PLANIFICACIÓN DINÁMICA EN EL PROCESADOR MIPS R4000

Jose Alberto Benítez Andrades

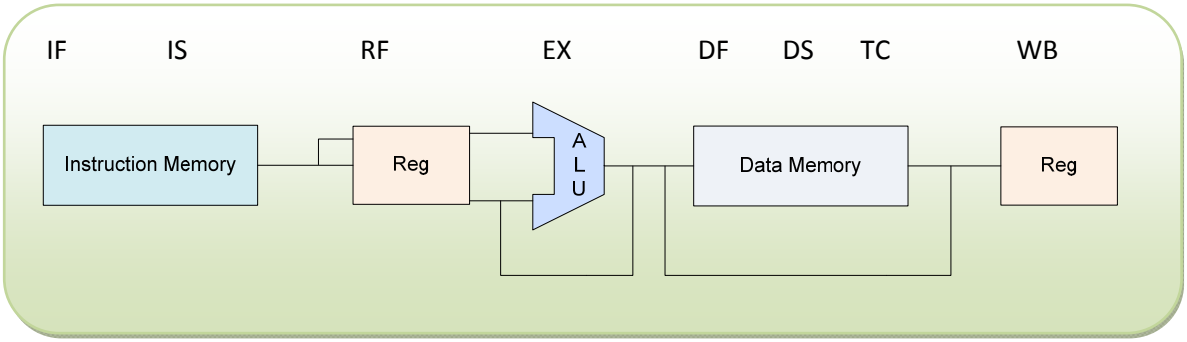
71454586A

Arquitectura e Ingeniería de Computadores

Ingeniería en Informática

1. Estructura de la ruta de datos enteros segmentada del procesador MIPS R4000

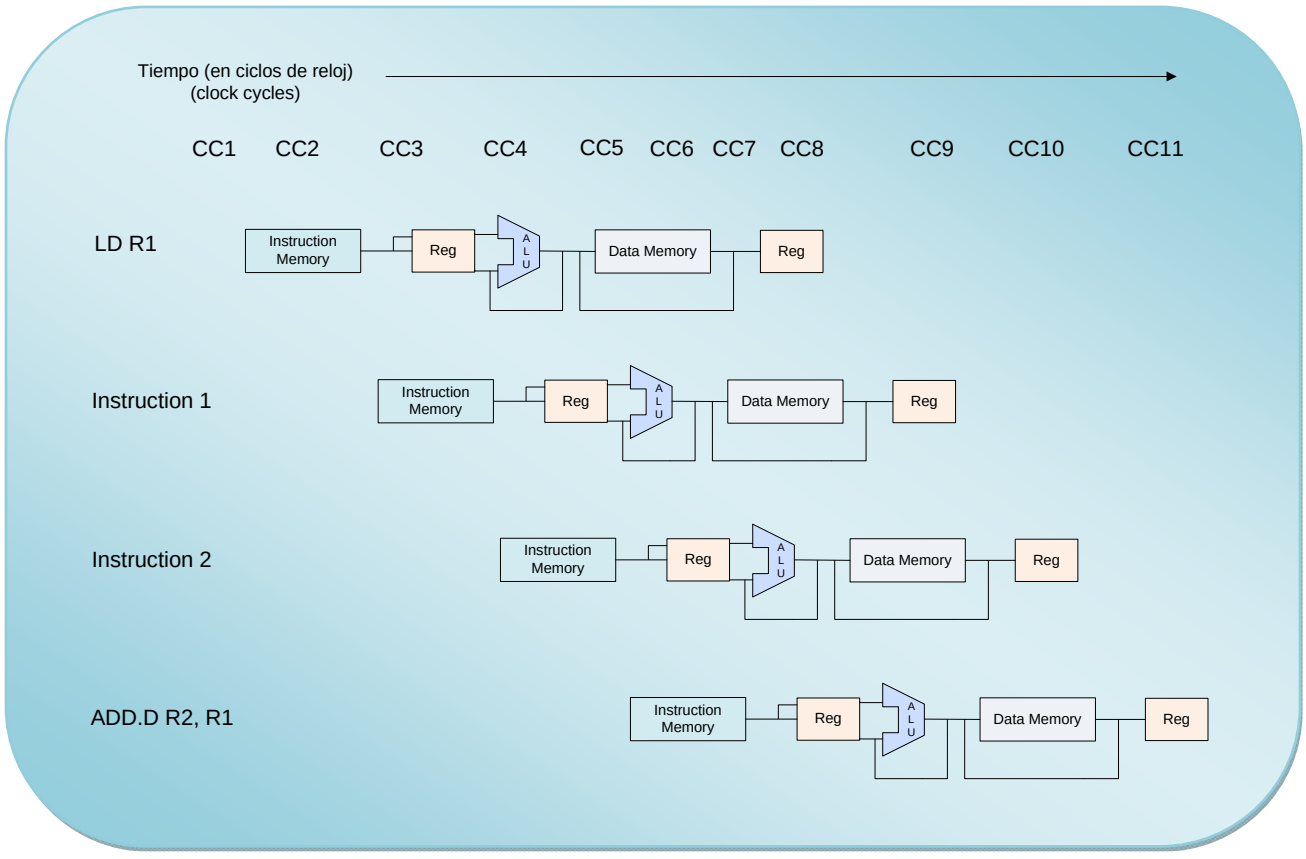
La estructura de la tubería del procesador MIPS R4000 es de la siguiente forma:



Posee los siguientes 8 estados:

1. IF - Instruction fetch. Selección de PC, iniciación de la instrucción de acceso a caché.
2. IS – Segunda parte del IF. Completa el acceso a caché.
3. RF - Register fetch. Decodifica instrucción, chequea riesgos y chequea etiquetas.
4. EX - Execution. Cálculo de dirección efectiva, operaciones de ALU y también las diferentes ramificaciones de computación así como la evaluación de condiciones.
5. DF - Data fetch. Inicia el acceso de datos a la caché.
6. DS - Second half of DF. Completa el acceso a caché.
7. TC - Tag check. Determina si la caché de datos tiene problemas.
8. WB - Write back. Para cargas y operaciones con registros.

La estructura de datos enteros segmentada para el procesador MIPS R4000 con un retardo (delay) de 2 ciclos, es la siguiente:



2. Esquema de la ruta de datos de coma flotante del MIPS R4000

La unidad R4000 para datos de coma flotante está formada por tres unidades funcionales: *divisor de coma flotante*, *multiplicador de coma flotante* y *sumador de coma flotante*. El *sumador* se usa en el final del paso del *multiplicador* o del *divisor*. Los coma flotante de doble precisión pueden tener de 2 a 112 ciclos para una raíz cuadrada. Varias unidades tienen diferentes ratios de iniciación. La unidad funcional de coma flotante pueden pasar por 8 estados diferentes, estos estados combinados en diferente orden, ejecutan varias operaciones para los coma flotante:

Estados de las tuberías de coma flotante:

Estado	Unidad funcional	Descripción
A	FP adder (sumador)	Suma de Mantisa
D	FP divider (divisor)	Divisor de la tubería
E	FP multiplier (multiplicador)	Prueba de excepciones
M	FP multiplier (multiplicador)	Primer multiplicador
N	FP multiplier (multiplicador)	Segundo multiplicador
R	FP adder (sumador)	Redondeo
S	FP adder (sumador)	Cambio de operando
U		Desempaqueta números.

Latencias, intervalos y unidades usadas para las operaciones de los números de coma flotante:

Instrucción	Latencia	Intervalo de iniciación	Estado de tuberías
Sumar, restar	4	3	U,S+A,A+R,R+S
Multiplicar	8	4	U,E+M,M,M,M,N,N+A,R
Dividir	36	35	U,A,R,D^{28},D+A,D+R,D+A,D+R,A,R
Raíz cuadrada	112	111	U,E,$(A+R)^{108}$,A,R
Negación	2	1	U,S
Valor Absoluto	2	1	U,S
Comparación	3	2	U,A,R

En la tabla anterior, se refleja el tiempo que se tarda en ejecutar una instrucción (latencia), cuánto tiempo consume al iniciarse y los estados por los que pasa, dependiendo de la operación que se realice. La relación de estados se interpreta con ayuda de la tabla anterior.

En la notación de los estados, S + A, indica que ambos se ejecutan en un ciclo de reloj, es decir, en un ciclo de reloj, son usados esos dos estados. Y la notación D^{28} indica que el estado D es usado 28 veces en una fila.

