1. SEGMENTACIÓN AVANZADA

El objetivo de las técnicas explicadas en esta sección es permitir *emitir múltiples instrucciones en un ciclo de reloj* ⇒ CPI (ciclos de reloj por instrucción) menor que la unidad.

Para mantener el procesador a pleno rendimiento, se debe explotar el paralelismo entre las instrucciones buscando secuencia de instrucciones no relacionadas que se puedan solapar en la segmentación. Dos instrucciones relacionadas deben estar separadas por una distancia igual a la latencia de la segmentación de la primera de las instrucciones. Se supondrá:

- Latencias de operaciones utilizadas en esta sección:

Instrucción que produce resultado	**Instrucción destino**	**Latencia en ciclos**
Op FP ALU	Otra op FP ALU	3
Op FP ALU	Almacenamiento doble	2
Carga doble	Op FP ALU	1
Carga doble	Almacenamiento doble	0

- Los saltos tienen un retardo de un ciclo de reloj.
- Las unidades funcionales están completamente segmentadas o replicadas, y que se puede emitir una operación cada ciclo de reloj.

DESENRROLLAMIENTO O EXTENSIÓN DE BUCLES.

Técnica sencilla de compilación que nos permite incrementar el paralelismo a nivel de instrucción.

Ejemplo de un bucle que suma un valor escalar a un vector en memoria. (Se supone que el array comienza en la posición de memoria 0; si no fuese así, habría que añadir una instrucción adicional entera).

```
LOOP:  LD      F0, 0(R1)      ; Carga el elemento del vector (carga doble)
                              ; empezando por el último
       ADDD    F4, F0, F2     ; Suma el escalar de F2 (suma en coma flotante de
                              ; doble precisión)
       SD      0(R1), F4      ; Almacena (almacenamiento doble) el elemento
                              ; del vector
       SUB     R1, R1, #8     ; Decrementa el puntero en 8 bytes (cada dato son
                              ; 64 bits por ser en doble precisión)
       BNEZ    R1, LOOP       ; Salta cuando no es cero
```

EJECUCIÓN SIN PLANIFICACIÓN (9 ciclos de reloj por iteración)

				Ciclo de reloj en el que se emite
LOOP:	**LD**	**F0, 0(R1)**		1
	Detención			2
	ADDD	**F4, F0, F2**		3
	Detención			4
	Detención			5
	SD	**0(R1), F4**		6
	SUB	**R1, R1, #8**		7
	BNEZ	**R1, LOOP**		8
	Detención			9

EJECUCIÓN CON PLANIFICACIÓN (6 ciclos de reloj por iteración)

				Ciclo de reloj en el que se emite
LOOP:	**LD**	**F0, 0(R1)**		1
	Detención			2
	ADDD	**F4, F0, F2**		3
	SUB	**R1, R1, #8**		4
	BNEZ	**R1, LOOP**	; Salto retardado	5
	SD	**8(R1), F4**	; Cambiado	6

Para la mayoría de los compiladores, el intercambio que se hace de la instrucción **SUB** por la instrucción **SD**, no es trivial ya que verían que la instrucción **SD** depende de la de **SUB**. El compilador tendría que ser lo suficientemente "inteligente" para realizar dicho cambio.

Con la planificación realizada, se termina un elemento del vector cada 6 ciclos de reloj, siendo el trabajo real de operación sobre el elemento del vector de tres ciclos (correspondientes a **LD**, **ADDD** y **SD**); los tres ciclos de reloj restantes se emplean en gastos de bucles (**SUB**, **BNEZ** y una detención). Para eliminar estos tres ciclos de reloj, necesitamos obtener más operaciones en el bucle $\Rightarrow$ Un sencillo esquema para incrementar el número de instrucciones entre las ejecuciones de los saltos del bucle es ***desenrrollar el bucle*** *(loop unrolling)* $\Rightarrow$ Se replica múltiples veces el cuerpo del bucle, ajustando su código de terminación y planificando entonces el bucle desenrrollado. Para lograr una planificación efectiva, utilizaremos diferentes registros para cada iteración, incrementando así el número de registros.

