*Nota: Si hay algún error sepan disculpar, estuve trabajando en base a la teoría de arquitectura de computadoras del 2023, videos de Genaro Camele y conocimiento propio, cualquier apoyo a mejorar este documento bienvenido es, nos vemos en el parcial del 14/11. Gracias por leer <3*

**Manual para Arquitectura de Computadoras 2023**

Hecho por: San de **La Fuente**

Índices

[1\_ INTRODUCCIÓN A RISC 1](#_Toc149215373)

[2\_ INSTRUCCIONES BÁSICAS WINMIPS64 3](#_Toc149215374)

[3\_ ATASCOS Y CONFIGURACIONES 6](#_Toc149215375)

[4\_ PUNTO FLOTANTE 9](#_Toc149215376)

# 1\_ INTRODUCCIÓN A RISC

**RISC (*Reduced Instruction Set Computer*)**

El hardware se organiza de una manera nueva llamada Segmentación de Cauce (Pipeline), quiere decir que se puede realizar más de una operación al mismo tiempo (¡PARALELISMO! WOW!!)

Las instrucciones se organizan en FASES de manera que esto sea posible (el paralelismo)

Explota el *paralelismo* entre las fases de distintas instrucciones

**Ejecución secuencial**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INSTRUCCIÓN 1** | **->** | **INSTRUCCIÓN 2** | **->** | **INSTRUCCIÓN 3** |

**Ejecución segmentada**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **INSTRUCCIÓN 1** | **IF** | **ID** | **EX** | **MEM** | **WB** |  |  |
| **INSTRUCCIÓN 2** |  | **IF** | **ID** | **EX** | **MEM** | **WB** |  |
| **INSTRUCCIÓN 3** |  |  | **IF** | **ID** | **EX** | **MEM** | **WB** |

**Las instrucciones se organizan en fases**

**[IF]**

**Instruction FETCH**

Se accede a memoria por la instrucción

Se incrementa el PC (Program Counter o antes conocido como Instruction Pointer)

**[ID]**

**Instruction DECODING**

Se decodifica la instrucción

Se accede al banco de registros por los operandos. Se ATASCA si NO están disponibles

Se calcula el valor del operando inmediato

Si es un salto, se calcula el destino y si se toma o no (Requiere acceder el banco de registro)

**[EX]**

**EXECUTION**

Si es una instrucción de cómputo, se ejecuta en la ALU

Si es un acceso a memoria, se calcula la dirección efectiva

Si es un salto, se realiza (Se modifica el registro PC)

**[ME]**

**MEMORY ACCESS**

Si es un acceso a memoria, se lee/escribe el dato

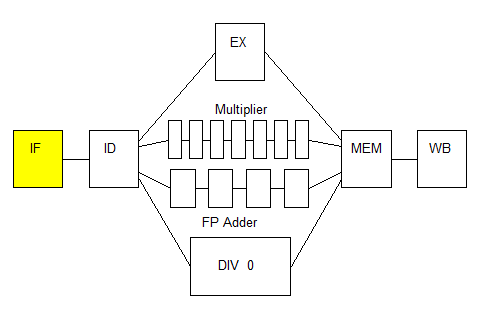
**[WB]**

**WRITE BACK**

Se almacena el resultado (si lo hay) en los registros

*Nota: no todas las etapas duran lo mismo.*

Por ende, no todas pueden manejarse en paralelo (¿No que si se podía?)



**En la etapa de EX tenemos lo siguiente**

Suma: Sale así nomas (está bien)

Multiplicación: Se tarda maso (es mucho más cara que la suma)

División: Se tarda mucho (una piña)

# 2\_ INSTRUCCIONES BÁSICAS WINMIPS64

*Aclaración: DES = Destino, FTE = Fuente, (I) = Índice, Ídem = Igual*

**Las INSTRUCCIONES se dividen en 6 tipos**

Transferencia de Datos

- lb DES, FTE(I) (Copia en DES, un byte [8 BITS] desde la dirección FTE+I, CON SIGNO)

- lbu DES, FTE(I) (Ídem, SIN SIGNO)

- sb FTE, DES(I) (Guarda los 8 bits menos significativos de FTE a DES+I)

- lh DES, FTE(I) (Copia en DES, un half-word [16 BITS], desde la dirección FTE+I, CON SIGNO)

- lhu DES, FTE(I) (Ídem, SIN SIGNO)

- sh FTE, DES(I) (Ídem "sb" pero con 16 bits)

- lw DES, FTE(I) (Copia en DES, un word [32 BITS], desde la dirección FTE+I, CON SIGNO)

