



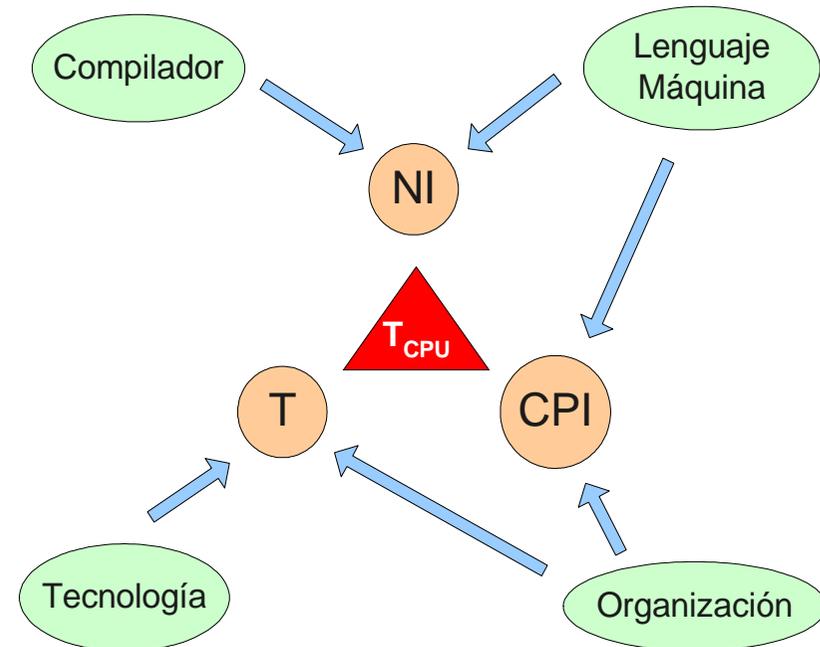
Mejoras en el Juego de Instrucciones

- Procesadores CISC y RISC
- Procesadores SIMD
 - ✓ Procesadores Matriciales
 - ✓ Procesadores Vectoriales

Influencia del Juego de Instrucciones

$$\text{Tiempo}_{\text{CPU}} = \text{NI} \times \text{CPI} \times T$$

- ✓ El compromiso entre NI, CPI y T ha guiado el desarrollo de las nuevas arquitecturas



CPU	NI _{rel}	CPI	1/T	1/T _{CPU}
i386	1.0	4.5	25	5.5
R/3000	1.2	1.25	25	16.6





Características de CISC y RISC

❑ CISC

- ✓ Instrucciones con alto contenido semántico (complejas)
- ✓ Modos de dirección numerosos y complejos
- ✓ Unidades de control microprogramadas



