Tema 2: PARALELISMO DE DATOS
Parte 1: Procesadores Vectoriales

Universidad de Cádiz

Referencias

(Hennesy J.L., Patterson D.A.)
Computer Architecture: A Quantitative Approach.
Morgan Kaufmann

Capitulo 10 o Apéndice F o G
(dependiendo de la versión)

ÍNDICE

Procesadores vectoriales

- **Introducción**
- **Ventaja de los operadores vectoriales**
- **Arquitectura**
  - Procesadores vectoriales
    - ✓ memoria-memoria y registro-registro
  - Procesadores con registros vectoriales
- **Problemas y soluciones**
  - Longitud del vector
  - Separación vectorial
- **Memoria**
- **Mejora del rendimiento**
  - Encadenamiento
  - Ejecución condicional
  - Matrices dispersas
- **Medición del rendimiento**

Introducción

- Se llama **vector** a una secuencia de datos escalares del mismo tipo almacenados en memoria, normalmente en posiciones contiguas, aunque no siempre.

- Un **procesador vectorial** es procesador que posee un conjunto de recursos para operar sobre vectores. Estas recursos consistirán en funciones aritméticas y lógicas aplicadas sobre las componentes de los vectores.

- Se llama **vectorización** a la conversión de un programa correspondiente a un procesador escalar a otro vectorial.

Introducción

Otra definición de **vector** mas acorde a este contexto es sería…

…. Estructura de datos que se puede definir por tres parámetros:

- **Dirección de comienzo**. Dirección en memoria del primer elemento del vector.
- **Longitud**. Numero de elementos del vector (no el tamaño en bytes)
- **Paso *(stride)***. Distancia en memoria de dos elementos consecutivos.

Ojo: El paso no tiene por que estar asociado con el tamaño del dato

Introducción

Ejemplo:

Datos almacenados en posiciones de memoria

3000, 3008, 3016, 3024, 3032

- Dirección de comienzo: 3000
- Longitud: 5
- Paso: 8

Estos datos, normalmente hay que tenerlos en cuenta al programar y pasarlos al procesador por registros especiales. (VL y VS - Vector Lenght y Stride)

Introducción

Un procesador **escalar** convierte un código como este:

```
do i = 0, N-1
      C(i) = A(i) + B(i)
enddo
```

En algo similar a esto:

```
buc:    FLD F1,A(R1)
        FLD F2,B(R1)
        FADD F3,F2,F1
        FST C(R1),F3
        ADDI R1,R1,#8
        SUBI R2,R2,#1
        BNZ R2,buc
```

Introducción

Un procesador **vectorial** convierte un código como este:

```
do i = 0, N-1
      C(i) = A(i) + B(i)
enddo
```

En algo similar a esto:

```
LV V1,A(R1)
LV V2,B(R1)
ADDV V3,V2,V1
SV C(R1),V3
```

Introducción

ESCALAR

```
buc:    FLD F1,A(R1)
        FLD F2,B(R1)
        FADD F3,F2,F1
        FST C(R1),F3
        ADDI R1,R1,#8
        SUBI R2,R2,#1
        BNZ R2,buc
```

VECTORIAL

```
LV V1,A(R1)
LV V2,B(R1)
ADDV V3,V2,V1
SV C(R1),V3
```

¿ D I F E R E N C I A S ?

Introducción

ESCALAR

```
buc:    FLD F1,A(R1)
        FLD F2,B(R1)
        FADD F3,F2,F1
        FST C(R1),F3
        ADDI R1,R1,#8
        SUBI R2,R2,#1
        BNZ R2,buc
```

7 Instrucciones
Bucle que se repite R2 veces

VECTORIAL

```
LV V1,A(R1)
LV V2,B(R1)
ADDV V3,V2,V1
SV C(R1),V3
```

4 Instrucciones
Sin bucle

Las instrucciones vectoriales son más lentas, pero se dedican a realizar las operaciones por lo que hay poca perdida en control, carga y decodificación de instrucción.

Permite una segmentación en operaciones muy elevada

Introducción

BD- Búsqueda y decodificación
AM-Calculo dirección memoria
M-operación en memoria

L-Lectura operando
A-operación
E-Escritura en registros

LV v1,…	BD	L	AM	M	M	M	E		
					M	M	M	E	
						M	M	M	E
							….	….	….

¿ REPLICACIÓN O SEGMENTACIÓN ?

Introducción

BD- Búsqueda y decodificación
AM-Calculo dirección memoria
M-operación en memoria

L-Lectura operando
A-operación
E-Escritura en registros

LV v1,…	BD	L	AM	M	M	M	E		
					M	M	M	E	
						M	M	M	E
							….	….	….

Esta representación segmentada la podemos representar así:

Lv V1,…	BD	L	AM	M	M	M	E	…	E

Introducción

El anterior programa, Un simple programa que suma dos vectores tendría la siguiente forma:

```
LV V1,A(R1)
LV V2,B(R1)
ADDV V3,V2,V1
SV C(R1),V3
```

LV	BD	L	AM	M	M	M	E	E	…	…	E		
LV		BD	L	AM	M	M	M	E	E	…	…	E	
ADDV			BD	**	**	**	**	L	A	A	E	…	E
SV				BD	L	AM	**	**	**	**	L	M	M	M

Si realizamos con exactitud los cálculos de ciclos consumidos tanto en vectorial como en escalar, darían 896 ciclos escalar frente a 142 ciclos en su versión vectorial.

(Para un vector de 128 elemento y un t_i de 14 ciclos)

Introducción

Un **operando vectorial** contiene una secuencia de n elementos, llamados componentes, donde n es la **longitud del vector**.

Cada componente del vector es un escalar de cualquier tipo (entero, punto flotante, etc.).

Las operaciones vectoriales pueden ser de uno de los siguientes 5 tipos:

Operación		*Ejemplo*
Op. vectorial unitaria	$V \rightarrow V$	*Raíz Vectorial*
Op. vectorial binaria	$V \times V \rightarrow V$	*Suma de vectores*
Reducción unitaria	$V \rightarrow K$	*Máximo / Modulo*
Reducción binaria	$V \times V \rightarrow K$	*Producto escalar*
Op. de escalado	$K \times V \rightarrow V$	*Prod. escalar por vector*

Ventajas vectoriales frente a escalares

- En una operación vectorial, cada resultado es independiente, permite efectuar cálculos en procesadores segmentados evitando conflictos.
- Una instrucción vectorial equivale a muchas secuenciales, se reduce el cuello de botella al leer las instrucciones.
- Las instrucciones vectoriales acceden a memoria con un patrón fijo, aumenta la eficiencia del acceso a caché y memorias entrelazadas.
- Evitamos riesgos de control de flujo a evitar bucles.

Arquitectura

Los procesadores vectoriales se basan en la segmentación.

La segmentación de los procesadores vectoriales procesará uno a uno cada componente del vector

Durante la ejecución de una instrucción vectorial:

- No habrá dependencia de datos.
- No habrá dependencia de control

En principio **NO** hay limitación del tamaño del vector

Arquitectura

Propiedades de los procesadores vectoriales.

- **Longitud de palabra**
  - Suele ser mayor que el empleado en el resto de procesadores en todos los niveles (memoria, unidades de ejecución, registros, etc.)
- **Longitud de los vectores**
  - Los vectores se pueden operar con una sola instrucción, así que cuanto mayores sean, mejor
- **Uso de la segmentación**
  - En el procesamiento de instrucciones se usa para ganar ILP
  - En el procesamiento de datos se usa a varios niveles:
    - **Microsegmentación**: Segmentación de las unidades vectoriales
    - **Macrosegmentación**: Encadenamiento de las operaciones vectoriales

