DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
ST JOSEPH ENGINEERING COLLEGE, MANGALURU-28
ASSIGNMENT

Semester: VII A & B | **Date of Issue: 22/12/2020**
Subject: Advanced Computer Architecture | **Date of Submission: 09/01/2021**
Subject Code: 17CS72 | **Max. Marks: 10**

Instructions:

- The cover page should bear all required details of the course and the student.
- The assignment must be handwritten and a scanned copy of the same must be uploaded in the google classroom.
- The scan should be neat and has to be a single document.
- The scanned document should be named with your complete USN in capitals.
- The hard copy will be collected when you return back to classes.
- Evaluation will be done based on the soft copy (scanned) submitted on or before the deadline.

Q.NO	Question	TLO	CO	Bloom's Level
1	Identify the three generations of the computers. Explain	4.1	4	Apply (Level 2)
2	Write a note on compound vector processing.	4.1	4	Apply (Level 2)
3	Explain the concept of pipenet and its implementation with a neat diagram..	4.2	4	Apply (Level 2)
4	What are the implementation models of SIMD? Explain them.	4.2	4	Apply (Level 2)
5	With a neat diagram explain the message driven processor architecture.	4.2	4	Apply (Level 2)
6	With a neat diagram explain the architecture of connection machine CM-2	5.1	5	Apply (Level 2)
7	With a neat diagram explain MasPar MP-1 architecture	5.2	5	Apply (Level 2)
8	Explain testing algorithm for dependence testing	5.2	5	Apply (Level 2)

(1) Identify the 3 generations of the computers. Explain.

- First Generation: Caltechs Cosmic Cube, was the first of the first generation multicomputers. The Intel iPSC/1, Ametek S/14, and nCUBE/10 were various evolution of the original Cosmic Cube. For eg, the iPSC/1 used i80286 processors with 512 Kbytes of local memory per node. Each node was implemented on a single printed-circuit board with 8 I/O port. Seven I/O ports were used to form a 7-D hypercube. The eight port was used for an ethernet connection from each node to the host.

Vector hardware was added on a separate board attached to each processing node board. Or one could use the 2nd board to hold extended local memory. The host used in the iPSC/1 was an Intel 310 microprocessor. All I/O must be done through the host.

Present & Future Development:

The second and third generations of multicomputers are introduced below. The Intel Paragon is presented as a case study. Most recent advances in high-performance computing are discussed.

The Second Generation: A major improvement of the 2nd gen included the use of better processors, such as i386 in the iPSC/2 & i860 in the iPSC/860 & in the Delta. The nCUBE/2 implemented 64 custom-designed VLSI processors on a single PC board. The memory per node was also increased to 10 times that of the first generation.

(1)

most importantly, hardware-supported routing such as wormhole routing, reduced the communication latency significantly from 6000 μs to less than 5 μs. In fact, the latency for remote & local communication became almost the same, independent of the number of hops between any 2 nodes.

The Third Generation: These designs laid the foundation for the current generation of multicomputers. Caltech had the Mosaic C project designed to use VLSI-implemented nodes, each containing a 14-MIPS processor, 20-Mbytes/s routing channels & 16 Kbytes of RAM integrated on a single chip.

The full size of the Mosaic was targeted to have a total of 16,384 nodes organized in a 3-D mesh architecture. MIT built the J-machine which it planned to extend to a 65K-node multicomputer with VLSI node interconnected by a 3D mesh network. We will study the J-machine experience. The J-machine planned to use message-driven processors to reduce the message handling overhead to less than 1 μs. Each processor chip would contain a 512 Kbit DRAM, a 32-bit processor, a floating-point unit, and a communication controller. The communication latency in systems was later reduced to a few ns using high-speed links & sophisticated communication protocols. The significant reduction of overhead in communication & synchronization would permit the execution of much shorter tasks with grain sizes of

5 μs per processor in the J-machine, as opposed to executing tasks of 100 μs in the iPSC/1. This implies that concurrency may increase from 10^2 in the iPSC/1 to 10^5 in the J-machine.

(2) – Write a note on compound vector processing.

A compound vector function (CVF) is defined as a composite function of vector operations converted from a looping structure of linked scalar operations. Typical operations appearing in these CVFs include load, store, multiply, divide, logical and shifting vector operations. We use "slash" to represent the divide operations. All vector operations are defined on a component-wise basis unless otherwise specified. The purpose of studying CVF is to explore opportunities for concurrent processing of linked vector operations. The numbers of available vector registers & functional pipelines impose some limitations on how many CVFs can be executed simultaneously.

Vector pipelining & chaining are an integral part of all vector processors. Concurrent processing of several vector arithmetic, logic, shift and memory-access operations require the chaining of multiple pipelines in a linear cascade.

Vector loops or strip mining: When a vector has a length greater that of the vector registers, segmentation of the long vector into fixed-length segments is necessary. This technique has been called strip-mining. One vector segment is processed at a time. In the case of Cray computers, the vector segment length is

(3)

64 elements. Until all the vector elements in each segment are processed, the vector register cannot be assigned to another vector operation. Strip-mining is restricted by the number of available vector registers and so is vector chaining.

