CE4302 – Arquitectura de Computadores II

Arquitectura vectorial

PROFESOR: ING. LUIS BARBOZA ARTAVIA

Agenda

- Introducción.
- VMIPS.
- Tiempos de ejecución.
- Carriles.
- Otros aspectos.

Variaciones de SIMD

- **Arquitecturas vectoriales.**
  - Ejecución por *pipeline* de muchas operaciones de datos.
- **Extensión SIMD.**
  - Habilitar la posibilidad de brindar operaciones de datos en paralelo.
- **Unidades de procesamiento gráfico.**
  - Ofrecer mayor rendimiento que en los multicore tradicionales.

SIMD

- Significa *Single Instruction Multiple Data.*
- Incluye procesadores de arreglo y vectoriales.
- **Procesadores de arreglo:** Instrucciones operan en múltiples elementos al mismo tiempo.
  - Se conocen como *massively parallel processor*.
- **Procesadores vectoriales:** Instrucciones operan en múltiples elementos en tiempos consecutivos.

SIMD

LD VR ← A[3:0]

ADD VR ← VR,1

MUL VR ← VR,2

ST A[3:0] ← VR

SIMD

Procesador de arreglo

LD0	LD1	LD2	LD3
AD0	AD1	AD2	AD3
MU0	MU1	MU2	MU3
ST0	ST1	ST2	ST3

Procesador vectorial

LD0			
LD1	AD0		
LD2	AD1	MU1	
LD3	AD2	MU2	ST0
	AD3	MU3	ST1
		MU4	ST2
			ST3

Introducción

En enfoque de procesadores vectoriales es el más antiguo (1960) de los tipos de paralelismo a nivel de datos:

- **CDC STAR**
- **Máquinas Cray**
- **Unidades vectoriales en IBM3033**

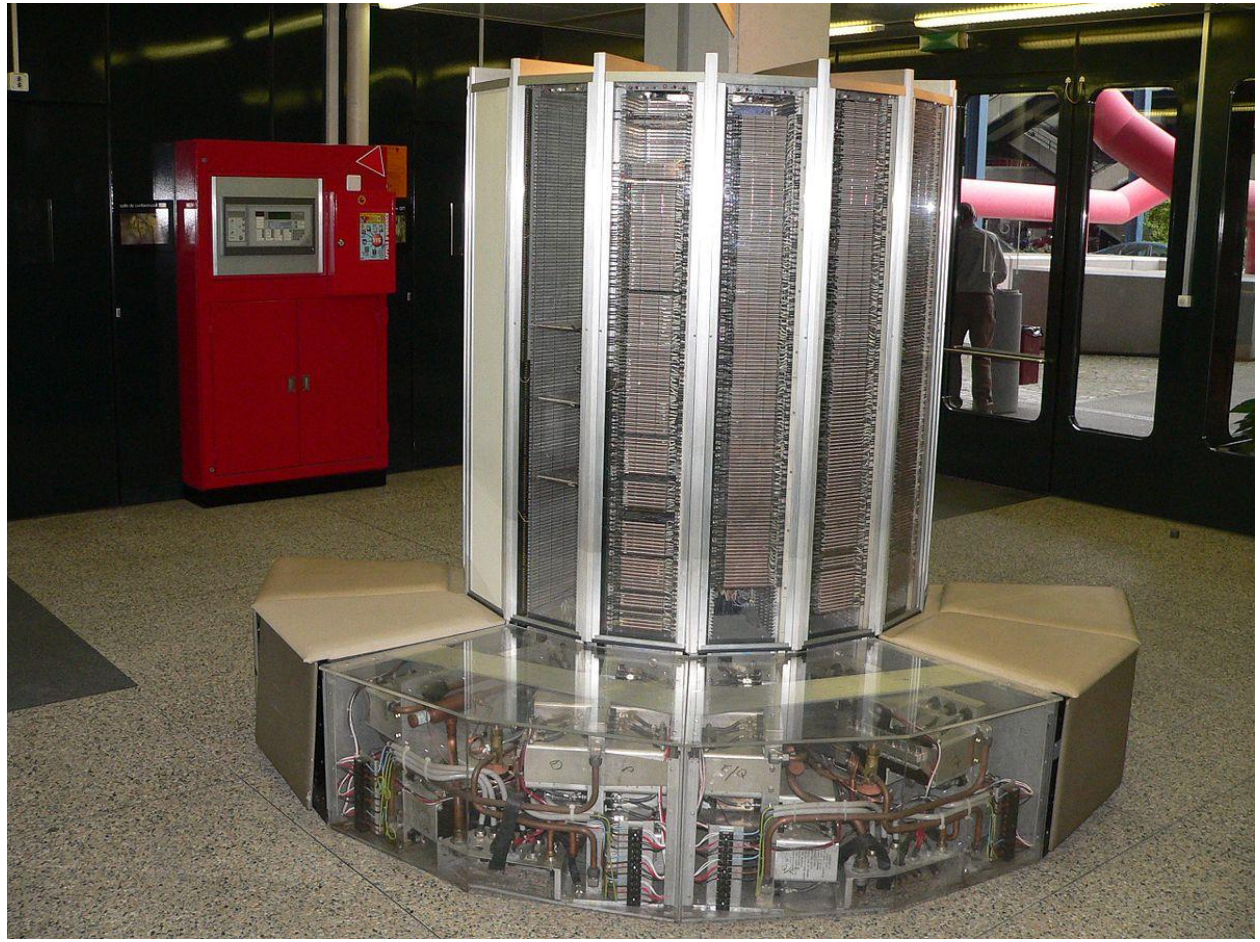
Características

- Los elementos dispersos en memoria se colocan en archivos de registro secuenciales.
- Las operaciones se ejecutan sobre estos registros.
- Los elementos se vuelven a dispersar de vuelta a memoria.
- Una operación vectorial equivale a múltiples operaciones registro-registro de datos independientes.

Características

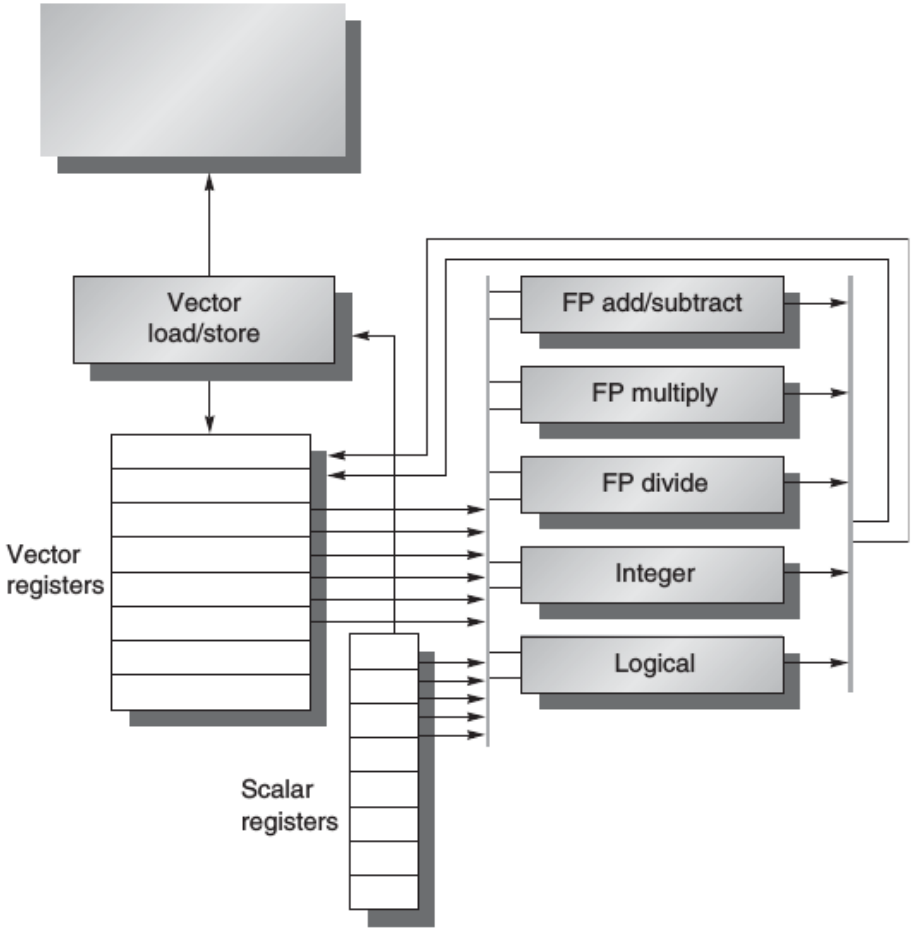
- Los registros actúan como buffers controlados por el compilador (cargados y escritos de manera secuencial).
- Carga y almacenamiento de vectores utilizan pipeline. Esta operación sólo se realiza una vez y no múltiples.

