

Formatos de instrucción en RISC-V:

Determinan la ubicación y codificación de los campos de una instrucción máquina.

Tienen los siguientes campos:

- (1) Código de operación (op): Indica el tipo de instr.
- (2) Código de función (func3 y func7):
Determina la instrucción concreta dentro del tipo
- (3) Operando de registro (rs1, rs2, rd): n° de registro
- (4) Operando inmediato (imm): valor inmediato que se extiende 20 bits

Tipos de formatos

- (1) Tipo-R: Aritmético-lógicas y desplazamiento
- (2) Tipo-I: Aritmético-lógicas y desplazamiento con intervención de inmediatos
- (3) Tipo S/B: Almacenamiento(S) y salto(B)
- (4) Tipo U/J: lui(U), auipc(U) y jal(J)

Todos tienen una anchura de 32 bits

El rendimiento de un ordenador se mide en el tiempo que tarda en ejecutar programas, lo cual depende de:

- (1) N° de instrucciones \Leftrightarrow programador
 - (2) N° de ciclos que tarda cada instrucción
 - (3) Tiempo de ciclo (frecuencia de reloj)
- \Leftrightarrow HW

Cualquier procesador está formado por:

- ↳ **Ruta de datos**: realiza operaciones y almacena los resultados
- ↳ **Controlador**: Secuencia la realización de las transferencias entre registros para cada instrucción.

Según el diseño tendremos:

- a) **Procesadores monociclo**: todas las transferencias se realizan en 1 ciclo de reloj
- b) **Procesador multiciclo**: las transferencias se reparten entre varios ciclos de reloj

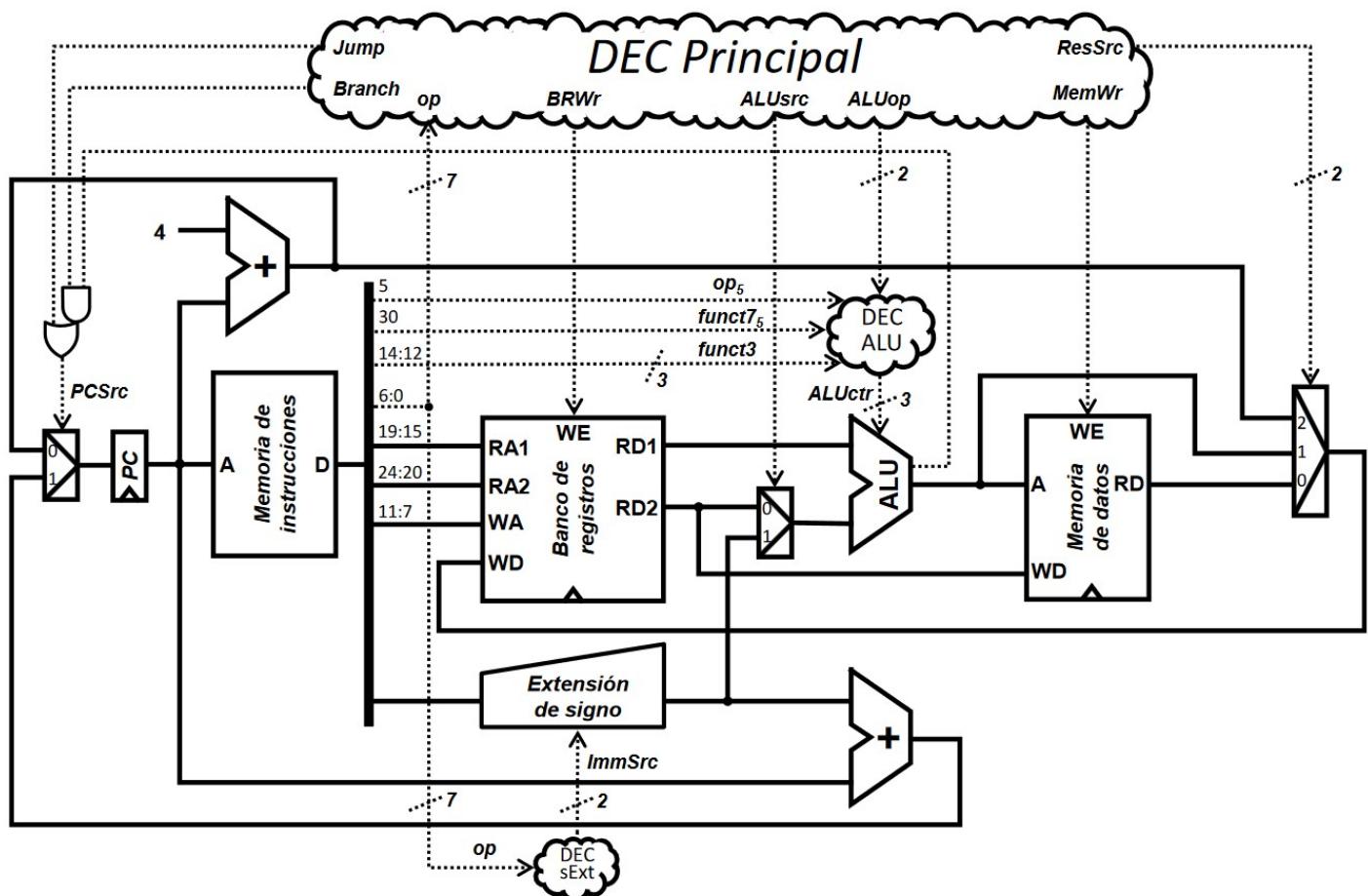
Procesador Monocubo

Diseño de la ruta de datos:

(1) Almacenamiento: Memoria (Memoria de instrucciones y memoria de datos), PC y banco de registros

(2) Elementos funcionales: ALU (operaciones aritmético-lógicas), extensor de signo, sumador adicional (para incrementar el PC), otro sumador adicional (para instrucciones de salto *jal* y *beq*)

Diseño del controlador: Es un circuito combinacional formado por 4 subcircuitos



ALUop	op₅	funct7₅	funct3	ALUctr
00 ^(sumar)	X	X	XXX	000 ^(A + B)
01 ^(restar)	X	X	XXX	001 ^(A - B)
10 ^(operar)	0	X	000 ^(addi)	000 ^(A + B)
10 ^(operar)	1	0	000 ^(add)	000 ^(A + B)
10 ^(operar)	1	1	000 ^(sub)	001 ^(A - B)
10 ^(operar)	X	X	010 ^(slt/slti)	101 ^(A < B)
10 ^(operar)	X	X	110 ^(or/ori)	011 ^(A B)
10 ^(operar)	X	X	111 ^(and/andi)	010 ^(A & B)

Op	ImmSrc
0000011 ^(lw)	00 ^(tipo-I)
0100011 ^(sw)	01 ^(tipo-S)
0010011 ^(tipo-I)	00 ^(tipo-I)
0110011 ^(tipo-R)	—
1100011 ^(beq)	10 ^(tipo-B)
1101111 ^(jal)	11 ^(tipo-J)

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 ^(lw)	0	0	1	1	00 ^(sumar)	0	00
0100011 ^(sw)	0	0	0	1	00 ^(sumar)	1	—
0010011 ^(tipo-I)	0	0	1	1	10 ^(operar)	0	01
0110011 ^(tipo-R)	0	0	1	0	10 ^(operar)	0	01
1100011 ^(beq)	1	0	0	0	01 ^(restar)	0	—
1101111 ^(jal)	0	1	1	—	—	0	10

Problemas de los procesadores monociclo:

(1) Todas las instrucciones tardan lo mismo independientemente de su complejidad
(1 ciclo = tiempo de la instrucción + lenta)
Es decir, se desperdicia tiempo en las instrucciones más rápidas

(2) No es posible reutilizar HW:
Requiere de HW que ya existe en el circuito, pero que no se puede reutilizar

Procesador Multi ciclo

El procesador multiciclo trata de solventar los problemas del monociclo dividiendo en varias etapas la ejecución de una instrucción.

En este caso 1 ciclo = El tiempo que tarda la instrucción + larga

Cada instrucción tarda un tiempo distinto en ejecutarse, proporcional a su complejidad

Además, se puede reusar hardware (siempre que su uso esté en distintos ciclos)

Diseño de la ruta de datos:

(1) Elementos de almacenamiento:

Memoria unificada de datos e instrucciones, banco de registros y PC (no se actualiza en cada ciclo)

(2) Elementos adicionales: ALU, extensor de signo, sin sumadores adicionales

Diseño del controlador: Circuito secuencial formado por 4 subcircuitos.

La ejecución de una instrucción se divide en:

(1) Búsqueda en memoria de la instrucción (IF)

(2) Decodificación y obtención de los operandos (ID)

(3) Ejecución o cálculo de dirección (Ex)

(4) Acceso a memoria de datos (MEM)