Functional Units Independence: In order for vector operations to be linked, they must follow a linear data flow pattern, and all functional pipeline units employed must be independent of each other. The same unit cannot be assigned to execute more than one instruction in the same chain. Furthermore, vector registers must be lined up as interfaces between functional pipelines. The successive output results of a pipeline unit are fed into a vector register, one element per cycle. This vector register is then used as an input register for the next pipeline unit in the chain. With the requirement of continuous data flow in the successive pipelines, the interface registers must be able to pass one vector per cycle between adjacent pipelines.

(3) Explain the concept of pipenet and its implementation with a neat diagram.

Pipeline Net (Pipenet): A pipenet has programmable connectivity. It is constructed from interconnecting multiple functional pipelines through 2 buffered crossbar networks which are themselves pipelined. A two-level pipeline architecture is seen in a pipeline net. The lower level corresponds to pipelining within each functional unit. The higher level is the pipelining of FPs through the BCNs. The set of functional pipelines should be able to handle important vector

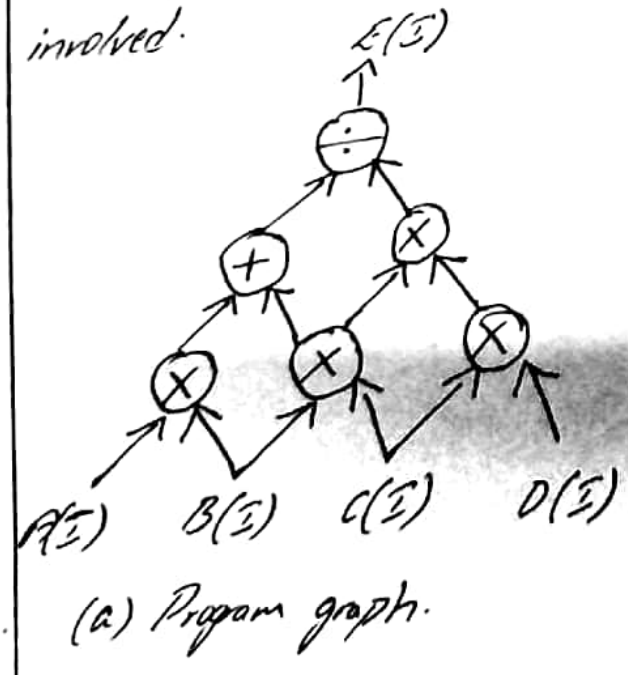
(4)

arithmetic, logic, shifting and masking operations. Each FP_i is pipelined with k_i stages. The output terminals of each BCN are buffered with programmable delays. BCN1 is used to establish the dynamic connections between the register file and the FPs. BCN2 sets up the dynamic connections among the FPs.

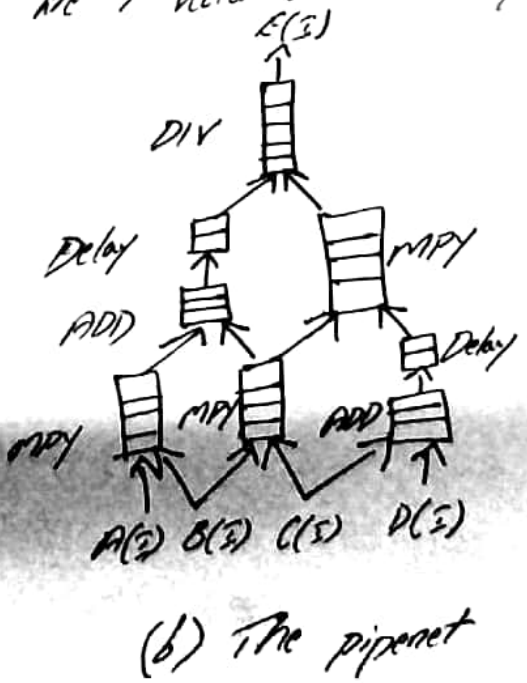
Setup of the Pipenet: Fig shows how to convert from a program graph to a pipenet. Whenever a CVF is to be evaluated, the crossbar networks are programmed to set up a connectivity pattern among the FPs that matches the data flow pattern in the CVF. The program graph represents the data flow pattern in a given CVF. Nodes on the graph correspond to vector operators, & edges show the data dependence, with delays properly labeled among the operators. The prog graph in fig corresponds to the full CVF.

$$E(I) = [A(I) \times B(I) \times C(I)] / [B(I) \times C[I] \times [C[I]) + D(I)]]$$

