Centro Asociado Palma de Mallorca

Actividades Ingeniería Computadores II

Tutor: Antonio Rivero Cuesta

Centro Asociado Palma de Mallorca

Actividad 3.1

Tutor: Antonio Rivero Cuesta

Considere un procesador VLIW con un formato de instrucción que permite emitir simultáneamente operaciones a 2 unidades funcionales para el acceso a memoria (2 ciclos de latencia), a 2 unidades funcionales para operaciones en coma flotante (3 ciclos de latencia) y a una unidad funcional para operaciones enteras y de salto (1 ciclo de latencia donde el salto tiene un hueco de retardo de 1 ciclo). Dado el siguiente fragmento de código intermedio:

```
inicio:  LD     F2,0(R1)
         MULTD  F2,F2,F0
         LD     F4,0(R2)
         ADDD   F4,F2,F4
         SD     0(R2),F4
         SUBI   R1,R1,#8
         SUBI   R2,R2,#8
         BNEZ   R1,inicio
```

- Desenrolle la secuencia cuatro veces y planifique el bucle desenrollado agrupando las instrucciones por su tipo.
- Planifique el bucle desenrollado en el apartado anterior en forma de instrucciones VLIW teniendo en cuenta las características del procesador para evitar cualquier detención del cauce.
- Si el tamaño de una instrucción de código intermedio es 4 bytes, calcule el tamaño del código VLIW resultante y el espacio de almacenamiento que se desaprovecha.
- Considerando que el bucle original y el desenrollado y planificado se ejecuta sin detenciones, esto es, un ciclo por instrucción, calcule los ciclos consumidos al ejecutar 1000 veces el código original, al desenrollado planificado y el VLIW.

```
Inicio: LD     F2, 0(R1)     //Iteración i
        MULTD  F2, F2, F0
        LD     F4, 0(R2)
        ADDD   F4, F2, F4
        SD     0(R2), F4
        LD     F6, -8(R1)    //Iteración i+1
        MULTD  F6, F6, F0
        LD     F8, -8(R2)
        ADDD   F8, F6, F8
        SD     -8(R2), F8
        LD     F10, -16(R1)  //Iteración i+2
        MULTD  F10, F10, F0
        LD     F12, -16(R2)
        ADDD   F12, F10, F12
        SD     -16(R2), F12
        LD     F14, -24(R1)  //Iteración i+3
        MULTD  F14, F14, F0
        LD     F16, -24(R2)
        ADDD   F16, F14, F16
        SD     -24(R2), F16
        SUBI   R1, R1, #32
        SUBI   R2, R2, #32
        BNEZ   R1, inicio
```

```
inicio:LD     F2, 0(R1)
       LD     F4, 0(R2)
       LD     F6, -8(R1)
       LD     F8, -8(R2)
       LD     F10, -16(R1)
       LD     F12, -16(R2)
       LD     F14, -24(R1)
       LD     F16, -24(R2)
       MULTD  F2,F2,F0
       MULTD  F6,F6,F0
       MULTD  F10,F10,F0
       MULTD  F14, F14, F0
       ADDD   F4, F2, F4
       ADDD   F8, F6, F8
       ADDD   F12,F10,F12
       ADDD   F16, F14, F16
       SD     0(R2), F4
       SD     -8(R2), F8
       SD     -16(R2), F12
       SD     -24(R2), F16
       SUBI   R1, R1, #32
       SUBI   R2, R2, #32
       BNEZ   R1, inicio
```

Código VLIW

Nº	LD/SD1	LD/SD2	FPU1	FPU2	ALU
1	i1 LD F2,0(R1)	i2 LD F6,-8(R1)			
2	i3 LD F10,-16(R1)	i4 LD F14,-24(R1)			
3	i9 LD F4, 0(R2)	i10 LD F8,-8(R2)	i5 MULTD F2,F2,F0	i6 MULTD F6,F6,F0	
4	i11 LD F12,-16(R2)	i12 LDF16,-24(R2)	i7 MULTD F10,F10,F0	i8 MULTD F14,F14,F0	
5					
6			i13 ADDD F4,F2,F4	i14 ADDD F8,F6,F8	
7			i15 ADDD F12,F10,F12	i16 ADDD F16,F14,F16	
8					
9	i17 SD 0(R2),F4	i18 SD -8(R2),F8			i21 SUBI R1,R1,#32
10	i19 SD -16(R2),F12	i20 SD -24(R2),F16			i22 BENZ R1,inicio
11					i23 SUBI R2,R2,#32

La reducción a 11 instrucciones se consigue con la estrategia indicada en el libro de adelantar el decremento de los índices, pasando i21 e i22 a huecos en la ALU después de las cargas, por ejemplo las VLIW Nº 5 y 6, y luego actualizando en los almacenamientos (i17-i20) el desplazamiento sobre R2 (lo pongo en azul en la tabla)

Apartado c.

Tamaño código original: 8 instrucciones · 4 bytes = 32 bytes
Tamaño código desenrollado: 23 instrucciones · 4 bytes = 92 bytes
Tamaño código VLIW: 11 instrucciones · 20 bytes = 220 bytes
Espacio desaprovechado en código VLIW: 32 slots · 4 bytes = 128 bytes

Apartado d.

Ciclos código original: 1000 iteraciones · 8 ciclos = 8000 ciclos
Ciclos código desenrollado: 250 iteraciones · 23 ciclos = 5750 ciclos
Ciclos código VLIW: 250 iteraciones · 11 ciclos = 2750 ciclos

Centro Asociado Palma de Mallorca

Actividad 3.2

Tutor: Antonio Rivero Cuesta

Considere el siguiente bucle:

```
inicio: LD    F0,0(R1)
        ADDD  F4,F0,F2
        SD    0(R1),F4
        SUBI  R1,R1,#8
        BNEZ  R1,inicio
```

- Genere el código intermedio aplicando segmentación software pero evite la utilización de desplazamientos negativos en los accesos a memoria del patrón y considere que la latencia de las instrucciones es de 1 ciclo.

