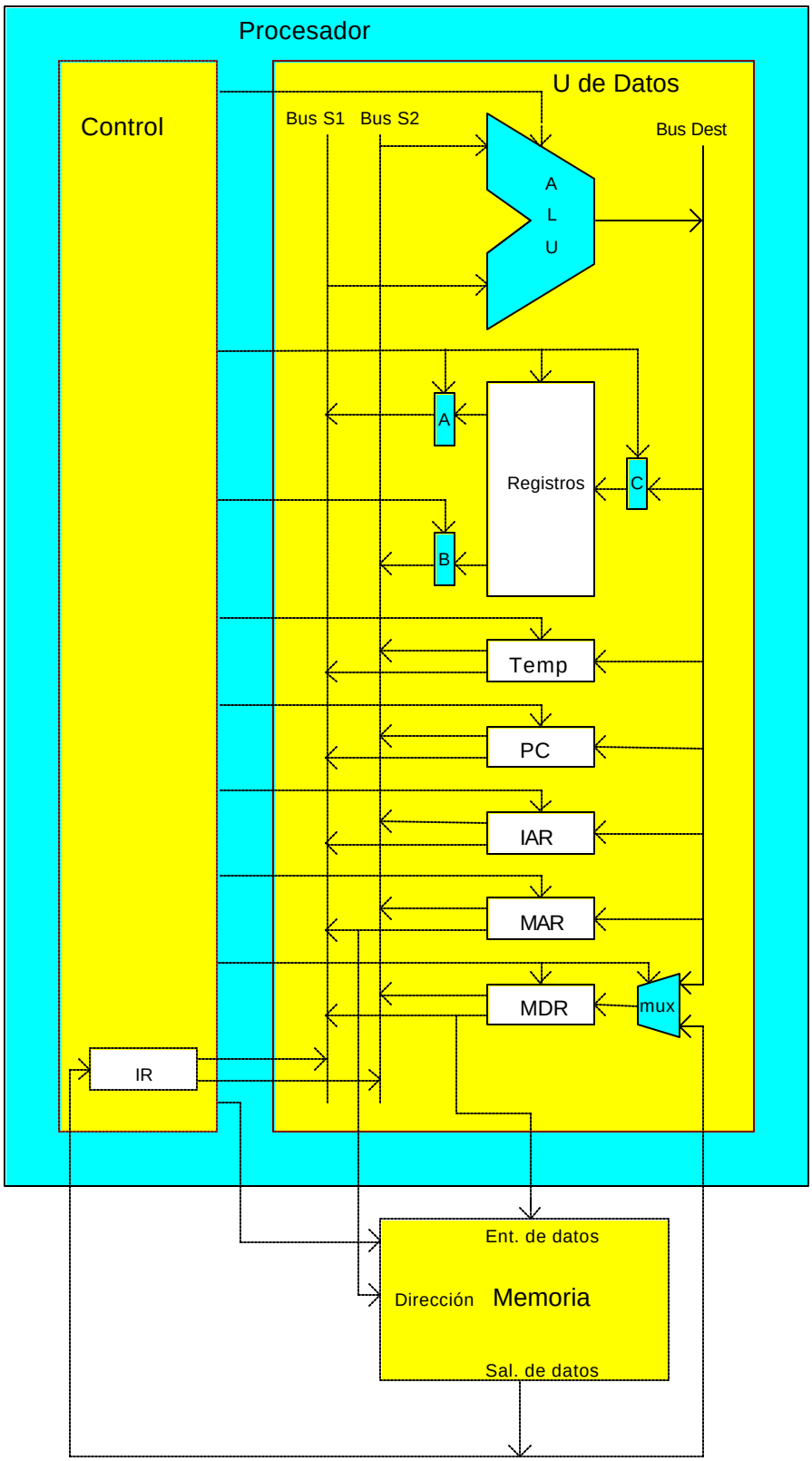
Estructura de Computadores II

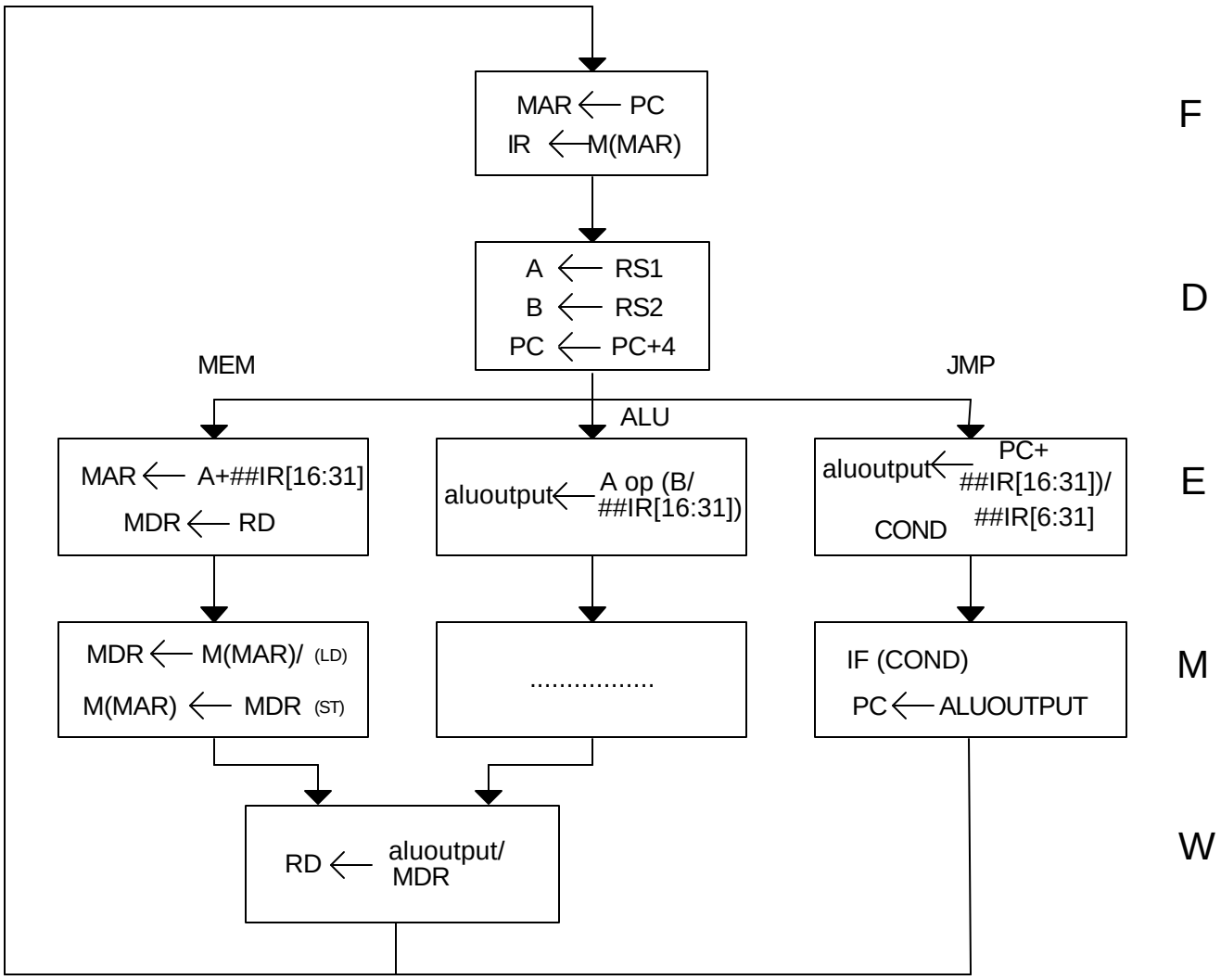
Folleto Nº 1

SEGMENTACIÓN

Jesús Ruiz M.



S1+S2	S1-S2
S1&S2	S1|S2
S1^S2	S1<<S2
S1>>S2	S1>>a S2
S1	S2
0	1



Segmentación es una técnica de implementación de computadores mediante la cual se traslapa la ejecución de múltiples instrucciones. La segmentación es clave en la implementación de computadores modernos.

La segmentación es como una línea de ensamblaje de automóviles; cada etapa completa una parte de una instrucción, cada una de estas etapas se denomina segmento. Las etapas están conectadas entre sí formando un cauce.- Las instrucciones entran por un extremo del cauce, son procesadas por las etapas y salen por el otro extremo.

