TEMA 3

PLANIFICACIÓN O REORDENAMI
(SCHEDULING) DE INSTRUCCIO

ÍNDICE

3.1. CONCEPTOS FUNDAMENT

- Definición de Planificación, secuenciamiento o reordenamien*ng*): cambiar el orden (reordenar) instrucciones para evitar bloqueos.
- Objetivo: eliminar riesgos (y reducir bloqueos). Típicamente se pu los bloqueos de datos. Bq de control se vieron 2.8,2.9. Bq estructurales hw
  - Recordar: dependencias WAR, WAW ficticias, se pueden evitar ado de registros. Luego es importante tener un conjunto amplio para renombrar y reordenar (ventaja RISC).
  - Pero dependencias reales RAW no pueden eliminarse, so a algoritmo (habrá algoritmos con más RAW y otros con menos no es tema relativo a la Arquitectura sino a la Algorítmica.
- **Conclusión**: realmente un programa podría escribirse atendiendo e te a las dependencias reales, sin ser necesarios en un primer momento lo
- Claro que cuando se diseña un procesador como máquina sír cen obligatoriamente los registros como el estado de la máquina alma ada periodo.

  ⇒ Objetivo: intentar que el programa se ejecute como el grafo sig

Renombrado. Ejemplo

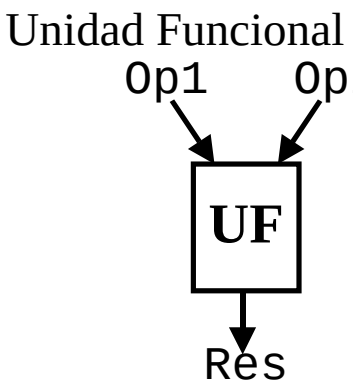
```
MULTF   F1, F2, F3
DIVF    F2, F1, F4
ADDF    F1, F2, F3
SF      0(R1), F1
```

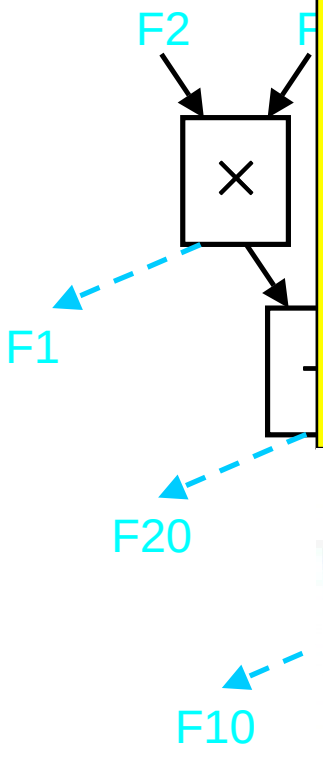
```
MULTF
DIVF
ADDF
SF
```

"Ensamblador" con grafo de s

Ensamblador clásico

```
MULTF   F1, F2, F3
DIVF    F20, F1, F4
ADDF    F10, F20, F3
SF      0(R1), F10
```



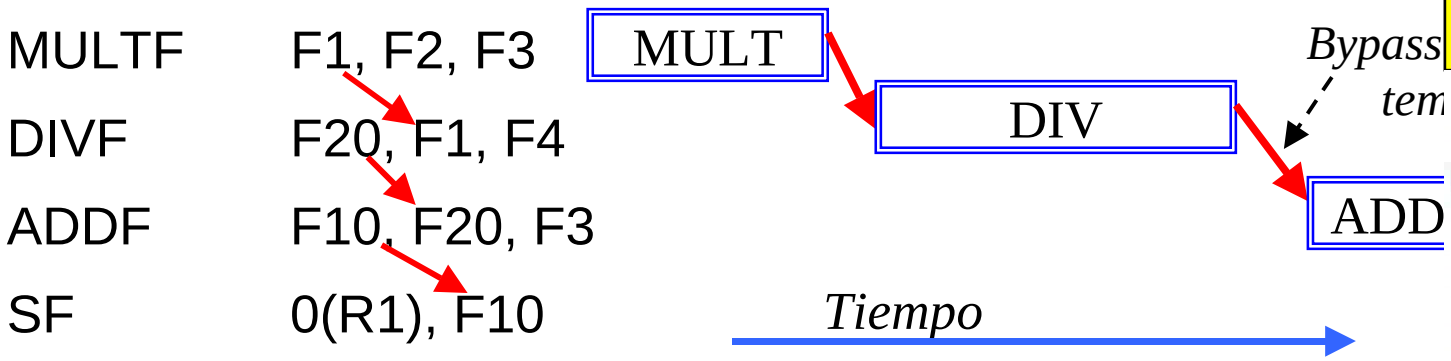


Los registros (celeste) son prescindibles

F1

Límite de las prestaciones

- En encadenamiento básico, un bloqueo por cualquier deper ...uea) totalmente la cadena (máquina): aumenta el CPI
- Luego, reordenando podemos eliminar c. bq. (típicamente de datos).
- Pero existe un límite (en la aceleración alcanzable...):
  - o ⇒ límite de CPI (*data flow limit* o límite de flujo de datos)
- Ejemplo: Supongamos que las fases "accesorias" de una instrucci ...e la instrucción se busca, decodifica, etc.) duran cero ciclos, y que únic ...un tiempo apreciable las operaciones (entre ellas el acceso a la memoria
  - o Realmente los procesadores actuales han llegado prácticamente a
- El grafo dependencias se convertiría en un cronograma (sólo fases E



Planificación instrucciones FP - INT

- Generalmente la duración de las unidades funcionales U.F. FP es ma…bq… datos: problema clásico de los años '60 (código científico, su… …ores electrónica discreta).
- Típicamente un bucle (más del 90% t. de ejec.) de código FP se … en instr. INT e instr. FP. Cada grupo de instrucciones tendría su grafo …cias por separado; cada grupo usa registros diferentes y propios (except… …tore que mezclan un reg. INT con otro FP) ⇒ no hay dependencias entre …s.

⇒ Reordenación estática de instr. INT con FP muy evide…

LF	F2, 0(R1)	LF	F2, 0(R
MULTF	F1, F2, F3	MULTF	F1, F2,
DIVF	F2, F1, F4	ADDI	R1, R1,
ADDF	F1, F2, F3	DIVF	F2, F1,
SF	0(R1), F1	SLT	R2, R1,
----------	----------	ADDF	F1, F2,
ADDI	R1, R1, 4	SF	-4(R1),
SLT	R2, R1, R7	BNEZ	R2, buc
BNEZ	R2, bucle		

• Planificación estática

- PERO PROBLEMA: el compilador debe conocer la arquitectura (la… las operaciones).
- ¡Y esto puede cambiar con la versión del procesador! El progr…ado estáticamente que hoy es rápido, mañana puede ser lento. Lu…que recompilar todos los programas para conseguir el ajuste ideal ("sinte…ntre código y procesador (técnicas dinámicas no sufren de este defecto)…ción entre aplicación (en este caso, su rendimiento), compilador y hw. no…rio.
- Lo que siempre es más ventajoso en el uso de técnicas estáticas es…vare puede simplificarse (o usar estos recursos –transistores- para otras p…más cachés, más registros, más unidades funcionales –ALU's-, etc.).

• Planificación dinámica

- Veremos que la reordenación se puede hacer dinámicamente (3.3).
- *IDEA*: la CPU puede anotar en tiempo de ejecución las depen…van surgiendo sin bloquear la máquina (ya no se ve tan clara la ejecució… La instr. que necesita un registro (debido a una RAW) se espera, sin …ndo los operandos de tal instrucción estén disponibles realiza su fase EX…ras instrucciones (sin RAW) seguirán adelante con todas sus fases.

3.2, 3.4. PLANIFICACIÓN ESTÁTICA. DESEN[illegible]O DE BUCLES.