- Utilizando el código obtenido en el apartado anterior, escriba el código VLIW teniendo en cuenta que el formato de instrucción admite una operación de carga/almacenamiento (2 ciclos de latencia), una operación de coma flotante (3 ciclos de latencia) y una operación entera/salto (1 ciclo de latencia y el salto consume un hueco de retardo).

- ¿Por qué si se aplica la técnica de segmentación software al bucle original sin considerar las latencias verdaderas no se aprovecha realmente la segmentación hardware de las unidades funcionales?

Apartado a

```
          LD    F0,0(R1)    // Carga de X[i]
          ADDD  F4,F0,F2    // X[i] = X[i] + a
          LD    F0,-8(R1)   // Carga de X[i-1]
          SUBI  R1,R1,#16
inicio:   SD    16(R1),F4   // Almacenamiento X[i]
          ADDD  F4,F0,F2    // X[i-1] = X[i-1] + a
          LD    F0,0(R1)    // Carga de X[i-2]
          SUBI  R1,R1,#8
          BNEZ  R1,inicio   // ¿Fin del bucle: R1=0?
          SD    16(R1),F4   // Almacenamiento  X[2]
          ADDD  F4,F0,F2    // X[1] = X[1] + a
          SD    8(R1),F4    // Almacenamiento  X[1]
```

Apartado b. El código VLIW sería el siguiente:

	Memoria	Coma Flotante	Entero/Salto
prólogo:	`LD F0,0(R1)`	-	-
	`LD F0,-8(R1)`	-	-
	-	`ADDD F4,F0,F2`	`SUBI R1,R1,#16`
	-	-	-
	-	-	-
inicio:	`SD 16(R1),F4`	`ADDD F4,F0,F2`	`SUBI R1,R1,#8`
	`LD F0,8(R1)`	-	`BNEZ R1,inicio`
	-	-	-
epílogo:	`SD 16(R1),F4`	`ADDD F4,F0,F2`	-
	-	-	-
	-	-	-
	`SD 8(R1),F4`	-	-

El total de ciclos ejecutados para procesar un vector de 1000 elementos sería:

3009 ciclos

$5+1000 \cdot 3+ 4 = 3009$ ciclos

Apartado c

No se aprovechan correctamente los cauces de las unidades funcionales ya que en un momento dado solo hay una operación en cada unidad funcional: solo se admite un dato en las unidades funcionales, no tantos como segmentos tienen. Ello es debido a que si no se consideran las latencias de las unidades, la técnica no se aplica de forma óptima.

El patrón correcto se obtiene reflejando las latencias de las unidades:

LD F0,0(R1)	-	-	-	-	-
-	LD F0,-8(R1)	-	-	-	-
ADDD F4,F0,F2	-	LD F0,-16(R1)	-	-	-
-	ADDD F4,F0,F2	-	LD F0,-24(R1)	-	-
-	-	ADDD F4,F0,F2	-	LD F0,-32(R1)	-
SD 0(R1),F4	-	-	ADDD F4,F0,F2	-	LD F0,-40(R1)
-	SD -8(R1),F4	-	-	ADDD F4,F0,F2	-
-	-	SD -16(R1),F4	-	-	ADDD F4,F0,F2
-	-	-	SD -24(R1),F4	-	-
-	-	-	-	SD -32(R1),F4	-
-	-	-	-	-	SD -40(R1),F4
-	-	-	-	-	

Si se transforma la secuencia anterior en instrucciones VLIW genéricas, se obtiene el siguiente fragmento de código:

```
LD F0,0(R1)
LD F0,-8(R1)
LD F0,-16(R1)  ADDD F4, F0, F2
LD F0,-24(R1)  ADDD F4, F0, F2
LD F0,-32(R1)  ADDD F4, F0, F2
SD 0(R1),F4    ADDD F4, F0, F2  LD F0, -40(R1)
SD -8(R1),F4   ADDD F4, F0, F2
SD -16(R1),F4  ADDD F4, F0, F2
SD -24(R1),F4
SD -32(R1),F4
SD -40(R1),F4
```

Observe que si el formato de la instrucción VLIW no se ajustase al patrón, por ejemplo, por disponer solo de una operación de acceso a memoria, habría problemas ya que el

patrón habría que descomponerlo en dos instrucciones VLIW al contar con una operación de carga y otra de almacenamiento.
Esto produciría la pérdida de datos al romperse la secuencialidad en el flujo de datos:

```
LD F0,0(R1)
LD F0,-8(R1)
LD F0,-16(R1)  ADDD F4,F0,F2
LD F0,-24(R1)  ADDD F4,F0,F2  // Se pierde su resultado
LD F0,-32(R1)  ADDD F4,F0,F2
SD 0(R1),F4    ADDD F4,F0,F2
LD F0,-40(R1)                 // Nadie lee el valor de F4
SD -8(R1),F4   ADDD F4,F0,F2  // ya que se machaca el valor previo de F4
SD -16(R1),F4  ADDD F4,F0,F2
SD -24(R1),F4
SD -32(R1),F4
SD -40(R1),F4
```

Centro Asociado Palma de Mallorca

Actividad 3.3

Tutor: Antonio Rivero Cuesta

Dado el siguiente fragmento de código:

```
inicio: LD    F2,0(R1)
        MULTD F4,F2,F10
        LD    F6,0(R2)
        ADDD  F8,F6,F4
        SD    0(R2),F8
        SUBI  R1,R1,#8
        SUBI  R2,R2,#8
        BNEZ  R1,inicio
```

Donde **F6** y **F10** contienen valores previamente cargados. Se pide que:

- Aplique segmentación software y obtenga el patrón de comportamiento considerando que los accesos a memoria consumen dos ciclos y las operaciones de coma flotante tres ciclos.
- Genere el correspondiente pseudocódigo VLIW para un procesador VLIW genérico en el que la unidad de salto consume un ciclo.

a) La siguiente tabla muestra el patrón de ejecución que se ha obtenido al realizar once iteraciones del bucle y considerando las latencias de las unidades funcionales.

