北京郵電大學
Beijing University of Posts and Telecommunications

Computer Architecture

计算机系统结构

（向量处理＋阵列机）

北京邮电大学
邝 坚
2020年5月

Vector process principles

- Vector – a set of scalar data items, all of same type, stored in memory.

$$A = (a_1, a_2, a_3, ..., a_n)$$

- Vector processor – an ensemble of HW resource, including vector registers, functional pipelines, processing elements, and register counters, for performing vector operations.
- Vectorization – the conversion from scalar code to vector code.

Vectorization

- Y = a * X

```
LD      F0,a          ;F0 ← a
LV      V1,M(X)       ;V1 ← X
MULV    V2,F0,V1      ;V2 ← a * X
SV      M(Y),V2       ;Y ← V2
```

Unrolled-loop-level parallelism

用标量指令集如何实现?

Vectorization

- 计算： $c_i = a_{i+5} + b_i \quad ; i = 10,11,12,...,1000$

- FORTRAN(标量)

```
      DO  40  I = 10,1000
40    C(I) = A(I+5) + B(I)
```

- FORTRAN(向量)

```
C(10:1000) = A(10+5:1000+5) + B(10:1000)
```

Vector instruction types

- Vector-vector instructions

$$V_1 \rightarrow V_2 \qquad ;V_2 = \sin(V_1)$$

$$V_j \; op \; V_k \rightarrow V_i \qquad ;V_3 = V_1 + V_2$$

- Vector-scalar instructions

$$s \; op \; V_k \rightarrow V_i \qquad ;V_2 = s \times V_1$$

- Vector-memory instructions

$$M \rightarrow V \qquad ;\text{Vector load}$$

$$V \rightarrow M \qquad ;\text{Vector store}$$

Vector instruction types

- Vector-reduction instructions

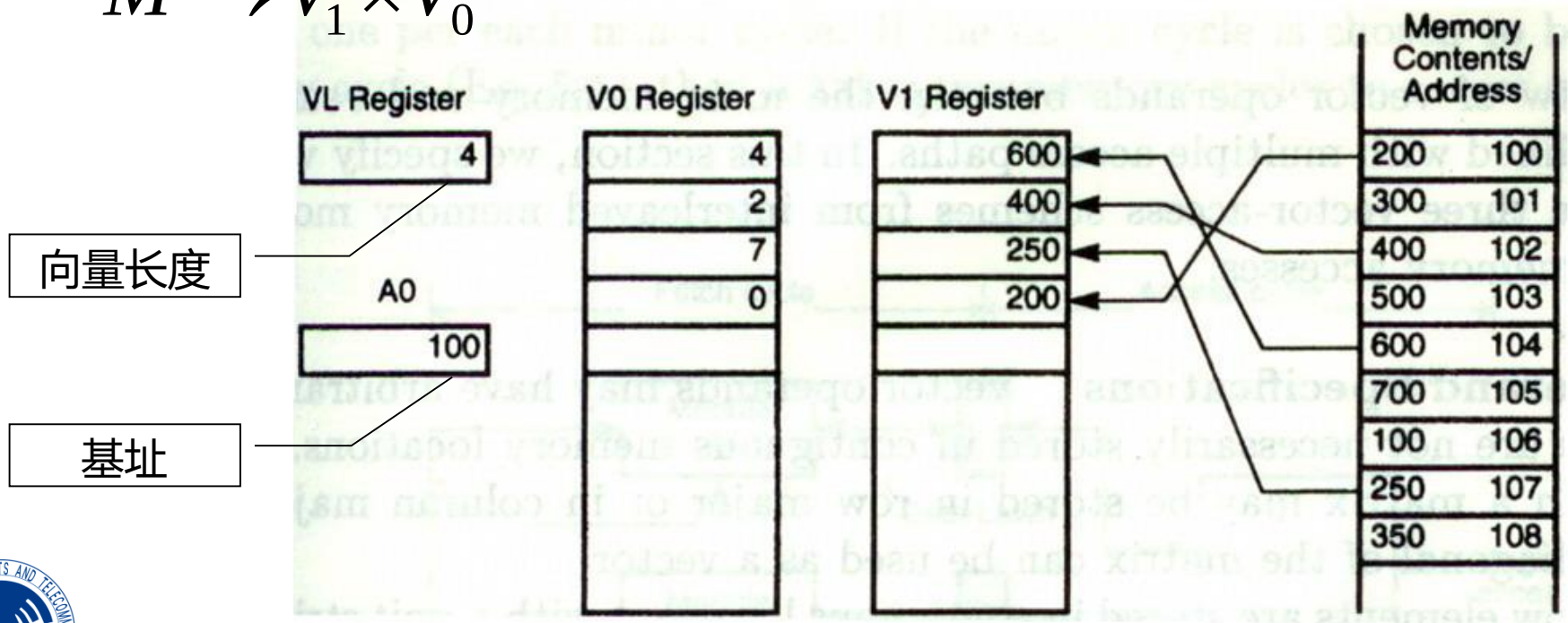
$$V_i \rightarrow s_j \qquad ;\text{max, min, sum, mean value}$$

$$V_j \; op \; V_k \rightarrow s_i \qquad ;\text{dot product, } s = \sum_{i=1}^{n} a_i \times b_i$$

Vector instruction types

- Gather instruction – fetch from memory the nonzero element of a sparse vector using indices themselves are indexed.

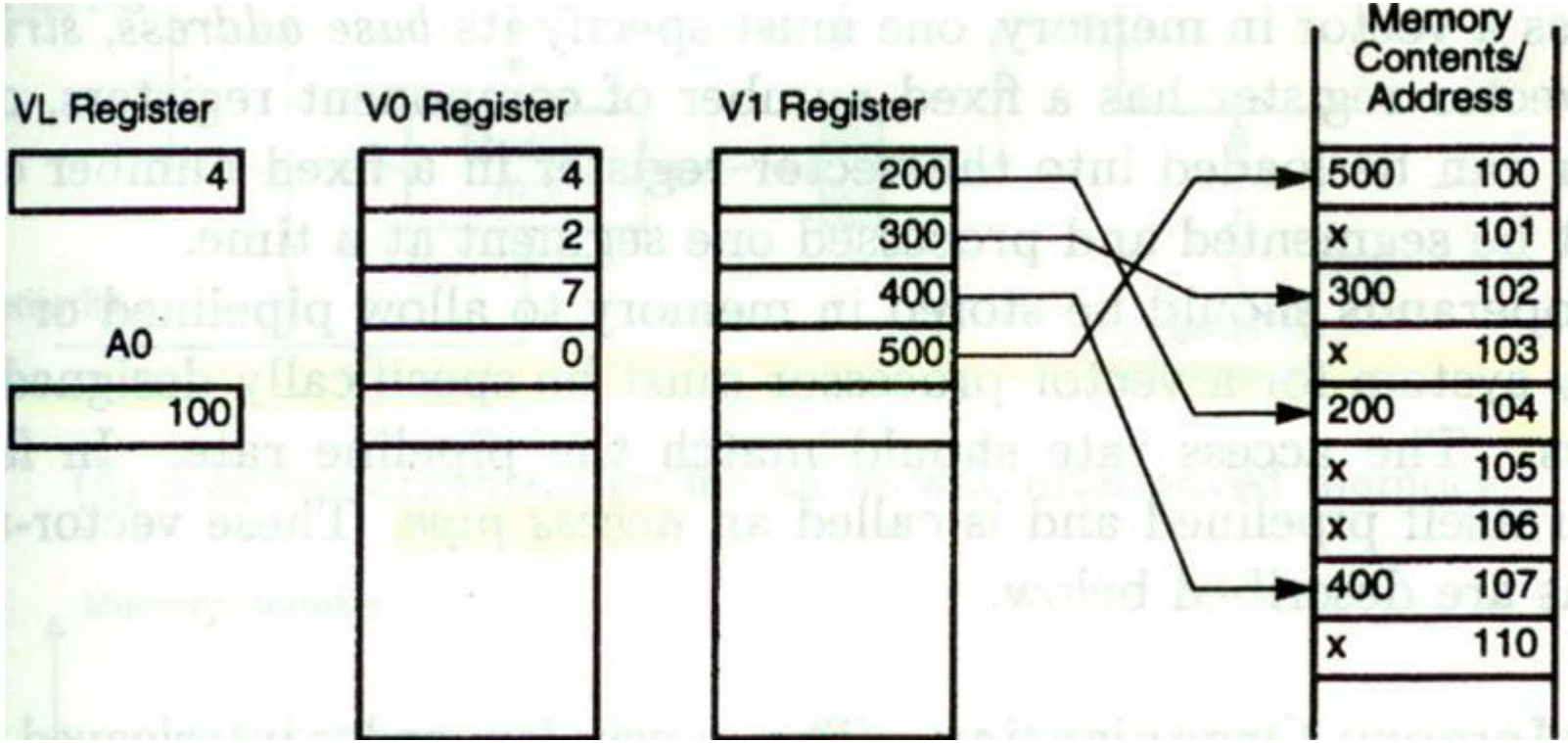
$$M \rightarrow V_1 \times V_0$$



Vector instruction types

- Scatter instruction - does the opposite, …

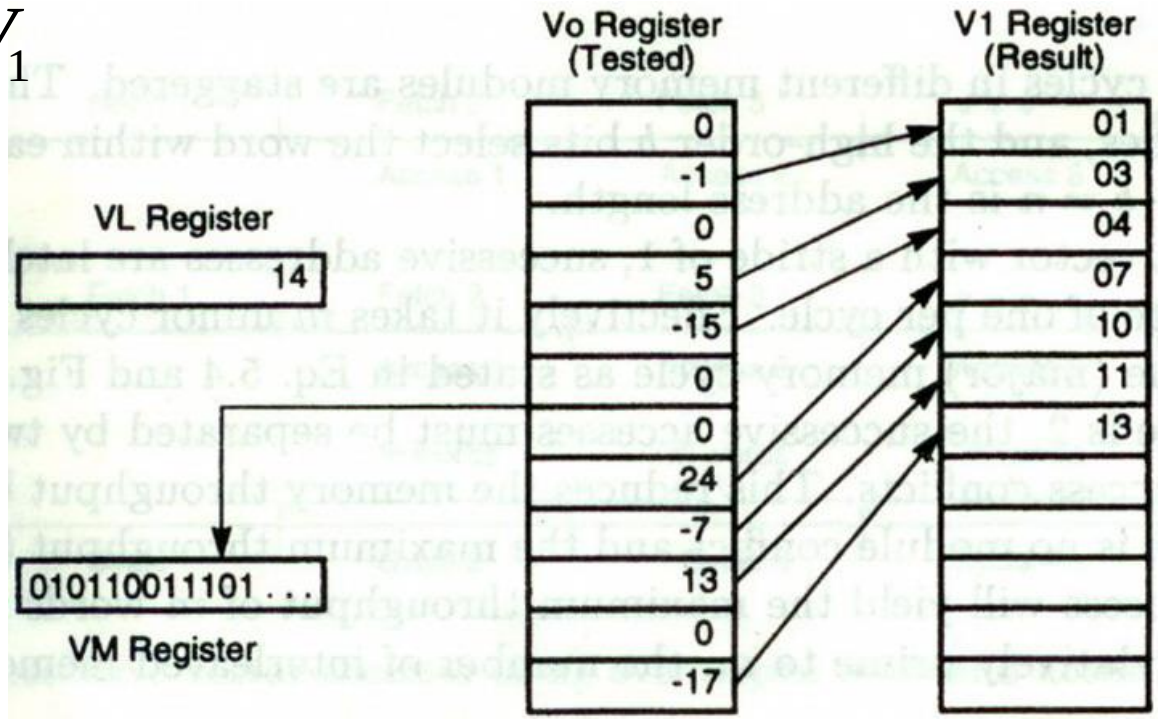
$$V_1 \times V_0 \rightarrow M$$



Vector instruction types

- Masking instructions
  - Use a mask vector to compress or to expand a vector to a shorter or longer vector