- lwu DES, FTE(I) (Ídem, SIN SIGNO)

- sw FTE, DES(I) (Ídem "sb" pero con 32 bits)

- ld DES, FTE(I) (Copia en DES, un double word [64 BITS], desde la dirección FTE+I)

- sd FTE, DES(I) (Ídem "sb" pero con 64 bits)

- l.d DES, FTE(I) (Copia en DES, un punto flotante [64 BITS], desde la dirección FTE+I)

- s.d FTE, DES(I) (Ídem "sb" pero con punto flotante)

- mov.d DES, FTE (Copia el valor del registro FTE al registro DES)

- mtc1 FTE, DES (Copia 64 bits del registro ENTERO FTE al registro FLOTANTE DES) -> INT FTE to FLOAT DES

- mfc1 FTE, DES (Copia 64 bits del registro FLOTANTE FTE al registro ENTERO DES) -> FLOAT FTE to INT DES

- cvt.d.l DES, FTE (Convierte a un FLOTANTE el valor ENTERO de FTE hacia DES) -> INT FTE into FLOAT to DES

- cvt.l.d DES, FTE (Convierte a un ENTERO el valor FLOTANTE de FTE hacia DES) -> FLOAT FTE into INT to DES

Aclaración: DES = Destino, OP1 = Operando 1, OP2 = Operando 2, INM = Inmediato, (±) = Positivo o Negativo

Instrucciones Aritméticas

**Suma**

- dadd DES, OP1, OP2 (Suma OP1 con OP2 y lo deja en DES, valores CON SIGNO) des = (±) op1 + (±) op2

- daddi DES, OP1, INM (Suma OP1 con Valor Inmediato y lo deja en DES, valores CON SIGNO) des = (±) op1 + (±) Inmediato

- daddu DES, OP1, OP2 (Suma OP1 con OP2 y lo deja en DES, valores SIN SIGNO) des = op1 + op2

- daddui DES, OP1, INM (Suma OP1 con Valor Inmediato y lo deja en DES, valores SIN SIGNO) des = op1 + Inmediato

- add.d DES, OP1, OP2 (Suma OP1 con OP2 y lo deja en DES, PUNTO FLOTANTE) des = op1 + op2 (donde op es un número con coma)

**Resta**

- dsub DES, OP1, OP2 (Resta OP2 con OP1 y lo deja en DES, valores CON SIGNO) des = (±) op2 - (±) op1

- dsubu DES, OP1, OP2 (Resta OP2 con OP1 y lo deja en DES, valores SIN SIGNO) des = op2 - op1

- sub.d DES, OP1, OP2 (Resta OP2 con OP1 y lo deja en DES, PUNTO FLOTANTE) des = op2 - op1 (donde op es un número con coma)

**Multiplicación**

- dmul DES, OP1, OP2 (Multiplica OP1 con OP2 y lo deja en DES, valores CON SIGNO) des = (±) op1 \* (±) op2

- dmulu DES, OP1, OP2 (Multiplica OP1 con OP2 y lo deja en DES, valores SIN SIGNO) des = op1 \* op2

- mul.d DES, OP1, OP2 (Multiplica OP1 con OP2 y lo deja en DES, PUNTO FLOTANTE) des = op1 \* op2 (donde op es un número con coma)

**División**

- ddiv DES, OP1, OP2 (Divide OP1 con OP2 y lo deja en DES, valores CON SIGNO) des = (±) op1 / (±) op2

- ddivu DES, OP1, OP2 (Divide OP1 con OP2 y lo deja en DES, valores SIN SIGNO) des = op1 / op2

- div.d DES, OP1, OP2 (Divide OP1 con OP2 y lo deja en DES, PUNTO FLOTANTE) des = op1 / op2 (donde op es un número con coma)

**Comparación (Son las nuevas condiciones de salto)**

- slt DES, OP1, OP2 (Compara OP1 con OP2 solo si OP1 < OP2 -> DES = 1, valores CON SIGNO) si (±) op1 menor (±) op2 entonces des = 1

- slti DES, OP1, INM (Compara OP1 con Valor Inmediato solo si OP1 < INM -> DES = 1, valores CON SIGNO) si (±) op1 menor (±) Inmediato entonces des = 1

- c.lt.d OP1, OP2 (Compara OP1 con OP2, solo si OP1 < OP2 -> flag FP = 1, PUNTO FLOTANTE) si OP1 menor OP2 entonces FP = 1 (donde op son números con coma)

- c.le.d OP1, OP2 (Ídem, solo si OP1 <= OP2 -> flag FP = 1, PUNTO FLOTANTE) si OP1 menor ó igual OP2 entonces FP = 1 (donde op son números con coma)

- c.eq.d OP1, OP2 (Ídem, solo si OP1 = OP2 -> flag FP = 1, PUNTO FLOTANTE) si OP1 igual OP2 entonces FP = 1 (donde op son números con coma)

Instrucciones Lógicas

- and DES, OP1, OP2 (Realiza un AND entre OP1 y OP2 bit a bit y lo deja en DES) des = nbits AND nbits

- andi DES, OP1, INM (Ídem, pero con un valor inmediato) des = nbits AND nbits inmediatos

- or (¿Hace falta que lo explique?)

