Tema 5. Segmentación: conceptos básicos

Organización de Computadores

LUIS ENRIQUE MORENO LORENTE
RAÚL PÉRULA MARTÍNEZ
ALBERTO BRUNETE GONZALEZ
DOMINGO MIGUEL GUINEA GARCIA ALEGRE
CESAR AUGUSTO ARISMENDI GUTIERREZ
JOSÉ CARLOS CASTILLO MONTOYA

Departamento de Ingeniería de Sistemas y Automática

CONCEPTO

- El concepto básico de la segmentación es el de la división del trabajo
  - Aparece por vez primera en la industria del automóvil, en la producción del Ford T. En este contexto se denomino producción en cadena.
  - La mayoría de los sectores industriales con alta producción manufacturera utilizan la producción en cadena para abaratar los costes de producción.
  - Algo similar ocurre en el campo de los computadores, aunque históricamente es algo posterior.

COMPUTADORES SEGMENTADOS

- Ejecutan billones de instrucciones, por lo que su productividad (throughput) es lo que importa
- Entre las características deseables (MIPS - DLX):
  - RISC -> Simple.
  - Todas las instrucciones tengan la misma longitud,
  - Los registros estén localizados siempre en la misma posición del formato de instrucción (formato fijo de instrucción).
  - Que sólo haya operandos de memoria en los loads y stores.
    - Sólo estas instrucciones acceden a memoria.

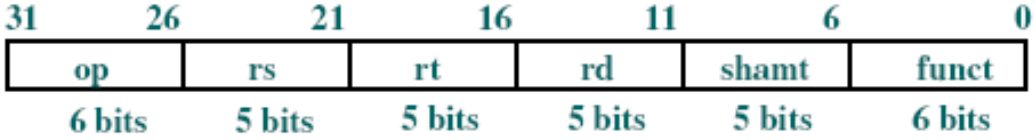
UN RISC "TÍPICO"

- Formato de instrucción fijo de 32-bit (3 formatos)
- 32 Registros de propósito general (GPR) de 32-bit (R0 contiene cero, DP se toman en pares)
- Instrucciones aritméticas reg-reg de 3-direcciones
- Un único modo de direccionamiento en los load/store:
  - reg base + desplazamiento
  - no hay direccionamientos indirectos
- Condiciones de salto simples
- Saltos retardados

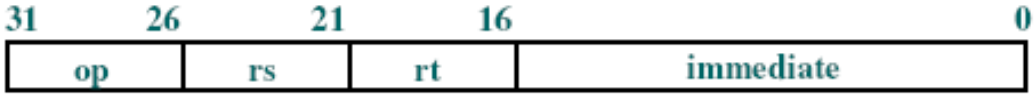
Ejemplos: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

MIPS: REPERTORIO DE INSTRUCCIONES

- R-type (register insts)



- I-type (Load, Store, Branch, inst's w/imm data)



- J-type (Jump)

op: operation of the instruction
rs, rt, rd: the source and destination register specifiers
shamt: shift amount
funct: selects the variant of the operation in the "op" field
immediate: address offset or immediate value
target address: target address of the jump instruction

37

SEGMENTACION DE UN PROCESADOR

- Queremos dividir en varias etapas las operaciones que se realizan en el flujo de datos de la máquina de forma que:
  - El tiempo en cada etapa sea aproximadamente el mismo
  - Cada etapa sea utilizada como mucho una vez en cada instrucción (cauce lineal)
  - Los registros puedan diseñarse con claridad, para mantener la información de estado de las instrucciones entre las operaciones.
  - Que no haya demasiadas etapas como para que el tiempo perdido en los latch llegue a predominar
  - Que haya suficientes etapas como para que se consiga una ganancia significativa en las prestaciones

SEGMENTACION DEL PROCESADOR DLX

- Para este procesador simple, se estructurará en 5 etapas:
  - Etapa IF (Instruction fetch): búsqueda de la instrucción
  - Etapa ID (Instruction decode and register fetch): descodificación de la instrucción y búsqueda de operandos en los registros.
  - Etapa EX (Execution): ejecución de las operaciones en la ALU
  - Etapa MEM (Memory access): acceso a memoria.
  - Etapa WB (Write back): escritura de los resultados en los registros.
- Recuerde que:
  - Es un proc. RISC (simplicidad del juego de instr.)
  - Acceso a las caches en un ciclo
  - Dos caches: una de instrucciones y una de datos
  - Integradas en el cauce

RUTA DE DATOS BÁSICA DLX (5 ETAPAS)

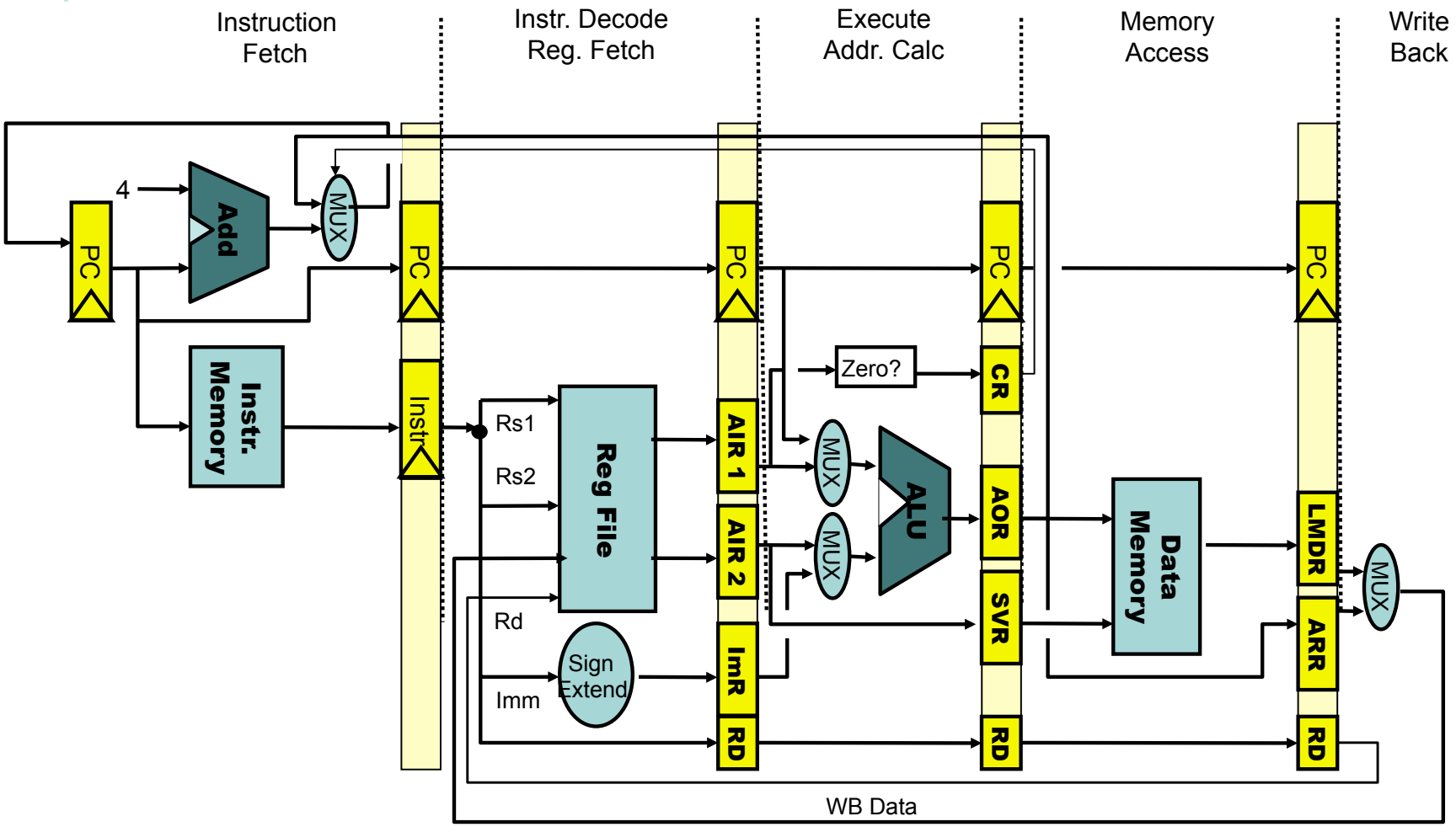
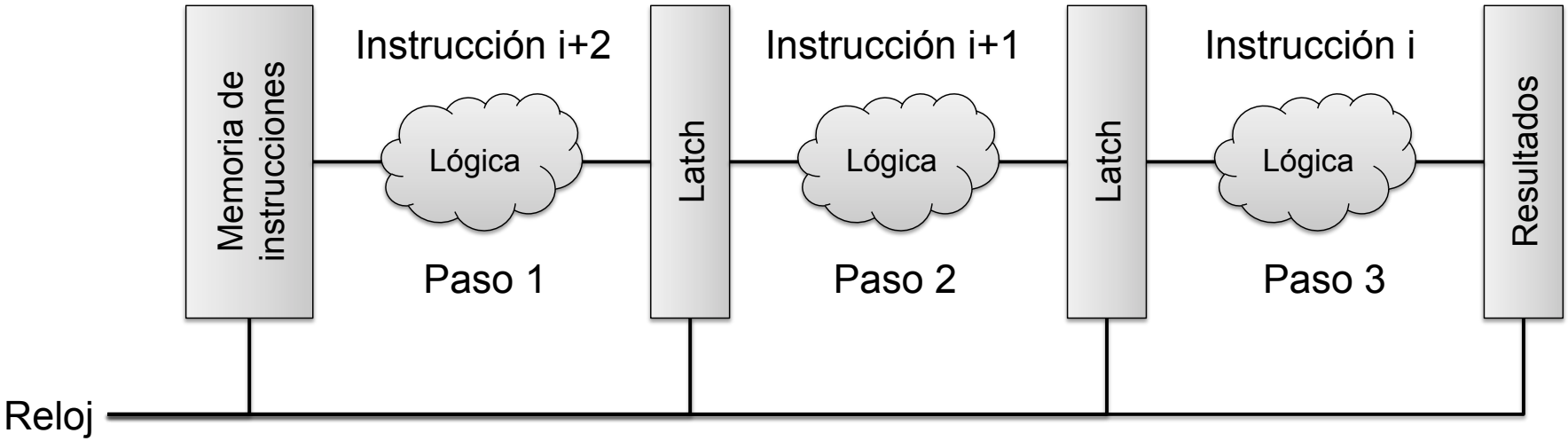
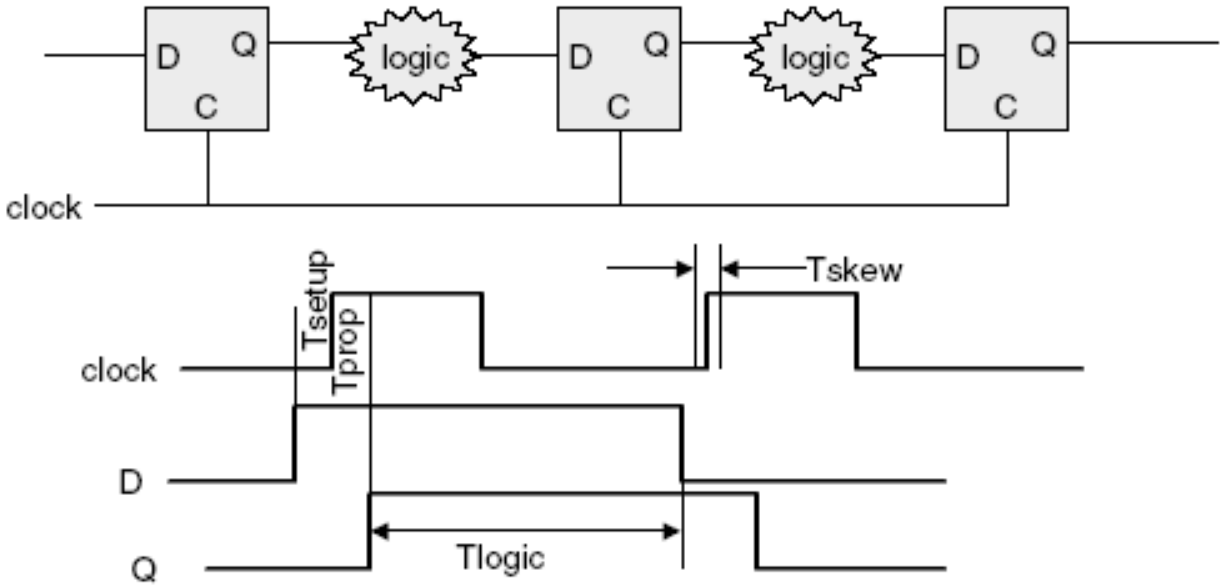


