Arquitectura y Tecnología de Computadores

Tema 2 - **DISEÑO DE UN PROCESADOR:** Segmentación en la ejecución de instrucciones

4º Curso de Ingeniería de Telecomunicación

Profesor: Francisco Javier Gómez Arribas

Departamento de Ingeniería Informática y Telecomunicaciones

UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

ATC

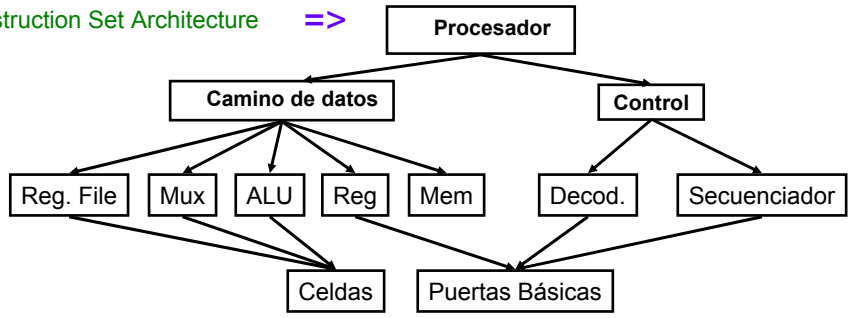
Contenidos

ATC

Arquitectura ISA: Especificación del conjunto de Instrucciones

- **Recursos visibles al programador**
- **Formato Instrucciones, modos de direccionamiento. => CISC vs RISC**

Instruction Set Architecture =>



ATC

ADD y SUB

addU rd, rs, rt

OR Inmediato:

ori rt, rs, imm16

LOAD y STORE Word

lw rt, rs, imm16

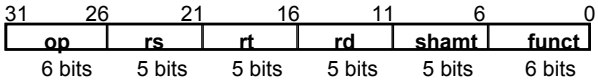
sw rt, rs, imm16

SALTOS:

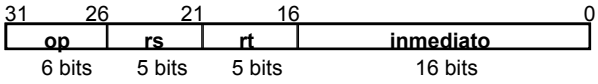
beq rs, rt, imm16

jmp imm16

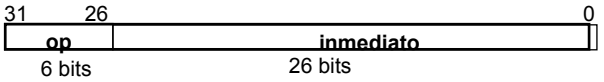
- **Tipo-R**



- **Tipo-I**



- **Tipo-J**



- Todas las instrucciones de igual tamaño
- Los registros y los valores inmediatos en la misma posición
- Las operaciones se realizan entre registros

El repertorio de Instrucciones RISC

Filosofía RISC

"Hacer más sencillo y en consecuencia más rápido casi todas las tareas que se pueden ejecutar en un sistema CISC "

Instrucciones complejas ☑ **Mejor no tenerlas**

☑ Sustituirlas por software

VLSI ☑ **Un sistema sencillo ocupa menos espacio**

☑ **Un sistema sencillo es más rápido**

☑ **Facilita ejecución segmentada**

No utilizar Microcódigo, basta con controladores secuenciales (FSM)

☑ **Minimizar riesgos estructurales de acceso a recursos**

☑ **Aumentar el paralelismo**

Características de un RISC genérico

Utilizan arquitecturas Harvard.

No se permiten códigos automodificables.

Utilizan sistemas con cache de uno o mas niveles.

Utilizan caches independientes para intrucciones y datos.

Utilizan registros FIFO para precaptura de instrucciones.

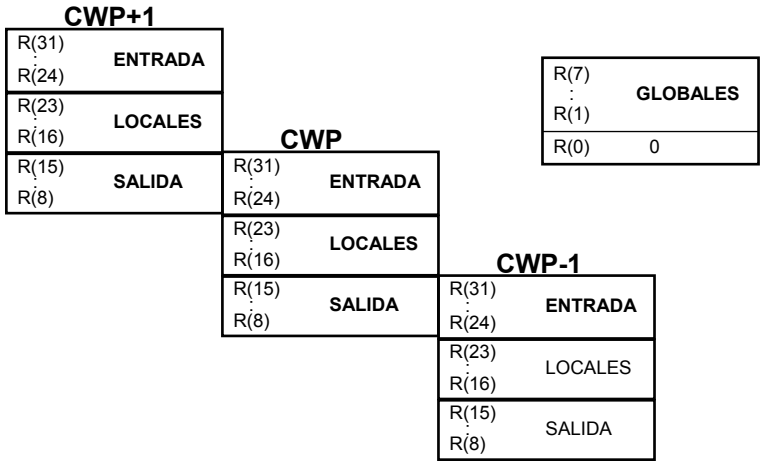
Sólo dos operaciones con memoria: LOAD Y STORE.

Los accesos a memoria quedan minimizados por disponer de un gran número de registros internos.

Algunos RISC utilizan registros internos como pila de memoria para guardar datos temporales entre subrutinas (ventana de registros)

La ventana de registros en un sistema RISC

SPARC



Comparación entre procesadores CISC y RISC.

	CISC	RISC
Nº Instrucciones	100-300	< 128 (2^7)
Instrucciones complejas	Si	No
Modos de Direccionamiento	8-20	2-3
Nº Formatos	> 10	≤ 4
Palabras/Instrucción	1-10	1
Set ortogonal de Instruc.	No	Si
Instruc. con memoria	Varias	2 (Load/Store)
Ciclos/Instrucción	3-10	~ 1
Nº Registros internos	2-16	> 32
Procesador segmentado	A Veces	Siempre
Unidad de Control	µCódigo	Secuencial

Argumentos de los partidarios de CISC

Los programas son más cortos

- **Se reducen las llamadas a memoria y por tanto hay menos latencias de acceso**

Un set de instrucciones complejo no es más caro que uno simple

- **Las instrucciones son más eficientes**
- **Mayor dificultad para clonar el sistema**

Permite la compatibilidad entre elementos de una misma familia

Facilita el trabajo de los compiladores

- **Mayor similitud entre los lenguajes de alto nivel y el ensamblador**

Argumentos de los partidarios de RISC

El Hardware es más sencillo

- **Los sistemas ejecutan las tareas más rápido**
- **El diseño de los sistemas es menos costoso**

El set de instrucciones es sencillo y con pocos formatos, lo que permite homogeneizar el tiempo de ejecución de las instrucciones

- **Facilita el uso de procesadores segmentados**
- **Facilita el procesamiento paralelo**

Facilita el trabajo de los compiladores

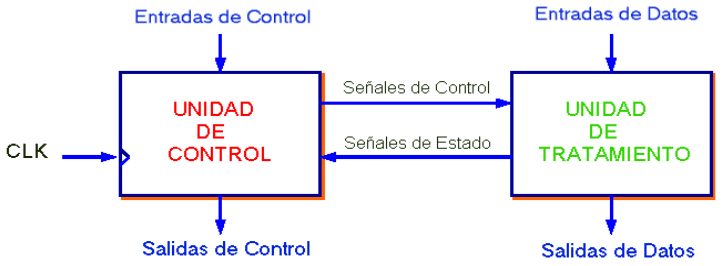
- **Debido a la menor complejidad del set de instrucciones**

Diseño de un procesador

1. Analizar el conjunto de instrucciones => requisitos de la unidad de tratamiento o camino de datos (datapath)
   - **El significado de cada instrucción viene dado por su funcionamiento a nivel de transferencia de registros (RTL).**
   - **El datapath debe incluir elementos de almacenamiento para los registros accesibles en el modelo de programación del procesador.**
   - **El datapath debe soportar todas las transferencias entre registros definidas en el conjunto de instrucciones.**
2. Selección de los componentes y de la metodología de reloj.
3. Implementación del datapath cumpliendo los requisitos.
4. Análisis de cada instrucción para determinar el mecanismo de control que efectúe la transferencia entre registros.
5. Implementación de la lógica de control.

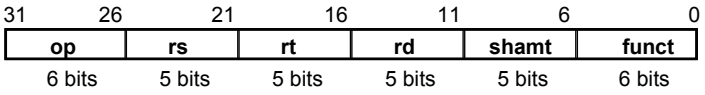
Diseño de un procesador

1.- Unidad de tratamiento + Unidad de Control



Unidad de Control: encargada de realizar el control del proceso, es decir de generar las señales necesarias para activar los componentes de la unidad de tratamiento que actuarán sobre los datos en el instante de tiempo que corresponda.

Unidad de Tratamiento o camino de datos: agrupa a todos los componentes capaces de manipular los datos, y un conjunto de elementos de almacenamiento en en el interior del procesador.



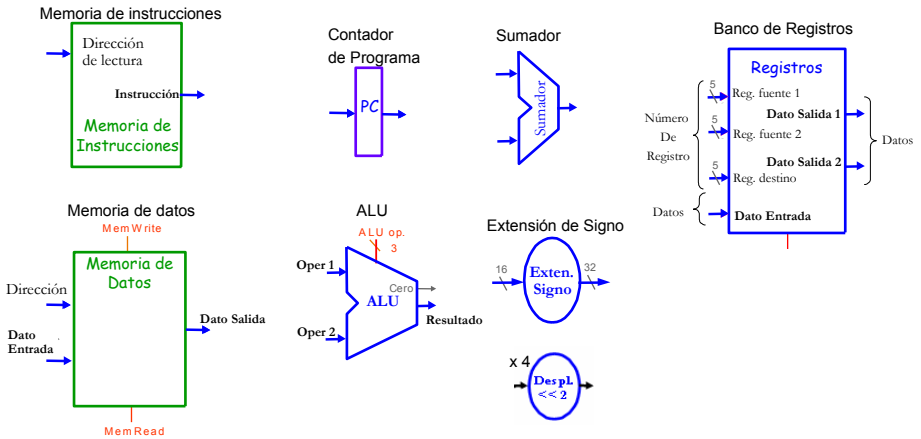
2.- Fase de Ejecución: En un único ciclo o multiciclo

Instrucción addU rd, rs, rt

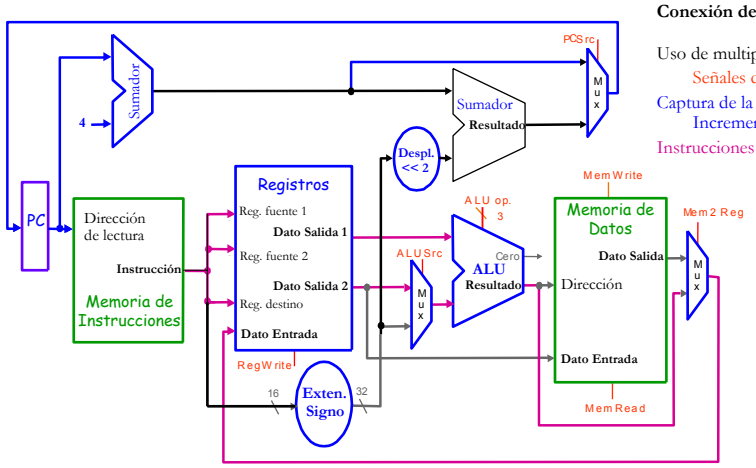
Descripción de la ejecución (RTL)

Mem[PC]	**Carga de la instrucción desde memoria**
R[rd] ← R[rs] + R[rt]	**Realiza la operación (SUMAR)**
PC ← PC + 4	**Calcula la dirección de la siguiente instrucción.**

Unidades funcionales necesarias para cada instrucción



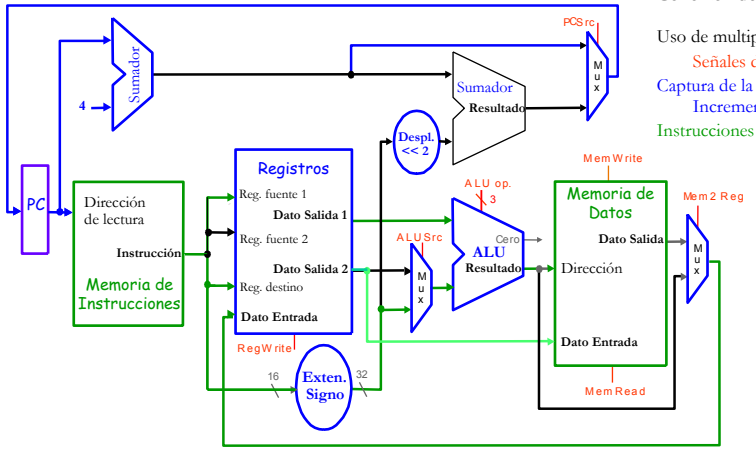
Diseño del camino de datos (Datapath)



Conexión de los elementos:

Uso de multiplexores.
Señales de control
Captura de la siguiente instruc.
Incremento de PC.
Instrucciones de la ALU entre registros.