Cray-1



- Desarrollado por Cray Research Inc.
- Procesador vectorial.
- 80 MFLOPS.
  - i9 ejecuta 1.3TFLOPS.

VMIPS



- Unidades de carga y escritura.
- Unidades funcionales.
- Registros vectoriales
- Registros escalares.

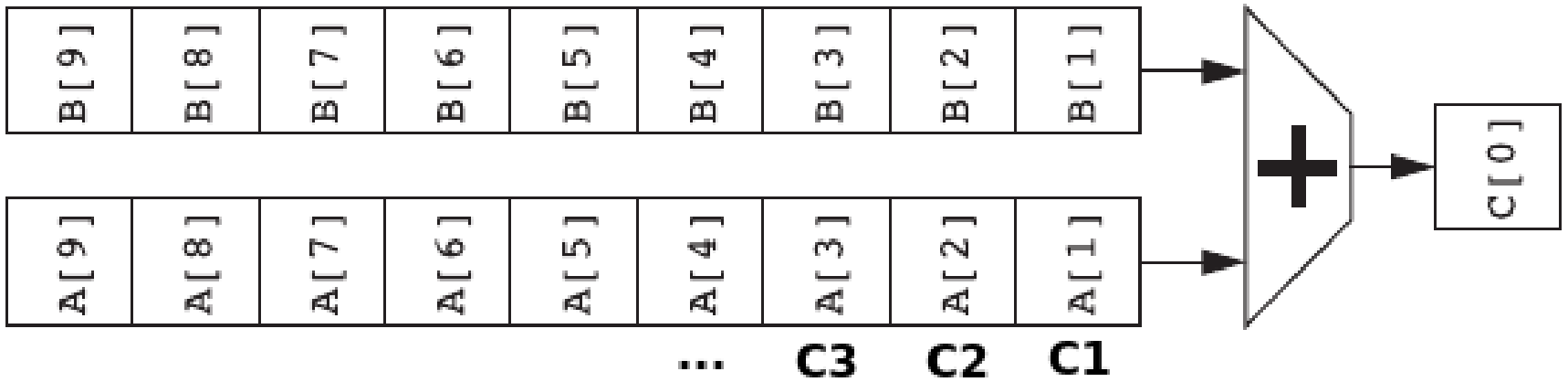
Registros vectoriales

- Cada vector es un espacio de tamaño fijo.
- VMIPS tiene 8 registros vectoriales.
- Cada uno almacena 64 elementos.
- Cada elemento es de 64 bits.
- Posee 16 puertos de lectura y 8 puertos de escritura conectados a las entradas y salidas de las unidades funcionales.

Unidades funcionales

- Cada FU utiliza pipeline (segmentadas).
- Ejecuta una operación cada ciclo de reloj.
- Es necesaria una unidad de control para detectar riesgos estructurales y por accesos a registros.
- FU para suma/resta, división, multiplicación, integer, lógicas.

Unidades funcionales



Unidad de carga y escritura

- La acción de cargar y escribir utiliza pipeline.
  - Mover palabra por palabra entre el registro y memoria.
- Se mueve una palabra por ciclo de reloj.
  - Se debe contabilizar una latencia inicial (llenar el pipe).
- También soporta carga y escritura de escalares.

Registros escalares

- Los registros escalares también pueden ser entradas de las unidades funcionales.
- Sirven para calcular direcciones para las unidades de carga y escritura.
- Son 32 registros de propósito general y 32 de punto flotante.
- Una entrada de las unidades funcionales vectoriales está conectada al banco de registros.