Imagen de Wikipedia, 2014

TEMPORIZACIÓN DEL CAUCE



TEMPORIZACIÓN: SEÑALES

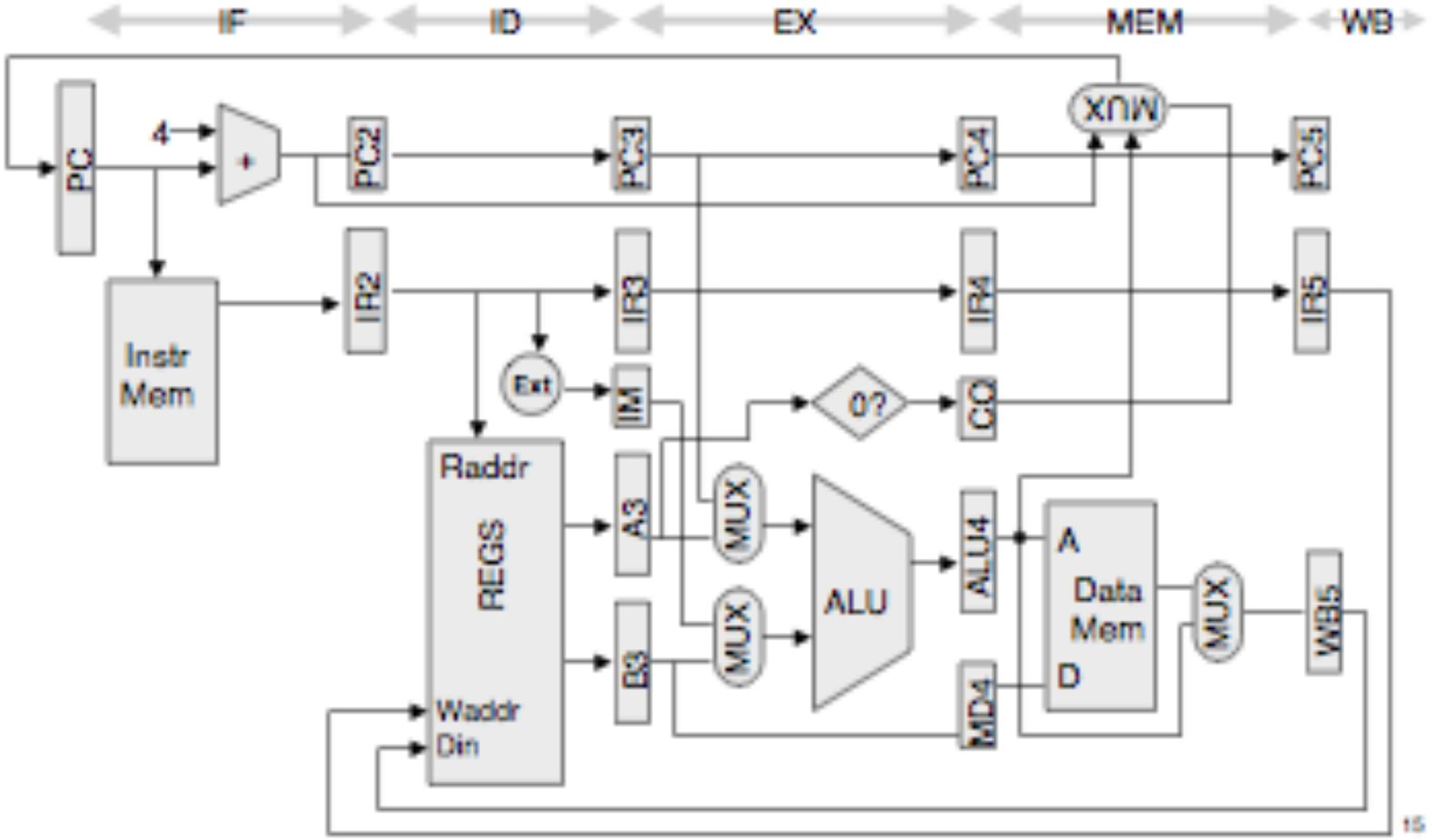


Period >= Tlogic + Tsetup + Tprop + Tskew

FLUJO DE LOS DATOS EN EL CAUCE SEGÚN EL TIPO DE INSTRUCCIÓN (I)

	Reg-Reg ALU	*Reg-immed ALU*	*Load*	*Store*	*Branch*	*Jump*
IF	IR2=IMem[PC]; PC2 = PC; PC=PC+4					
ID	A3=Regs[IR25..21]; B3=Regs[20..16]; IR3=IR2; PC3=PC2; IM3=IR2[15]^{16}##IR2[14..0];					
EX	ALU4= A3 op B3; IR4=IR3; PC4=PC3;	ALU4= A3 op IM3; IR4=IR3; PC4=PC3;	ALU4 =A3 + IM3; IR4=IR3; PC4=PC3; MD4=B3;		ALU4 =PC3+IM3; CO4= A3 op 0; IR4=IR3; PC4=PC3;	ALU4 = PC3+IM3; IR4=IR3; PC4=PC3;
MEM	IR5=IR4; PC5=PC4;	IR5=IR4; PC5=PC4;	WB5 = DMem[ALU4];	DMem[ALU4] = MD4;	IR5=IR4; PC5=PC4; if (CO4) PC=ALU4;	IR5=IR4; PC5=PC4; PC=ALU4;
WB	Din=WB;	Din=WB;	Din=WB;			

FLUJO DE LOS DATOS EN EL CAUCE SEGÚN EL TIPO DE INSTRUCCIÓN(II)



DLX: FASE DE BÚSQUEDA DE INSTRUCCIÓN (IF)

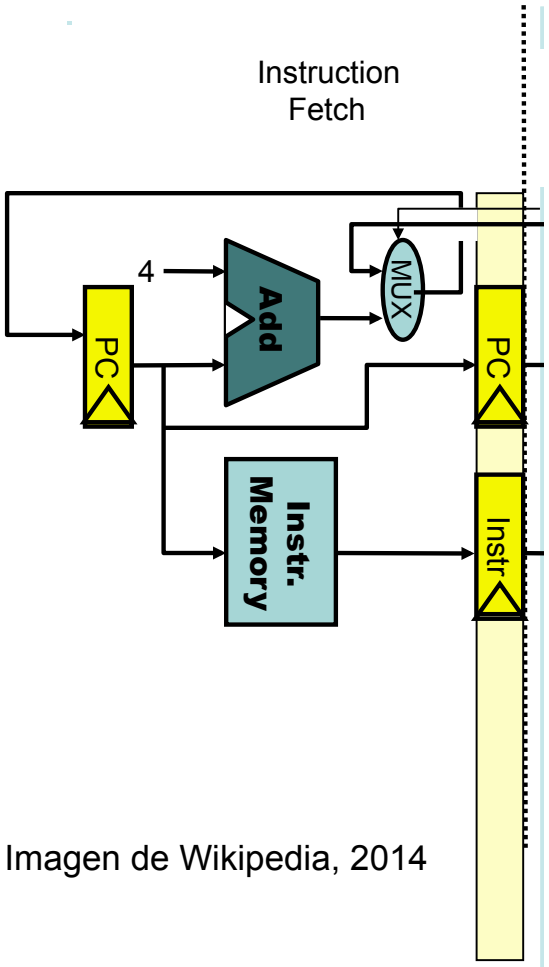
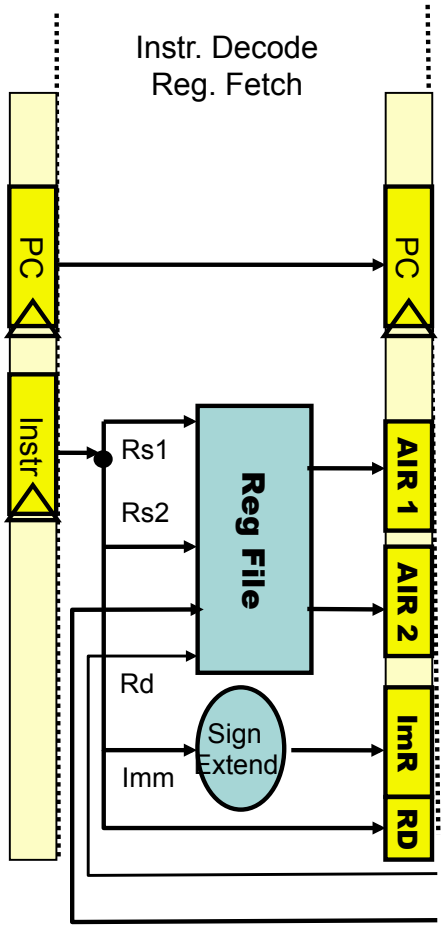


Imagen de Wikipedia, 2014

- Se busca la instrucción a la que apunta el PC en la caché de instrucciones.
  - Se lleva al registro de instrucción.
- Se lleva el PC al latch de interfaz entre etapas.
- Se incrementa el PC en 4.
  - Se lleva de nuevo al PC.
  - Salvo que una instrucción previa de transferencia de control salto requiera su modificación, el multiplexor elije en este caso la dirección objetivo del salto.

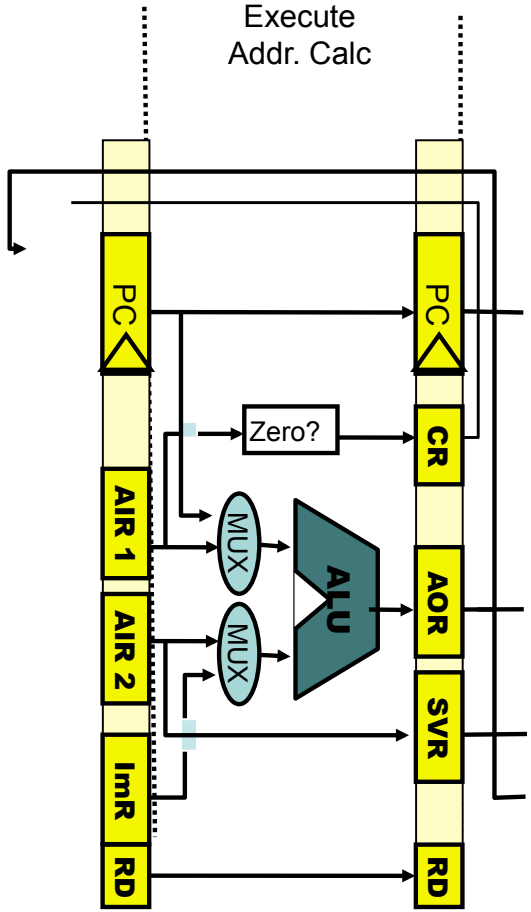
DLX: FASE DE DESCODIFICACIÓN Y LECTURA DE OPERANDOS (ID)



- Se descodifica la instrucción en la 1ª mitad del ciclo. Se leen los registros en la 2ª mitad.
- Dependiendo del tipo de instr. se realiza lo siguiente en la 2ª mitad del ciclo:
  - Registro-registro (aritméticas/lógicas)
    - Se leen los registros Rs1 y Rs2 del banco de registros y se llevan a los reg. de entrada a la ALU (AIR1 y AIR2).
    - El registro destino se lleva a RD
  - Referencia a memoria (load/store)
    - Se lee el registro base de la dirección de memoria y se lleva al reg de entrada a la alu 1 (Air 1).
    - Se extiende el signo al desplazamiento y se lleva al inmediate register (IMR).
    - Si es un store se lee el registro a almacenar en memoria y se lleva al r. de entrada a la alu 2 (AIR2).
  - Transferencia de control (branch)
    - Se extiende el signo al desplazamiento y se lleva al inmediate register (IMR).
    - Se lee el registro que determina la dirección base del salto y se lleva al reg de entrada a la ALU (AIR1).

