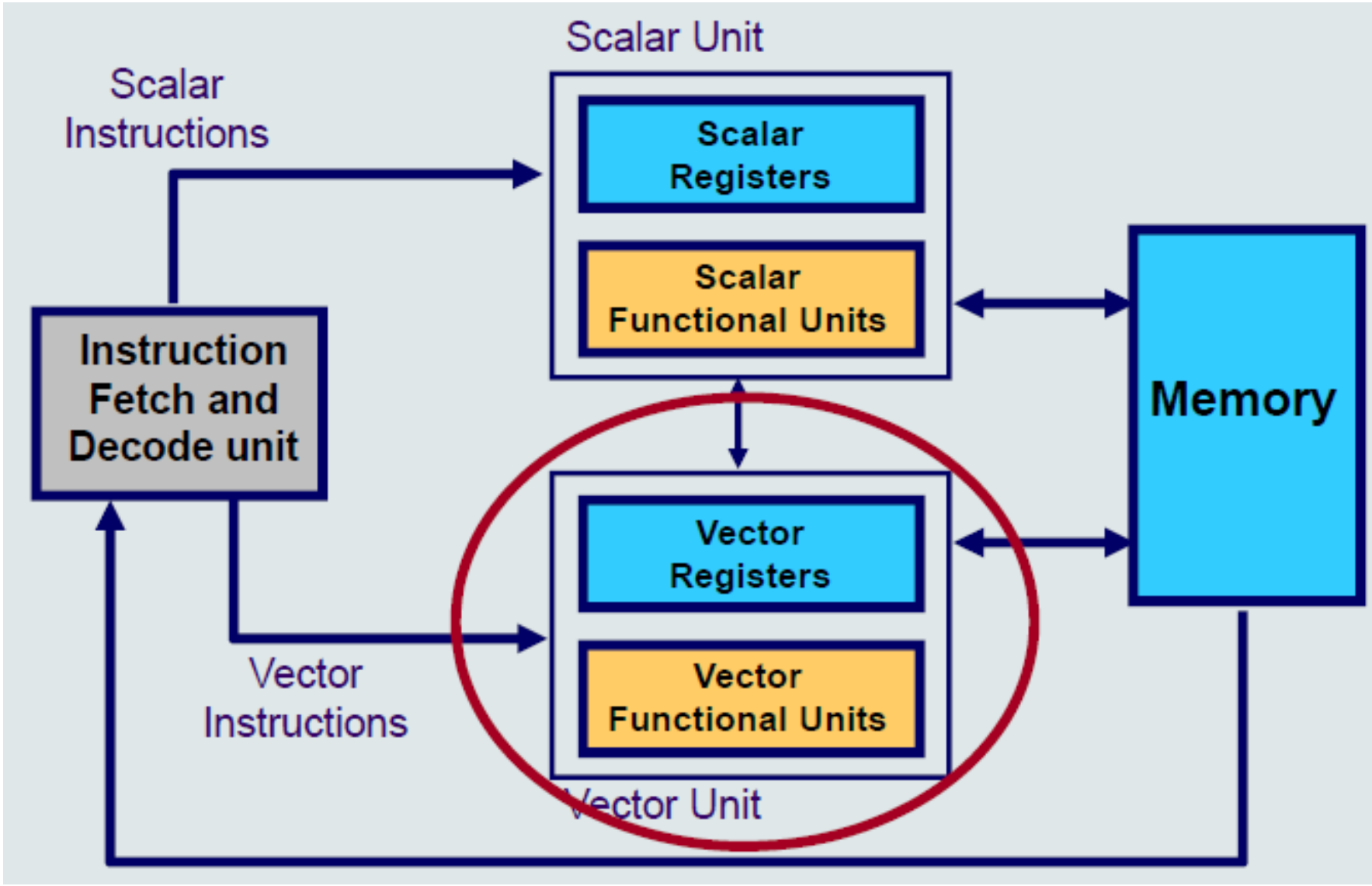
Vector Processor(4)



Vector Architecture(1)

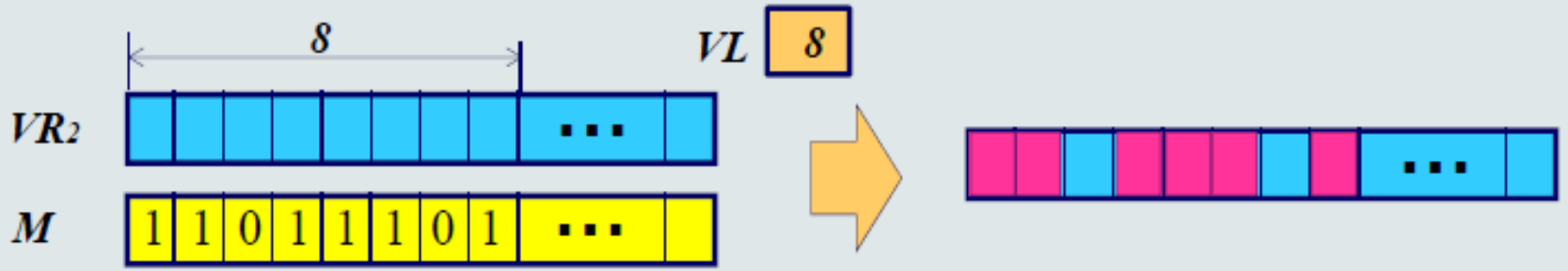
- Basic idea:
  - Read sets of data elements into “vector registers”
  - Operate on those registers
  - Disperse the results back into memory
- Registers are controlled by compiler
  - Register files act as compiler controlled buffers
  - Used to hide memory latency
  - Leverage memory bandwidth
- Vector loads/stores deeply pipelined
  - pay for memory latency once per vector ld/st!
- Regular loads/stores
  - pay for memory latency for each vector element

Vector Architecture(2)