$$V_0 \times V_m \rightarrow V_1$$



向量处理方式

- 向量处理方式，如D=A×(B＋C)
  - A、B、C、D —— 长度为N的向量
- 横向(水平)处理方式
  - 向量计算是按行的方式从左到右横向地进行。
    - 先计算：　$d_1 \leftarrow a_1 \times (b_1 + c_1)$
    - 再计算：　$d_2 \leftarrow a_2 \times (b_2 + c_2)$
    - ……
    - 最后计算：　$d_N \leftarrow a_N \times (b_N + c_N)$
  - 组成循环程序进行处理。

$$k_i \leftarrow b_i + c_i$$
$$d_i \leftarrow k_i \times a_i$$

    - 数据相关：*N*次　　功能切换：*2N*次
  - 不适合于向量处理机的并行处理

向量处理方式

- 纵向 (垂直)处理方式
  - 向量计算是按列的方式从上到下纵向地进行。

$$\text{先计算}\begin{cases} k_1 \leftarrow b_1 + c_1 \\ \cdots\cdots \\ k_N \leftarrow b_N + c_N \end{cases} \qquad \text{再计算}\begin{cases} d_1 \leftarrow k_1 \times a_1 \\ \cdots\cdots \\ d_N \leftarrow k_N \times a_N \end{cases}$$

  - 表示成向量指令：

$$K = B + C$$

$$D = K \times A$$

    - 两条向量指令之间：

      数据相关：1次　　功能切换：1次

  - 向量处理机方式

向量处理方式

- 纵横 (分组)处理方式
  - 又称为分组处理方式。
  - 把向量分成若干组，组内按纵向方式处理，依次处理各组。
  - 对于上述的例子，设：

$$N = S \times n + r$$

    - 其中N为向量长度，S为组数，n为每组的长度，r为余数。
    - 若余下的r个数也作为一组处理，则共有$S+1$组。

向量处理方式

- 运算过程为：
  - 先算第1组：

$$k_{1\sim n} \leftarrow b_{1\sim n} + c_{1\sim n}$$

$$d_{1\sim n} \leftarrow k_{1\sim n} \times a_{1\sim n}$$

  - 再算第2组：

$$k_{(n+1)\sim 2n} \leftarrow b_{(n+1)\sim 2n} + c_{(n+1)\sim 2n}$$

$$d_{(n+1)\sim 2n} \leftarrow k_{(n+1)\sim 2n} \times a_{(n+1)\sim 2n}$$

  - 依次进行下去，直到最后一组：第$S+1$组。
  - 每组内各用两条向量指令。

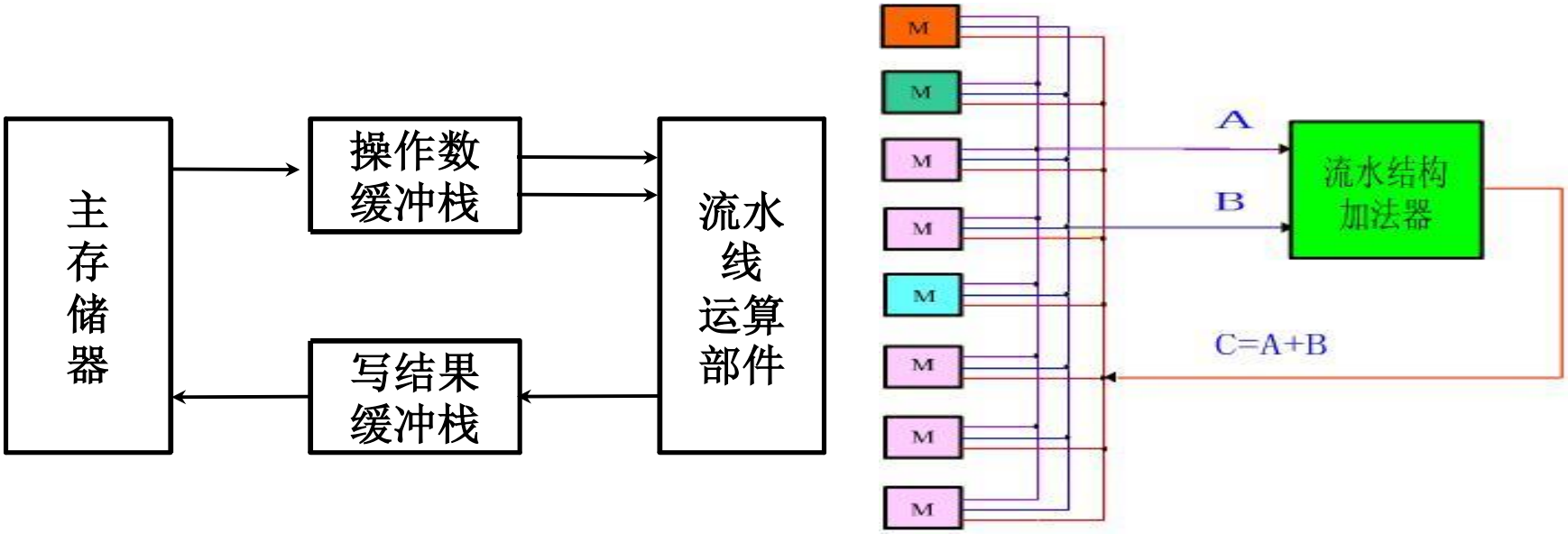
    数据相关：1次　　功能切换：2次

向量处理机及其系统结构

- 在流水线处理机中，设置向量数据表示和相应的向量指令，称为向量处理机。
- 不具有向量数据表示和相应的向量指令的流水线处理机，称为标量处理机。

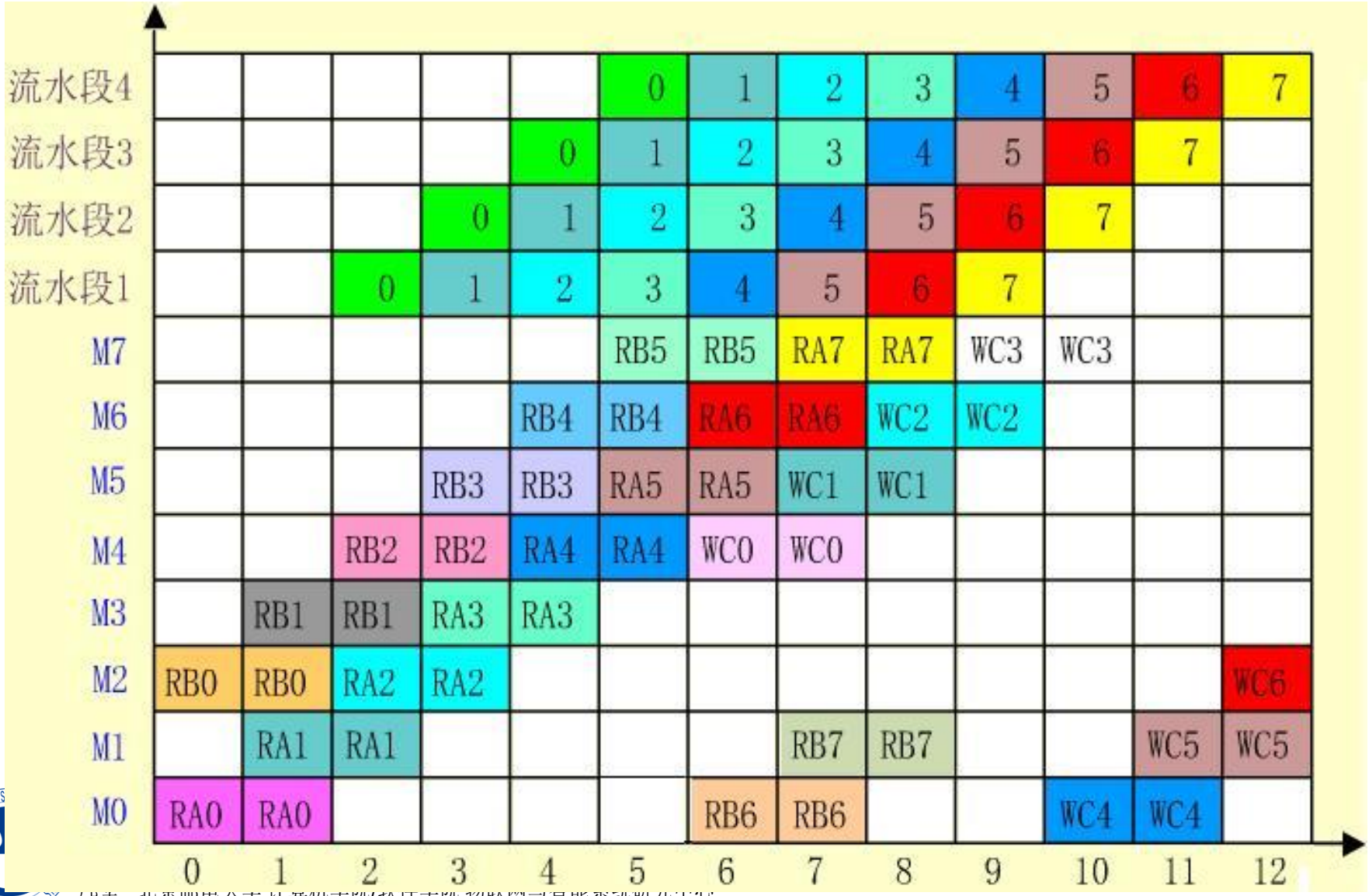
- Vector processor architecture
  - Memory-to-memory architecture
  - Register-to-register architecture