Imagen de Wikipedia, 2014

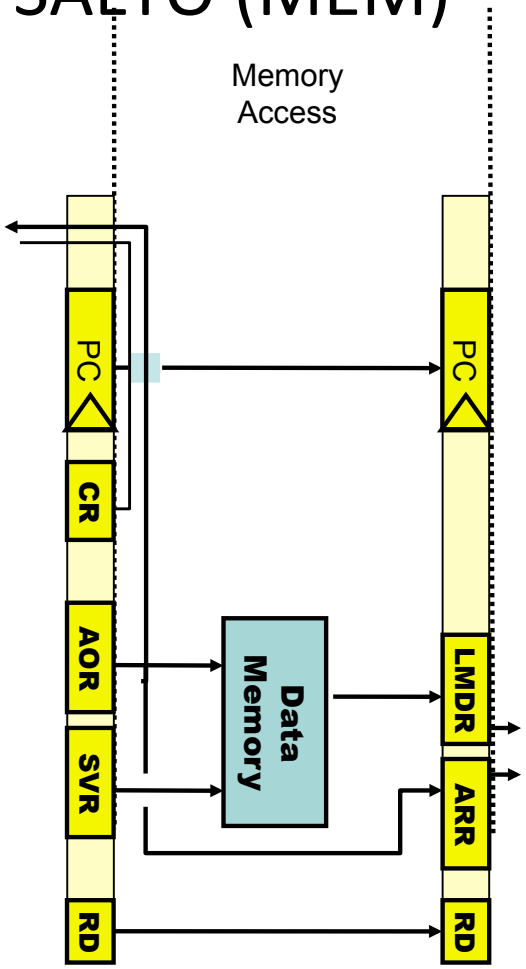
DLX: FASE DE EJECUCIÓN (EX)



- La ALU ejecuta la operación sobre los datos situados en :
  - Los reg de entrada a la alu (Air 1y Air 2) (arit./log)
  - El PC y el Imr (direc. de salto relativa a PC)
  - El Air 1 y el Imr (direc de salto relativa a reg o direc de memoria)
- Dependiendo del tipo de instr. :
  - Registro-registro (aritméticas/lógicas)
    - El reg de salida de Alu contiene el resultado de la operación sobre los registros Rs1 y Rs2, y se lleva al reg de salida de resultados (AOR).
  - Referencia a memoria (load/store)
    - El reg de salida de Alu contiene la dirección efectiva de memoria (a leer o escribir), y se lleva al reg de salida de resultados (AOR).
    - Si es una instr store, el contenido del Air 2 (reg a almacenar en mem) se lleva al store value reg (SVR).
  - Transferencia de control (branch)
    - El reg de salida de Alu contiene la dirección efectiva del salto, y se lleva al reg de salida de resultados (AOR).
    - Se determina si la condición de salto se verifica o no y se lleva al condition reg (CR).

Imagen de Wikipedia, 2014

DLX: FASE DE ACCESO A MEMORIA / FINAL. DE SALTO (MEM)



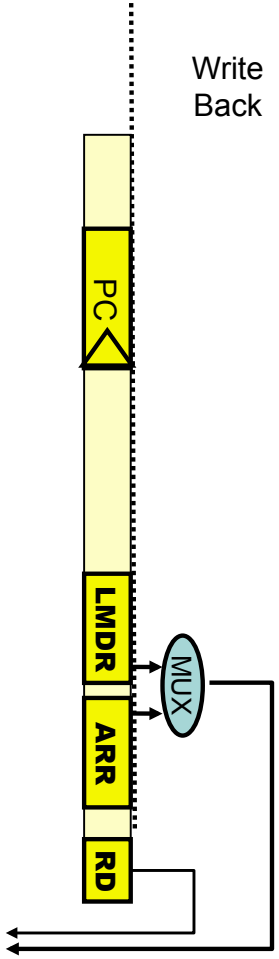
- Se realizan operaciones sólo en el caso de instrucciones load, store y branch:

Dependiendo del tipo de instr.:

- Registro-registro (aritméticas/lógicas)
  - El reg de salida de alu (AOR) se transfiere la alu result register (ARR).
- Referencia a memoria (load/store)
  - Load: se lee el dato de la caché de datos (AOR contiene la dirección efectiva) y lo leído se sitúa en el load memory data reg (LMDR).
  - Store: se escribe en la memoria (AOR contiene la dirección efectiva) el dato contenido en el SVR en la caché de datos.
- Transferencia de control (branch)
  - Tanto para los saltos incondicionales como para los condicionales tomados se modifica el PC de la etapa inicial IF por el contenido del registro de salida de la alu AOR.
  - Para los saltos condicionales no tomados, el PC no se modificará. En el mux de la etapa IF se elije el PC+4.

Imagen de Wikipedia, 2014

DLX: FASE DE ESCRITURA DEL RESULTADO (WB)



- Durante esta fase el resultado de la ejecución de una instrucción registro-registro o load a memoria es almacenado en el banco de registros de la etapa ID.
- Los registros se escriben en la 1ª mitad del ciclo.
- Esta operación se realiza durante la 1ª mitad del ciclo.
- Dependiendo del tipo de instr. :
- Registro-registro (aritméticas/lógicas)
  - El alu result register (ARR) se escribe en el banco de registros (en el registro RD) de la etapa ID.
- Referencia a memoria (load)
  - El dato que está en el load memory data reg (LMDR) se escribe en el banco de registros (en el registro RD) de la etapa ID.

Imagen de Wikipedia, 2014

INFORMACIÓN DE CONTROL EN DLX

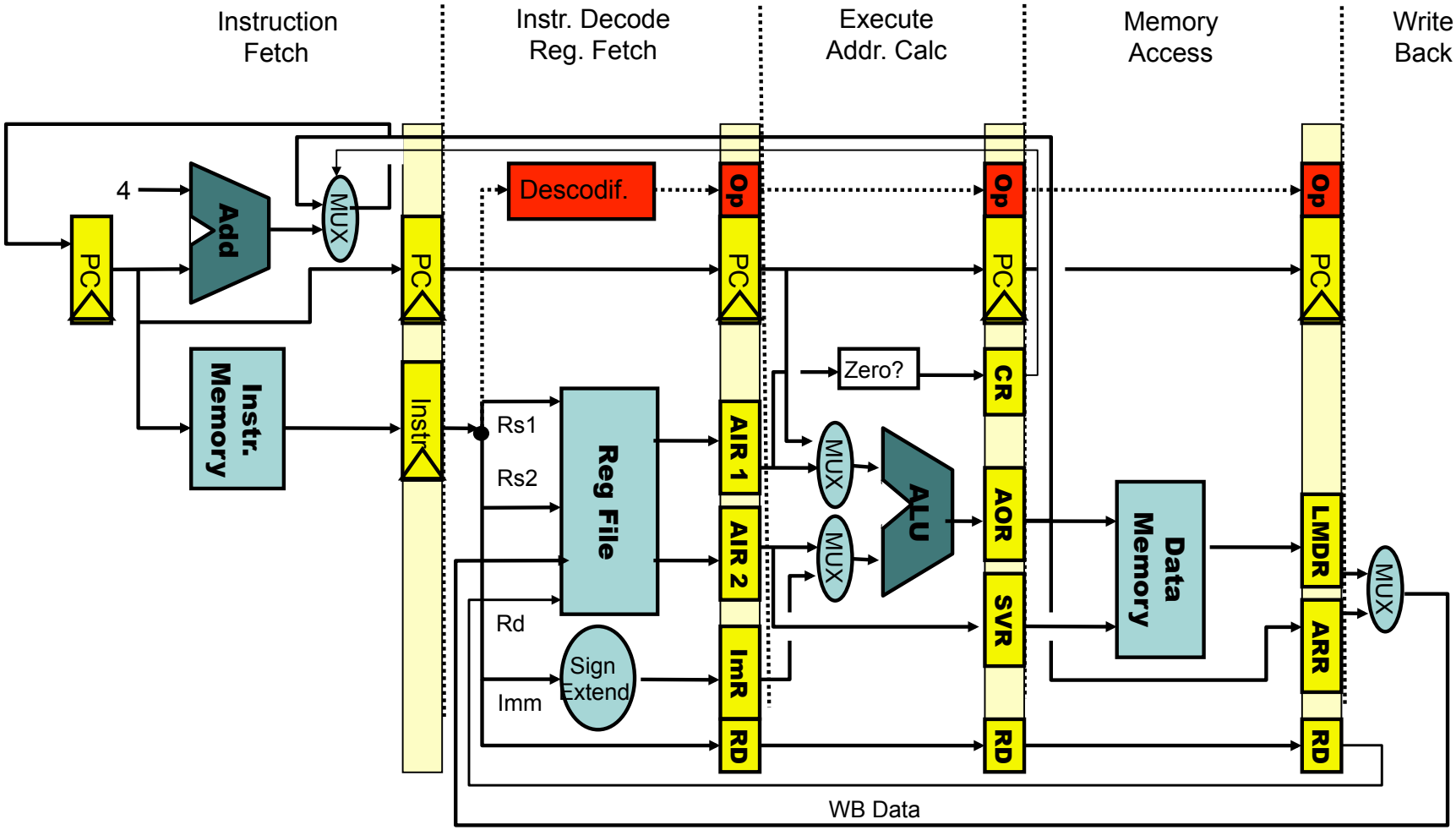


Imagen adaptada de Wikipedia, 2014

CRONOGRAMA (IDEAL) DE UNA EJECUCIÓN SEGMENTADA

Instruction	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	MEM	WB				
i+1		IF	ID	EX	MEM	WB			
i+2			IF	ID	EX	MEM	WB		
i+3				IF	ID	EX	MEM	WB	
i+4					IF	ID	EX	MEM	WB

Un funcionamiento ideal de cauce originaría el cronograma anterior.

DISEÑO DEL CAUCE

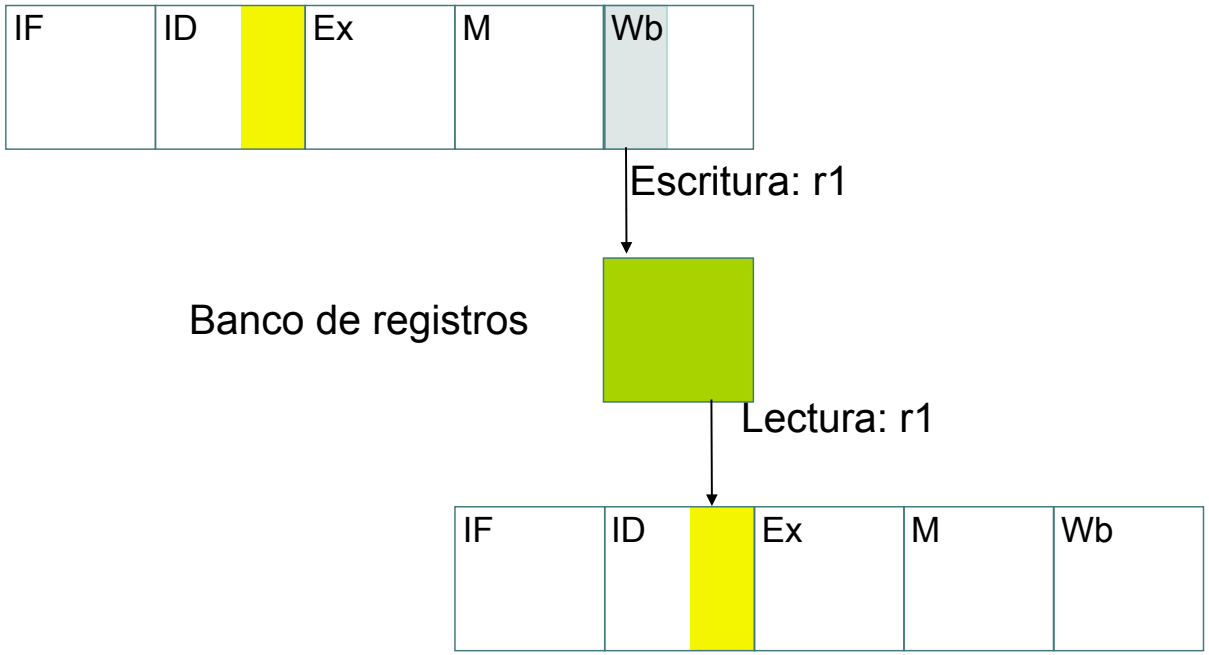
- Cada etapa del cauce debe ser independiente, por lo que resulta imprescindible la existencia de registros entre las etapas con objeto de mantener:
  - Los valores de los datos
  - Las señales de control, que incluyen:
    - Campos de la instrucción descodificada
    - Señales de control de los MUX
    - Señales de control de la ALU
- Este tipo de estructura de control se denomina estacionaria en datos
  - La información de control se mueve internamente por el cauce a la vez que los datos que se van utilizando u obteniendo en la ejecución de la instrucción.