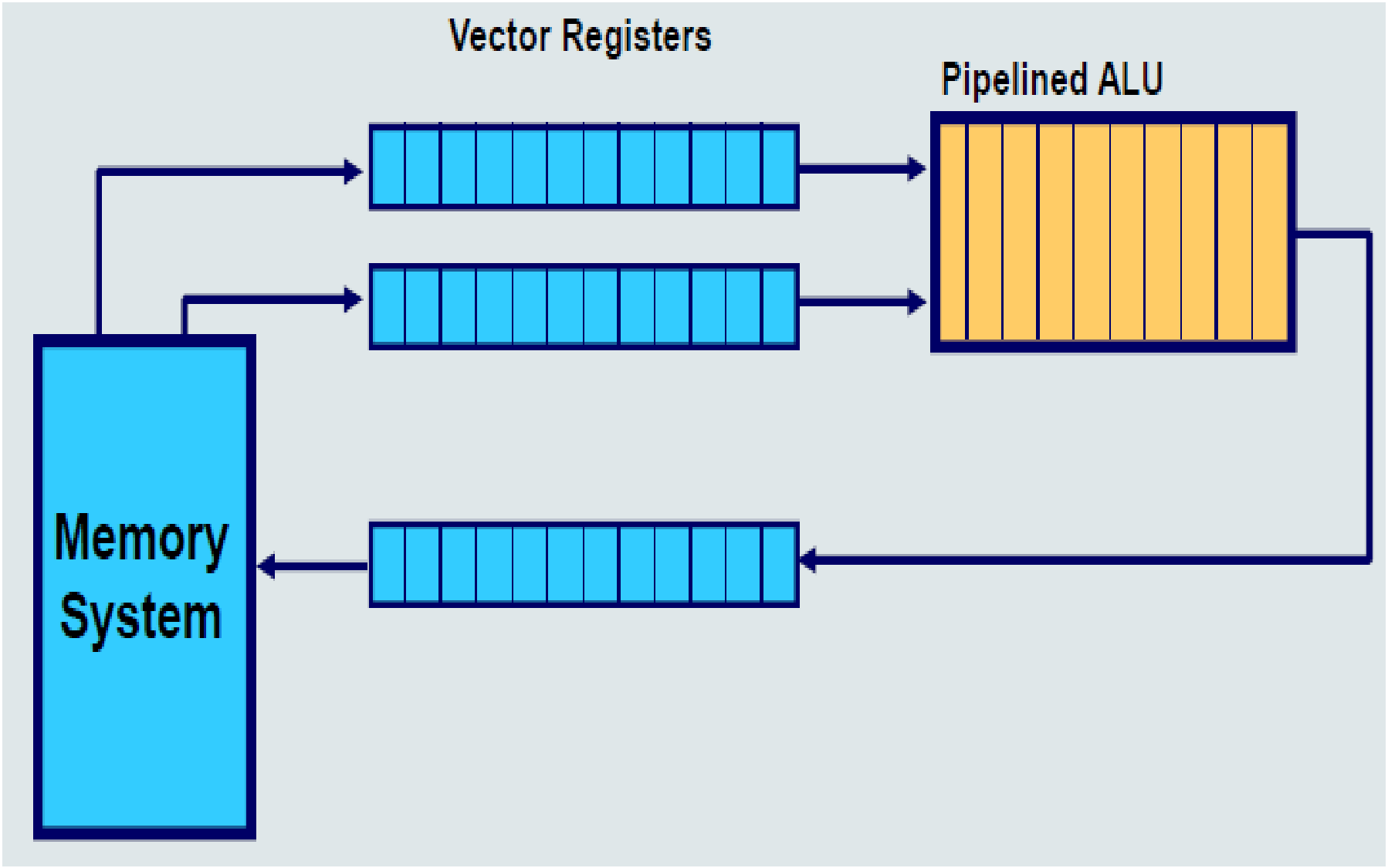


Vector Unit

- A vector unit consists of a pipelined functional unit, which perform ALU operation of vectors in a pipeline.
- It has also vector registers, including:
  - A set of general purpose vector registers, each of length *s* (e.g., 128)
  - A vector length register *VL*, which stores the length *l* ($0 \leq l \leq s$) of the currently processed vector(s);
  - A mask register *M*, which stores a set of *l* bits, one for each element in a vector, interpreted as Boolean values;
    - Vector instructions can be executed in masked mode so that vector elements corresponding to a false value in *M* are ignored.



Vector Unit Operation Model



Vector Program

- Consider an element-by-element addition of two *N*-element vectors **A** and **B** to create the sum vector **C**.

- On an SISD machine, this computation will be implemented as:

```
for i = 0 to N-1 do
        C[i] := A[i] + B[i];
```

  - There will be *N*K* instruction fetches (assuming that *K* instructions are needed for each iteration) and *N* additions.
  - There will also be *N* conditional branches, if loop unrolling is not used.