- La planificación estática está limitada por los saltos (es más difí[illegible]" y existen pocos casos en que se pueda).
- Siempre se puede apostar por un comportamiento T/NT en tiempo[illegible]ción (predicción estática) y reordenar usando también instrucciones de [illegible]cha (*especulación software*). Pero esto conlleva a complicaciones y en g[illegible]muy eficaz (se verá en ASP2).
- NOTA: la predicción dinámica es mucho más certera que la estática[illegible]esto la planificación dinámica puede sacar más partido (las reordenan[illegible]rán más veces). La *especulación hardware* es más poderosa.
- Sin embargo, el desenrollado de bucles (*loop-unrolling*) va algo [illegible]e la simple reordenación, pues elimina instrucciones (de *overhead* o sol[illegible]n los cual elimina también ciertas dependencias. Por tanto, en principio s[illegible]las técnicas dinámicas, con lo que respecta a la eliminación de esas instr[illegible]
- Veremos que con planificación dinámica también se hace [illegible]de desenrollado (simplemente usando el código original).

Resumen de desenrollado (ver práctica 4)

Suponemos bucle paralelizable (iteraciones independientes).

Ejemplo:
```
double s, x[M] , y[M];  int i;
for (i=0 ; i<M ; i++ )  y[i]= x[i] * s ;
```

Duración U.F. MULT (la dependencia que dará más c. bq.) = **4 ciclos**.

Nº c. bq. entre MULT y Store = Duración – 1 (–1) = 4 –1 –1 = 2

(el último –1 porque el Store de DLX usa el dato [illegible]M)

Como regla general, necesito desenrollar **1+(Nº c. bq. en la dependen[illegible]a)**=
3+1 iteraciones. Si Nº c.bq.=0 habría que desenrollar 1 iter., es dec[illegible]ucle como está. Nos concentramos en un desenrollado sistemático en tal de[illegible]

```
for (i=0 ; i<M%3 ; i++)  y[i]= x[i] * s; //star-up[illegible]e)
for (    ; i<M ; i+=3 ) {
   y[i+0]= x[i+0] * s ;
   y[i+1]= x[i+1] * s ;
   y[i+2]= x[i+2] * s ;
 }
```


- En ensamblador, se distingue entre **instrucciones útiles** (de cómpu so a memoria de los datos y resultados) e instrucciones de sobrecarga (stas últimas son las que regulan el bucle (por estar escrito en le tivo imperativo) y pueden eliminarse si se desenrollara "infinitas" veces . en un lenguaje orientado a vectores, no existiría tal bucle.

```
bucle_orig:
    LD     F2, 0(R1)
    MULTD  F4, F2, F24  ; F24 contiene el valor de s
    SD     (R3)0, F4
    ;-----------------------
    ADDI   R1, R1, 8
    ADDI   R3, R3, 8
    SLTI   R7, R1, fin_array_x; constante apunta al final
    BNEZ   R7, bucle_orig
```

Desenrollado sistemático

1. Se despliegan varias iteraciones

```
bucle_intermedio1:
    LD     F2, 0(R1)
    MULTD  F4, F2, F24 ; F24 contiene el valor de s
    SD     (R3)0, F4
    ADDI   R1, R1, 8
    ADDI   R3, R3, 8
    SLTI   R7, R1, fin_array_x;
```


```
      ; el salto sobra, luego la instr. anterior de comp
    LD     F2, 0(R1)
    MULTD  F4, F2, F24
    SD     (R3)0, F4
    ADDI   R1, R1, 8
    ADDI   R3, R3, 8
    SLTI   R7, R1, fin_array_x;
      ; el salto sobra, luego la instr. anterior de comp

    LD     F2, 0(R1)
    MULTD  F4, F2, F24
    SD     (R3)0, F4
    ADDI   R1, R1, 8
    ADDI   R3, R3, 8
    SLTI   R7, R1, fin_array_x
    BNEZ   R7, bucle_intermedio1
```

2. Se eliminan las instrucciones inútiles, modificando direccionar ...tros operandos inmediatos. Notar como ciertas dependencias reales en ...nes *overhead* van desapareciendo. Y que la cantidad de instrucciones ...se eliminan guarda una proporción directa con el número de iteraciones ...s.

```
 bucle_intermedio2:
    LD     F2, 0(R1)
    MULTD  F4, F2, F24 ; F24 contiene el valor de s
    SD     (R3)0, F4
```

```
LD     F2, 8(R1)
MULTD  F4, F2, F24
SD     (R3)8, F4

LD     F2, 16(R1)
MULTD  F4, F2, F24
SD     (R3)16, F4
;----------------------
ADDI   R1, R1, 8*3
ADDI   R3, R3, 8*3
SLTI   R7, R1, fin_array_x
BNEZ   R7, bucle_intermedio2
```

3. Se renombran registros para evitar dependencias ficticias WAR, [illegible] mos notación con primas (‘) por simplicidad (el DLX utilizaría registros [illegible] 30)

```
bucle_intermedio3:
  LD     F2, 0(R1)
  MULTD  F4, F2, F24 ; F24 contiene el valor de s
  SD     (R3)0, F4

  LD     F2’, 8(R1)
  MULTD  F4’, F2’, F24
  SD     (R3)8, F4’
```


```
    LD     F2'', 16(R1)
    MULTD  F4'', F2'', F24
    SD     (R3)16, F4''
    ;----------------------
    ADDI   R1, R1, 8*3
    ADDI   R3, R3, 8*3
    SLTI   R7, R1, fin_array_x
    BNEZ   R7, bucle_intermedio3
```

4. Se entrelazan sistemáticamente las instrucciones de las distintas itera

```
  bucle_desenrollado:
    LD     F2, 0(R1)
    LD     F2', 8(R1)
    LD     F2'', 16(R1)

    MULTD  F4, F2, F24 ;   F24 contiene el valor de s
    MULTD  F4', F2', F24
    MULTD  F4'', F2'', F24

    SD     (R3)0, F4
    SD     (R3)8, F4'
    SD     (R3)16, F4''
    ;----------------------
    ADDI   R1, R1, 8*3
```

```
    ADDI   R3, R3, 8*3
    SLTI   R7, R1, fin_array_x
    BNEZ   R7, bucle_desenrollado
```

Ahora se ve que las dependencias más largas no producen c.bq. (de [illegible] yan desenrollado 3 iteraciones)

5. Opcionalmente se reordenan las instrucciones para reducir algún c. [illegible] a e[illegible] de la dependencia más larga). Típicamente se mezclan las instr. FP c[illegible]

```
bucle_desenrollado_opc:
    LD     F2, 0(R1)
    LD     F2', 8(R1)
    LD     F2'', 16(R1)
    MULTD  F4, F2, F24 ;   F24 contiene el valor de s
    MULTD  F4', F2', F24
    MULTD  F4'', F2'', F24
    ADDI   R1, R1, 8*3
    SD     (R3)0, F4
    SD     (R3)8, F4'
    SD     (R3)16, F4''
    SLTI   R7, R1, fin_array_x ;  Quito c.bq. entre SLTI y
    ADDI   R3, R3, 8*3
    BNEZ   R7, bucle_desenrollado_opc
```

Nota: gracias a utilizar ADDI R1,... de "colchón" entre la dependen[illegible], e[illegible] nº de iter. desenrolladas podría haber sido menor.

- **Aceleración**: ¡cuidado! al reducir el número de instrucciones, el C… …ena medida de rendimiento. Usar N_{instr}*CPI o tb **Ciclos / elem. proce… …ay**.