❑ RISC

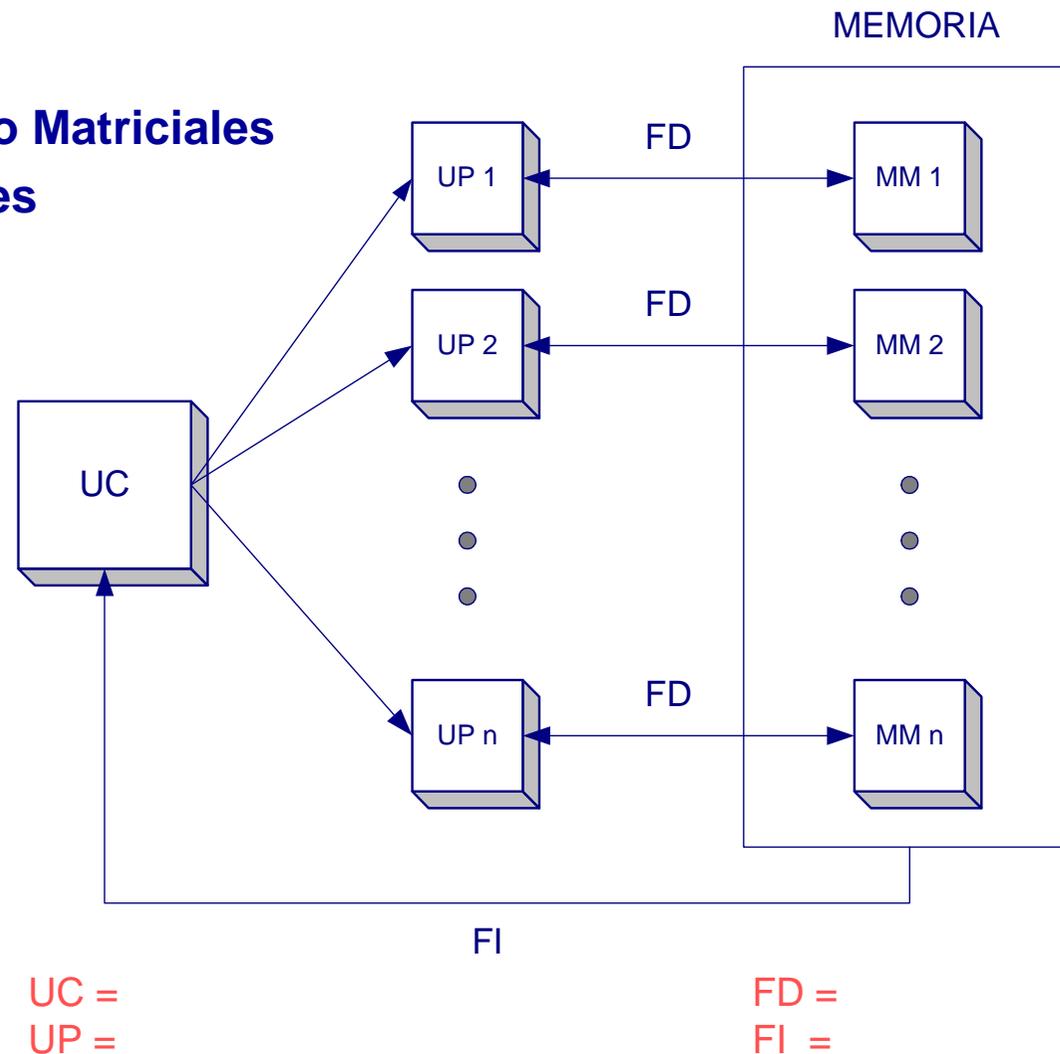
- ✓ Instrucciones con bajo contenido semántico (sencillas)
- ✓ Acceso masivo a registros → Amplios juegos de registros
- ✓ Modos de dirección escasos y sencillos
- ✓ Unidades de control cableadas
- ✓ Fácil adaptación a los lenguajes de alto nivel
- ✓ Sencilla y eficiente segmentación