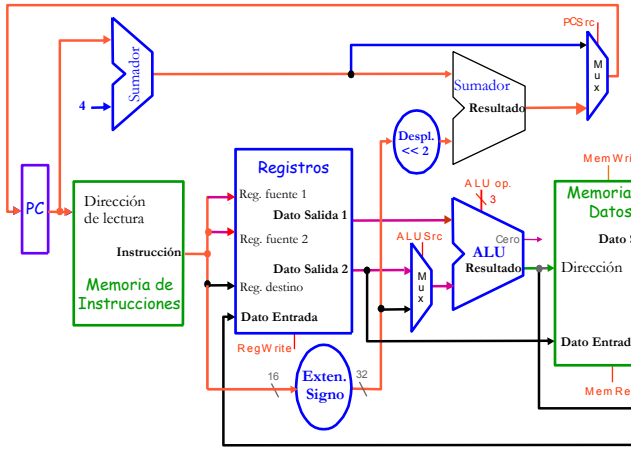
Diseño del camino de datos (Datapath)



Conexión de los elementos:

Uso de multiplexores.
Señales de control
Captura de la siguiente instruc.
Incremento de PC.
Instrucciones LOAD/STORE.

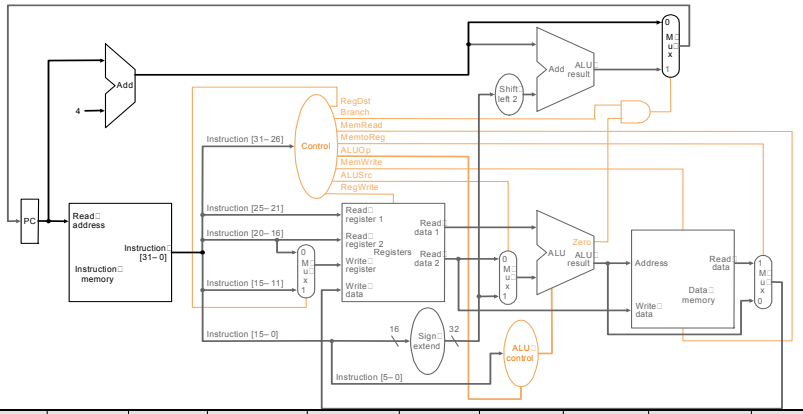
Diseño del camino de datos (Datapath)



Conexión de los elementos:

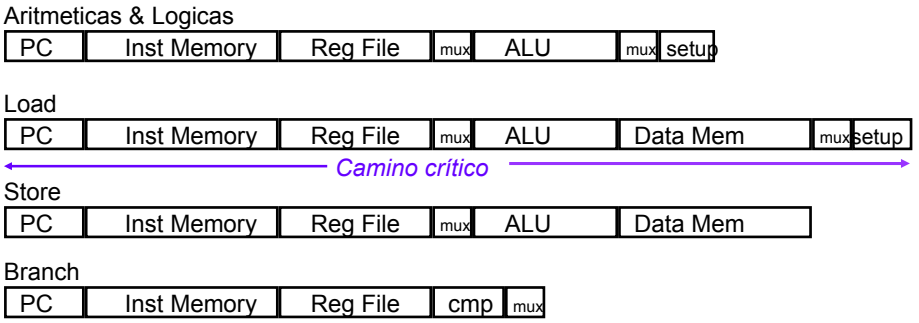
Uso de multiplexores.
Señales de control
Captura de la siguiente instrucción o del destino de salto =>Incremento de PC.
Instrucciones de Salto y Salto Condicional

Control (Ciclo único)



Instrucción	RegDst	ALUSrc	Mem2Reg	RegWrite	MemRd	MemWr	Branch	ALUOp1	ALUp0
Oper. Reg	1	0	0	1	0	0	0	1	0
LOAD	0	1	1	1	1	0	0	0	0
STORE	X	1	X	0	0	1	0	0	0
BEQ	X	0	X	0	0	0	1	0	1

Desventajas del diseño uniciclo (CPI=1)



Tiempo de ciclo muy largo.

Todas las instrucciones utilizan, sin necesidad, tanto tiempo como la instrucción más lenta.

La memoria Real no es ideal, existen detenciones y no siempre se accede en un solo ciclo (CDM).

Control Multiciclo

Las instrucciones pueden tardar diferente número de ciclos.

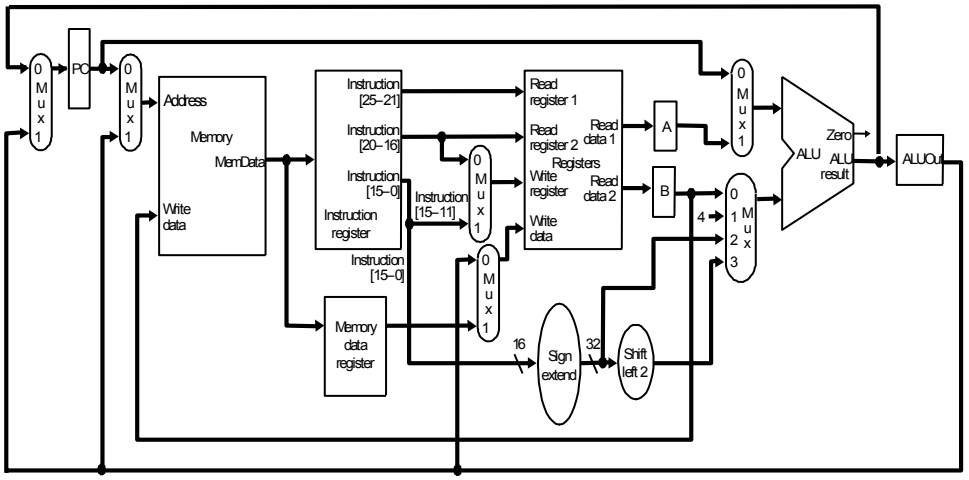
Un "datapath" multiciclo con ligeras modificaciones.

- **Dividir las instrucciones en pasos, cada paso tarda un ciclo**
- **Balancear la cantidad de trabajo a realizar en un paso**
- **En cada ciclo solo se utiliza una unidad funcional (se reduce el número de U.F.)**

Al final del ciclo

- **Almacenar los valores para su uso en posteriores pasos (es la mas fácil)**
- **Añadir registros internos adicionales**

Cambios para una aproximación multiciclo



Ejecución en cinco pasos (5 ciclos)

1.-Carga de Instrucción (todas)

IR = Memory [PC];
PC = PC + 4;

2.- Decodificación de la instrucción y lectura de registros (todas)

- **Lee registros rs y rt cuando sean necesarios.**
  A = Reg[IR[25-21]];
  B = Reg[IR[20-16]];
- **En saltos calcula la dirección**

  ALUOut = PC + (sign-extend(IR[15-0]) << 2);
- **En este ciclo no se da valor a las líneas de control porque se está decodificando la instrucción**

Ejecución en cinco pasos (5 ciclos)

3.-Ejecución, Calcula dirección de memoria, Finalización de salto.

La unidad ALU realiza una de las siguientes funciones dependiendo del tipo de instrucciones:

- **Referencia a Memoria:** **ALUOut = A+sign-extend(IR[15-0]);**
- **Operación entre registros:** **ALUOut = A op B;**
- **Saltos:** **if (A==B) PC = ALUOut;**

4.-Acceso a Memoria o escritura del registro resultado.

Acceso a memoria en Loads y Stores.

Load	MDR = Memory[ALUOut];
Store	Memory[ALUOut] = B;

Escritura del registro destino en instrucciones entre registros.

Reg[IR[15-11]] = ALUOut;

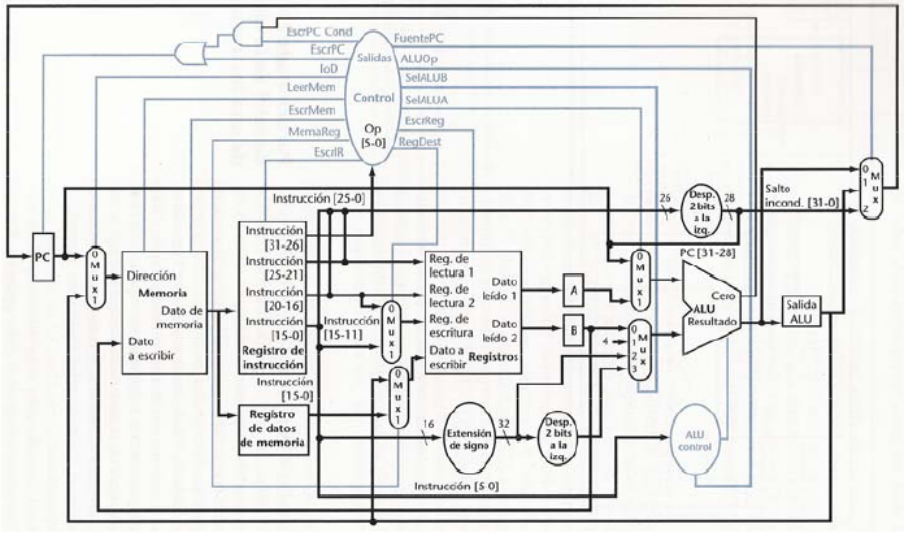
La escritura tiene lugar en el flanco al final del ciclo.

5.- Escritura del valor leído de memoria en el registro destino (Write-back).

