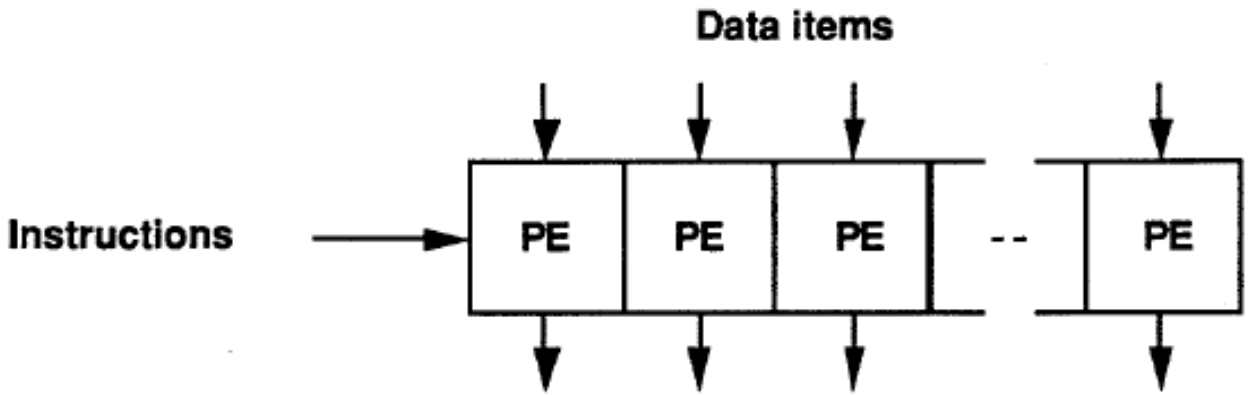
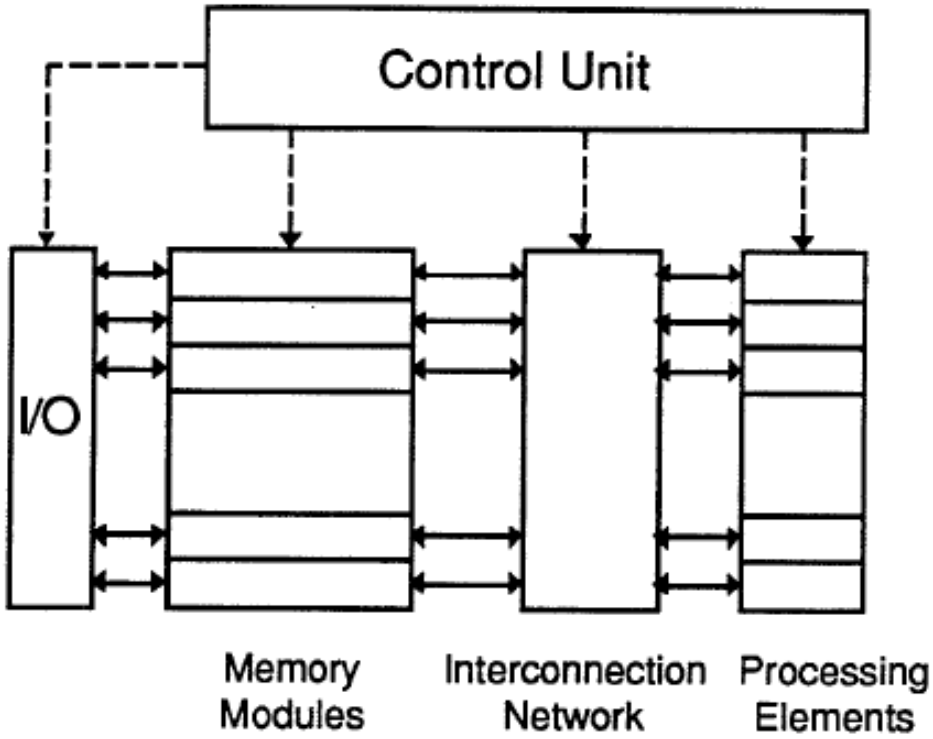
SIMD Architectures