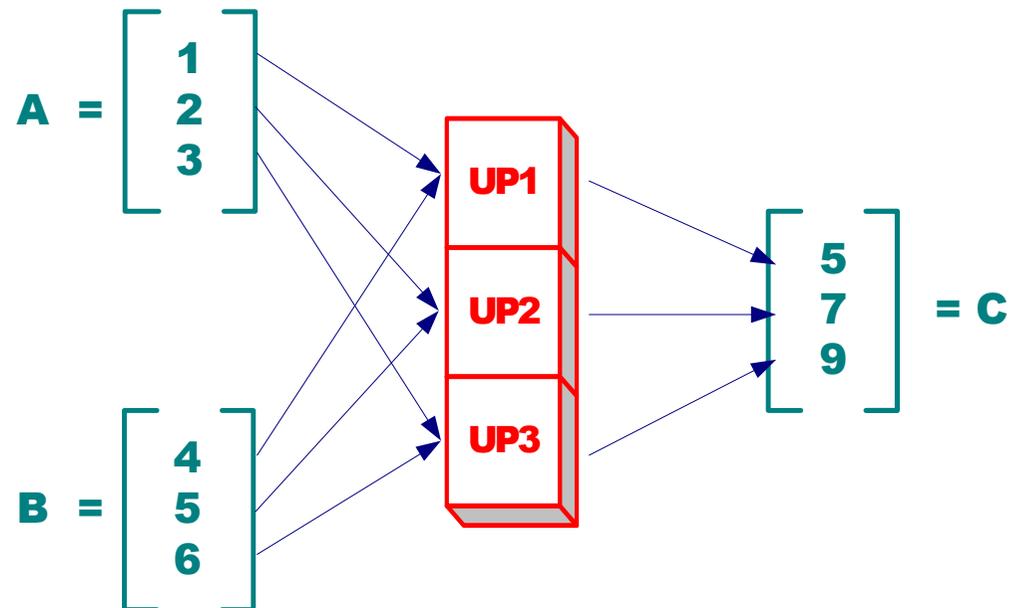
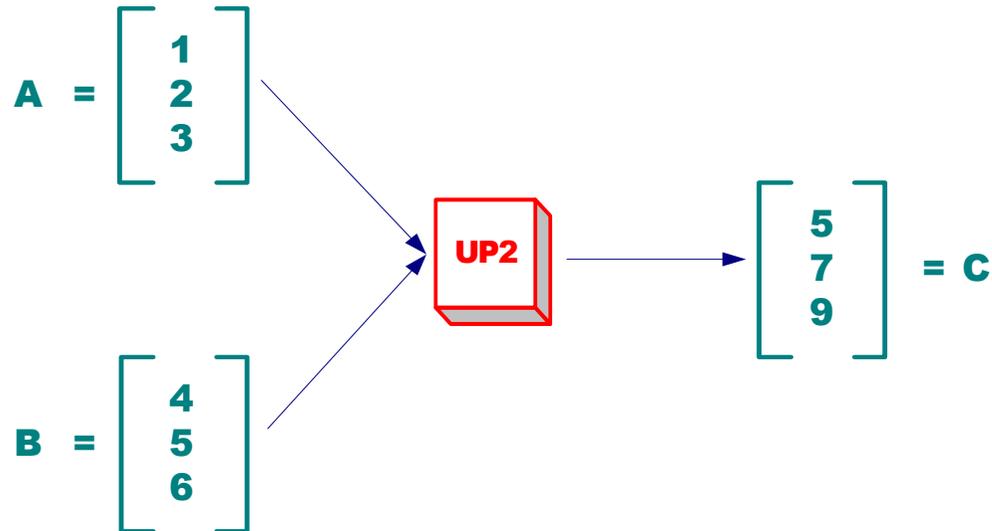
Procesadores SIMD

- ✓ **SIMD =**
- ✓ **Procesadores en Array o Matriciales**
- ✓ **Procesadores Vectoriales**