PROBLEMA DEL BANCO DE REGISTROS

- En el diseño se ha separado en dos el acceso a la memoria
  - Existe una cache de instrucciones en la etapa IF, de donde se leen las instrucciones
  - Existe una cache de datos en la etapa MEM, donde se leen y escriben los datos
- Esto no resulta posible con el banco de registros, ya que tiene que ser leído y escrito
  - El problema se origina en que la lectura se realiza en la etapa ID y la escritura en la etapa WB
- Solución: separar en dos subfases los ciclos y realizar ambas operaciones en subfases diferentes
  - La escritura en el registro se hace en la 1ª mitad del ciclo de la etapa WB
  - La lectura de los registros se hace en la 2ª mitad del ciclo de la etapa ID
  - La primera mitad de la etapa ID se utiliza para descodificar la instrucción

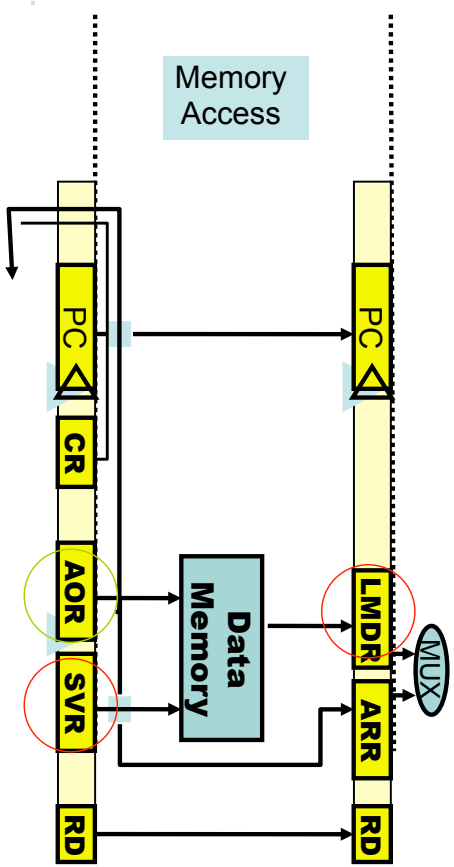
EJEMPLO DE SOLAPAMIENTO

ld r1, 8 (r2)



add r5, r1, r3

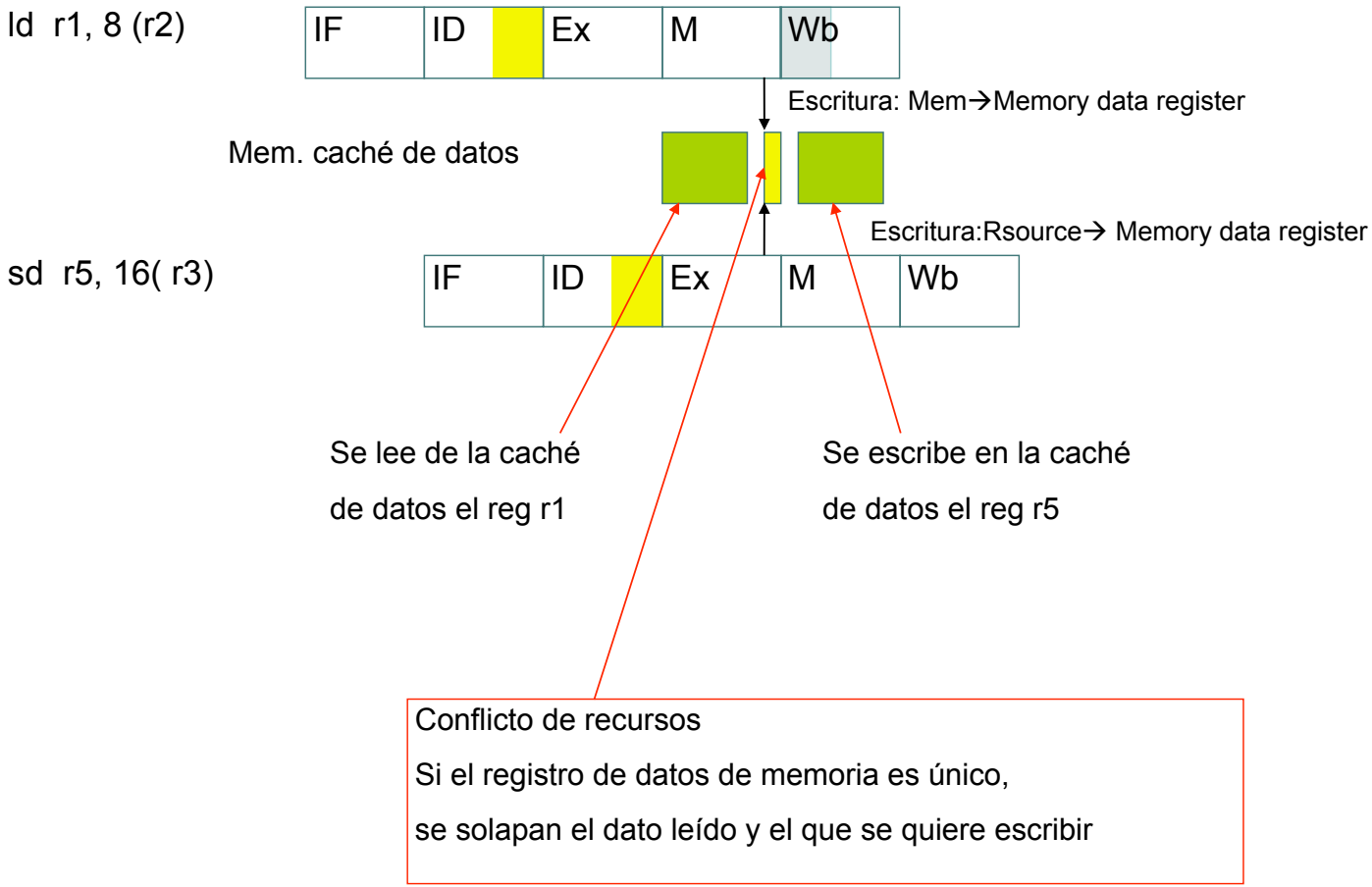
PROBLEMA DE LA CACHÉ DE DATOS



- En el diseño que se ha realizado del cauce se ha separado en dos registros el acceso de datos a la memoria caché de datos.
  - Si la operación es de lectura de la caché de datos, en la etapa MEM, se usa el registro de direccionamiento de la memoria AOR, y se dejan los datos leídos en el LMDR.
  - Si la operación es de escritura, la dirección en la que se va a escribir está en el registro AOR, y el dato que se va a escribir está contenido en el SVR.
  - El problema se origina en que a diferencia de la caché de instrucciones donde hay un único registro de direccionamiento de memoria y un registro de datos para leer o escribir en ella, en la de datos, si sólo hubiese un registro para acceder a la caché , cuando se realiza una lectura seguida de una escritura en la caché habría un conflicto de recursos en el mismo.

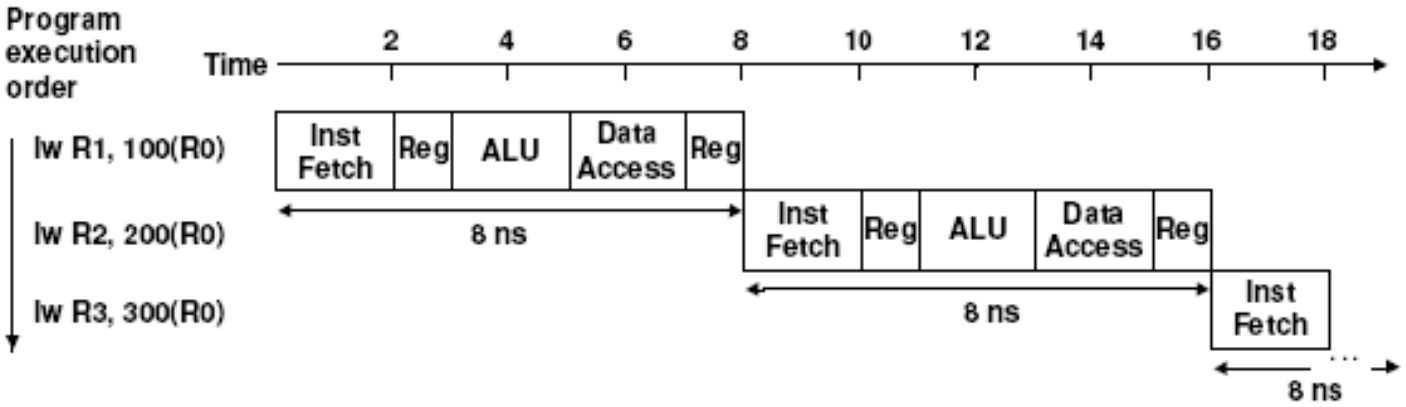
Imagen de Wikipedia, 2014

EJEMPLO DE SOLAPAMIENTO EN LA CACHÉ DE DATOS

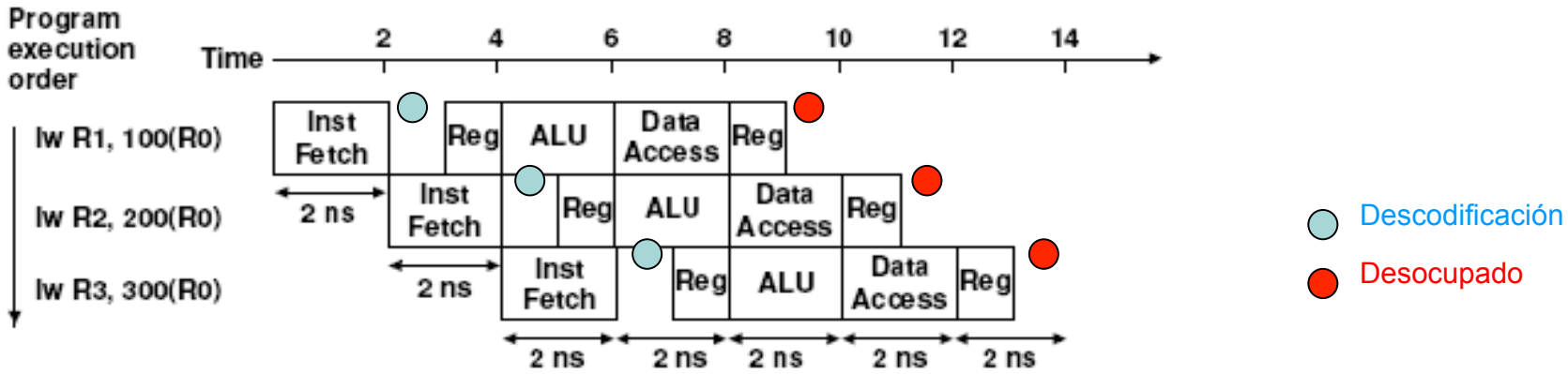


SEGMENTACIÓN DE LAS INSTRUCCIONES

Ejecución secuencial



Ejecución segmentada



CPU CON EJECUCIÓN EN UN SOLO CICLO (SIN SEGMENTAR)