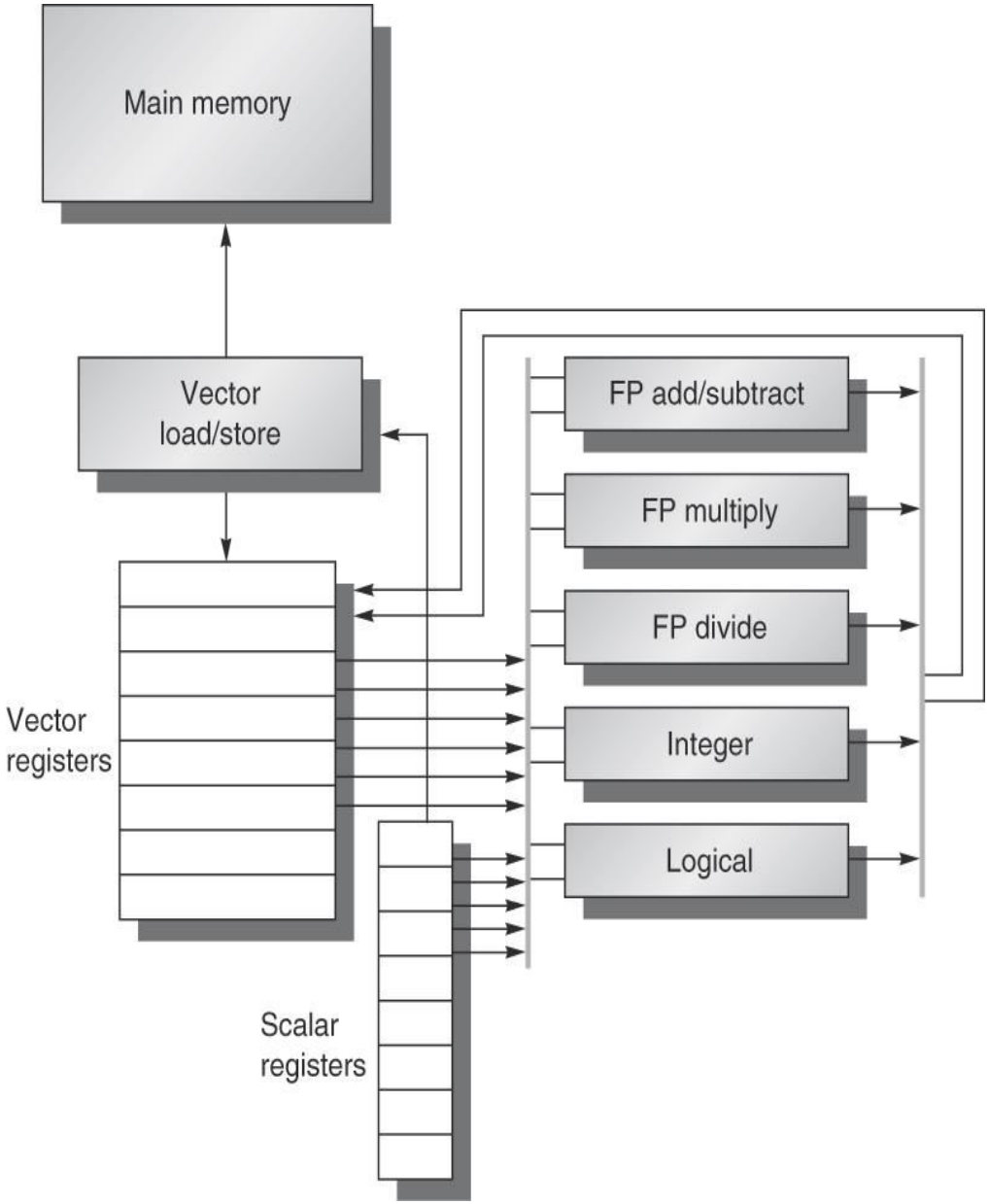
- A compiler for a vector computer generates something like:

```
C[0:N-1] ← A[0:N-1] + B[0:N-1];
```

  - Even though N additions will still be performed, there will only be K' instruction fetches (e.g., Load A, Load B, Add_vector, Write C → 4 instructions).
  - No conditional branch is needed.

Example: VMIPS

- Vector registers
  - Each register holds a 64-element, 64 bits/element vector
  - Register file has 16 read ports and 8 write ports
- Vector functional units
  - Fully pipelined
  - Data and control hazards are detected
- Vector load-store unit
  - Fully pipelined
  - Words move between registers
  - One word per clock cycle after initial latency
- Scalar registers
  - 32 general-purpose registers
  - 32 floating-point registers



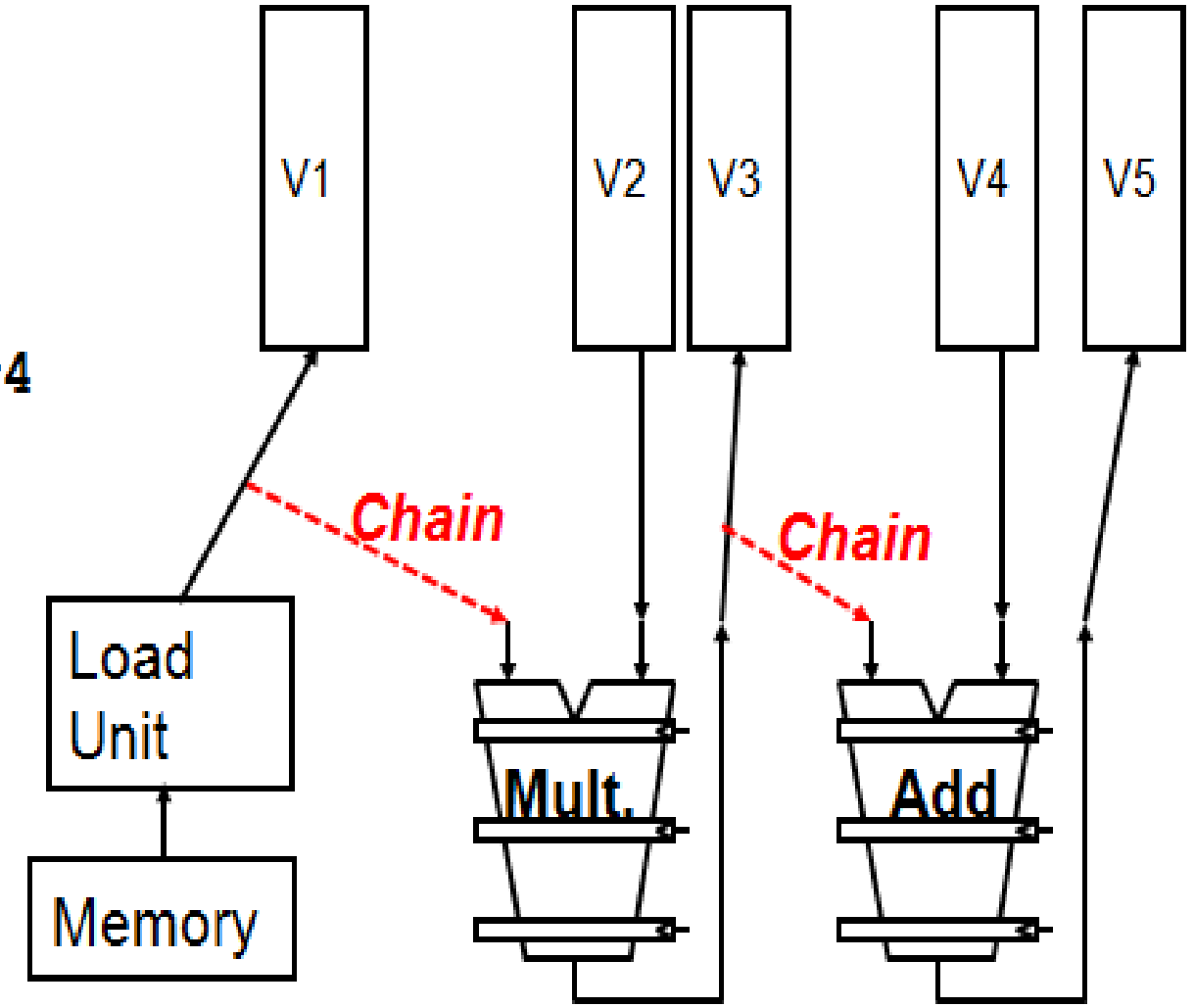
Vector Chaining(1)