LD F2, 0(R1)								
	LD F2, -8(R1)							
MULTD F4,F2,F10		LD F2, -16(R1)						
	MULTD F4,F2,F10		LD F2, -24(R1)					
		MULTD F4,F2,F10		LD F2, -32(R1)				
LD F6, 0(R2)			MULTD F4,F2,F10		LD F2, -40(R1)			
	LD F6, -8(R2)			MULTD F4,F2,F10		LD F2, -48(R1)		
ADDD F8, F6, F4		LD F6, -16(R2)			MULTD F4,F2,F10		LD F2, -56(R1)	
	ADDD F8, F6, F4		LD F6, -24(R2)			**MULTD F4,F2,F10**		**LD F2, -64(R1)**
		ADDD F8, F6, F4		LD F6, -32(R2)				
SD 0(R2), F8			**ADDD F8, F6, F4**		LD F6, -40(R2)		MULTD F4,F2,F10	
	SD -8(R2), F8			ADDD F8, F6, F4		LD F6, -48(R2)		MULTD F4,F2,F10
		SD -16(R2), F8			ADDD F8, F6, F4		LD F6, -56(R2)	
			SD -24(R2), F8			ADDD F8, F6, F4		**LD F6, -64(R2)**
				SD -32(R2), F8			ADDD F8, F6, F4	
					SD -40(R2), F8			ADDD F8, F6, F4
						SD -48(R2), F8		
							SD -56(R2), F8	
								SD -64(R2), F8

b) El pseudocódigo VLIW derivado del anterior patrón de comportamiento es el siguiente:

	LD F2, 0(R1)				
	LD F2, -8(R1)				
	LD F2, -16(R1)	MULTD F4, F2, F10			
	LD F2, -24(R1)	MULTD F4, F2, F10			
	LD F2, -32(R1)	MULTD F4, F2, F10			
	LD F6, 0(R2)	LD F2, -40(R1)	MULTD F4, F2, F10		
	LD F6, -8(R2)	LD F2, -48(R1)	MULTD F4, F2, F10		
	LD F6, -16(R2)	LD F2, -56(R1)	ADDD F8, F6, F4	MULTD F4, F2, F10	
	LD F6, -24(R2)	LD F2, -64(R1)	ADDD F8, F6, F4	MULTD F4, F2, F10	
	LD F6, -32(R2)	LD F2, -72(R1)	ADDD F8, F6, F4	MULTD F4, F2, F10	
Inicio:	**LD F2, -64(R1)**	**LD F6, -40(R2)**	**ADDD F8, F6, F4**	**MULTD F4, F2, F10**	**LD F2, -80(R1) if (R1<>80) go inicio**
	SD -8(R2), F8	LD F6, -48(R2)	ADDD F8, F6, F4	MULTD F4, F2, F10	
	SD -16(R2), F8	LD F6, -56(R2)	ADDD F8, F6, F4	MULTD F4, F2, F10	
	SD -24(R2), F8	LD F6, -64(R2)	ADDD F8, F6, F4		
	SD -32(R2), F8	LD F6, -72(R2)	ADDD F8, F6, F4		
	SD -40(R2), F8	LD F6, -80(R2)	ADDD F8, F6, F4		
	SD -48(R2), F8	ADDD F8, F6, F4			
	SD -56(R2), F8	ADDD F8, F6, F4			
	SD -64(R2), F8				

Observe que si se trata de transformar el pseudocódigo en instrucciones que admiten una o dos operaciones de acceso a memoria surgen problemas.

El cuerpo del bucle segmentado se debe descomponer en dos o tres instrucciones VLIW lo que rompe la secuencialidad en

el flujo de datos que se obtiene al aplicar segmentación software en el patrón.

Al partir el cuerpo del bucle en dos o tres instrucciones VLIW, las unidades de coma flotante producen resultados que nadie recoge en las primeras iteraciones ya que sus segmentaciones estaban cargadas con tantas operaciones consecutivas como profundas son sus cauces.

Al tener que añadir instrucciones VLIW con slots de operación vacíos, los resultados que quedan en las etapas interiores de su segmentación machacan los resultados previos, ya que nadie los recoge.

Centro Asociado Palma de Mallorca

Actividad 3.4

Tutor: Antonio Rivero Cuesta

Un procesador VLIW cuenta con un formato de instrucción que admite una operación de carga/almacenamiento (2 ciclos de latencia), una unidad de coma flotante (3 ciclos de latencia) y una unidad para operaciones enteras y de salto (1 ciclo de latencia y un hueco de retardo). Aplique segmentación software al siguiente bucle y genere el correspondiente código VLIW:

```
inicio:  LD    F0,0(R1)
         ADDD  F4,F0,F2
         SD    0(R1),F4
         SUBI  R1,R1,#8
         BNEZ  R1,inicio
```

LD F0, 0(R1)	-	-	-	-	-
-	LD F0, -8(R1)	-	-	-	-
ADDD F4, F0, F2	-	LD F0, -16(R1)	-	-	-
-	ADDD F4, F0, F2	-	LD F0, -24(R1)	-	-
-	-	ADDD F4, F0, F2	-	LD F0, -32(R1)	-
SD 0(R1), F4	-	-	**ADDD F4, F0, F2**	-	**LD F0, -40(R1)**
-	SD -8(R1), F4	-	-	ADDD F4, F0, F2	-
-	-	SD -16(R1), F4	-	-	ADDD F4, F0, F2
-	-	-	SD -24(R1), F4	-	-
-	-	-	-	SD -32(R1), F4	-
-	-	-	-	-	SD -40(R1), F4

El principal problema que surge es que el formato de instrucción VLIW dispone de únicamente un campo para operaciones de acceso a memoria y el cuerpo del bucle consta de una carga y un almacenamiento.
Ello obliga a generar varias instrucciones VLIW para el cuerpo del bucle segmentado con el riesgo de pérdidas de datos.