La productividad de la segmentación está determinada por la frecuencia con que las instrucciones salen del cauce (son completadas). El tiempo requerido para desplazar una instrucción un paso a lo largo del cauce es un ciclo de máquina. La duración del ciclo de máquina está determinado por el tiempo de la etapa más lenta (por que todas las etapas progresan a la vez). A menudo el ciclo de máquina es un ciclo de reloj (a veces dos y raramente más), aunque el reloj puede tener múltiples fases.

Un aspecto clave en la segmentación es el equilibrio en la duración de las diferentes etapas, si las etapas están perfectamente equilibradas, el tiempo por instrucción, suponiendo condiciones ideales es:

$$\frac{\text{Tiempo por instrucción en máquina no segmentada}}{\text{Nº de etapas}}$$

En estas condiciones, la mejora de velocidad es igual al número de etapas. Sin embargo, las etapas nunca están perfectamente equilibradas. Además, la segmentación implica alguna penalización; por lo tanto, el tiempo por instrucción en una máquina no segmentada no tendrá su valor mínimo posible, aunque puede estar próximo.

La segmentación es una técnica de implementación que explota el paralelismo entre las instrucciones en un fluo secuencial. Tiene la ventaja de que no es visible (transparente) al programador.

Segmentación Básica

Para una máquina de cinco etapas,

F: Fetch
D: Decodificación/Búsqueda de Registros
E: Ejecución/Cálculo de Dirección Efectiva
M: Completar Salto/Acceso a Memoria
W: Retroescritura

Se puede segmentar la máquina iniciando una nueva instrucción en cada ciclo de reloj, convirtiendo cada una de las etapas anteriores en un segmento. Aunque cada instrucción dura cinco ciclos de reloj, durante cada ciclo se está ejecutando alguna parte de cinco instrucciones diferentes.

Inst\Ciclos	1	2	3	4	5	6	7	8	9
i	F	D	E	M	W				
ii		F	D	E	M	W			
iii			F	D	E	M	W		
iv				F	D	E	M	W	
v					F	D	E	M	W

La segmentación incrementa la productividad (throughput) de la cpu, pero no reduce el tiempo de ejecución de cada instrucción. En realidad, aumenta ligeramente el tiempo de

ejecución debido a la penalidad (gasto) del control de segmentación. El incremento de la productividad significa que los programas corren más rápido (menor tiempo de ejecución) aún cuando ninguna instrucción se ejecute con mayor rapidez.

El hecho de que el tiempo de ejecución permanezca inalterable, pone límites a la profundidad de la segmentación. Una vez que el ciclo de reloj sea tan pequeño como la penalidad de implementar la segmentación, no es útil una mayor segmentación.

EJEMPLO: Considerar una máquina de cinco etapas cuyas duraciones son 50ns, 50ns, 60ns, 50ns y 50ns respectivamente. Debido a la penalidad de la segmentación estos tiempos aumentan en 5ns. Determinar la aceleración en condiciones ideales.

Máquina no segmentada: $\text{Tpo}_{\text{ejecución}} = 50\text{ns} + 50\text{ns} + 60\text{ns} + 50\text{ns} + 50\text{ns} = 260\text{ns}$

En la máquina segmentada el tiempo medio de ejecución está dado por la etapa más lenta más la penalización, esto es 65ns. Por lo tanto:

$$\text{Aceleración} = \frac{T_{\text{ejec.}}\text{Máquina no segmentada}}{T_{\text{ejec.}}\text{Máquina segmentada}} = \frac{260}{65} = 400\%$$

Implementación de la Segmentación

La implementación de la segmentación no es tan sencilla como se ha presentado hasta ahora ya que requiere de recursos adicionales. Por ejemplo, no es posible que la alu incremente el contador de programa, calcule una dirección efectiva y realice una operación aritmética o lógica al mismo tiempo. Sin embargo, la simplicidad de una máquina RISC hace relativamente fácil la evaluación de los recursos. Para esta evaluación dividiremos la cpu entres unidades: PC, Unidad de manejo de memoria y Unidad de Datos.

