Arquitectura de Computadores

Tema 3-1. Planificación Estática de Instrucciones

Sumario

- Terminología
- Reordenación de instrucciones
- Desenrollamiento de bucles
- Procesadores de gran tamaño de palabra
- Desenrollamiento simbólico de bucles

Terminología

- Planificación estática de instrucciones
  - Tarea realizada por el compilador antes de ejecutar el código que implementa un determinado algoritmo
  - Esta tarea consiste en la ordenación previa de las instrucciones que ejecutará posteriormente el procesador de forma secuencial en el tiempo

Jerarquía de los niveles de abstracción del computador

Lenguaje de Programación

Programa de Lenguaje de alto Nivel

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compilador

*(incluye **Planificación Estática de Instrucciones**)*

Modelo del Programador de la Arquitectura de Repertorio de Instrucciones

Programa en Lenguaje Ensamblador

```
lw $15,0($2)
lw $16,4($2)
sw $16,0($2)
sw $15,4($2)
```

Ensamblador

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Modelo del Hardware de la Arquitectura de Repertorio de Instrucciones

Programa en Lenguaje Máquina

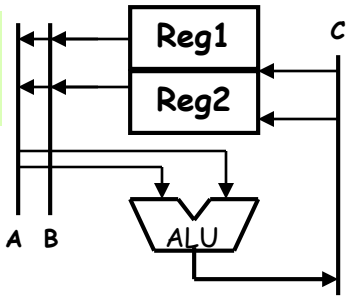
Arquitectura del Repertorio de Instrucciones

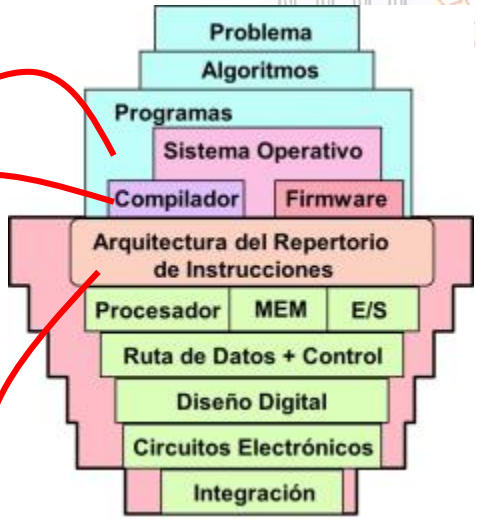
Interpretación máquina → ALUOP[0:3] <= InstReg[9:11] & MASK

Microarquitectura: Modelo Hardware de los circuitos electrónicos que forman el procesador

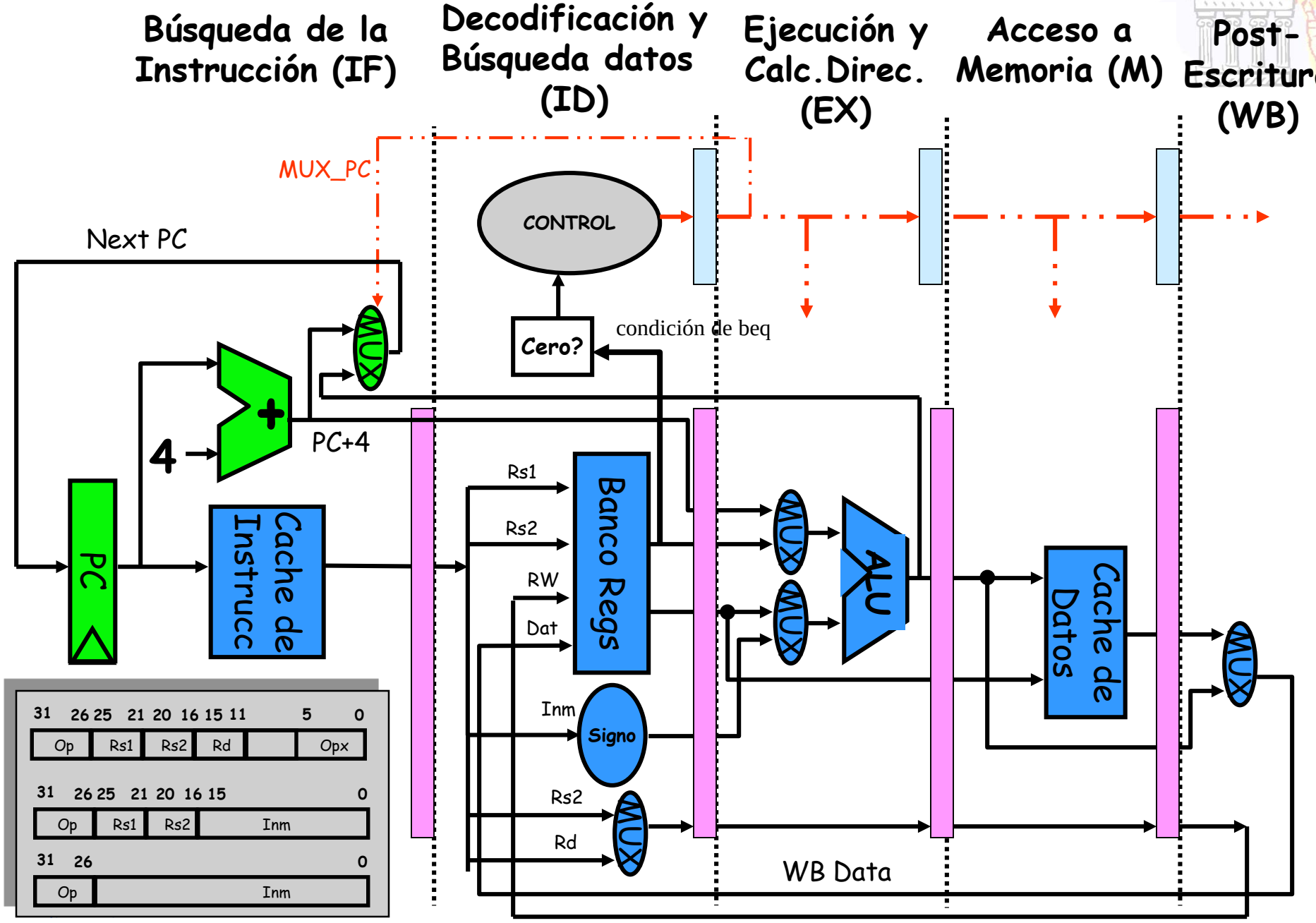
Especificación de la ruta de datos y el control