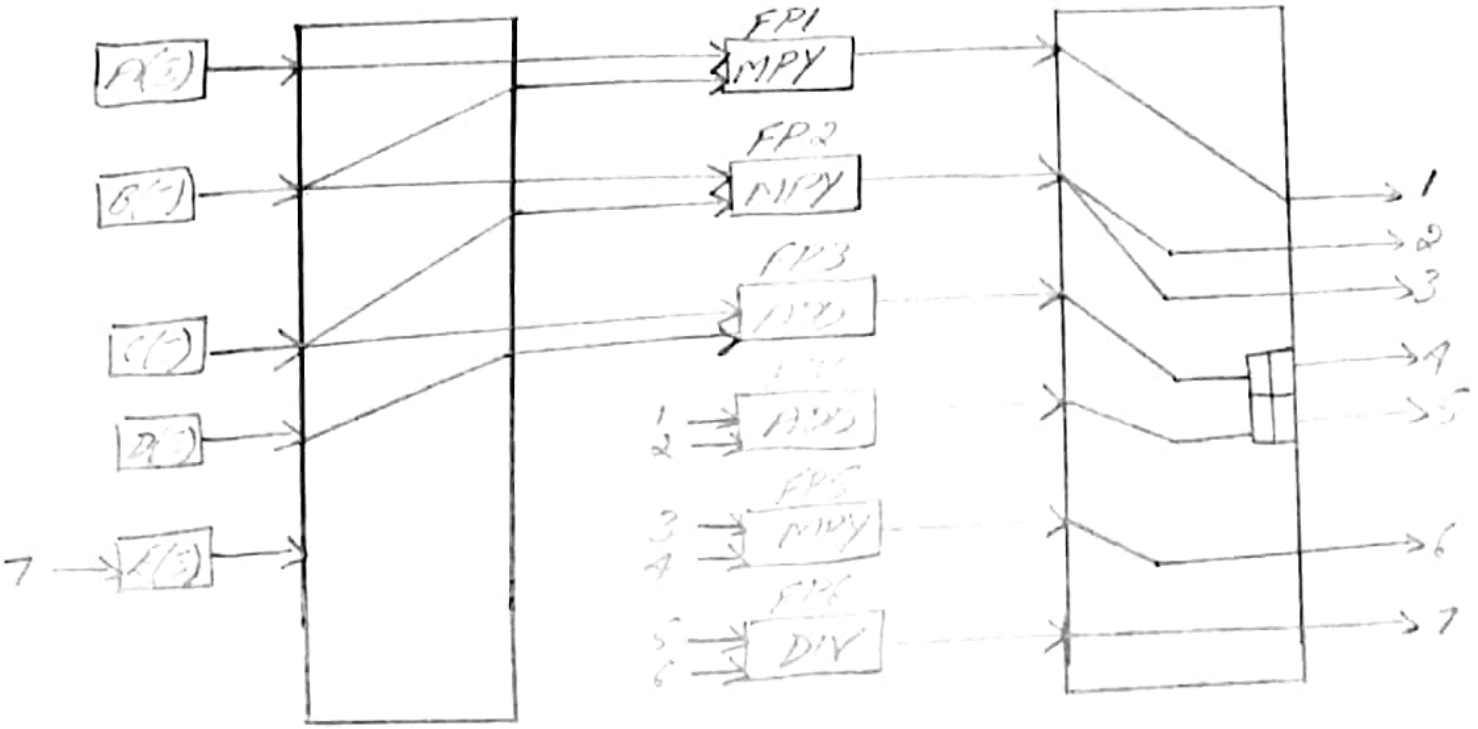
for $I = 1, 2, \ldots, n$. This CVF has 4 input vectors $A(I)$, $B(I)$, $C(I)$ & $D(I)$ & one output vector $E(I)$ which demand 5 memory-access operations. In addition, there are 7 vector arithmetic operations involved.



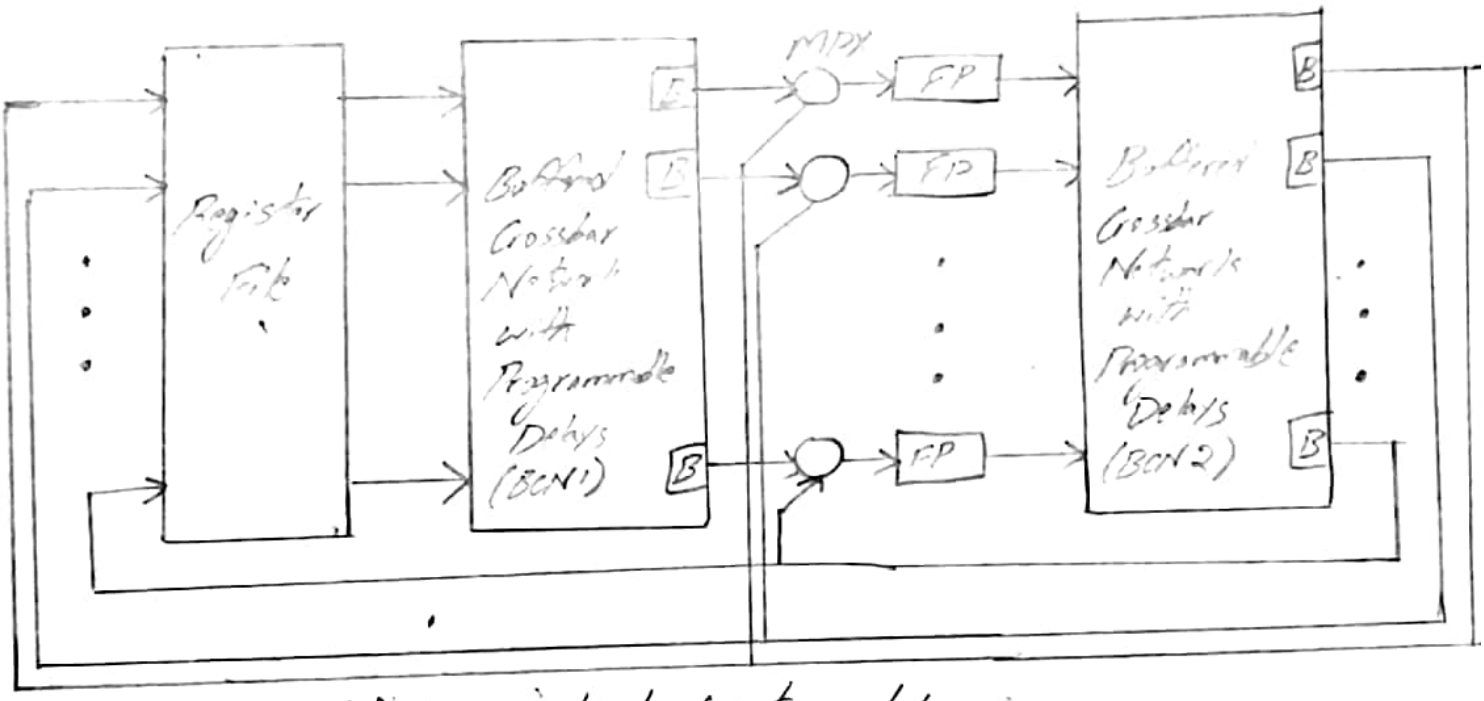
(a) Program graph.