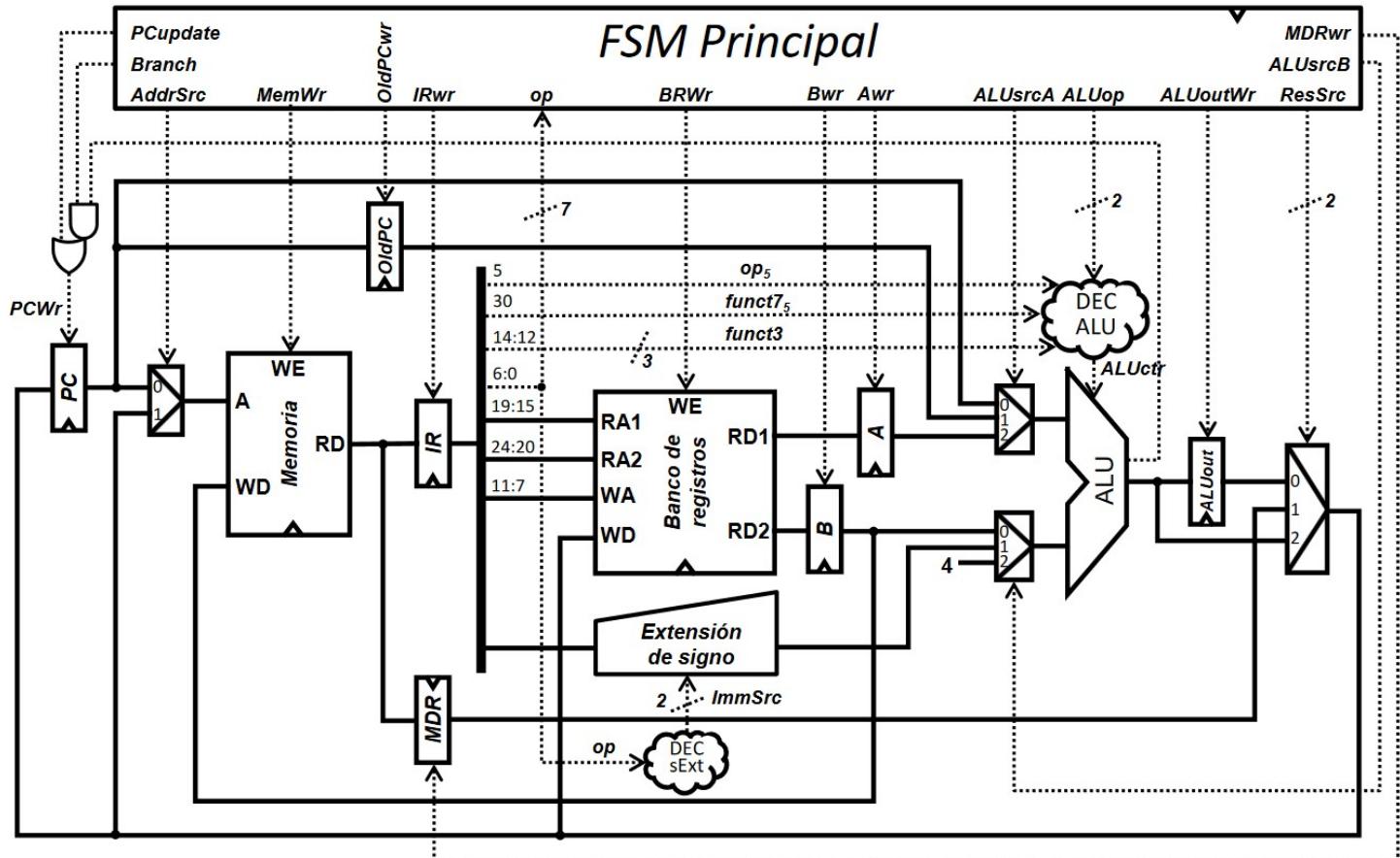
(5) Escritura del resultado (WB)

El tiempo que tarda cada instrucción es:

lw : 5 ciclos

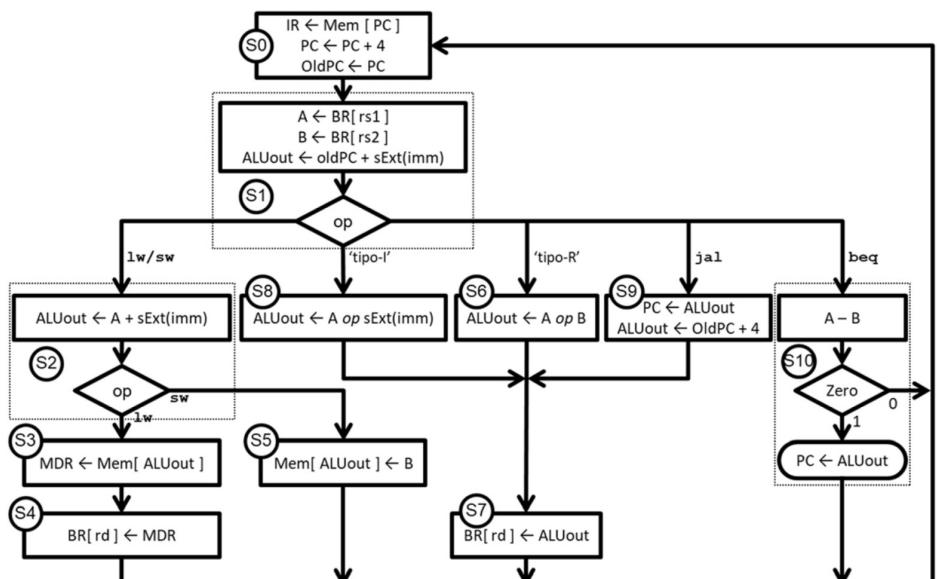
sw/jal/aritméticas lógicas : 4 ciclos

beq : 3 ciclos

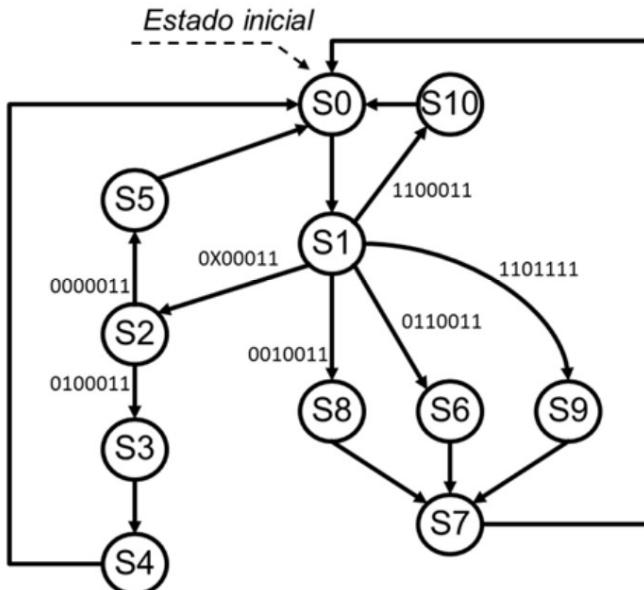


ALUop	op _s	funct7 ₅	funct3	ALUctr
00 (sumar)	X	X	XXX	000 ($A + B$)
01 (restar)	X	X	XXX	001 ($A - B$)
10 (operar)	0	X	000 (addi)	000 ($A + B$)
10 (operar)	1	0	000 (add)	000 ($A + B$)
10 (operar)	1	1	000 (sub)	001 ($A - B$)
10 (operar)	X	X	010 (slt/slti)	101 ($A < B$)
10 (operar)	X	X	110 (or/ori)	011 ($A \mid B$)
10 (operar)	X	X	111 (and/andi)	010 ($A \& B$)