3. Planificación estática vs planificación dinámica.

Las tuberías sencillas esperan una instrucción y la ponen en marcha, a menos que haya una dependencia de datos entre una instrucción que no puede ser ocultada ni evitando ni sobrepasando la anterior. Lógicamente sobrepasando se reduce la efectividad de la latencia de la tubería, así ciertas dependencias no resultan peligrosas. Si hay un riesgo inevitable, entonces el hardware que detecta el riesgo, bloquea la tubería (comenzando con la instrucción que usó el resultado). No hay nuevas instrucciones funcionando hasta que la dependencia es borrada. Para vencer estas pérdidas de desempeño, el compilador puede planificar instrucciones para evitar el riesgo, en esto consiste la *planificación estática.*

Pero los procesadores no solo utilizan planificación estática, muchos utilizan planificación dinámica, mediante la cual, el propio hardware vuelve a arreglar la ejecución de la instrucción para reducir los bloqueos o paradas. En la tubería de MIPS, los riesgos tanto estructurales como de datos, son comprobados durante la decodificación de la instrucción (ID): Cuando una instrucción puede ejecutarse correctamente, esta es puesta en marcha desde el estado *ID.* Para permitir una instrucción comenzar su ejecución tan pronto como sus operandos estén disponibles, si su predecesor está bloqueando la tubería, nosotros deberemos separar el proceso publicado en 2 partes: comprobando el riesgo estructural y esperando para la ausencia del riesgo de datos. Nosotros decodificaremos y publicaremos instrucciones en orden. Sin embargo, queremos las instrucciones para comenzar la ejecución tan pronto como sus operandos de datos estén disponibles. Así, las tuberías pueden hacer *out-of-order execution*, que implica *out-of-order completion* (ejecución de forma desordenada que implica completar la ejecución de forma desordenada también). Para implementar este método, debemos separar el estado ID (Instruction Decode) en dos estados:

1. *Issue* (puesta en marcha) – Decodifica las instrucciones, comprueba los peligros estructurales.
2. *Read operands* (leer operandos) – Espera hasta que no haya riesgos en los datos, y entonces lee los operandos.

El estado *IF (Instruction Fetch)* se ejecuta junto con el estado 1 (*issue)* y el estado *EX (Execution)* va seguido del estado 2 *(Read operands)*, así como en la tubería de MIPS. En la tubería de MIPS para los coma-flotante, su ejecución puede tener múltiples ciclos, dependiendo de la operación. De esta forma, nosotros podemos necesitar distinguir cuándo una instrucción comienza su ejecución y cuándo completa su ejecución; entre los dos tiempos, la instrucción está ejecutándose. Esto permite que haya múltiples instrucciones ejecutándose simultáneamente. Además de estos cambios en la estructura de la tubería, nosotros también cambiaremos el diseño de las unidades funcionales, para variar el número de unidades, la latencia de las operaciones, y la tubería de unidades funcionales, así se explorará mejor de forma más avanzada.