BUCLE DESENRROLLADO TRES VECES (CUATRO COPIAS DEL CUERPO), **SIN PLANIFICACIÓN**, Y SUPONIENDO QUE INICIALMENTE R1 ES MÚLTIPLO DE CUATRO

```
LOOP: LD      F0, 0(R1)      ; Carga el elemento del vector (carga doble)
                             ; empezando por el último
      Detención
      ADDD    F4, F0, F2     ; Suma el escalar de F2 (suma en coma flotante
                             ; de doble precisión)
      Detención
      Detención
      SD      0(R1), F4      ; Almacena (almacenamiento doble) el elemento
                             ; del vector
      LD      F6, -8(R1)     ; Se repite la secuencia de instrucciones y se han
                             ; eliminado SUB y BNEZ una vez
      Detención
      ADDD    F8, F6, F2
      Detención
      Detención
      SD      -8(R1), F8
      LD      F10, -16(R1)   ; Se repite la secuencia de instrucciones y se han
                             ; eliminado SUB y BNEZ una vez
      Detención
      ADDD    F12, F10, F2
      Detención
      Detención
      SD      -16(R1), F12
      LD      F14, -24(R1)   ; Se repite la secuencia de instrucciones y se han
                             ; eliminado SUB y BNEZ una vez
      Detención
      ADDD    F16, F14, F2
      Detención
      Detención
      SD      -24(R1), F16
      SUB     R1, R1, #32    ; Decrementa el puntero en 8*4 bytes (cada dato
                             ; son 64 bits por ser en doble precisión)
      BNEZ    R1, LOOP       ; Salta cuando no es cero
      Detención
```

Se han eliminado tres saltos y tres decrementos de R1. Se han ajustado las direcciones de las cargas y almacenamientos. Vemos que sin planificación, cada operación va seguida de una operación dependiente, se provocarán detenciones. Este bucle se ejecutará en 27 ciclos de reloj ó **6.75 ciclos de reloj por cada uno de los cuatro elementos**.

BUCLE DESENRROLLADO TRES VECES, **UNA VEZ PLANIFICADO PARA DLX**, Y SUPONIENDO QUE INICIALMENTE R1 ES MÚLTIPLO DE CUATRO

```
LOOP:  LD      F0, 0(R1)      ; Carga el elemento del vector (carga doble)
                              ; empezando por el último
       LD      F6, -8(R1)
       LD      F10, -16(R1)
       LD      F14, -24(R1)
       ADDD    F4, F0, F2     ; Suma el escalar de F2 (suma en coma flotante
                              ; de doble precisión)
       ADDD    F8, F6, F2
       ADDD    F12, F10, F2
       ADDD    F16, F14, F2
       SD      0(R1), F4      ; Almacena (almacenamiento doble) el elemento
                              ; del vector
       SD      -8(R1), F8
       SD      -16(R1), F12
       SUB     R1, R1, #32    ; Decrementa el puntero en 8*4 bytes (cada dato
                              ; son 64 bits por ser en doble precisión)
       BNEZ    R1, LOOP       ; Salta cuando no es cero
       SD      8(R1), F16     ; 8 – 32 = -24
```

Ahora el tiempo de ejecución del bucle desenrollado ha caído a un total de 14 ciclos de reloj ó **3.5 ciclos de reloj por cada uno de los cuatro elementos**.

CONCLUSIÓN: *Desenrrollar el bucle es un método sencillo pero útil para incrementar el tamaño de los fragmentos de código lineal que pueden ser planificados efectivamente, consiguiéndose reducir el número de ciclos por elemento*. Esta transformación en tiempo de compilación es similar a la que hace el Algoritmo de Tomasulo con el renombramiento de registros y la ejecución fuera de orden.

AUMENTO DEL PARALELISMO A NIVEL DE INSTRUCCIÓN CON SEGMENTACIÓN SOFTWARE Y PLANIFICACIÓN DE TRAZAS.

Desenrrollar bucles crea secuencias más largas de código, que se pueden utilizar para explotar más el paralelismo a nivel de instrucción. Para este mismo propósito se han desarrollado dos técnicas más generales:

- Segmentación software.
- Planificación de trazas.

SEGMENTACIÓN SOFTWARE.

Es una técnica para reorganizar bucles, de tal forma que, cada iteración en el código segmentado por software se haga a partir de secuencias de instrucciones escogidas en diferentes iteraciones del segmento de código original.