- ori (Ídem a "or" pero con un valor inmediato)

- xor (¿Esto también?)

- xori (Ídem a "xor" pero con un valor inmediato)

Instrucciones de desplazamiento de bits

- dsll DES, FTE, N (Desplaza a la izquierda N veces de FTE y lo deja en DES)

- dsllv DES, FTE, OP2 (Desplaza a la izquierda OP2 veces de FTE y lo deja en DES)

- dsrl DES, FTE, N (Desplaza a la derecha N veces de FTE y lo deja en DES)

- dsrlv DES, FTE, OP2 (Desplaza a la derecha OP2 veces de FTE y lo deja en DES)

- dsra DES, FTE, N (Ídem “dsrl” pero mantiene el signo del valor desplazado)

- dsra DES, FTE, N (Ídem “dsrl” pero mantiene el signo del valor desplazado)

Instrucciones de Transferencia de Control

- j OffN (Salta a la dirección rotulada offN, offN es una ETIQUETA)

- jal OffN (Salta a la dirección rotulada offN, offN es una ETIQUETA y copia en R31 la dirección de retorno)

- jr DES (Salta a la dirección contenida en DES)

- beq OP1, OP2, offN (Si OP1 = OP2, salta a la dirección rotulada offN, offN es una ETIQUETA)

- bne OP1, OP2, offN (Si OP1 ≠ OP2, salta a la dirección rotulada offN, offN es una ETIQUETA)

- beqz OP1, offN (Si OP1 = 0, salta a la dirección rotulada offN, offN es una ETIQUETA)

- bnez OP1, offN (Si OP1 ≠ 0, salta a la dirección rotulada offN, offN es una ETIQUETA)

- bc1f offN (Salta a la dirección rotulada offN si flag FP=0 (false) en PUNTO FLOTANTE)

- bc1t offN (Salta a la dirección rotulada offN si flag FP=1 (true) en PUNTO FLOTANTE)

Instrucciones de Control

- nop (Operación nula)

- halt (Detiene el simulador)

# 3\_ ATASCOS Y CONFIGURACIONES

*Los ATASCOS se pueden dar en situaciones que impiden a la siguiente instrucción que se ejecute en el ciclo que le corresponde*

Los ATASCOS se dividen en 3

**Estructurales**

Provocados por conflicto de recursos

**Dependencia de control**

Se producen al esperar la decisión de otra instrucción anterior (Por ejemplo: si se realiza un salto o no)

**Dependencia de datos**

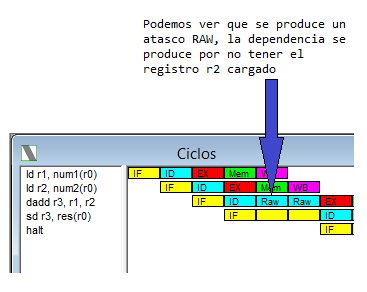
**RAW:** Read After Write

**WAR:** Write After Read

**WAW:** Write After Write

RAW

Se produce cuando se está leyendo un dato que todavía está en la etapa de escritura, es un atasco de **DEPENDENCIA DE DATOS**, al igual que **WAR** y **WAW**



Analizando el siguiente código obtenemos lo siguiente:

.data

num1: .word 10

num2: .word 15

res: .word 0

.code

ld r1, num1(r0) ; CARGO en r1 lo que vale num1(r0), donde r0 me indica el índice

ld r2, num2(r0) ; CARGO en r2 lo que vale num2(r0), donde r0 me indica el índice

dadd r3, r1, r2 ; SUMO lo que vale r1 con r2 y lo dejo en r3

sd r3, res(r0) ; GUARDO lo que hay en r3 y lo dejo en res(r0), donde r0 nuevamente, me indica el índice

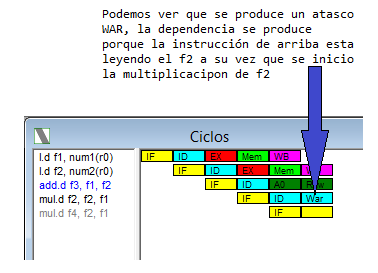
halt ; Instrucción que finaliza el programa