Single Instruction stream, Multiple Data streams



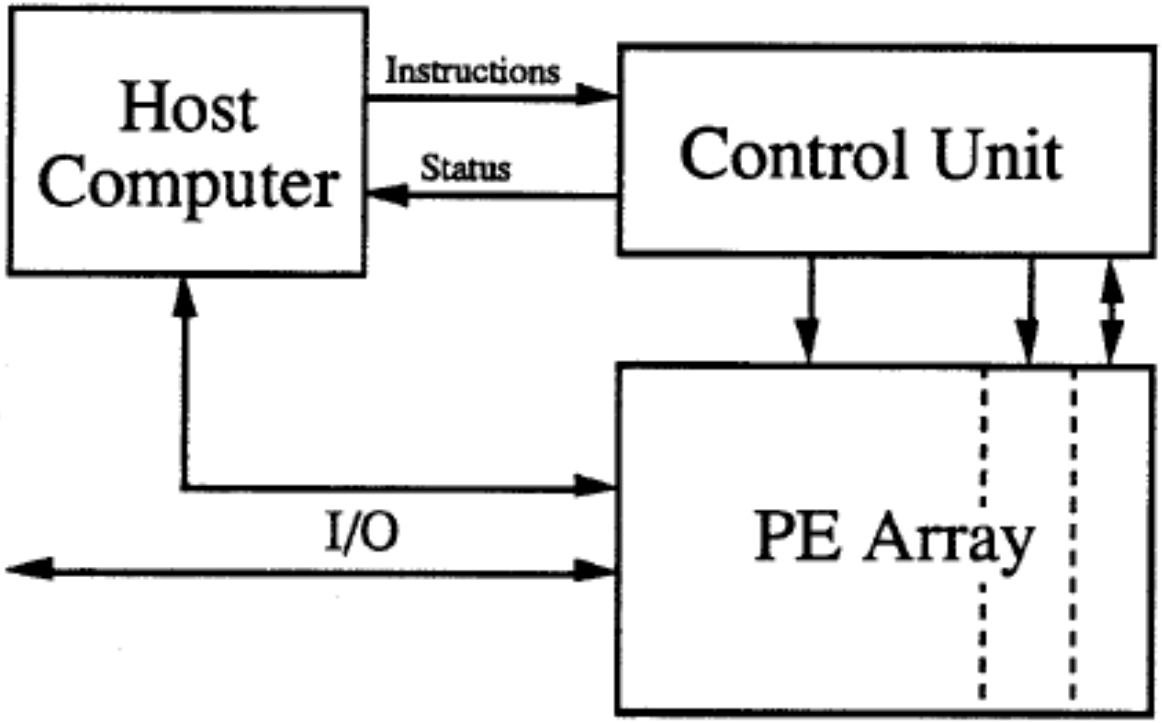
Principle of SIMD processor.

Figure SIMD-1.