Una posible solución es intentar generar un patrón que responda al formato de instrucción VLIW, esto es, una operación de acceso a memoria, una de coma flotante y una entera.
La siguiente tabla presenta una solución.

LD F0, 0(R1)	-	-	-
-	-	-	-
ADDD F4, F0, F2	LD F0, -8(R1)	-	-
-	-	-	-
-	ADDD F4, F0, F2	LD F0, -16(R1)	-
SD 0(R1), F4	-	-	-
-	-	**ADDD F4, F0, F2**	**LD F0, -24(R1)**
-	SD -8(R1), F4	-	-
-	-	-	ADDD F4, F0, F2
-	-	SD -16(R1), F4	-
-	-	-	-
-	-	-	SD -24(R1), F4

Una posible transformación de la secuencia previa en instrucciones VLIW se presenta a continuación. Preste atención al juego que se tiene que hacer con los valores del desplazamiento para aprovechar el hueco de retardo del salto.

	Carga/almacenamiento	Coma flotante	Entera
	LD F0, 0(R1)		
	LD F0, -8(R1)	ADDD F4, F0, F2	
	LD F0, -16(R1)	ADDD F4, F0, F2	SUBI R1, R1, #32
Inicio:	**SD 32(R1), F4**	**-**	**BNEZ R1, inicio**
	LD F0, 8(R1)	**ADDD F4, F0, F2**	**SUBI R1, R1, #8**
	SD 32(R1), F4		
		ADDD F4, F0, F2	
	SD 24(R1), F4		
	SD 16(R1), F4		

Centro Asociado Palma de Mallorca

Actividad 3.5

Tutor: Antonio Rivero Cuesta

Dado el siguiente bucle escrito en pseudocódigo:

```
for (i=0; i<n; i++)
  if (A[i]==0) then
    X[i]:= X[i]+a;
  else
    Y[i]:= Y[i]-a;
  end if;
end for;
```

- Escriba el código intermedio genérico.

- Escriba el código intermedio aplicando la técnica ***if – conversion***.

- Escriba el código VLIW de los apartados anteriores teniendo en cuenta que el formato de instrucción admite una operación de carga/almacenamiento (2 ciclos de latencia), una operación de coma flotante (3 ciclos de latencia) y una operación entera/salto (1 ciclo de latencia y el salto consume un hueco de retardo). Compare los tiempos de ejecución para procesar un vector de 1000 elementos suponiendo que la probabilidad de ambas ramas del **if** es la misma.

a) Un posible código intermedio genérico correspondiente al bucle es el siguiente:

```
inicio: LD R5,0(R1)     // Cargar A[i]
        BNEZ R5,else    // Si (A[i] <> 0) saltar a else
then:   LD F4,0(R2)     // Cargar X[i]
        ADDD F4,F4,F2   // X[i] = X[i] + a
        SD 0(R2),F4     // Almacenar X[i]
        JMP final
else:   LD F4,0(R3)     // Cargar Y[i]
        SUBD F4,F4,F2   // Y[i] = Y[i] - a
        SD 0(R3),F4     // Almacenar Y[i]
final:  SUBI R2,R2,#8   // Decrementar índice de X en 8 bytes
        SUBI R3,R3,#8   // Decrementar índice de Y en 8 bytes
        SUBI R1,R1,#4   // Decrementar índice de A en 4 bytes
        BNEZ R1,inicio  // Comenzar una nueva iteración
```

b)

```
inicio: LD R5,0(R1)
        PRED_EQ p1,p2,R5,#0
        LD F4,0(R2) (p1)
        ADDD F4,F4,F2 (p1) SD 0(R2),F4 (p1)
        LD F4,0(R3) (p2) SUBD F4,F4,F2 (p2)
        SD 0(R3),F4 (p2)
final:  SUBI R2,R2,#8
        SUBI R3,R3,#8
        SUBI R1,R1,#4
        BNEZ R1,inicio
```

c) Un posible código VLIW con optimizaciones derivado del código intermedio del apartado "a" sería el siguiente:

	Carga/almacenamiento	Coma flotante	Entera/salto	
Inicio:	LD R5, 0(R1)		SUBI R1, R1, #4	1
Then:	LD F4, 0(R2)			2
			BNEZ R5, else	3
		ADDD F4, F4, F2		4
				5
				6
	SD 0(R2), F4		JMP final	7
Else:	LD F4, 0(R3)		SUBI R2, R2, #8	8
				9
		SUBD F4, F4, F2		10
				11
				12
	SD 0(R3), F4			13
Final:			BNEZ R1, inicio	14
			SUBI R3, R3, #8	15

Observe que se ha realizado una carga especulativa en la rama then ya que la instrucción de carga LD F4, 0(R2) se ejecuta con independencia del resultado de la comparación. De esta forma, si el salto es efectivo ya se habrá iniciado el procesamiento de la carga y si no es efectivo, el valor almacenado en F4 quedará anulado al iniciarse la ejecución de la instrucción de carga correspondiente a la rama else, esto es, LD F4, 0(R3).

Ciclos consumidos al ejecutar la rama then: 10 ciclos
Ciclos consumidos al ejecutar la rama else:12 ciclos
Ciclos consumidos ejecutar bucle de 1000 elementos: 10*500+12*500=11000 ciclos
Tamaño del código VLIW: 15 instrucciones * 12 bytes = 180 bytes
Porción de código VLIW vacío: 32 slots * 4 bytes = 128 bytes

Y para el apartado "b", un posible código VLIW sería:

	Carga/almacenamiento	**Coma flotante**	**Entera/salto**
Inicio:	LD R5, 0(R1)		
			PRED_EQ p1, p2, R5, #0
	LD F4, 0(R2) (p1)		
	LD F4, 0(R3) (p2)		
		ADDD F4, F4, F2 (p1)	
		SUBD F4, F4, F2 (p2)	
			SUBI R2, R2, #8
	SD 0(R2), F4 (p1)		SUBI R1, R1, #4
	SD 0(R3), F4 (p2)		BNEZ R1, inicio
			SUBI R3, R3, #8

Ciclos consumidos ejecutar bucle de 1000 elementos: 11*1000=11000 ciclos

Tamaño del código VLIW: 11 instrucciones * 12 bytes = 132 bytes

Porción de código VLIW vacío: 21 slots * 4 bytes = 84 bytes

Centro Asociado Palma de Mallorca

Actividad 3.6

Tutor: Antonio Rivero Cuesta

Dado el siguiente bucle escrito en pseudocódigo:

```
for (i=0; i<n; i++)
  if (A[i]==0) then
    X[i]:= X[i]+a;
  else
    Y[i]:= Y[i]-a;
  end if;
end for;
```

Generar el código VLIW aplicando predicación y realizando la carga especulativa de los datos.