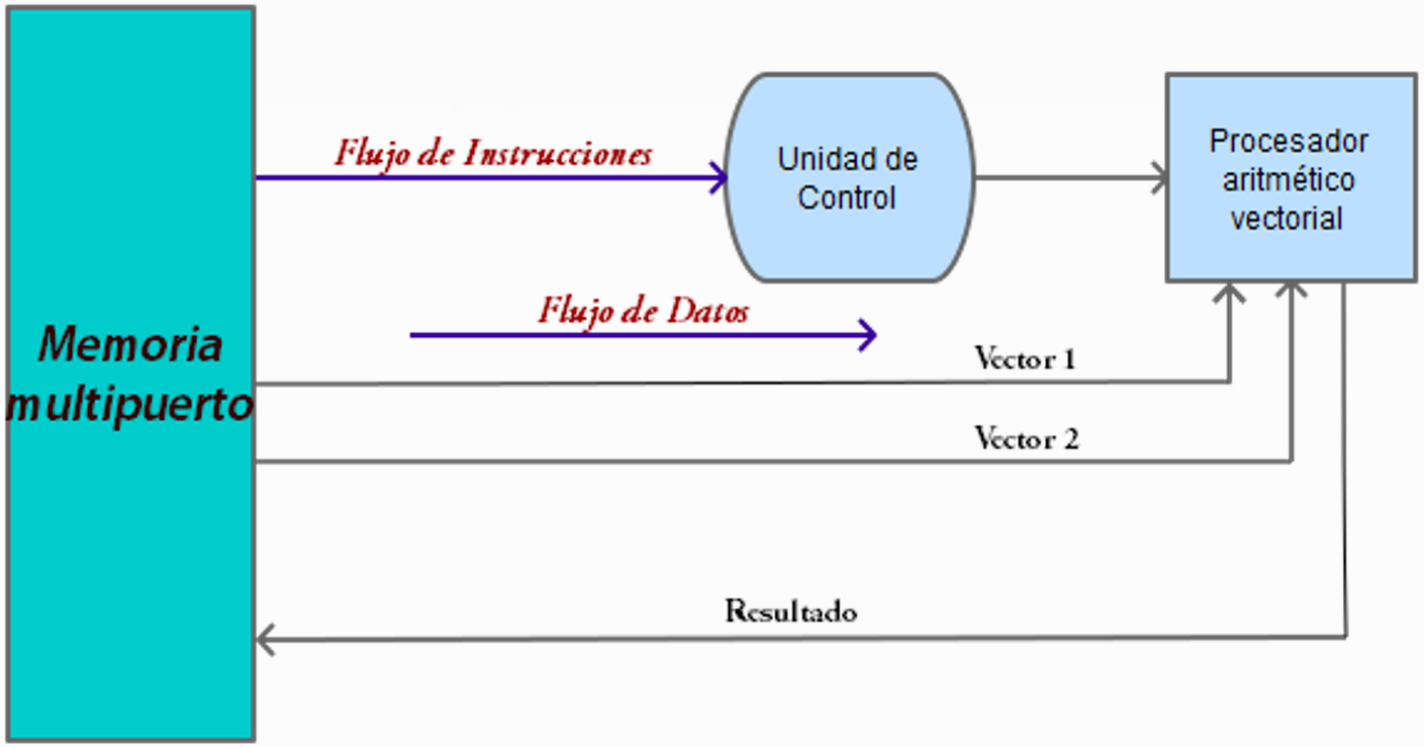
Arquitectura

Propiedades de los procesadores vectoriales.

- **Uso de flujos de datos múltiples** (replicación):
  - Si se dispone de varios cauces vectoriales segmentados que implementan la misma operación vectorial (ej. suma), se podrán realizar varias sumas segmentadas a la vez
  - Se puede usar para lograr:
    - **Paralelismo funcional**:
      Si las operaciones son de instrucciones diferentes
    - **Paralelismo de datos**:
      Si se reparten los elementos de un vector entre los cauces

Procesadores Vectoriales
Memoria - Memoria

Versión simplificada del Procesador Vectorial:



- Procesadores vectoriales memoria-memoria: todas las operaciones vectoriales son memoria-memoria.

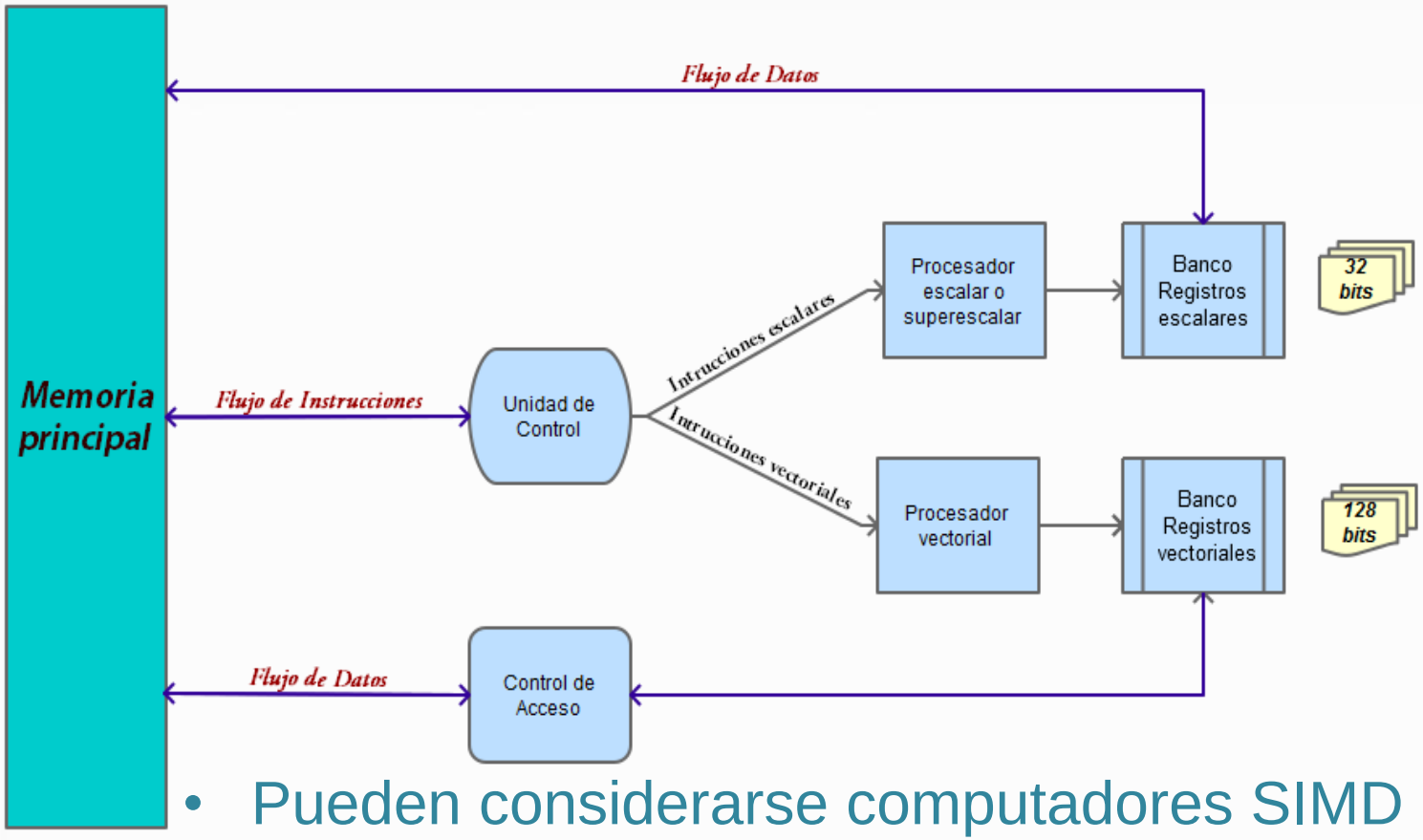
Transformando Procesadores Vectoriales Memoria - Memoria A Registro - Registro

El inconveniente de este tipo de máquina sería el cuello de botella que supondrían los accesos a memoria. Por ello, sobre esta arquitectura simplificada se pueden hacer algunas mejoras:

- Aumentar el ancho de banda de la memoria: esto se consigue entrelazando la memoria
- Añadir una memoria intermedia de mayor velocidad entre la memoria y el procesador. (Para ello usamos bancos de registros)
- Se separa en 2 procesadores, uno escalar y un coprocesadores vectorial.