VMIPS

Instruction	Operands	Function
ADDVV.D	V1,V2,V3	Add elements of V2 and V3, then put each result in V1.
ADDVS.D	V1,V2,F0	Add F0 to each element of V2, then put each result in V1.
SUBVV.D	V1,V2,V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS.D	V1,V2,F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV.D	V1,F0,V2	Subtract elements of V2 from F0, then put each result in V1.
MULVV.D	V1,V2,V3	Multiply elements of V2 and V3, then put each result in V1.
MULVS.D	V1,V2,F0	Multiply each element of V2 by F0, then put each result in V1.
DIVVV.D	V1,V2,V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS.D	V1,V2,F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV.D	V1,F0,V2	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 to 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--VV.D S--VS.D	V1,V2 V1,F0	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--VS.D performs the same compare but using a scalar value as one operand.
POP	R1,VM	Count the 1s in vector-mask register VM and store count in R1.
CVM		Set the vector-mask register to all 1s.
MTC1 MFC1	VLR,R1 R1,VLR	Move contents of R1 to vector-length register VL. Move the contents of vector-length register VL to R1.
MVTM MVFM	VM,F0 F0,VM	Move contents of F0 to vector-mask register VM. Move contents of vector-mask register VM to F0.

MIPS vs VMIPS

La operación DAXPY.

$$Y = a \times X + Y$$

DAXPY en MIPS

```
        L.D     F0,a            ;load scalar a
        DADDIU  R4,Rx,#512      ;last address to load
Loop:   L.D     F2,0(Rx)        ;load X[i]
        MUL.D   F2,F2,F0        ;a × X[i]
        L.D     F4,0(Ry)        ;load Y[i]
        ADD.D   F4,F4,F2        ;a × X[i] + Y[i]
        S.D     F4,9(Ry)        ;store into Y[i]
        DADDIU  Rx,Rx,#8        ;increment index to X
        DADDIU  Ry,Ry,#8        ;increment index to Y
        DSUBU   R20,R4,Rx       ;compute bound
        BNEZ    R20,Loop        ;check if done
```

DAXPY en VMIPS

```
L.D       F0,a       ;load scalar a
LV        V1,Rx      ;load vector X
MULVS.D   V2,V1,F0   ;vector-scalar multiply
LV        V3,Ry      ;load vector Y
ADDVV.D   V4,V2,V3   ;add
SV        V4,Ry      ;store the result
```

MIPS vs VMIPS

```
        L.D       F0,a          ;load scalar a
        DADDIU    R4,Rx,#512    ;last address to load
Loop:   L.D       F2,0(Rx)      ;load X[i]
        MUL.D     F2,F2,F0      ;a × X[i]
        L.D       F4,0(Ry)      ;load Y[i]
        ADD.D     F4,F4,F2      ;a × X[i] + Y[i]
        S.D       F4,9(Ry)      ;store into Y[i]
        DADDIU    Rx,Rx,#8      ;increment index to X
        DADDIU    Ry,Ry,#8      ;increment index to Y
        DSUBU     R20,R4,Rx     ;compute bound
        BNEZ      R20,Loop      ;check if done
```

600 instrucciones

```
L.D       F0,a          ;load scalar a
LV        V1,Rx         ;load vector X
MULVS.D   V2,V1,F0      ;vector-scalar multiply
LV        V3,Ry         ;load vector Y
ADDVV.D   V4,V2,V3      ;add
SV        V4,Ry         ;store the result
```

6 instrucciones

Análisis

- En vectorial se trabaja con los 64 elementos.
- En MIPS se genera un *stall* entre el ADD y MUL y entre STORE y ADD.

```
        L.D       F0,a          ;load scalar a
        DADDIU    R4,Rx,#512    ;last address to load
Loop:   L.D       F2,0(Rx)      ;load X[i]
        MUL.D     F2,F2,F0      ;a × X[i]
        L.D       F4,0(Ry)      ;load Y[i]
        ADD.D     F4,F4,F2      ;a × X[i] + Y[i]
        S.D       F4,9(Ry)      ;store into Y[i]
        DADDIU    Rx,Rx,#8      ;increment index to X
        DADDIU    Ry,Ry,#8      ;increment index to Y
        DSUBU     R20,R4,Rx     ;compute bound
        BNEZ      R20,Loop      ;check if done
```

Análisis

- En vectorial se trabaja con los 64 elementos.
- En MIPS se genera un *stall* entre el ADD y MUL y entre STORE y ADD.
- En VMIPS este *stall* sólo se realiza la primera vez.
- El *stall* se requiere por vector, no por dato.
- **Encadenamiento:** adelantamiento de elementos con dependencia de datos.