Memory-to-memory

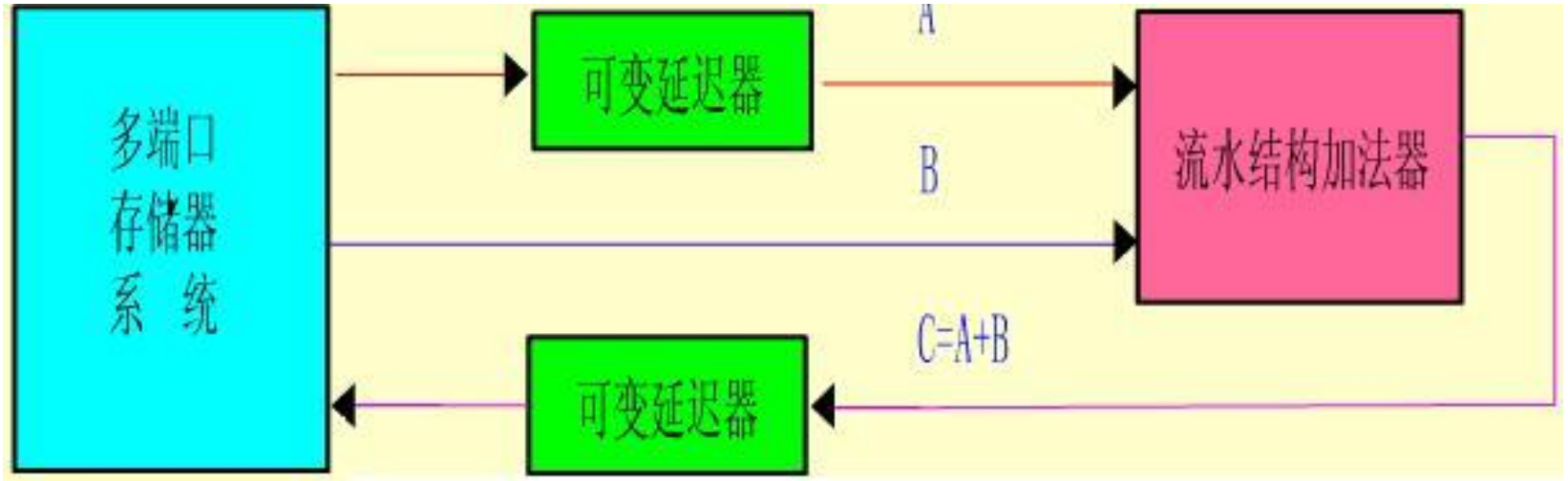


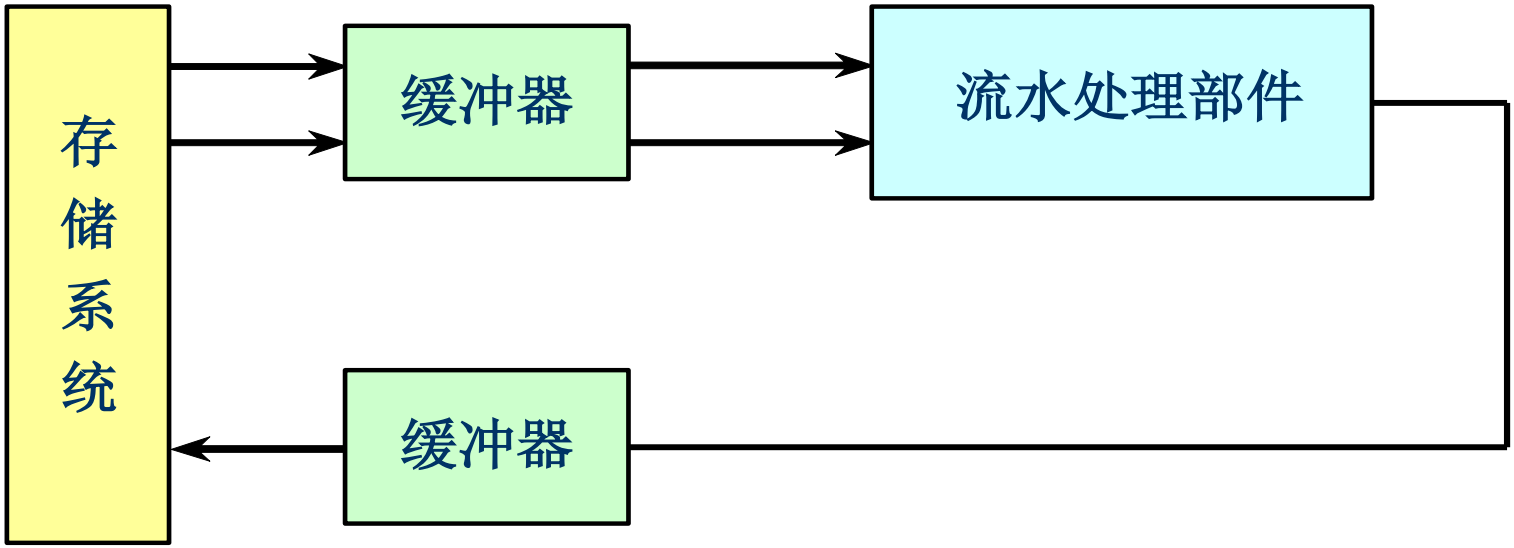
- 采用多个存储体交叉和并行访问来提高存储器速度
- 操作数缓冲栈和写结果缓冲栈主要用于解决访问存储器冲突
  - 如STAR-100，CYBER-205。 STAR-100采用32个存储体交叉访问，每个存储体，每个周期并行读出8个64位数据