Procesadores Vectoriales Registro - Registro

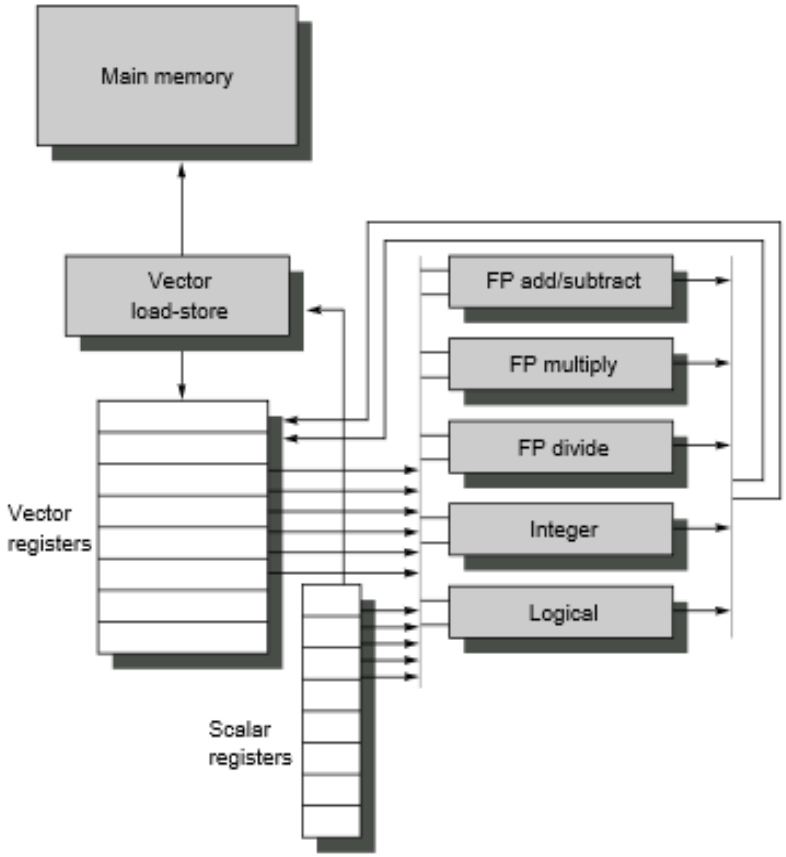
Versión mejorada del Procesador Vectorial simplificado:



- Pueden considerarse computadores SIMD (aunque por sus características no lo son puros)

Como los datos son del mismo flujo algunos autores lo denominan **SIMD-Vectorial**

Procesadores con registros vectoriales



Estructura básica de una arquitectura registro vectorial, DLXV.

Todas las operaciones en procesadores vectoriales **se realizan entre vectores**, excepto la lectura (LOAD) y la escritura (STORE).

Ejemplos: Cray Research (CRAY-1,CRAY-2, X-MP, Y-MP y C-90)

Procesadores con registros vectoriales

- Esta es una arquitectura básica de un procesador vectorial con registros vectoriales, la denominaremos DLXV; donde su parte entera es DLX y su parte vectorial es la extensión vectorial del DLX.

Los principales componentes del DLXV son:

- Registros vectoriales. Posee 8 registros vectoriales de tamaño fijo que permite almacenar un solo vector.
- Unidades vectoriales funcionales. Posee 5 unidades funcionales completamente segmentadas por lo que puede comenzar una nueva instrucción en cada ciclo de reloj.
- Unidad de carga/almacenamiento. Se encarga de manejar la escritura/lectura de la lectura. Ambas operaciones están segmentadas de manera que es posible mover los datos por ciclo de reloj.
- Un conjunto de registros escalares. Posee 32 registros de propósito general y 32 registros de punto flotante.

Procesadores con registros vectoriales

Instruction	Operands	Function
ADDV ADDSV	V1,V2,V3 V1,F0,V2	Add elements of V2 and V3, then put each result in V1. Add F0 to each element of V2, then put each result in V1.
SUBV SUBVS SUBSV	V1,V2,V3 V1,V2,F0 V1,F0,V2	Subtract elements of V3 from V2, then put each result in V1. Subtract F0 from elements of V2, then put each result in V1. Subtract elements of V2 from F0, then put each result in V1.
MULTV MULTSV	V1,V2,V3 V1,F0,V2	Multiply elements of V2 and V3, then put each result in V1. Multiply F0 by each element of V2, then put each result in V1.
DIVV DIVVS DIVSV	V1,V2,V3 V1,V2,F0 V1,F0,V2	Divide elements of V2 by V3, then put each result in V1. Divide elements of V2 by F0, then put each result in V1. Divide F0 by elements of V2, then put each result in V1.
LV	V1,R1	Load vector register V1 from memory starting at address R1.
SV	R1,V1	Store vector register V1 into memory starting at address R1.
LVWS	V1,(R1,R2)	Load V1 from address at R1 with stride in R2, i.e., R1+i × R2.
SVWS	(R1,R2),V1	Store V1 from address at R1 with stride in R2, i.e., R1+i × R2.
LVI	V1,(R1+V2)	Load V1 with vector whose elements are at R1+V2(i), i.e., V2 is an index.
SVI	(R1+V2),V1	Store V1 to vector whose elements are at R1+V2(i), i.e., V2 is an index.
CVI	V1,R1	Create an index vector by storing the values 0, 1 × R1, 2 × R1, ..., 63 × R1 into V1.
S--V S--SV	V1,V2 F0,V1	Compare the elements (EQ, NE, GT, LT, GE, LE) in V1 and V2. If condition is true, put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S--SV performs the same compare but using a scalar value as one operand.
POP	R1,VM	Count the 1s in the vector-mask register and store count in R1.
CVM		Set the vector-mask register to all 1s.
MOVI2S MOVS2I	VLR,R1 R1,VLR	Move contents of R1 to the vector-length register. Move the contents of the vector-length register to R1.
MOVF2S MOVS2F	VM,F0 F0,VM	Move contents of F0 to the vector-mask register. Move contents of vector-mask register to F0.

En las operaciones vectoriales del DLXV se usan los siguientes nemotécnicos, se le añade al final la letra …

- V para indicar que es una operación vectorial.
- SV si la operación se efectúa sobre escalares.

Procesadores con registros vectoriales

Supongamos que tenemos que calcular ***Y*** = ***a*** * ***X*** + ***Y***. Donde ***Y*** e ***X*** son dos vectores almacenados en memoria y ***a*** es un escalar. También supongamos que la longitud del vector es de 64 elementos.

```
      LD      F0,a
      ADDI    R4,Rx,#512      ;ultima dirección a cargar
loop: LD      F2,0(Rx)        ;carga X(i)
      MULTD   F2,F0,F2        ;a x X(i)
      LD      F4,0(Ry)        ;carga Y(i)
      ADDD    F4,F2,F4        ;a x X(i) + Y(i)
      SD      F4,0(Ry)        ;guarda en Y(i)
      ADDI    Rx,Rx,#8        ;incrementa índice de X
      ADDI    Ry,Ry,#8        ;incrementa índice de Y
      SUB     R20,R4,Rx       ;calcula finalización
      BNZ     R20,loop        ;salta si no se ha terminado
```

```
LD      F0,A        ;carga escalar a
LV      V1,Rx       ;carga vector X
MULTSV  V2,F0,V1    ;multiplicación vector-escalar
LV      V3,Ry       ;carga vector Y
ADDV    V4,V2,V3    ;suma
SV      Ry,V4       ;guarda el resultado
```

Las principales diferencias entre los códigos:

1. *El número de instrucciones utilizadas case se reduce a la mitad.*
2. *Reduce la frecuencias de interbloques.*
3. *Reduce el número de instrucciones de ejecución de 600 a 6.*
4. *Elimina la utilización de sentencias loop.*

Problemas y soluciones

Posibles problemas que nos podemos encontrar en operaciones matemáticas sobre vectores en los computadores vectoriales:

- *Accesos a Memoria*
- *Dependencia de datos*
- *Longitud del vector*
- *Separación vectorial*

Accesos a memoria

Si usamos un solo bus, solo una instrucción podría acceder a memoria. Lo que provocaría demasiados tiempos perdidos.

Solución: Usar múltiples buses

El Cargar o salvar un vector a memoria es muy lento.

Solución: Usar memorias entrelazadas.

Aún así genera conflictos.

Solución: Muchos módulos entrelazados

Dependencias de datos

- Al igual que en los procesadores (super)escalares las dependencias de datos juegan un papel importante en el rendimiento de la arquitectura.

- Pese a que no hay dependencia en la operación de un vector si puede haber dependencias en dos instrucciones vectoriales consecutivas.

  Solución. Usar una equivalencia de las técnicas de *forwarding* que usan las maquinas escalares
  Esta técnica aplicada en arquitecturas vectoriales y que veremos más adelante se denomina **encadenamiento (*Chaining*)**.