Procesadores Matriciales

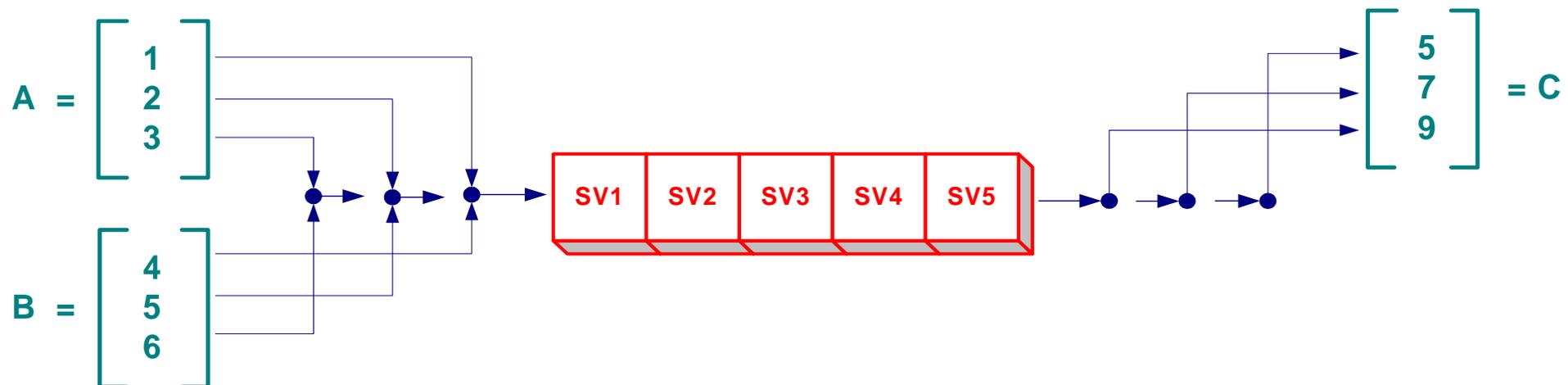




Procesadores Vectoriales

❑ Incorporan un juego de instrucciones vectorial

- ✓ 1 instrucción vectorial con vectores de n elementos sustituye a n instrucciones escalares
- ✓ las operaciones sobre los elementos del vector se realizan de forma solapada sobre cauces vectoriales segmentados





Ventajas de los Procesadores Vectoriales

- ❑ Disminución drástica de coste respecto a los Matriciales
 - ✓ segmentación frente a replicación

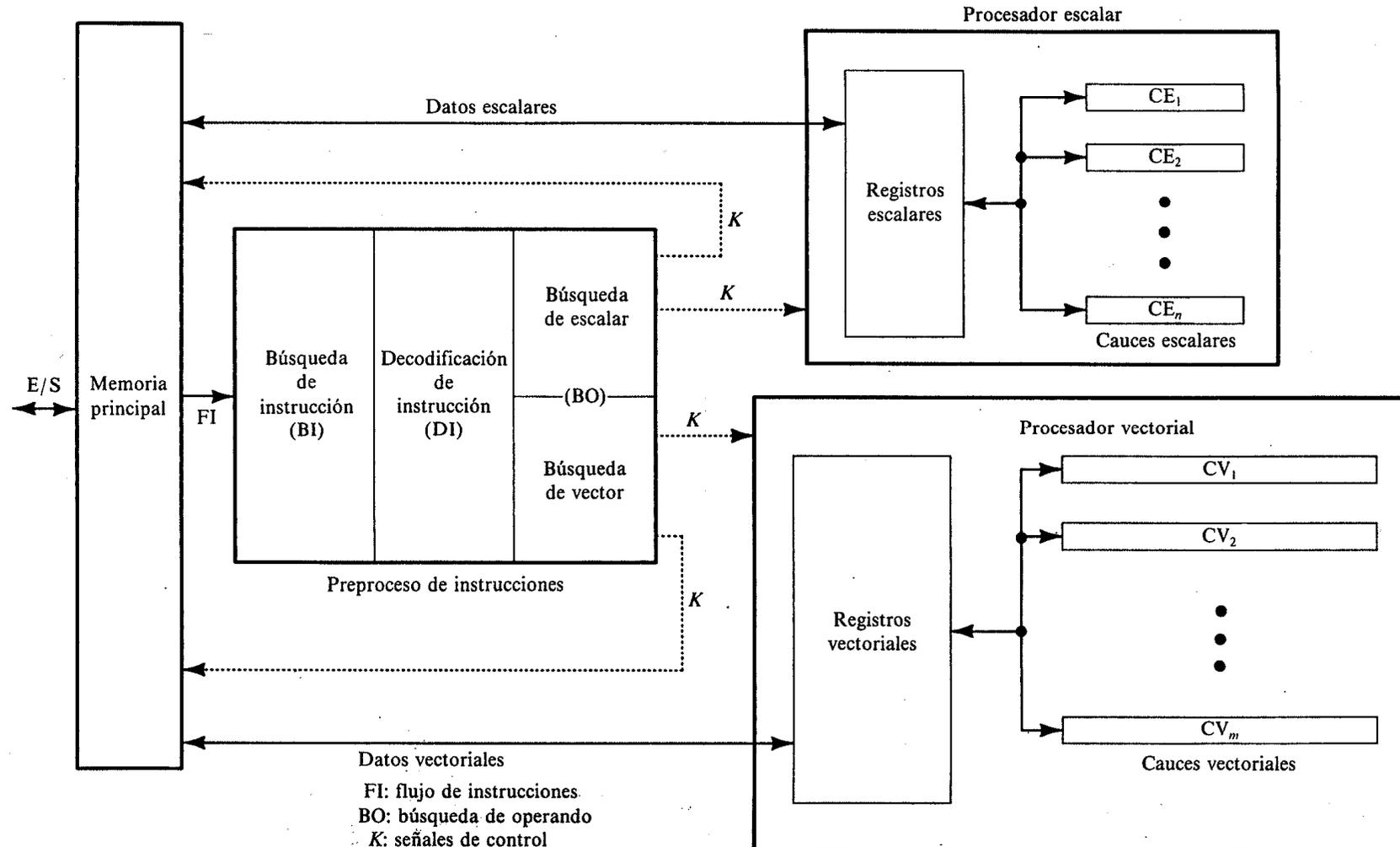
- ❑ Calculo independiente de cada elemento del resultado
 - ✓ se puede trabajar con mayores profundidades de segmentación sin problemas de dependencias