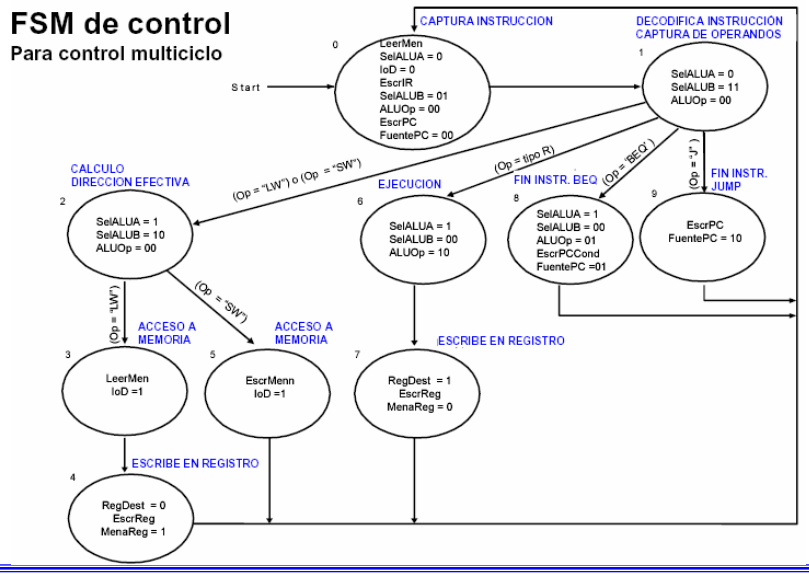
- Reg[IR[20-16]]= MDR;

"UNA INSTRUCCIÓN TARDA DE 3 A 5 CICLOS"

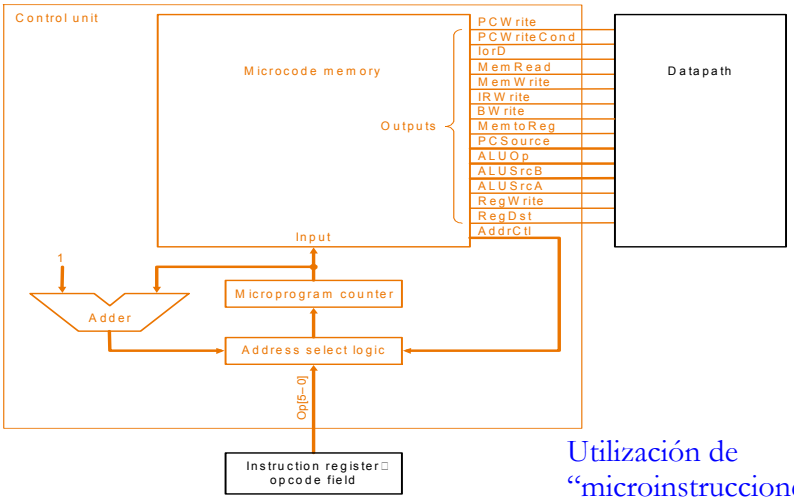
Control Multiciclo



Control Secuencial



Control Microprogramado



Utilización de "microinstrucciones"

- **La Técnica de la Segmentación.**
  - Funcionamiento ideal.
  - Conceptos asociados: Latencia y *Throughput*.
- **Segmentación de Instrucciones.**
  - Conflictos que producen retardo en la ejecución.
    - **Riesgos estructurales.**
    - **Riesgos de Control.**
    - **Riesgos por la dependencia de datos**
- **Técnicas para evitar riesgos**
    - Adelantamiento de datos
    - Predicción de saltos

Técnica utilizada para optimizar el tiempo de ejecución de procesos que se realizan mediante la repetición de una secuencia de pasos básicos.

Permite la ejecución de procesos concurrentemente.

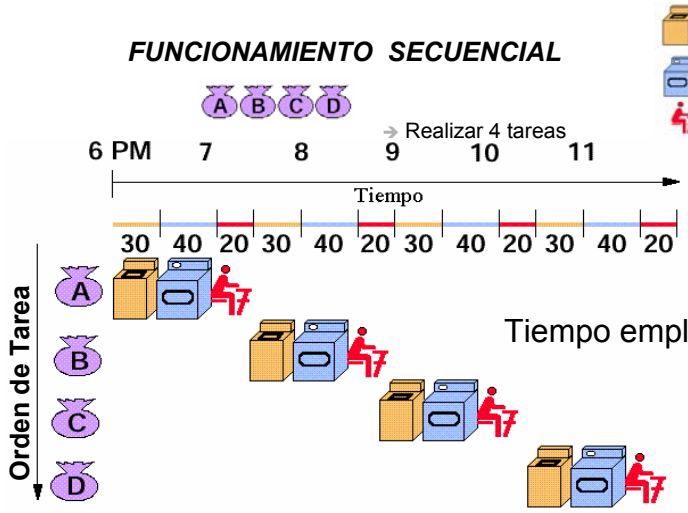
- ***Fundamento:*** Separar el proceso en etapas y ejecutar cada etapa en un recurso independiente.
- ***Objetivo:*** Mejorar la productividad, aumentando el número de procesos ejecutados por unidad de tiempo.
- ***Funcionamiento:*** Cuando una etapa del proceso termina, el recurso liberado puede empezar a ejecutar la misma etapa del siguiente proceso.
  - Se consigue la ejecución de varios procesos en paralelo cada uno en una etapa diferente.
  - Las etapas son ejecutadas secuencialmente.

Segmentación: Funcionamiento Ideal


→ Lavado 30 MIN


→ Secado 40 MIN


→ Planchado 20 MIN



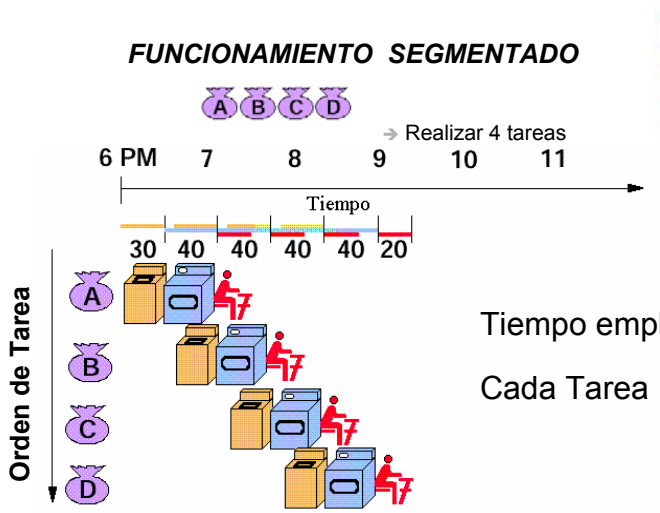
Tiempo empleado: 6 horas

Segmentación: Funcionamiento Ideal


→ Lavado 30 MIN


→ Secado 40 MIN


→ Planchado 20 MIN

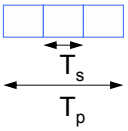


Tiempo empleado: 3.5 horas

Cada Tarea : 1.5 horas

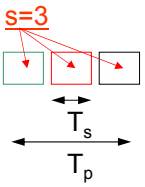
Segmentación: Funcionamiento Ideal

T_p es el tiempo de ejecución de un proceso.
Se puede descomponer en s (s=3) etapas de duración T_s

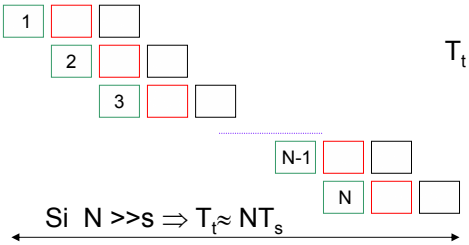


SECUENCIALMENTE (1 unidad de proceso para todas las etapas)

Ejecutar m procesos $T_t = NT_p = NsT_s$



Con SEGMENTACION (unidades independientes para cada etapa)



$$T_t = T_p + (N-1)T_s$$
$$= sT_s + (N-1)T_s$$
$$= (N+s-1)T_s$$

Si $N >> s \Rightarrow T_t \approx NT_s$

Segmentación: Funcionamiento Ideal

- ***Proceso segmentado*** vs ***Proceso secuencial***.

VENTAJAS

- La segmentación aunque no mejora la *latencia* de un solo proceso. mejora la *productividad* de una tarea con muchos procesos.
- Varios procesos se ejecutan "en paralelo".

RESTRICCIONES

- La razón de segmentación está limitada por la etapa más lenta.
- La aceleración máxima posible = Número de etapas de segmentación.
- Etapas de segmentación desequilibradas ⇒ Reducción de productividad.

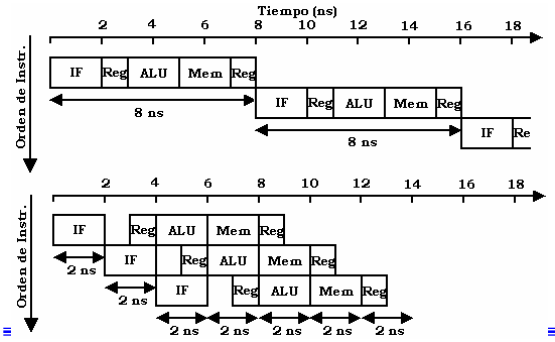
Segmentación de instrucciones.

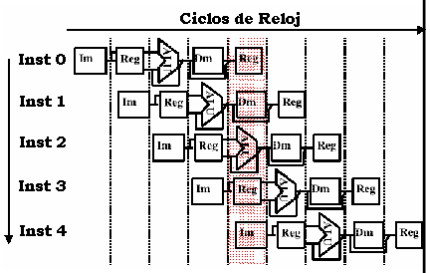
- Un procesador segmentado perfecto consigue ejecutar una instrucción por ciclo.
- La segmentación mas evidente consta de tres etapas :
  - Obtener instrucción (*Fetch*).
  - Decodificar instrucción (*Decode*).
  - Ejecutar instrucción (*Execute*).
- La frecuencia de funcionamiento es mayor si el numero de etapas de segmentación se incrementa. Aunque:
  - La segmentación fina es muy difícil.
  - Cada nueva etapa añade un retardo de registro.
  - La independencia entre etapas es mas difícil de conseguir.

Segmentación de instrucciones.

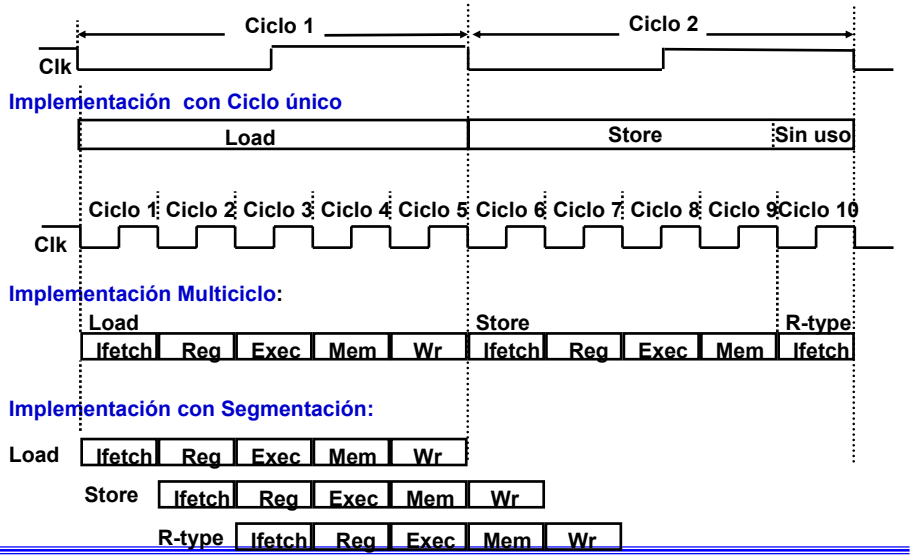
- EJEMPLO: Segmentación de instrucciones con 5 etapas:

IMem	Reg	ALU	DMem	Reg
Etapa IF	Etapa ID	Etapa EX	Etapa MEM	Etapa WB
→ Obtener instrucción. → Acceso a la memoria de instrucciones	→ Decodificar instrucción. → Lectura de operandos, carga de registros.	→ Ejecutar instrucción. o bien → Calcular dirección efectiva memoria.	→ Acceso a Memoria. o bien → Escribir en PC la dirección de salto.	→ Escribir en un registro el resultado de la operación.