Op	ImmSrc
0000011 (lw)	00 (tipo-I)
0100011 (sw)	01 (tipo-S)
0010011 (tipo-I)	00 (tipo-I)
0110011 (tipo-R)	-
1100011 (beq)	10 (tipo-B)
1101111 (jal)	11 (tipo-J)



Función de transición de estados



estado	op	estado'
S0	XXXXXXX	S1
S1	0X00011 (lw/sw)	S2
S1	0010011 (tipo-I)	S8
S1	0110011 (tipo-R)	S6
S1	1101111 (jal)	S9
S1	1100011 (beq)	S10
S2	0000011 (lw)	S3
S2	0100011 (sw)	S5
S3	XXXXXXX	S4
S4	XXXXXXX	S0
S5	XXXXXXX	S0
S6	XXXXXXX	S7
S7	XXXXXXX	S0
S8	XXXXXXX	S7
S9	XXXXXXX	S7
S10	XXXXXXX	S0

Función de salida

estado	Branch	PCupdate	Addrsrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	AluOp	AluOutWr	Ressrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	-	0	0	0	0	0	1	1	01	01	00	1	-
S2	0	0	-	0	0	0	0	0	0	0	10	01	00	1	-
S3	0	0	1	0	0	0	1	0	0	0	-	-	-	0	00
S4	0	0	-	0	0	0	0	1	0	0	-	-	-	0	01
S5	0	0	1	1	0	0	0	0	0	0	-	-	-	0	00
S6	0	0	-	0	0	0	0	0	0	0	10	00	10	1	-
S7	0	0	-	0	0	0	0	1	0	0	-	-	-	0	00
S8	0	0	-	0	0	0	0	0	0	0	10	01	10	1	-
S9	0	1	-	0	0	0	0	0	0	0	01	10	00	1	00
S10	1	0	-	0	0	0	0	0	0	0	10	00	01	0	00

Comparativa

Procesador monociclo

1 ciclo / instrucción

Tiempo de ciclo largo

Todos los recursos están enfocados en una sola operación

Memorias separadas



Es más barato (es más barato añadir sumadores)

Mejor en arquitecturas simplificadas

Procesador multiciclo

Más de 1 ciclo por instrucción

Tiempo de ciclo corto

Los recursos pueden reutilizarse para hacer operaciones en distintos ciclos de reloj

Memorias unificadas

Requiere usar registros intermedios, invisibles al programador



Es más caro (añadir multiplexores, registros, etc para reutilizar la AW)

Pior rendimiento

Mejor en operaciones más complejas

Métricas de rendimiento: Permiten comparar objetivamente las prestaciones de distintos computadores.

Tiempo de ejecución:

$$t_{ejec} = t_{cic} \cdot \sum c_i \cdot n_i = \sum c_i \cdot \frac{n_i}{f_{cic}} \Rightarrow$$

tiempo de ciclo → n° de instrucciones del tipo i
 ↓ → frecuencia de reloj
 n° de ciclos que tarda en ejecutarse la instrucción de tipo i

Ciclos promedio por instrucción: Número de ciclos que tarda por separado cada tipo de instrucción

$$CPI = \frac{\sum c_i \cdot n_i}{\sum n_i} \rightarrow$$

→ n° de ciclos por instrucción i
 → n° de instrucciones del tipo i
 ↓ → n° de instrucciones totales

$$t_{ejec} = N \cdot CPI \cdot t_{cic} = \frac{N \cdot CPI}{f_{cic}}$$

$$\text{con } N = \sum n_i \Rightarrow \text{nº total de instrucciones}$$

Millones de instrucciones por segundo:

$$MIPS = \frac{N}{10^6 \cdot t_{ejec}} = \frac{f_{cic}}{10^6 \cdot CPI}$$

Millones de instrucciones de punto flotante por sec.: Se escogen por ser las que más tardan en ejecutarse

Speedup: Relación entre los tiempos de ejecución de un mismo programa entre 2 procesadores A y B

$$\text{speedup} = \frac{t_{\text{ejec. B}}}{t_{\text{ejec. A}}}$$

Ejemplo:

Sea un programa que genera 10^8 instrucciones tal que:

- 25% lw
- 2% jal
- 10% sw
- 52% aritmético-lógicas
- 11% beg

Procesador monociclo:

$$CPI = 1$$

$$t_{\text{ejec.}} = 276 \cdot 10^{-8} \cdot 1 = 276 \text{ s}$$

$$MIPS = \frac{10^8 \cdot 2}{10^8 \cdot 276} = 32.6 \text{ inst/s}$$

Procesador multiciclo:

$$CPI = [(0.25 \cdot 10^8) \cdot 5 + (0.10 \cdot 10^8) \cdot 4 + (0.11 \cdot 10^8) \cdot 3 + (0.02 \cdot 10^8) \cdot 4 + (0.52 \cdot 10^8) \cdot 4] \cdot \frac{1}{10^8} = 4.14$$

$$t_{\text{ejec.}} = 0.8 \cdot 10^8 \cdot 4.14 = 4.06 \text{ s}$$

$$MIPS = \frac{10^8}{10^8 \cdot 4.06} = 24.63$$

$$\text{Speedup} = \frac{t_{\text{multiciclo}}}{t_{\text{monociclo}}} = \frac{4.06}{276} = 1.47$$

Procesador Segmentado

Un procesador segmentado (pipelined) solapa la ejecución de varias instrucciones, en cada ciclo se lanza una nueva instrucción sin esperar a que la anterior termine.

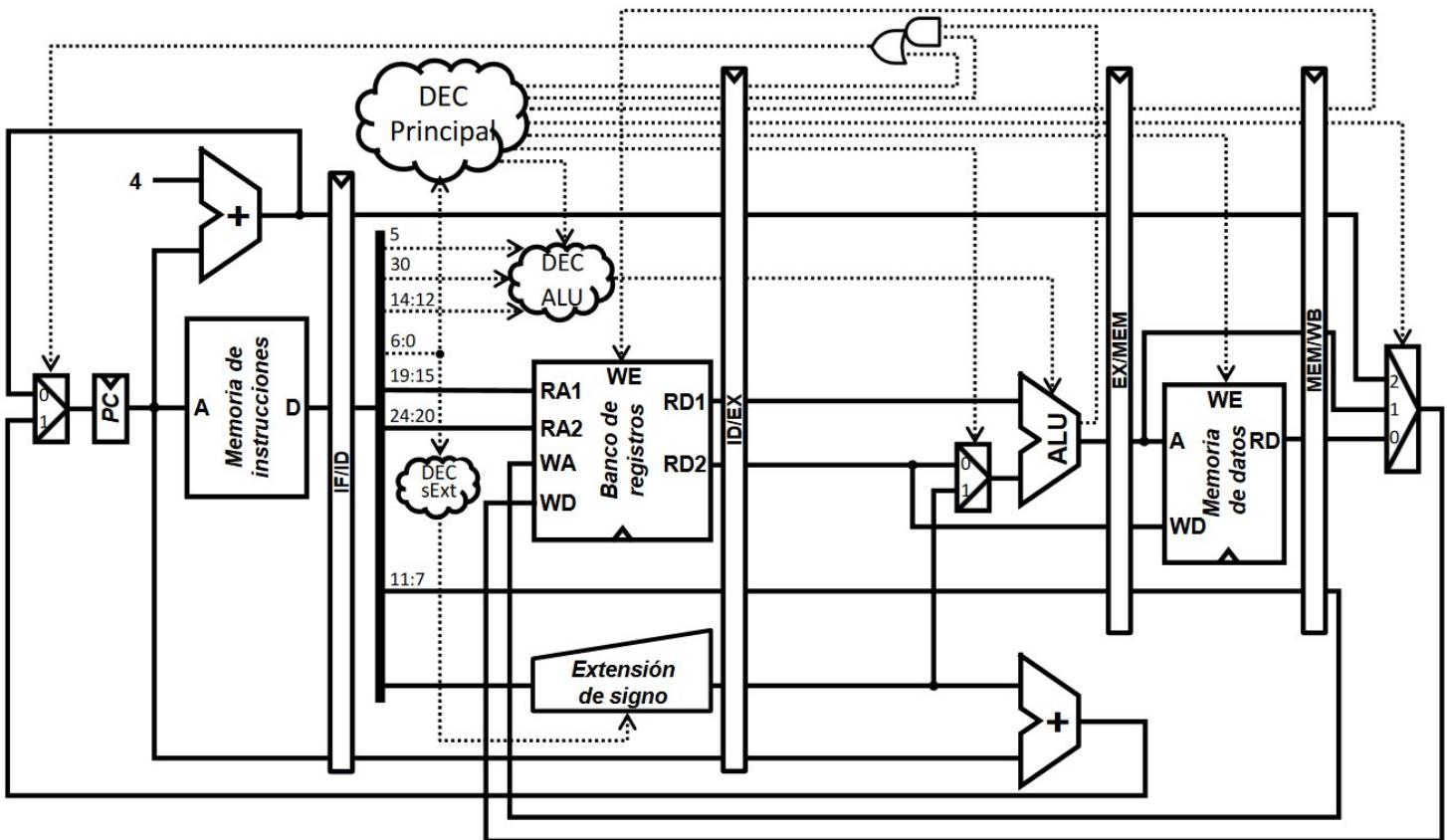
El tyc será menor ya que:

- Varias instrucciones se ejecutan simultáneamente
- El tiempo de ciclo podrá ser corto, como en los multiciclos
- Idealmente $CPI = 1$ como en los monociclos. De forma generalizada un procesador segmentado tiene:

$$CPI = \frac{5 \cdot N}{N} \xrightarrow{N \rightarrow \infty} 1$$

Cada instrucción pasa por 5 etapas tardando 5 ciclos en ejecutarse (IF, ID, EX, MEM, WB)

Además para guardar el resultado de cada etapa de cada instrucción se añaden bancos de registros.