(b) The pipenet



(c) A crossbar implementation.



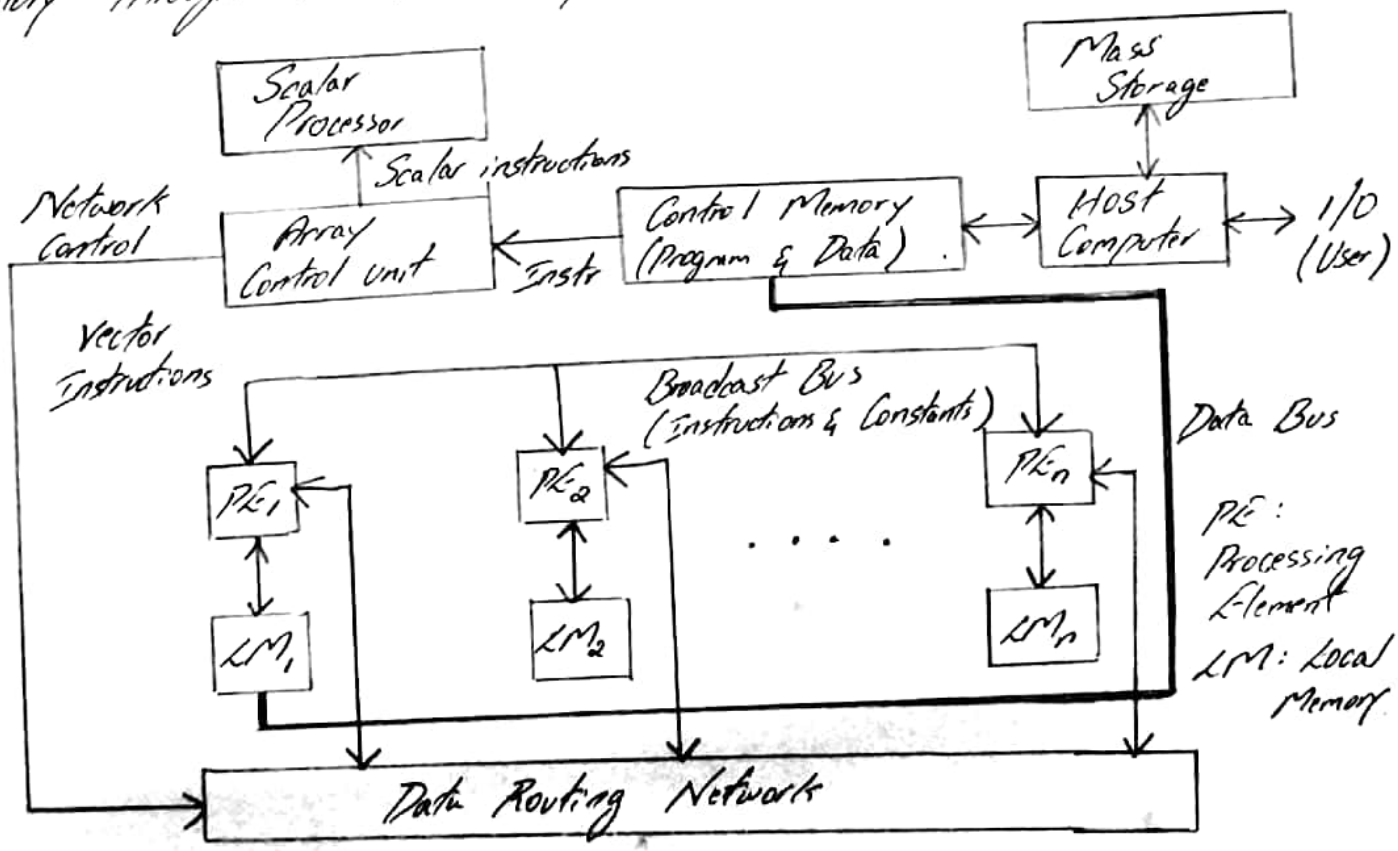
(d) A generalized pipeline model

Six FPs are employed to implement the 7 vector operations because the product vector $B(I) \times C(I)$, once generated, can be used in both the denominator & the numerator. We assume 2, 4 & 6 pipeline stages in the ADD, MPY & DIV units respectively. Two noncompute delays are being inserted, each with 2 clock delays, along 2 of the

connecting paths. The purpose is to equalize all the path delays from the input end to the output end. The connections among the FDs & the 2 inserted delays are shown in fig for a crossbar connected vector processor. The feedback connections are identified by numbers. The delays are set up in the appropriate buffers at the output terminals identified as 4 & 5. Usually, these buffers allow a range of delays to be set up at the time the resources are scheduled.