IFetch | I.Decode/Reg. fetch | Execute/Addr. | Memory Access | Write back

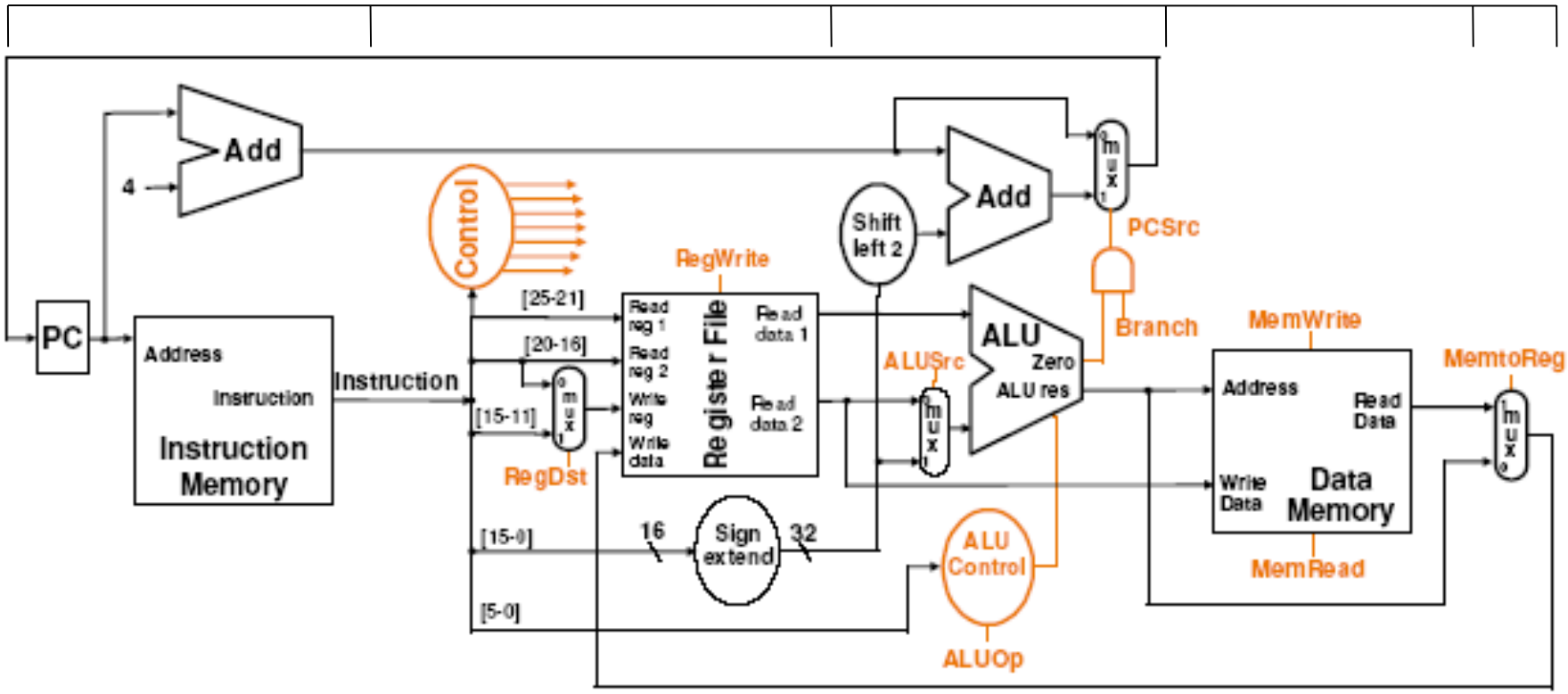
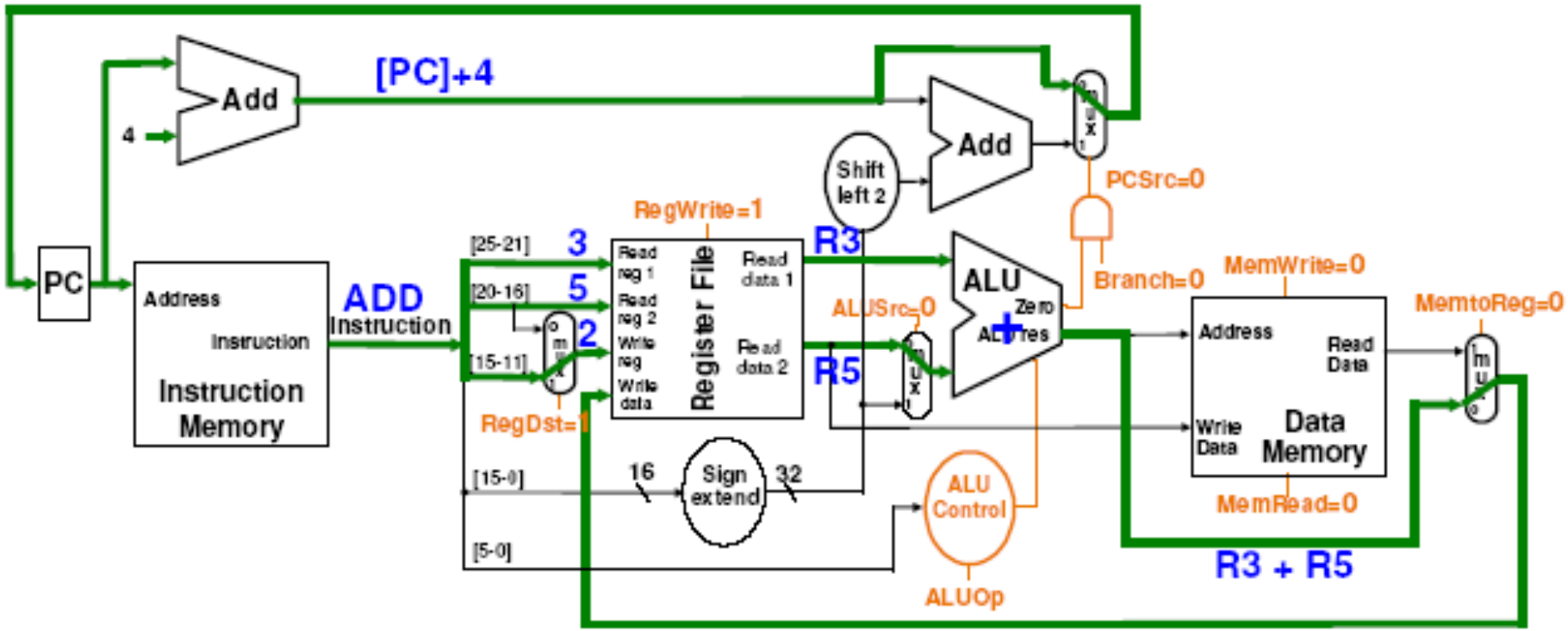


Imagen de Patterson, 1997

EJEMPLO: EJECUCIÓN DE UNA INSTRUCCIÓN "ADD"

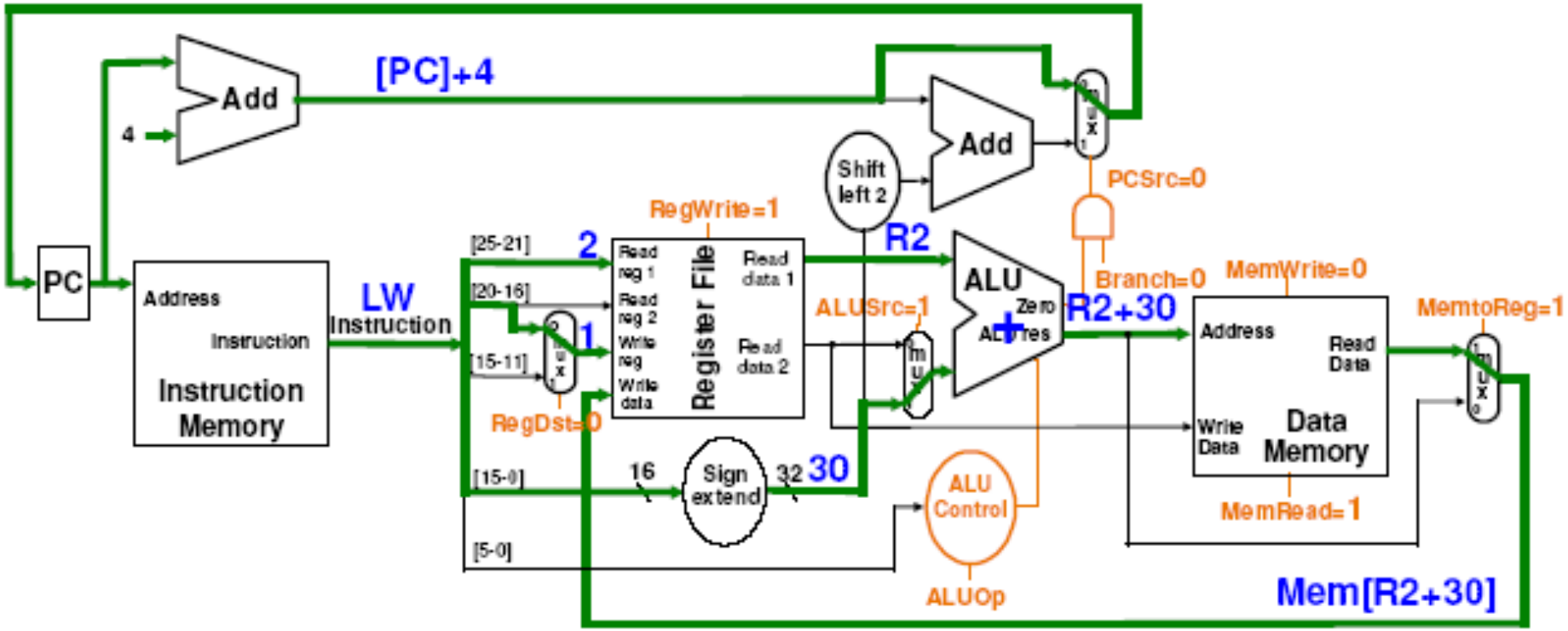


Add R2, R3, R5 ; R2 ← R3+R5

31 op 26	rs 21	rt 16	rd 11	shamt 6	funct 0
ALU	3	5	2	0	0 = Add

Imagen de Patterson, 1997

EJEMPLO: EJECUCIÓN DE UNA INSTRUCCIÓN "LOAD"

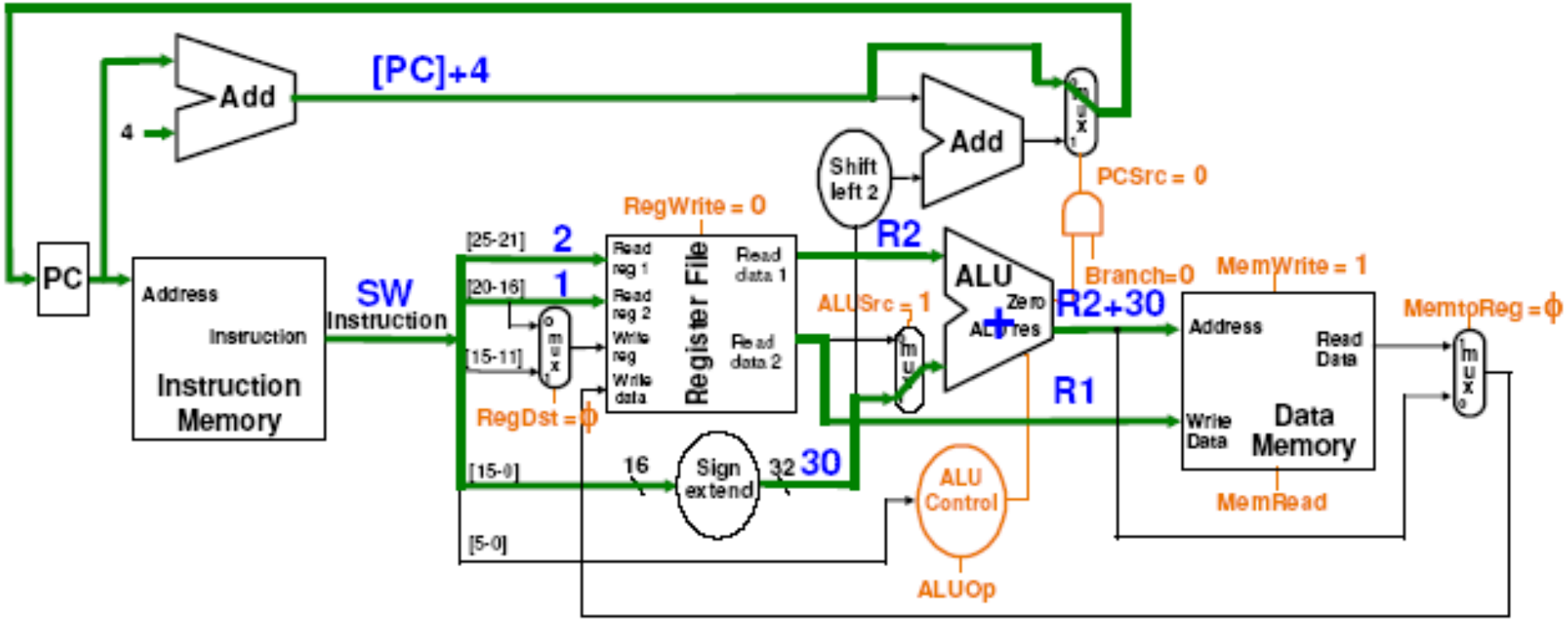


LW R1, (30)R2 ; R1 ← Mem[R2+30]