P

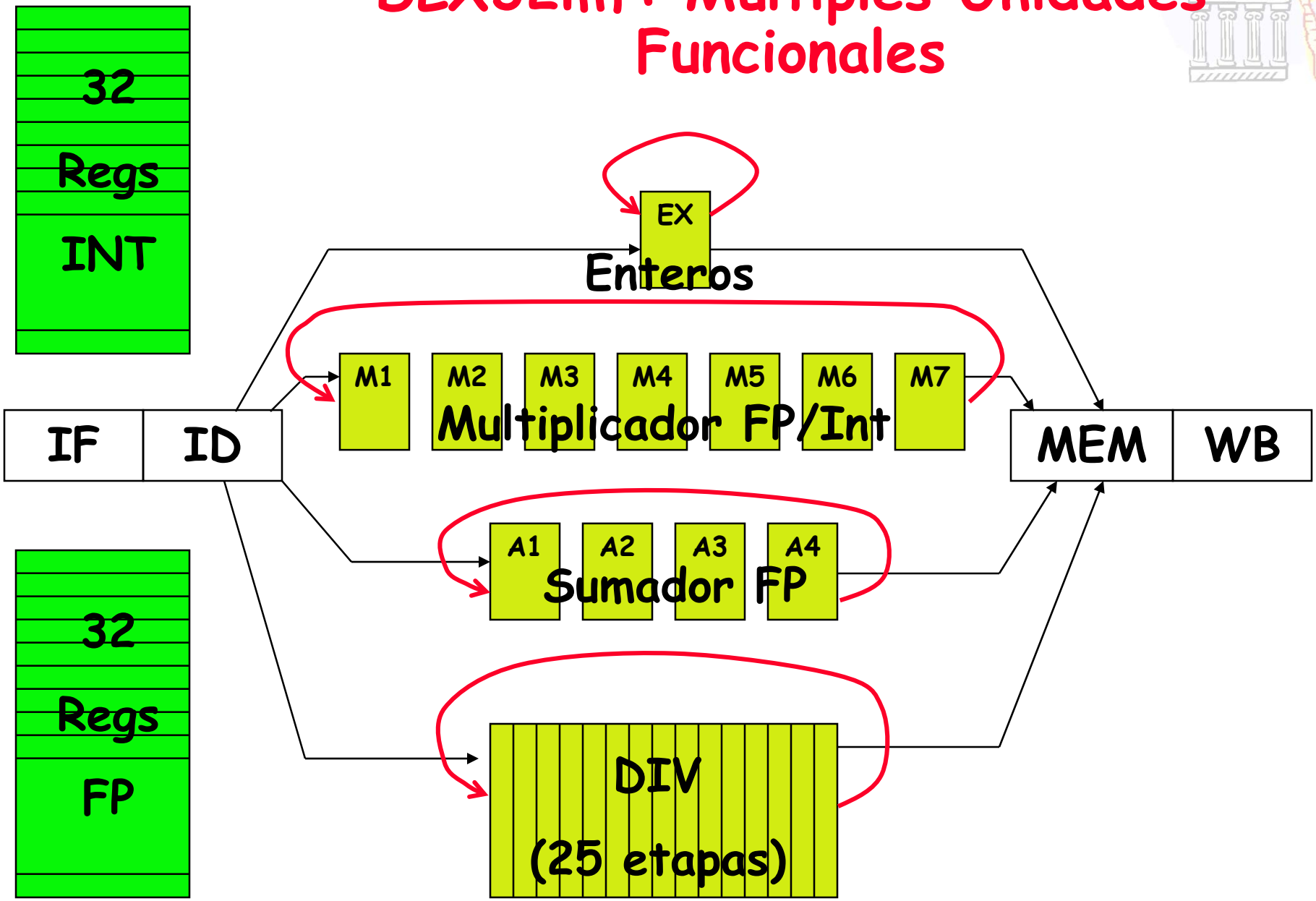




Microarquitectura de DLX32p (arquitec=DLX)



DLX32mf: Múltiples Unidades Funcionales



Instrucciones DLX que usaremos a continuación

- LD (Load Double-Precision Floating-Point)
  - `LD  Rd, offset(Rs1); LD F4, -95(R2)`
- SD (Store Double-Precision Floating-Point)
  - `SD offset(Rs1), Rd; SD -95(R2), F4`
- ADDD (Double-Precision Floating-Point Add Signed)
  - `ADDD Rd, Rs1, Rs2; ADDD F2, F0, F4`
- SUBI (Integer Substract Immediate Signed)
  - `SUBI Rd, Rs1, immediate; SUBI R1,R1,8`
- ADDI (Integer Add Immediate Signed)
  - `ADDI Rd, Rs1, immediate; ADDI R5, R2, -645`
- BNEZ (Branch On Integer NotEqual To Zero)
  - `BNEZ R8, Salto; BNEZ R1,Loop`
- MULTD (Double-Precision Floating-Point Multiply Signed)
  - `MULTD Rd, Rs1, Rs2; MULTD F2, F0, F4`

Bucles: ¿Dónde están las Penalizaciones?

```
double A[80],Cte;
for (i=79; i>=0; i--) A[i] = A[i] + Cte;
```

```
Loop:  LD    F0,0(R1)   ;F0=elemento de un vector
       ADDD  F4,F0,F2   ;suma cte en F2 y F0
       SD    0(R1),F4   ;almacena resultado
       SUBI  R1,R1,8    ;decrementa puntero 8B (DW)
       BNEZ  R1,Loop    ;salto condicional si R1!=0
       NOP              ;salto retardado
```

DLX32mf + Saltos retardados + resolución de saltos en ID

Instrucción produce resultado	*Instrucción dependiente utiliza resultado*	*Latencia (ciclos entre instrucciones dependientes)*
FP ALU op	Otra FP ALU op	3
FP ALU op	Almacenamiento DW	2
Carga DW	FP ALU op	1
Carga DW	Almacenamiento DW	0
Op enteros	Op enteros	0

Penalizaciones por Dependencias en la Máquina de Referencia (BASE)

```
ciclos
1 Loop: LD    F0,0(R1)   ; F0=elemento de un vector
2       parada           ; 1ª Dependencia RAW
3       ADDD  F4,F0,F2   ; suma cte en F2 y F0
4       parada           ; 2ª Dependencia RAW
5       parada
6       SD    0(R1),F4   ; almacena resultado
7       SUBI  R1,R1,8    ; decrementa puntero 8B (DW)
8       BNEZ  R1,Loop    ; salto condicional: R1!=0
9       parada           ; salto retardado
```

- Codificación Inicial: 9 ciclos/iteración
- Próxima Optimización: Reordenar para disminuir penalizaciones

Reordenando Instrucciones: Optimización "-O1"

```
ciclos
  1 Loop: LD    F0,0(R1)
  2       parada
  3       ADDD  F4,F0,F2
  4       SUBI  R1,R1,8
  5       BNEZ  R1,Loop    ;salto retardado
  6       SD    8(R1),F4   ;inmediato modificado
```

Reordenación Instrucciones: 6 ciclos/iteración (1.5X)

Próxima Optimización: Desenrollar 4 veces para mejorar prestaciones suponiendo que R1 es múltiplo de 4

Desenrollamiento de Bucles: Optimización "-O2"

Metodología-Paso1: copiar & pegar + actualización índices + eliminación saltos

instrucciones

```
1  Loop: LD    F0,0(R1)
2        ADDD  F4,F0,F2
3        SD    0(R1),F4
4        LD    F0,0(R1)
5        ADDD  F4,F0,F2
6        SD    0(R1),F4  ;eliminado SUBI & BNEZ
7        LD    F0,0(R1)
8        ADDD  F4,F0,F2
9        SD    0(R1),F4  ;eliminado SUBI & BNEZ
10       LD    F0,0(R1)
11       ADDD  F4,F0,F2
12       SD    0(R1),F4
13       SUBI  R1,R1,#32;alterar a 4*8
14       BNEZ  R1,LOOP
15       NOP
```

```
for (i=79; i>=0; i=i-4){
      A[i]   = A[i]   + Cte;
      A[i-1] = A[i-1] + Cte;
      A[i-2] = A[i-2] + Cte;
      A[i-3] = A[i-3] + Cte; }
```

Dependencias WAW + Memoria Ambigüa !!