	Carga/almacenamiento	Coma Flotante	Entera/Salto
inicio:	`LD R5,0(R1)`		
	`LD F4,0(R2)`		
	`LD F6,0(R3)`		`PRED EQ p1,p2,R5,#0`
		`ADDD F4,F4,F2 (p1)`	
		`SUBD F6,F6,F2 (p2)`	
			`SUBI R2,R2,#8`
	`SD 8(R2),F4(p1)`		`SUBI R1,R1,#4`
	`SD 0(R3),F6(p2)`		`BNEZ R1,inicio`
			`SUBI R3,R3,#8`

Ciclos consumidos ejecutar bucle de 1000 elementos: 9*1000 = 9000 ciclos
Tamaño del código VLIW: 9 instrucciones * 12 bytes = 108 bytes
Porción de código VLIW vacío: 15 slots * 4 bytes = 60 bytes

Centro Asociado Palma de Mallorca

Actividad 3.7

Tutor: Antonio Rivero Cuesta

Dado el siguiente bucle escrito en pseudocódigo:

```
if (a>0) then
  a:= a+a;
else
  if (b<0) then
    b:= b+b;
  else
    c:= c+c;
  end if;
  a:= b;
end if;
```

- Escriba el código intermedio utilizando instrucciones condicionales con el fin de reducir al mínimo las sentencias de salto condicional. Las direcciones de memoria de las variables enteras a, b y c se encuentran almacenadas en M[R5], M[R6] y M[R7], respectivamente.

```
LD R1,0(R5)                 // carga de a
LD R2,0(R6)                 // carga de b
LD R3,0(R7)                 // carga de c
PRED_GT p1,p2,R1,#1         // If (a>0)
PRED_LT p3,p4,R2,#-1(p2)    // If (b<0)
ADDD R1, R1, R1(p1)         // a:=a+a
ADDD R2,R2,R2 p3)           // b:=b+b
ADDD R3,R3,R3 p4)           // c:=c+c
SD 0(R7),R3(p4)             // almacenamiento de c:=c+c
SD 0(R6),R2(p3)             // almacenamiento de b:=b+b
SD 0(R5),R2(p2)             // almacenamiento de a:=b
SD 0(R5),R1(p1)             // almacenamiento de a:=a+a
```

Centro Asociado Palma de Mallorca

Actividad 3.8

Tutor: Antonio Rivero Cuesta

Centro Asociado Palma de Mallorca

Actividad 3.9

Tutor: Antonio Rivero Cuesta

Dado los dos ejemplos de código de la figura 3.17, intente reducir el número de ciclos de ejecución mediante una adecuada reorganización de las instrucciones.

	Carga/almacenamiento	Coma flotante	Enteras/salto	
inicio:	**LD R1,0(R2)**	--	--	1
	--	--	--	2
	--	--	**SUBI R3,R1,#50**	3
	--	--	**BNEZ R3,esle**	4
	--	--	--	5
then:	--	--	**ADDI R5,R5,#2**	6
	--	--	**JUMP final**	7
	--	--	--	8
else:	--	--	**ADDI R5,R5,#1**	9
final:	--	--	**SUBI R2,R2,#4**	10
	--	--	**BNEZ R2,inicio**	11
	--	--	--	12

Código figura 3.17a

	Carga/almacenamiento	Coma flotante	Enteras/salto	
inicio:	**LD R1,0(R2)**			1
				2
			SUBI R2,R2,#8	3
			BNEZ R1,inicio	4
			SUBI R2,R2,#8	5
then:			**JUMP final**	6
			ADDI R5,R5,#2	7
else:			**ADDI R5,R5,#1**	8
final:			**BNEZ R1,inicio**	9

Otra solución válida:

	Carga/almacenamiento	Coma flotante	Enteras/salto	
inicio:	**LD R1,0(R2)**			1
			SUBI R2,R2,#4	2
			SUBI R3,R1,#50	3
			BNEZ R3,final	4
then:			**ADDI R5,R5,#1**	5
			ADDI R5,R5,#1	6
final:			**BNEZ R2,inicio**	7

inicio:	**LD R1,0(R2)**	-----	-----	1
	-----	-----	-----	2
	-----	-----	**PRED_EQ P1,P2,R1,#50**	3
	-----	-----	**ADDI R5,R5,#2(P1)**	4
	-----	-----	**ADDI R5,R5,#1(P2)**	5
	-----	-----	**SUBI R2,R2,#8**	6
	-----	-----	**BNEZ R2,inicio**	7
	-----	-----	-----	8

Código figura 3.17b

	Carga/almacenamiento	Coma flotante	Enteras/salto	
inicio:	**LD R1,0(R2)**		**SUBI R2,R2,#8**	1
				2
			PRED_EQ P1,P2,R1,#50	3
			ADDI R5,R5,#2(P1)	4
				5
			BNEZ R2,inicio	6
			ADDI R5,R5,#1(P2)	7

Centro Asociado Palma de Mallorca

Actividad 3.10

Tutor: Antonio Rivero Cuesta

Centro Asociado Palma de Mallorca

Actividad 3.11

Tutor: Antonio Rivero Cuesta

En un procesador vectorial con las siguientes características:

- Longitud vectorial máxima 64 elementos.
- Una unidad de ***suma vectorial*** con tiempo de arranque de 6 ciclos.
- Una unidad de ***multiplicación*** con tiempo de arranque de 7 ciclos.
- Una unidad de ***carga/almacenamiento vectorial*** con tiempo de arranque de 12 ciclos.
- La frecuencia del trabajo del procesador es 500 MHz.