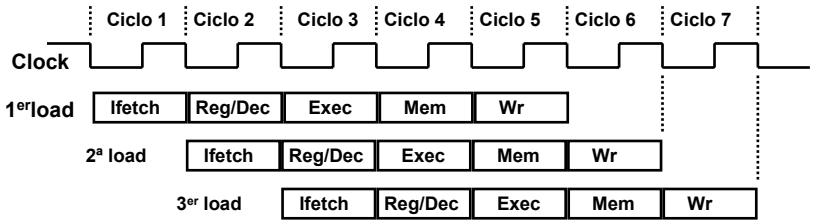


Comparación: Ciclo único / Multiciclo / Segmentado

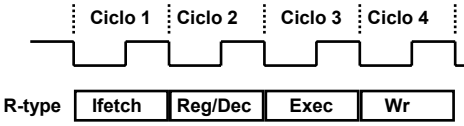


Muticiclo: Ciclos necesarios para Load y Op. entre registros

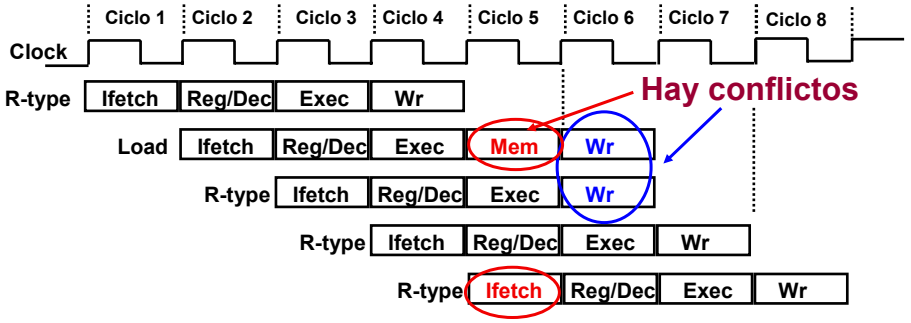
Las Cinco etapas de Load



Las Cuatro etapas en operaciones entre Registros



Ejecución de instrucciones: load seguido op. entre registros



Conflictos estructurales

- Hay dos instrucciones que intentan acceder a memoria al tiempo.
- Hay dos instrucciones que intentan escribir en el banco de registros al mismo tiempo y sólo existe un puerto de escritura.

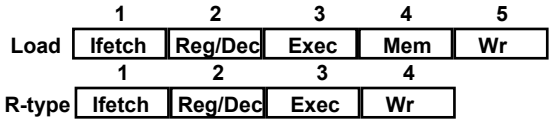
Consideraciones de Diseño de un cauce segmentado

Cada unidad funcional pueda usarse solo una vez por instrucción.

→ Deben aparecer dos unidades de memoria.

Cada unidad funcional se utiliza en la misma etapa para todas las instrucciones:

- Load usa el puerto de escritura en Registros durante su 5ª etapa.

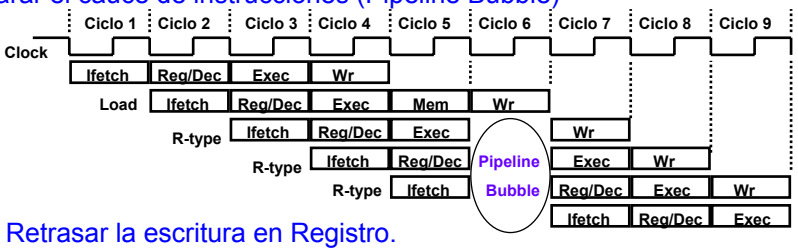


- Las operaciones entre Registros usan el puerto de escritura en Registros durante su 4ª etapa

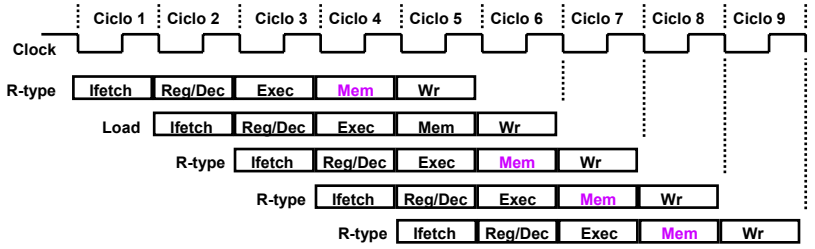
Soluciones: Paradas entre etapas, retraso de la escritura en registro, ...

Consideraciones de Diseño

Solución1: Parar el cauce de instrucciones (Pipeline Bubble)

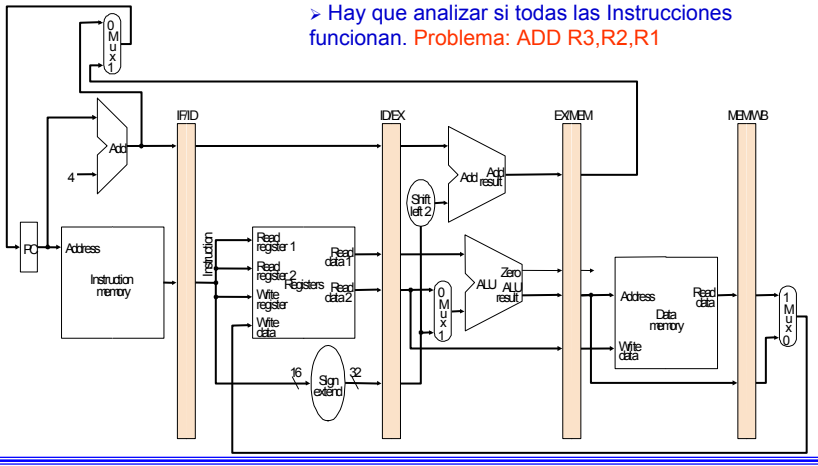


Solución 2: Retrasar la escritura en Registro.



Segmentación: Un diseño con 5 etapas

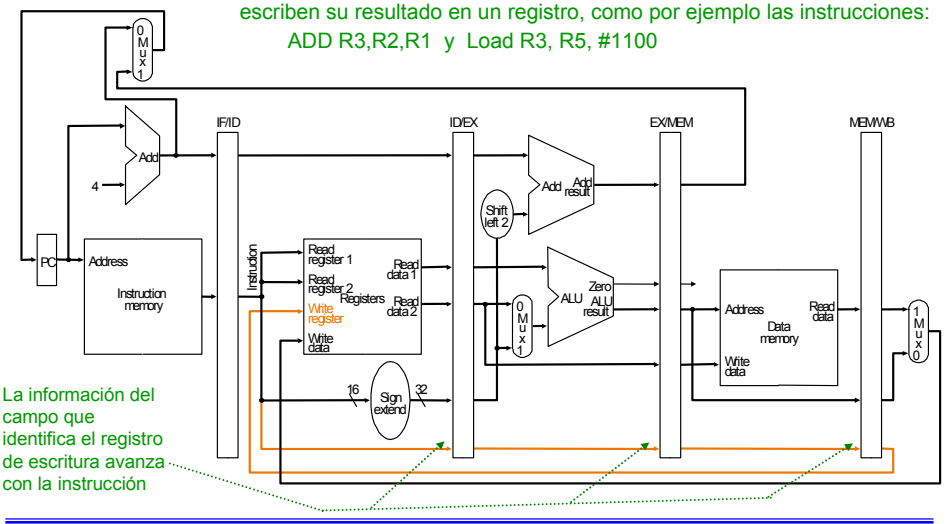
- La base es el camino de datos de un ciclo.
- Se añaden registros entre etapas.
- Hay que analizar si todas las Instrucciones funcionan. Problema: ADD R3,R2,R1



Segmentación: Modificación del datapath

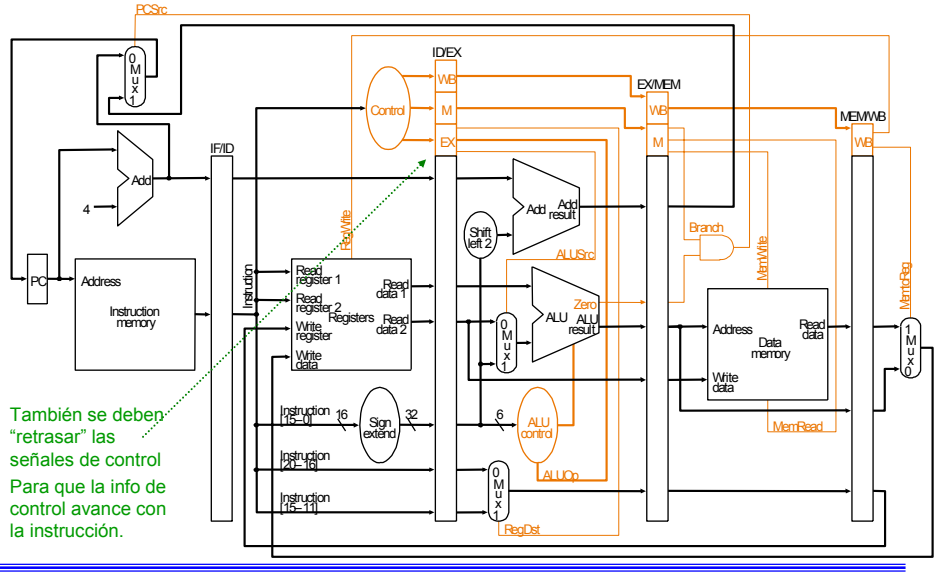
Para solucionar el problema de la ejecución de las instrucciones que escriben su resultado en un registro, como por ejemplo las instrucciones:

ADD R3,R2,R1 y Load R3, R5, #1100



La información del campo que identifica el registro de escritura avanza con la instrucción

Segmentación: Añadir el Control

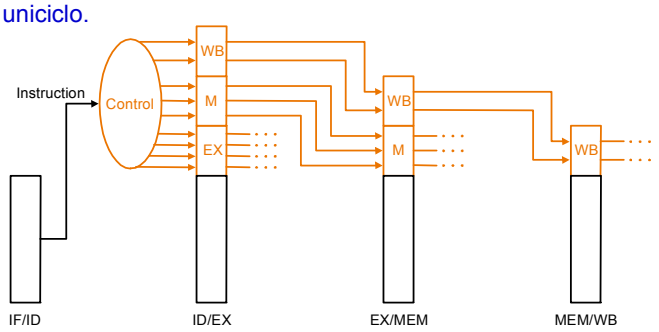


También se deben "retrasar" las señales de control

Para que la info de control avance con la instrucción.

Segmentación: Añadir el Control

- Todas las instrucciones tardan los mismos ciclos de reloj.
  - El secuenciamiento de la instrucción está implícito en la estructura del pipeline.
  - No hay un control especial para la duración de la instrucción
- Toda la información de control se calcula durante la decodificación, y se envía hacia delante a través de los registros de segmentación
- Los valores de las líneas de control son los mismos que los calculados en el control uniciclo.