"-O2"

Metodología-Paso2: Desambigüamiento de la Memoria

instrucciones

```
1  Loop: LD      F0,0(R1)
2        ADDD    F4,F0,F2
3        SD      0(R1),F4
4        LD      F0,-8(R1)
5        ADDD    F4,F0,F2
6        SD      -8(R1),F4
7        LD      F0,-16(R1)
8        ADDD    F4,F0,F2
9        SD      -16(R1),F4
10       LD      F0,-24(R1)
11       ADDD    F4,F0,F2
12       SD      -24(R1),F4
13       SUBI    R1,R1,#32
14       BNEZ    R1,LOOP
15       NOP
```

Permancen las Dependencias de Nombre WAW. ¿Se pueden eliminar?

"-O2"

Metodología-Paso3: Renombramiento de Registros

instrucciones

```
1  Loop:LD     F0,0(R1)
2       ADDD   F4,F0,F2
3       SD     0(R1),F4
4       LD     F6,-8(R1)
5       ADDD   F8,F6,F2
6       SD     -8(R1),F8
7       LD     F10,-16(R1)
8       ADDD   F12,F10,F2
9       SD     -16(R1),F12
10      LD     F14,-24(R1)
11      ADDD   F16,F14,F2
12      SD     -24(R1),F16
13      SUBI   R1,R1,#32
14      BNEZ   R1,LOOP
15      NOP
```

OBSERVACIÓN: LAS CUATRO ITERACIONES SE PODRÍAN EJECUTAR EN PARALELO

"-O2": Análisis de Prestaciones

instrucciones

```
1  Loop: LD     F0,0(R1)
2        ADDD   F4,F0,F2
3        SD     0(R1),F4
4        LD     F6,-8(R1)
5        ADDD   F8,F6,F2
6        SD     -8(R1),F8
7        LD     F10,-16(R1)
8        ADDD   F12,F10,F2
9        SD     -16(R1),F12
10       LD     F14,-24(R1)
11       ADDD   F16,F14,F2
12       SD     -24(R1),F16
13       SUBI   R1,R1,#32
14       BNEZ   R1,LOOP
15       NOP
```

Penalización 1 ciclo

Penalización 2 ciclos

Desenrrollamiento Bucles: 15 + 4 x (1+2) = 27 ciclos para 4 iteraciones ,, 6.8 ciclos/iteración (1.3X)

Próxima Optimización: Reordenar el código

Desenrrollamiento de Bucles + Reordenamiento de Código: Optimización "-O3"

```
1 Loop:LD     F0,0(R1)
2       LD     F6,-8(R1)
3       LD     F10,-16(R1)
4       LD     F14,-24(R1)
5       ADDD   F4,F0,F2
6       ADDD   F8,F6,F2
7       ADDD   F12,F10,F2
8       ADDD   F16,F14,F2
9       SD     0(R1),F4
10      SD     -8(R1),F8
11      SD     -16(R1),F12
12      SUBI   R1,R1,#32
13      BNEZ   R1,LOOP
14      SD     8(R1),F16     ; 8-32 = -24
```

- ¿Qué suposiciones se han hecho?
  - Se movió un SD después de SUBI cambiando su inmediato
  - Se movieron LDs antes de SDs: ¿correcto? ¿Memoria desambigüada?
  - ¿Cuándo es seguro realizar estos cambios por el compilador?

Desenrrollamiento Bucles + Reordenamiento: 14 ciclos para 4 iteraciones ,, 3.5 ciclos/iteración (2.6X)

Resumen de Prestaciones

- **Codificación Inicial sin optimización:** 9 ciclos/iteración
- **"-O1" Reordenación Instrucciones:** 6 ciclos/iteración (1.5X)
- **"-O2" Desenrollamiento Bucles:** 6.8 ciclos/iteración (1.3X)
- **"-O3" Desenrollamiento Bucles + Reordenamiento Instrucciones:** 3.5 ciclos/iteración (2.6X)

Modelo EPIC
(Explicitily Parallel Instruction Computer)

- El compilador o el programador pueden expresar paralelismo de instrucciones en el propio código, lo cual permite dirigir el funcionamiento del hardware para proporcionar la mayor eficiencia posible

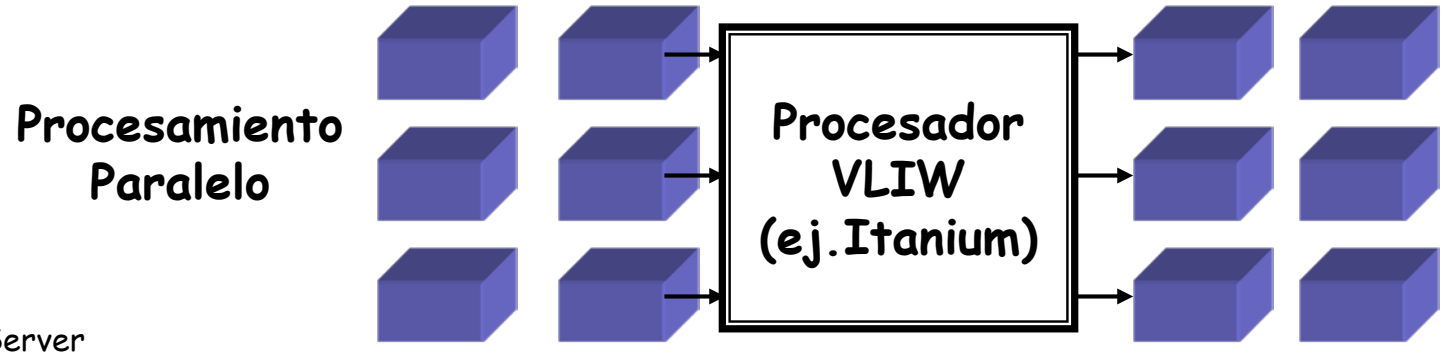
Arquitecturas VLIW
(Very Long Instrucion Word)