Longitud del Vector

Los registros vectoriales de los procesador vectoriales poseen una longitud limitada (de 64 elementos para DLXV), este tamaño normalmente no coincidirá con la longitud del vector de un programa real.

- Usaremos el ejemplo:

```
do i=1, n
    Y(i) = a * X(i) + Y(i)
```

Supongamos que los vectores X e Y tienen un número de elementos distinto al tamaño de los registros vectoriales del procesador. ¿Cómo solucionaríamos este problema?

Longitud del Vector

Este problema **solo** se da **en** los procesadores vectoriales con **arquitectura registro a registro**.

- Pese a que conceptualmente la longitud de los vectores pueden ser ilimitada, el banco de registros no lo es.
- La longitud de los vectores rara vez coincide con el tamaño del banco de registros.
- En muchos casos la longitud de los vectores es desconocida hasta el momento de ejecución

Longitud del Vector

Solución:

- Si la longitud del vector **es menor** que el tamaño de los registros vectoriales.

  Insertar en la arquitectura el un registro de longitud del vector (VL o VLR – ***Vector Length Register***).

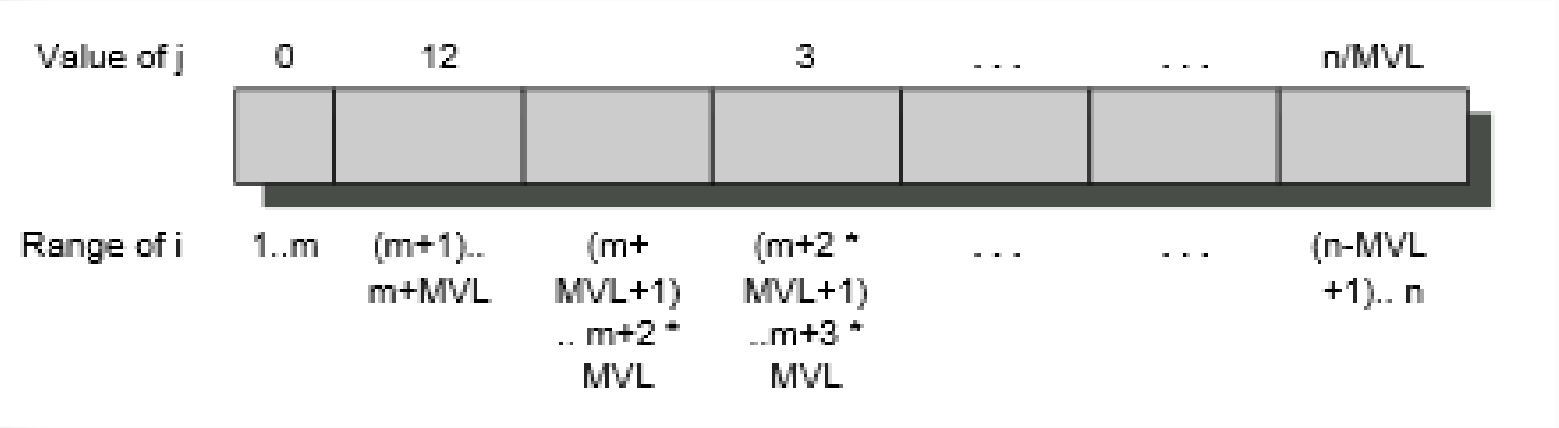
  Este registro contiene la longitud del vector a usar en cualquier operación vectorial

  El valor de este registro siempre ha de ser menor de la longitud de los registros vectoriales (MVL – ***Maximum Vector Length***)

Longitud del Vector

- Si la longitud del vector **es Mayor** que el tamaño de los registros vectoriales usamos la técnica de…
  - ➢ Strip-mining *(Seccionamiento)*

Esta técnica consiste en dividir el vector en segmentos que son procesados independientemente. El tamaño del primer segmento es (n mod MVL) y los demás segmentos son de longitud MVL.



El seccionamiento necesita de operaciones extra para poder realizarse, aunque por lo general en un coste compensado por la eficiencia de la arquitectura registro-registro.

Longitud del Vector

El compilador suele ser el encargado de la realización de esta tarea. (Strip-mining)

```
low=1
VL=(n mod MVL)                  ; determinamos el tamaño del resto
    do j=0, (n/MVL)             ; bucle exterior
        do i=low, low+VL-1      ; ejecutar para la longitud VL
            Y(i) = a * X(i) + Y(i)   ; operación principal
        enddo
        low=low+VL              ; empieza el siguiente vector
        VL=MVL                  ; se actualiza la longitud a la máxima
    enddo
```

En el código anterior, el efecto del bucle exterior es dividir el vector en segmentos de tamaño MLV, excepto el primer segmento que será de tamaño (n mod MLV), tal como se vio en la imagen de la diapositiva anterior.

Separación vectorial

Este problema se da en vectores cuyos elementos no están almacenados de forma adyacente en memoria, por ejemplo:

- Imaginemos que tenemos una matriz bidimensional en memoria.

  Recordemos que de esa matriz podemos obtener los vectores por filas, por columnas, diagonalmente, etc…

- Consideremos una multiplicación matrices vectorizando la multiplicación con fila B con columna C .

```
do 10 i=1, 100
      do 10 j=1, 100
            A(i,j) = 0.0
            do 10 k=1, 100
10                A(i,j) = A(i,j)+B(i,k)*C(k,j)
```

Separación vectorial

- El problema radica en la carga de las vectores no adyacentes de memoria en los registros vectoriales.
- En estos casos se denomina que las componentes del vector posee separación (*stride*).
- Por lo general de los procesadores vectoriales tienen un registro *stride* que gestiona una carga (y salvado) rápido con un *stride* distinto de 1

Separación vectorial

➢ *Para solucionar este problema deberemos:*

- Incorporar una instrucción en el procesador que cargue elementos vectoriales no adyacentes con una separación fija. Si se usa este tipo de instrucciones para cargar los elementos sería necesario conocer la dirección inicial del vector, la **separación(*stride*)** y la longitud del vector.
- En el caso de que el *stride* sea distinto de 1, pueden aparecer problemas con el acceso a la memoria, pudiéndose producir un ***conflicto del banco de memoria*** al intentar acceder a un dato de un vector de un banco de memoria cuando no a acabado de proveer un dato anterior. En ese caso se produciría una detención hasta que se completase la transferencia relentizando el sistema.

$$\frac{m.c.m\ (separación, nº\ de\ bancos)}{separación} < Latencia\ de\ acceso\ a\ memoria$$

Los *conflictos del banco de memoria* no se producirían si el *stride* y el *nº de bancos* fuesen **primos relativos**

Separación vectorial

- Hay arquitecturas que tienen instrucciones especificas para el tratamiento de vectores con *stride* distinto de 1.
- Otras arquitecturas han optado por tener dos instrucciones equivalentes, una con *stride* 1 y otras con *stride* variable, que puede ser pasado por parámetro o por un registro

➢ *Ejemplo:*

Tenemos 16 bancos de memoria con una latencia de 12 ciclos ¿Cuánto tiempo tardará en completarse una lectura de un vector de 64 elementos con un valor de separación de 1?.¿Y con un valor de separación de 32?.

Separación vectorial (Ejemplo)

- *Solución:*

Dado que el número de bancos es mayor que la latencia de lectura, para un valor de salto de 1,
La carga tardará 12+63=75 ciclos de reloj
(1.2 ciclos por elemento)

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Separación vectorial (Ejemplo)

- *Solución:*

  En el segundo supuesto tiene el peor valor de separación (stride), pues es un valor múltiplo del número de bancos de memoria, como es el caso de un valor de salto de 32 con 16 bancos de memoria. Cualquier acceso a la memoria colisiona con el acceso previo. Esto da lugar a una latencia de 12 ciclos por elemento y un tiempo total de 768 ciclos de reloj (12 * 64).

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16
X															
X															
X															
X															
X															
….															
….															

Separación vectorial (Ejemplo)

- *¿Cómo sería con un stride de 3?*
- *¿y de 4?*

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16
X															
X															
X															
X															
X															
….															
….															

Separación vectorial (Ejemplo)

- *Con un stride de 3 sería tan optimo como un stride de 1.*

B1		B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16
X			X			X			X			X			X
		X			X			X			X			X	
	X			X			X			X			X		
X			X			X			X			X			X
		X			….			..	..	.	..	..	..	..	.
..	.	..	.	..	.	..	.	.	.						
….															
.															