ALUop	op_5	$funct7_5$	$funct3$	ALUctr
00 ^(sumar)	X	X	XXX	000 ^(A + B)
01 ^(restar)	X	X	XXX	001 ^(A - B)
10 ^(operar)	0	X	000 ^(addi)	000 ^(A + B)
10 ^(operar)	1	0	000 ^(add)	000 ^(A + B)
10 ^(operar)	1	1	000 ^(sub)	001 ^(A - B)
10 ^(operar)	X	X	010 ^(slt/slti)	101 ^(A < B)
10 ^(operar)	X	X	110 ^(or/ori)	011 ^(A B)
10 ^(operar)	X	X	111 ^(and/andi)	010 ^(A & B)

Op	ImmSrc
0000011 ^(lw)	00 ^(tipo-I)
0100011 ^(sw)	01 ^(tipo-S)
0010011 ^(tipo-I)	00 ^(tipo-I)
0110011 ^(tipo-R)	-
1100011 ^(beq)	10 ^(tipo-B)
1101111 ^(jal)	11 ^(tipo-J)

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 ^(lw)	0	0	1	1	00 ^(sumar)	0	00
0100011 ^(sw)	0	0	0	1	00 ^(sumar)	1	-
0010011 ^(tipo-I)	0	0	1	1	10 ^(operar)	0	01
0110011 ^(tipo-R)	0	0	1	0	10 ^(operar)	0	01
1100011 ^(beq)	1	0	0	0	01 ^(restar)	0	-
1101111 ^(jal)	0	1	1	-	-	0	10

En los procesadores segmentados debido a que varias instrucciones se realizan simultáneamente pueden ocurrir **conflictos**.

Tipos de conflictos:

- 1) **Estructurales**: Una instrucción necesita usar MN que está siendo usado por otra instrucción.
- 2) **De datos**: Una instrucción necesita leer de un registro que todavía no contiene la información de la instrucción anterior.
- 3) **De control**: Para instrucciones de salto como `beq` ó `jal`, al ser un procesador segmentado ocurre que se producen las siguientes instrucciones antes de que se decida si se salta o no.

Para resolver conflictos de todo tipo existe la **inserción de nops** por software entre la instrucción que escribe en el registro y la que lo lee, e insertando 2 instrucciones `nop` tras instrucciones de salto. Aunque tiene **inconvenientes** complica la programación y la ejecución de cada `nop`, penaliza el tiempo de ejecución en el ciclo.

Para resolver conflictos de datos existe la anticipación, que consiste en recoger los datos necesarios, de las etapas posteriores en lugar de cargarlos de la memoria.

La unidad de anticipación es un circuito combinacional que controla los multiplexores previos a la ALU, para que ésta opere:

- con datos del banco de registros ó
- con datos actualizados en etapas posteriores de sus respectivos registros de segmentación EX/MEM ó MEM/WB

Para ello se añaden otras variables de control, las cuales actuarán de manera que si el dato necesario se necesita anticipar, lo cargue desde la etapa MEM o WB.

Aún así, existe un caso especial en el que la anticipación no funciona: cuando a una instrucción LW que carga en un registro, le sigue otra instrucción que lee de ese mismo registro. Y es que no se puede anticipar porque LW lee el dato de memoria en el ciclo 4, el mismo en el que la siguiente instrucción necesita el dato. La solución consiste en reditizar una parada.

La unidad de conflictos es un circuito combinacional que en todo ciclo determina si las etapas IF e ID del procesador deben pararse.

Se activa cuando:

- Hay una instrucción lw en la etapa EX
- El registro destino de la etapa EX coincide con alguno de los registros de origen de la etapa ID.

Si se produce el parado, se inhabilita la carga del PC y del registro de segmentación SP/TF y se borra el de ID/EX.

No obstante, este sistema provoca en ocasiones paradas innecesarias:

- cuando uno de los registros es el XO
- cuando una instrucción sw sigue a una lw

Para evitar el 2º caso se puede añadir un multiplexor en la entrada de los datos de memoria para anticipar datos del registro de segmentación MEM/WB o redesignar las unidades de anticipación o conflictos.

Una solución más práctica y sencilla es reordenar el código, ya sea el programador o el compilador.

En el caso de las instrucciones de salto (beq) las paradas resultan ineficaces pues siempre la ejecución se verá penalizada 2 ciclos.

En este caso será mejor predecir el salto, de manera que comiencen las instrucciones siguientes al salto y así:

- 1) si no se da el salto, no hay penalización
- 2) si se da el salto, habrá una penalización de 2 ciclos.

En el caso de la instrucción jal, siempre habrá penalización.