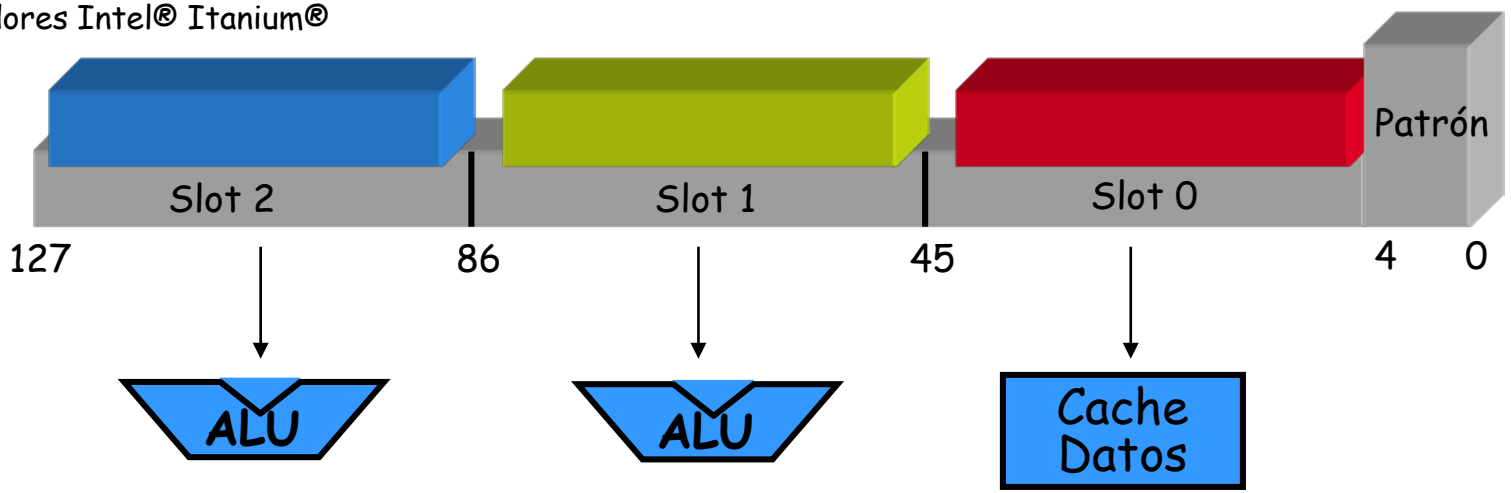
- El compilador agrupa explícitamente las instrucciones no dependientes en paquetes

Arquitecturas VLIW

- Cada paquete es enviado a ejecutar en paralelo por un hardware sencillo que no analiza dependencias

HP Integrity Superdome Server con chipset sx2000 y procesadores Intel® Itanium®





Arquitecturas VLIW

- Para el procesador VLIW se define tanto la Arquitectura ISA como las características de la ruta segmentada (operaciones en paralelo + latencia entre operaciones)
  - Decodificación simple.
  - El formato de instrucción largo tiene capacidad para muchas operaciones.
  - Por definición, todas las operaciones que el compilador incluye en una instrucción larga son independientes. Es decir, se ejecutan en paralelo.
- Mientras más ancha sea la instrucción VLIW, se obtendrán mayores prestaciones. Por ejemplo: 2 operaciones enteros, 2 operaciones FP, 2 accesos a memoria, 1 salto
  - 16 - 24 bits por campo => 7*16 = 112 bits ... 7*24 = 168 bits (ancho)

Modelo EPIC y Arquitecturas VLIW

- Tecnología Compiladores (optimización) es necesaria para extraer ILP (selección + planificacion de instrucciones) y conseguir altas prestaciones
- Objetivo: mantener todas las unidades funcionales ocupadas
- Técnicas de Compilación/Optimización:
  - "Desenrollamiento Simbólico de Bucles" (Software Pipelining)
  - "Planificación de Trazas" (Trace Schedulling). Se requieren técnicas de compilación que seleccione instrucciones a lo largo de varios saltos.

Ejemplo Académico: DLX32vliw

NUEVO Procesador "DLX32vliw"

- 1 operación enteros/salto, 2 operaciones FP, 2 accesos a memoria

Instrucción produce resultado	*Instrucción utiliza resultado*	*Latencia (ciclos)*
Operación ALU FP	Almacena DoblePal	2
Carga DoblePalab	Operación ALU FP	1

Acceso Memoria 1	*Acceso Memoria 2*	*Operación FP - 1*	*Operación FP - 2*	*Op. Int/ Salto*

- 32 Registros Int + 32 Registros FP: todos 32 bits
- Instrucción DLX32vliw: 160 bits (5 x 32)

```
1  Loop: LD     F0,0(R1)
2        ADDD   F4,F0,F2
3        SD     0(R1),F4
4        LD     F6,-8(R1)
5        ADDD   F8,F6,F2
6        SD     -8(R1),F8
7        LD     F10,-16(R1)
8        ADDD   F12,F10,F2
9        SD     -16(R1),F12
10       LD     F14,-24(R1)
11       ADDD   F16,F14,F2
12       SD     -24(R1),F16
13       LD     F18,-32(R1)
14       ADDD   F20,F18,F2
15       SD     -32(R1),F20
16       LD     F22,-40(R1)
17       ADDD   F24,F22,F2
18       SD     -40(R1),F24
19       LD     F26,-48(R1)
20       ADDD   F28,F26,F2
21       SD     -48(R1),F28
22       SUBI   R1,R1,#56
23       BNEZ   R1,LOOP
```

Optimización "-O4": Desenrollamiento de Bucles para DLX32vliw

Desenrollamiento de 7 iteraciones con desambiguamiento de la memoria y renombramiento de registros + reordenamiento (que no se ve!)

Datos en doble precisión

Optimización "-O4": Desenrollamiento de Bucles + Reordenamiento en DLX32vliw

Instrucción VLIW

Acceso Memoria 1	*Acceso Memoria 2*	*Operación FP - 1*	*Operación FP - 2*	*Op. Int/ Salto*	*Ciclo*
LD F0,0(R1)	LD F6,-8(R1)				1
LD F10,-16(R1)	LD F14,-24(R1)	≥ 1 ciclo			2
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2		3
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2		4
≥ 2 ciclos		ADDD F20,F18,F2	ADDD F24,F22,F2		5
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2			6
SD -16(R1),F12	SD -24(R1),F16				7
SD -32(R1),F20	SD -40(R1),F24			SUBI R1,R1,#56	8
SD 8(R1),F28				BNEZ R1,LOOP	9

7 iteraciones desenrolladas para evitar retardos (¿7?)