Se pretende ejecutar el siguiente fragmento de código vectorial:

```
LV      V1,R1
MULTSV  V2,V1,F1
ADDSV   V3,V1,F1
SV      R2,V2
SV      R3,V3
```

Calcule T_{64}, $T_{elemento}$ y R_{64} bajo las siguientes condiciones:

- Sin encadenamiento entre unidades.
- Con encadenamiento entre unidades.
- Con encadenamiento y tres unidades de carga/almacenamiento.

a) Del análisis de las dependencias estructurales y de datos que existen en el fragmento de código vectorial se obtendrían cuatro convoyes:

```
Convoy 1:   LV       V1, R1
Convoy 2:   MULTSV   V2, V1, F1    // Dependencia de datos RAW
            ADDSV    V3,V1,F1
Convoy 3:   SV       R2, V2        // Dependencia de datos RAW
Convoy 4:   SV       R3, V3        // Dependencia estructural
```

La siguiente tabla ilustra cómo se obtiene el tiempo total de ejecución de la secuencia de código vectorial para vectores de longitud 64.

Convoy	Comienza	Termina	Comentario
LV V1,R1	0	12+64=76	Latencia sencilla
MULTSV V2,V1,F1	76	76+7+64=147	Espera por convoy 1
ADDSV V3,V1,F1			
SV R2,V2	147	147+12+64=223	Espera por convoy 2
SV R3,V3	223	223+12+64=299	Espera por convoy 3

$T_{64} = (12 + 64) + (7 + 64) + (12 + 64) + (12 + 64) = 299$ ciclos

$$T_{64} = \frac{299 ciclos}{500 MHz} = 0,598 \mu seg.$$

El tiempo por elemento es equivalente al número de convoyes, es decir, 4 ciclos u 8 *nseg*. En lo que respecta al rendimiento que se alcanza:

$$R_{64} = \frac{64 elementos \cdot 2 FLOP}{299 ciclos} = 0,42 FLOP/ciclo$$

que expresado en FLOPS sería:

R_{64} = 0,42 FLOP ciclo $\cdot$ 500 MHz

R_{64} = 214 MFLOPS

b) Ahora ya se permite realizar el encadenamiento de resultados entre unidades funcionales. Por lo tanto, las instrucciones de multiplicación y suma del segundo convoy pueden fusionarse con la carga del primer convoy reduciendo el número total de convoyes a tres:

```
Convoy 1:   LV      V1, R1
            MULTSV  V2, V1, F1   // Encadenada a LV
            ADDSV   V3,V1,F1     // Encadenada a LV
Convoy 2:   SV      R2, V2       // Dependencia estructural y de datos RAW
Convoy 3:   SV      R3, V3       // Dependencia estructural
```

Se tendría ahora la siguiente secuencia temporal de ejecución:

Convoy	Comienza	Termina	Comentario
LV V1,R1 MULTSV V2,V1,F1 ADDSV V3,V1,F1	0	12+7+64=83	Latencia sencilla
SV R2,V2	83	83+12+64=159	Espera por convoy 1
SV R3,V3	159	159+12+64=235	Espera por convoy 2

$\boldsymbol{T_{64}} = (12 + 7 + 64) + (12 + 64) + (12 + 64) = 235$ ciclos

$$T_{64} = \frac{235 ciclos}{500 MHz} = 0,47 \mu seg.$$

El tiempo por elemento es equivalente al número de convoyes, es decir, 3 ciclos o 6 *nseg*. En lo que respecta al rendimiento que se alcanza:

$$R_{64} = \frac{64 elementos \cdot 2 FLOP}{235 ciclos} = 0,5447 FLOP/ciclo$$

que expresado en FLOPS sería:

$$R_{64} = 0,5447 \text{ FLOP ciclo} \cdot 500 \text{ MHz}$$

$$R_{64} = 272 \text{ MFLOPS}$$

c) Dado que ahora se dispone de tres unidades de carga/almacenamiento y el encadenamiento es posible, el número de convoyes se puede reducir a uno:

```
Convoy 1:   LV      V1, R1
            MULTSV  V2, V1, F1   // Encadenada a LV
            ADDSV   V3,V1,F1     // Encadenada a LV
            SV      R2, V2       // Encadenada a MULTV
            SV      R3, V3       // Encadenada a ADDV
```

La secuencia temporal de ejecución sería la siguiente:

Convoy	Comienza	Termina	Comentario
LV V1,R1 MULTSV V2,V1,F1 ADDSV V3,V1,F1 SV R2,V2 SV R3,V3	0	12+7+12+64=95	Latencia sencilla

T_{64} = (12 +7+ 12+64) = 95 ciclos

$$T_{64} = \frac{95 ciclos}{500 MHz} = 0{,}19 \mu seg.$$

El tiempo por elemento es equivalente al número de convoyes, en este caso 1 ciclo que equivale a 2 nseg. En lo que respecta al rendimiento que se alcanza:

$$R_{64} = \frac{64 elementos \cdot 2 FLOP}{95 ciclos} = 1,347 FLOP/ciclo$$

que expresado en FLOPS sería:

R_{64} = 1,347 FLOP ciclo $\cdot$ 500 MHz

R_{64} = 673 MFLOPS

Centro Asociado Palma de Mallorca

Actividad 3.12

Tutor: Antonio Rivero Cuesta

En un procesador vectorial con las siguientes características:

- Registros con una longitud vectorial máxima 64 elementos.
- Una unidad de ***suma vectorial*** con tiempo de arranque de 6 ciclos.
- Una unidad de ***multiplicación*** con tiempo de arranque de 7 ciclos.
- Una unidad de ***carga/almacenamiento vectorial*** con tiempo de arranque de 12 ciclos.
- La frecuencia del trabajo del procesador es 500 MHz.
- T_{base} de 10 ciclos y T_{bucle} de 15 ciclos.