En un bucle, con la ***planificación*** se intercalan instrucciones de diferentes iteraciones de bucles, juntando todas las cargas, después juntando todas las sumas, después juntando todos los almacenamientos. Un bucle, ***segmentado por software***, intercala las instrucciones de diferentes iteraciones sin desenrrollar el bucle; el bucle segmentado por software contendrá una carga, una suma y un almacenamiento, cada uno de una iteración diferente; También hay código de arranque que se necesita antes que comience el bucle, así como código para concluir una vez que se complete el bucle. Esta técnica es la contrapartida software a lo que el Algoritmo de Tomasulo hace en hardware.

La principal ventaja de la ***segmentación software*** sobre el ***desenrrollamiento directo de bucles*** es que consume menos espacio de código.

El ***desenrrollamiento de bucles*** reduce gastos del bucle (código de saltos y actualización de contadores), la ***segmentación software*** reduce la porción de tiempo en la que el bucle no se ejecuta a la velocidad máxima a una única vez al comienzo y al final del bucle. El mejor rendimiento se logra utilizando ambas.

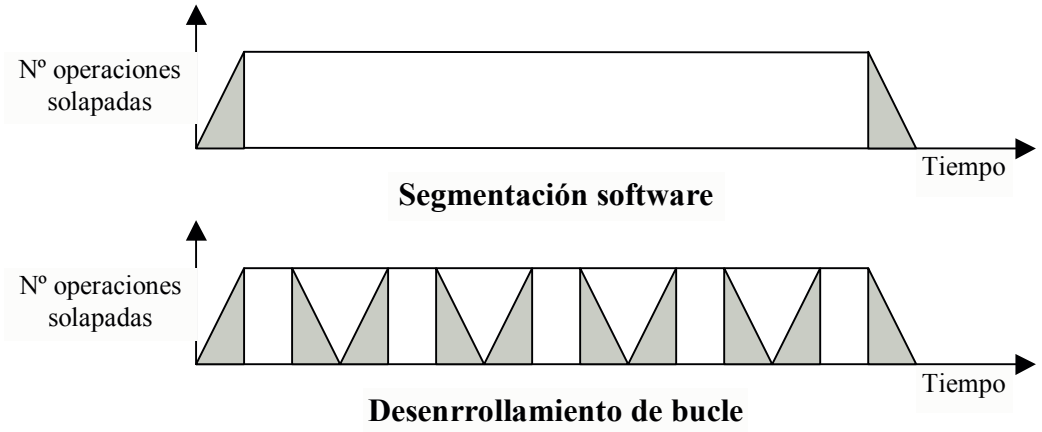


Figura 1.1. Patrón de ejecución para un bucle segmentado por software y un bucle desenrrollado. (En área sombreada no se ejecuta con solapamiento o paralelismo máximo entre instrucciones. Esto ocurre una vez al comienzo del bucle y otra al final para el bucle segmentado por software. Para el bucle desenrrollado ocurre m/n veces si el bucle tiene un total de m ejecuciones y se desenrrolla n veces).

PLANIFICACIÓN DE TRAZAS.

Esta técnica es particularmente útil para la arquitectura VLIW, para las que se desarrolló originalmente.

La ***planificación de trazas*** es una combinación de dos procesos separados:

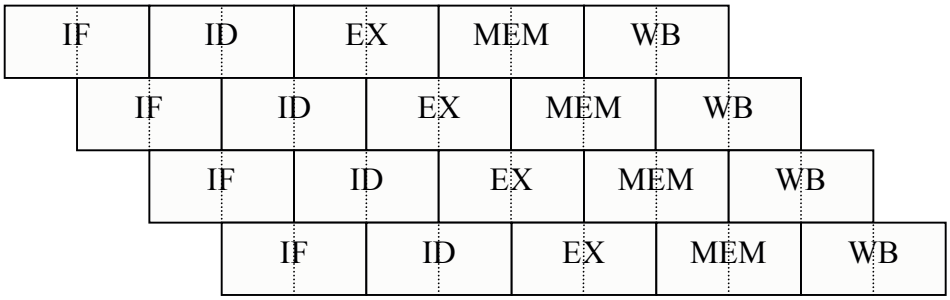
- *Selección de trazas* ⇒ Trata de encontrar la secuencia más probable de operaciones para juntarlas en un pequeño número de instrucciones; esta secuencia se denomina *traza*. (El desenrrollamiento de bucles se utiliza para generar trazas largas, ya que los saltos del bucle son efectivos con una probabilidad alta).
- *Compactación de trazas* ⇒ Trata de compactar la traza en un pequeño número de instrucciones anchas. La compactación de trazas intenta mover operaciones tan pronto como pueda en una secuencia (traza), empaquetando las operaciones en el mínimo número posible de instrucciones anchas. Dos consideraciones diferentes al compactar una traza:
    - *Las dependencias de los datos*, que fuerzan un orden parcial sobre las operaciones.
    - *Los puntos de salto*, que crean lugares a través de los cuales no se puede mover el código fácilmente. Los saltos son el principal impedimento para este proceso.