7 resultados en 9 ciclos: 1.3 ciclos/iteración (6.9X)

Promedio: 2.6 instruc/ciclo (23/9), 51% eficiencia (23/45)

Nota: Se requieren más registros VLIW (15 vs. 6 en SS)

Resumen de Prestaciones

- **Codificación Inicial sin optimización:** 9 ciclos/ iteración
- **"-O1" DLX32mf + Reordenamiento Instrucciones:** 6 ciclos/ iteración (1.5X)
- **"-O2" DLX32mf + Desenrollamiento Bucles:** 6.8 ciclos/ iteración (1.3X)
- **"-O3" DLX32mf + Desenrollamiento Bucles + Reordenamiento Instrucciones:** 3.5 ciclos/ iteración (2.6X)
- **"-O4" DLX32vliw + Desenrollamiento Bucles + Reordenamiento Instrucciones:** 1.3 ciclos/ iteración (6.9X)

Inconvenientes VLIW

- Aumento en el tamaño de código
  - Requiere mucho desenrrollamiento para rellenar los campos de las intrucciones VLIW
  - Los campos no ocupados de la instrucciones VLIW son NOP (baja eficiencia)
  - SOLUCIONES para reducir el tamaño del código:
    - » Limitar el número de inmediatos de una instrucción VLIW
    - » Comprimir instrucciones VLIW en memoria y expandirlas en IF o ID
- No análisis de dependencias de datos
  - Todas las unidades funcionales deben acabar al unísono y por eso, la ejecución de una nueva instrucción requiere que la anterior haya acabado completamente. Puede aceptarse !
  - Problemas cuando se accede a la cache que a veces puede ser acierto y otras fallo. Inaceptable !
  - SOLUCIONES:
    - » Compilador asegura la ausencia de dependencias a la hora de enviar a ejecutar en la etapa EX
    - » Hardware "arregla" las posibles desincronizaciones que surjan en la etapa EX

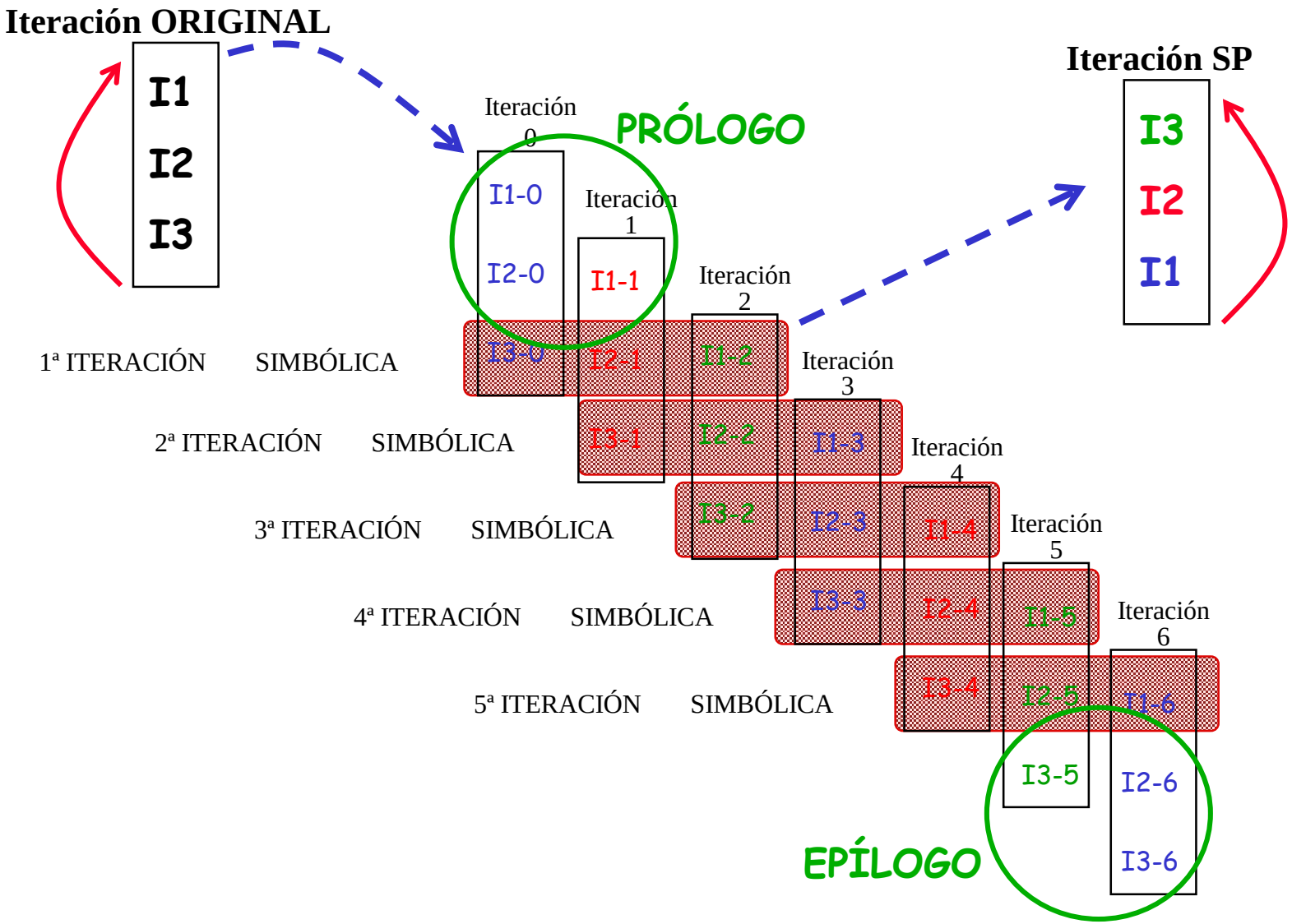
Inconvenientes VLIW

- Compatibilidad del código binario
  - Distintas versiones del hardware requieren "recompilar" el código, lo cual es una desventaja respecto a otras arquitecturas como la de los procesadores superescalares (Pentium 4)
  - SOLUCION:
    - » Traducción/Emulación de código en tiempo de ejecución
    - » IA-64: la compatibilidad de código es posible
- Aumento del número de registros en el Banco de Registros

Desenrrollamiento Simbólico de Bucles (Software Pipelining)

- **Objetivo**: aumentar el paralelismo ILP, encontrar instrucciones independientes que se puedan ejecutar en paralelo (por ejemplo, en una máquina VLIW)
- **Observación**: si las operaciones realizadas en distintas iteraciones son independientes, se consigue mayor ILP escogiendo instrucciones de iteraciones diferentes
- **Software Pipelining**: reorganiza los bucles de forma tal que cada iteración se construye a partir de instrucciones escogidas de iteraciones distintas del bucle original

Desenrrollamiento Simbólico de Bucles (Software Pipelining)



Ejemplo: Optimización "-O5" en DLX32p

i: orden del componente

ANTES: 3 veces

```
1   LD    F0,0(R1)
2   ADDD  F4,F0,F2
3   SD    0(R1),F4
4   LD    F6,-8(R1)
5   ADDD  F8,F6,F2
6   SD    -8(R1),F8
7   LD    F10,-16(R1)
8   ADDD  F12,F10,F2
9   SD    -16(R1),F12
10  SUBI  R1,R1,#24
11  BNEZ  R1,LOOP
```

DESPUES: Software Pipelined

```
1   SD    0(R1),F4 ; Almacena M[i]
2   ADDD  F4,F0,F2 ; Suma á M[i-1]
3   LD    F0,-16(R1);Carga  M[i-2]
4   SUBI  R1,R1,#8
5   BNEZ  R1,LOOP
```

Instrucción produce resultado	*Instrucción utiliza resultado*	*Latencia (ciclos)*
op FP ALU	Otra op FP ALU	3
op FP ALU	Almacena DW	2
Carga DW	op FP ALU	1