Etapa	Unidad PC	Memoria	Datapath
F	PC←PC+4	IR←Mem[PC]	
D	PC1←PC	IR1←IR	A←Rs1; B←Rs2
E			DMAR←A+##IR1[16:31] ó ALUoutput←Aop(B ó ##IR1[16:31]) ó ALUoutput←PC1+ ##IR1[16:31]; cond←(Rs1 op 0); SMDR←B;

M	if (cond) PC←ALUoutput	LMDR←Mem[DMA] ó Mem[DMAR]←SMDR	ALUoutput1←ALUoutput
W			Rd←ALUoutput1 ó LMDR

Cada etapa de la segmentación se activa en cada ciclo de reloj, esto impone los siguientes requerimientos:

- Todas las operaciones de la etapa se deben completar en un ciclo.
- Cualquier combinación de operaciones se puede presentar a la vez.

Las implicaciones más importantes de estos requerimientos son:

1. El PC se debe incrementar en cada ciclo de Reloj, lo que debe hacerse en F en vez de D. Esto requerirá un incrementador adicional dedicado, ya que la ALU está ocupada en cada ciclo y no puede utilizarse para incrementar el PC.

2. En cada ciclo se debe buscar una nueva instrucción, esto también se hace en F.

3. En cada ciclo de reloj se necesita una nueva palabra de datos, esto se hace en C.

4. Debe haber un MDR separado para cargas (LMDR) y almacenamiento (SMDR), ya que cuando son instrucciones consecutivas se traslapan en el tiempo.

5. Se necesitan tres Latches adicionales que contengan los valores que se necesitarán con posterioridad, pero que se puedan modificar con una instrucción posterior. Los valores almacenados son: la instrucción, la salida de la ALU y el siguiente PC.

Es posible que el mayor impacto de la segmentación se refleje en el sistema de memoria. Aunque el tiempo de acceso a memoria no haya cambiado, el ancho de banda máximo se debe aumentar cinco veces con respecto a la máquina no segmentada ya que se requieren como máximo dos accesos a memoria en cada ciclo, frente a dos accesos cada cinco ciclos en la máquina no segmentada, con el mismo número de pasos por instrucción. Para poder proporcionar dos accesos a memoria en cada ciclo es frecuente que las máquinas segmentadas utilicen caches separados para instrucciones y datos

Durante la etapa de ejecución (E), la ALU puede usarse para tres funciones diferentes: un cálculo de dirección efectiva, un cálculo de dirección de salto, o una operación ALU. Afortunadamente sólo una de estas funciones se llevará a cabo, así no surgen conflictos.

La segmentación, como se ha planteado hasta ahora, funcionaría sin problemas si las instrucciones fueran independientes entre sí. Sin embargo, esto no es así y se originan problemas adicionales.

EVENTOS EN CADA ETAPA DE LA SEGMENTACION

ETAPA	Instrucción ALU	Acceso a Memoria	Salto
F	IR←Mem[PC]; PC←PC+4	IR←Mem[PC]; PC←PC+4	IR←Mem[PC]; PC←PC+4
D	A←Rs1; B←Rs2; PCl←PC; IRl←IR	A←Rs1; B←Rs2; PCl←PC; IRl←IR	A←Rs1; B←Rs2; PCl←PC; IRl←IR
E	ALUoutput←AopB ó ALUoutput←Aop##IR[16:31]	DMAR←A+##IR[16:31]; SMDR←B	ALUoutput←PC + ##IR[16:31]; cond←(Rs1op0)
C	ALUoutputl←ALUoutput	LMDR←Mem[DMAR] ó Mem[DMAR]←SMDR	if (cond) PC←Aluoutput
W	Rd←ALUoutput1	Rd←SMDR	

PROBLEMAS DE LA SEGMENTACION

Existen tres situaciones, denominadas riesgos, que impiden que se ejecute la siguiente instrucción en una estructura segmentada. Los riesgos disminuyen el desempeño de la velocidad ideal lograda por la segmentación, los tres tipos de riesgos son:

i. Riesgos estructurales: Estos surgen de conflictos en el uso de los recursos, cuando el hardware no puede soportar todas las combinaciones posibles de ejecución de instrucciones traslapadas.

ii. Riesgos por dependencias de datos: Se producen cuando el resultado de una instrucción depende del resultado de una instrucción anterior.

iii. Riesgos de control: Surgen de la segmentación de los saltos y otras instrucciones que modifican el PC.