Memory-to-memory

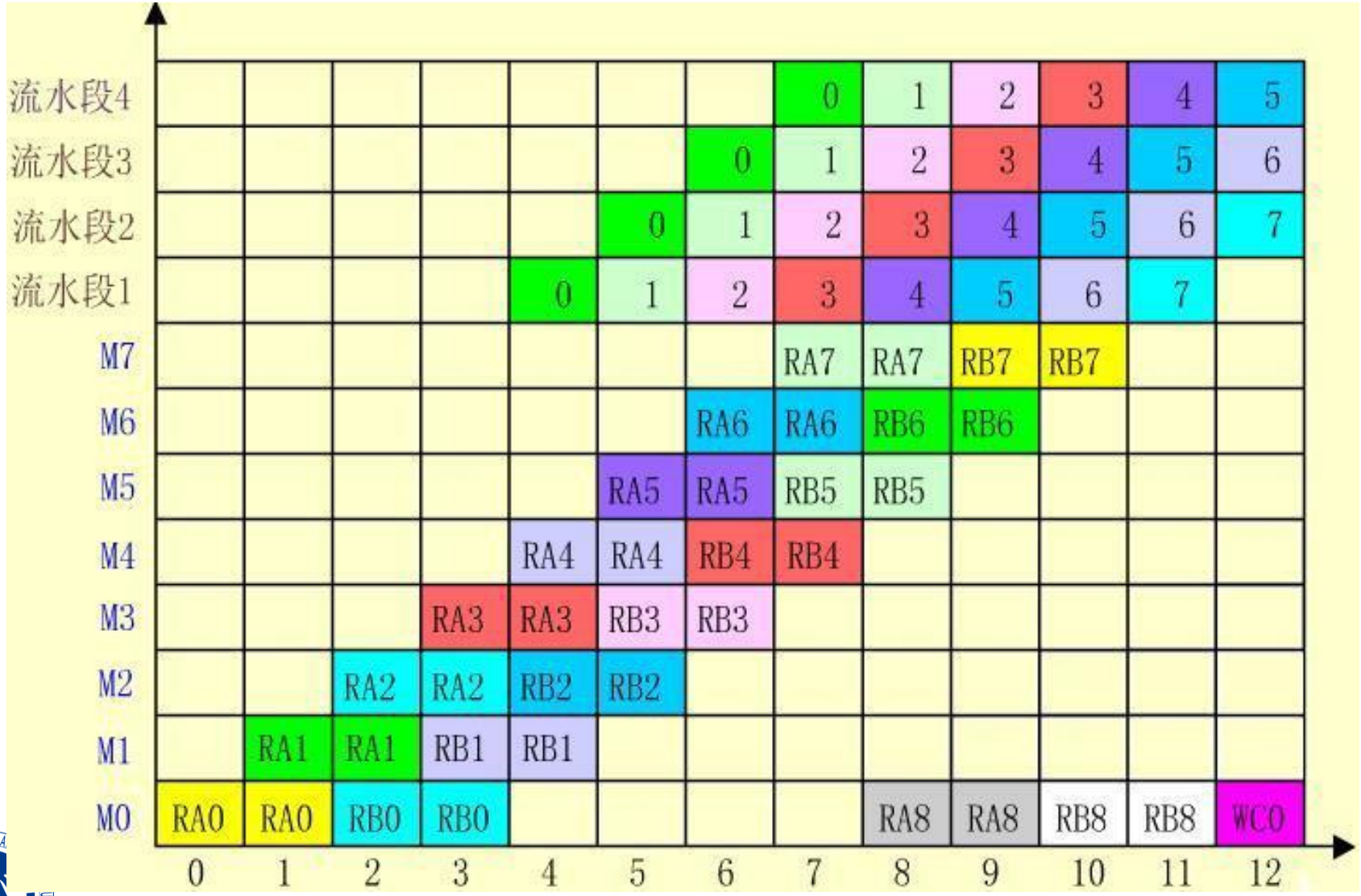


Memory-to-memory





Memory-to-memory

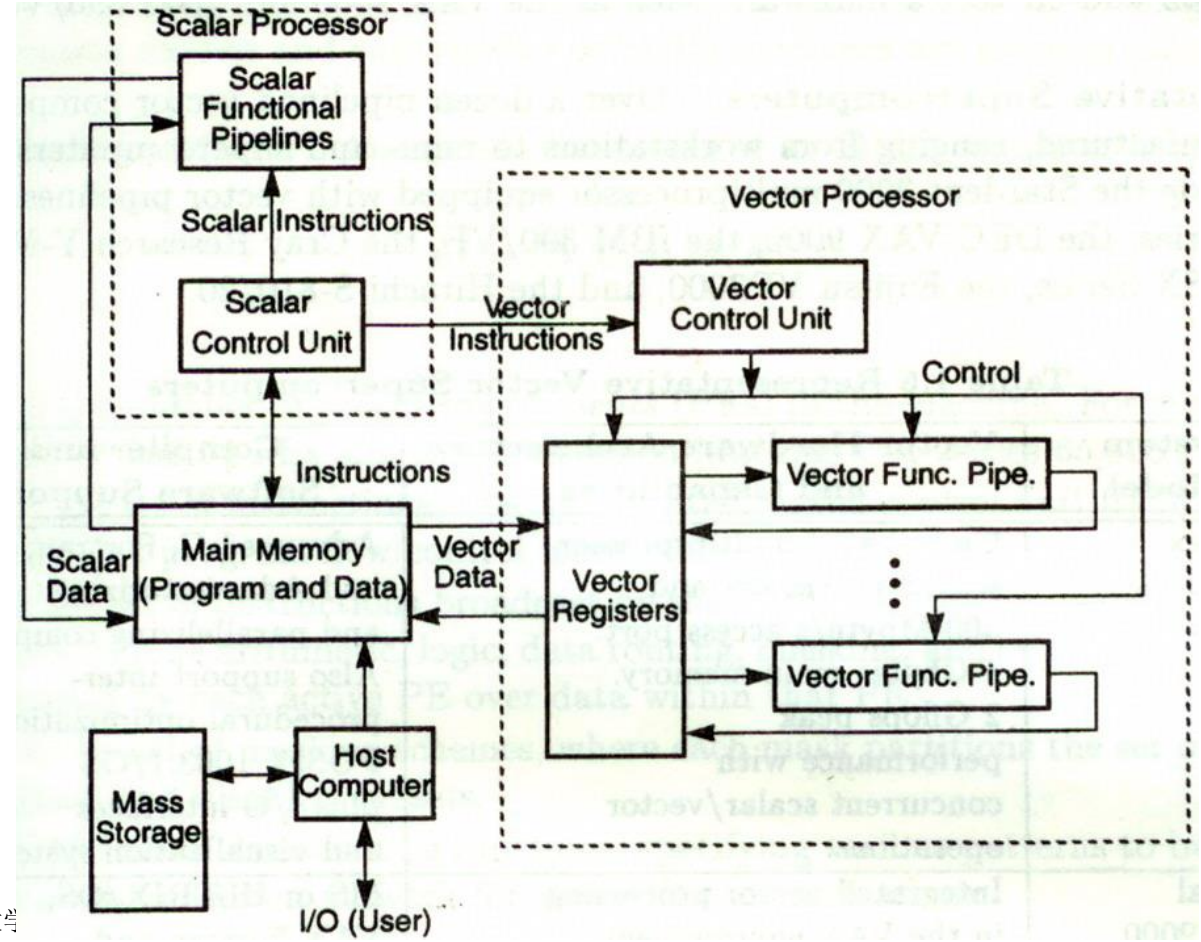


Register-to-Register

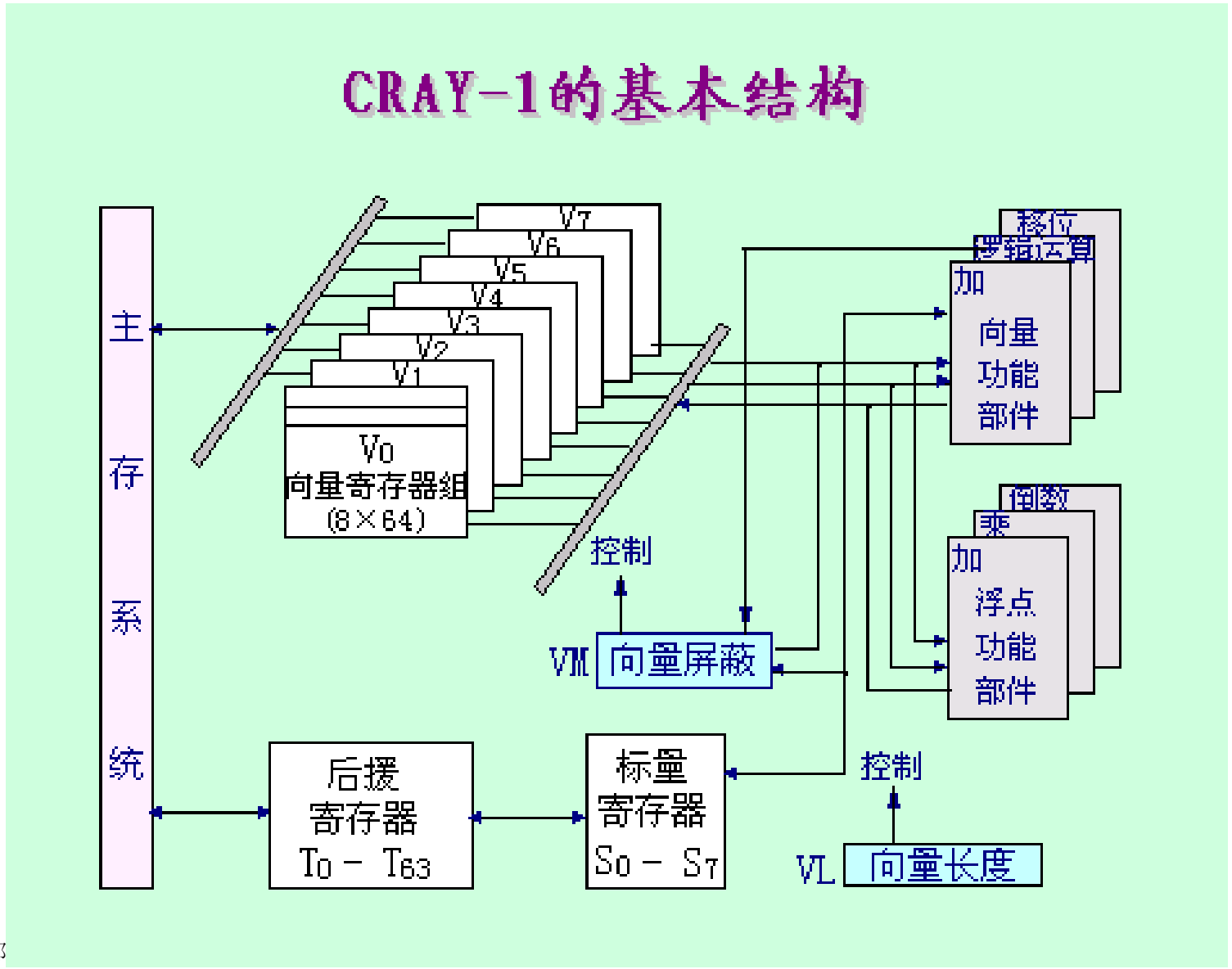
- 对处理机结构的要求：寄存器—寄存器结构
  - 设置能快速访问的向量寄存器，用于存放源向量、目的向量及中间结果，让运算部件的输入、输出端都与向量寄存器相联，构成寄存器—寄存器型操作的运算流水线。
  - 典型的寄存器—寄存器结构的向量处理机
    - 美国的CRAY-1、我国的YH-1巨型机
    - 关于存储体：CRAY-1有64个存储体，每个处理机访问4个存储体。YH-1有37个存储体

Register-to-Register

- 设置能快速访问的向量寄存器，用于存放源向量、的向量及中间结果，让运算部件的输入、输出端都与向量寄存器相联，构成寄存器—寄存器型操作的运算流水线。



Register-to-Register



Register-to-Register

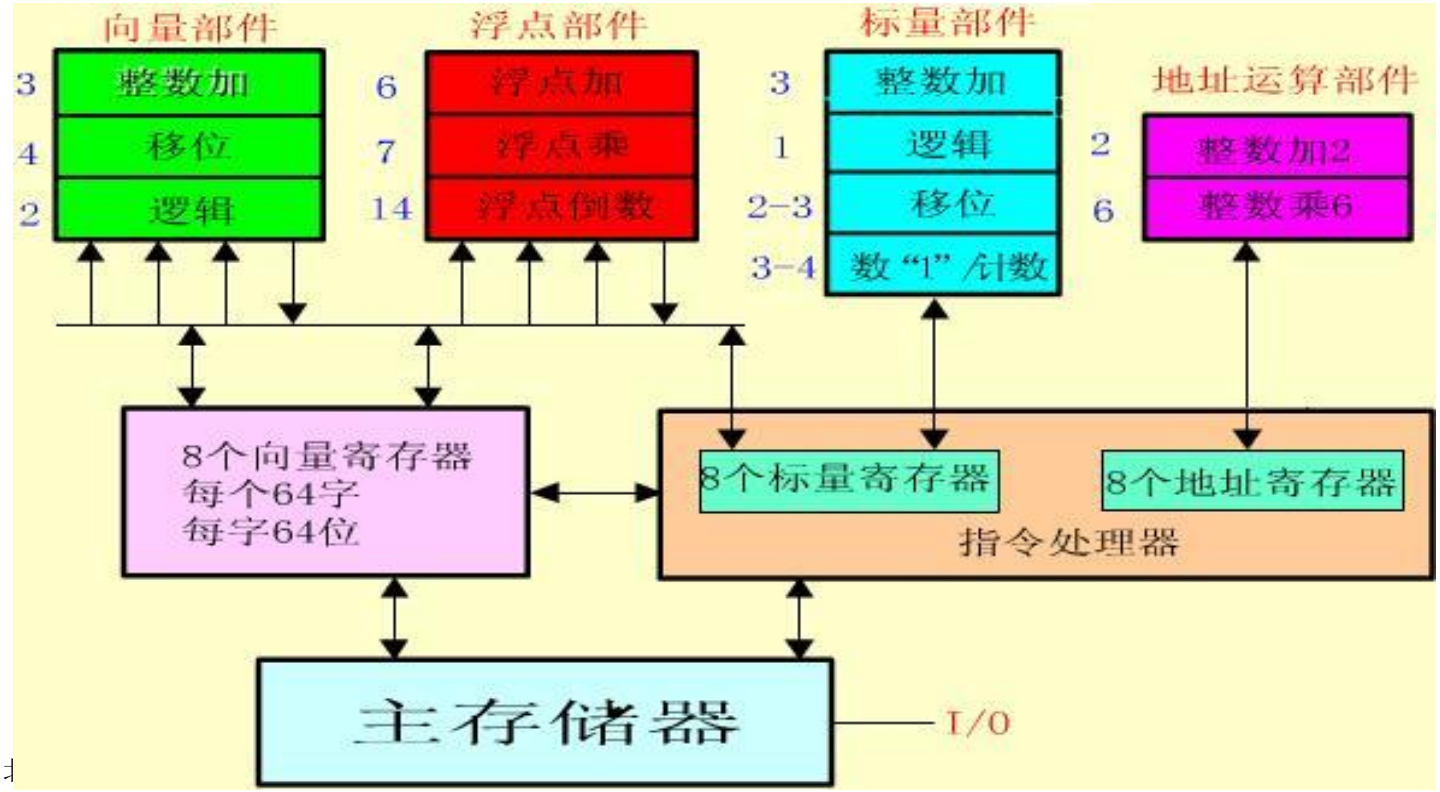
- CRAY-1：
  - 1976年，秒1亿次浮点运算，时钟周期：12.5 ns
  - 共有12条可并行工作的单功能流水线，可分别流水地进行地址、向量、标量的各种运算。
- 6个单功能流水部件：进行向量运算
  - 整数加（3拍）
  - 逻辑运算（2拍）
  - 移位（4拍）
  - 浮点加（6拍）
  - 浮点乘（7拍）
  - 浮点迭代求倒数（14拍）