	Bucle original	Bucle desenrollado	Desen… iter… …as" …)
Instr./elem array	(3 instr útiles + 4 instr overhead) / elem	(3*3 instr útiles + 4 instr overhead) / 3 elem array = = (3 instr útiles + 4/3 instr overhead) /elem	(3*K ins… overhea… = (K→∞ = 3 instr…
F_{saltos}	1/7 = 14.3%	1/13 = 7.7%	0%
Estimación ciclos / elem array	7 instr+(1 LD-MULTD + 2 MULTD-SD + 1 SLTI-BNEZ) c.bq. datos + casi 1 c.bq. control = = casi 12 Ciclos	(13 instr + casi 1 c.bq. control) / 3 elem array = = casi 14/3 ciclos / elem = = 4.66^	(3*K ins… control) = 3 cicl…
Tam código estático (bytes)	7 * 4	13 * 4	3 * …
Notas	Tal vez surja un bloqueo estructural. Se pueden eliminar algunos c.bq. datos reordenando	Tal vez surjan bq estr.; Desprecio iteraciones de arranque	Prestaci… Tamaño estático … fallos de…

- *Ejercicio*: eliminar c.bq. control con diversas técnicas de saltos retra r en uso de BTB.
- *Ejercicio*: se podrían haber eliminado algunos c.bq. datos reordenand
- Prestaciones máximas si desenrollo “infinitas” veces

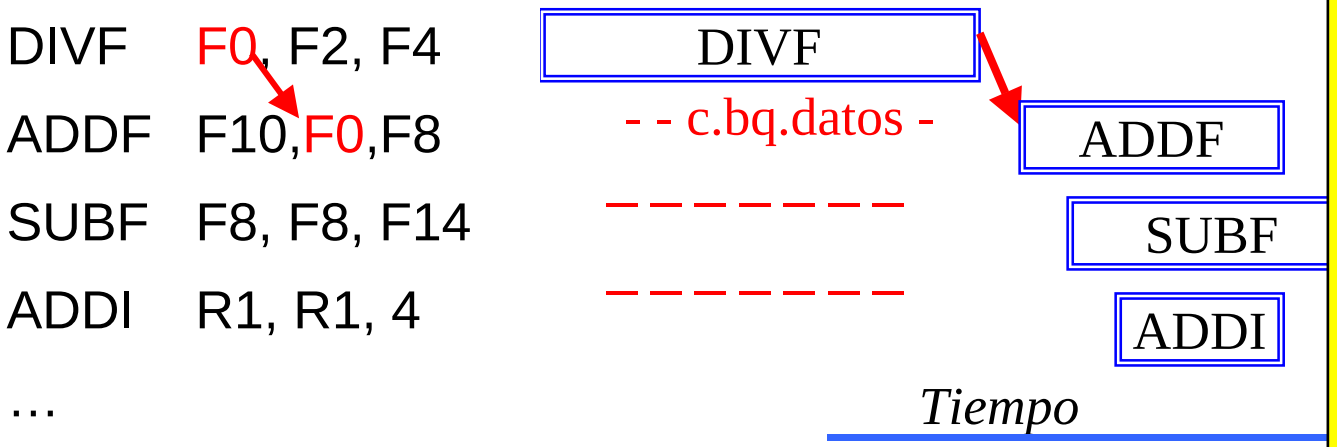
```
x[0]= x[0] * s ;
x[1]= x[1] * s ;
…
x[M-1]= x[M-1] * s ;
```

- CONCLUSIONES:
  - Se han reducido las instr. overhead (estamos “rescribiendo” el e de leng. iterativo para hacerlo más rápido)
  - Se aumenta las posibilidades de planif. estática, en cuanto se r porcentaje de saltos (por cada salto, hay más instr. de otro tipo
  - Se necesitan más registros
  - Se ha de conocer la endoarquitectura de la CPU (duración de l a de bypass, etc.)

3.3. PLANIFICACIÓN DINÁMIC

- Hasta ahora todas las técnicas de segmentación ejecutan las instruc…den
  Si una instrucción no puede ejecutarse, la cadena se para completan…

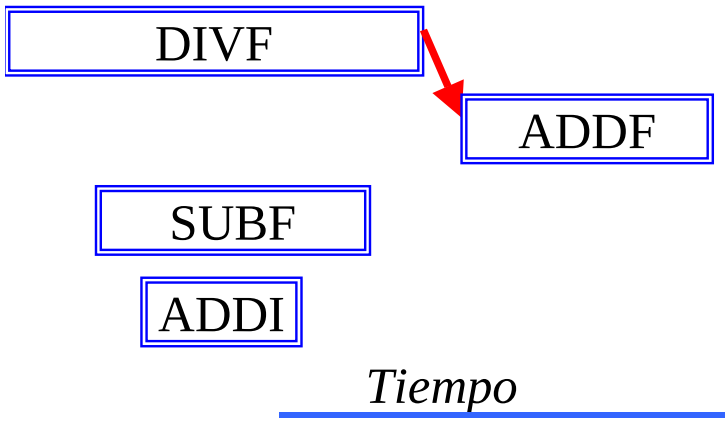
Cronograma ***sin*** *planificación dinámica*



- La espera que ADDF introduce en SUBF, ADDI, ..., podría … se permitiera que la ejecución (EX) no se hiciera en orden (*out-of-*…*n*) permitiendo que SUBF y ADDI comiencen antes su ejecución (EX…
- Es decir, estamos reordenando la fase EX en tiempo de ejecución (…e)

Cronograma usando Planificación dinámica

```
DIVF   F0, F2, F4
ADDF   F10,F0,F8
SUBF   F8, F8, F14
ADDI   R1, R1, 4
…
```



- Recordar grafo de dependencias ⇒ límite de CPI (*data flow limit* o l … de datos). Esto se intenta conseguir dinámicamente como en el cronogra…

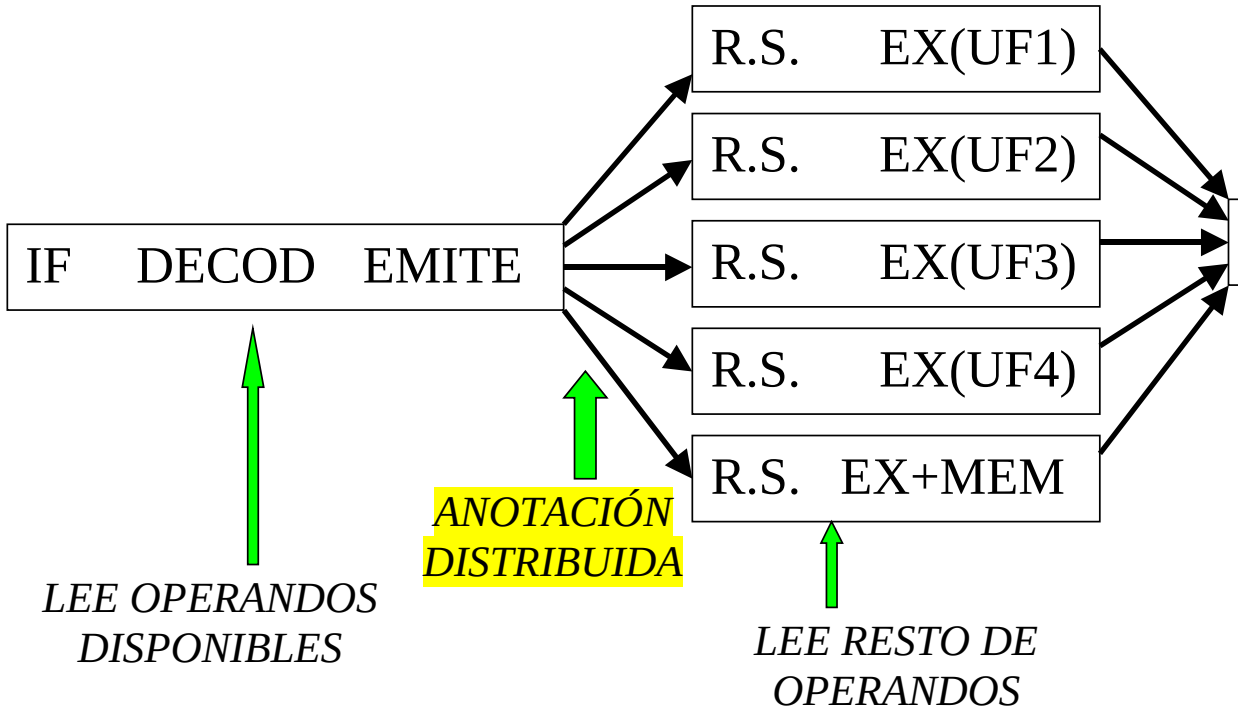
➢ La fase de decodificación (ID) siempre debe ir en orden (…bar dependencias).