Los riesgos en la segmentación pueden hacer que la máquina se detenga, la detención en una máquina segmentada difiere respecto a una máquina no segmentada en el hecho de que hay varias instrucciones en ejecución simultáneamente. Esto significa que algunas instrucciones siguen en ejecución, las anteriores a la instrucción detenida, pero no se buscan (Fetch) nuevas instrucciones.

Una detención significa una degradación del desempeño, esto puede evaluarse como:

$$\text{Aceleración} = \frac{\text{Tiempo medio sin segmentación}}{\text{Tiempo medio con segmentación}}$$

$$= \frac{\text{CPI sin segmentación} \quad * \text{ Ciclo sin segmentación}}{\text{CPI con segmentación} \quad * \text{ Ciclo con segmentación}}$$

Si consideramos la segmentación como disminución del CPI, el CPI ideal de una máquina segmentada es:

$$\text{CPI ideal} = \frac{\text{CPI sin segmentación}}{\text{Profundidad de la segmentación}}$$

Sustituyendo este valor,

$$\text{Aceleración} = \frac{\text{CPIideal} * \text{profundidad} * \text{Ciclo sin segmentación}}{\text{CPI con segmentación} * \text{Ciclo con segmentación}}$$

Por otro lado:

CPI con segmentación=CPIideal*Ciclos de detención por instrucción

Reemplazando:

$$\text{Aceleración} = \frac{\text{CPIideal} * \text{Pr ofundidad}}{\text{CPIideal} + \text{Ciclos de detención por inst .}} * \frac{\text{Ciclo sin seg .}}{\text{Ciclo con seg .}}$$

Como normalmente Ciclo con seg. $\approx$ Ciclo sin seg. puede simplificarse esta expresión.

Riesgos Estructurales :

Se producen por segmentaciòn incompleta o falta de duplicaciòn de unidades funcionales. Por ejemplo, muchas màquinas segmentadas comparten un ùnico puerto de memoria para datos e instrucciones. Esto significa que cada vez que una instrucción contenga una referencia a memoria no puede iniciarse un nuevo Fetch deteniendo la segmentación durante un ciclo.

inst\ciclo	1	2	3	4	5	6	7	8	9
Ref. a Memoria	F	D	E	M	W				
i+1		F	D	E	M	W			
i+2			F	D	E	M	W		
i+3				*****	F	D	E	M	W
i+4						F	D	E	M

La instrucción i+3 no puede iniciarse en el ciclo 4 ya que el único puerto a memoria está siendo usado por la instruccón que hace referencia a memoria.

EJEMPLO: una máquina segmentada que posee un único puerto de acceso a memoria tiene un CPI = 1.2. Cuántas veces más rápidas será una máquina sin este riesgo estructural si los accesos a memoria constituyen un 30% de las instrucciones.

La máquina sin riesgos será más rápida según la relación de las aceleraciones.

$$\text{Aceleración sin riesgos} = \frac{\text{CPI} * \text{Profundidad}}{\text{CPI} + \text{Ciclos detención}} * \frac{\text{Css}}{\text{Ccs}}$$

$$= \frac{1.2 * \text{profundidad}}{1.2 + 0} \frac{\text{Css}}{\text{Ccs}} = \text{profundidad} \ \frac{\text{Css}}{\text{Ccs}}$$

$$\text{Aceleración con riesgos} = \frac{1.2 \ * \ \text{profundidad}}{1.2 + 0.3} \frac{\text{Css}}{\text{Ccs}}$$

$$\frac{\text{Aceleración sin riesgos}}{\text{Aceleración con riesgos}} = \frac{\text{Pr ofundidad}}{\frac{1.2 * \text{profundidad}}{1.5}} \frac{\text{Css} / \text{Ccs}}{\text{Css} / \text{Ccs}} = \frac{1.5}{1.2 = 1.25}$$

Por lo tanto la máquina sin riesgos es 25% más rápida

Riesgos por dependencias de datos

Se producen cuando una instrucción tiene como fuente un operando que es destino en una instrucción anterior, por ejemplo en la siguiente secuencia.

```
ADD  R1,R2,R3
SUB  R4,R1,R2
```

Instrucción\ciclo	1	2	3	4	5	6

ADD	F	D	E	M	W	
SUB		F	D	E	M	W

La instrucción SUB utiliza el operando R1 calculado en la instrucción ADD, el oeprando R1 es leído por la instrucción SUB en el ciclo 3; sin embargo el valor correcto es escrito por la instrucción ADD en el ciclo 5, si no se toman medidas para evitarlo, la instrucción SUB leerá el valor anterior (incorrecto) de R1.