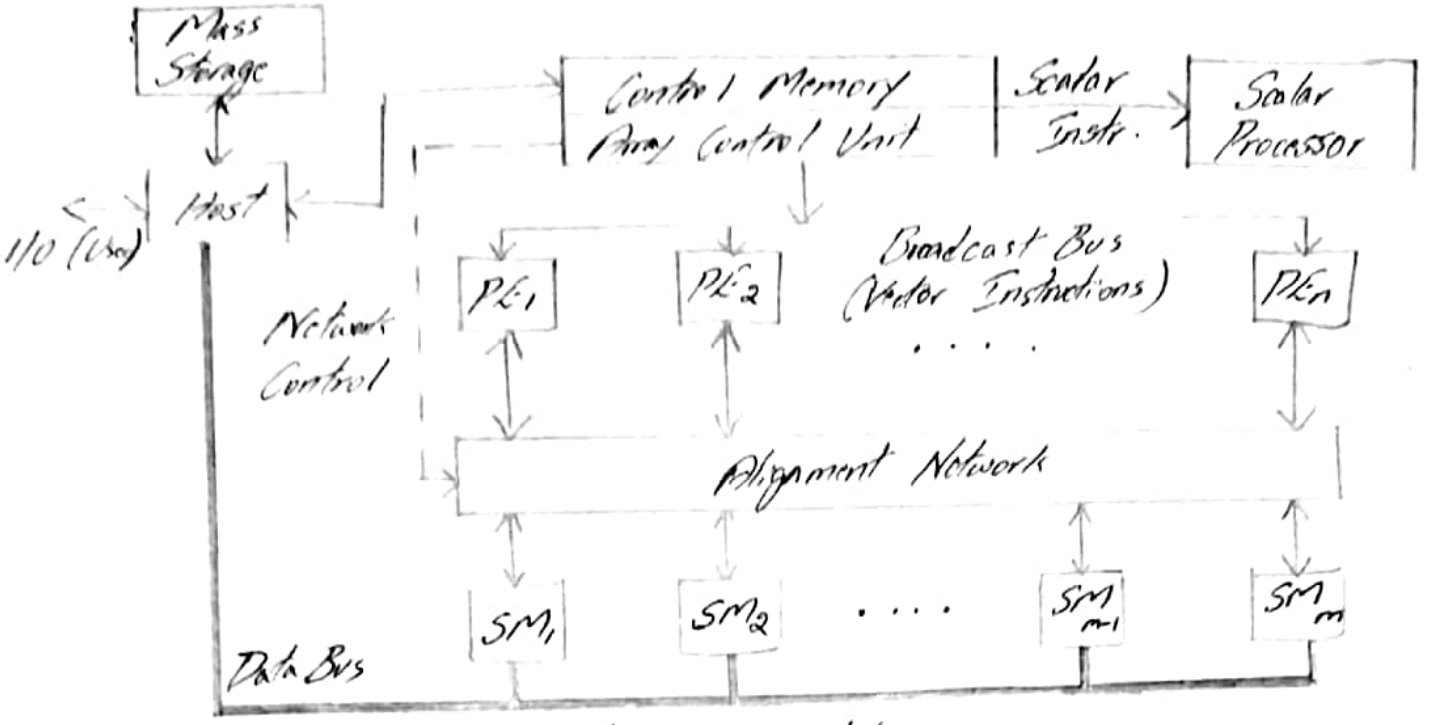
(4) What are the implementation models of SIMD? Explain them.

— Distributed-Memory model: Spatial parallelism is exploited among the PEs in an SIMD computer. A distributed-memory SIMD computer consists of an array of PEs which are controlled by the same array control unit. Program & data are loaded into the control memory through the host computer.



(a) Using distributed local memories.

(7)