Separación vectorial (Ejemplo)

- ***Con un stride de 4***

Al intentar leer el cuarto dato estaríamos en conflicto pues aún no a acabado de proporcionar el primer dato, teniendo que esperar 8 ciclos adicionales por cada dato.

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16
X				X				X				X			
X				X				X				X			
X				X				X				X			
…				…				…				…			
…				…				…				…			

Memoria

¿Como optimizar/acelerar la velocidad de transferencia memoria-procesador?

Entrelazado de memoria.

El entrelazado consiste en distribuir el espacio de memoria del procesador entre distintos módulos de memoria.

Entrelazado de memoria

- Con el entrelazado podemos conseguir aumentar la transferencia de datos entre memoria y procesador (o banco de registros)
- Si no se pudiese disponer de memorias entrelazadas la memoria cache podría ahorrar tiempo
  (las memorias cache pierden eficiencia con las entrelazadas)
- Este tipo de arquitectura suele usar muchos módulos de memoria maximizando el entrelazado
  (hay ordenadores que llegan a 128 módulos)
- Dos tipos de memoria entrelazada
  - Bajo o inferior
  - Alto o superior

Entrelazado de memoria

- **Memoria entrelazada baja o inferior**
  Las posiciones contiguas de memoria se asignan a módulos distintos.

  Esto se hace usando los bits de orden mas bajo para identificar el modulo de memoria

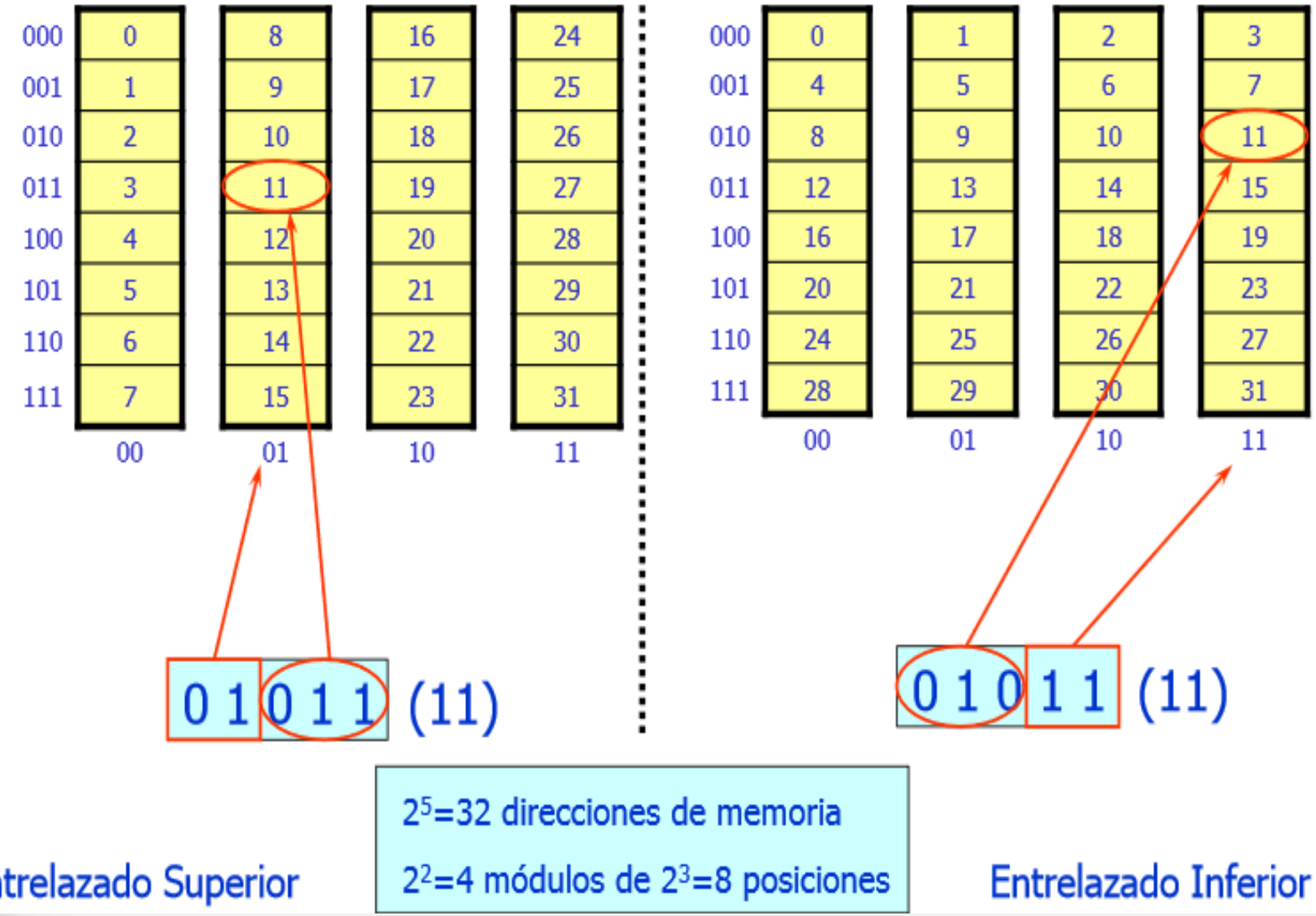
  Se usan los bits de orden superior para identificar la palabra dentro del modulo

- **Memoria entrelazada alta o superior**
  Las posiciones contiguas de memoria tienen a asignarse en el mismo modulo

  Esto se hace usando los bits de orden mas alto para identificar el modulo de memoria

  Se usan los bits de orden inferior para identificar la palabra dentro del modulo

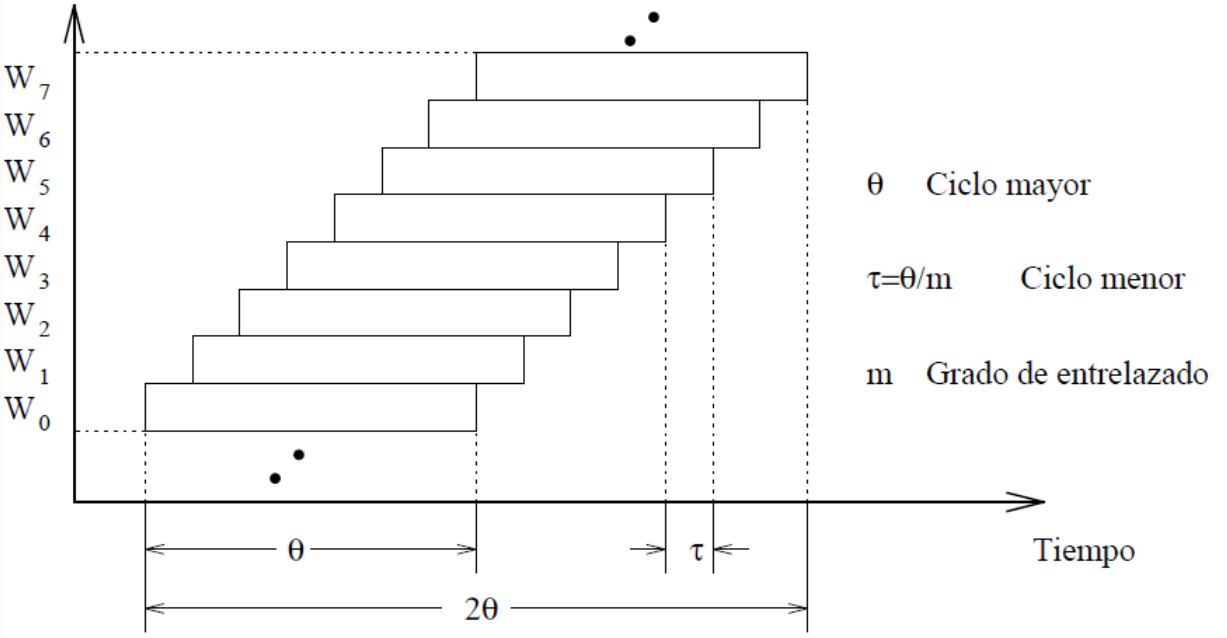
Entrelazado de memoria



Acceso Concurrente a Memoria

También denominado acceso tipo C.