Se pretende ejecutar el siguiente bucle:

```
for (i=1; i<n; i++)
  A[i]:= A[i]+B[i]);
  B[i]:= a*B[i];
end for;
```

- Escriba el código vectorial que realizaría las operaciones ubicadas en el interior del bucle considerando la posibilidad de encadenamiento de resultados.
- Calcule T_n, T_{1000} , R_{1000} y R_∞.
- Calcule T_n, T_{1000} , R_{1000} y R_∞ pero ahora considerando encadenamientos y dos unidades de carga/almacenamiento.
- ¿Qué ocurre si se considera encadenamientos, dos unidades de carga/almacenamiento y se permite el solapamiento entre convoyes?

a) El mejor código vectorial será aquel que aproveche la posibilidad del encadenamiento. De acuerdo con esto, el código vectorial sería:

```
Convoy 1:  LV       V2, R2       // Carga de B
Convoy 2:  LV       V1, R1       // Carga de A
           MULTSV   V3, F0, V2   // B := x * B
           ADDV     V1, V1, V2   // A := A+B
Convoy 3:  SV       R1, V1       // Almacenamiento de A
Convoy 4:  SV       R2, V3       // Almacenamiento de B
```

Observe que ya que primero se realiza la multiplicación y para evitar un riesgo WAR es necesario renombrar el registro destino de la operación de multiplicación. De esta forma, la instrucción de suma lee el valor correcto de B (el almacenado en V2) y no el B resultante de la multiplicación (el V3).

b) La secuencia temporal de ejecución sería la siguiente:

Convoy	Comienza	Termina	Comentario
LV V2, R2	0	12+64=76	Latencia sencilla
LV V1, R1 MULTSV V3, F0, V2 ADDV V1, V1, V2	76	76+12+6+64=158	Espera por convoy 1
SV R1, V1	158	158+12+64=234	Espera por convoy 2
SV R2, V3	234	234+12+64=310	Espera por convoy 3

El tiempo en ejecutar ese conjunto de 6 instrucciones vectoriales para un vector de longitud 64 sería 310 ciclos. No serían necesarios ciclos adicionales por costes de seccionamiento dado que, si se conoce a priori que el número de elementos a procesar es 64, no sería necesario aplicar la técnica de seccionamiento por parte del compilador.

El $T_{elemento}$ es 4 ciclos dado que se tienen 4 convoyes.

El $T_{arranque}$ total se obtiene de sumar los tiempos de arranque visibles de las unidades funcionales. Si se analiza la tabla anterior, se obtiene:

$$T_{arranque} = 2 \cdot T_{arranque}\ \text{LV} + T_{arranque}\ \text{ADDV} + 2 \cdot T_{arranque}\ \text{SV}$$

$$T_{arranque} = (2 \cdot 12 + 4 + 2 \cdot 12)\ \text{ciclos} = 54\ \text{ciclos}$$

Con estos valores la expresión del tiempo total de ejecución queda

$$T_n = T_{base} + \left\lceil \frac{n}{64} \right\rceil \cdot \left(T_{bucle} + T_{arranque} \right) + n \cdot T_{elemento}$$

$$T_n = 10 + \left\lceil \frac{n}{64} \right\rceil \cdot (15 + 55) + 4n = 10 + \left\lceil \frac{n}{64} \right\rceil \cdot (70) + 4n$$

que para el caso particular de $n = 1000$

$$T_{1000} = 10 + \left\lceil \frac{1000}{64} \right\rceil \cdot (15 + 55) + 4 \cdot 1000 =$$

$$T_{1000} = 10 + 16 \cdot 70 + 4000 = 5130 \;\; ciclos$$

que expresado en segundos

$$T_{1000} = \frac{5130 \;\; ciclos}{500 \cdot 10^6 \, Hz} = 10,26 \; mseg.$$

Dado que

R_n = (Operaciones en coma flotante · *n* elementos) / T*n*

el rendimiento para un vector de 1000 elementos sería

R_{1000} = (2 FLOP · *1000* elementos) / 5130 ciclos

R_{1000} = 0,389 FLOP/ciclo

que expresado en segundos

R_{1000} = 0,389 FLOP/ciclo · 500 MHz

R_{1000} = 194,5 MFLOPS

En lo que respecta al rendimiento de un vector de longitud infinita expresado en FLOP por ciclo

$$R_\infty = \lim_{n\to\infty}\left(\frac{2n}{T_n}\right) =$$

$$R_\infty = \lim_{n\to\infty}\left(\frac{2n}{10+\left\lceil\frac{n}{64}\right\rceil\cdot(15+55)+4n}\right) = \lim_{n\to\infty}\left(\frac{2n}{10+\left(\frac{n}{64}+1\right)\cdot 70+4n}\right) =$$

$$= \lim_{n\to\infty}\left(\frac{2n}{80+5,0937\cdot n}\right) = 0,3926 FLOP/ciclo$$

Para expresar R_∞ en FLOPS ha que multiplicar el valor anterior por la frecuencia del procesador.

Se tiene así:

$$R_\infty = 0{,}3926 \text{ FLOP/ciclo} \cdot 500 \text{ MHz}$$

$$R_\infty = 196{,}3 \text{ MFLOPS}$$

c) Dada la existencia de dos unidades de carga/almacenamiento, ya no existen riesgos estructurales entre las instrucciones LV y SV lo que permite reducir el número de convoyes a dos.

```
Convoy 1:  LV      V1, R1       // Carga de A
           LV      V2, R2       // Carga de B
           ADDV    V1, V1, V2   // A := A+B
           MULTSV  V2, F0, V2   // B := x * B
Convoy 2:  SV      R1, V1       // Almacenamiento de A
           SV      R2, V2       // Almacenamiento de B
```

La secuencia temporal de ejecución sería la mostrada en la tabla siguiente.