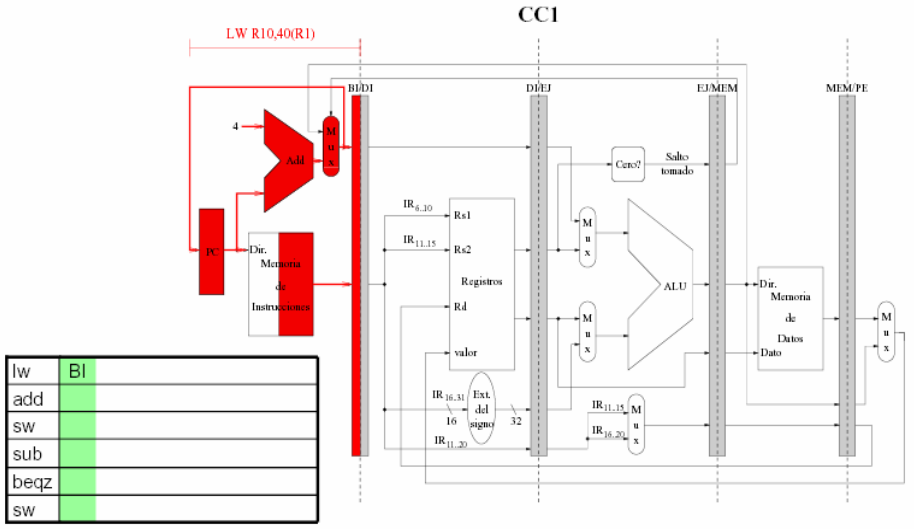


Segmentación: Ejemplo I

Pos. mem	etiqueta	instrucción
I	dest:	lw r10,40(r1)
I+4		add r1,r2,r3
I+8		sw 0(r4),r5
I+12		sub r2,r2,r6
I+16		beqz r1,dest
I+20		sw 40(r1),r10

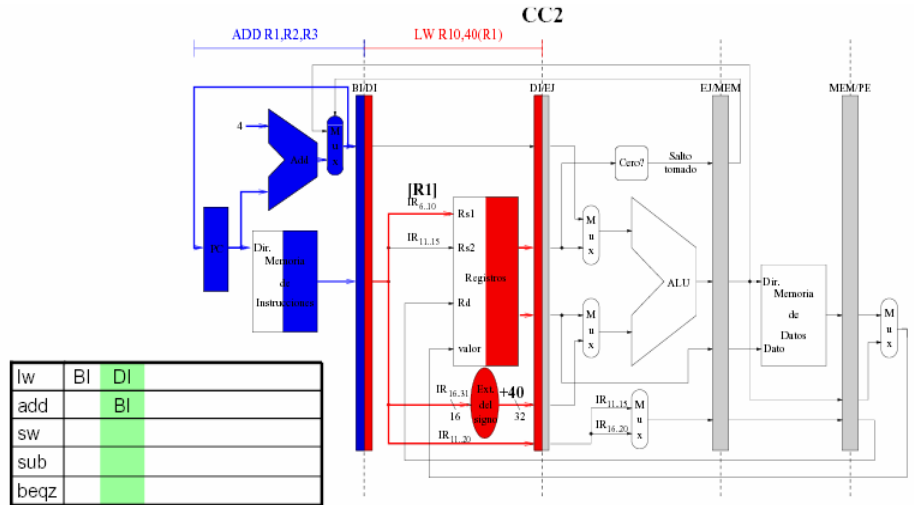
lw	1	10	40	
aritmética	2	3	1	add
sw	4	5	0	
aritmética	2	6	2	sub
beqz	1	0	-20	
sw	1	10	40	

Segmentación: Ejemplo II



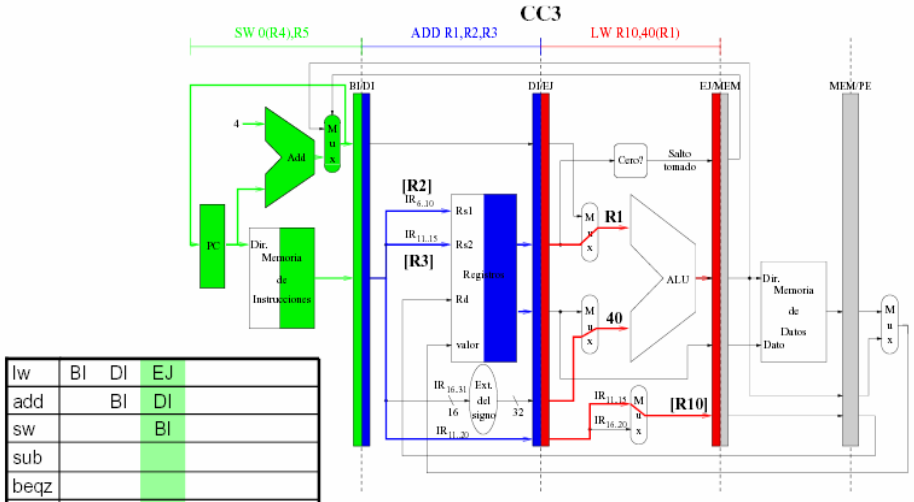
lw	BI
add	
sw	
sub	
beqz	
sw	

Segmentación: Ejemplo III



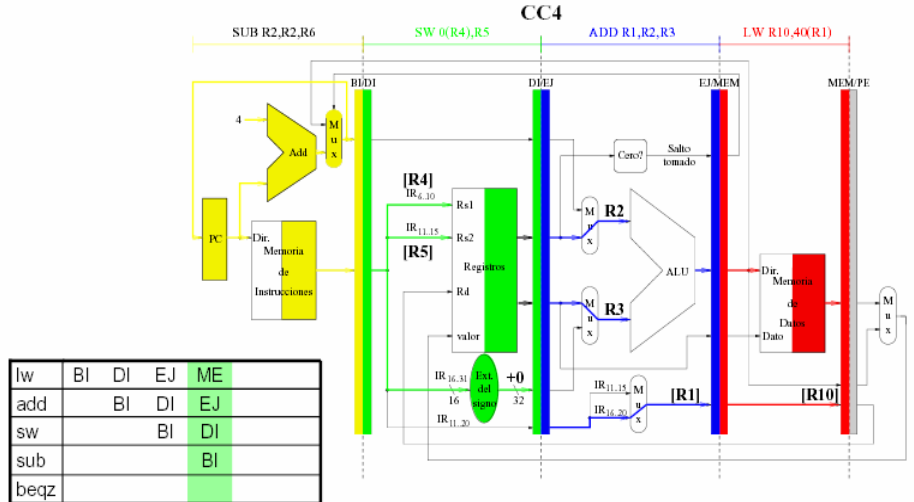
lw	BI	DI
add		BI
sw		
sub		
beqz		
sw		

Segmentación: Ejemplo IV



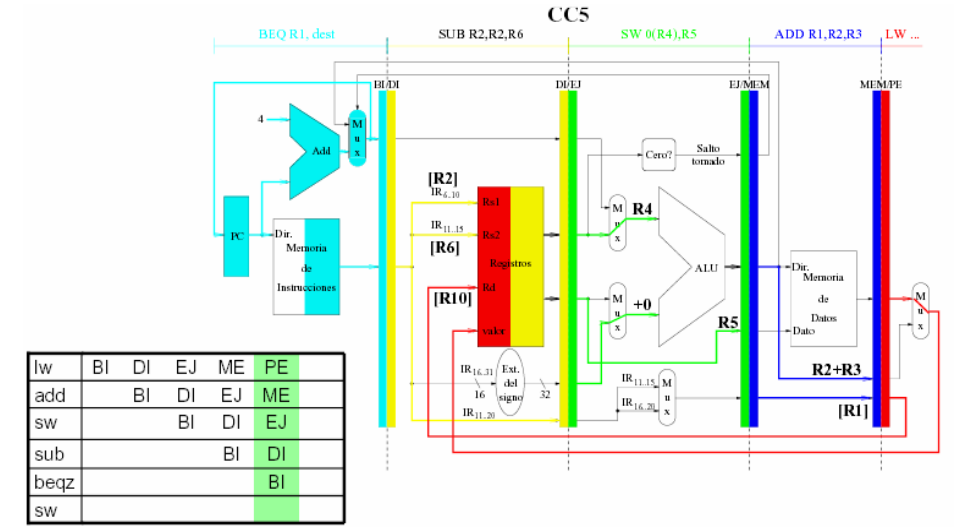
lw	BI	DI	EJ	
add		BI	DI	
sw			BI	
sub				
beqz				
sw				

Segmentación: Ejemplo V



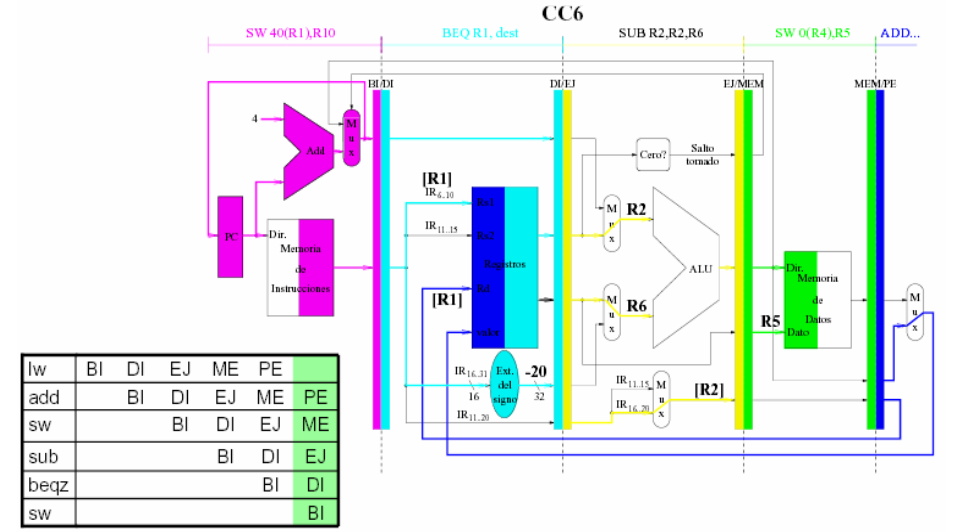
lw	BI	DI	EJ	ME	
add		BI	DI	EJ	
sw			BI	DI	
sub				BI	
beqz					
sw					

Segmentación: Ejemplo VI



lw	BI	DI	EJ	ME	PE
add		BI	DI	EJ	ME
sw			BI	DI	EJ
sub				BI	DI
beqz					BI
sw					

Segmentación: Ejemplo VII

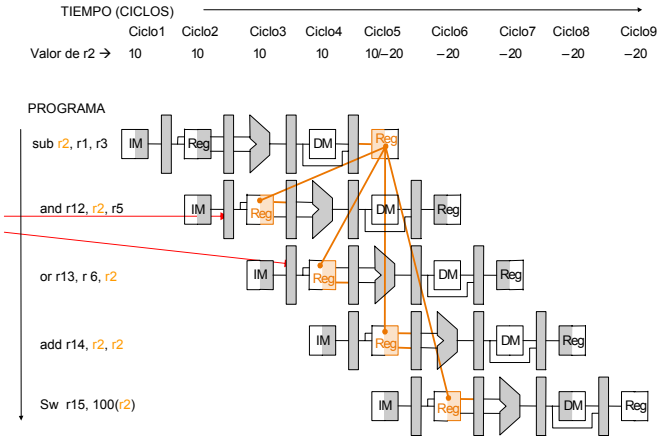


lw	BI	DI	EJ	ME	PE	
add		BI	DI	EJ	ME	PE
sw			BI	DI	EJ	ME
sub				BI	DI	EJ
beqz					BI	DI
sw						BI

Conflictos en funcionamiento real.

- Aparecen problemas al poder empezar la siguiente instrucción antes de que la primera haya terminado.

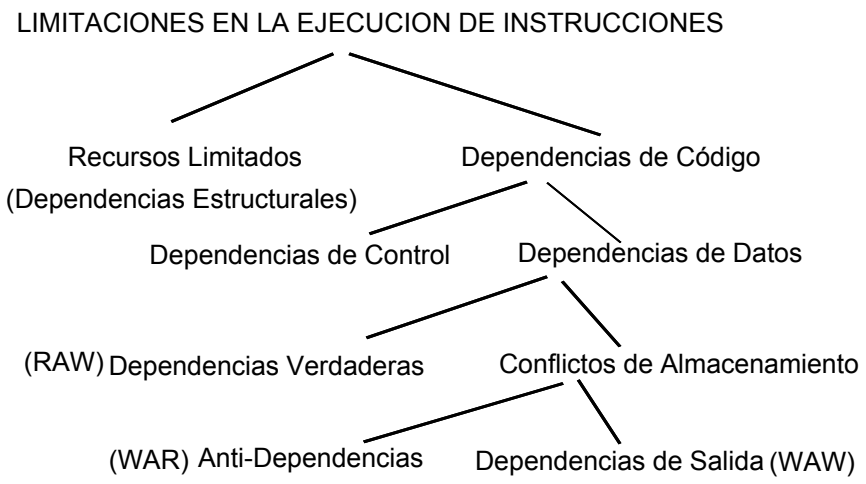
Dependencias problemáticas son aquellas que necesitan datos que hay que buscar " hacia atrás" en el esquema de tiempos.