Tiempo de ejecución vectorial

El tiempo de ejecución de una secuencia de operaciones vectoriales depende de:

- Longitud de los operandos vectoriales.
- Riesgos estructurales en las operaciones.
- Dependencia de los datos.

Conceptos

- **Tasa de iniciación:** tasa en que una unidad vectorial consume nuevos operandos y produce nuevos resultados.
  - 2 resultados por ciclo de reloj.
- **Carril (lane):** pipeline en paralelo para ejecutar más de una instrucción a la vez. Afecta la tasa de iniciación. (1 lane = 1 FU segmentada)
- **Convoy:** conjunto de instrucciones vectoriales que potencialmente se pueden ejecutar juntas.
  - No pueden existir riesgos estructurales.

Convoy – Ejemplo

Determine cuántos convoys hay en el siguiente código VMIPS. Asuma un único *lane* por FU.

```
LV      V1,Rx
MULVS.D V2,V1,F0
LV      V3,Ry
ADDVV.D V4,V2,V3
SV      V4,Ry
```

Convoy – Ejemplo

Determine cuántos convoys hay en el siguiente código VMIPS. Asuma un único *lane* por FU.

```
LV      V1,Rx
MULVS.D V2,V1,F0
LV      V3,Ry
ADDVV.D V4,V2,V3
SV      V4,Ry
```

Conceptos

- **Chime:** unidad de tiempo que toma ejecutar un convoy.
- En vectores de largo *n* el tiempo de ejecución es aproximadamente *m***n* ciclos.
- 1 lane: $T_e = n * m$
- n es el número de operandos en vector.
- m número de convoys en la secuencia de instrucciones.

Tiempo de ejecución – Ejemplo

1. ¿Cuántos *chimes* se requieren para ejecutar la secuencia de instrucciones vectoriales?
2. ¿Cuántos ciclos tomará ejecutar la secuencia?
3. Determine la cantidad de FLOPS si se trabaja a una frecuencia de 1GHz.

```
LV        V1,Rx
MULVS.D   V2,V1,F0
LV        V3,Ry
ADDVV.D   V4,V2,V3
SV        V4,Ry
```

Tiempo de ejecución – Ejemplo

1. 3 convoys = 3 chimes.
2. m = 3; n = 64; $Tiempo\ ejecución = 64 \times 3 = 192\ ciclos$
3. Los FLOPS se calculan de la siguiente manera:

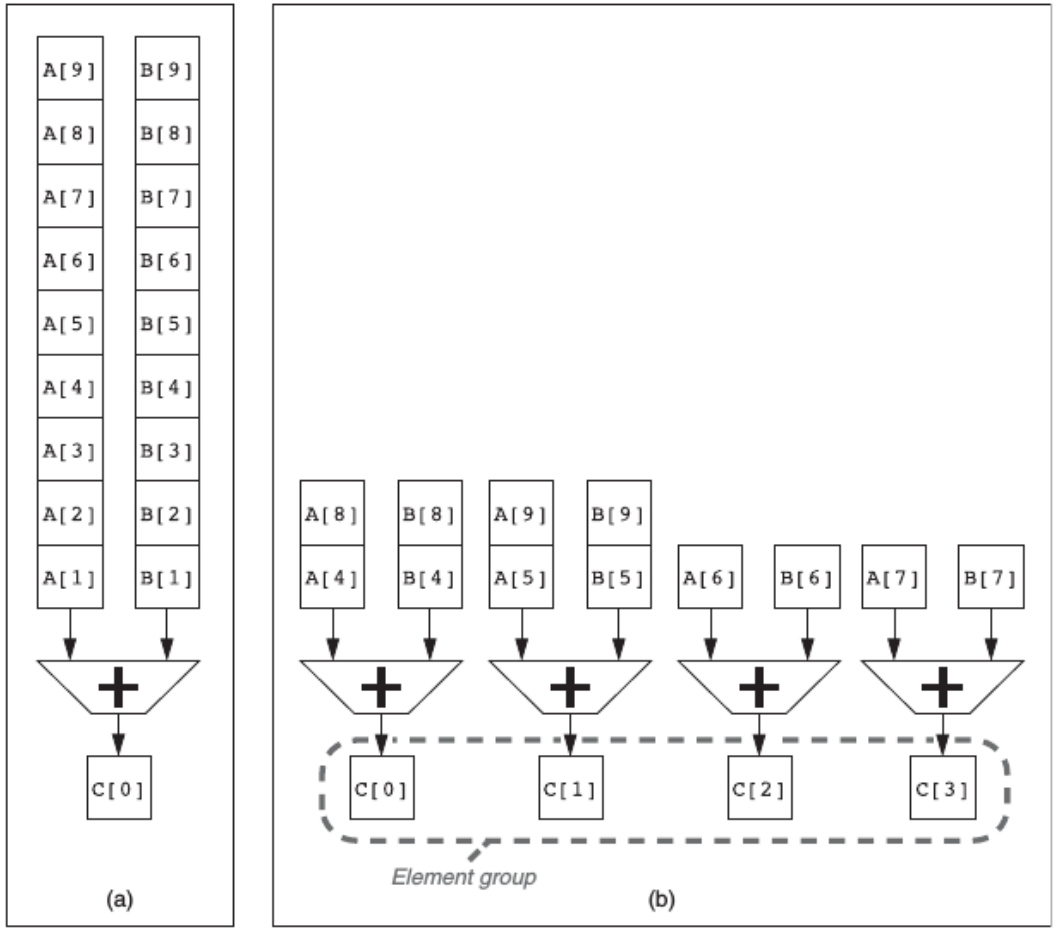
$$FLOPS = \frac{2FLOP}{3chime} \times \frac{1chime}{64ciclos} \times \frac{1ciclo}{10^{-9}s} = 10.42MFLOPS$$

No se cuentan tiempos de iniciación.