➢ En el DLX vamos a suponer que se comprueban las dependencia… los riesgos estructurales en una única fase (ID o IS).

➢ La etapa ID (IS), por tanto, se puede considerar que debe hacer dos …

- Decodificación y Emisión (*Issue*): Decodificar la instrucci… bar dependencias de datos y riesgos estructurales (anotándolos d… a).
- Lectura de operandos (*Read Operands*): Leer los operando… (no tienen RAW cercana) y esperar por los operandos que tiener…

➢ Todas las instrucciones pasan en orden por la etapa ID o ISsue, y aquí pueden ser detenidas o desviadas a la unidad de ejecución correspon apa Issue no puede desordenarse, porque entonces no se sabrían las dependencias).



- Apuntar la información de dependencias en algún sitio:
  - Si la anotación es una tabla central, el algoritmo se llama dor centralizado (*scoreboard*).
  - Si la anotación es distribuida, el principal algoritmo es el de R. T 67)

Algoritmo de Tomasulo.

- Técnica de planificación dinámica de instrucciones con ges uida. Inicialmente, en el IBM/360 (1967, Robert Tomasulo) era sólo pa s de FP. Hoy se aplica a todas las instrucciones, y la mayoría de proces ados llevan un algoritmo similar al clásico (estudiaremos el clásico pero o de instrucciones, porque es el explicado en los libros generalmente).
- Este algoritmo se puede aplicar a otro tipo de sistemas donde hay rsos compartidos por diversos agentes, y se quieren gestionar los re orma distribuida. Evidentemente la implementación que veremos aquí es
- Durante su ejecución provoca un reordenamiento dinámico en la ej EX) de las instrucciones, aunque la emisión sigue siendo en el orden del al.
- Permite la ejecución de instrucciones no dependientes de otras a nque estas últimas estén bloqueadas esperando por algún operando fuente
- Va a realizar un renombrado dinámico de registros para ev por dependencias (WAR, WAW) entre instrucciones. Todos los regist o se renombran y el nuevo nombre que toman se llama **etiqueta** (*tag*). ncia 26.

Definiciones

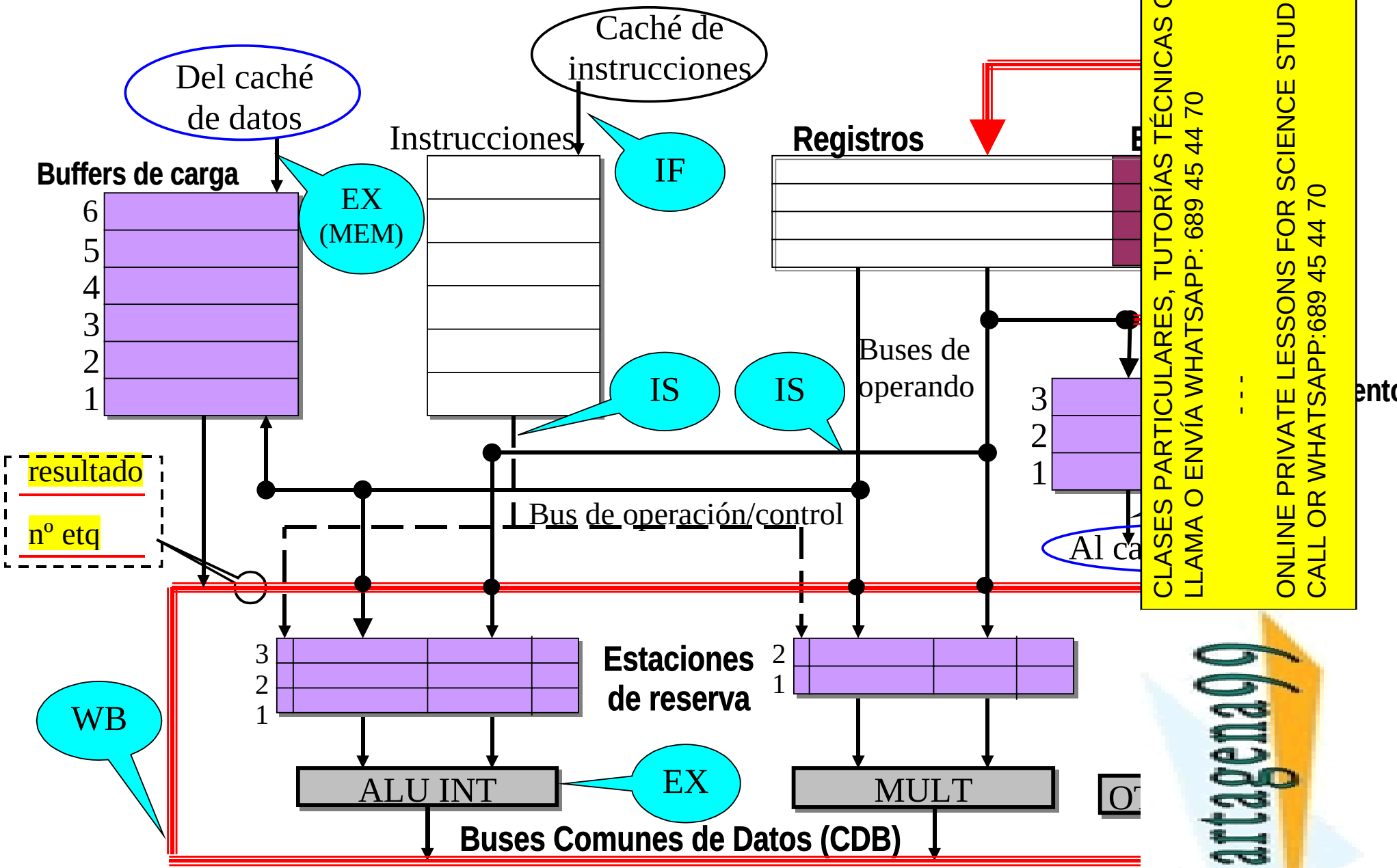
En este algoritmo se ponen una serie de entradas en cada UF, llama es de reserva RS. A éstas llegan las emisiones (IS) de instrucciones con to ción de la misma, también llegan los valores de los operandos fuente, si ha se, o cierta información (etiqueta) que indica que operando falta (no se bloq).

- ***Estación de Reserva (Reservation Stations, R.S.)***: Lugar dó las instrucciones emitidas junto a las ALU's (U.F.), hasta que pueden en la U.F. Esta espera se debe a la no disponibilidad de todos los opera (hay RAW: una instrucción anterior aún no ha generado algún operando ndrá una serie de campos para anotar el valor de los operandos fuente (y io el valor del resultado de la operación y la etiqueta asociada al registro d

  En la implementación del alg. Tomasulo, las estaciones de son registros asociados a las unidades funcionales. De manera que for uenã cola de instrucciones a la entrada de la unidad funcional (de unas p si la operación no es muy común, como DIV, a muchas si es muy frecuer stro contendrá la información necesaria para la ejecución de la instr a la llegada de los operandos que le falten (en caso de que la instrucción e).