- ❑ Cada instrucción vectorial realiza gran cantidad de trabajo
 - ✓ disminuye el ancho de banda de instrucciones demandado
 - ✓ disminuye el tiempo de decodificación

- ❑ Cada instrucción equivale a un bucle escalar completo
 - ✓ disminuyen los riesgos de control



Arquitectura Vectorial Básica





Ejemplo de Programación Vectorial

$Y[i] = a * X[i] + Y[i]$ (Bucle DAXPY)

(Rx) = Dirección de comienzo del vector X

(Ry) = Dirección de comienzo del vector Y

Tamaño del vector =

Código para máquina vectorial:

LD	F0, a	; carga del escalar a
LV	V1, Rx	; carga del vector X
MULTSV	V2, F0, V1	; producto vector-escalar
LV	V3, Ry	; carga del vector Y
ADDV	V4, V2, V3	; suma vectorial
SV	Ry, V4	; almacenamiento vector Y

Código para máquina NO vectorial:

```

LD      F0, a           ; carga del escalar a
ADDI    R4, Rx, #512    ; última dirección a cargar

loop:
LD      F2, 0(Rx)       ; carga de X(i)
MULTD  F2, F0, F2      ; a*X(i)
LD      F4, 0(Ry)       ; carga de Y(i)
ADD    F4, F2, F4      ; a*X(i) + Y(i)
SD      0(Ry), F4       ; almacenamiento en Y(i)
ADDI    Rx, Rx, #8      ; avance de posición en X
ADDI    Ry, Ry, #8      ; avance de posición en Y
SUB     R20, R4, Rx     ; chequeo de fin de bucle
BNZ    R20, loop

```

NI =
 NI (control) =
 Riesgos LDE =

NI =
 NI (control) =
 Riesgos LDE =





Ejemplos de Arquitecturas Vectoriales Actuales

- ❑ Soporte añadido a procesadores de propósito general:
 - Intel Pentium **MMX** (*MultiMedia eXtensions*)
 - AMD **3DNow**
 - Intel Pentium III **SSE** (*Streaming SIMD Extensions*)
 - Intel Pentium IV **SSE2 / SSE3**

- ❑ Implican juegos de instrucciones adicionales que potencian sobre todo el rendimiento de aplicaciones multimedia:
 - ✓
 - ✓