Convoy	Comienza	Termina	Comentario
LV V1, R1 LV V2, R2 ADDV V1, V1, V2 MULTSV V2, F0, V2	0	12+7+64=83	Latencia sencilla
SV R1, V1 SV R2, V2	83	83+12+64=159	Espera por convoy 1

Ahora, gracias a las 2 unidades de carga/almacenamiento el tiempo en ejecutar ese conjunto de seis instrucciones vectoriales para un vector de longitud 64 ha pasado a ser de 159 ciclos. El $T_{elemento}$ es 2 ciclos ya que hay dos convoyes. El $T_{arranque}$ total si se analiza la tabla anterior es:

$$T_{arranque} = T_{arranque}\ \text{LV} + T_{arranque}\ \text{MULTSV} + T_{arranque}\ \text{SV}$$

$$T_{arranque} = (12+7+12)\ \text{ciclos} = 31\ \text{ciclos}$$

Con estos valores la expresión del tiempo total de ejecución queda

$$T_n = T_{base} + \left\lceil \frac{n}{64} \right\rceil \cdot \left(T_{bucle} + T_{arranque} \right) + n \cdot T_{elemento}$$

$$T_n = 10 + \left\lceil \frac{n}{64} \right\rceil \cdot (15 + 31) + 2n = 10 + \left\lceil \frac{n}{64} \right\rceil \cdot 46 + 2n$$

que para el caso particular de $n = 1000$

$$T_{1000} = 10 + \left\lceil \frac{1000}{64} \right\rceil \cdot 46 + 2 \cdot 1000 = 10 + 16 \cdot 46 + 2000 = 2746 \ \ ciclos$$

que expresado en segundos:

$$T_{1000} = \frac{2746 \ \ ciclos}{500 \cdot 10^6 \, Hz} = 5,492 \ mseg.$$

Dado que:

Rn = (Operaciones en coma flotante · n elementos) / Tn

el rendimiento para un vector de 1000 elementos sería

R_{1000} = (2 FLOP * *1000* elementos) / 2746 ciclos

R_{1000} = 0,7283 FLOP/ciclo

que expresado en segundos:

R_{1000} = 0,7283 FLOP/ciclo·500 MHz

R_{1000} = 364,15 MFLOPS

En lo que respecta al rendimiento de un vector de longitud infinita expresado en FLOP por ciclo

$$R_\infty = \lim_{n\to\infty}\left(\frac{2n}{T_n}\right) =$$

$$R_\infty = \lim_{n\to\infty}\left(\frac{2n}{10+\left\lceil\frac{n}{64}\right\rceil\cdot(15+31)+2n}\right) = \lim_{n\to\infty}\left(\frac{2n}{10+\left(\frac{n}{64}+1\right)\cdot 46+2n}\right) =$$

$$= \lim_{n\to\infty}\left(\frac{2n}{80+2,7187\cdot n}\right) = 0,7356 FLOP/ciclo$$

Para expresar R_∞ en FLOPS ha que multiplicar el valor anterior por la frecuencia del procesador.

Se tiene así:

$$R_\infty = 0{,}7356 \text{ FLOP/ciclo} \cdot 500 \text{ MHz}$$

$$R_\infty = 367{,}8 \text{ MFLOPS}$$

d) Si se considera la posibilidad de solapamiento entre convoyes, los tiempos de arranque queden ocultos con la excepción de los del primer convoy:

Convoy	Comienza	Termina	Comentario
LV V1, R1 LV V2, R2 ADDV V1, V1, V2 MULTSV V2, F0, V2	0	12+7+64=83	Latencia sencilla
SV R1, V1 SV R2, V2	64	64+12+64=140	Solapamiento con el convoy 1

Centro Asociado Palma de Mallorca

Actividad 3.13

Tutor: Antonio Rivero Cuesta

Centro Asociado Palma de Mallorca

Actividad 3.14

Tutor: Antonio Rivero Cuesta

Centro Asociado Palma de Mallorca

Actividad 3.15

Tutor: Antonio Rivero Cuesta

Centro Asociado Palma de Mallorca

Actividad 3.16

Tutor: Antonio Rivero Cuesta

Dado el siguiente bucle:

```
for (i=0; i<256; i++)
  if (i mod 2) then
    B[i]:= B[i]+A[i];
  end if;
end for;
```

Genere el correspondiente código vectorial y calcule el tiempo de ejecución del bucle vectorizado considerando que MVL es 64, T_{bucle} de 10 ciclos y T_{base} de 5 ciclos. El coste de arranque de la unidad de suma vectorial es de 6 ciclos y el de la unidad de carga/almacenamiento de 12 ciclos y ambas se pueden encadenar. Las direcciones de `A` y `B` se encuentran ubicadas en los registros `R1` y `R2`.

```
       ADDI R10,R0,#8     ;Separación entre elementos, 8 bytes, 1 elemento de separación
       ADDI R15,R1,#2048  ;Dirección de fin del vector A
       ADDI R20,R0,#64    ;Elementos de la sección a procesar
inicio: MOVI2S VLR,R20    ;Fijamos a 64 elementos por sección con el registro especial VLR
       LVWS V1,(R1,R10)   ;Cargar elementos pares del vector A
       LVWS V2,(R2,R10)   ;Cargar elementos pares del vector B
       ADDV V3,V2,V1 ;B+A
       SVWS (R2,R10),V3   ;B:=B+A
       ADDI R1,R1,#1024   ;Avanzar 128 elementos del vector A
       ADDI R2,R2,#1024   ;Avanzar 128 elementos del vector B
       SGE  R25,R1,R15    ;Si R1>=R15 (i>=256) entonces R25:=1 si no R25:=0.
       BEQZ R25,inicio    ;Si R25=0 entonces ir a inicio
```

```
convoy1: LVWS V1,(R1,R10)
convoy2: LVWS V2,(R2,R10)
         ADDV V3,V2,V1
convoy3: SVWS (R2,R10),V3
```