Register-to-Register

- 向量寄存组V
  - 由512个64位的寄存器组成，分成8块
  - 编号：V0～V7
  - 每一个块称为一个向量寄存器，可存放一个长度（即元素个数）不超过64的向量
  - 每个向量寄存器可以每拍向功能部件提供一个数据元素，或者每拍接收一个从功能部件来的结果元素。
- 标量寄存器S和快速暂存器T
  - 标量寄存器有8个：S0～S7　64位
  - 快速暂存器T用于在标量寄存器和存储器之间提供缓冲。

Register-to-Register

- CRAY-1向量处理的一个显著特点
  - 每个向量寄存器Vi都有连到6个向量功能部件的单独总线。
  - 每个向量功能部件也都有把运算结果送回向量寄存器组的总线。

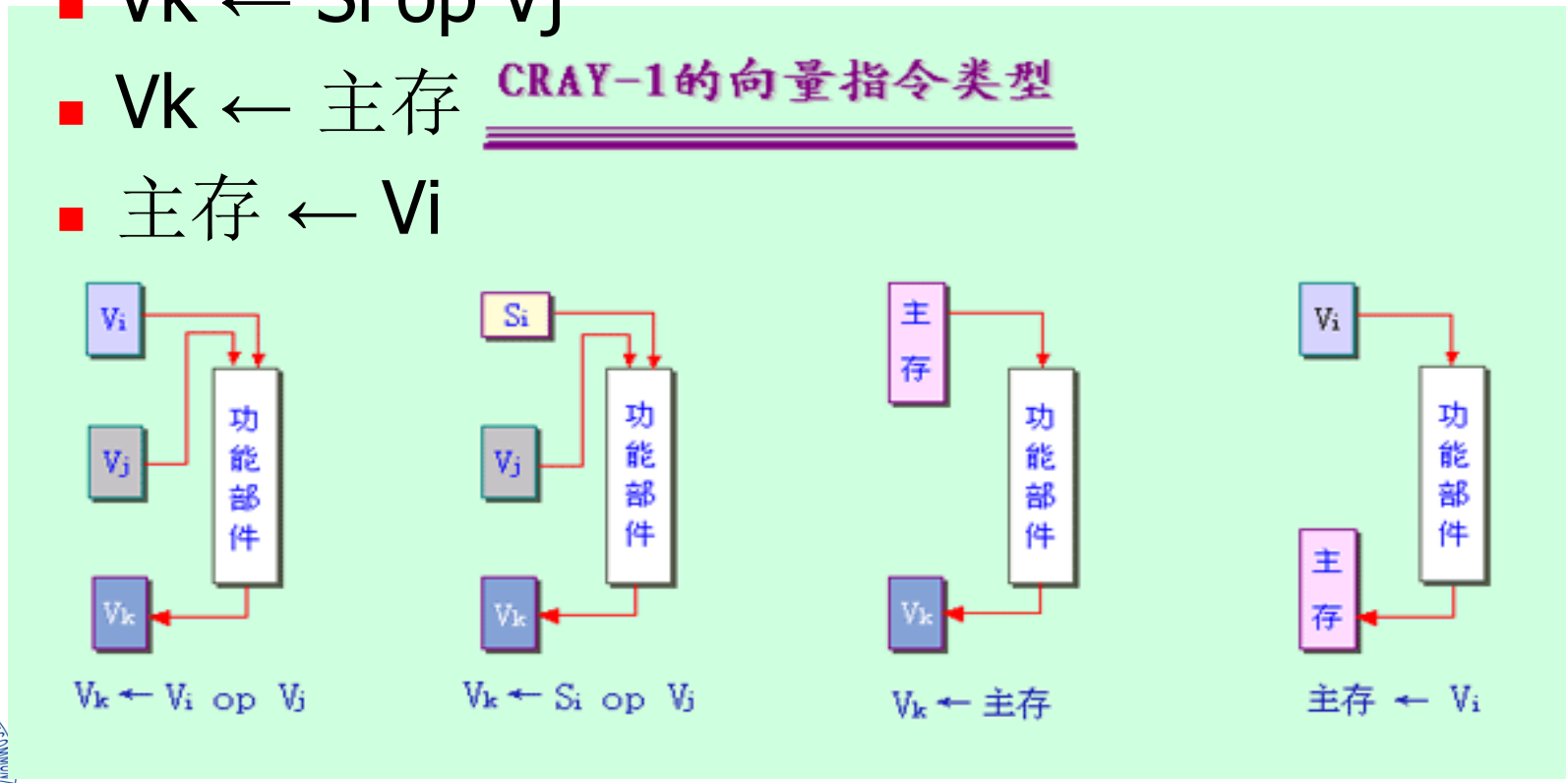


Register-to-Register

- 只要不出现Vi冲突和功能部件冲突，各Vi之间和各功能部件之间都能并行工作，大大加快了向量指令的处理。
  - Vi冲突：并行工作的各向量指令的源向量或结果向量使用了相同的Vi。
    - V4←V1+V2　　V5←V1+V2
    - V5←V2^V3　　V5←V3^V4
  - 功能部件冲突：并行工作的各向量指令要使用同一个功能部件。
    - V3←V1×V2
    - V5←V4×V6

Register-to-Register

- CRAY-1向量指令类型
  - Vk ← Vi op Vj
  - Vk ← Si op Vj
  - Vk ← 主存
  - 主存 ← Vi



Register-to-Register

- 提高向量处理机性能的方法
  - 设置多个功能部件，使它们并行工作。
  - 采用链接技术，加快一串向量指令的执行。
  - 采用循环开采技术，加快循环的处理。
  - 采用多处理机系统，进一步提高性能。

Register-to-Register

- 设置多个功能部件
  - 这些部件能并行工作，并各自按流水方式工作，从而形成了多条并行工作的运算操作流水线。

- But
  - LDV V3, M(A)
  - ADDV V2, V1, V0
  - MULV V4, V2, V3

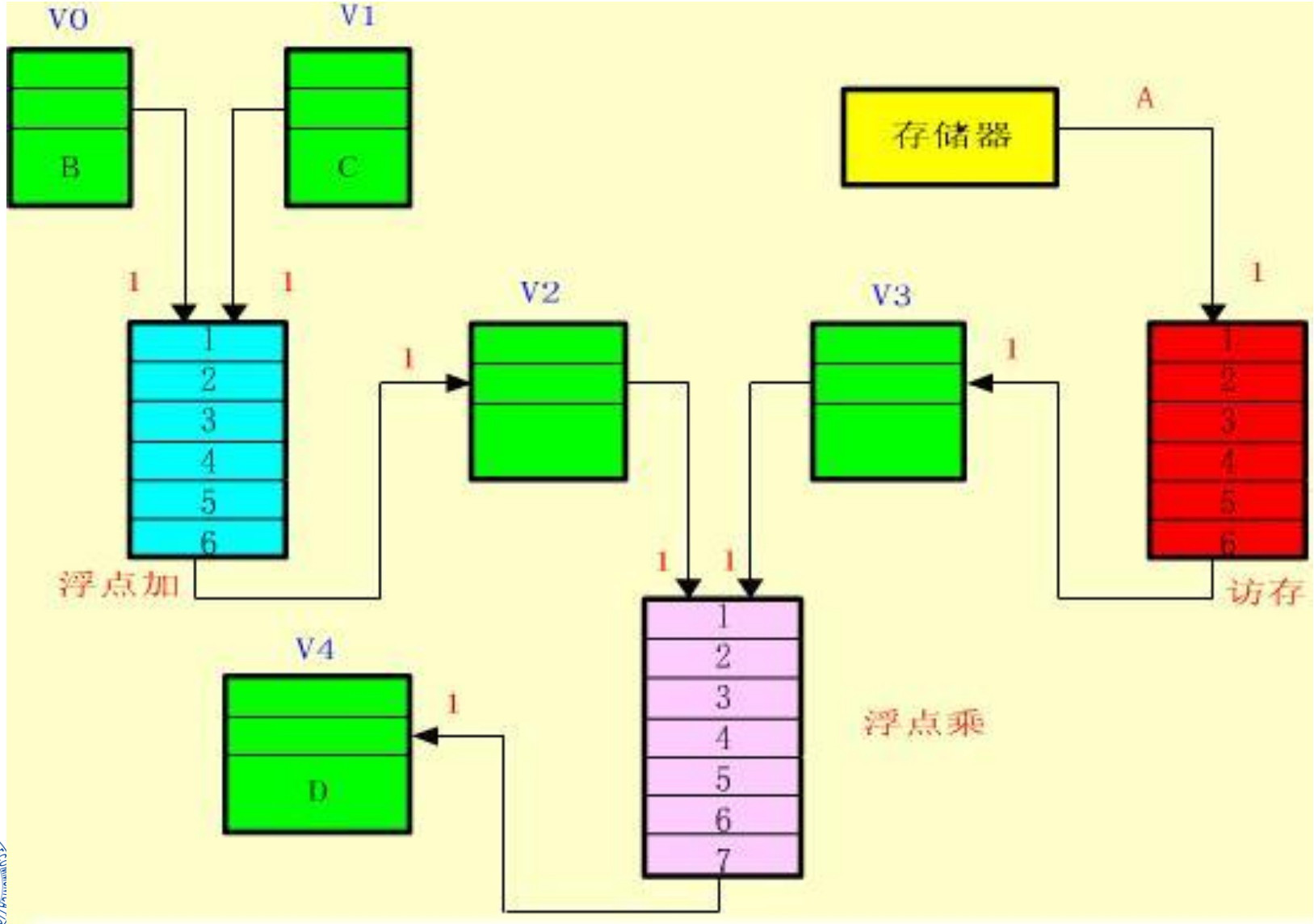
- Vector function units tend to process one element/cycle
  - long latency. Next vector inst has to wait for first to complete, even though vector elements are independent.