- Los accesos a los m módulos de una memoria se pueden solapar de forma segmentada.

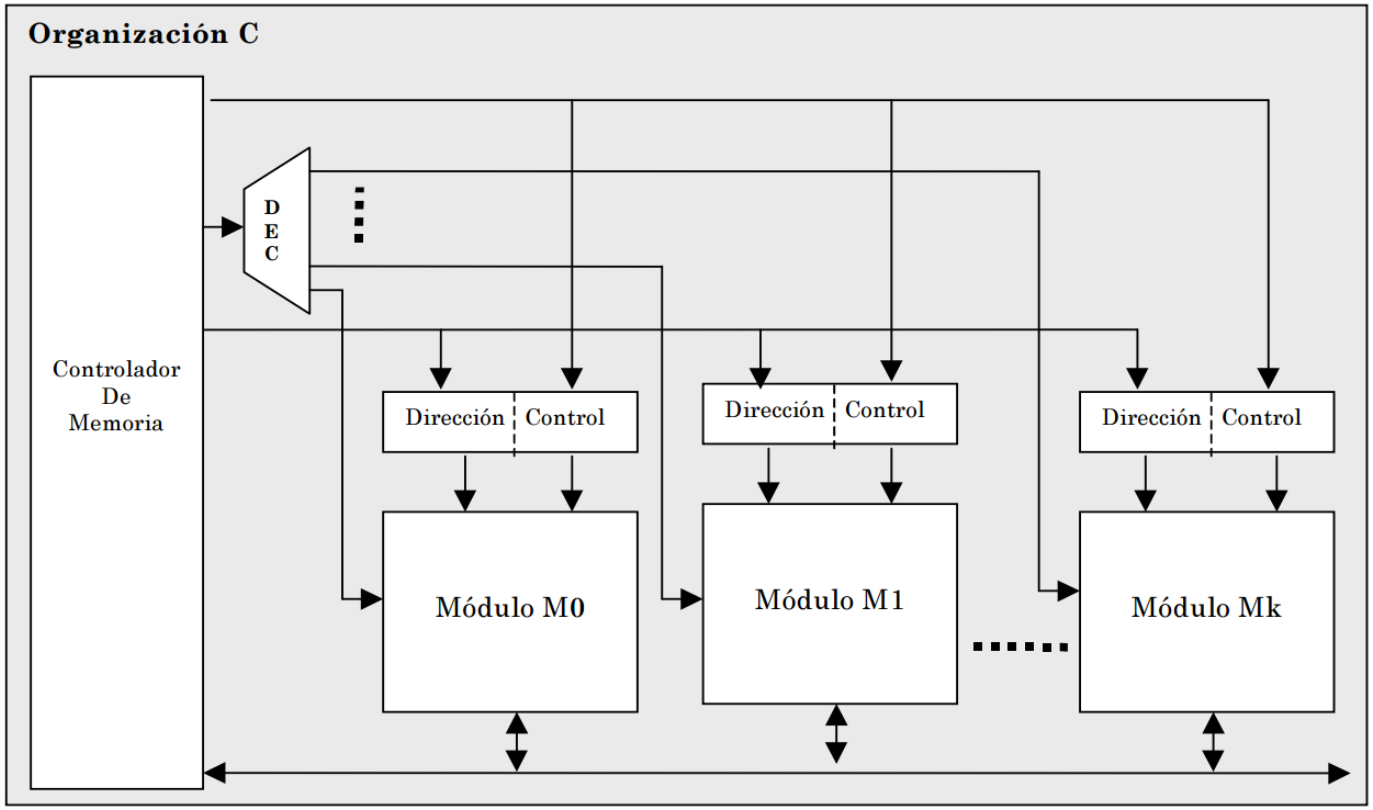


En este ejemplo leemos 8 palabras y el tiempo total de lectura sería tiempo de lectura de una palabra + (numero de palabras-1) * ciclo menor.

Acceso Concurrente a Memoria

También denominado acceso tipo C.

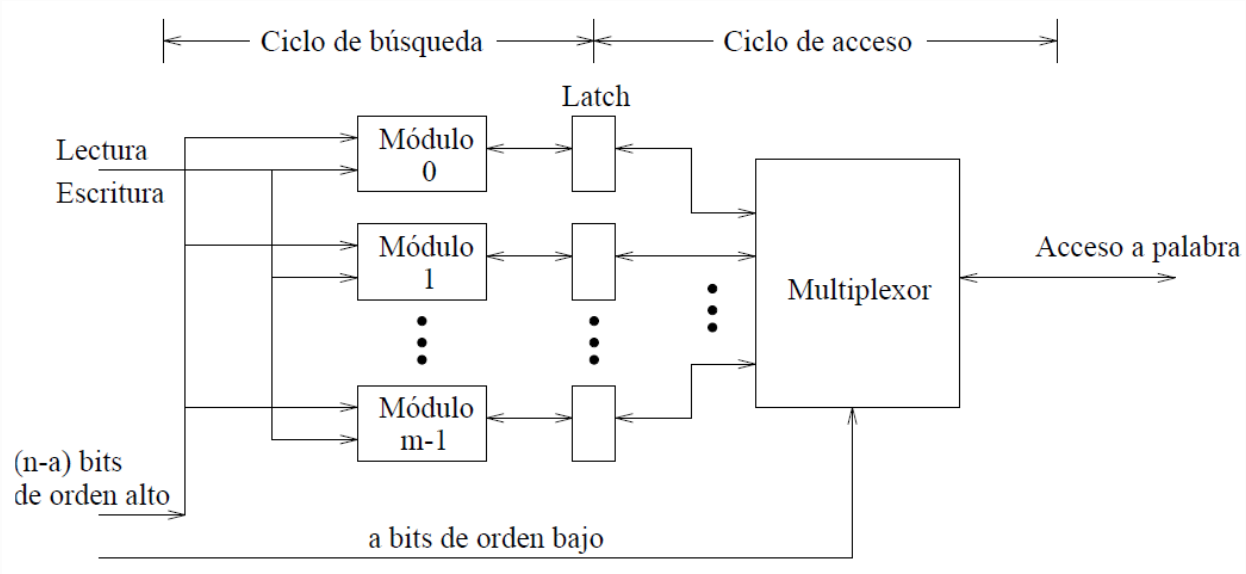
- Los accesos a los m módulos de una memoria se pueden solapar de forma segmentada.



Acceso Simultaneo a Memoria

También denominado acceso tipo S.

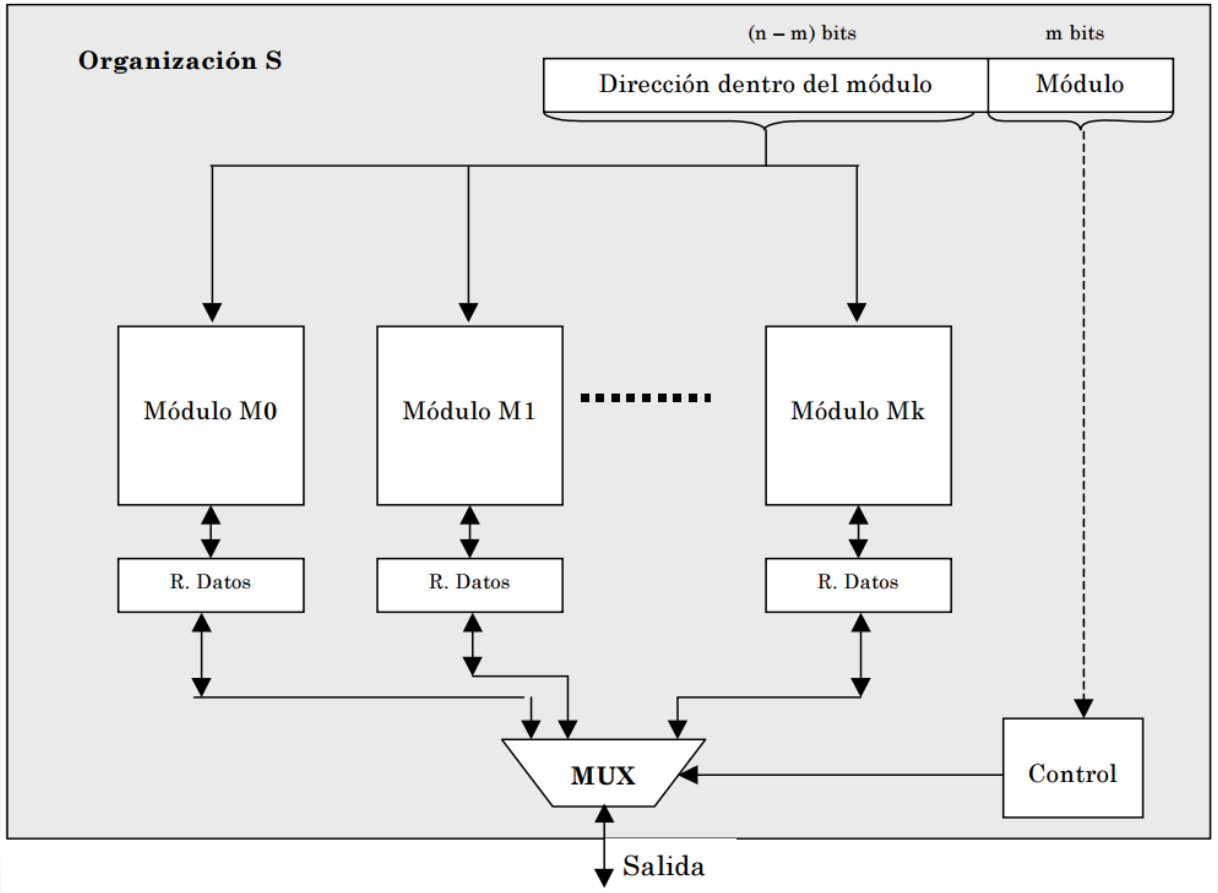
- La memoria entrelazada de orden bajo puede ser dispuesta de manera que permita accesos simultáneos