Ahora bien, pasemos a la explicación paso por paso de porque se produce un RAW en la suma:

Cuando yo quiero sumar 2 registros, en esos registros ya debe estar cargado el dato a sumar, en **r1** yo ya tengo el dato a sumar, **r1 = 15**, pero en r2 **todavía** no tengo el 15, como aparece en la imagen, luego de que se termine la etapa WB puedo realizar la instrucción de suma, ahora ¿Qué pasa en la siguiente instrucción? “sd r3, res (r0)”, como r3 **todavía** no tiene el resultado de la suma, se vuelve a producir otro atasco RAW (teóricamente lo solucionaríamos con la configuración **FORWARDING**)

WAR

Se produce cuando se está **ESCRIBIENDO** un dato que se está **LEYENDO** por otra instrucción, es posible que este atasco se produzca mayormente en **PUNTO FLOTANTE**



Analizando el siguiente código obtenemos lo siguiente:

.data

num1: .word 8.5

num2: .word 2.5

res: .word 0

.code

l.d f1, num1(r0) ; CARGO en f1 lo que vale num1(r0)

l.d f2, num2(r0) ; CARGO en f2 lo que vale num2(r0)

add.d f3, f1, f2 ; SUMO lo que vale f1 con f2 y lo dejo en f3 en PUNTO FLOTANTE

mul.d f2, f2, f1 ; MULTIPLICO lo que vale f2 con f1 y lo dejo en f2 en PUNTO FLOTANTE

mul.d f4, f2, f1 ; MULTIPLICO lo que vale f2 con f1 y lo dejo en f4 en PUNTO FLOTANTE

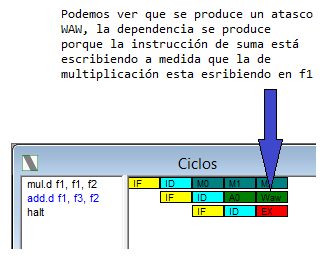
s.d f3, res(r0) ; GUARDO lo que hay en f3 y lo dejo en res(r0)

halt ; Instrucción que finaliza el programa

El WAR (Write After Read) se produce cuando la instrucción que **ESTA LEYENDO** un dato, a su vez está siendo **MODIFICADO**, por lo tanto, la instrucción para por un momento su ejecución con este atasco y permite que la instrucción anterior pueda ejecutarse sin problemas.

WAW

Se produce cuando se está **ESCRIBIENDO** un dato que se está **ESCRIBIENDO**, lo que produce un atasco WAW (Write After Write)



Analizando el siguiente código obtenemos lo siguiente:

.data

.code

mul.d f1, f1, f2 ; MULTIPLICO f1 con f2 y lo dejo en f1 (PUNTO FLOTANTE)

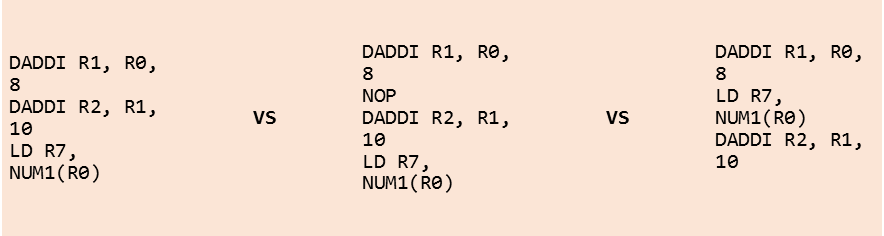
add.d f1, f3, f2 ; SUMO f3 con f2 y lo dejo en f1 (PUNTO FLOTANTE)

halt ; Instrucción de fin

Tanto la instrucción “mul.d” y “add.d” tienen más o menos ciclos (la multiplicación se tarda más que la suma), en este caso, la suma tendría que terminar **ANTES** que la multiplicación, entonces, se sobrescribe el valor de f1 con el de la **suma** y se lo pasaría a la **multiplicación** PERO, en este caso se produce un atasco **WAW** lo que permite que se ejecute *primero* la multiplicación y así ejecutar en orden correcto las instrucciones

**¿Cómo solucionar ATASCOS?**

**Por software**



**Con NOPs Ordenando sentencias**

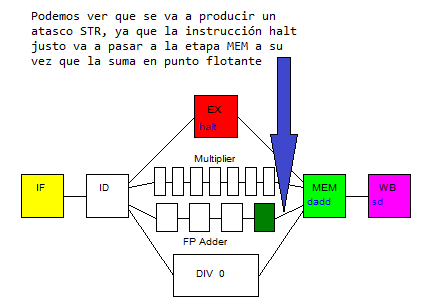
**Por hardware**

Si ya tenemos los valores necesarios, podemos “adelantarlos” con la opción de Forwarding.