- Vector version of register bypassing

- Chaining
  - Allows a vector operation to start as soon as the individual elements of its vector source operand become available
    - Results from the first functional unit are forwarded to the second unit

Vector Chaining(2)

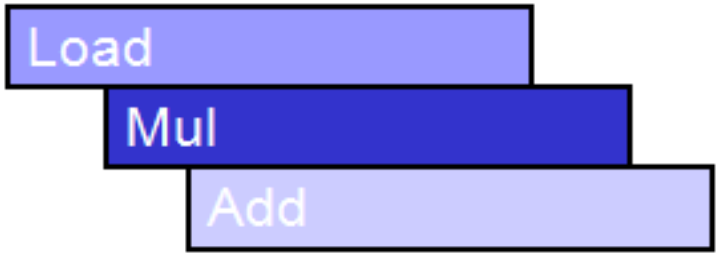
```
LV   v1
MULV v3,v1,v2
ADDV v5,v3, v4
```



Vector Chaining(3)

- Without chaining, must wait for last element of result to be written before starting dependent instruction

- With chaining, can start dependent instruction as soon as first result appears



Advantages of Vector Processing

- Each result is independent of previous results - allowing deep pipelines and high clock rates.
- A single vector instruction performs a great deal of work - meaning less fetches and ewer branches (and in turn fewer mispredictions).
- Vector instructions access memory a block at a time which allows memory latency to be amortized over many elements.
- Vector instructions access memory with known patterns, which allows multiple memory banks to simultaneously supply operands.

- Less memory access = faster processing time.

Disvantages of Vector Processing

- Not as fast with scalar instructions
- Complexity of the multi-ported VRF
- Difficulties implementing precise exceptions
- High price of on-chip vector memory systems
- Increased code complexity

Array Processors

Array processor(1)

- A vector (one dimensional array)
- A set of vectors (multi-dimensional array)
- Array processor ─ a processor capable of processing array elements
- Suitable for scientific computations involving two dimensional matrices
- Array processor performs a single instruction in multiple execution units in the same clock cycle
- The different execution units have same instruction using same set of vectors in the array

Array processor(2)

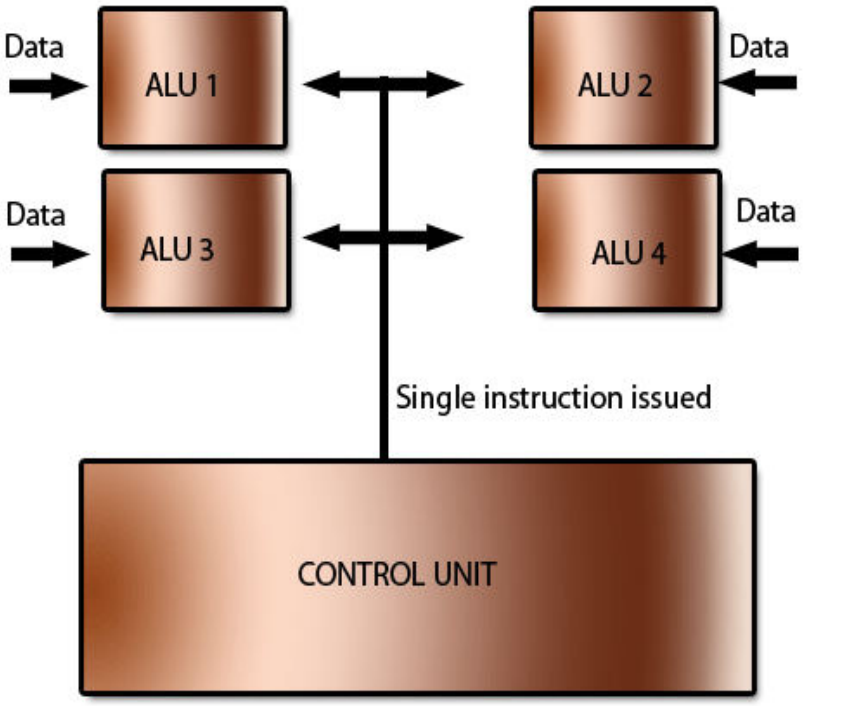
- Array processor is a synchronous parallel computer with multiple ALU called processing elements (PE) that can operate in parallel in lock step fashion.
- It is composed of N identical PE under the control of a single control unit and a number of memory modules.
- Array processors also frequently use a form of parallel computation called pipelining where an operation is divided into smaller steps and the steps are performed simultaneously.
- It can greatly improve performance on certain workloads mainly in numerical simulation.