Acceso Simultaneo a Memoria

También denominado acceso tipo S.

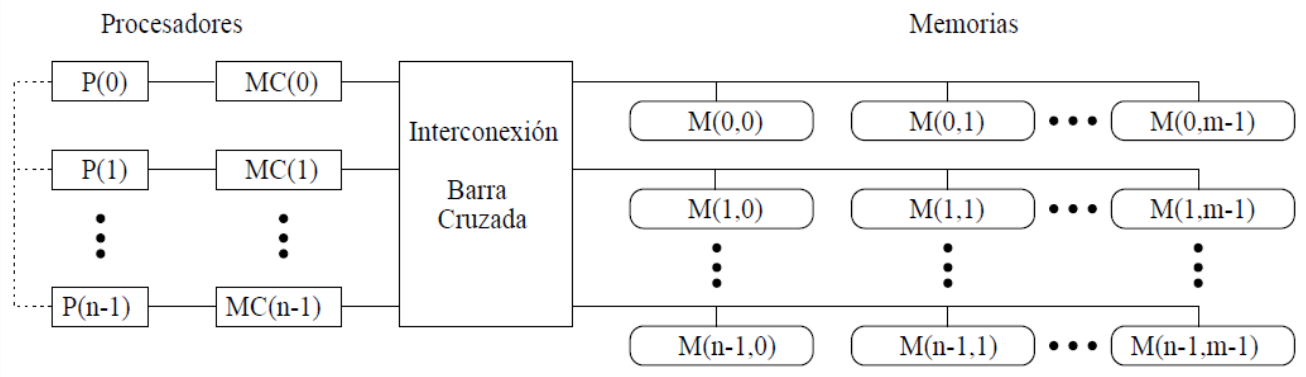
- La memoria entrelazada de orden bajo puede ser dispuesta de manera que permita accesos simultáneos



Memoria de Acceso C/S

Mezcla los dos tipos de accesos.

Indicada para multiprocesadores vectoriales



Se utilizan *n* buses de acceso junto a *m* módulos de memoria por bus. Además van acompañadas de memorias caches por procesador.

Los ***m*** módulos por bus son entrelazados por m vías (acceso C)
Los ***n*** buses operan en paralelo para permitir los accesos S.

Idóneamente en cada ciclo se obtendrían m*n palabras.

Mejora del Rendimiento

Estudiaremos tres técnicas con las que podremos aumentar el rendimiento:

- Encadenamiento
- Ejecución condicional
- Matrices dispersas

Encadenamiento

- Hasta el momento, si teníamos que realizar varias operaciones dependientes, estas se realizaban de manera secuencial, es decir, una empezaría cuando acabara la otra.

Supongamos que tenemos las siguientes instrucciones dependientes:

```
MULTV   v1, v2, v3
ADDV    v4, v1, v5
```

Para poder realizar la suma debemos esperar a que acabe la operación de multiplicación de vectores.

Podríamos asociarlo al RAW en los procesadores escalares

Encadenamiento

- El **encadenamiento** permite que las operaciones vectoriales empiecen tan pronto como los elementos individuales de sus vectores estén disponibles.

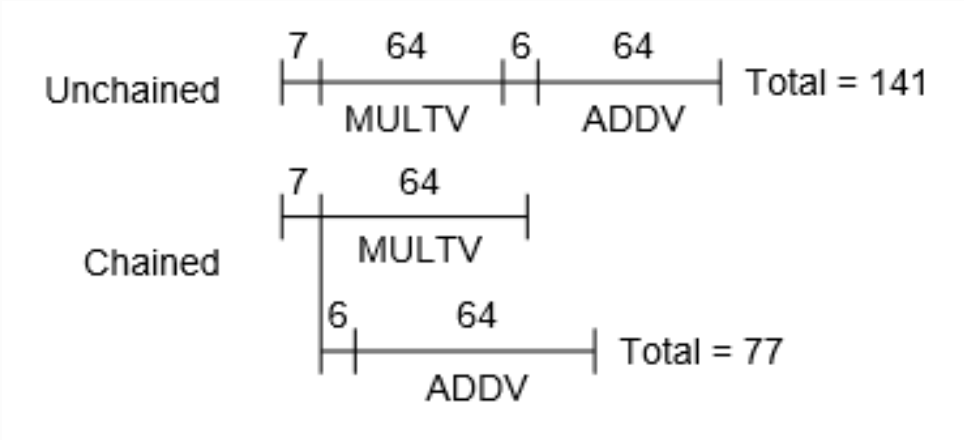
Si las unidades están completamente segmentadas, basta retrasar el comienzo de la siguiente instrucción durante el tiempo de arranque de la primera unidad.

- El tiempo total de ejecución para la secuencia anterior sería:

Longitud del vector + Tiempo de arranque suma + Tiempo de arranque multiplicación

Encadenamiento

- Supongamos que los vectores anteriores tenían una longitud de 64 elementos. Comparativa del número total de ciclos de reloj que tarda la instrucciones en versión encadenada y sin encadenar.



Encadenamiento

- El encadenamiento puede estar limitado al hacerlo con dos instrucciones. Por ejemplo C=A+B

Lv v1,a	**BD**	**L**	**AM**	**M**	**M**	**M**	**E**	**E**	**…**	**…**	*n*	**…**	**…**	**…**	**...**		
Lv v2,b		**BD**	**L**	**AM**	**M**	**M**	**M**	**E**	**E**	**…**	**…**	*n*	**…**	**…**	**…**	**...**	
Addv v3, v1, v2			**BD**	**-**	**-**	**-**	**-**	**L**	**A**	**A**	**E**	**E**	**…**	**…**	*n*	**…**	**…**
Sv c,v3				**BD**	**L**	**AM**	**-**	**-**	**-**	**-**	**L**	**M**	**M**	**M**	**E**	**E**	**…**	**…**
Ciclo	1	2	3	4	5	6	7	8	9	0	11	12	13	14	15	16	…	…

Conflicto en el banco de registros pues se pierde el 7 de v1 al grabar en el 8 del v2.

En el ciclo 7 no se está preparado porque no se tiene el primer elemento de v2, pero en el ciclo 8 el elemento de v1 se pierde ¿que se hace?