4. Planificación dinámica con marcador (scoreboarding):

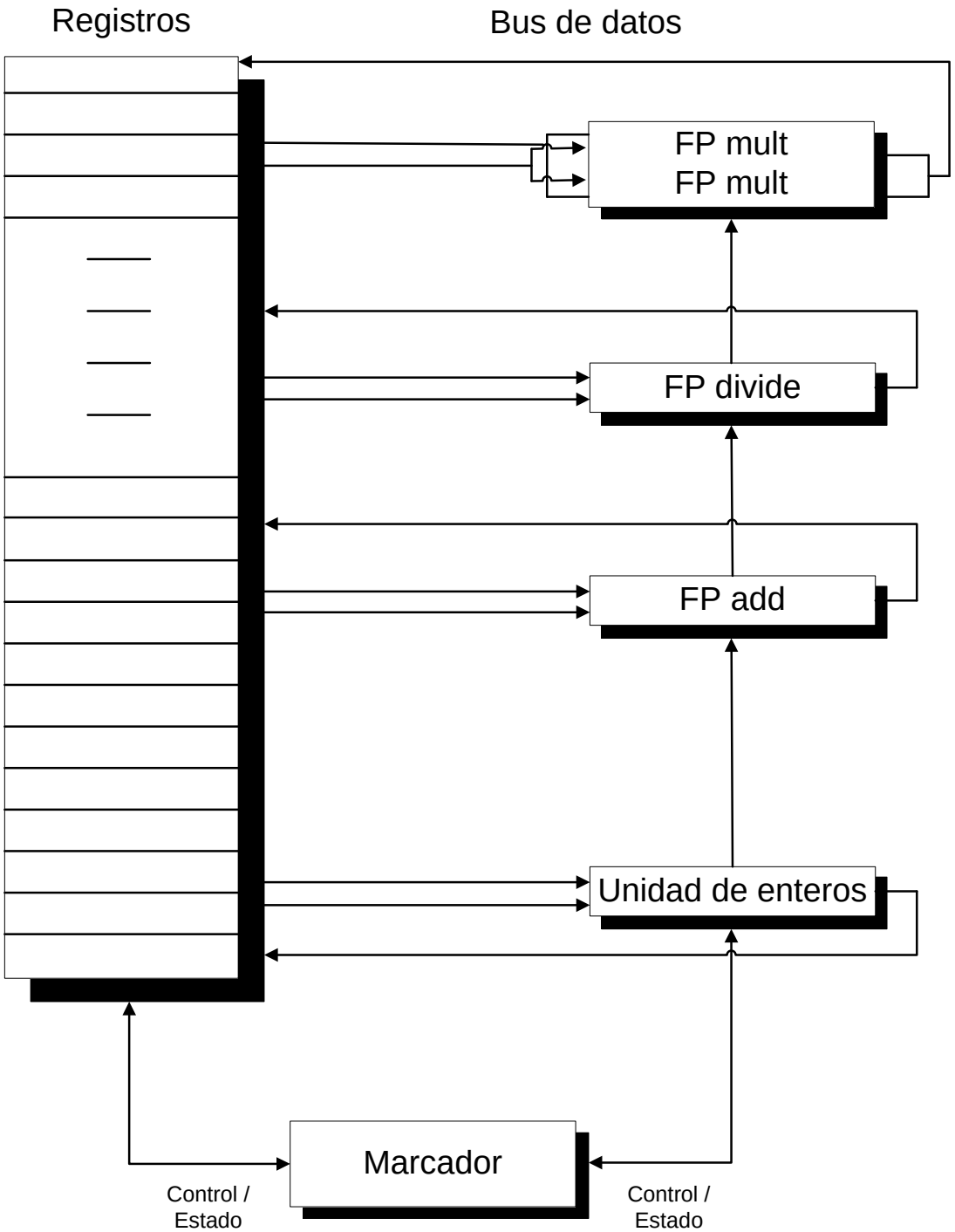
- Objetivo y fundamentos.

En la planificación dinámica, todas las instrucciones pasan el estado de publicación en orden; sin embargo, pueden ser saturadas en el segundo estado *(read operands)* y así ejecutarse en orden inverso. El *Scoreboarding* es una técnica para permitir ejecutar instrucciones de forma desordenada cuando hay suficientes recursos y no hay dependencia de datos.

Antes expliqué la importancia de la planificación dinámica para observar los peligros WAR, que no existían en las tuberías de coma flotante ni en las de los números enteros.

El objetivo del marcador es mantener un ratio de ejecución de una instrucción por ciclo de reloj (cuando no hay errores estructurales) ejecutando una instrucción es posible. Así, cuando la siguiente instrucción a ejecutar esté bloqueada, otras instrucciones pueden ser publicadas y ejecutadas si no hay dependencia en alguna otra instrucción bloqueada. El marcador es el completo responsable para la publicación y ejecución de una instrucción, incluyendo la detección de errores. Como ventaja de la *out-of-order execution* requiere múltiples instrucciones para estar en su estado EX simultáneamente. Esto puede lograrse con múltiples unidades funcionales, con tuberías de unidades funcionales o con ambas. Estas capacidades son esencialmente equivalentes a las propuestas por la tubería de control.

- Estructura básica del MIPS R4000 con marcador



- Etapas que sustituyen a las etapas ID, EX y WB de la segmentación estándar.

Las 4 etapas que sustituyen a las *ID, EX* y *WB* son las siguientes:

1. *Issue* – Si una unidad funcional para la instrucción tiene coste 0 y no hay otra instrucción activa que tenga el mismo registro de destino, el marcador pondrá en marcha la instrucción a la unidad funcional y actualizará su estructura interna de datos. Este paso reemplaza una parte del paso *ID* en las tuberías MIPS. Para asegurarse de que no hay otra unidad funcional activa que quiera escribir el resultado en el registro de destino, garantizaremos que los riesgos de WAW no estarán presentes. Si existiesen los riesgos estructurales, entonces la instrucción publicada se bloquearía y no podrán ser publicadas más instrucciones hasta que este riesgo desaparezca. Cuando el estado de publicación se bloquea, esto causa el buffer entre las instrucciones y su marcha se llene; si el buffer es una entrada simple, la instrucción *fetch* se bloquea inmediatamente. Si el buffer es una cola con múltiples instrucciones, se bloqueará cuando la cola esté completa.