Conflictos en funcionamiento real.

- Las causas que pueden reducir el rendimiento en un procesador segmentado de instrucciones son tres:

1. ***Riesgos estructurales***:
   - Se intenta usar el mismo recurso de dos maneras diferentes al mismo tiempo.
   - El hardware impide una cierta combinación de operaciones.
2. ***Riesgos por dependencia de datos***:
   - Se intenta usar un dato antes de que esté disponible.
   - El operando de una instrucción depende del resultado de otra instrucción precedente que todavía no se ha obtenido.
3. ***Riesgos de control***:
   - Se intenta tomar una decisión antes de evaluarse la condición.
   - La ejecución de una instrucción de salto supone un cierto retraso.



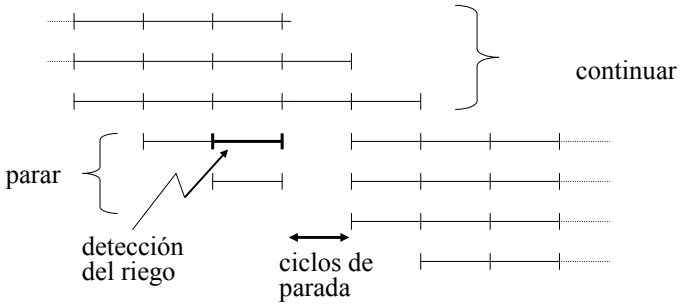
¿Todos estos riesgos se pueden solucionar?.....Si
¿Cómo?.....Esperando

- Cuando se detecta un riesgo, la solución más simple es parar la unidad de segmentación (Stall the Pipeline) hasta que desaparezca el riesgo.
  - Las instrucciones que preceden a la causante del riesgo pueden continuar.
  - La instrucción que causa el riesgo y siguientes, no continúan hasta que desaparece el riesgo.
- Se necesita que el control de la unidad de segmentación (*Pipeline)* sea capaz de:
  - Detectar las causas de riesgo.
  - Decidir acciones que resuelvan el riesgo (por ejemplo, esperar).

Dependencias : Cómo actuar

ATC

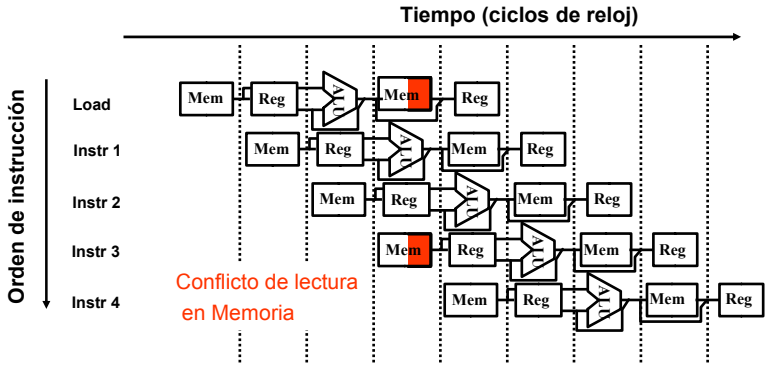
- Cuando se produce un riesgo hay que "parar" el sistema
- En un procesador segmentado hay que:
  - * Parar la instrucción que produce el riesgo
  - * Parar las instrucciones que siguen a la que produce el riesgo
  - * Continuar ejecutando instrucciones anteriores a la que produce el riesgo
  - * No se buscan más instrucciones mientras permanece el riesgo



Riesgos estructurales.

ATC

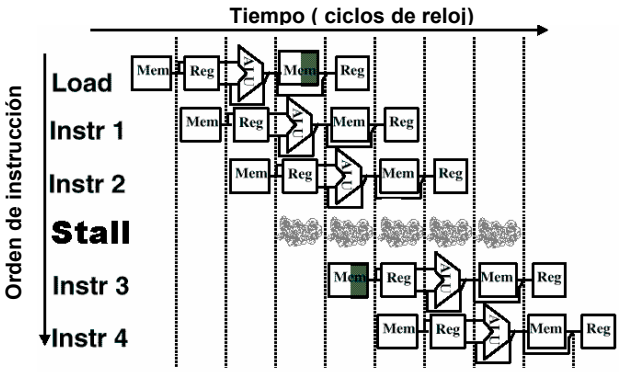
- Casos que se pueden presentar: Accesos simultáneos a:
  - Memoria (única para datos e instrucciones).
  - Unidades funcionales
  - Registros internos.



La detección es fácil en este caso! (La mitad izquierda coloreada indica escritura y la mitad derecha lectura)

Riesgos estructurales.

- Soluciones:
  - SW: El compilador estima el riesgo e introduce instrucciones NOPs
  - HW : El procesador lo detecta y para la ejecución de la instrucción.
  - Arquitectura: Duplicar recursos y separar las memorias de datos e instrucciones.



Riesgos por dependencia de datos.

- Dependencias que se presentan para 2 instrucciones i y j, con i ejecutandose antes que j.
  - RAW (Read After Write) Cuando la instrucción posterior j intenta leer una fuente antes que la instrucción anterior i la haya modificado.
  - WAR (Write After Read) Cuando la instrucción j intenta modificar un destino, antes que la instrucción i lo haya leído como fuente.
  - WAW (Write After Write) Cuando la instrucción j intenta modificar un destino, antes que la instrucción i lo haya hecho (Se modifica el orden normal de escritura).

✓ Ejemplos:

RAW

```
ADD  r1, r2, r3
SUB  r4, r1, r3
AND  r6, r5, r7
OR   r8, r1, r9
XOR  r10, r1, r11
```

WAR

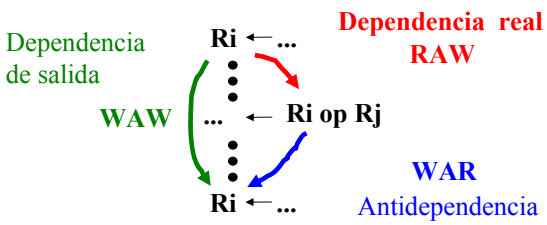
```
ADD r1, r2, r3
OR  r3,r4, r5
```

WAW

```
DIV  r1, r2, r3
AND r1,r4, r5
```

Ejemplos de dependencias de datos

I1: R3 = R3 op R5
I2: R4 = R3 + 1
I3: R3 = R5 + 1
I4: R7 = R3 op R4



Riesgos:

1) RAW (dependencia verdadera)
La salida de I1, R3, es uno de los operandos en I2
I2 tiene una dependencia de datos RAW con I1 por R3.

2) WAW (dependencia de salida)
I1 e I3 intentan escribir en R3
I3 tiene una dependencia de datos WAW con I1 por R3

3) WAR (antidependencia)
La instrucción I3 no puede comenzar hasta que la instrucción I2 haya obtenido uno de sus operandos (R3)
I3 tiene una dependencia de datos WAR con I2 por R3

Ejemplos de dependencias de datos

Código original

I1: R3 = R3 op R5
I2: R4 = R3 + 1
I3: R3 = R5 + 1
I4: R7 = R3 op R4

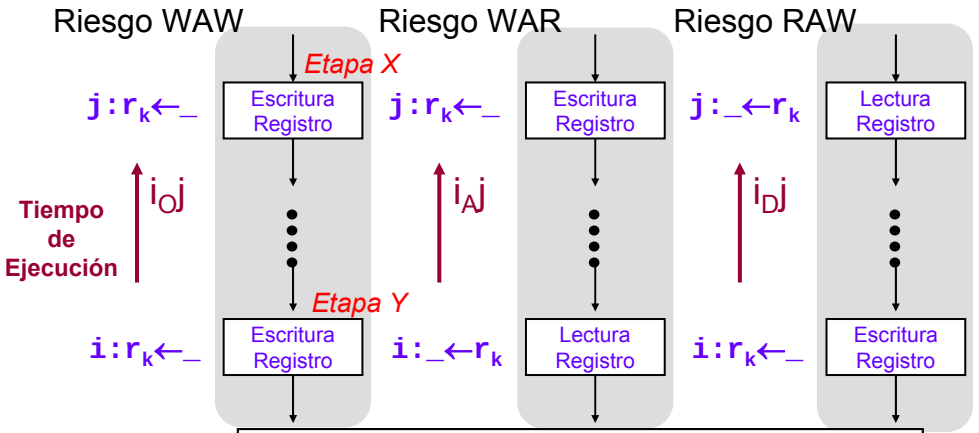
Código modificado compilado
sin limitaciones en el número de registros

I1: R23 = R13 op R15
I2: R14 = R23 + 1
I3: R33 = R25 + 1
I4: R17:= R33 op R14

Riesgos:

Sólo quedan los riesgos RAW (dependencia verdadera).
La salida de I1, R23, es uno de los operandos en I2
La salida de I3, R33, es uno de los operandos en I4

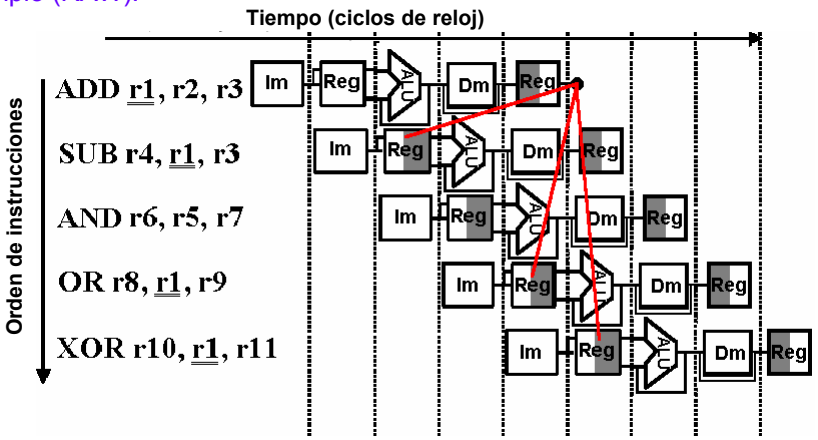
Dependencia de datos: condiciones necesarias



dist(i,j) ≤ *dist*(X,Y) ⇒ Dependencia de datos
dist(i,j) > *dist*(X,Y) ⇒ Independientes

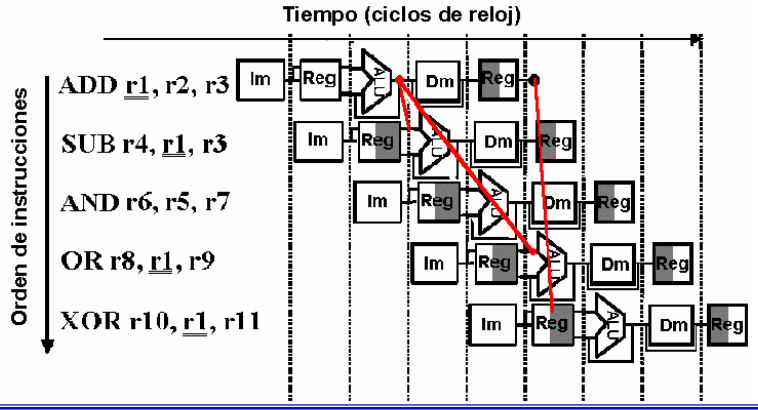
Riesgos por dependencia de datos.