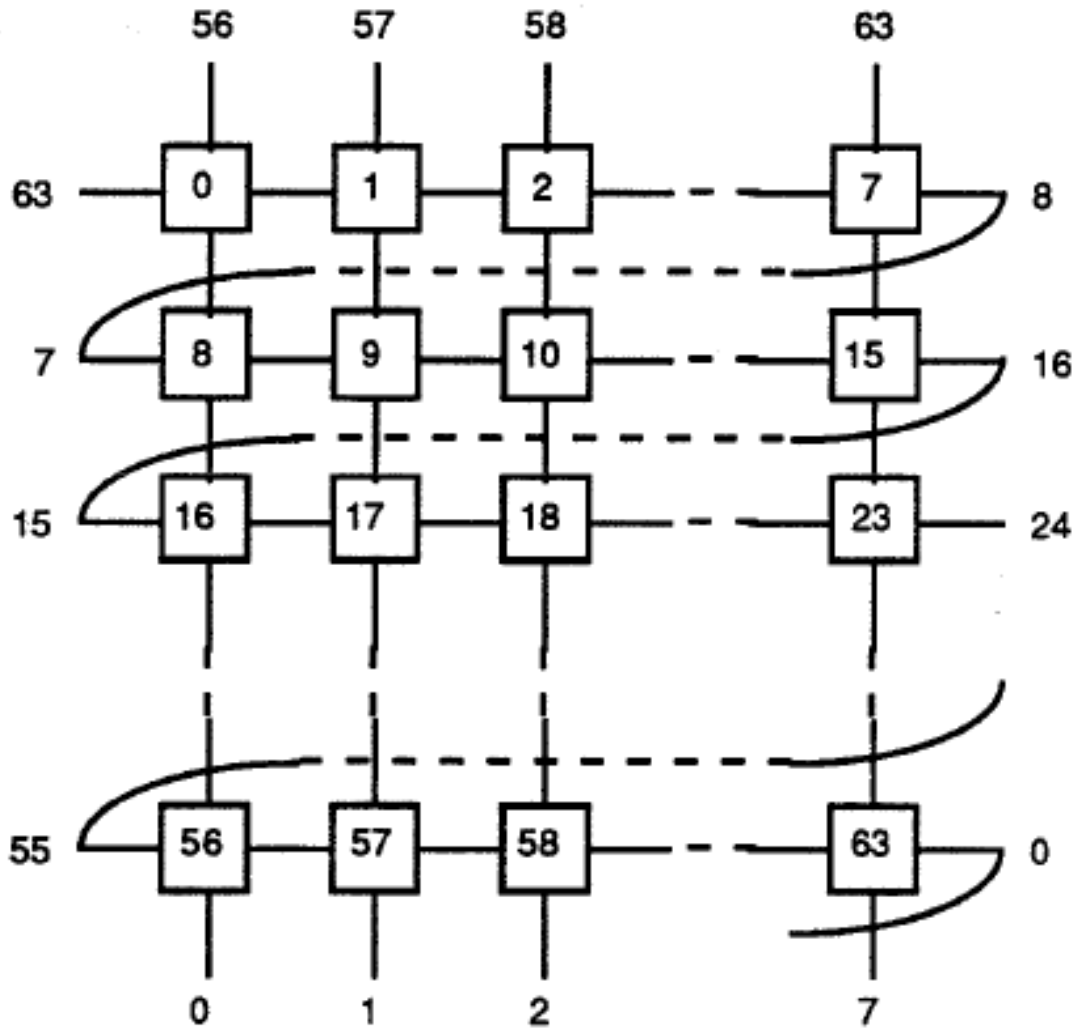
Processor Array

Figure SIMD-2a



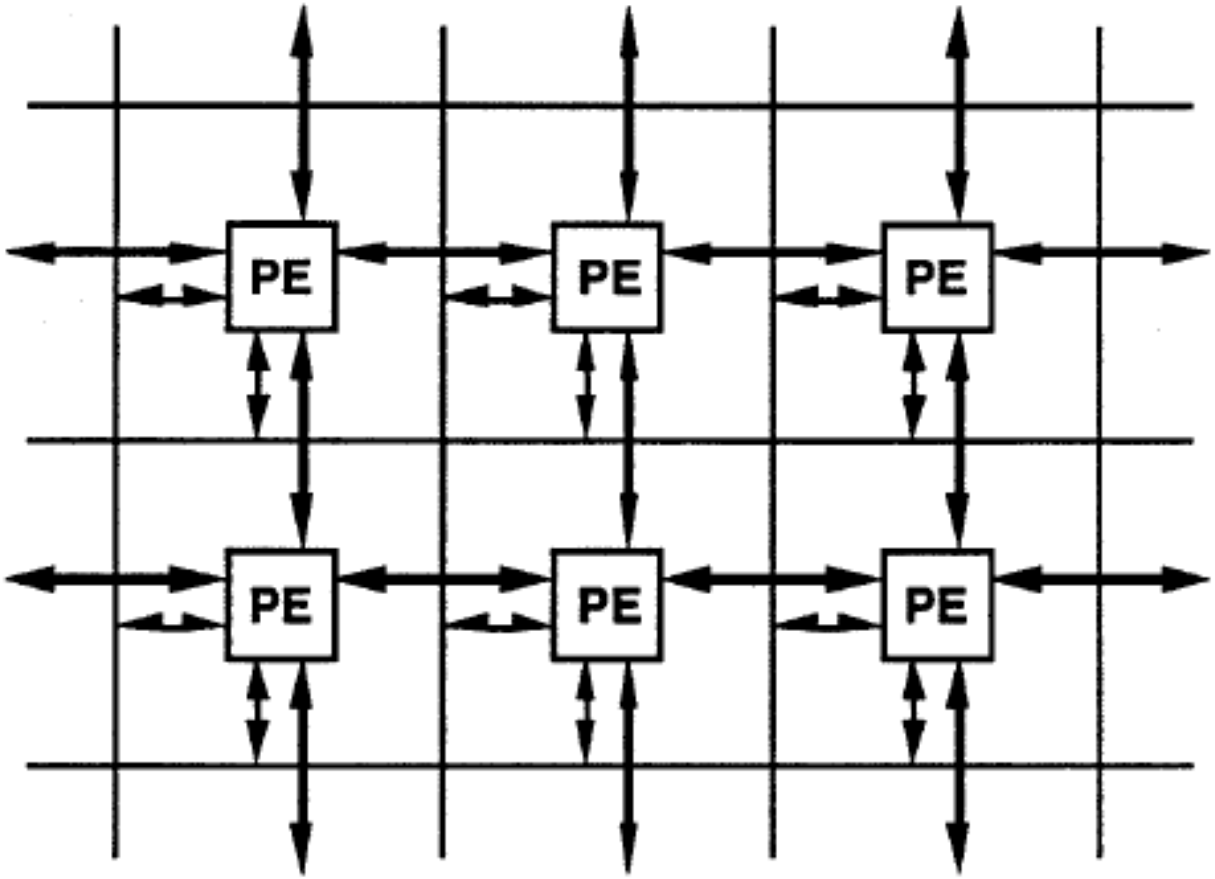
Host Computer - Processor Array Relationship