- **Desenrrollamiento Simbólico**
  - **PRO**: Maximiza la distancia resultado-uso
  - **PRO**: Menor cantidad de código que desenrrollamiento NORMAL
  - **CON**: Requiere ejecutar instrucciones Prólogo y Epílogo

Ejemplo: Optimización "-O5" en DLX32p
Modificaciones

Código fuente en C

```
double A[80],Cte;
for (i=79; i>=0; i--) A[i] = A[i] + Cte;
```

Prólogo

```
ADDI    R1,R0,79×8    ; inicialización "i=79"
LD      F0,0(R1)      ; Carga A[79]
ADDD    F4,F0,F2      ; Suma "A[79]+Cte"
LD      F0,-8(R1)     ; Carga A[78]
SUBI    R1,R1,#8      ; i = i-1 = 78
```

78 iteraciones

5 ciclos/iteración

```
LOOP:   SD      8(R1),F4      ; Almacena A[i]
        SUBI    R1,R1,#8      ; i=i-1
        ADDD    F4,F0,F2      ; Suma "A[i-1]+Cte"
        BNEZ    R1,LOOP
        LD      F0,0(R1)      ; Carga A[i-2]
```

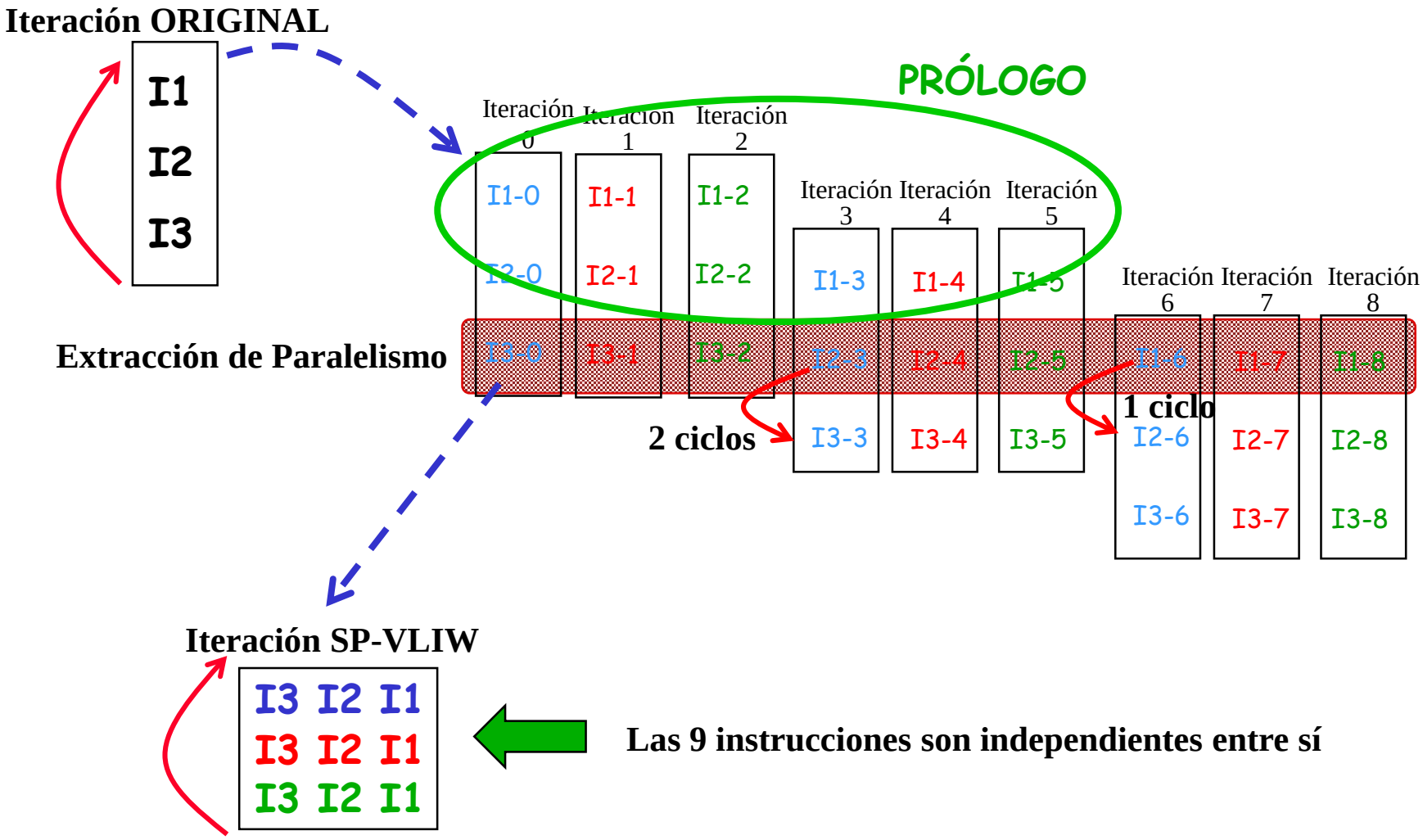
Epílogo

```
SD      8(R1),F4    ; R1=0; Almacena A[1]
ADDD    F12,F0,F2   ; Suma A[0]+Cte
SD      0(R1),F12   ; Almacena A[0]
```

Resumen de Prestaciones

- **Codificación Inicial sin optimización:** 9 ciclo/iteración
- **"-O1" Reordenamiento Instrucciones:** 6 ciclo/iteración (1.5X)
- **"-O2" Desenrollamiento Bucles:** 6.8 ciclo/iteración (1.3X)
- **"-O3" Desenrollamiento Bucles + Reordenamiento Instrucciones:** 3.5 ciclo/iteración (2.6X)
- **"-O4" DLX32vliw + Desenrollamiento Bucles + Reordenamiento Instrucciones:** 1.3 ciclo/iteración (6.9X)
- **"-O5" DLX32p + Software Pipelining + Reordenación:** 5 ciclo/iteración (1.8X)

Desenrrollamiento Simbólico de Bucles (Software Pipelining, SP)



Cálculo de número de desenrollamiento para extraer el paralelismo de instrucciones

```
LOOP:
LD   F0,0(R1)
ADDD F4,F0,F2
SD   0(R1),F4
```

Potencial de paralelismo de instrucciones para VLIW: 3 instrucciones (2 accesos a memoria + 1 operación FP)

Es necesario buscar 3 instrucciones independientes con desenrrollamiento de 3 iteraciones

Máxima penalización: 2 ciclos entre operación ADDD y almacenamiento SD

Es necesario llenar 2 instrucciones VLIW después de una que incluye instrucciones que generan dependencas

Total: 3 instrucciones VLIW con 3 instrucciones en cada una de ellas, en total 9 iteraciones (3 x 3)