- ***Buffers de Memoria***: Son las R.S. para instrucciones de carga/alma Es el lugar dónde esperan los acceso a memoria emitidos hasta que dispo sus operandos y el bus de memoria (caché) está libre. En el alg. clásico hay buffers separados para instrucciones de carga y para los de almacena
- ***Bus Común de Datos (Common Data Bus, CDB)***: Une la salida ades funcionales y la carga de memoria con el fichero de registros, las esta erva (y los buffers). Las UF mandan el dato de salida a través de él para q esto de elementos de la CPU (es decir, se usa en la fase WB, produciéndo). Se trata de un recurso común por el que hay que competir. En gene inas suelen tener más de un CDB (p.ej. uno para enteros y otro para F e el número de WB por cada ciclo puede ser mayor que 1. Cualquier CD uses: uno para el resultado de una operación (32 líneas para INT/float, 6 e) y otro para la etiqueta asociada al mismo (tamaño $\log_2$ del número de e
- ***Etiqueta (tag)***: Son identificadores que se asocian a los registros des que se emite una instrucción. A partir de renombrar un registro destino ieta, es ésta la que circula por toda la CPU (estaciones de reserva, buff de registros, CDB, etc.). Una etiqueta estará en uso desde que la ins nite hasta que su resultado es puesto en el CDB. En el alg. clásico el nom ueta coincide con el de la R.S. Por ej. si la UF de MULT tiene 3 F las etiquetas serán MULT1, MULT2, MULT3 (codificadas en binario).

Esquemático Algoritmo Tomasulo clásico



Etapas Pipeline

- Se puede definir la especificación exacta (en lenguaje C o con del algoritmo de Tomasulo. Esto se implementa luego en hardware.

  Dado que con la arquitectura hardware del algoritmo de Tomasulo se bloqueará cuando haya RAW, la cadena (unidad de control) debe can

  - **Fetch (IF):** Acceso a caché de instrucciones para leer instrucción a PC
  - **Issue (IS):** Emisión de la instrucción. En el alg. clásico en esta et n un solo ciclo (hoy en día esta etapa dura varios ciclos de reloj). Realiz :
    - Decodificación
    - Lectura de operandos ya disponibles
    - Envío hacia la R.S. (esta sería la auténtica emisión).

    Al final de IS, se intenta enviar una instrucción a una R de carga/almacenamiento), asignando una etiqueta al reg. destino.

    - Si hay una RS. (o buffer) libre en la UF requerida por la oper e la instrucción mandando a la RS los operandos disponibles. Para stén disponibles, se anota la etiqueta por la que esperan.
    - Si no hay RS (o buffer) libre o no quedan etiquetas libres, la eta ará hasta que una RS de la UF afectada se libere. En el alg. clásico (y s lo se realice en una única fase IS) esto provoca una detención de la caden

- **Execution (EX):** Cuando una RS (buffer) obtuvo todos sus oper…za a ejecutarse. La RS puede quedar libre:
  - Al comenzar a ejecutarse en la UF (primer EX1). Entonces, esta… varía en algún otro sitio dos campos: uno para el resultado y otro… queta asociada al valor de salida (se envía en WB junto al resultado de…).
  - O al final de WB. La R.S. tendrá todos los campos necesario… ndos: fuente, resultado y etiqueta. Por motivos que veremos en ASP2… er lo normal y será el criterio para ASP1 (mientras no se indique lo co…
- **Write (WB):** El resultado de una operación y su etiqueta se escrib…B, de manera que las RS o buffers que estén esperando por este valor… isma etiqueta en alguno de sus operandos) lo toman, es decir, se proc… ss o camino de desvío a través del CDB. Igualmente el valor del CDB… en el fichero de registros FP si su etiqueta coincide con alguno. Tr… ra la etiqueta se libera. El fichero de registros deberá, por tanto, tener i… obre la última etiqueta que va asociada a cada registro.

NOTA: fase MEM no existe (se considera dentro del EX para los buffe…).

Especificación de Arquitectura de Tomasu

- Campos de cada Estación de Reserva (según versión, algunos son):
  - Op: operación
  - Qj, Qk: Etiqueta de los operandos j y k. Si está vacío, el operando .
  - Vj, Vk: Valores de los operandos.
  - Ocupado: Indica que la RS está en uso.
- Buffer de LD:
  - Dirección: para el acceso a memoria (etiqueta si oper. no disponib)
  - Ocupado: Indica que el buffer está en uso.
- Buffer de ST:
  - Qi: Etiqueta de la RS que producirá el valor a almacenar en memo
  - Dirección: para el acceso a memoria (etiqueta si oper. no disponib)
  - Vi: valor a almacenar. Estará vacio cuando Qi esté lleno.
  - Ocupado: Indica que el buffer está en uso.
- Fichero de Registros FP. Cada registro tendrá la estructura:
  - Qi: Etiqueta de la RS que producirá el valor a almacenar en el regi
  - Vi: valor del registro.
  - Ocupado: Indica que el valor actual del registro no es válido (espe).

Resolución de riesgos

Con el alg. Tomasulo, se eliminan los riesgos por dependencias ólo existe bloqueo (de tipo estructural) cuando se agotan las R.S. o las = 0

- RAW: las anota en las estaciones de reserva y espera por las etiq
- WAW: al renombrar registros, nunca se les asocia la misma los registros de destino que aún estén "circulando" por la CPU.
- WAR: al renombrar registros para evitar la WAW, la WAR desa

Ejemplo de renombrado: etiquetas se llaman como R.S: (de las cuale s). Se supone que los reg. R1, R2, F0,... no estaban "en uso" antes de es tar que sólo quedan las RAW reales, que van formando el grafo de depe

```
LD      F0, 0(R1)
LD      F2, 0(R3)
MULTD   F4, F0, F2
LD      F6, 0(R2)
ADDD    F4, F6, F4
ADDD    F4, F10, F4
ADDI    R1, R1, 16
LD      F0, 8(R1)
```

```
LD      Carga1, 0(R1)
LD      Carga2, 0(R3)
MULTD   Multfp1, Carga1, Ca
LD      Carga3,  0(R2)
ADDD    Addfp1, Carga3, M
ADDD    Addfp2, F10, Addfp
ADDI    Ent1, R1, 16
LD      Carga1, 0(Ent1); el
```

Carga1 ya est

Ejemplo 1

Cronograma del fragmento: Suponemos 3 RS suma, 2 RS multiplica[illegible]2 de carga. (Carga3 no existe, deberá usar Carga1 otra vez y esperar a que

Duración Ld/St : 2ciclos (como EX+MEM). MULTD: 4 ciclos, ADD[illegible]

```
LD      F0, 0(R1)
LD      F2, 0(R3)
MULTD   F4, F0, F2
LD      F6, 0(R2)
ADDD    F4, F6,  F4
ADDD    F4, F10, F4
ADDI    R1, R1, 16
LD      F0, 8(R1)
```

Notar el reordenamiento dinámico de las fases EX. Típicamente las I[illegible]an a las FP.

Mostrar el estado de las RS y de los registros, tras emitir la última sun[illegible]

Ejemplo 2: Bucles con Tomasul

Sea el procesador DLX con algoritmo de Tomasulo con las siguientes s:

- 1 UF de operaciones enteras simples de un ciclo de duración, co
- 1 UF de Ld/St duración 2 ciclos, con buffers indep. de LD y ST, s.
- 1 UF de operaciones FP, 4 ciclos para sumas y mult., con 3 RS.
- 1 UF de saltos de un ciclo de duración, con 1 RS.
- Existen 2 CDB, uno para operaciones enteras, y otro para punto