31 op 26	rs 21	rt 16	immediate 0
LW	2	1	30

Imagen de Patterson, 1997

EJEMPLO: EJECUCIÓN DE UNA INSTRUCCIÓN "STORE"

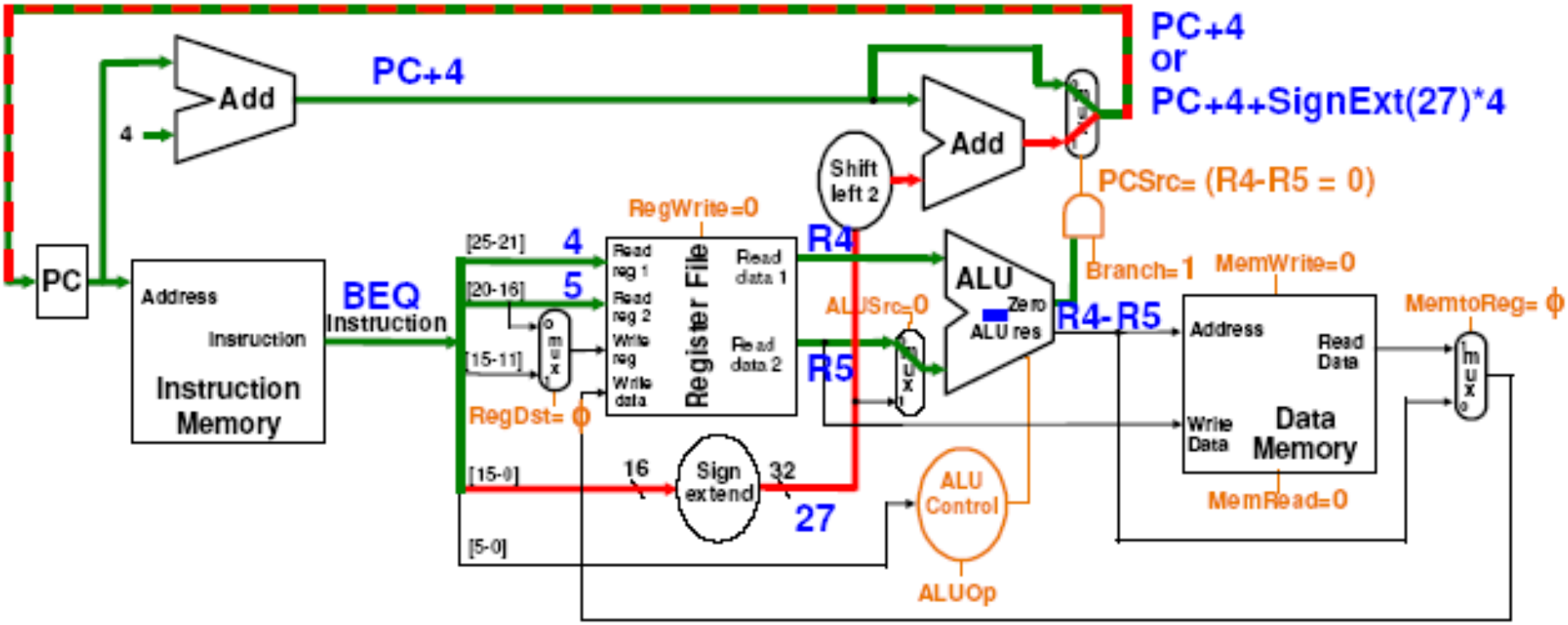


SW R1, (30)R2 ; Mem[R2+30] ← R1

31 op 26	rs 21	rt 16	immediate 0
SW	2	1	30

Imagen de Patterson, 1997

EJEMPLO: EJECUCIÓN DE UNA INSTRUCCIÓN DE "SALTO"



BEQ R4, R5, 27 ; if (R4-R5=0) then PC ← PC+4+SignExt(27)*4 ;
else PC ← PC+4

31 op 26	rs 21	rt 16	immediate 0
BEQ	4	5	27

Imagen de Patterson, 1997

SEÑALES DE CONTROL

func	10 0000	10 0010	**Don't Care**				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	**sub**	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
Branch	0	0	0	0	0	1	x
Jump	0	0	0	0	0	0	1
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	xxx