- Processing units and memory elements communicate with each other through an interconnection network.
  - Different topologies can be used.

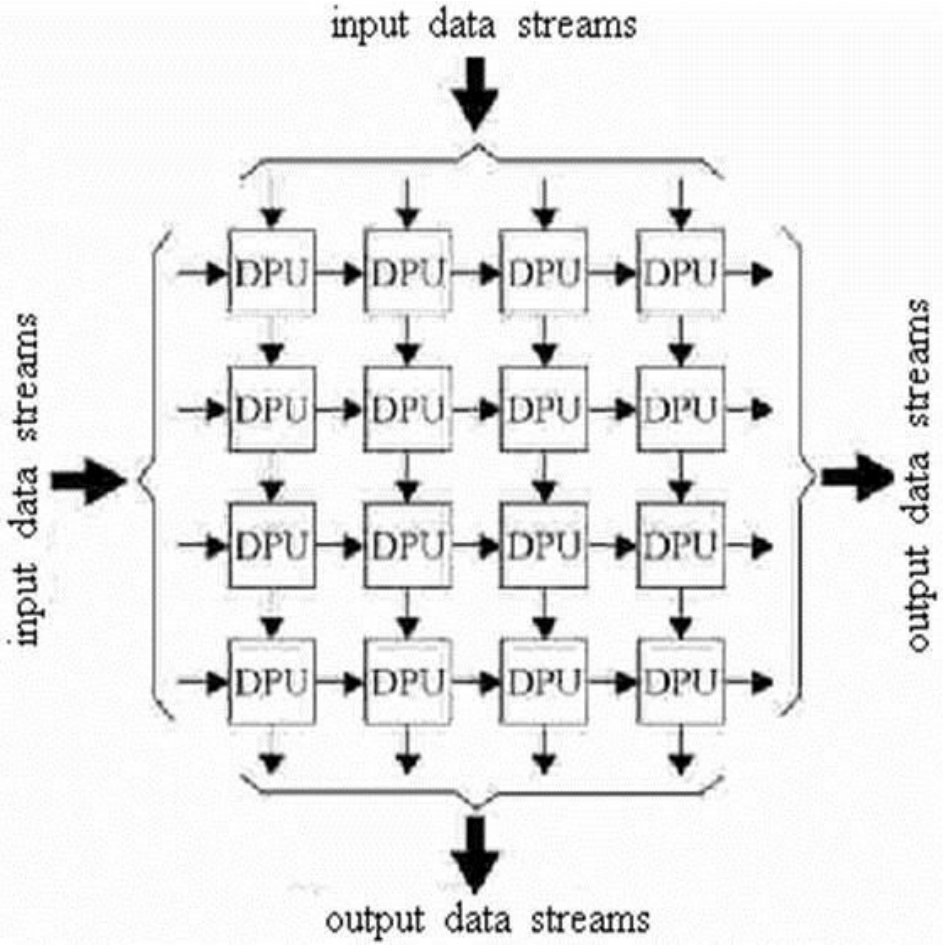
Array processor(3)

- In order to reduce the amount of time this takes, most modern CPUs use a technique known as instruction pipelining in which the instructions pass through several sub-units in turn.
- Array processors take this concept one step further. Instead of pipelining just the instructions, they also pipeline the data itself. This allows for significant savings in decoding time.

Array processor(4)



An array processor - a Single Instruction Multiple Data computer



How Array Processor Can Help?

- Consider the simple task of adding two groups of 10 numbers together. In a normal programming language you might have done something as
- **execute this loop 10 times**
  - **read the next instruction and decode it**
  - **fetch this number fetch that number**
  - **add them**
  - **put the result here**
- **End loop**
- But to an array processor this tasks looks as
- **read instruction and decode it**
- **fetch these 10 numbers**
- **fetch those 10 numbers**
- **add them**
- **put the results here**

How Array Processor Can Help?

There are several savings inherent in this approach.(Based on the example in previous slide)