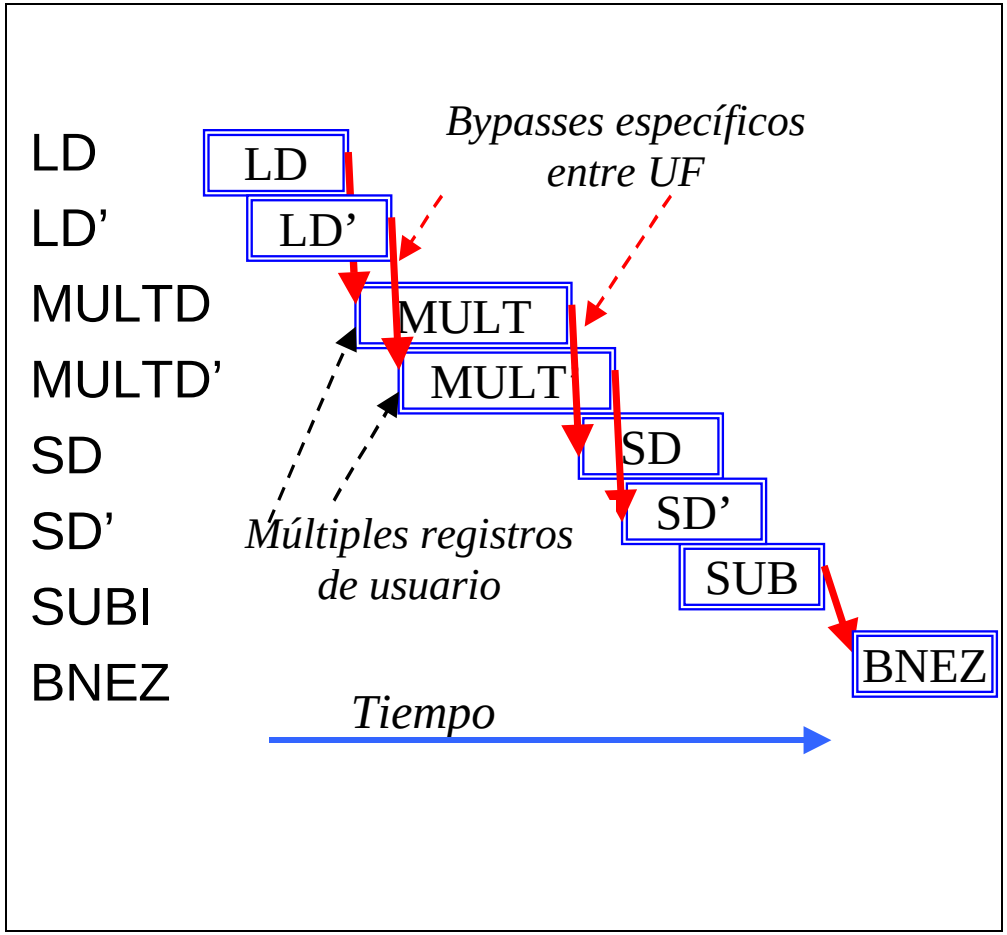
Cronograma del código para tal DLX. Suponer BTB accede en IF y si .

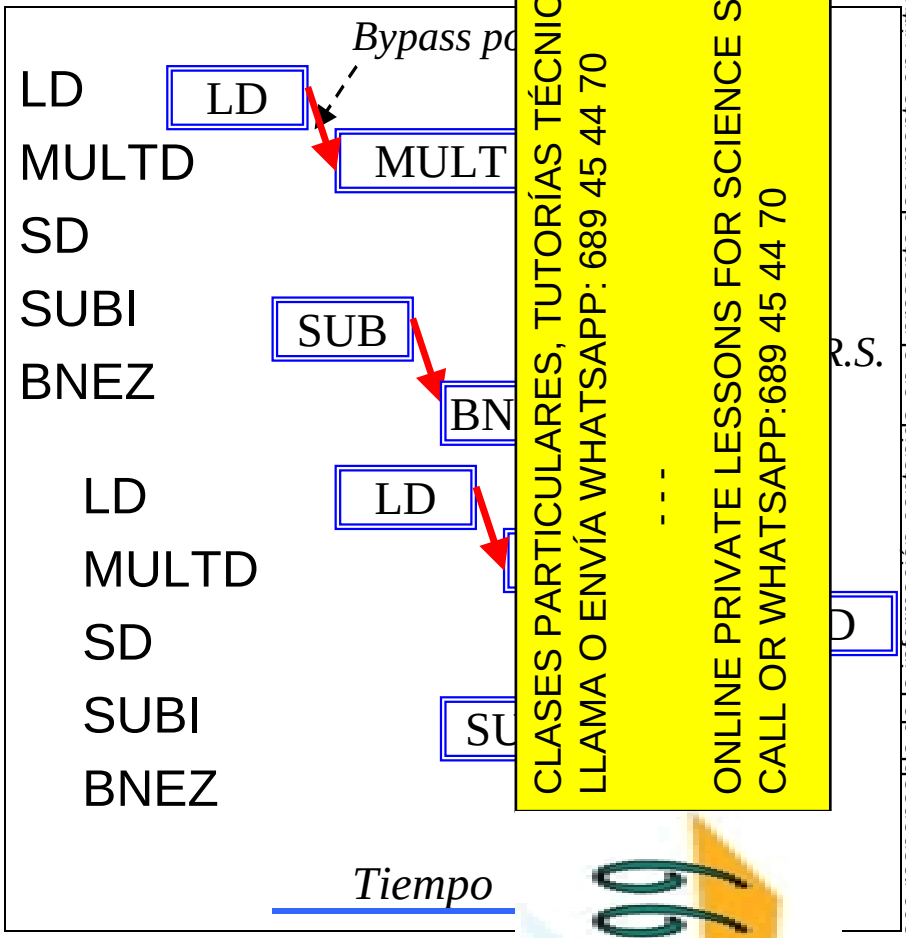
```
Loop:  LD     F0, 0(R1)
       MULTD  F4, F0, F2
       SD     0(R1), F4
       SUBI   R1, R1, 8
       BNEZ   R1, Loop
```

Ejecutar dos iteraciones seguidas y comprobar el desenrollado dinámi

NOTA: Desenrollar para una máquina con alg. Tomasulo puede pr os por agotarse las RS por lo que la cadena se bloquearía y se empeorarí

Croquis: diferencia entre planificación estática y
para bucle paralelizable (sólo flujo de dat





Desenrollado estático

Desenrollado diná

- Tb. conviven varias iteracion
- Adelantar emisión de instr de r

Variaciones del Alg. Tomasulo cl

La implementación del algoritmo de Tomasulo en las máquinas a la misma idea principal del algoritmo visto, pero se puede encontr tas modificaciones o adaptaciones:

- Antiguamente se aplicaba el algoritmo de Tomasulo sólo para ins rgas (típicamente FP, código para supercomputadores científicos). D la cadena era distinta para FP que para enteros. Actualmente las ope eras también llevan planif dinámica similar al algoritmo de Tomasu las secciones FP e INT están separadas y otras en común.
- En el algoritmo clásico etiqueta y R.S. son lo mismo. Se supone una RS luego para poder emitir, el requisito es único: ¿hay RS disponible UF termina la ejecución, se devuelve el resultado a la RS. La RS pondr en el CDB cuando este está libre, quedando libre la R.S. (y la eti tros usaremos el algoritmo clásico **(libro Henn-Patt)** mientras no se diga
- Arquitectura con RS que se liberan al empezar la fase EX (c los operandos fuente están disponibles). Cuando la UF termina la ejecu e el resultado de la operación en otro registro interno. En cuanto haya se libera tal registro interno.

- Arquitectura con "piscina de etiquetas" (*tagpool)*. Las etiquetas [illegible]urso común y compartido. Para emitir una instrucción, requisito doble: [illegible]una etiqueta disponible y una RS libre (la etiqueta tiene asociad[illegible]nto hardware). El Reg destino se renombra con la etiqueta (y no coincid[illegible].
- Un ejemplo usual de piscina de etiquetas donde las RS se liberan en [illegible] los micros que renombran, dinámicamente y para todas las instr. emiti[illegible]tros lógicos (de usuario, de las instr. de ensamblador) por registros físi[illegible] no visibles al programador). Los registros físicos funcionan como etiqu[illegible]aber muchos más registros físicos que lógicos. Ej Power PC, MIPS
- El conjunto de RS puede ser: *a)* común para todas las UF, *b)* separa[illegible]T y RS FP, *c)* cada UF tenga sus RS asociadas y propias (como Tomasul[illegible]
- Todos tienen varios CDB para permitir más de una escritura al mism[illegible]
- En algunos micros la fase WB no existe, en el último ciclo de ejecuc[illegible]e en el bus común (como se hace en el DLX sin planif dinámica, el desv[illegible]por un bypass no tarda ningún ciclo). Así se acercan al límite flujo de da[illegible]
- La cadena tiene diferente número de etapas (no es la clásica [illegible]B) Generalmente la fase IS se subdivide en varias subfases, pues hay q[illegible]has operaciones: decodificación, lectura operandos, renombrado, emisió[illegible]