Register-to-Register

- 链接技术 - Chain (Forward)
  - 具有先写后读相关的两条指令，在不出现功能部件冲突和源向量冲突的情况下，可以把功能部件链接起来进行流水处理，以达到加快执行的目的。
  - 链接特性的实质 - 把流水线定向的思想引入到向量执行过程的结果。

- LDV V3, M(A)
- ADDV V2, V1, V0
- MULV V4, V2, V3
  - 第1、2条指令无向量寄存器使用冲突，也无功能部件使用冲突，因而可以并行执行
  - 第3条指令与第1、2条指令均存在先写后读相关，因而可将第3条指令与第1、2条指令链接执行

Pipeline chaining



Register-to-Register

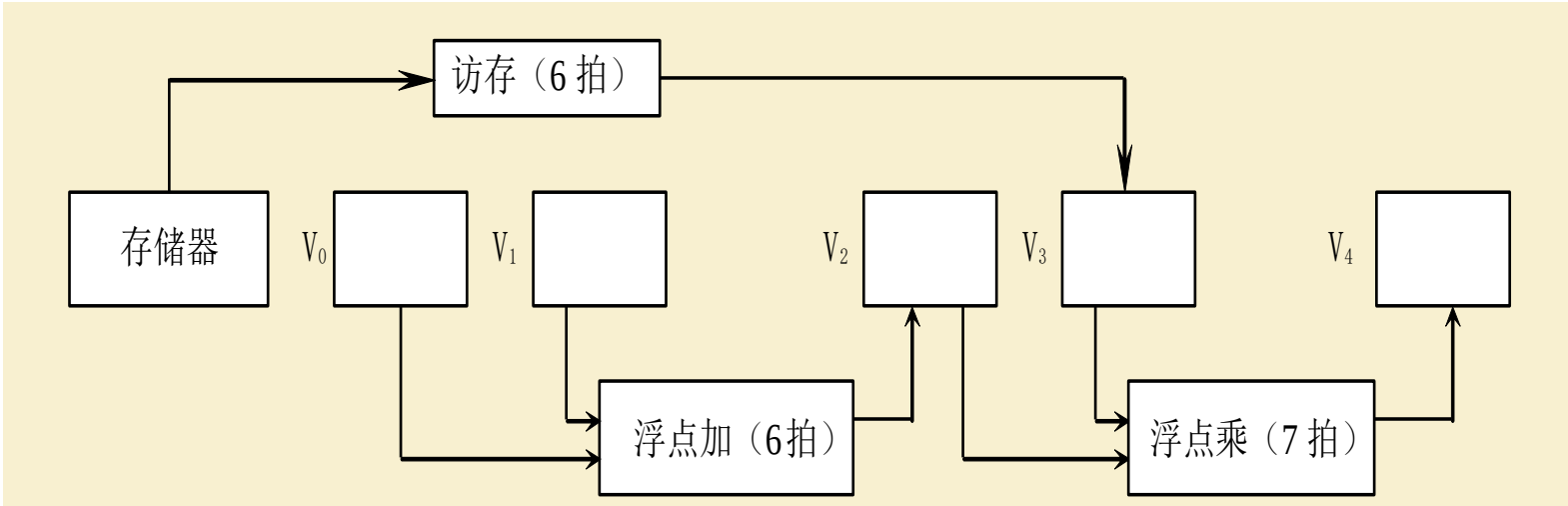
- 例：在CRAY-1上用链接技术进行向量运算D=A×(B+C)。假设向量长度N≤64，向量元素为浮点数，且向量B、C已存放在V0和V1中。画出链接示意图，并分析非链接执行和链接执行两种情况下的执行时间。
- 假设：把向量数据元素送往向量功能部件以及把结果存入向量寄存器需要一拍时间，从存储器中把数据送入访存功能部件需要一拍时间。
- 解：用以下三条向量完成上述运算：
  - V3 ← 存储器　　// 访存取向量A
  - V2 ← V0 ＋ V1　　// 向量B和向量C进行浮点加
  - V4 ← V2 × V3　　// 浮点乘，结果存入V4

Register-to-Register

- 3条指令全部用串行方法执行，则执行时间为：
  - [(1＋6＋1)＋N－1]＋[(1＋6＋1)＋N－1] ＋[(1＋7＋1)＋N－1] = 3N ＋22 (拍)
- 前两条指令并行执行，然后再串行执行第3条指令，则执行时间为：
  - [(1＋6＋1)＋N－1]＋[(1＋7＋1)＋N－1] = 2N ＋15 (拍)

Register-to-Register

- 第1、2条向量指令并行执行，并与第3条指令链接执行。
  - [(1＋6＋1)]＋ [(1＋7＋1)] ＋(N－1) = N＋16 (拍)



Register-to-Register

- 进行向量链接的要求
  - 保证：无向量寄存器使用冲突和无功能部件使用冲突
  - 只有在前一条指令的第一个结果元素送入结果向量寄存器的那一个时钟周期才可以进行链接
  - 当一条向量指令的两个源操作数分别是两条先行指令的结果寄存器时，要求先行的两条指令产生运算结果的时间必须相等，即要求有关功能部件的通过时间相等
  - 要进行链接执行的向量指令的向量长度必须相等，否则无法进行链接

Register-to-Register

- 分段开采技术 - 向量的长度大于向量寄存器的长度
  - 当向量的长度大于向量寄存器的长度时，必须把长向量分成长度固定的段，然后循环分段处理，每一次循环只处理一个向量段。
  - 这种技术称为分段开采技术 - 由系统硬件和软件控制完成，对程序员是透明的。

举例

- 设A和B是长度为N的向量，考虑在Cray-1向量处理器上实现以下的循环操作：

```
      DO   10 I = 1, N
10    A(I) = 5.0 * B(I) + 1.0
```

- 当N ≤64时，可以用以下指令序列：

```
S1 ← 5.0          ; 将常数5.0送入标量寄存器S1
S2 ← 1.0          ; 将常数1.0送入标量寄存器S2
VL ← N            ; 在向量长度寄存器VL中设置向量长度N
V0 ← B            ; 从存储器中将向量B读入向量寄存器V0
V1 ← S1 × V0      ; 向量B中的每个元素分别和常数S1相乘
V2 ← S2 + V1      ; 向量V1中的每个元素分别和常数S2相加
A ← V2            ; 将计算结果从向量寄存器V2存入存储器的向量A
```

举例

- 当N ＞64时，就需要进行分段开采。
  - 循环次数K：

$$K = \left\lfloor \frac{N}{64} \right\rfloor$$

  - 余数L:

$$L = N - 64 \times \left\lfloor \frac{N}{64} \right\rfloor$$

举例

```
S1 ← 5.0        ；将常数5.0送入标量寄存器S1
S2 ← 1.0        ；将常数1.0送入标量寄存器S2
VL ← L          ；在向量长度寄存器VL中设置向量长度L
V0 ← B          ；从存储器中将向量B[0..L-1]读入向量
                ；    寄存器V0
V1 ← S1 * V0    ；向量B中的每个元素分别和常数S1相乘
V2 ← S2 + V1    ；向量V1中的每个元素分别和常数S2相加
A ← V2          ；将计算结果从向量寄存器V2存入存储器
                ；的向量A[0..L-1]
```

处理余数部分，计算L个元素

举例

For (I=0 to K-1) {

V_0 ← B ；从存储器中将向量B[L+I*64..L+I*64+63]

；读入向量寄存器V_0

V_1 ← S_1 * V_0 ；向量B中的每个元素分别和常数S_1

；相乘；

V_2 ← S_2 + V_1 ；向量V_1中的每个元素分别和常数S_2

；相加

A ← V_2 ；将计算结果V_2存入存储器的向量

； A[L+I*64… L+I*64+63]

}

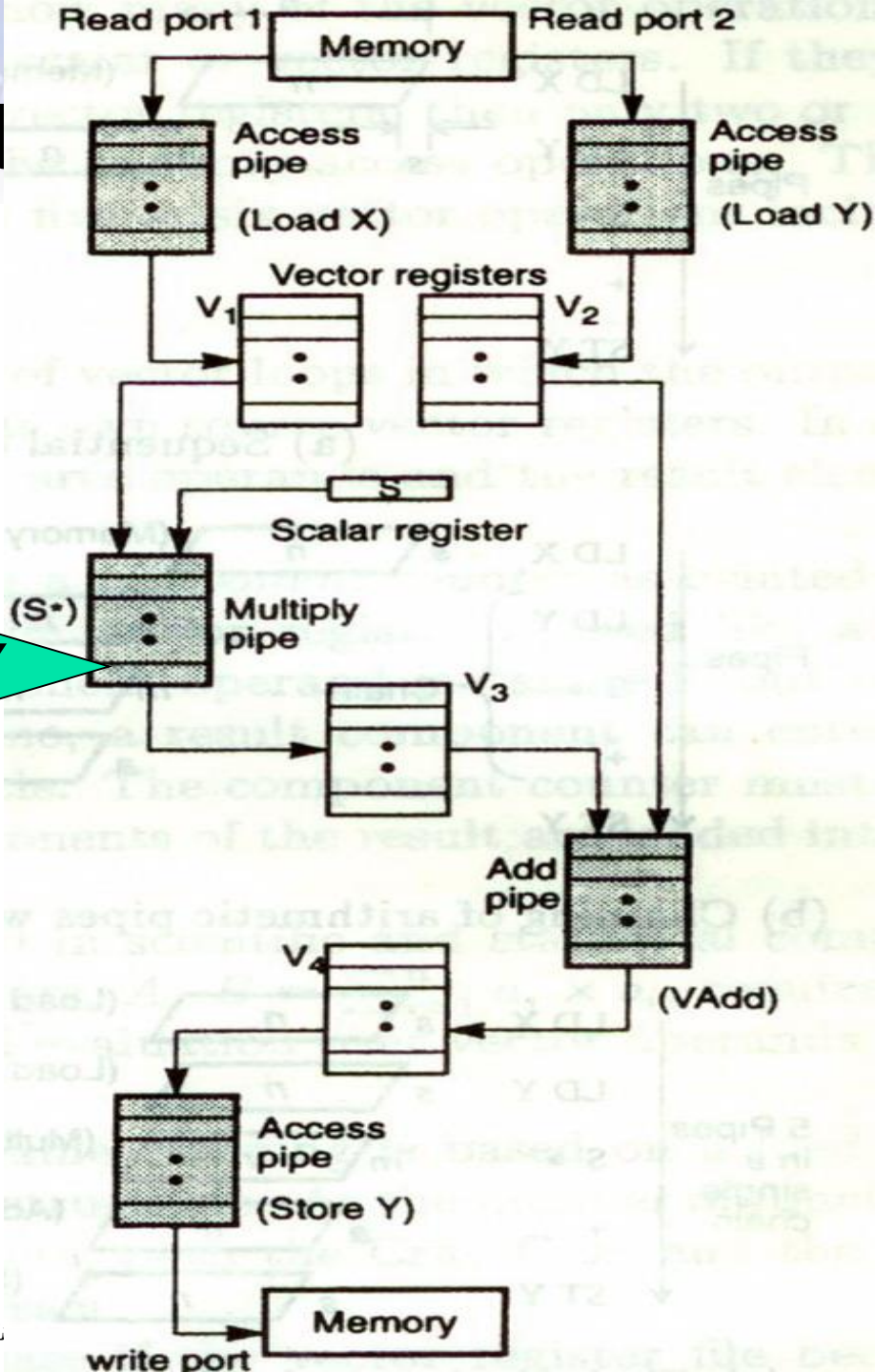
循环K次，分段处理

Register-to-Regist

- 采用多个访存部件
  - Cray-1 只有一个访存部件；
  - Cray X-MP有三个访存部件，两个用于向量load，一个用于向量store，并且三个部件可同时使用。