Figure SIMD-2b



The topology of ILLIAC IV

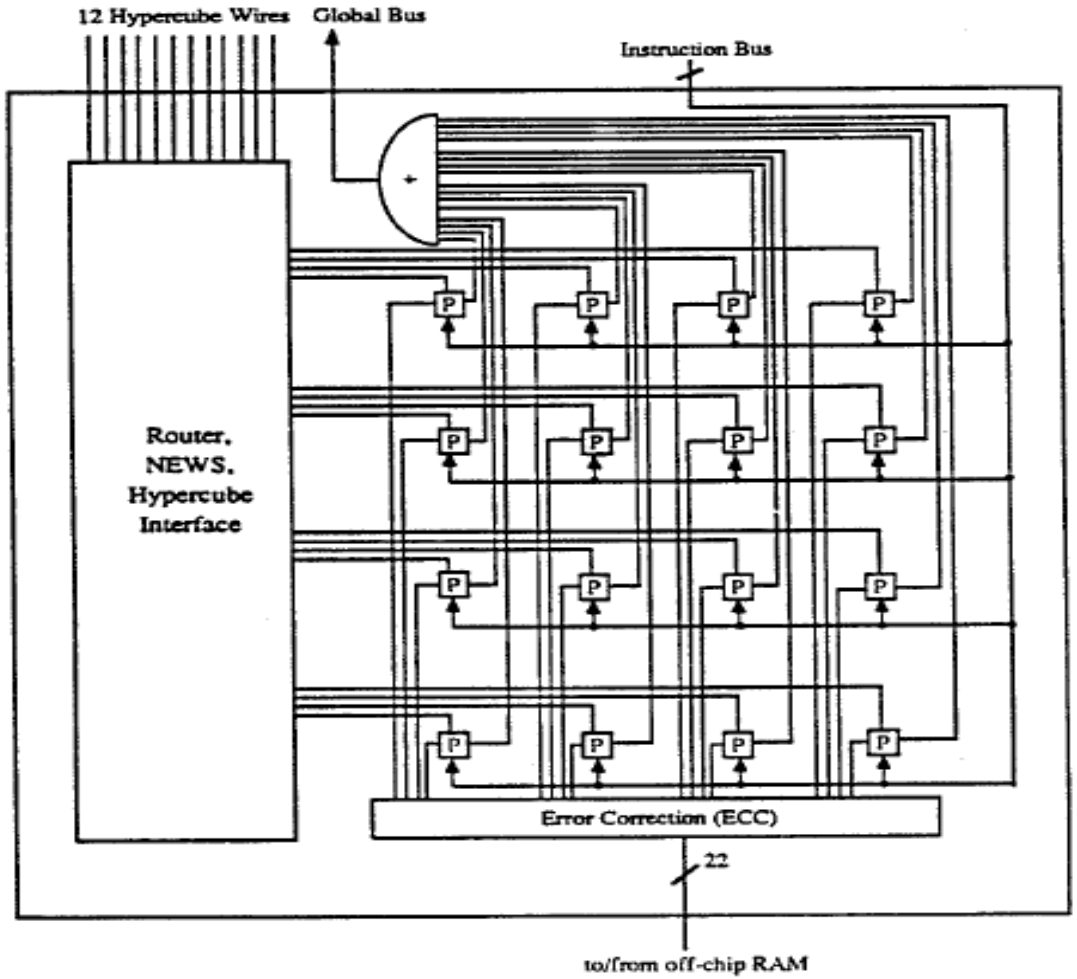
Figure SIMD-4



Connectivity for a small section of the DAP.
Horizontal and vertical lines are Row Highways
and Column Highways, respectively.

Figure SIMD-5

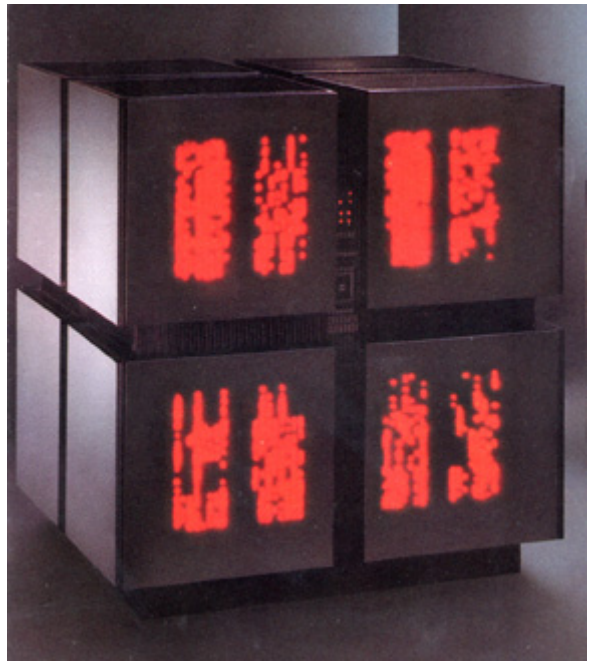
The Connection Machine



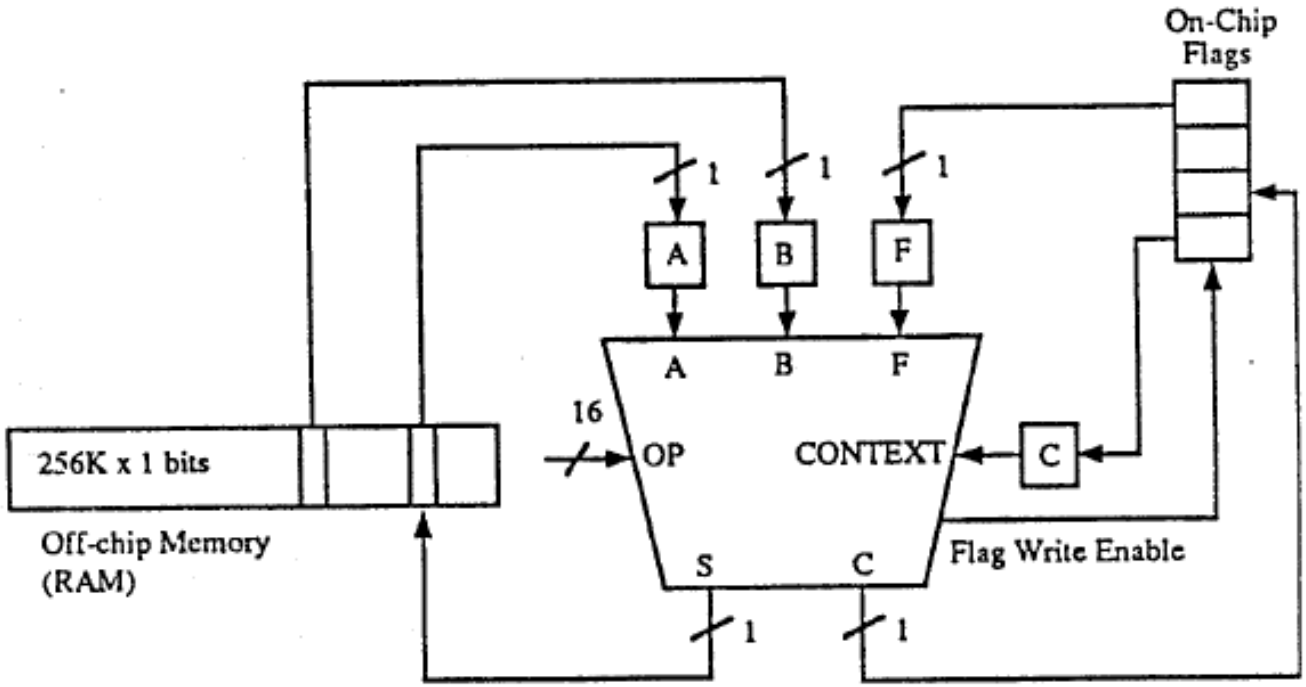
CM-2 Processor Chip (from (Thinking Machines Corporation, 1989))