Dependencias de Memoria

- Los accesos a memoria (ejecutados en búferes de carga y de ...nto) pueden tener dependencias entre los datos que leen o escriben en me...
- Para ello debe ocurrir que la dirección de un acceso sea igua... ...tro. Evidentemente esta dependencia no se puede detectar en t. de compi...
  - `SW  20(R3), R9`
  - `LW  R1,  8(R4)` Si ocurre que: 20+R3 = 8+R4 ⇒ RA... ...ria
- Ej: Antes de que un Load acceda al caché, se debe comparar su d... ...tiva con todas las de los búferes de almacenamiento. Si hay coinciden... ...ecta una dependencia real en memoria, y hay que esperar a que el Stor... ...no existir tal comparación de direcciones, no se podría ejecutar un Lo... ...que acabaran todos los Store (caché actualizado). Y los Store suelen... ...mas instrucciones en ejecutarse (ver cronogramas, p ej. bucle del Ejemplo...
- Los búferes de Store son como un *buffer de escritura* (conserva... ...ato escrito). Los Store se guardan en el buffer y se dan por realizados. ... ...son más prioritarias (hay otras instr. esperando por sus datos).
- En procesadores con pocos registros de usuario (CISC) el código... ...hos accesos a memoria. El riesgo es mayor y se necesita más hw para res...

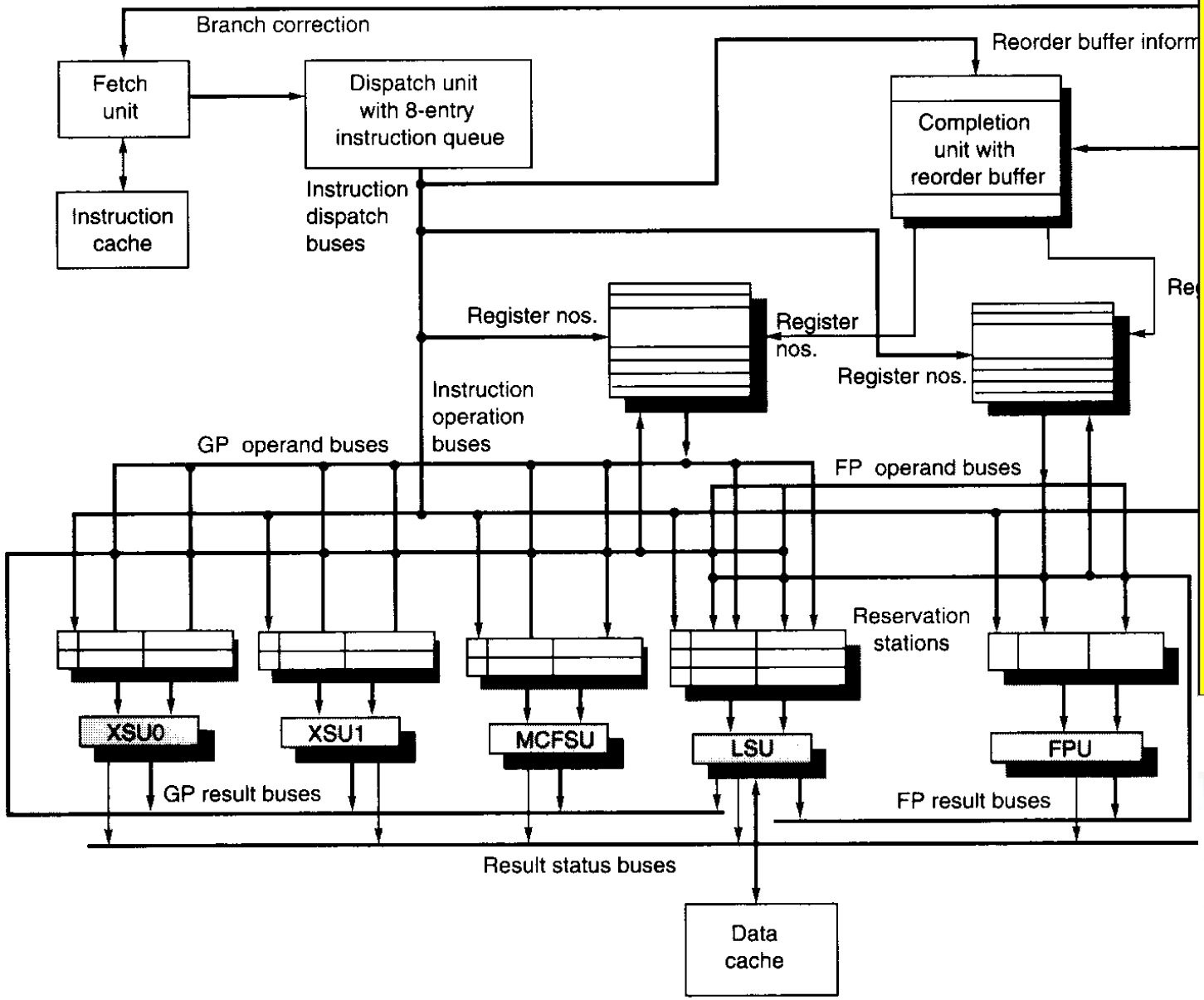
Tomasulo en procesadores actua

- En general los micros con planif dinámica son capaces de emitir rias instrucciones a la vez, y su arquitectura es muy potente.
- Uno de los primeros microprocesadores con planif dinámica fue Mo PC **620** (1995). Es un ejemplo muy bueno: su cadena (4 fases: I) y arquitectura es muy similar al algoritmo clásico Tomasulo. Poseía R.S por cada UF.
  - Dos unidades enteras simples (1 ciclo).
  - Una unidad entera compleja (3 a 20 ciclos) para MULT y DIV
  - Una unidad de carga-almacenamiento (1 ciclo si acierta en cac
  - Una unidad de punto flotante (31 ciclos DIVFP, 2 ciclos ADD P).
  - Y una unidad de saltos, que actualice BTB en caso de fallo de p
- **Pentium Pro** (su parte INT es idem que P.II, P.III y similar al P4). de 11 fases. Posee 40 R.S. comunes a todas las UF:
  - Dos unidades enteras (1 ciclo). Una de ellas se usan para saltos TB en caso de fallo de predicción). La otra se usa también para mu s y divisiones enteras.

- Dos unidades FP para multiplicaciones y divisiones de ...te. Realmente solo puede empezar en el mismo ciclo una de ellas.
- Una unidad de carga (1 ciclo).
- Una unidad de almacenamiento más compleja (lleva un buffer ...l).

- **MIPS R10000** (similar a los actuales). Posee 16 R.S. para INT, 16 p... para Ld-St:
  - Dos unidades enteras (una también sirve para saltos, y la o... LT INT).
  - Una unidad para cálculo de direcciones de acceso a memo... ga-almacenamiento).
  - Una unidad de punto flotante simple (sólo para ADDFP).
  - Una unidad de punto flotante compleja (DIV, MULT, SQRT-F...
- Las unidades funcionales suelen ser segmentadas excepto en las divi...

Ej: Power PC 620 (similar al Tomasulo)



Rendimiento del Scheduling Dinám

- Dotar de muchas RS a una UF de corta duración no tiene sentido p…erar rápidamente. Aunque cuantas más dependencias reales existan men…n y más R.S. se necesitan.
- Pero las U.F. de larga duración suelen ser de instr poco comunes (co…ego tampoco se requieren muchas RS. Hay que calcular el nº RS en fun…esto y no excederse en RS, puesto que:
- Principal inconveniente del Alg. Tomasulo: alto coste hardware, sob…gica de comparación de las R.S y búferes acceso a memoria. Cada R.S de…la/s etiqueta/s por la/s que espera con la/s que aparece/n en el/los CDB.
- La aceleración que se consigue para un procesador DLX pipeline …de Tomasulo es de 1.58, con respecto al que no tiene planificación din…tica del compilador). No es muy alta porque el DLX tiene muchos …que permite reducir el número de antidependencias y por tanto reordenar
- La aceleración sería mucho más alta comparando para una máq…cos registros.