La técnica de ***flexible chaining*** permite la lectura y escritura simultanea en el mismo registro

Ejecución Condicional

- Debido a la ley de Amdahl el aumento de velocidad en estos programas es muy limitado. No se puede conseguir un nivel de vectorización alto debido al uso de sentencias condicionales (if) dentro de bucles.
- Los programas que contienen sentencias if en los bucles no pueden ejecutarse en modo vectorial usando las técnicas anteriores ya que las sentencias if introducen dependencias de control en el bucle.

```
do 100 i=1, 64
      if (A(i).ne.0) then
            A(i)=A(i)-B(i)
      endif
continue
```

Ejecución Condicional

- Para solucionar el problema que presentan las sentencias if en la vectorización de la operación, se emplea una **máscara** sobre el vector.
- El control de máscara vectorial es un vector booleano de longitud MVL. Cuando el registro de máscara este activo, la operación vectorial se realizara únicamente sobre aquellos elementos cuya entrada del registro de máscara este a 1.
- Para gestionar óptimamente esta mascar los procesadores suelen incluir un registro especial llamado ***VM – Vector Mask***. Este vector contiene valores booleanos que indicaran si se ejecuta o no la instrucción sobre las componentes del vector.
  - Normalmente todas las operaciones vectoriales tienen en cuenta este vector de mascara

Ejecución Condicional

Los Procesadores Vectoriales disponen de instrucciones especificas para gestionar rápidamente el vector de mascara.

SxxV V1,V2	Compara dos vectores dejando los resultados en el VM. La comparación xx, puede ser EQ, NE, GR…
SxxVS V1,F1	Igual pero comparando con escalares
CVM	Inicializa el VM
POP R1,VM	Cuenta el numero de bits activos en el VM

```
LV      V1, Ra      ; carga el vector A en V1
LV      V2, Rb      ; carga el vector B en V2
LD      F0, #0      ; F0 = 0
SNESV   F0, V1      ; pone VM a 1 si V1(i)!=F0
SUBV    V1, V1, V2  ; resta utilizando la mascara
CVM                 ; pone el vector de mascara a 1
SV      Ra, V1      ; almacena el resultado en A
```

Ejecución Condicional

- Desventajas de utilizar una máscara
  - Siempre se consume el mismo tiempo ya que tendrá que comprobar la condición para todos los elementos.
  - Algunos procesadores con la máscara solo deshabilitan el almacenamiento del resultado. Esto puede provocar si estamos realizando una división, que realicemos la división por cero y generemos una excepción.

```
do 100 i=1, 64
    if (A(i).ne.0) then
        C(i) = B(i) / A(i)
    endif
continue
```

Matrices Dispersas

- Las matrices dispersas son matrices con una gran cantidad de elementos, siendo la mayoría de los elementos ceros.
- En estos casos, es más rápido trabajar directamente con los componentes validos o con datos, descargando al procesador de trabajar sobre los elementos nulos.
- Una de las formas habituales de trabajar es acceder indirectamente, es decir tener un vector con los índices de los elementos validos, así estas matrices están almacenadas de forma compacta y son accedidas indirectamente.

Matrices Dispersas

- El vector de índices (en algunas arquitecturas se implementan ***registros de índices***) guardan las posiciones concretas de los elementos que queremos acceder.
- De esta forma se consigue evitar:
  - Tratar datos innecesarios.
  - Múltiples operaciones al realizar el *Strip-mining* innecesario

Matrices Dispersas

Una forma típica de trabajar con matrices dispersas:

```
do i = 1, n
      A( K(i) )  =  A( K(i) ) + C( M(i) )
```

- En A y C tenemos las matrices dispersas
- En K y M tenemos los índices de los valores no vacíos

Matrices Dispersas

Agrupamiento y dispersión

Una operación vectorial de este estilo se suele dividir en tres fases:

1. **Fase de agrupamiento** (***gather***): se utiliza el registro de índices para leer los elementos que nos interesan (base + desplazamient), y se cargan en un registro vectorial.
2. **Fase de ejecución**: se ejecuta la operación indicada.
3. **Fase de dispersión** (***scatter***): se llevan los resultados de la operación vectorial a memoria, a las posiciones correspondientes, utilizando nuevamente el registro de índices

Matrices Dispersas

Los Procesadores Vectoriales disponen de instrucciones especificas para gestionar rápidamente estas tareas

LVI	Cargar vector indexado
SVI	Almacenar vector indexado
CVI	Crea un vector índice

Ejemplo:

```
LV      Vk, Rk          ; carga el vector K (índices)
LVI     Va, (Ra+Vk)     ; carga el vector A (sobre índices)
LV      Vm, Rm          ; carga el vector M (índices)
LVI     Vc, (Rc+Vm)     ; carga el vector C (sobre índices)
ADDV    Va, Va, Vc      ; suma A y C
SVI     (Ra+Vk), Va     ; Almacena en A la suma usando índices
```

Matrices Dispersas

- Si la propiedad de dispersión de un matriz cambia es necesario calcular un nuevo índice.
- La instrucción CVI (Create Vector Index) crea un vector índice dado un valor de salto (m).
- Algunos procesadores poseen una instrucción para crear un vector índice cuyas entradas son las posiciones a 1 en el registro máscara.

Matrices Dispersas

Ejemplo: ¿ Que hace este programa?

```
LV      V1, Ra
LD      F0, #0
SNESV   F0, V1
CVI     V2, #8
POP     R1, VM
MOVI2S  VLR, R1      ; carga el registro de longitud vectorial
CVM
LVI     V3, (Ra + V2)
LVI     V4, (Rb + V2)
SUBV    V3, V3, V4
SVI     (Ra + V2), V3
```

Matrices Dispersas

Ejemplo: ¿ Que hace este programa?

```
LV      V1, Ra          ; Carga vector A en V1
LD      F0, #0          ; Carga en F0 el valor 0
SNESV   F0, V1          ; pone el VMask a 1 en elms de  V1 != 0
CVI     V2, #8          ; Genera índices en V2
POP     R1, VM          ; Calcula el nº de unos en VM
MOVI2S  VLR, R1         ; Carga el registro de longitud vectorial
CVM                     ; Pone a 1 los elementos de la máscara
LVI     V3, (Ra + V2)   ; Carga los elementos de A distintos de 0
LVI     V4, (Rb + V2)   ; Carga los elementos correspondientes de B
SUBV    V3, V3, V4      ; Hace la Resta
SVI     (Ra + V2), V3   ; Almacena A
```

¿algún comentario sobre la operación?

Medición del Rendimiento

¿Tiempo de ejecución de una operación de un vector?

Procesador escalar

$$T_E = t_e * N$$

t_e = Tiempo de procesamiento de un componente en una pasada del bucle.

Procesador vectorial

$$T_V = t_i + t_v * N$$

t_i = Tiempo de inicialización del bucle provocado por el seccionamiento.

t_v = Tiempo de procesamiento de un elemento del vector:

(Normalmente en ciclos o nano segundos)

Medición del Rendimiento

La velocidad de calculo o ***rendimiento*** de los procesadores vectoriales se suele dar en

MFLOPS (**M**ega **FL**oat **OP**eration per **S**econd)

R_N es la velocidad en MFLOPS sobre un vector de longitud N

$$\boldsymbol{R_N = \frac{N}{T_V} = \frac{N}{t_i + t_v * N}}$$

(Para esta formula los tiempos han de estar en segundos, no en ciclos)

Medición del Rendimiento

$\boldsymbol{R_\infty}$: Valor máximo de R cuando N tiende a Infinito

$\textbf{Eficiencia} = \boldsymbol{R/R_\infty}$

Como R_∞ es imposible de obtener se suele usar $N_{1/2}$

$\boldsymbol{N_{1/2}}$: Tamaño mínimo de los vectores que permite alcanzar al menos la MITAD de la velocidad máxima teorica (R_∞)

$\boldsymbol{N_{1/2}} = \frac{\boldsymbol{t_i}}{\boldsymbol{t_v}}$ (entero superior)

$\boldsymbol{N_v}$: Longitud necesaria para hacer el modo vectorial más eficiente que el escalar

$$\boldsymbol{N_v} = \frac{\boldsymbol{t_i}}{\boldsymbol{t_e - t_v}}$$

Medición del Rendimiento

¿ y si N > MVL ? Strip-mining

$$T_v = t_b + \left[\frac{N}{MVL}\right](t_i + t_{buc}) + t_p N$$

$\left[\frac{N}{MVL}\right]$ Número de iteraciones.

t_{buc} Tiempo necesario para el control del bucle.

$(t_i + t_{buc})$ Tiempo de inicialización provocado por el seccionamiento.

t_b Tiempo de arranque del programa, carga de vectores, reg., etc…

t_p Tiempo necesario para procesar cada componente.