Figure SIMD-9

The Connection Machine

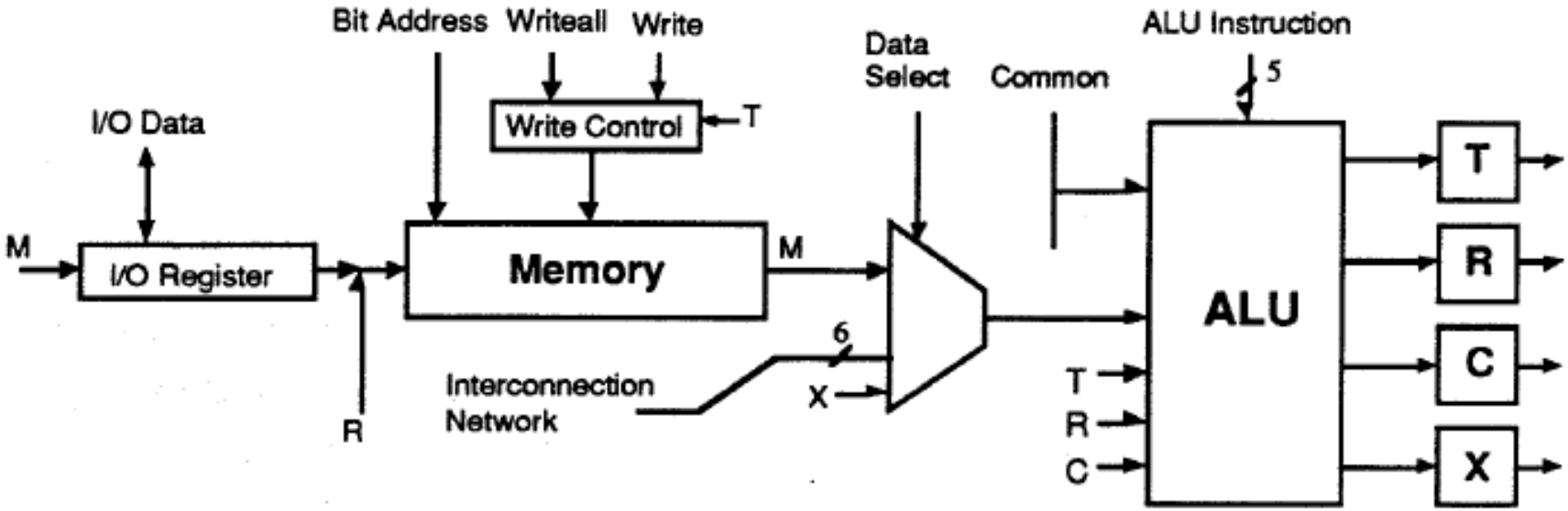


Processing Element of The Connection Machine



LUCAS

Processing Element of LUCAS



Programming SIMD Arrays

- Instruction level
  - Examples:

Maximum value of a field (vector, matrix etc.). On a bit-serial array this is found by traversing the bit-slices from most to least significant bit and successively discarding values with a 'zero' if there are others still active with a 'one'.

Exact match, or *Closest match* between a constant and the values of a field.

Pairwise multiply between the elements of two vectors, matrices etc.

Six instruction types

Based on the type of operands and type of result six basic types of instructions to manipulate data in the memory array can be identified:

Instruction type	*Example*
field --> field	Increment field, permute field
field --> selector	Maximum/minimum value of a field
field,field --> field	Multiply fields, pairwise max of vector elements
field,field --> selector	Pairwise equality between vector elements
constant,field --> field	Multiply by constant, AND with constant
constant,field --> selector	Exact match, closest match, greater than

High-level languages for SIMD (Data-parallel languages)

High-level languages for SIMD processor arrays are often referred to as *data parallel languages*. Typically, they are very similar to conventional languages but allow the programmer to organize data so that operations may be applied to many elements of data simultaneously. This is accomplished by adding new data types and extending the meaning of existing program syntax when applied to parallel data. Extension of the control structure is often also done.

For example,

```
var   SEL       : selector [0..999] := (0..399 -> TRUE);
      WEIGHTS   : parallel array [0..999,0..999] of integer (12);
```