(b) Using Shared-memory modules

Two models for constructing SIMD supercomputers

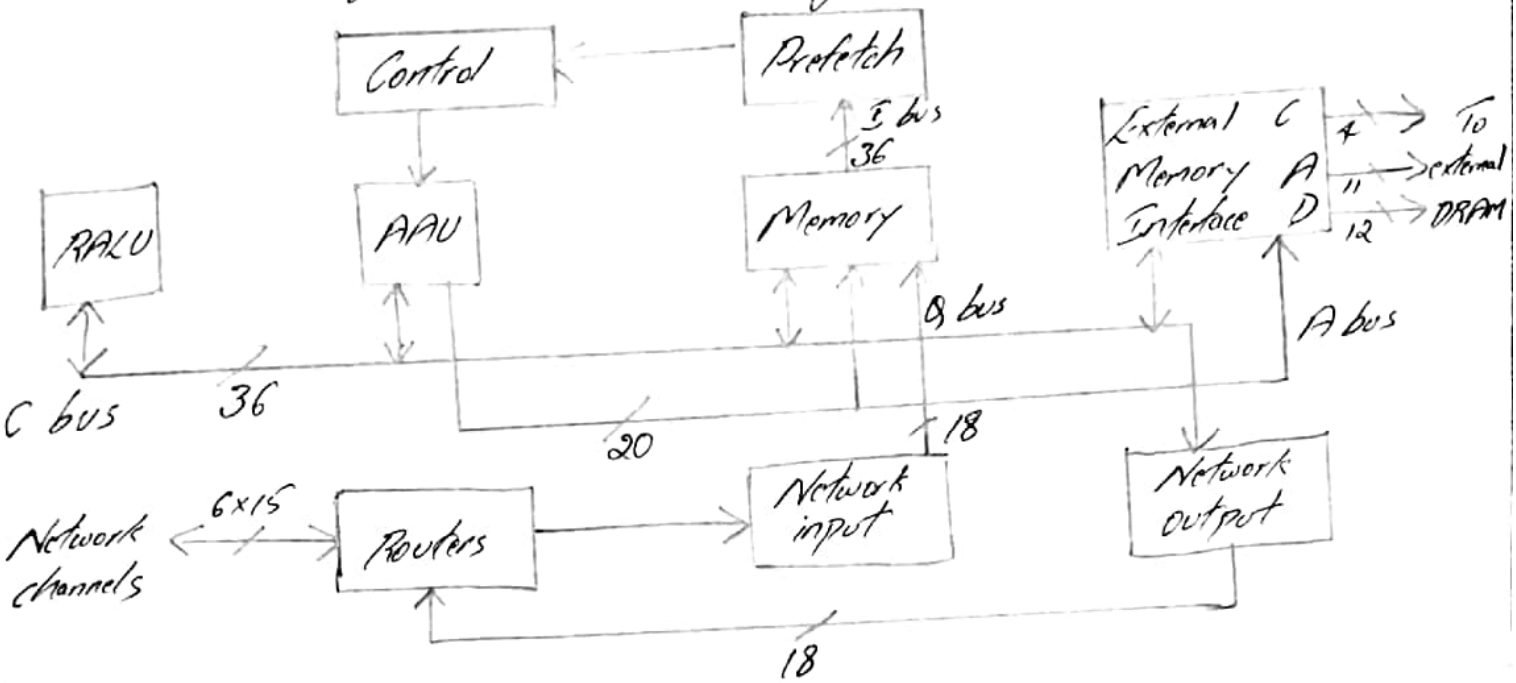
An instn is sent to the CU for decoding. If it is a scalar or program control operation, it will be directly executed by a scalar processor attached to the CU. If the decoded instn is a vector operation, it will be broadcast to all the PEs for parallel execution.

Partitioned data sets are distributed to all the local memories attached to the PEs through a vector data bus. The PEs are interconnected by a data-routing network which performs inter-PE data communications such as shifting, permutation & other routing operations.

Shared-Memory Model: An alignment network is used as the inter-PE memory communication network. Again this network is controlled by the CU. The Burroughs Scientific Processor had adopted this architecture, with n=16 PEs updating m=17 shared-memory modules through a 16x17 alignment network. It should be noted that the value m is often chosen to be

relatively prime with respect to n, so that parallel memory access can be achieved through skewing without conflicts. The alignment network must be properly set to avoid access conflicts. Most SIMD computers were built with distributed memories.

(5) With a neat diagram explain the message driven processor architecture.



(c) Schematic block diagram

The MDP created a task to handle each arriving message. Messages carrying these tasks drove each computation. MDP was a general-purpose. Multiprocessor computing node that provided the communication, synchronization & global naming mechanisms required to efficiently support fine-grain, concurrent programming models. The grain size was as small as 8-word objects or 20-instruction tasks. As we have seen, fine-grain prog typically execute from 10 to 100 instructions between communication & synchronization actions. MDP chips provided inexpensive processing nodes with plentiful VLSI commodity parts to construct the Jellybean Machine

multicomputer. The MDP appeared as a component with a memory port, six two-way network ports, & a diagnostic port. The memory port provided a direct interface to upto 1M words of ECC DRAM, consisting of 11 multiplexed address lines, a 12-bit data bus, & 3 control signals. Prototype J-Machines used 3 1M×4 static-column DRAMs to form a 4 chip processing node with 262,144 words of memory. The DRAMs cycled 3 times to access a 36-bit data word & a fourth time to check or update the ECC check bits. The network ports connected MDPs together in a 3D mesh network. Each of the 6 ports corresponded to one of the six cardinal dirn & consisted of 9 data & 6 control lines. Each port connected directly to the opposite port on an adjacent MDP.