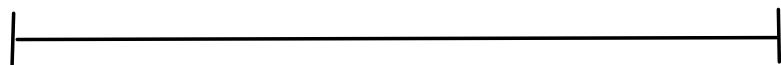
Para añadir esta función es necesario modificar la unidad de conflictos, para añadir variables de control y borrar el registro de segmentación IF/ID además, del ID / EX

Resumen:

Tipo de Conflicto	Instrucciones implicadas	Penalización (ciclos)		Solución HW implementada
		SW	HW	
Estructural Datos	(no existe este tipo de conflicto)	-	-	-
	tipo add/i → resto	1, 2 ó 3	0	anticipación
	lw → tipo add/i	1, 2 ó 3	1	parada + anticipación (evitable reordenando código)
	lw → lw	1, 2 ó 3	1	parada + anticipación (evitable reordenando código)
Control	lw → sw	1, 2 ó 3	1	parada + anticipación (evitable optimizando anticipación)
	beq	2	0 ó 2	predicción de salto
	jal	2	2	predicción de salto

Procesador superscalar: Procesador con varias copias de la ruta de datos: Ejecuta en paralelo varias instrucciones del mismo programa. No obstante, la probabilidad de conflicto es mayor. Además tiene duplicado casi todo el HW. CPI ideal = $1/2$.

Procesador multicore: Procesador con varias copias del procesador completo, de manera que puede ejecutar simultáneamente varios programas y comparten memoria.



Trap: suceso imprevisto que interrumpe la ejecución del programa.

↳ **Excepción**: la causa es interna al procesador.

↳ **Interrupción**: la causa es externa al procesador.

Cuando un procesador detecta una trap, salta a una dirección donde existe una junión tras la que el procesador aborta el programa o lo continua. Estas junciones suelen formar parte del sistema operativo; Rutina de tratamiento de excepción o interrupción.

Para que las rutinas de tratamiento sean eficaces se debe conocer el tipo de trap. Para ello existen 2 métodos:

- 1) **Directo**: El procesador almacena un valor distinto para cada trap en un registro especial y salta a una dirección común para redirigir una rutina común la cual lee el tipo de trap y redirige lo más propio.
- 2) **Vectorizado**: el procesador salta a una dirección distinta según el tipo de trap
⇒ Existen tantas rutinas como traps.

Un RISC-V posee hasta 4096 registros de control y estados para controlar el sistema son distintos a los convencionales y necesitan de privilegios para copiarlos en registros usables para poder operar con ellos.

Los principales registros de control y estados son:

- ↳ **mtvec**: Dirección de la rutina a redirigir
- ↳ **mcause**: Identificador de la causa
- ↳ **mepc**: Dirección de la instrucción que ha provocado la excepción
- ↳ **mtval**: Dirección de memoria que ha causado la excepción.

En el caso de haber un trap, un RISC-V, actúa de la siguiente manera:

- 1) Cancela la ejecución
- 2) Salva la dirección en mepc
↳ si la excepción se produce al intentar acceder a un dato en memoria se guarda la misma en mtvec
- 3) Salva un código de identificación de la excepción en mcause
- 4) Salta a la dirección guardada en mtvec.

En este proceso, además, también apila en memoria los registros de los que haga uso la rutina de excepción. Luego lee mcause y actúa en consecuencia.

Si la excepción permite continuar la ejecución se desapilan los registros y se ejecuta mret para volver al programa.

Un RISC-V, puede operar a distintos niveles de privilegio llamados modos, que determinan qué instrucciones puede realizar un procesador: Existen 3 - primordiales: el modo usuario, modo máquina y modo supervisor.

Existen instrucciones para tratar las excepciones que requieren de privilegios, un estatus, el cual viene dado por el registro mstatus. Las instrucciones de los CSR son todas de tipo I.

Debido a que hay que tener en cuenta todas las excepciones y posibilidades, hay que realizar modificaciones al circuito.

En el caso de los procesadores segmentados, las excepciones sólo saltan cuando se sitúan en la etapa MEM, a pesar de que se hayan dado los errores en una etapa anterior.

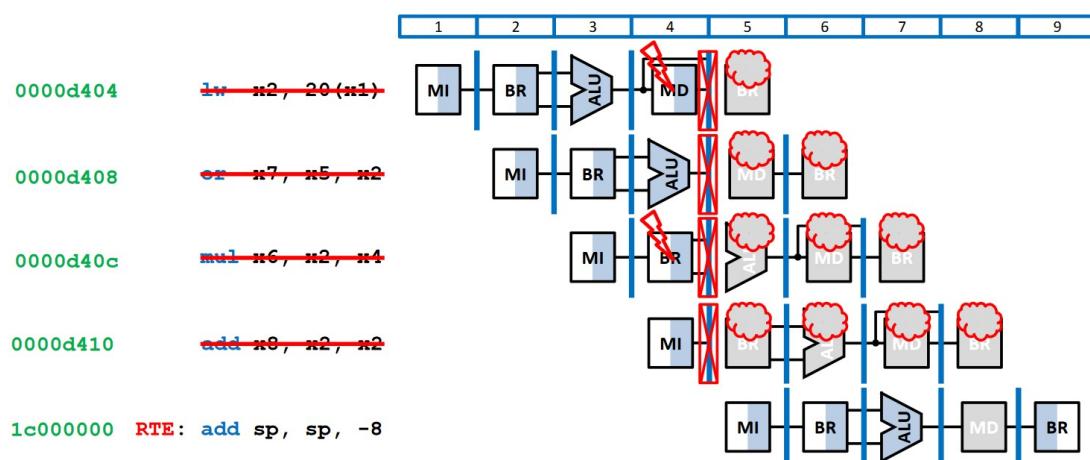
Así, por ejemplo:

- 1) Si la excepción se da en ID, no se paraá el programador hasta MEM y se guardará & en cause.
- 2) En un mismo ciclo pueden darse varias excepciones siendo aquella que esté más arriba la que se considerará como responsable.