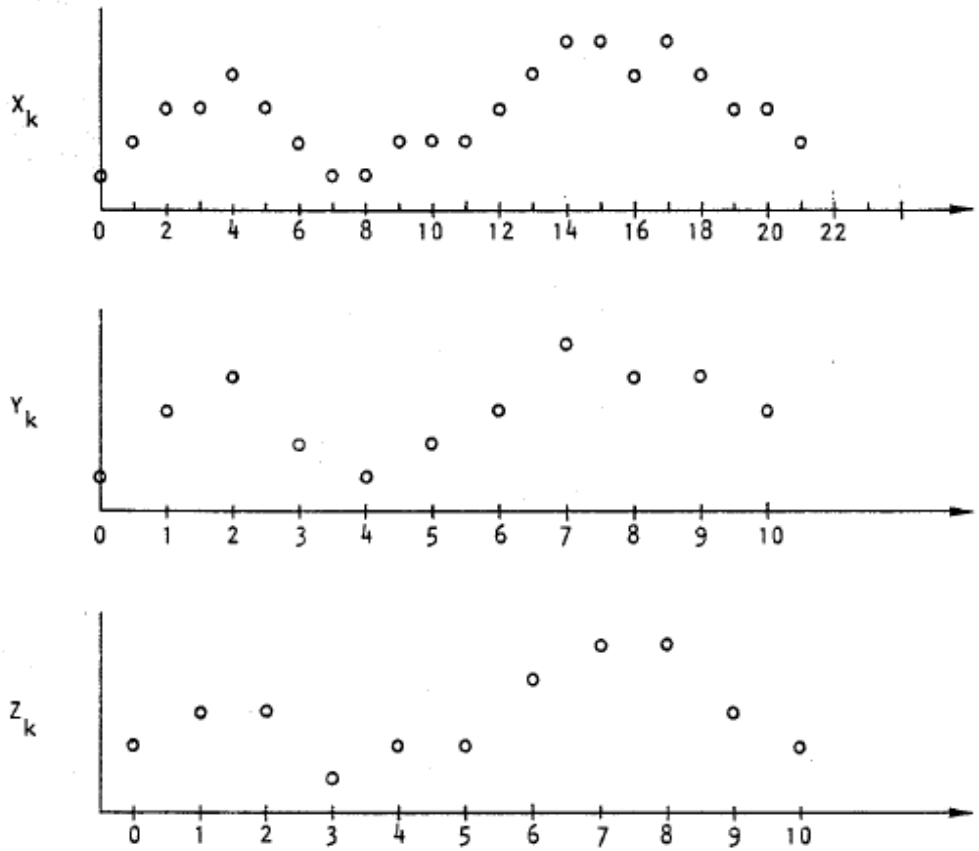
declares a selector with elements in the first 400 MMs selected and a 1000 by 1000 matrix of 12-bit integers located in the first 1000 MMs, 1000 components in each MM.

New control structures
(example from Pascal/L)

The **where do elsewhere** construct defines different actions to take place in different Memory Modules depending on the contents of a selector:

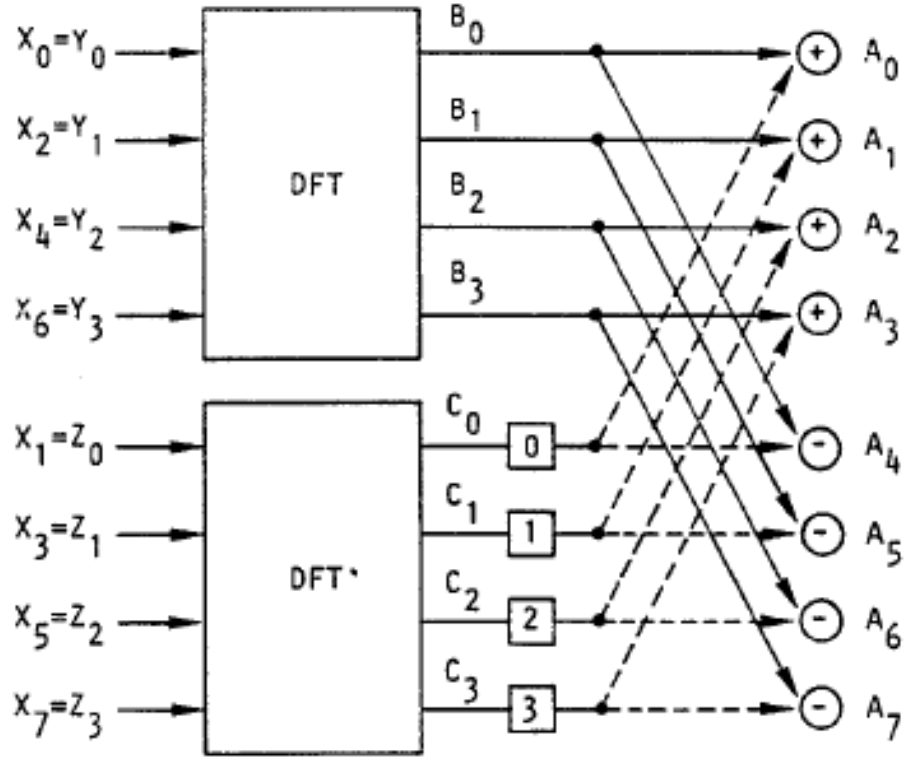
```
where SEL do WEIGHTS[*,10] := 2*WEIGHTS[*,10]
      elsewhere  WEIGHTS[*,10] := 0;
```