3.5. TÉCNICAS SOFTWARE AVAN S

Existe una gran variedad de casos donde métodos simples de planific no son aplicables directamente. Veamos algunos ejemplos típicos e ilustr

- Los bucles de vectores son paralelizables. Pero no todos som paralelizables. P. ej. cuando hay una dependencia entre la iteració las anteriores, no podrían entrelazarse las instr de las diferentes iteracio . de dependencia: Serie de Fibonacci (suponemos que la operación suma

```
for (i=2;i<M; i++) x[i]=x[i-1]+x[i-2]; //(2 Load, 1 S                m
// Si desenrollo:
for (i=2;i<M; i+=3) {
        x[i]=x[i-1]+x[i-2];
        x[i+1]=x[i]+x[i-1];
        x[i+2]=x[i+1]+x[i]; } // En total: (2 Load, 3 S                em
```

- Por supuesto siempre se obtiene cierta aceleración porque se elimi nes de overhead (e incluso algunas de acceso a memoria si el compilado
- En general cuando en una serie hay recurrencia: $a_i=f(a_{i-k})$ ó $a_{i+k}=f(a_i$ s k
- Pero si k es grande puedo desenrollar k veces, con lo cual las d se pueden alejar (entrelazando las k iteraciones):

```
for (i=0;i<M; i++) x[i+4]=x[i]*s;  //Si desenrollo 4
```

```
for (i=0;i<M; i+=4) {
        x[i+4]=x[i+0]*s;
        x[i+5]=x[i+1]*s;
        x[i+6]=x[i+2]*s;
        x[i+7]=x[i+3]*s; }
```

- Si un bucle tiene varias líneas de código con posibles depend… …nas, conviene separar cada línea en bucles diferentes y luego comp… …stem dependencias. Finalmente desenrollar si ya se puede. Ej:

```
- for (i=0;i<M; i++) {
        x[i]= x[i]* y[i];
        y[i+1]= z[i]* s;
        }   //parece que existe una dependencia entre i…
```

```
- for (i=0;i<M; i+= 2) {
        x[i]= x[i]* y[i];
        y[i+1]= z[i]* s;
        x[i+1]= x[i+1]* y[i+1];
        y[i+2]= z[i+1]* s;
        } // es ficticia si cambio el orden y separo en…
```


```
- for (i=0;i<M; i+=2) {
        y[i+1]= z[i]* s;
        y[i+2]= z[i+1]* s;
        x[i]= x[i]* y[i];
        x[i+1]= x[i+1]* y[i+1];
        } // Luego finalmente (el orden contrario sería
```

```
  for (i=0;i<M; i++) y[i+1]= z[i]* s;
  for (i=0;i<M; i++) x[i]= x[i]* y[i];
```

- Otro ej de bucle muy usual: sumatorio (idem para producto múltip… …ipio existe dependencia entre una iteración y la siguiente a través… …able acumuladora. Pero tal variable ha surgido de la forma en que se escr… …rio. Si lo rescribo, desaparece. Por ej con varias variable acumuladora… …otra forma sería escribiendo un sumatorio con algoritmo tipo árbol):

```
- for (i=0;i<M;i++) t = t + y[i];
```

```
- for (i=0;i<M; i+=3){
        t0 += y[i+0];
        t1 += y[i+1];
        t2 += y[i+2];
```

```
            }
    t = t0 + t1 + t2;
//Ahora puedo ejecutar estas 3 iteraciones en paralelo (y
// quedando las dos sumas de los resultados parciales al f
```

Ejercicio: Desenrollar el Producto escalar de dos vectores

- Encadenamiento software de iteraciones (*software pipelining*). Cua… …ucle paralelizable no se puede desenrollar porque no hay registros s… …n el procesador (CISC), o porque no interesa tamaño grande de código, …ejar las dependencias reales rescribiendo el código con esta técnica.

Sea el ejemplo del apartado 3.2, 3.4: `for (i=0 ; i<M ; i++ ) y… …s;`

```
bucle_orig:
    LD     F2, 0(R1)
    MULTD  F4, F2, F24  ; F24 contiene el valor de s
    SD     (R3)0, F4
    ; instr overhead a continuación (increm. punteros, sa…  )
    ADDI   R1, R1, 8
    ADDI   R3, R3, 8
    SLTI   R7, R1, fin_array_x; constante apunta al final
    BNEZ   R7, bucle_orig
```

Se ve claro con un esquema en la siguiente tabla:

iter 0	LD^0	$MULTD^0$	SD^0	instr overhead^0		
iter 1		LD^1	$MULTD^1$	SD^1	instr overhead^1	
iter 2			LD^2	$MULTD^2$	SD^2	i ove
iter 3				LD^3	$MULTD^3$	
iter 4					LD^4	MU

← *instrucciones de arranque* →← *iter. reordenadas*→

De la traza de ejecución normal por filas en la tabla (flechas roj a la ejecución por columnas (flechas verdes).

El código resultante tendría unas cuantas de instrucciones de arranqu (en este ejemplo las tres primeras columnas) y unas cuantas de instruccio ra o "finish-up" (serían las últimas 4 columnas). El resto de columna se pueden ejecutar de arriba abajo, habiendo alejado las dependencias que retocar las constantes para que sea correcto).

Sólo quedan antidependencias que no producen ciclos de bloqueo. alto condicional esté abajo, cambio de posición las instr de overhead eva retocar las constantes de los Ld/St):

;instr overhead^0 SD^1 MULTD^2 LD^3	bucle_encadenado: ;instr overhead^0 SD^1 (R3)0, F4 MULTD^2 F4, F2, F24 LD^3 F2, (R1)2*8 // esta columna no es correcta por que el salto estaría arriba	// código bucle_enc SD^1 MULTD^2 4 LD^3 ;instr // código

Los desplazamientos de los Ld/St se han calculado en función de la i que pertenece cada Ld/St. En la tercera columna se han cambiado los de s (y la cte de SLTI, que sería `fin_array_x-24` para que LD F2,(R1)2 del array) porque las instr. overhead se han movido.

- Notar que las RAW se han convertido en WAR (que no produce
- Notar que no puede haber WAW o WAR en el bucle original (a a al orden de las instr. se podrían convertir en dependencias reales e

3.6 CONCLUSIONES: PLANIFICACIÓN ESTÁ. DINÁMICA. RESUMEN: PROS Y CONTRAS

PLANIFICACIÓN ESTÁTICA	PLANIFICACIÓN I
Menos Hardware	Complicación har
Compilador más difícil	Compilador no tiene qu
Posibles problemas de herencia (compilador debe conocer endoarquitectura)	Transparente al u
Inconveniente: Dependencia compilación-rendimiento	El hardware extrae el rendir de en cada versi
Tamaño de código estático puede crecer ⇒ más fallos de caché	Tamaño de código estát
Ventaja: Ventana de instrucciones infinita (análisis global)	Defecto: Ventana de instru a (fase IF) (análisis
El compilador no puede conocer: - valores de registros (dir. acceso) - predicción dinámica, etc.	En tiempo de ejecución s - valores de registros - predicción dinámica,

Puede eliminar instrucciones (de overhead u otras)	No puede eliminar ins
Puede necesitar muchos registros de usuario	No necesita muchos regis (son internos, ocultos al us

- Conocimiento de programas reales (o los que se van a ejecutar) es ante antes de ponerse a diseñar una máquina (o a elegir una)
- Dos Tendencias:

  Hw simple y Compilador complejo vs. Hw complejo y Compi

  Esta disyuntiva se está dando actualmente con micros avanzados (SP2) aunque cada vez se traspasa más funcionalidad al hardware.