- Ejemplo (RAW):



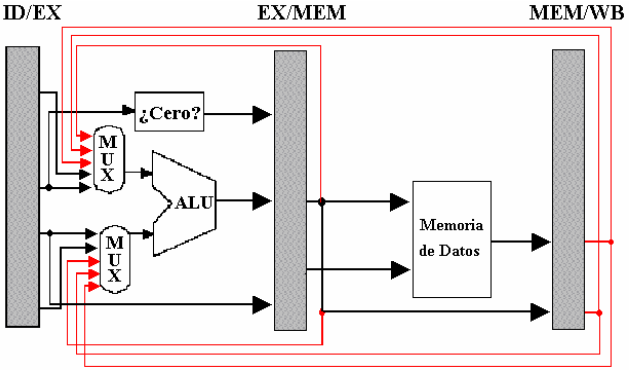
Riesgos por dependencia de datos.

- Soluciones:
  - "Adelantar" (Forward) el resultado de una etapa a las siguientes.
  - Definir adecuadamente la secuencia Read/Write (la instrucción OR funciona correctamente si en la etapa WB, Write se realiza en la 1ª mitad del ciclo y Read en la 2ª).



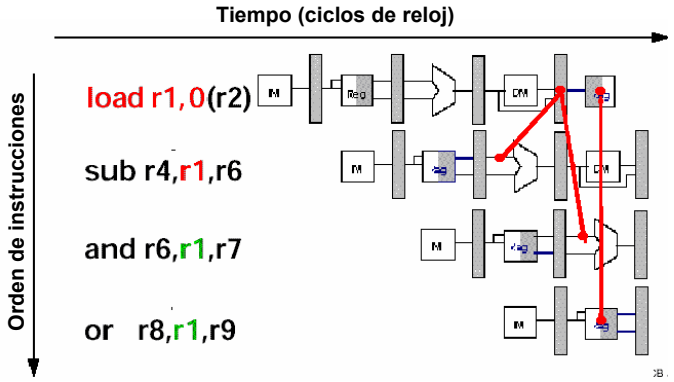
Riesgos por dependencia de datos.

- Necesidades Hardware para adelantar resultados:
  - Multiplexores adicionales en las entradas de datos de la ALU.
  - Buses extra entre registros internos y multiplexores.
  - Comparadores entre los operandos de una instrucción y los operandos destino de instrucciones previas.



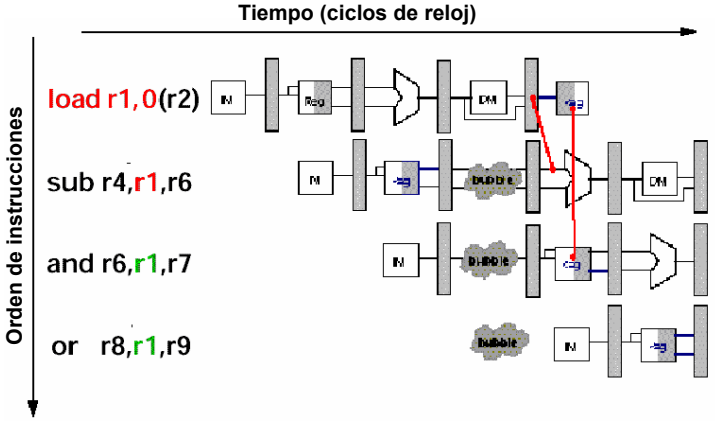
Riesgos por dependencia de datos.

- Los Riesgos persisten incluso con adelanto de resultados:
  - La instrucción LOAD puede evitar con adelanto de resultados los riesgos en AND y en OR pero no de SUB (No puede adelantar resultados a etapas que son de tiempos anteriores)

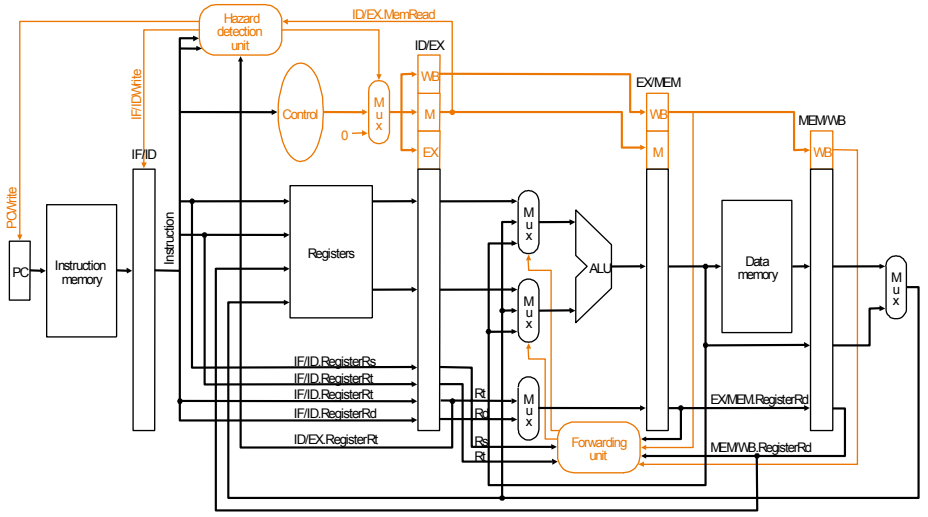


Riesgos por dependencia de datos.

- Solución:
  - Insertar un ciclo de espera (Stall) en el ciclo 3º, para la instrucción SUB y siguientes



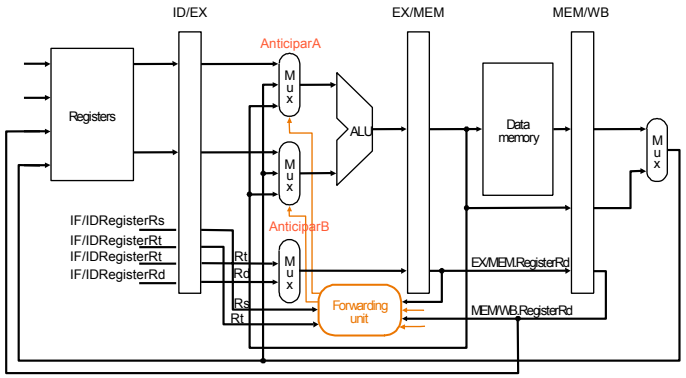
Control para adelantamiento de datos y detección de riesgos



Control de dependencias para adelantamiento de datos

Unidad de Adelantamiento de Datos (Forwarding)

- Se debe detectar el riesgo y luego "anticipar" el valor a la ALU
- Se agrega un bloque combinacional para detectar y multiplexores para adelantar los datos oportunamente

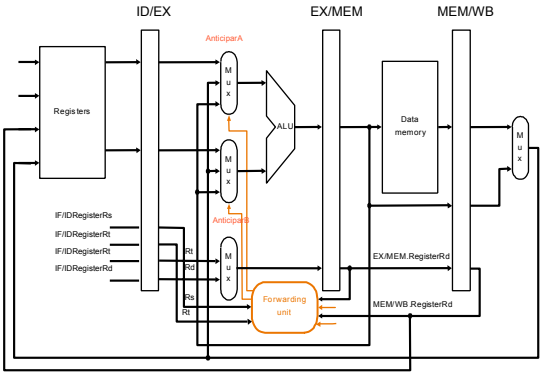


Unidad de Adelantamiento de Datos (Forwarding)

- **Se debe detectar el riesgo y luego "anticipar" el valor a la ALU**
- **Existen 4 Riesgos Potenciales**

1a. EX/MEM.Registro.Rd = ID/EX.Registro.Rs
1b. EX/MEM.Registro.Rd = ID/EX.Registro.Rt
2a. MEM/WB.Registro.Rd = ID/EX.Registro.Rs
2b. MEM/WB.Registro.Rd = ID/EX.Registro.Rt

Por ejemplo con:
Add r1, r2, r3
Sub r5, r1, r6
And r6, r5, r1
Add r4, r1, r3

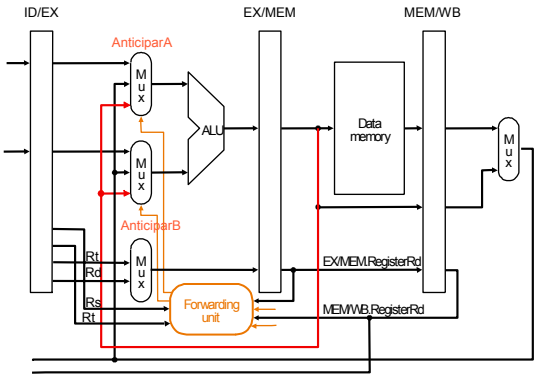


Unidad de Adelantamiento de Datos (Forwarding)

- **Pseudo-código del funcionamiento:**

(Análisis de Riesgos en EX)
if (EX/MEM.EscrReg **and**
EX/MEM.RegistroRd ≠ 0 **and**
EX/MEM.Registro.Rd = ID/EX.Registro.Rs)
then AnticiparA = 10
else AnticiparA = 00

if (EX/MEM.EscrReg **and**
EX/MEM.RegistroRd ≠ 0 **and**
EX/MEM.Registro.Rd = ID/EX.Registro.Rt)
Then AnticiparB = 10
else AnticiparB = 00



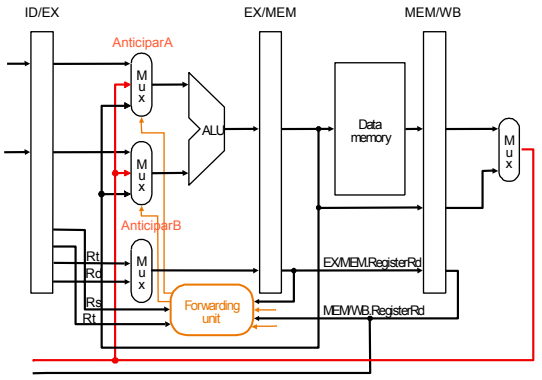
Unidad de Adelantamiento de Datos (Forwarding)

- **Pseudo-código del funcionamiento:**

(Análisis de Riesgos en MEM)

if (MEM/WB.EscrReg **and**
MEM/WB.RegistroRd ≠ 0 **and**
EX/MEM.Registro.Rd ≠ ID/EX.RegistroRs **and**
MEM/WB.Registro.Rd = ID/EX.RegistroRs)
then AnticiparA = 01
else AnticiparA = 00

if (MEM/WB.EscrReg **and**
MEM/WB.RegistroRd ≠ 0 **and**
EX/MEM.Registro.Rd ≠ ID/EX.RegistroRt **and**
MEM/WB.Registro.Rd = ID/EX.Registro.Rt)
then AnticiparB = 01
else AnticiparB = 00



Unidad de Detección de Riesgos (Hazard detection unit)

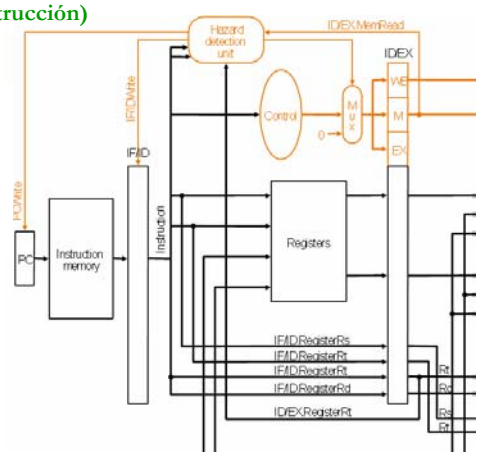
- **Necesario cuando el adelantamiento no resuelve los riesgos (caso de load y uso del registro destino en la siguiente instrucción)**

Pseudocódigo del funcionamiento:

If (ID/EX.LeerMem **and**
((ID/EX.Registro.Rt = IF/ID.Registro.Rs) **or**
(ID/EX.Registro.Rt = IF/ID.Registro.Rt)))
then *Bloquear el pipeline*

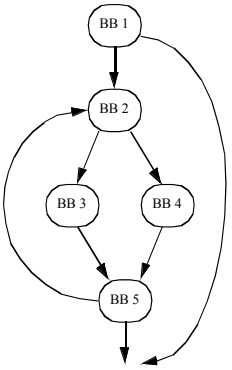
Bloquear el pipeline:

- ✓ PCWrite **= 0**
- ✓ IF/IDWrite **= 0**
- ✓ MuxNOP **= 1**



Dependencia de control

Flujo de ejecución de instrucciones: representa las posibles trayectorias.




```
main:
      addi r2, r0, A
      addi r3, r0, B
      addi r4, r0, C        BB 1
      addi r5, r0, N
      add  r10,r0, r0
      bge  r10,r5, end
loop:
      lw   r20, 0(r2)
      lw   r21, 0(r3)       BB 2
      bge  r20,r21,T1
      sw   r21, 0(r4)       BB 3
      b    T2
T1:
      sw   r20, 0(r4)       BB 4
T2:
      addi r10,r10,1
      addi r2, r2, 4
      addi r3, r3, 4        BB 5
      addi r4, r4, 4
      blt  r10,r5, loop
end:
```

Dependencia de Control: El nodo X es dependiente del nodo Y si la ejecución de X depende de los cálculos en el nodo Y.