Example: Fast Fourier Transform

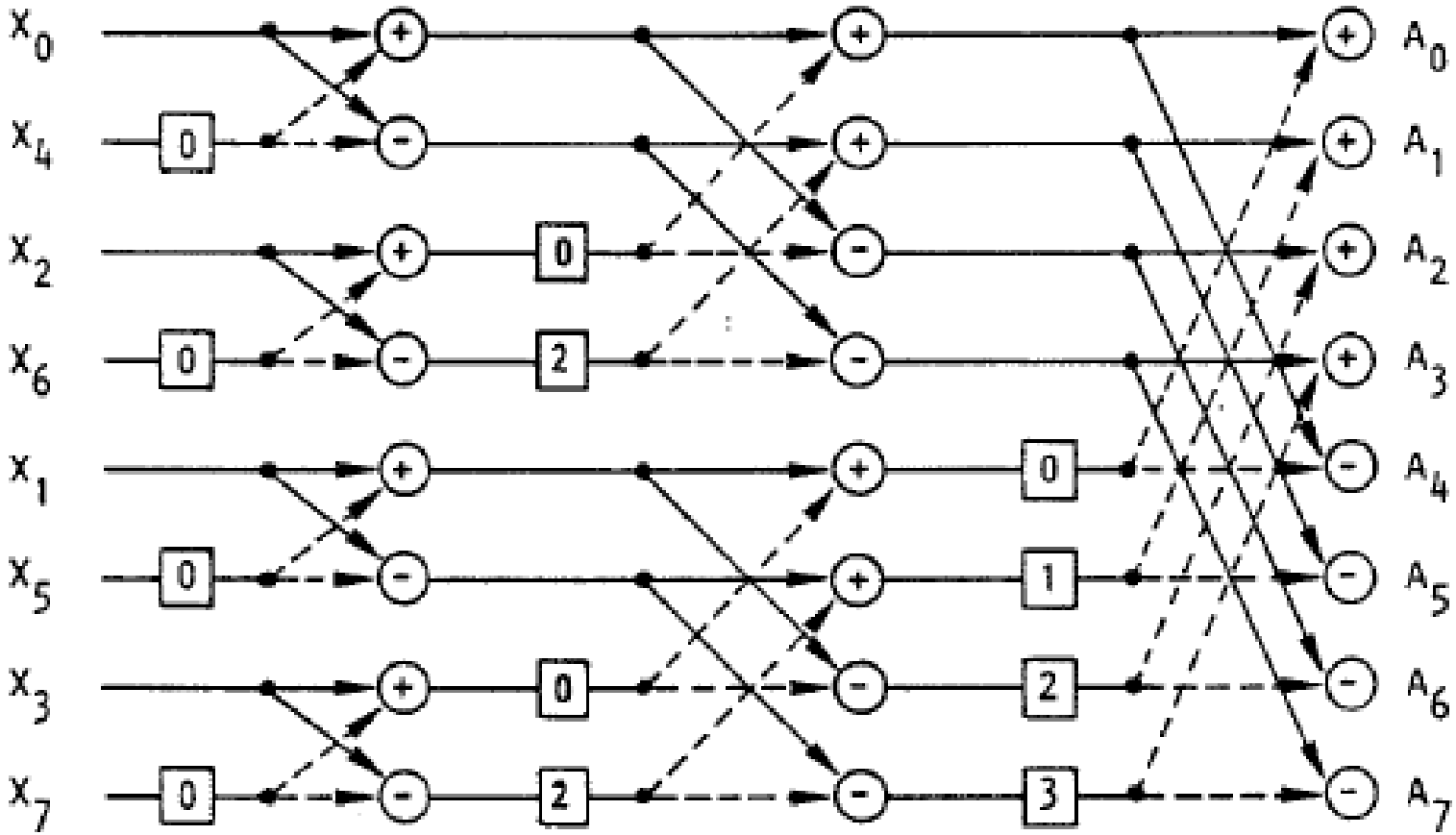


Decomposition of a time series into two half as long series

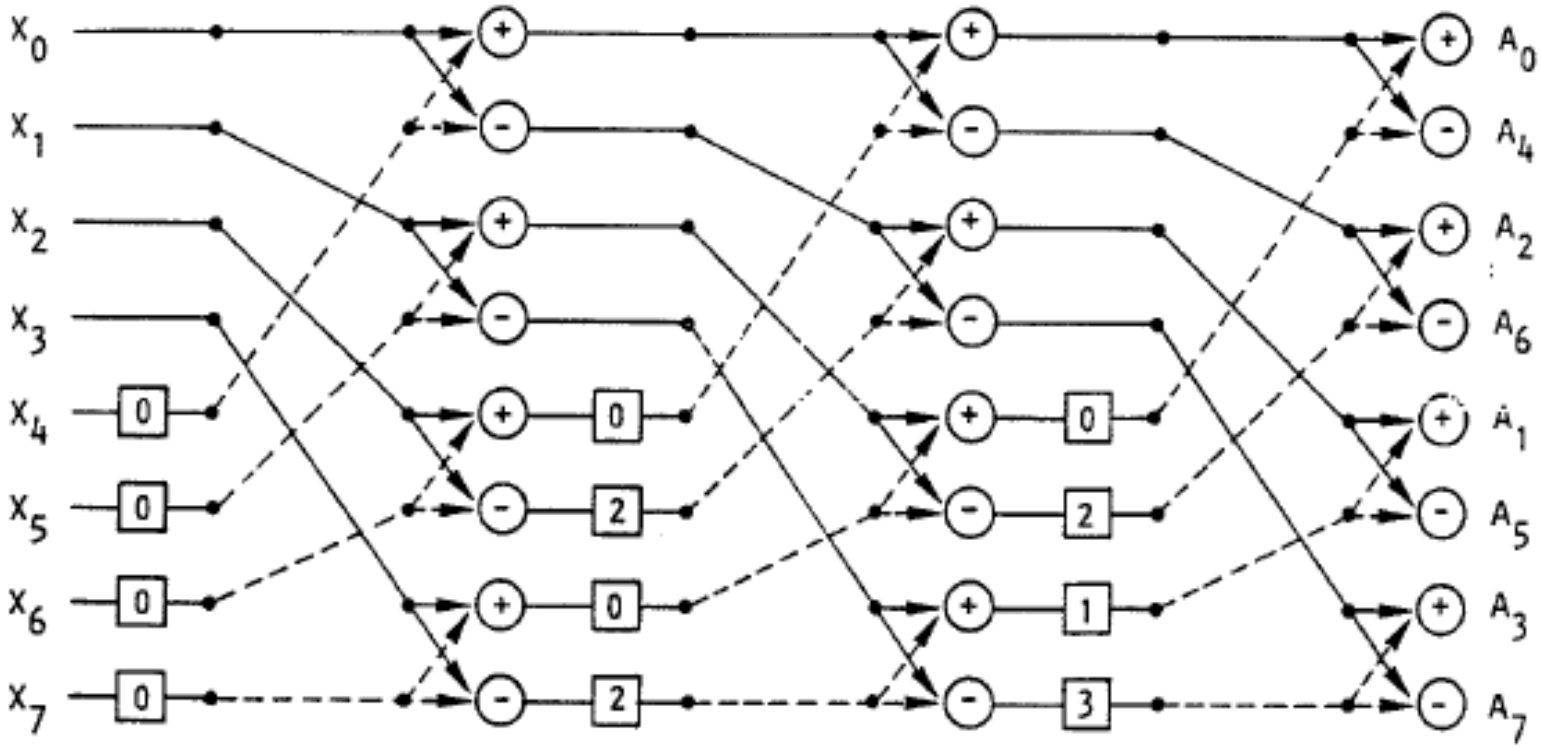
Figure SIMD-21



Signal flow graph illustrating how calculation of an 8-point DFT can be reduced to the calculation of two 4-point DFTs. A number within a square represents multiplication by $e^{-2\pi j/N}$ raised to the number. In the lower half, the value arriving by the