A. Only two address translations are needed

B. Fetching and decoding the instruction is done only one time instead of ten times

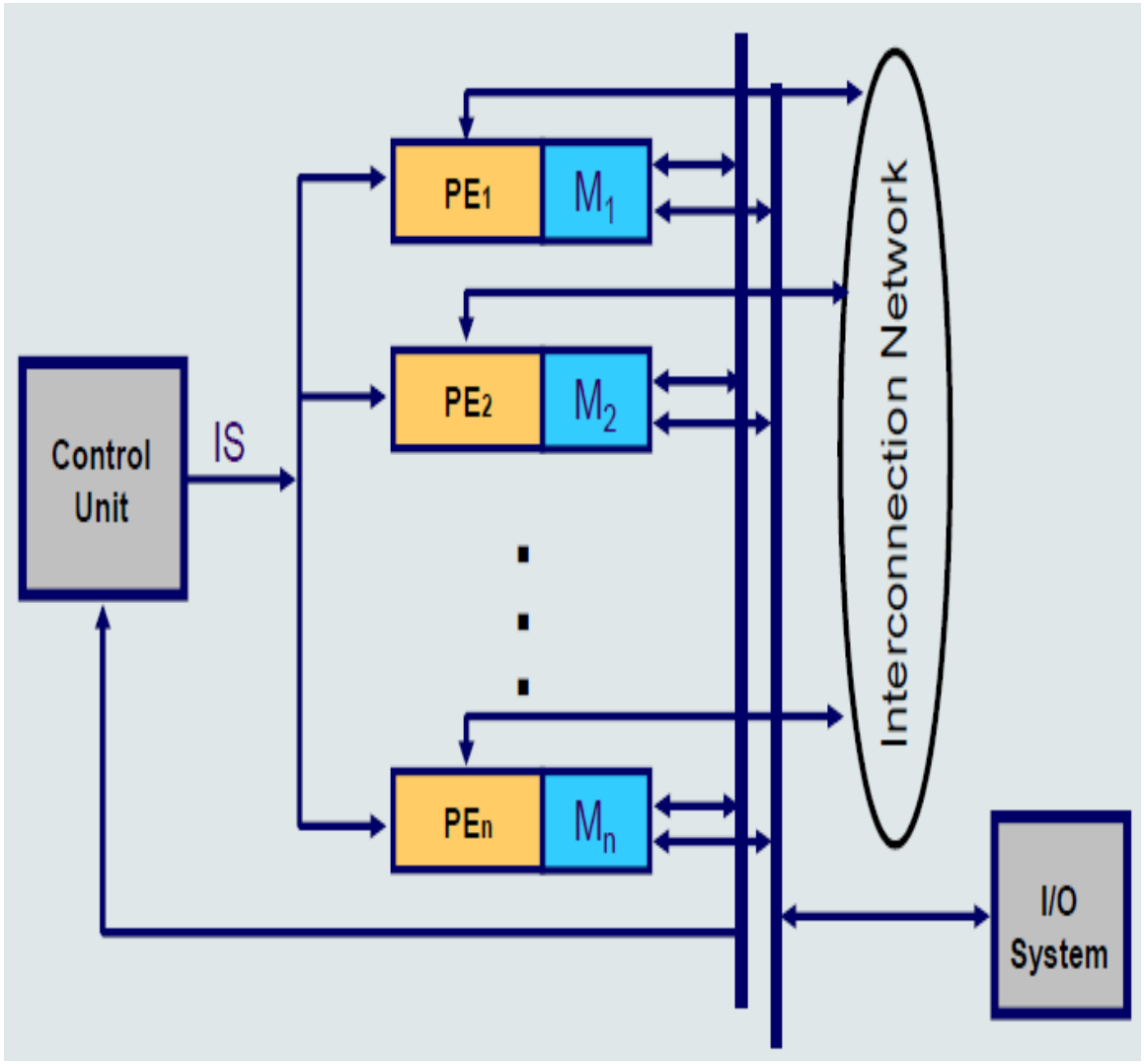
C. The code itself is also smaller, which can lead to more efficient memory use.

D. It improve performance by avoiding stalls.

Classification

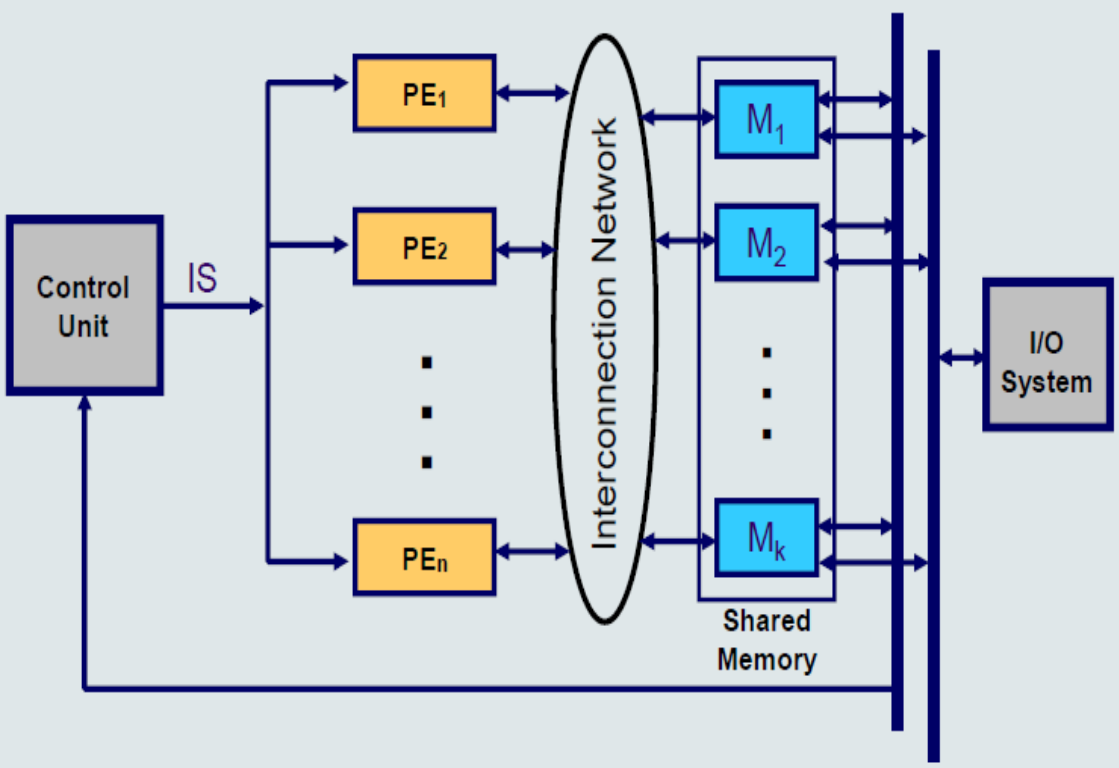
- Processing element complexity
  - Single-bit processors
    - Connection Machine (CM-2) — 65536 PEs connected by a hypercube network (by Thinking Machine Corporation).
  - Multi-bit processors
    - ILLIAC IV (64-bit), MasPar MP-1 (32-bit)
- Processor-memory interconnection
  - Dedicated memory organization
    - ILLIAC IV, CM-2, MP-1
  - Global memory organization
    - Bulk Synchronous Parallel (BSP) computer

Dedicated Memory Organization



- Here we have a Control Unit and multiple synchronized PE.
- The control unit controls all the PE below it .
- Control unit decode all the instructions given to it and decides where the decoded instruction should be executed.
- The vector instructions are broadcasted to all the PE. This broad casting is to get spatial parallelism through duplicate PE.
- The scalar instructions are executed directly inside the CU.