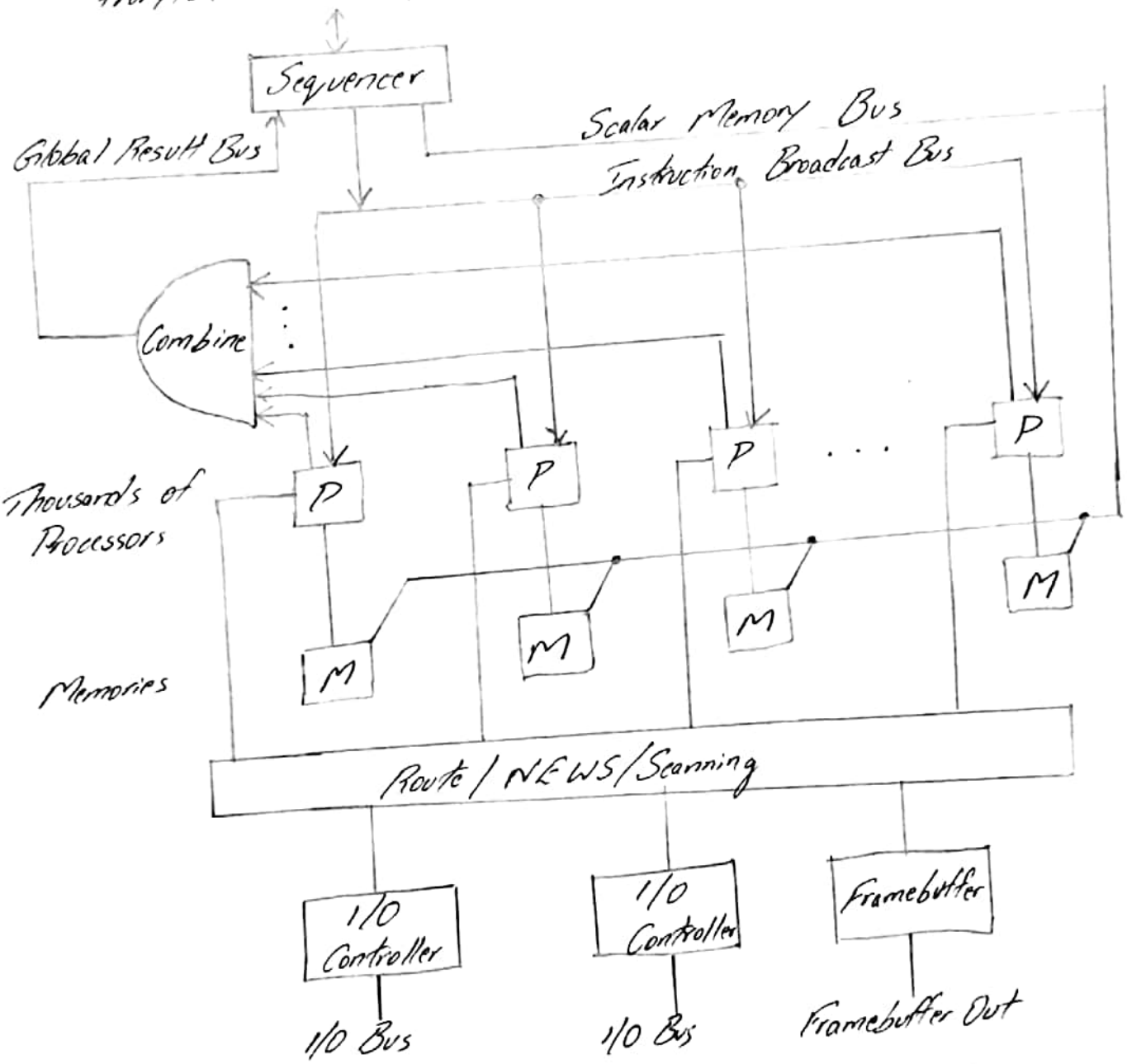
The diagnostic port could issue supervisory commands & read & write MDP memory from a console processor. Using this port, a host could read or write MDP memory from a console MDPs address space as well as reset, interrupt, halt or single step the processor.

6) With a neat diagram explain the architecture of connection machine CM-2.

- Program Execution Paradigm: All programs started execution on a front-end, which issued micro instructions to the back-end processing array when data-parallel operations were desired. The sequencer broke down these microinstructions & broadcast them to all data processors in the array.

Data sets & results could be exchanged between the front-end (10)

and the processing array in one of 3 ways: broadcasting, global combining & scalar memory bus as in fig. Broadcasting was carried out through the broadcast bus to all data processors at once. Global combining allowed the front-end to obtain the sum, largest value, logical OR, etc, of values, one from each processor.