- Ciclo 4: las instrucciones **lw** (está en MEM) y **mul** (esta en ID) generan excepciones. Se **descarta lw** y **todas las que le siguen**.

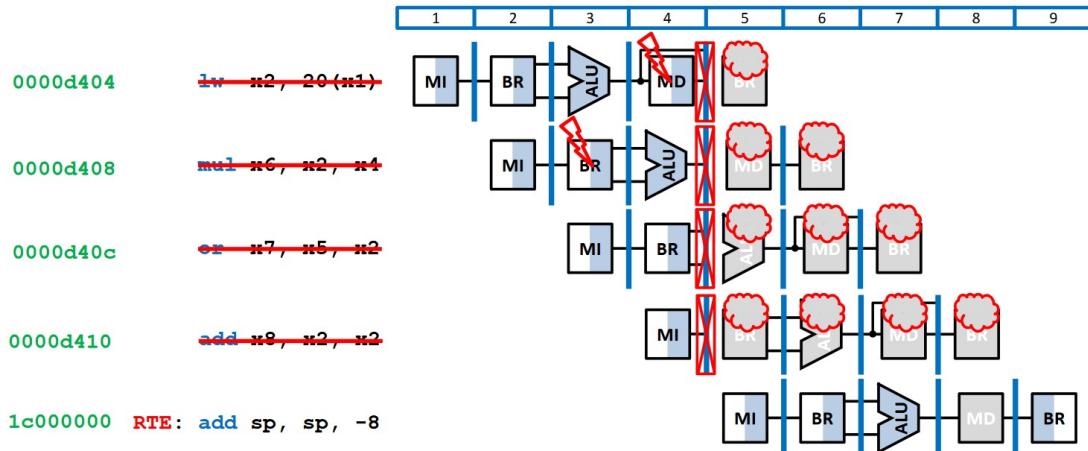
- mepc = 0x0000d404 y cause = 4

- Ciclo 5: se lanza la instrucción cuya dirección almacena **mtvec**.



3) Si se produce un error antes que otro, pero el 2º error procede de una instrucción anterior, se guardará esa como responsable

- Ciclo 3: `mul` (está en ID) genera excepción.
- Ciclo 4: `lw` (está en MEM) genera excepción después que `mul` aún siendo una instrucción previa. Se **descarta** `lw` y **siguientes**.
 - `mepc = 0x0000d404` y `cause = 4`
- Ciclo 5: se lanza la instrucción cuya dirección almacena `mtvec`.



En resumen, los procesadores segmentados, generalmente, replican lo que ocurre en los no segmentados:

- 1) se trata 1º la excepción de la instrucción que esté antes
- 2) se completan las instrucciones anteriores a la de la excepción
- 3) cortan la instrucción siguiente de la excepción, y sus posteriores
- 4) Se almacena con exactitud la dirección y causa de la excepción.

En aquellos procesadores segmentados que no se dan estas características, se dice que tiene **excepciones imprecisas**.

la instrucción mret (para volver a la función en la que estaba el procesador antes de la excepción) actúa como cualquier otra instrucción de salto, sin usar memoria en las etapas MEM y WB y sin usar la ALU en la EX.

La instrucción csrrw (intercambia un registro de control y estatus con un registro convencional) actúa sin usar la ALU en EX, y la memoria en MEM.

Estas instrucciones generan nuevos conflictos:

- 1) Al igual que jal, con mret comienzan a restringirse las 2 instrucciones próximas. Se resuelve igual: con predicción del salto a tomar y con una penalización máxima de 2 ciclos y mínima de 0. Y la unidad de conflictos debe ampliarse para borrar los registros de segmentación de IF/ID si detectan una mret.
- 2) Existe conflicto de datos al ejecutar una instrucción csrrw que lee de un registro modificado por otra instrucción previamente. Se resuelve con anticipación o restringiendo una parada si la previa ha sido lw. No requiere de cambios.

3) Que la instrucción csrrw modifique un registro que va a ser leído por una instrucción posterior.

Se resuelve pausando la siguiente instrucción durante un ciclo y ampliando la unidad de conflictos para parar la secuencia si detecta csrrw.

4) Cuando con la instrucción csrrw se modifica el mepc y luego se ejecuta la instrucción mret.

Se solventa escribiendo mepc al final de la primera mitad del ciclo de salto y requiere hacer cambios en la ruta de datos

En resumen, un procesador con gestión de excepciones podrá gestionar los conflictos con: la adición de los multiplexores de anticipación, las unidades de anticipación y de conflictos con las modificaciones anteriormente dichas.

Las interrupciones son trags que se producen por que dispositivos externos activan determinados señales que las provocan.

Con las excepciones el procesador cancela la instrucción causante, en las interrupciones las finalizan.

cuando se produce una interrupción se salta al estado SI:

$mepc \leftarrow PC$ (dirección de la siguiente instrucción)

$PC \leftarrow mirec$

$mcause \leftarrow "cause"$

Se llama al estado SI después de finalizar toda instrucción.

En el caso de las excepciones el estado se llama SE y se lanza después de revisar que ha ocurrido una en aquellos estados en los que se ve involucrada la memoria o el banco de registros.