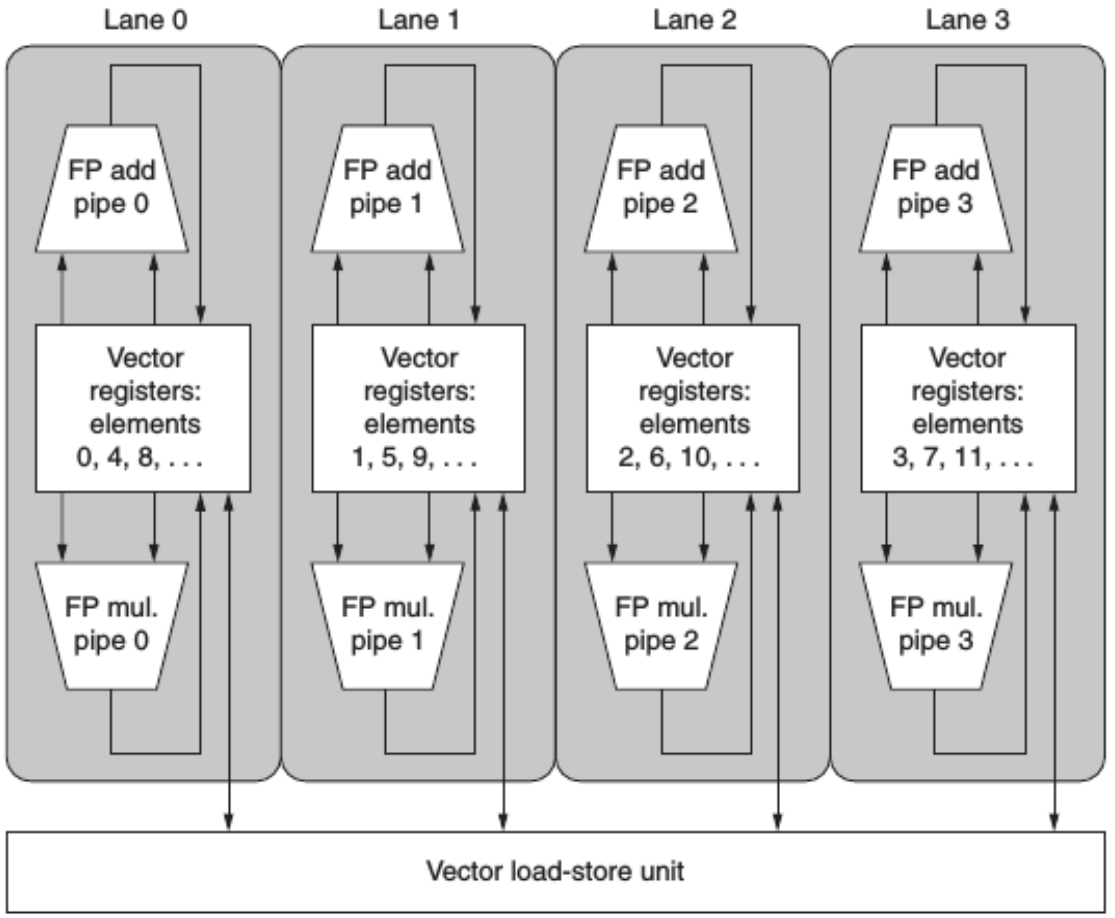
Múltiples carriles

- En otras técnicas utilizar pipeline ha aumentado el desempeño.
- En el caso de vectorial podemos aumentar las unidades funcionales (lanes).

Múltiples carriles



Múltiples carriles



Condicionales en VMIPS

```
for (i = 0; i < 64;  i=i+1)
   if (X[i] != 0)
      X[i] = X[i] – Y[i];
```

- El vector no puede ejecutarse con operaciones vectoriales por la presencia del *if*.
- Se utiliza una extensión llamada *control de vector-máscara.*

Vector máscara

- Busca la ejecución de condiciones de cada elemento en una instrucción vectorial.
- Utiliza un vector booleano para controlar la ejecución de la instrucción vectorial.
- Cuando está habilitado, cualquier instrucción opera sólo en los elementos del vector que tienen un “uno” en el vector máscara.
- Los que tienen “cero” no se ven afectadas.

Vector máscara

```
for (i = 0; i < 64;  i=i+1)
    if (X[i] != 0)
        X[i] = X[i] – Y[i];
```

```
LV          V1,Rx        ;load vector X into V1
LV          V2,Ry        ;load vector Y
L.D         F0,#0        ;load FP zero into F0
SNEVS.D     V1,F0        ;sets VM(i) to 1 if V1(i)!=F0
SUBVV.D     V1,V1,V2     ;subtract under vector mask
SV          V1,Rx        ;store the result in X
```

Vector máscara

```
for (i = 0; i < 64;  i=i+1)
    if (X[i] != 0)
        X[i] = X[i] – Y[i];
```

```
LV          V1,Rx        ;load vector X into V1
LV          V2,Ry        ;load vector Y
L.D         F0,#0        ;load FP zero into F0
SNEVS.D     V1,F0        ;sets VM(i) to 1 if V1(i)!=F0
SUBVV.D     V1,V1,V2     ;subtract under vector mask
SV          V1,Rx        ;store the result in X
```

Investigar

- ¿Qué sucede con el tamaño del vector (más/menos elementos que el tamaño del vector?
- ¿Qué sucede con memoria y las estructuras de datos (tensores, matrices, etc)?
- ¿Cómo lidia el compilador con estos casos? Vectorización o loop unrolling?

Referencias

- Stallings, W. (2003). Computer organization and architecture: designing for performance. Pearson Education India.
- Hennessy, J., & Patterson, D. (2012). Computer Architecture: A Quantitative Approach (5th ed.). Elsevier Science.

CE4302 – Arquitectura de Computadores II

Arquitectura vectorial

PROFESOR: ING. LUIS BARBOZA ARTAVIA