La ventaja principal de la ***planificación de trazas*** frente a las técnicas más sencillas de planificación de procesadores segmentados, es que incluye un método para mover código a través de los saltos.

Desenrrollamiento de bucles, *planificación de trazas*, y *segmentación software*, todos ayudan a intentar incrementar la cantidad de paralelismo local de las instrucciones que se puede explotar por una máquina.

TÉCNICAS HARDWARE PARA HACER EL CPI MENOR QUE LA UNIDAD.

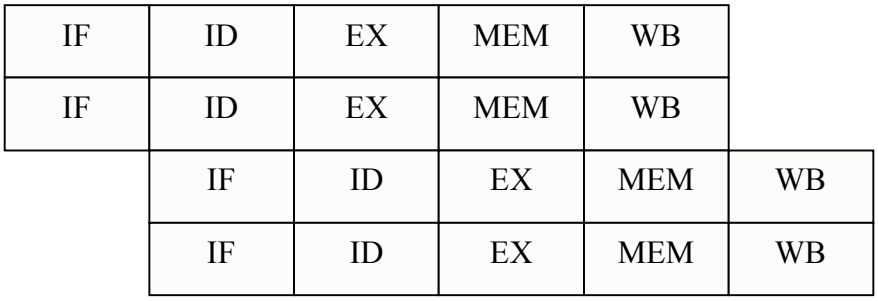
- Computadores Supersegmentados ⇒ Ejecución segmentada de instrucciones con muchas etapas.



IF ≡ Búsqueda de la instrucción
ID ≡ Decodificación de la instrucción y búsqueda de registros
EX ≡ Ejecución y cálculo de direcciones efectivas
MEM ≡ Acceso a memoria
WB ≡ Postescritura (Write Back)

Figura 1.2. Supersegmentación.

- Computadores Superescalares ⇒ Ejecución segmentada de instrucciones en varios cauces de instrucciones.



IF ≡ Búsqueda de la instrucción
ID ≡ Decodificación de la instrucción y búsqueda de registros
EX ≡ Ejecución y cálculo de direcciones efectivas
MEM ≡ Acceso a memoria
WB ≡ Postescritura (Write Back)

Figura 1.3. Superescalar con dos cauces.

ARQUITECTURA VLIW (Very Long Instruction Word – Palabra de Instrucción Muy Larga).

La arquitectura VLIW empaqueta múltiples operaciones en una instrucción muy larga. Incluye una combinación de instrucciones según la arquitectura hardware existente. Por lo tanto, podemos decir que la arquitectura VLIW implica tanto mecanismos software como mecanismos hardware.

¿Cuáles son las limitaciones de un enfoque VLIW?.

- Paralelismo limitado ⇒ Hay una cantidad limitada de paralelismo disponible en las secuencias de instrucciones.
- Recursos hardware limitados ⇒ Duplicar las unidades funcionales enteras de punto flotante es fácil y el coste aumenta linealmente. Sin embargo hay un gran incremento en el ancho de banda del fichero de registros y de la memoria. Por lo tanto, añadir unidades aritméticas solamente no ayudaría, ya que la máquina estaría limitada por el ancho de banda de memoria.
- Explosión del tamaño del código ⇒ Hay dos elementos diferentes que se combinan para aumentar sustancialmente el tamaño del código:
    - Generar suficientes operaciones en un fragmento de código lineal requiere bucles ambiciosamente desenrollados, lo que incrementa el tamaño del código.
    - Siempre que las instrucciones no sean completas, las unidades funcionales no utilizadas implican bits desaprovechados en la codificación de las instrucciones.

El reto más importantes para estas máquinas es tratar de explotar grandes cantidades de paralelismo a nivel de instrucción. Cuando el paralelismo proviene de desenrrollar bucles, el bucle original probablemente se habría ejecutado eficientemente en una máquina vectorial. No está claro que sea preferible una VLIW sobre una máquina vectorial para estas aplicaciones; los costes son similares y la máquina vectorial tiene normalmente la misma o mayor velocidad.