UNIDAD SEGMENTADA: ESTRUCTURA

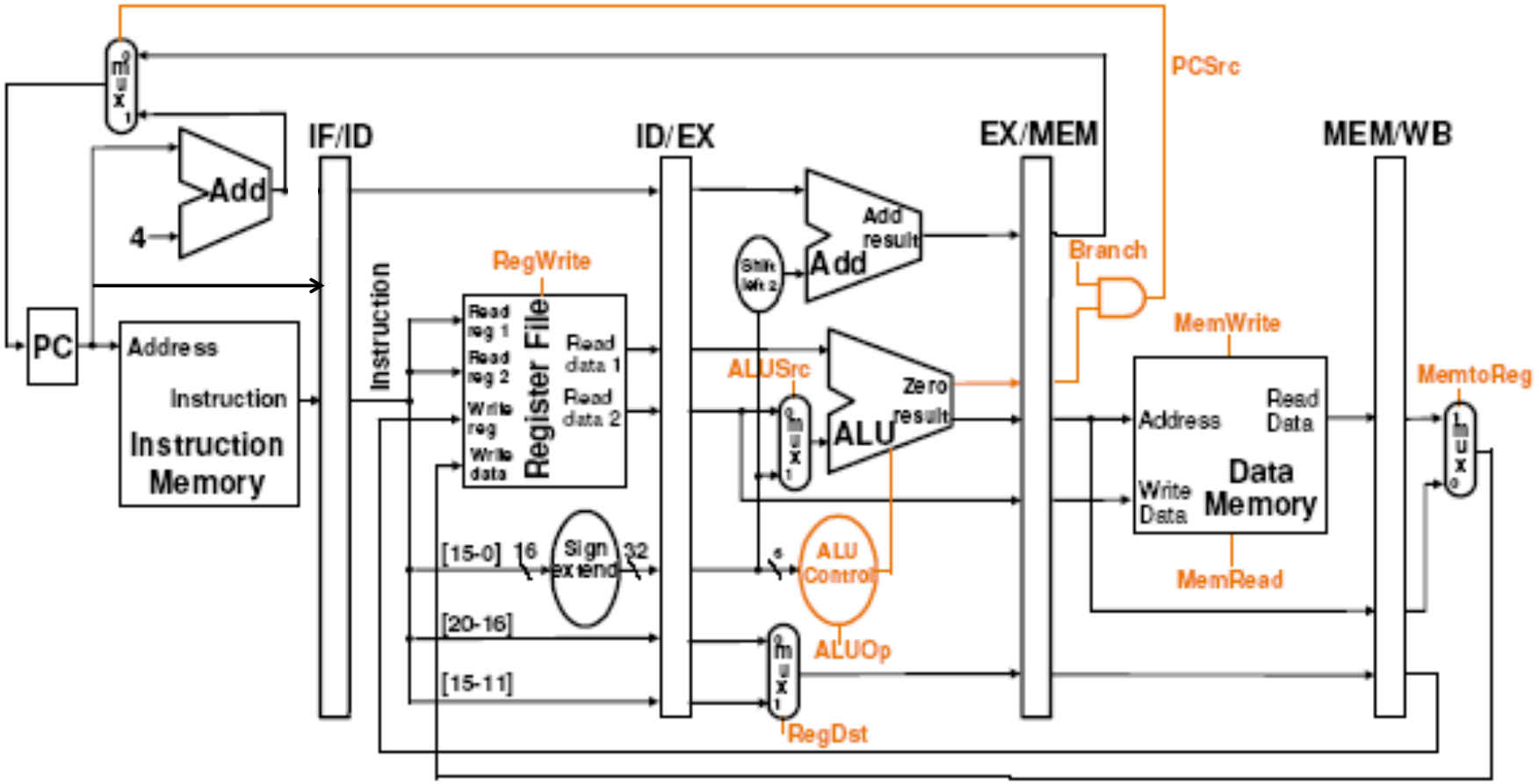


Imagen de Patterson, 1997

UNIDAD SEGMENTADA: ESTRUCTURA + CONTROL

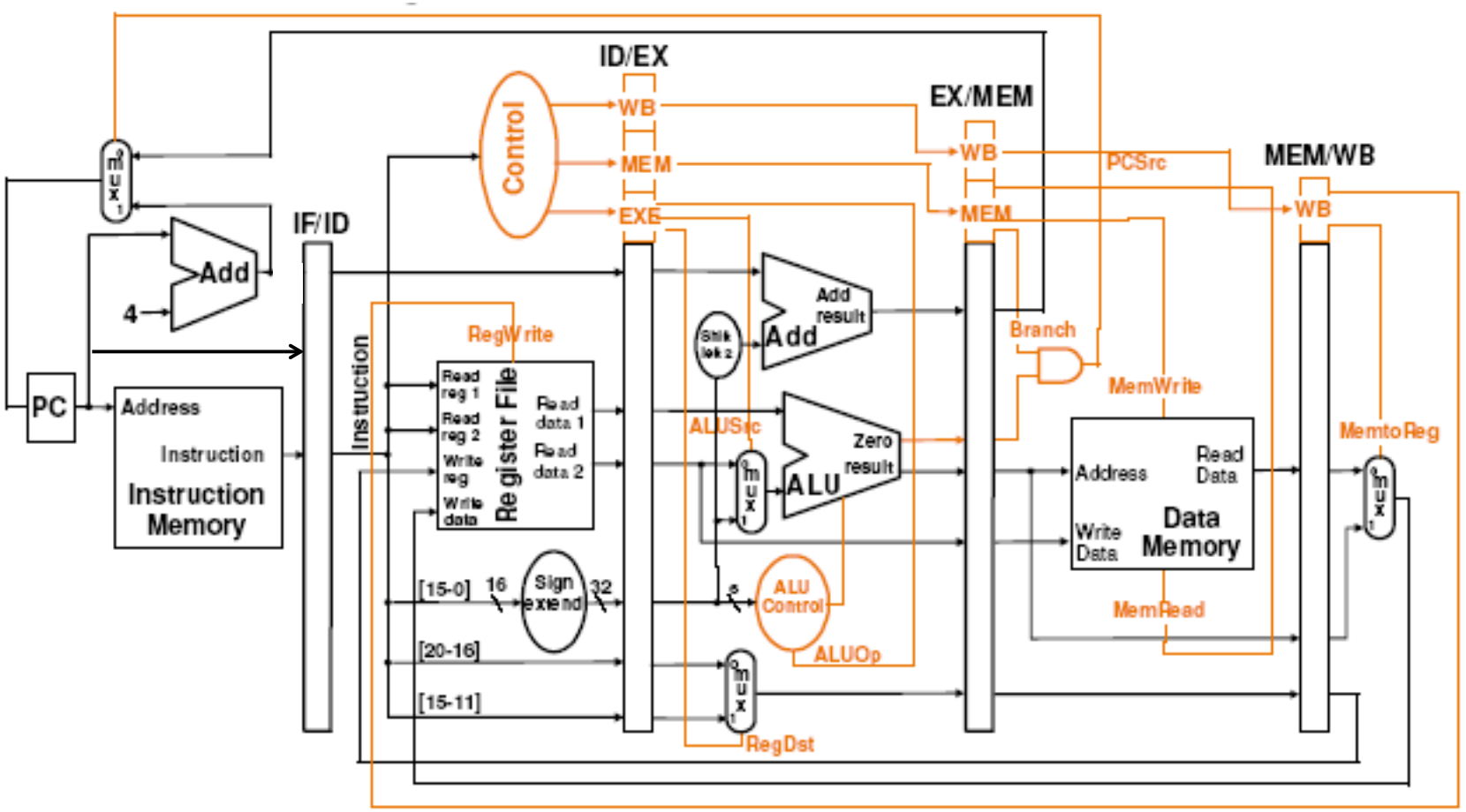
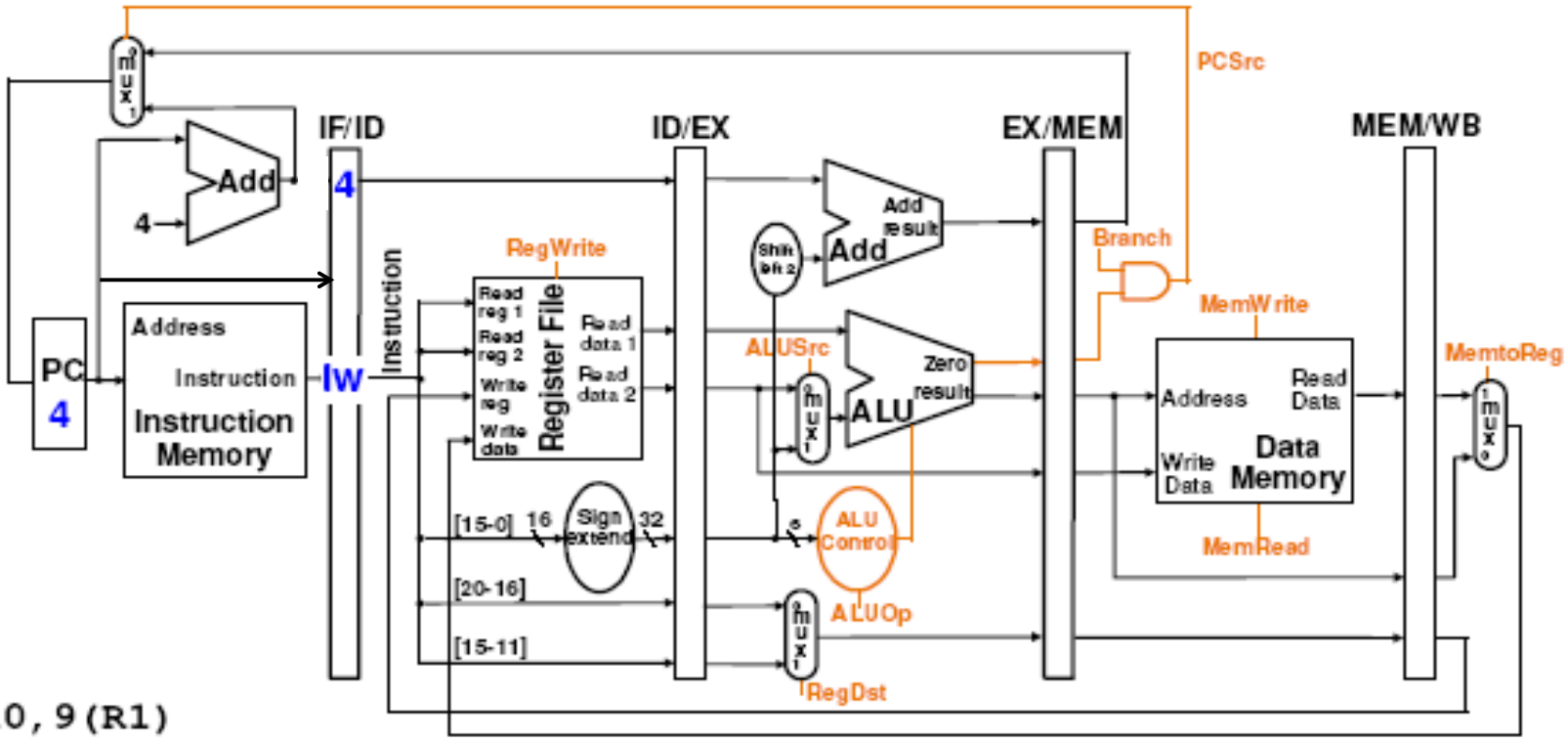


Imagen de Patterson, 1997

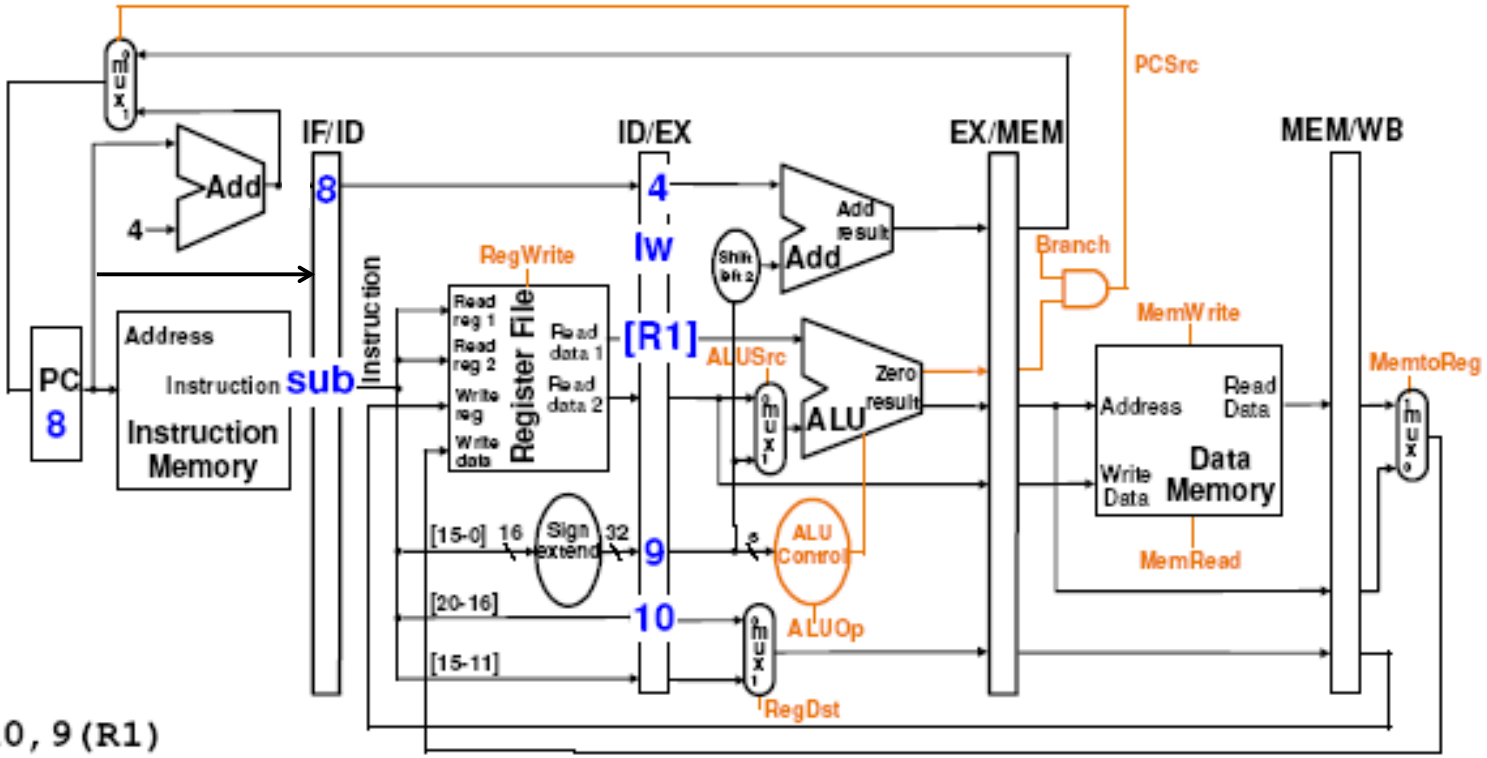
EJEMPLO: CICLO 1




```
lw   R10,9(R1)
sub  R11,R2, R3
and  R12,R4, R5
or   R13,R6, R7
```

Imagen de Patterson, 1997

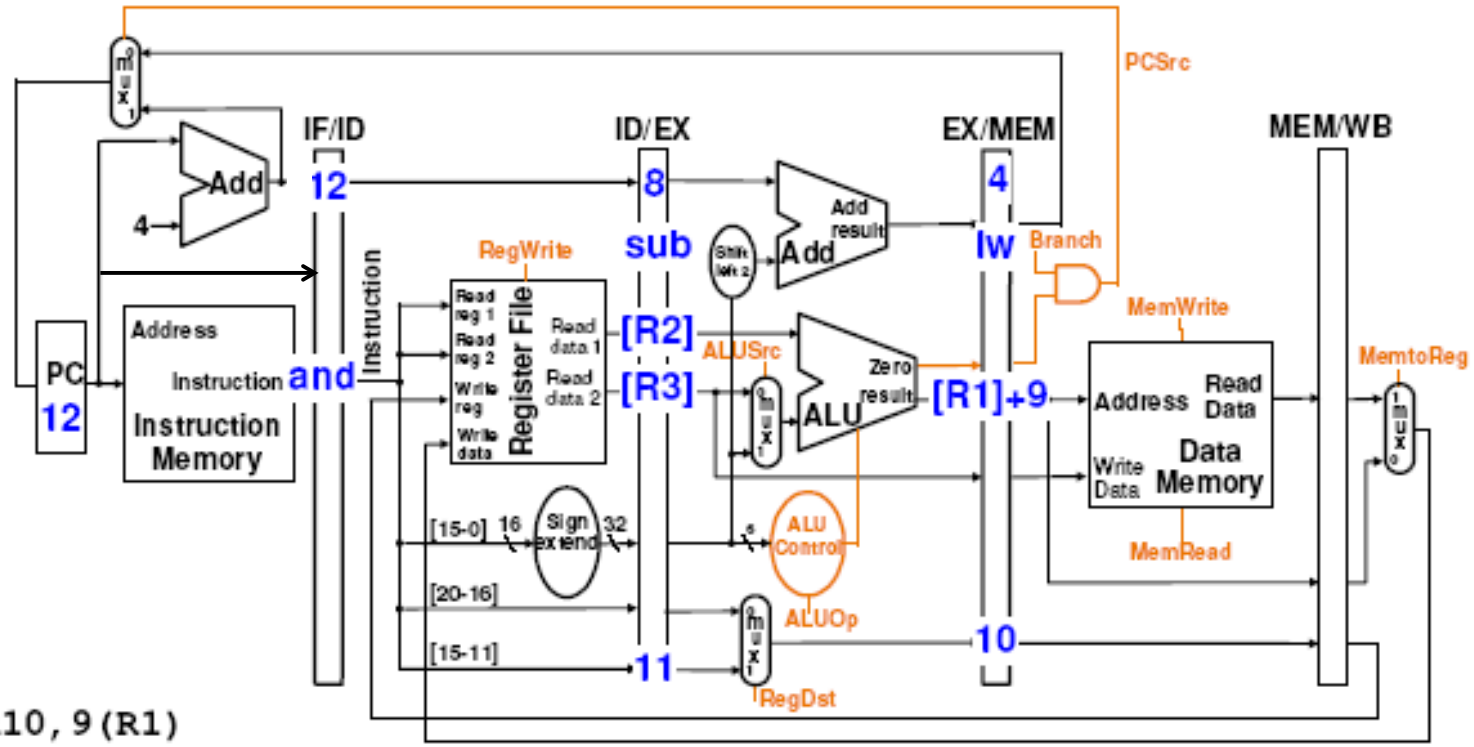
EJEMPLO: CICLO 2




```
lw  R10,9(R1)
sub R11,R2, R3
and R12,R4, R5
or  R13,R6, R7
```