2. *Read operands* – Los marcadores monitorizan la disponibilidad de las fuentes de los operandos. Un operando está disponible si no está en marcha para ser escrito. Cuando los operandos fuente están disponibles, el marcador dice a la unidad funcional que proceda a leer los operandos desde los registros y comience su ejecución. El marcador resuelve los problemas RAW dinámicamente en este paso, y las instrucciones pueden ser enviadas dentro de la ejecución *out of order*. Este paso junto con el primero, completan la función *ID* del MIPS.

3. *Execution* – Esta unidad funcional comienza la ejecución una vez recibe los operandos. Cuando el resultado está listo es notificado al marcador que ha completado la ejecución. Este paso reemplaza el estado *EX* y posee múltiples ciclos en la línea de los coma-flotante de MIPS.

4. *Write result* – Una vez el marcador sabe que la unidad funcional ha completado la ejecución, el marcador prueba los peligros WAR y atasca completando la ejecución, si es necesario.

Un ejemplo en el que existan riesgos WAR podría ser el siguiente:

- Teniendo esta secuencia de código:

DIV.D F0,F2,F4

ADD.D F10,F0,F8

SUB.D F8,F8,F14

- ADD.D posee como registro fuente el F8, que es el mismo registro que tiene como destino en la operación SUB.D. Pero ADD.D depende de una instrucción anterior. El marcador bloqueará el SUB.D en su estado de *Write Result* hasta que ADD.D haya leído el registro F8. Así, en general, una instrucción no estará lista para escribir sus resultados en dos ocasiones principalmente:

1) Cuando hay una instrucción que no ha leído los operandos que preceden completando la instrucción.

2) Y otra ocasión sería, cuando uno de los operandos es el mismo registro que el resultado de una instrucción completada.

- Si no existen problemas WAR, o ya se han solventado, el marcador dice a la unidad funcional que escriba el resultado en el registro de destino. Este paso reemplaza el *WB*.

- Componentes del marcador

El marcador tiene 3 partes:

1) *Instruction status* – Indica en cuál de los 4 estados nos encontramos.
2) *Function unit status* – Indica el estado de la unidad funcional (FU). Hay 9 campos para cada unidad:
    a. *Busy* – Indica si está o no ocupada una unidad.
    b. *Op* – Operación a realizar por la unidad (suma, resta…).
    c. *Fi* – Registro de destino.
    d. *Fj, Fk* – Registros fuente, con los que vamos a operar.
    e. *Qj: Qk* – Unidades funcionales produciendo los registros Fj y Fk.
    f. *Rj, Rk* – Banderas que indican cuándo Fj y Fk están listos para ser leídos. Después de ser leídos los registros, las banderas se ponen en NO.

3) *Register result status* – Indica qué unidad funcional va a escribir cada registro. Este campo tiene el valor *blank* cuando no hay instrucciones pendientes que escriban un registro.

- Ejemplo

Vamos a suponer los siguientes ciclos de latencias para la etapa *EX* para las unidades funcionales de coma flotante: la suma son dos ciclos de reloj, la multiplicación son 10 ciclos, y la división son 40 ciclos. Mostrar las tablas de estados cuando un MUL.D y DIV.D están listas para avanzar en el estado de *Write Result*.

Respuesta:

Existen unos riesgos RAW en los datos en el segundo L.D hacia el MUL.D, ADD.D y SUB.D, desde MULD.D hacia DIV.D, y desde el SUB.D hacia el ADD.D. Hay un riego WAR de datos entre el DIV.D, el ADD.D y el SUB.D. Finalmente, hay un riesgo estructural en la unidad funcional de suma para el ADD.D y el SUB.D.

Estado de las instrucciones

Instrucción		Issue	Read Operands	Execution complete	Write Result
LD	F6,34(R2)	V	V	V	V
LD	F2,45(R3)	V	V	V	
MUL.D	F0,F2,F4	V			
SUB.D	F8,F6,F2	V			
DIV.D	F10.FC,F6	V			
ADD.D	F6,F8,F2	V			

Estado de las unidades funcionales

Nombre	Ocupado	Op	Fl	Fj	Fk	Cj	Ck	Rj	Rk
Entero	Sí	Load	**F2**	**R3**				No	
Multi1	Sí	Mult	**F0**	**F2**	F4	Entero		No	Sí
Multi2	No								
Sumar	Sí	Sub	**F8**	**F6**	F2		Entero	Sí	No
Dividir	Sí	Div	F10	**F0**	F6	Multi		No	Sí

Estado de resultado de registros

	F0	F2	F4	F6	F8	F10	F12		F30
FU	Multi1	Entero			Sumar	Dividir			