dotted line is subtracted from the value arriving by the solid line. In the upper half the two values are added.

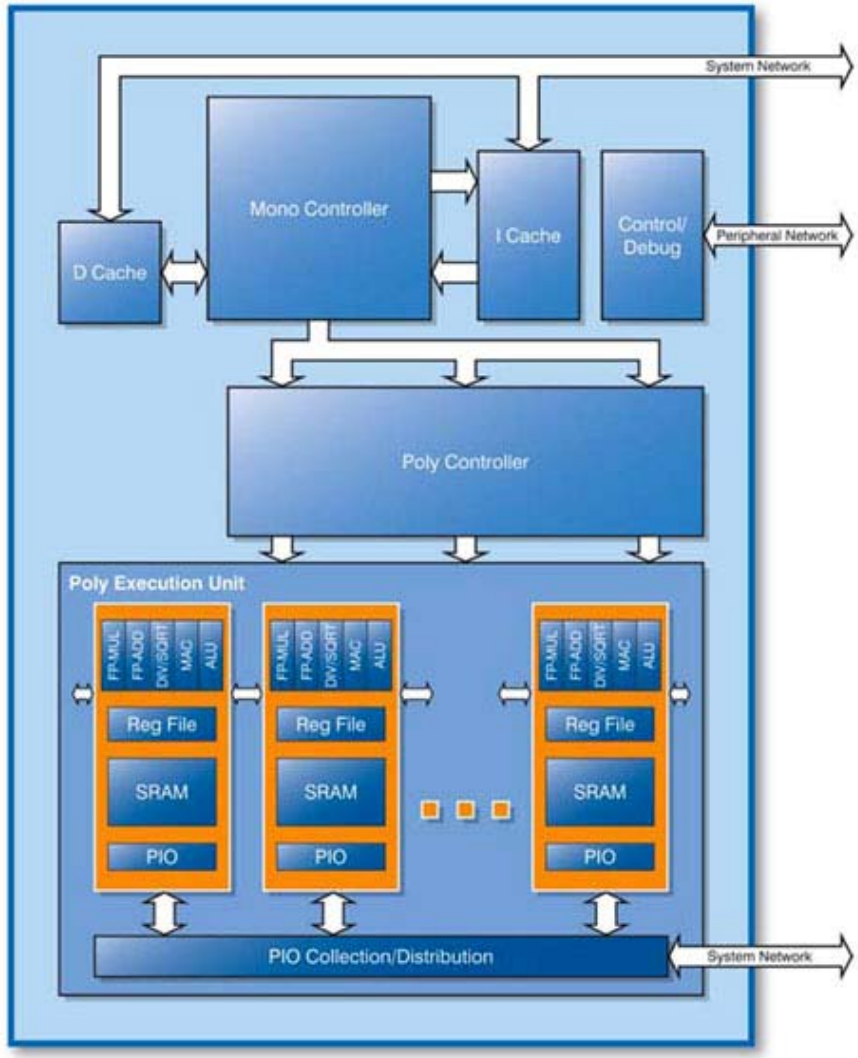


Calculation of an 8-point DFT using the FFT algorithm



Adaptation of the FFT algorithm to the perfect shuffle-exchange interconnection structure

Modern product: ClearSpeed CSX700



Features
192 high-performance processing elements, each with:

Dual 32 & 64-bit FPU

ClearSpeed: FFTs per second