**Atascos – Estructurales**

Los atascos **estructurales** son provocados por conflictos por los recursos

En MIPS, sucede cuando dos instrucciones intentan acceder a la etapa **MEM** simultáneamente



Cuando dos instrucciones van a parar a la misma etapa, se produce un atasco de tipo STR, lo que permite que una instrucción se ejecute y que **LUEGO** pase la siguiente instrucción

Vemos que donde está la flecha apunta a la etapa EX del cauce “FP Adder” que sería la suma en punto flotante, cuando termina de sumar, también termina de ejecutar el “halt”, y se produce un atasco STR para evitar que ambas se encuentren en la misma etapa

**Atascos – Dependencia de control**

Tenemos dos tipos de saltos

**Incondicionales:** Salta siempre

**Condicionales:** Salta dependiendo de que se cumpla una condición

**Configuraciones**

**Forwarding**

**BTB** (Branch Target Buffer)

**Delay Slot**

Forwarding

Ahora tendremos dos buffers entre antes y despues de la etapa MEM, se almacenan los valores para que los puedan usar en las próximas instrucciones

De esta manera no hace falta esperar a las etapas **MEM** y **WB** para usar los valores, a esta configuración se le llama Forwarding, además con esta opción habilitada, podremos postergar la “necesidad” de los operandos.

Branch Target Buffer

Al igual que con los atascos de dependencia de datos, tenemos diferentes técnicas para evitar los atascos por saltos

Consiste en tener un flag que indicaue si debe saltar incondicionalmente o no dependiendo de qué hizo antes (es decir, predice)

Cada vez que ese flag/buffer se actualiza cuenta como un *atasco de salto*, cada vez que **erramos** en la predicción se produce un atasco

Delay Slot

Consiste en ejecutar **SIEMPRE** la siguiente instrucción a un salto, el problema de esta opción es cuando tenemos un salto y luego una instrucción “halt”, lo que produce es que se ejecute el “halt” antes que el salto y termine el programa, usarlo sabiamente

# 4\_ PUNTO FLOTANTE

MIPS utiliza IEEE 754 para números en punto flotante (en el caso de los registros R, utiliza BSS o CA2 para números con signo)

Contamos con 32 registros: Desde el F0 (siempre vale 0) hasta el F31 (F0 – F31) igual que con los registros de tipo R (R0 – R31)

Para declarar un tipo flotante se utiliza el “.double” (antes usábamos .word y asi)

Ejemplo:

.data

A: .double 8.10

.code

...

Ejemplos de instrucciones

*Nota: Recuerden que las posiciones siempre son enteras, yo no puedo acceder a la posición 3,5, súper ilegal eso.*

- Carga L.D F1, NUM1(R0) (Load.Double de F1 en NUM1(0))

- Suma ADD.D F1, F2, F3 (Add.Double de F2 + F3 en F1)

- Resta SUB.D F1, F2, F3 (Sub.Double de F3 - F2 en F1)

- Multiplicación MUL.D F1, F2, F3 (Mul.Double de F2 \* F3 en F1)

- División DIV.D F1, F2, F3 (Div.Double de F2 / F3 en F1)

- Almacenamiento S.D F1, RES(R0) (Store.Double de F1 en RES(0))

Ejemplo de un código en PUNTO FLOTANTE:

RES = (A+B)\*C (¿Cómo implementamos esto en código?)

.data

A: .double 5.5

B: .double 8.6

C: .double 10.0

RES: .double 0.0

.code

L.D F1, A(R0)

L.D F2, B(R0)

L.D F3, C(R0)

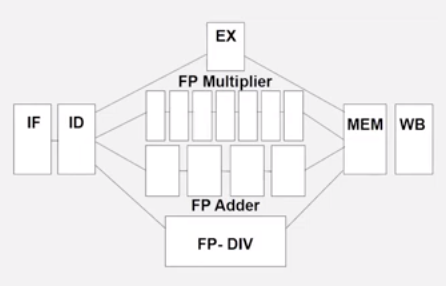
ADD.D F1, F1, F2

MUL.D F1, F1, F3

S.D F1, RES(R0)

HALT

Recordar que **no todas las etapas duran lo mismo**



**Etapa EX = 1 ciclo**

**Sumar en Pto. F. = 4 ciclos**

**Multiplicar en Pto. F. = 7 ciclos**

**Dividir en Pto. F. = 24 ciclos**