Optimización "-O6" en DLX32vliw

ANTES: 9 veces (3 x 3)

```
LOOP:
M[i]   1  LD    F0,0(R1)
       2  ADDD  F4,F0,F2
       3  SD    0(R1),F4
M[i-1] 4  LD    F6,-8(R1)
       5  ADDD  F8,F6,F2
       6  SD    -8(R1),F8
M[i-2] 7  LD    F10,-16(R1)
       8  ADDD  F12,F10,F2
       9  SD    -16(R1),F12
M[i-3] 10 LD    F0,-24(R1)
       11 ADDD  F4,F0,F2
       12 SD    -24(R1),F4
M[i-4] 13 LD    F6,-32(R1)
       14 ADDD  F8,F6,F2
       15 SD    -32(R1),F8
M[i-5] 16 LD    F10,-40(R1)
       17 ADDD  F12,F10,F2
       18 SD    -40(R1),F12
M[i-6] 19 LD    F0,-48(R1)
       20 ADDD  F4,F0,F2
       21 SD    -48(R1),F4
M[i-7] 22 LD    F6,-56(R1)
       23 ADDD  F8,F6,F2
       24 SD    -56(R1),F8
M[i-8] 25 LD    F10,-64(R1)
       26 ADDD  F12,F10,F2
       27 SD    -64(R1),F12
       28 SUBI  R1,R1,#72
       29 BNEZ  R1,LOOP
```

DESPUES: Software Pipelined

```
LOOP:
1  SD    0(R1),F4 ;     Almacena A[i]
2  ADDD  F4,F0,F2 ;     Suma a A[i-3]
3  LD    F0,-48(R1);    Carga A[i-6]

4  SD    -8(R1),F8 ;    Almacena A[i-1]
5  ADDD  F8,F6,F2 ;     Suma a A[i-4]
6  LD    F6,-56(R1);    Carga A[i-7]

7  SD    -16(R1),F12 ;  Almacena A[i-2]
8  ADDD  F12,F10,F2 ;   Suma a A[i-5]
9  LD    F10,-64(R1);   Carga A[i-8]

10 SUBI  R1,R1,#24
11 BNEZ  R1,LOOP
```

Software Pipelining en 9 iteraciones del bucle original

- En cada iteración:
  - » Almacena M[R1], M[R1-8], M[R1-16] (iteraciones i,i+1,i+2)
  - » Calcula M[R1-24], M[R1-32], M[R1-40] (iteraciones i+3,i+4,i+5)
  - » Carga M[R1-48], M[R1-56], M[R1-64] (iteraciones i+6,i+7,i+8)

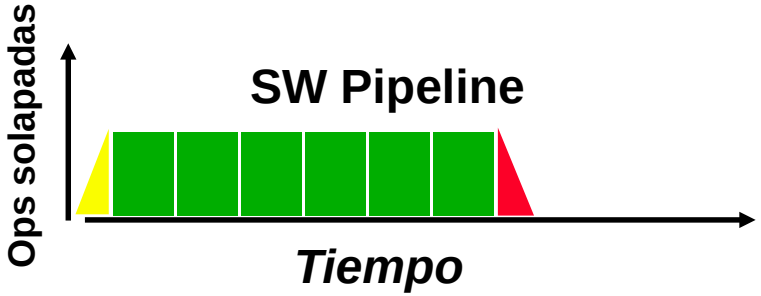
"-O6" Software Pipelining + Desenrollamiento de Bucles en VLIW

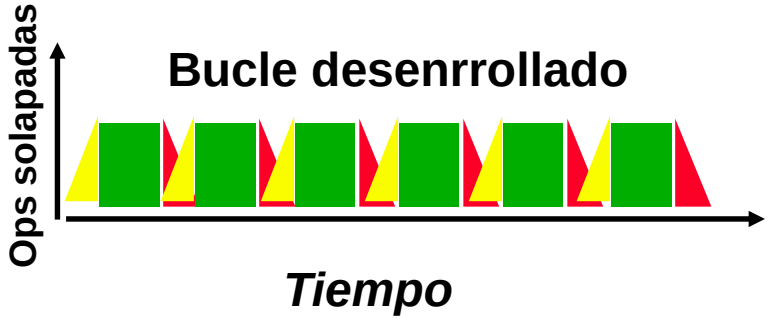
Instrucción VLIW

Acceso Memoria 1	*Acceso Memoria 2*	*Operación FP - 1*	*Operación FP - 2*	*Int/ salto*	*Ciclo*
LD F0,-48(R1)	SD 0(R1),F4	ADDD F4,F0,F2			1
LD F6,-56(R1)	SD -8(R1),F8	ADDD F8,F6,F2		SUBI R1,R1,#24	2
LD F10,-40(R1)	SD 8(R1),F12	ADDD F12,F10,F2		BNEZ R1,LOOP	3

- 9 resultados en 9 ciclos, 1 ciclo por iteración (9X)
- Promedio: 3.6 ops por ciclo (11/3), 73% eficiencia (11/15)

Note: Requiere menos registros con SP que DB (sólo 7 registros FP "-O6", frente a los 15 de "-O4")