Y = s * X + Y

- 采用多处理机系统，例如：
  - CRAY-2
    - 包含了4个向量处理机
    - 浮点运算速度最高可达1800MFLOPS
  - CRAY Y-MP、C90
    - 最多可包含16个向量处理机



向量处理机的性能评价

- 向量指令的处理时间：

  $T_{vp} = T_s + T_e + (n - 1) T_c$

  - 其中：T_s为向量处理部件流水线的建立时间
  - T_e为第一对向量元素流水线的流过时间
  - T_c为流水线时钟周期

  如果以时钟周期折算：

  $T_{vp} = [s + e + (n - 1)] T_c$

  - s为T_s对应的时钟周期数，e为T_e对应的时钟周期数

  如果不考虑T_s，且令$T_{start} = e - 1$，则：

  $T_{vp} = (T_{start} + n) T_c$

  - T_{start} 为从一条向量指令开始到还差一个时钟周期产生结果所需的时钟周期数

向量处理机的性能评价

- 一组向量指令的总执行时间：
  - 编队：在同一个时钟周期内可一起开始执行的几条向量指令
    - 同一个编队中的指令一定不存在功能部件冲突和数据相关。
  - 例：

		编队
LV	V1, Rx	(1)
MULTSV	V2, R0, V1	(2)
LV	V3, Ry	(2)
ADDV	V4, V2, V3	(3)
SV	V4, Ry	(4)

向量处理机的性能评价

一组向量的执行时间

$$T_{all} = \sum_{i=1}^{m} T_{vp}^{(i)}$$

编队数

编队i的执行时间

- 若一个编队有若干指令，其执行时间由执行时间最长的指令确定

$$T_{all} = \sum_{i=1}^{m} T_{vp}^{(i)}$$

$$= \sum_{i=1}^{m} \left[T_{start}^{(i)} + n \right] T_c = \left[\sum_{i=1}^{m} T_{start}^{(i)} + mn \right] T_c = [T_{start} + mn] T_c$$

该组指令总启动时间

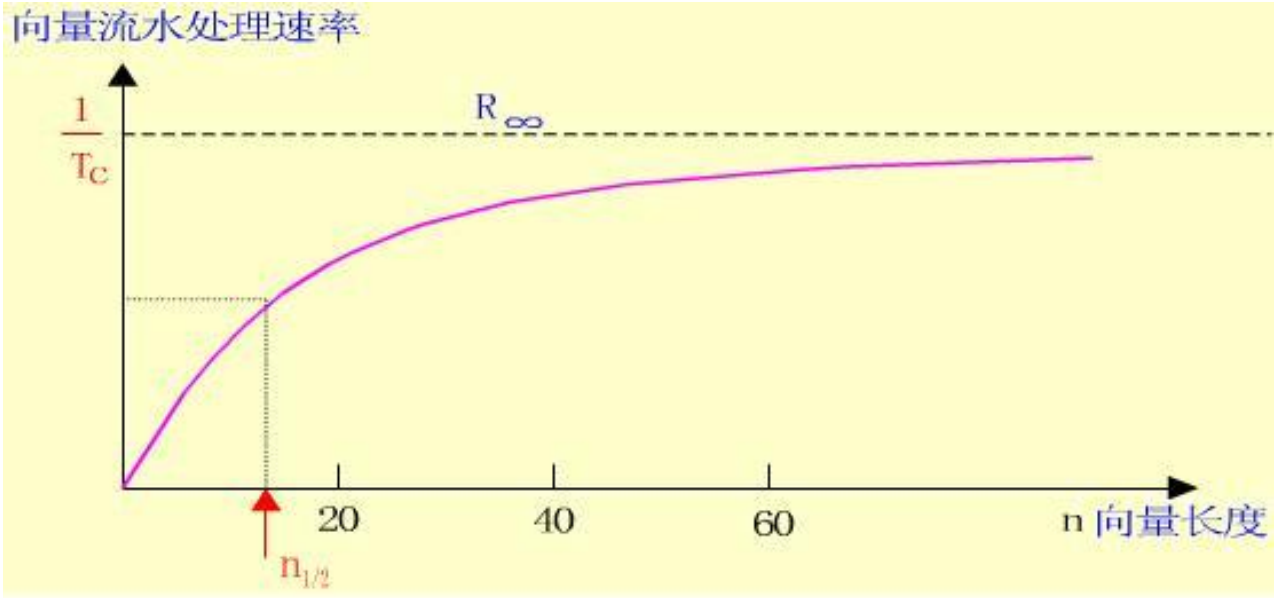
编队i指令启动时间的最大值

向量处理机的性能评价

- 分段开采时一组向量的总执行时间
  - 向量长度n若大于寄存器长度MVL，需要分段开采。所需额外的时间为Tloop
  - 若有m个编队，假设：$\left\lfloor \frac{n}{MVL} \right\rfloor = p$ 余数 $= q$
  - 最后一次循环：$T_{last} = T_{start} + T_{loop} + m \times q$
  - 其他循环：$T_{last} = T_{start} + T_{loop} + m \times \mathrm{MVL}$
  - 总执行时间：

$$T_{all} = T_{step} \times p + T_{last}$$
$$= \left(T_{start} + T_{loop} + m \times MVL\right) \times p + \left(T_{start} + T_{loop} + m \times q\right)$$
$$= (p+1) \times \left(T_{start} + T_{loop}\right) + \mathrm{m}(\mathrm{MVL} \times p + q)$$
$$= \left\lceil \frac{n}{MVL} \right\rceil \times \left(T_{start} + T_{loop}\right) + mn$$

向量处理机的性能评价



- R_∞ = FLOPS rate with infinite-length vectors

$$R_\infty = \lim_{n\to\infty} \frac{\text{向量指令序列中浮点运算次数} \times \text{时钟频率}}{\text{向量指令序列执行所需的时钟周期数}}$$

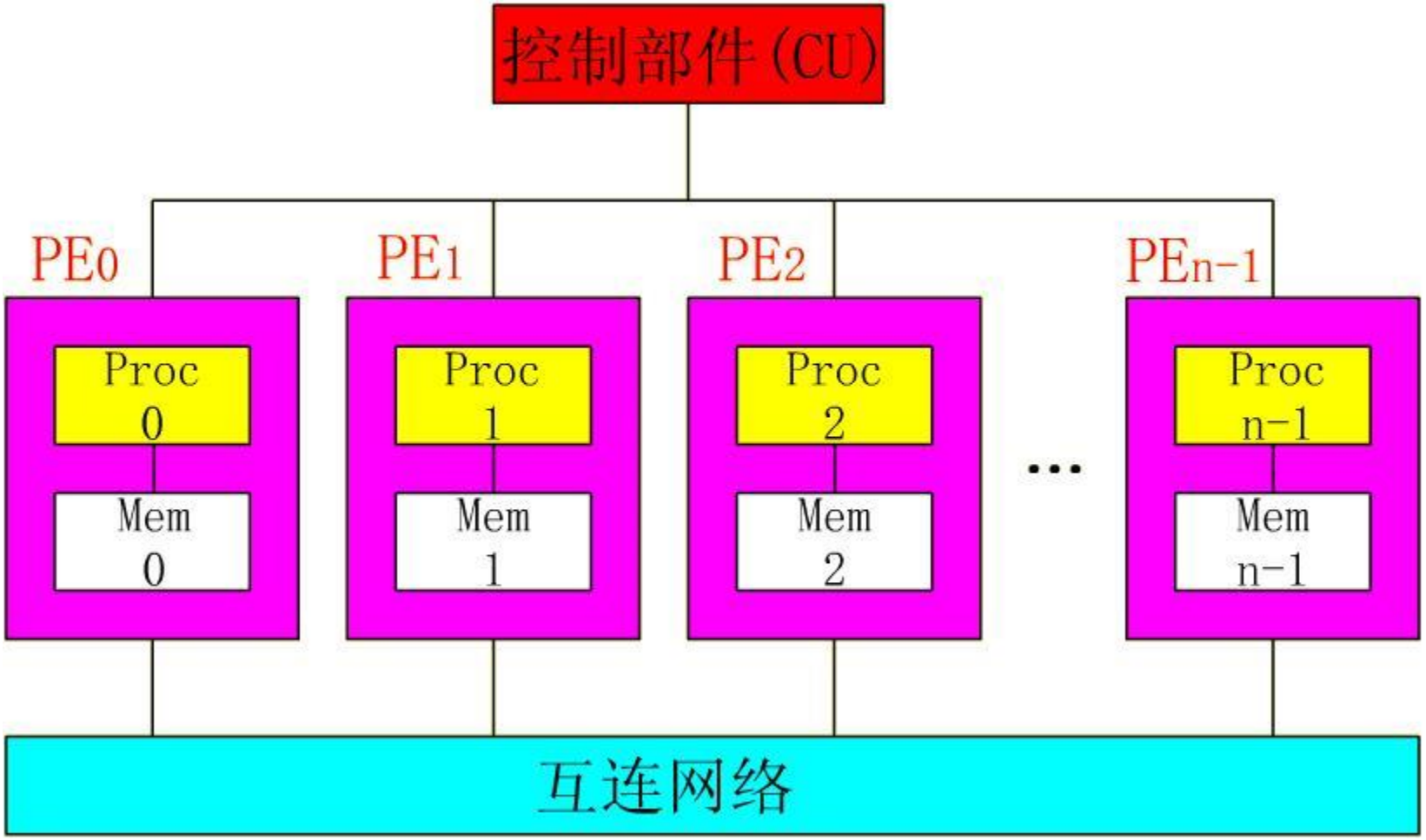
- $N_{1/2}$ = Vector length needed to get performance one-half of R_∞

向量处理机的性能评价

- 向量长度临界值n_v
  - 指向量流水方式的处理速度优于标量串行指令处理速度时所需的向量长度最小值

SIMD Supercomputer
阵列处理机

SIMD abstract module



使用资源重复的方法

SIMD machine model

- 5-tuple: **SIMD = 〈N, C, I, M, R〉**
  - **N** is the number of *processing elements* (PEs). Illiac IV has 64 PEs & CM-2 uses 65536 PEs.
  - **C** is the set of instructions directly executed by the *control init* (CU), including scalar and flow control insts.
  - **I** is the set of instructions broadcast by the CU to PEs for parallel execution (arithmetic, logic, data routing, masking by each active PE).
  - **M** is the set of masking schemes, to make PE enable/disable.
  - **R** is the set of data-routing functions, specifying various patterns to be set up in the network of PEs.