Afortunadamente, existen varias técnicas que permiten evitar este problema:

Detención : Mediante el uso de comparadores se determina que una instrucción intenta utilizar un operando que no esta listo, y detiene la ejecución de la instrucción hasta que el operando necesario esté listo.

Para el caso anterior :

Instrucción	1	2	3	4	5	6	7	8	9	10	11
ADD	F	D	E	M	W						
SUB		F	-	-	-	D	E	C	W		
---			-	-	-	F	D	E	C	W	
---				-	-	-	F	D	E	C	W

Esta técnica es muy fácil de implementar pero su principal desventaja es la degradación del desempeño de la máquina, si la segmentación es muy profunda la degradación puede ser enorme.

Reordenamiento de Instrucciones: En este caso el compilador debe detectar el riesgo y reordenar las instrucciones de manera que no se produzca detención. Por ejemplo:

Original	Reordenado
ADD R1,R2,R3	ADD R1,R2,R3
SUB R4,R1,R2	OR R5,R2,R6
OR R5,R2,R6	AND R7,R2,R3
AND R7,R2,R3	XOR R8,R9,R6
XOR R8,R9,R6	SUB R4,R1,R2

	1	2	3	4	5	6	7	8	9
ADD	F	D	E	M	W				
OR		F	D	E	M	W			
AND			F	D	E	M	W		

XOR	F	D	E	M	W	
SUB		F	D	E	M	W

Cuando el cauce es muy profundo, puede resultar difícil encontrar las instrucciones necesarias para llenar los espacios requeridos, en algunas máquinas si no es posible encontrar instrucciones útiles se bloquea la búsqueda de nuevas instrucciones (FETCH) por hardware, en otro casos donde no hay etapas de bloqueo, el compilador debe introducir NOP'S para rellenar los espacios.

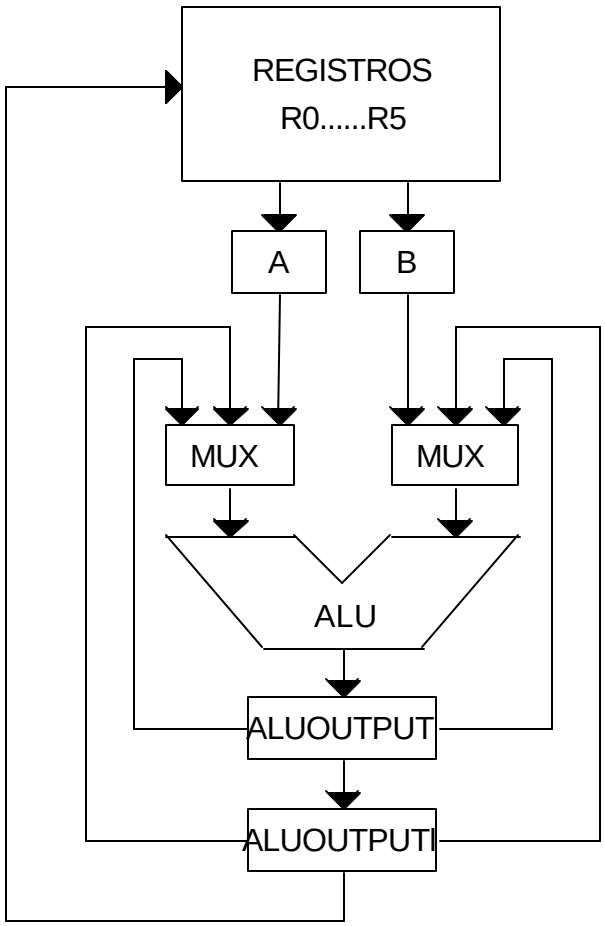
Adelantamiento (Forwarding): Como hemos visto en el problema planteado originalmente:

```
ADD R1,R2,R3
SUB R4,R1,R2
```

Instrucción\ciclo	1	2	3	4	5	6
ADD	F	D	E	M	W	
SUB		F	D	E	M	W

Los latches A y B se cargan en el ciclo 3 y el valor de R1 necesario no está disponible en R1 hasta el ciclo 6. Sin embargo, observando la tabla el valor calculado por la operación ADD está disponible al final de la etapa de ejecución (ciclo 3) en el latch aluoutput. Por otro lado el operando R1 de la instrucciión sub sólo se necesita al comienzo de la etapa de ejecución de la instrucción SUB (ciclo 4). Por lo tanto, cuando realmente se necesita el resultado de la suma, está disponible; sólo que se encuentra en otro lugar. El adelantamiento (Forwarding) consiste en desviar el valor calculado por la alu, directamente a la entrada de ésta, sin esperar su retroescritura en el registro destino.