Imagen de Patterson, 1997

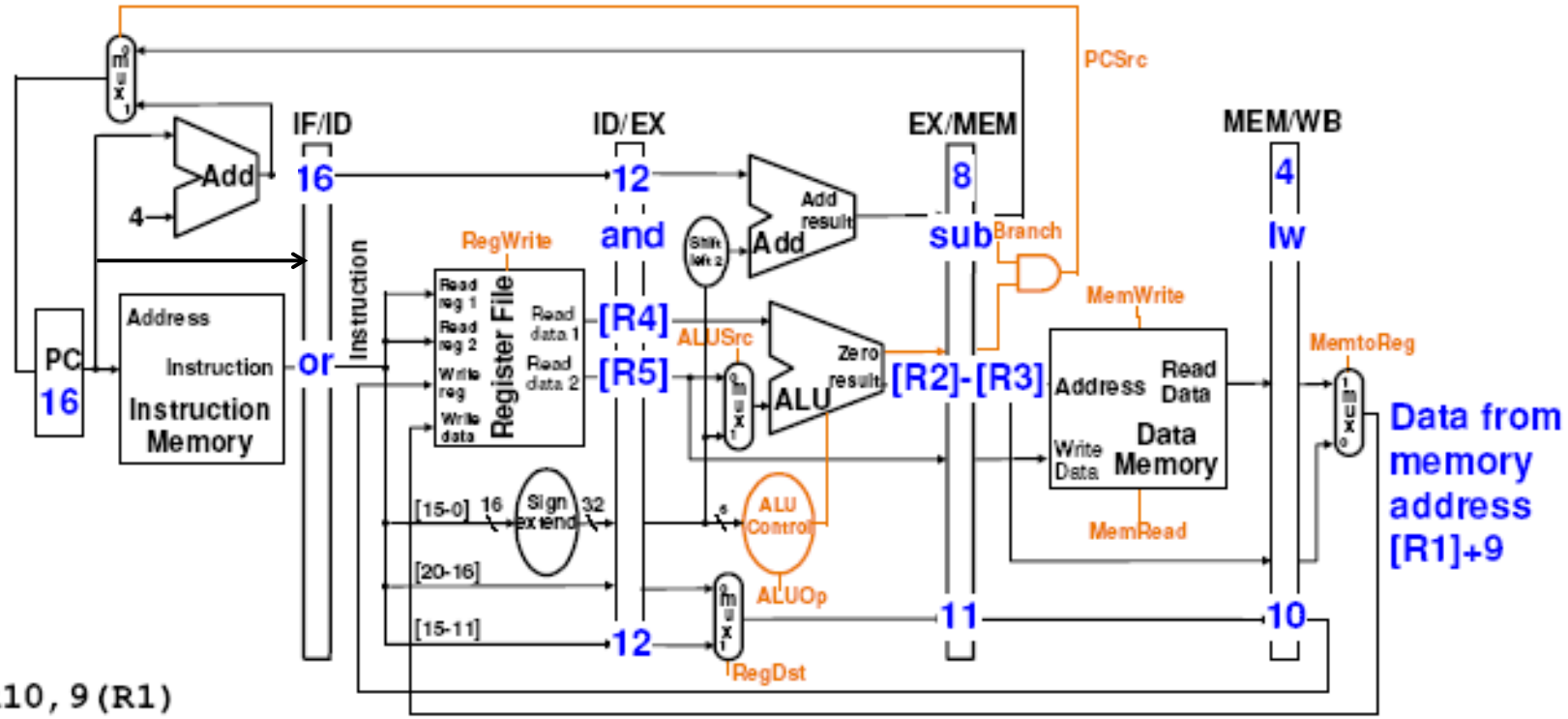
EJEMPLO: CICLO 3




```
lw   R10,9(R1)
sub  R11,R2,  R3
and  R12,R4,  R5
or   R13,R6,  R7
```

Imagen de Patterson, 1997

EJEMPLO: CICLO 4

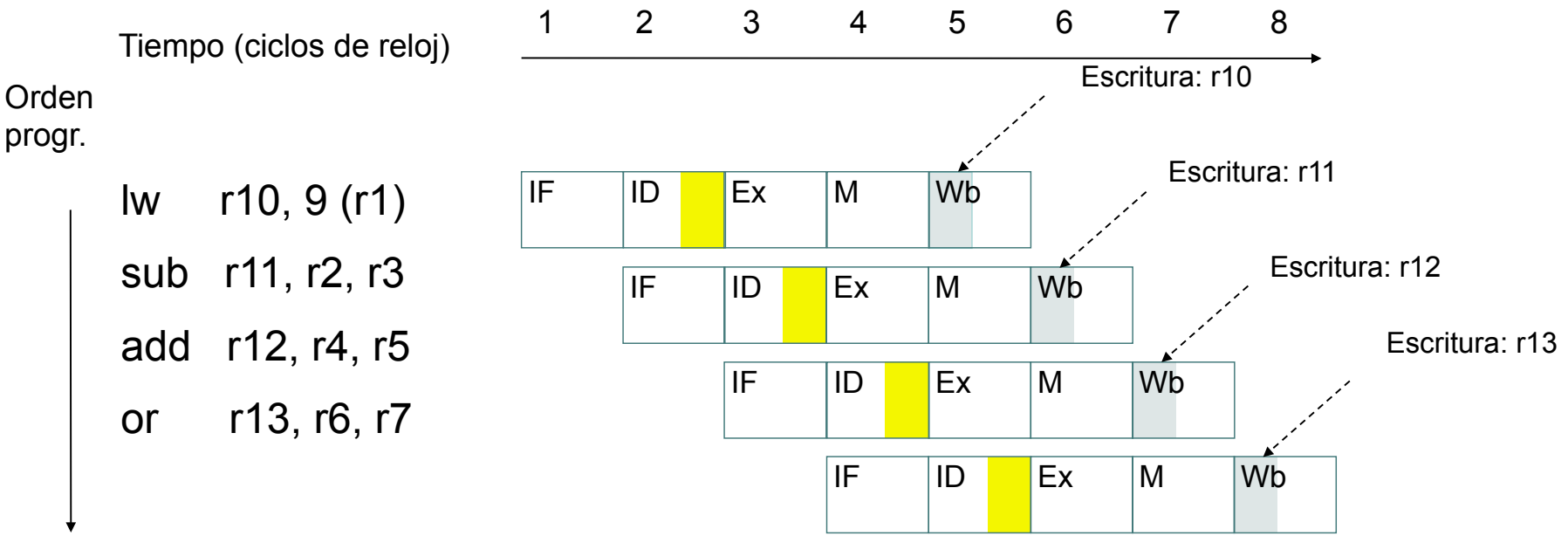



```
lw  R10,9(R1)
sub R11,R2, R3
and R12,R4, R5
or  R13,R6, R7
```

Imagen de Patterson, 1997

EJEMPLO

- Ejemplo de cronograma de ejecución.



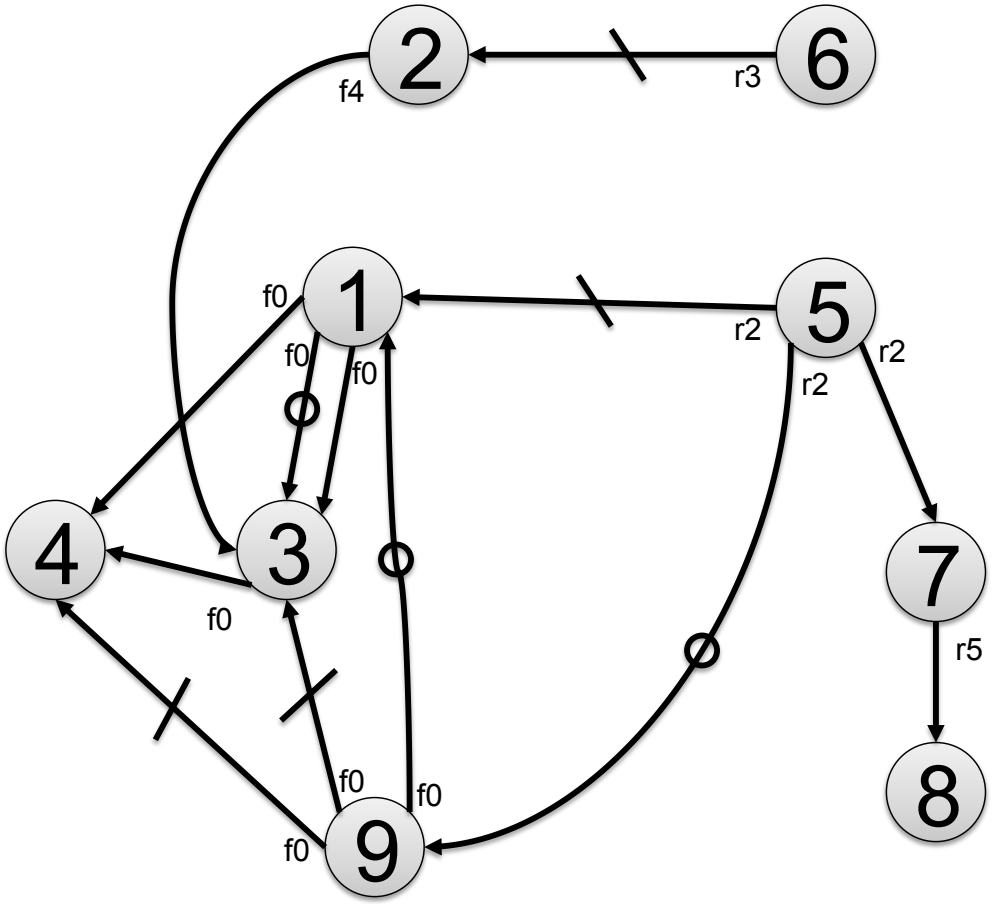
EJEMPLO

- Supongamos que las multiplicaciones en punto flotante necesitan 7 ciclos en la etapa de ejecución y que las sumas en punto flotante necesitan 4. Si estamos en un cauce segmentado ¿cuál sería el grafo de dependencias y cronograma de ejecución del siguiente código?

```
bucle:  1   LD F0, 0(R2)
        2   LD F4, 0(R3)
        3   MULTD F0,F0,F4
        4   ADDD F2,F0,F2
        5   ADDI R2,R2 #8
        6   ADDI R3,R3,#8
        7   SUB R5,R4,R2
        8   BNEZ R5, bucle
        9   LD F0,0(R2)
```

EJEMPLO

```
bucle:  1   LD F0, 0(R2)
        2   LD F4, 0(R3)
        3   MULTD F0,F0,F4
        4   ADDD F2,F0,F2
        5   ADDI R2,R2 #8
        6   ADDI R3,R3,#8
        7   SUB R5,R4,R2
        8   BNEZ R5, bucle
        9   LD F0,0(R2)
```



EJEMPLO

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
bucle	LD F0, 0(R2)	F	D	X	M	W																								
	LD F4, 0(R3)		F	D	X	M	W																							
	MULTD F0,F0,F4			F	-	-	D	X	X	X	X	X	X	X	M	W														
	ADDD F2,F0,F2						F	-	-	-	-	-	-	-	-	D	X	X	X	X	M	W								
	ADDI R2,R2 #8															F	D	X	M	W										
	ADDI R3,R3,#8																F	D	X	M	W									
	SUB R5,R4,R2																	F	-	D	X	M	W							
	BNEZ R5, bucle																			F	-	-	D	X	M	W				
	LD F0,0(R2)																						F	-	-	F	D	X	M	W

REFERENCIAS

[Wikipedia, 2014] Procesador MIPS, http://es.wikipedia.org/wiki/MIPS_(procesador)

[Patterson, 1997]David A. Patterson, John L. Hennessy (1997) Computer Organization & Design: The Hardware/Software Interface, Second Edition. Morgan Kaufmann.