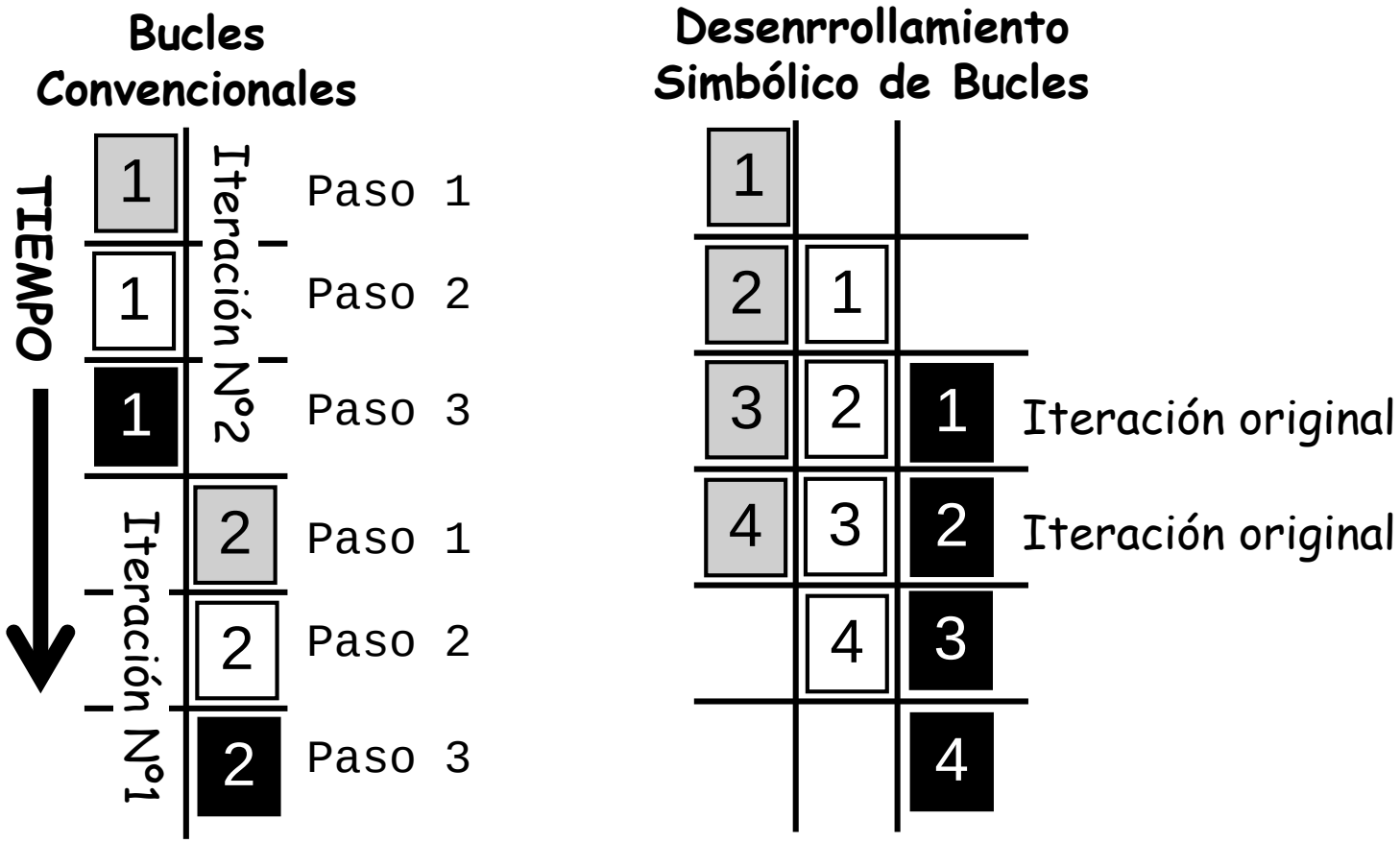




Resumen de Prestaciones

- Codificación Inicial sin optimización: 9 ciclo/iteración
- "-O1" Reordenamiento Instrucciones: 6 ciclo/iteración (1.5X)
- "-O2" Desenrollamiento Bucles: 6.8 ciclo/iteración (1.3X)
- "-O3" Desenrollamiento Bucles + Reordenamiento Instrucciones: 3.5 ciclo/iteración (2.6X)
- "-O4" DLX32vliw + Desenrollamiento Bucles + Reordenamiento Instrucciones: 1.3 ciclo/iteración (6.9X)
- "-O5" DLX32p + Software Pipelining + Reordenación: 5 ciclo/iteración (1.8X)
- "-O6" DLX32vliw + Software Pipelining + Desenrollamiento Bucles : 1 ciclo/iteración (9X)

Software Pipelining en IA-64



Recursos Propios de IA-64 implementados en HW para SP

- Instrucciones especiales de saltos condicionales
- Registros LC (loop count) y EC (epilogue count)
- Registros rotantes

Desenrrollamiento Simbólico de Bucles en IA-64

```
for(i=0; i<n; i++) x[i]=y[i]+1;
```

```
loop
 (p16) ld1 r32  = [r12], 1
 (p17) add r34  = 1, r33
 (p18) st1 [r13]= r35, 1
       br.cond loop
```

Software Pipelining

```
// Inicialización
        mov pr.rot     = 0     // Inicializa los registros predicados rotan
        cmp.eq p16,p0  = r0,r0 // p16=1
        mov ar.lc      = 4     // Inicializa contador de bucle a n-1
        mov ar.ec      = 3     // Inicializa epílogo a 3
// Bucle
loop:
 (p16) ld1 r32  = [r12], 1    // Paso 1: carga X
 (p17) add r34  = 1, r33      // Paso 2: Y=X+1
 (p18) st1 [r13]= r35, 1      // Paso 3: almacena Y
        br.cond loop          // Salto atrás
```

Software Pipelining: Estado inicial después del ejecutar el código de inicialización

Se suponen 5 iteraciones y latencias de 1 ciclo de reloj

```
loop:
 (p16) ld1 r32  = [r12], 1
 (p17) add r34  = 1, r33
 (p18) st1 [r13]= r35, 1
       br.ctop loop
```

Registros Globales

r32
r33
r34
r35
r36
r37
r38

Registros Predicado

1	0	0

p16 p17 p18

LC

4

EC

3

Primera iteración en la etapa Prólogo: sólo se ejecuta "ld1"

```
loop:
 (p16) ld1 r32  = [r12], 1
 (p17) add r34  = 1, r33
 (p18) st1 [r13]= r35, 1
       br.ctop loop
```

Registros Globales

r32
r33=X1
r34
r35
r36
r37
r38

Registros Predicado

1	1	0
p16	p17	p18

LC

3

EC

3

Segunda iteración en la etapa Prólogo: sólo se ejecuta "ld1" y "add"

```
loop:
 (p16) ld1 r32  = [r12], 1
 (p17) add r34  = 1, r33
 (p18) st1 [r13]= r35, 1
       br.ctop loop
```

Registros Globales

r32
r33=X2
r34=X1
r35=Y1
r36
r37
r38

Registros Predicado

1	1	1

p16 p17 p18

LC

2

EC

3

Primera iteración completa

```
loop:
 (p16) ld1 r32  = [r12], 1
 (p17) add r34  = 1, r33
 (p18) st1 [r13]= r35, 1
       br.ctop loop
```

Registros Globales

r32
r33=X3
r34=X2
r35=Y2
r36=Y1
r37
r38

Registros Predicado

1	1	1

p16 p17 p18

LC

1

EC

3

Segunda iteración completa

```
loop:
 (p16) ld1 r32  = [r12], 1
 (p17) add r34  = 1, r33
 (p18) st1 [r13]= r35, 1
       br.ctop loop
```

Registros Globales

r32
r33=X4
r34=X3
r35=Y3
r36=Y2
r37=Y1
r38

Registros Predicado

1	1	1

p16 p17 p18

LC

0

EC

3

Tercera iteración completa

```
loop:
 (p16) ld1 r32  = [r12], 1
 (p17) add r34  = 1, r33
 (p18) st1 [r13]= r35, 1
       br.ctop loop
```

Registros Globales

r32
r33=X5
r34=X4
r35=Y4
r36=Y3
r37=Y2
r38=Y1

Registros Predicado

0	1	1

p16 p17 p18

LC

0

EC

2

Penúltima iteración de la etapa Epílogo

```
loop:
 (p16) ld1 r32  = [r12], 1
 (p17) add r34  = 1, r33
 (p18) st1 [r13]= r35, 1
       br.ctop loop
```

Registros Globales

r32
r33
r34=X5
r35=Y5
r36=Y4
r37=Y3
r38=Y2

Registros Predicado

0	0	1

p16 p17 p18

LC

0

EC

1

Última iteración de la etapa Epílogo

```
loop:
 (p16) ld1 r32  = [r12], 1
 (p17) add r34  = 1, r33
 (p18) st1 [r13]= r35, 1
       br.ctop loop
```

Registros Globales

r32
r33
r34
r35=X5
r36=Y5
r37=Y4
r38=Y3

Registros Predicado

0	0	0

p16 p17 p18

LC

0

EC

0