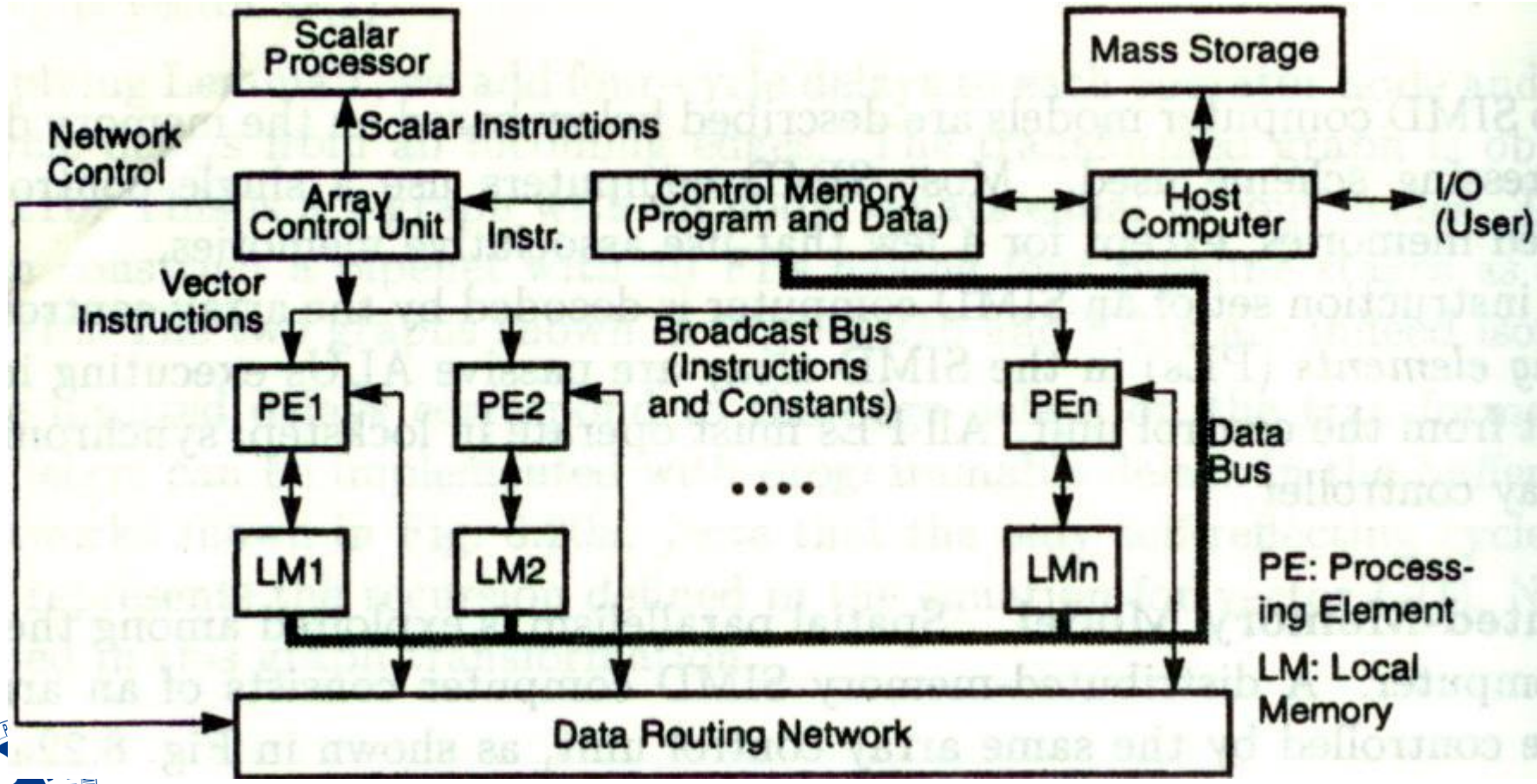
SIMD features

1. 阵列机采用**资源重复**。在系统中设置多个相同的处理单元来开发并行性，利用并行性中的同时性，所有处理单元必须同时进行相同操作；
2. 阵列机是**以某一类算法为背景的专用计算机**。由于阵列机中通常都采用简单、规整的互连网络来实现处理单元间的连接操作，从而**限定了它所适用的求解算法类别**。
3. 阵列机的研究**必须与并行算法的研究密切结合**；
4. 由于PE结构相同，属于**同构型**并行机。控制器实质上是一个标量处理机，而为了完成I/O操作以及操作系统的管理，尚需一个前端机，因此实际的阵列机系统是一个**异构型**多处理机系统。

特点

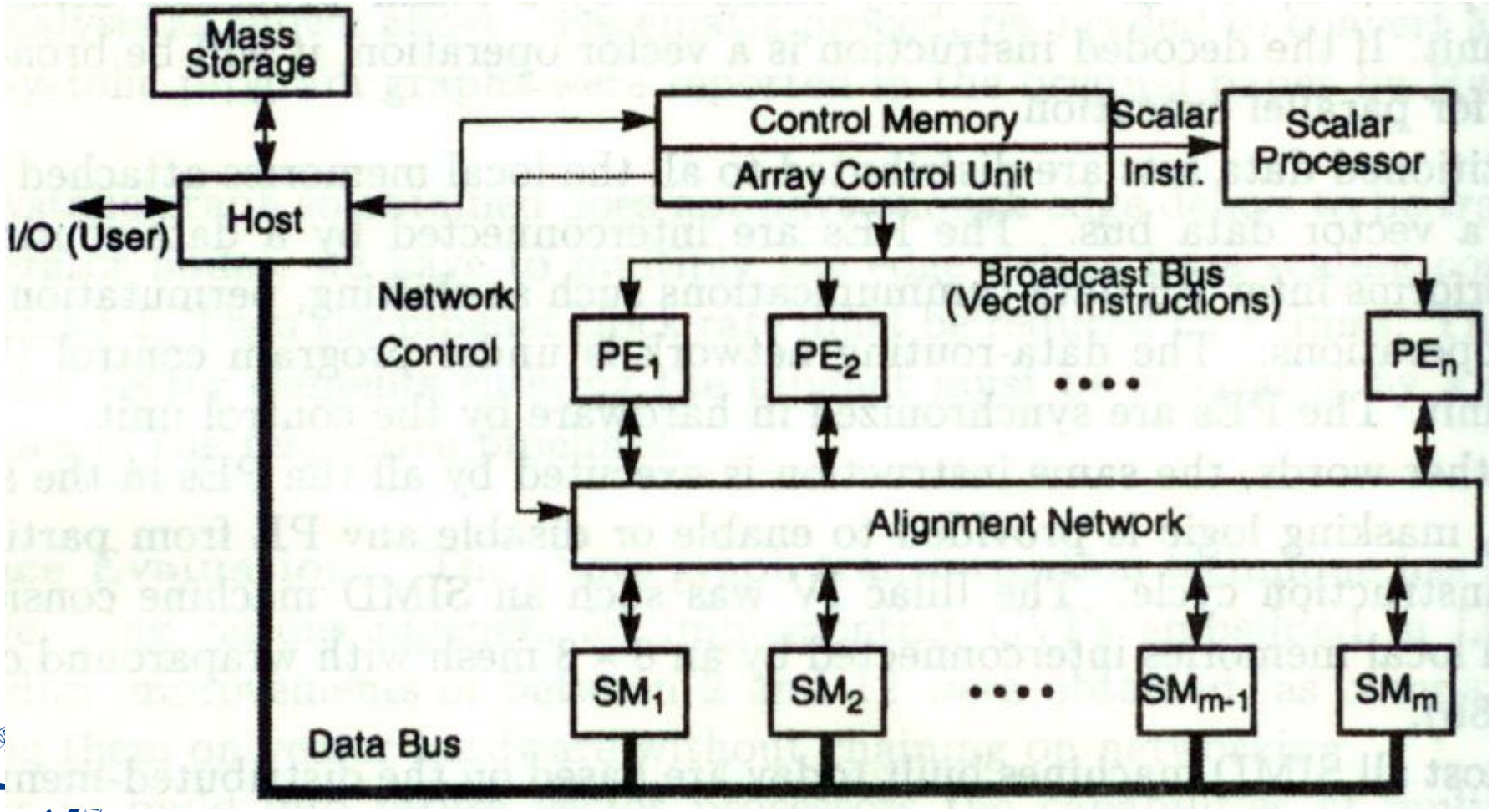
Distributed-Memory Model

- Illiac IV



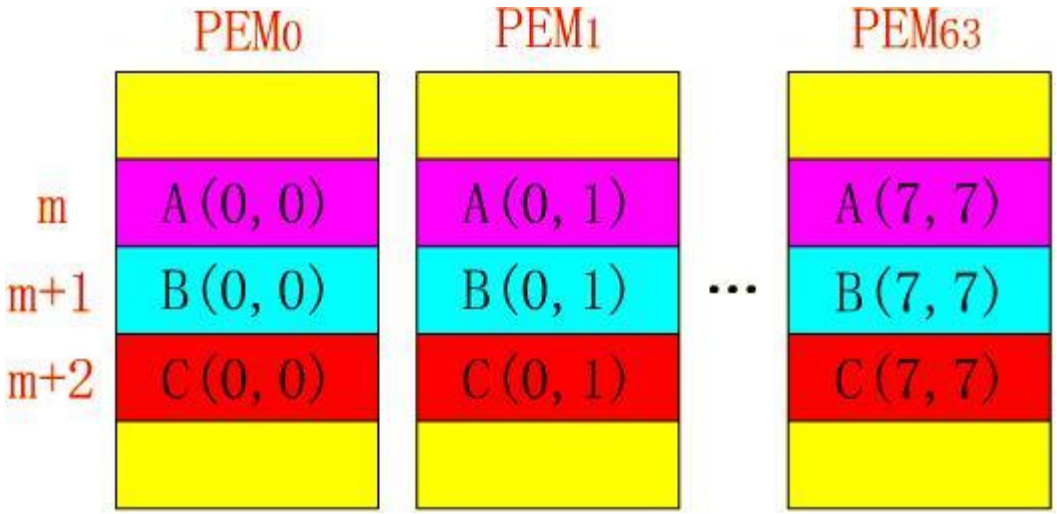
Shared-Memory Model

- BSP – Burroughs Scientific Processor



SIMD calculate example

- 矩阵加（8x8）




```
LDA   m      ;全部(m)由PEM送至PE的累加器
ADRN  m+1    ;全部(m+1)与累加器内容进行浮点加，结果送累加器
STA   m+2    ;全部累加器内容由PE传送到PEM的m+2单元
```

SIMD calculate example

- 递归折叠求和算法