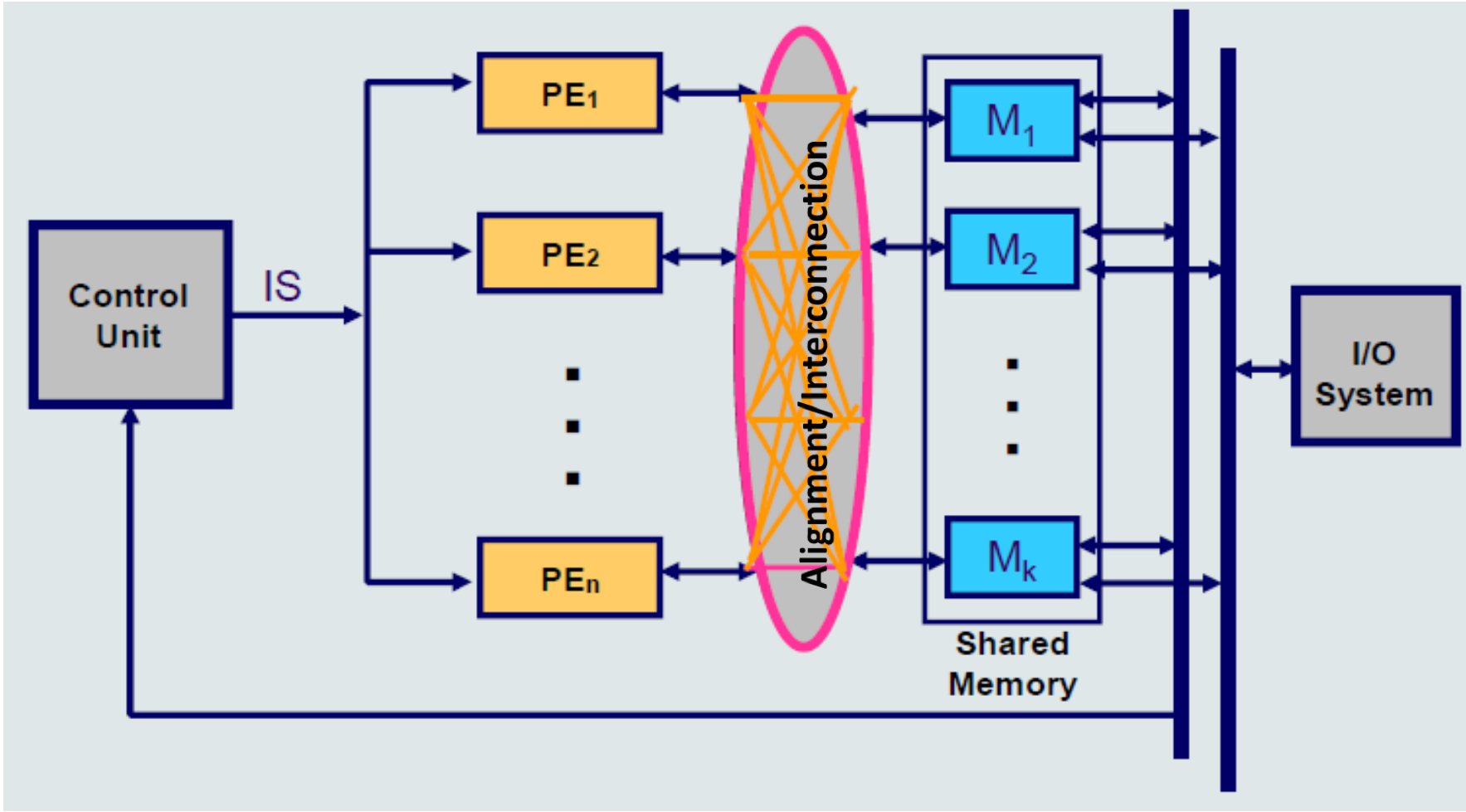
Global Memory Organization(1)



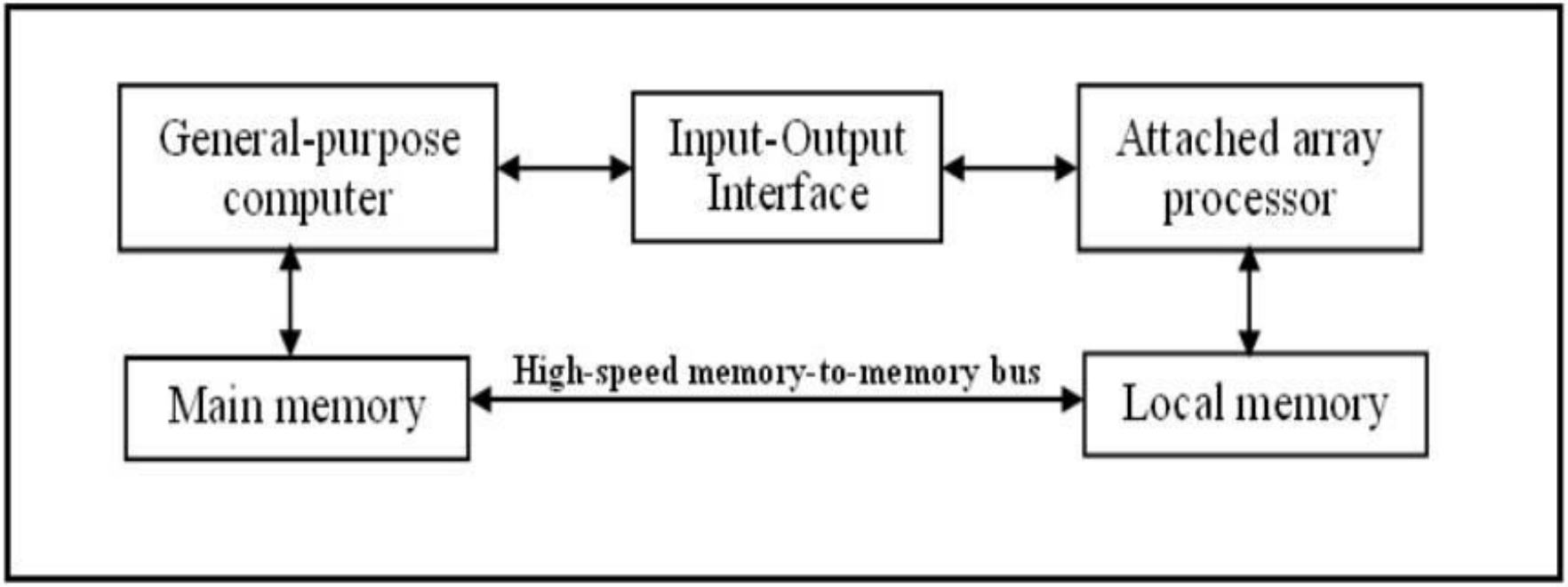
- In this configuration PE does not have private memory. Memories attached to PE are replaced by parallel memory modules shared to all PE via an alignment network.
- Alignment network does path switching between PE and parallel memory.
- The PE to PE communication is also via alignment network .
- The alignment network is controlled by the CU.