Esta técnica de Hardware, se implementa usado los latches de salida de la alu (aluoutput y aluoutputl) y un par de multiplexores.



En una máquina segmentada, puede ser necesario adelantar resultados no solo a la instrucción inmediatamente siguiente sino también a las que siguen. Por ejemplo:

Inst\Ciclo	1	2	3	4	5	6	7	8	9
ADD R1,R2,R3	F	D	E	M	W				
SUB R4,R1,R5		F	D	E	M	W			
AND R6,R1,R7			F	D	E	M	W		
OR R8,R1,R9				F	D	E	M	W	
XOR R10,R11,R1					F	D	E	M	W

El resultado de la instrucción ADD se debe adelantar a las tres instrucciones siguientes. La instrucción XOR (y las siguientes) lee el resultado desde el fichero en el ciclo 6, como el resultado es escrito en el ciclo 5 no requiere adelantamiento.

PREDICCION DEL DESTINO DEL SALTO

Predecir destino y seguir ese camino.

⇨ Resultados intermedios marcados como tentativos.

- Hasta determinar instrucción destino.

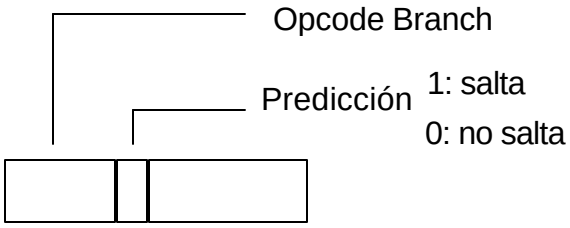
- No se modifica estado de CPU en forma permanente.

Predicción Correcta.
⇨ Resultados tentativos se trasnforman en efectivos.

Predicciones incorrectas.
⇨ Resultados tentativos anulados.
⇨ Operaciones tentativas en desarrollo son canceladas.

Técnica efectiva si predicciones mayoritariamente correctas.
⇨ Buenas predicciones si se conoce estilo y características de programa.
⇨ Saltos al final (comienzo) del lazo iterativo.

Predicciones estáticas.
⇨ Mas simple.
⇨ Requiere ser validado c/programas reales.
⇨ Poco Hardware adicional.



Predicción Dinámica

Adaptarse al comportamiento dinámico del programa.
⇨ Idea similar a memoria Cache.

Acumular historia sobre saltos.
⇨ Tabla histórica
  - Branch - History Table
  - Branch - Prediction Table
⇨ Tabla acumular comportamiento histórico reciente de cada salto.
  - Los más recientes.
  - Predecir comportamiento idéntico al anterior.
⇨ Tabla similar a Cache.
  - Accesada al decodificar instrucción.

- Indica predicción en base a comportamiento previo.
- Actualizada según resultado de instrucción.
- Puede indicar directamente.
    - ♦ Direcc. instrucción destino.
    - ♦ Instrucción destino.

MODELO PARA INICIACION DE INSTRUCCIONES CONCURRENTES

Las intrucciones pueden iniciarse si hay recursos disponibles y no hay conflictos entre operandos . Dos instrucciones S_i y S_j pueden ejecutarse si y solo si:

⇨ Los resultados no se guardan en el mismo lugar
⇨ Resultado de una no es operando de la otra.

Si I_i: Dominio de S_i y θ_i: Rango de S_i pueden darse cuatro situaciones de las cuales solo tres representan conflicto.

RAR.: $I_j \cap I_i \neq 0$
RAW: $I_j \cap \theta_i \neq 0$
WAR: $\theta_j \cap I_i \neq 0$
WAW. $\theta_j \cap \theta_i \neq 0$

La primera situación no representa conflicto, pero las tres restantes si.