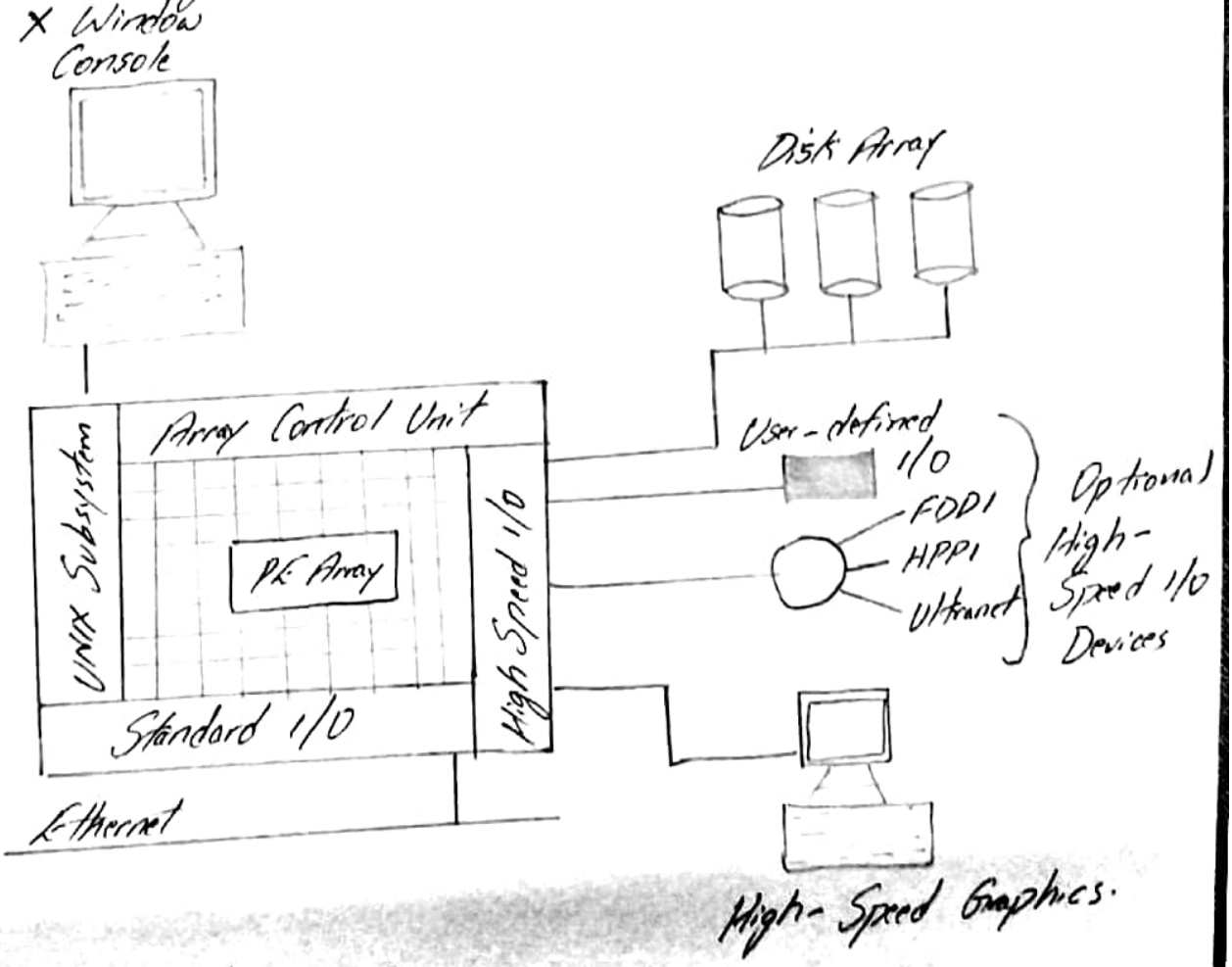


The Processing Array: The CM-2 was a back-end machine for data-parallel computation. The processing array contained from 4k to 64k bit-slice data-processors all of which were controlled

by a sequencer as shown in fig. The sequencer decodes microinstructions from the front-end and broadcast nanoinstructions to the processors in the array. All processors could access their memories simultaneously. All processors executed the broadcast instructions in a lockstep manner.

The processors exchanged data among themselves in parallel through the router, NEWS grids or a scanning mechanism. These network elements were also connected to I/O interfaces. A mass storage subsystem, called the data vault, was connected through the I/O for storing upto 60 Gbytes of data.

7) With a neat diagram explain MasPar MP-1 architecture.



System Block Diagram.

The MasPar MP-1: The MP-1 architecture consisted of 4 subsystems: the PE array, the array control unit (ACU), a UNIX subsystem with standard I/O & a high speed I/O subsystem as depicted in fig. The UNIX subsystem handled traditional serial processing. The high-speed I/O, working together with the PE array, handled massively parallel computing.

The MP-1 family included configurations with 1024, 4096 & upto 16,384 processors. The peak performance of the 16k-processor configuration was 26,000 MIPS in 32-bit RISC integer operations. The system also had a peak floating-point capability of 1.5 Gflops in single-precision & 650 Mflops in double-precision operations.

Array Control Unit: The ACU was a 14-MIPS scalar RISC processor using a demand-paging instruction memory. The ACU fetched & decoded MP-1 instn, computed addresses & scalar data values, issued control signals to the PE array & monitored the status of the PE array.

Like the sequencer in CM-2, the ACU was microcoded to achieve horizontal control of the PE array. Most scalar ACU instructions executed in one 70-ns clock. The whole ACU was implemented on one PC board. An implemented functional unit, called a memory machine, was used in parallel with the ACU. The memory machine performed PE array load & store operations, while the ACU broadcast arithmetic, logic & routing instructions to the PEs for parallel execution.

(8) Explain testing algorithm for dependence testing.

The following procedure is for dependence testing based on a partitioning approach, which can isolate unrelated indices & localize the computation involved & thus is easier to implement.

(1) Partition the subscripts into separable & minimal coupled groups using the foll alg:

Subscript Partitioning Algorithm (Goff, Kennedy, and Tseng, 1991)

Input: A pair of m-dimensional array references containing subscripts $S_1, \ldots S_m$ enclosed in n loops with indices $I_1 \ldots I_n$.

Output: A set of partitions $P_1 \ldots P_{n'}$, $n' \leq n$, each containing a separable or minimal coupled group.

```
For each i, 1 ≤ i ≤ n Do
    P_i ← {S_i}
Endfor
For each index I_i, 1 ≤ i ≤ n Do
    k ← {none}
    For each remaining partition P_j Do
        if ∃S_i ∈ P_j such that S_i contains I_i, then
            if k = {none} then
                k ← j
            else
                P_k ← P_k ∪ P_j
                Discard P_j
            Endif
        Endif
    Endfor
Endfor
```

(2) Label each subscript as ZIV, SIV or MIV.
(3) For each separable subscript, apply the appropriate single subscript test (ZIV, SIV, MIV) based on the complexity of the subscript. This will produce independence or direction vectors for the indices occuring in the subscript.
(4) For each coupled group, apply a multiple subscript test to produce a set of direction vectors for the indices occuring within that group.
(5) If any test yields independence, no dependences exist.
(6) Otherwise merge all the direction vectors computed in the previous steps into a single set of direction vectors for the 2 references.