Riesgos de Control (Instrucciones de salto)

- Las instrucciones de salto pueden suponer una alteración del orden secuencial de ejecución.

Etapa IF	Etapa ID	Etapa EX	Etapa MEM	Etapa WB
→ Obtener instrucción. → Acceso a la memoria de instrucciones	→ Decodificar instrucción. → Lectura de operandos, carga de registros.	→Ejecutar instrucción. →Calcular la condición → Calcular la dirección efectiva salto (PC+Inm.)	→ Acceso a Memoria. o bien → Escribir en PC la dirección de salto.	→ Escribir en un registro el resultado de la operación.

- No se sabe si el salto es efectivo hasta la etapa de ejecución y no se dispone de la dirección destino caso de que sea efectivo hasta el final de la cuarta etapa => Perdida de 3 ciclos
- Mejora: Adelantar el cálculo de PC+ Inmediato a la 2ª etapa.

Riesgos de Control (Instrucciones de Salto)

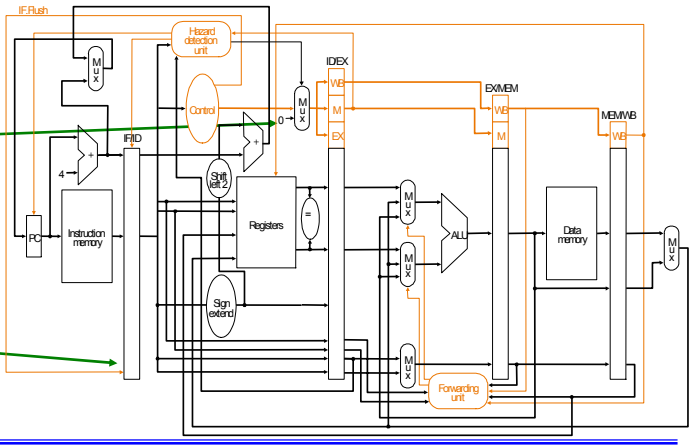
Cuando se decide saltar a una instrucción distinta de la siguiente, ya se están ejecutando otras instrucciones en el cauce segmentado. => Se necesita incluir hardware para vaciar (Flushing) el pipeline

Para descartar una instrucción cambia a 0 el valor de control mux

⇒ **NOP**

Para descartar una instrucción cambia a 0 el campo de instrucción del registro de segmentación IF/ID

⇒ **NOP**



Riesgos de Control (Instrucciones de salto)

Se necesita tomar una decisión sobre la siguiente instrucción a ejecutar antes que las condiciones de la instrucción de salto sean evaluadas.

¿Cómo realizar la ejecución de los SALTOS CONDICIONALES?

1. Esperar hasta que la dirección y condición del salto estén definidas.
   - ✓ Conviene conocer la dirección de salto y la condición tan pronto como se a posible.
2. Retardar la ejecución hasta conocer los parámetros del salto.
   - ✓ El compilador rellena con instrucciones los huecos de retardo.
3. Predecir el sentido del salto.
   - ✓ Se ejecuta especulativamente una rama, en caso de error se debe "vaciar" el procesador.
4. Especular anticipando la predicción de salto antes de su decodificación.

Riesgos de Control: Retardar la ejecución

2.- Técnicas de compilación: tres estrategias para buscar instrucciones de relleno

	■ DEL BLOQUE BASICO.	■ SI SALTO PROBABLE DEL BLOQUE DESTINO	■ SI SALTO NO PROBABLE DEL BLOQUE SECUENCIAL

add r1, r2, r3
bnz r2, L1
hueco
	.....
	bnz r2, L1
	add r1, r2, r3
		add r1, r2, r3
		bnz r2, L1
		and r2, r3, r2
			add r1, r2, r3
			bnz r2, L1
			sub r6,r7,r6

sub r6,r7,r6
mul r2, r3, r8
.....
	sub r6,r7,r6
	mul r2, r3, r8
	.....
		sub r6,r7,r6
		mul r2, r3, r8
		.....
			.....
			mul r2, r3, r8
			.....

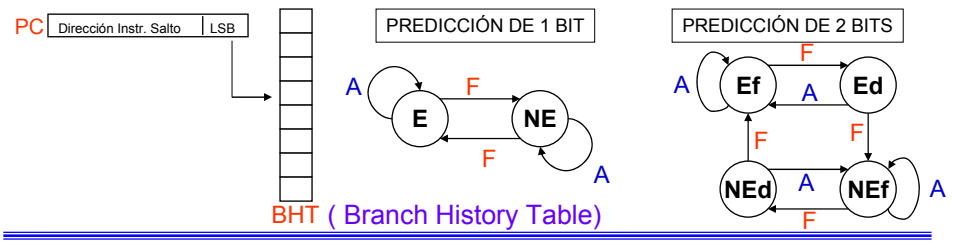
L1: and r2, r3, r2 andi r5,r6,inm	L1: and r2, r3, r2 andi r5,r6,inm	L1: andi r5,r6,inm	L1: and r2, r3, r2 andi r5,r6,inm
	■ Operación siempre válida. ■Siempre se realiza trabajo útil.	■ Operación siempre correcta mientras que r2 no sea fuente en el otro bloque antes de ser usado como registro destino. ■El ejemplo es **CORRECTO**	■ Operación válida siempre que r6 no se utilice como fuente en el bloque destino, antes de ser usado como registro destino. ■El ejemplo es **INCORRECTO**

Riesgos de Control: Predecir el salto

3- Ejecución especulativa de instrucciones:

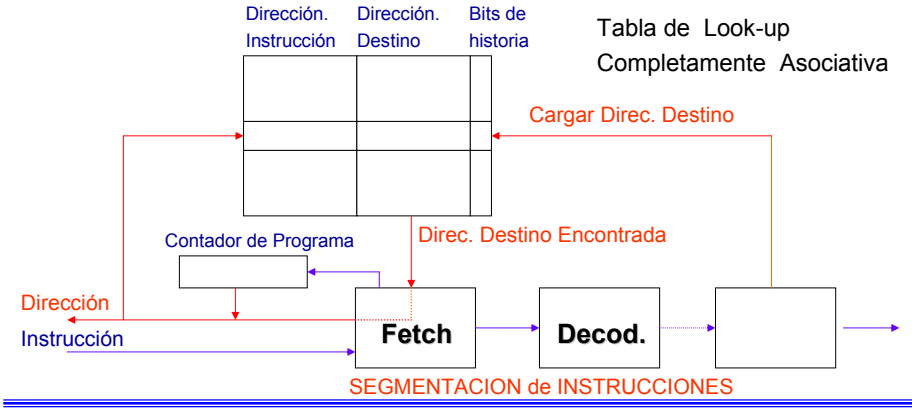
- Hasta conocer si el salto se realiza (la condición se evalúa en la 3ª ó 4ª etapa)
- En caso de error se deben eliminar los resultados especulativos

- Predicción estática: siempre predice el mismo sentido del salto.
  - Predicción efectiva (E), el salto se realiza.
  - Predicción no efectiva (NE), el salto no se realiza.
    - Predicción NE si el salto es adelante y E si es hacia atrás.
- Predicción dinámica: cambia la predicción en función de la historia del salto.
  - Utiliza una pequeña memoria asociada a cada dirección de salto (BHT)

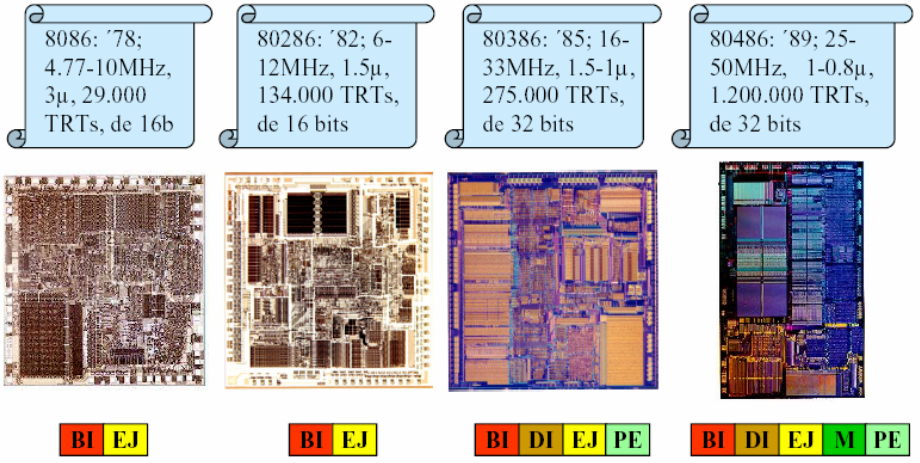


Riesgos de Control. Anticipar la dirección más probable

- 4.- Especular anticipando la predicción (antes de decodificar el salto)
  - ✓ Utiliza una tabla asociativa que incorpora, para cada instrucción de salto, la dirección de la instrucción de la predicción anterior.
  - ✓ A la tabla se la conoce como Buffer de destino de saltos o BTB.



Ejemplos de segmentación en procesadores



"CISC" Pipeline de 5-etapas en Intel i486

Etapas	Funciones realizadas
1. Captura de Instrucciones (Instruction Fetch)	Captura de instrucciones desde la cola de precaptura de 32 bytes. (Una unidad de precaptura se encarga de llenar y vaciar esta cola)
2. Decodificación de Instrucciones- 1 (Instruction Decode-1)	Traduce instrucciones en señales de control o microcódigo. Inicia la generación de direcciones y el acceso a memoria.
3. Decodificación de Instrucciones- 2 (Instruction Decode-2)	Acceso a la memoria de microcódigo Envía las microinstrucciones a la unidad de ejecución
4. Ejecución (Execute)	Ejecución de operaciones en ALU y op. de acceso a memoria
5. Escritura de registros. (Register Write-back)	Escritura en el banco de registros

Ejemplos de procesadores : MIPS R2000

Etapas	Fase	Función realizada
1. IF	ϕ1	Traducción de dirección virtual a real en el TLB
	ϕ2	Acceso a I-cache usando la dirección real
2. RD	ϕ1	Retorno de instr. de I-cache, comprueba tags & paridad
	ϕ2	Lectura del banco de reg.; en saltos genera direcc. destino
3. ALU	ϕ1	Inicio op en ALU; en saltos comprueba la condición
	ϕ2	Termina op en ALU; en load/store, traduce la dir. virtual
4. MEM	ϕ1	Acceso a D-cache
	ϕ2	Devuelve el dato de D-cache, comprueba tags & paridad
5. WB	ϕ1	Escritura en el banco de registros
	ϕ2	---