- The number of PE (N) and the number of memory modules (K) may not be equal , in fact they are chosen to be prime to each other.
- An alignment network should allow conflict free access of shared memories by as many PEs as possible.

Global Memory Organization(2)



Attached Array Processor

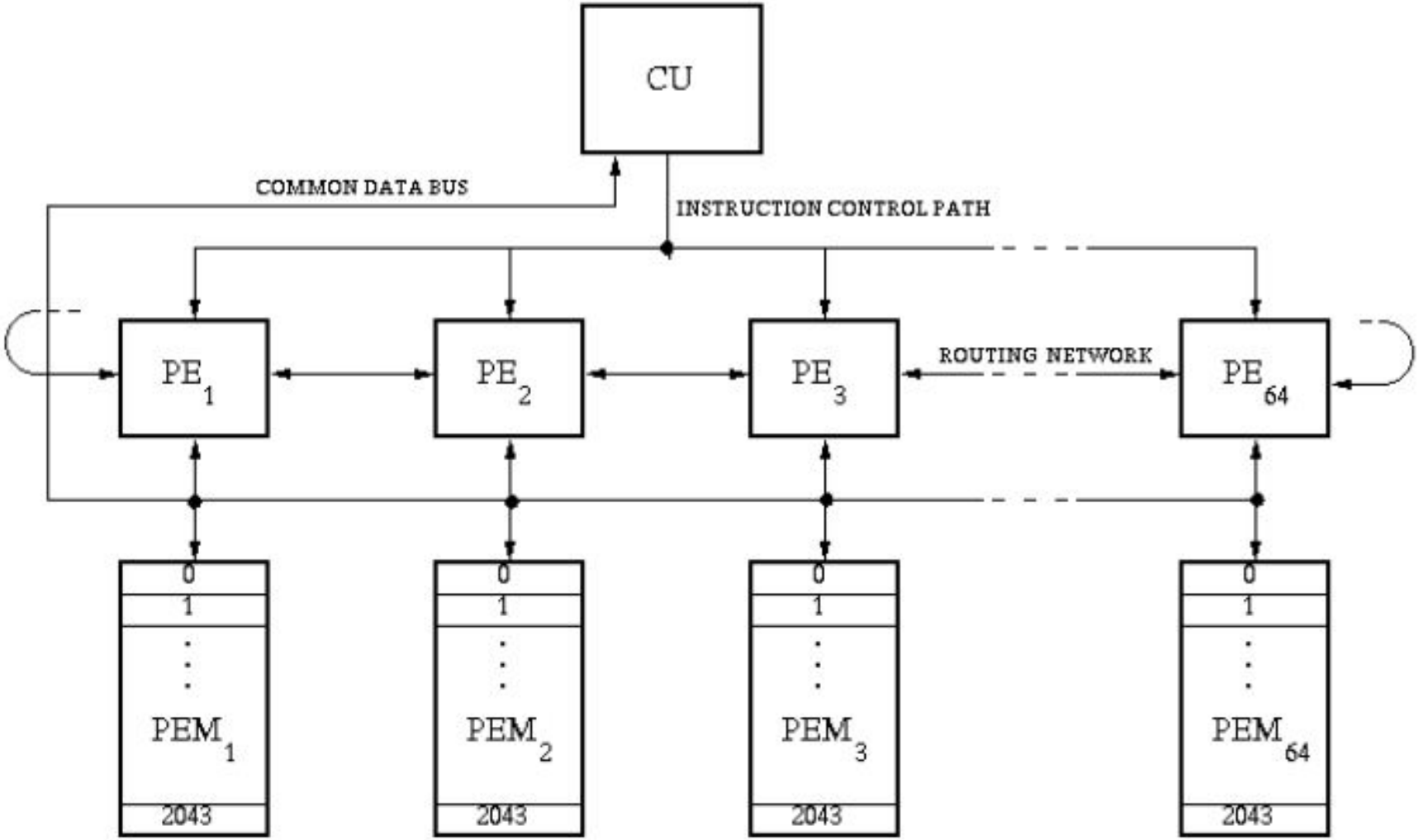


- In this configuration the attached array processor has an input output interface to common processor and another interface with a local memory.
- The local memory connects to the main memory with the help of a high speed memory bus.

ILLIAC IV

- ILLIAC IV is a classical example of Array Processors.
- A typical SIMD computer for array processing.
- 64 Processing Elements (PEs), each with its local memory.
- One single Control Unit (CU).
- CU can access all memory.
- PEs can access local memory and communicate with neighbors.
- CU reads program and broadcasts instructions to PEs.

ILLIAC IV Architecture



References

1. PPT on Pipelining from CS303 (3rd Semester)
2. Advanced Computer Architecture – Kai Hwang
3. Advanced Computer Architectures – Dezso Sima, Peter Karsuk
4. Computer Organization – Carl Hamacher
5. Computer Architecture & Organization – John P. Hayes
6. Computer System Architecture – M. Morris Mano
7. Computer Organization & Architecture – T. K. Ghosh
8. Computer Organization & Architecture – Xpress Learning