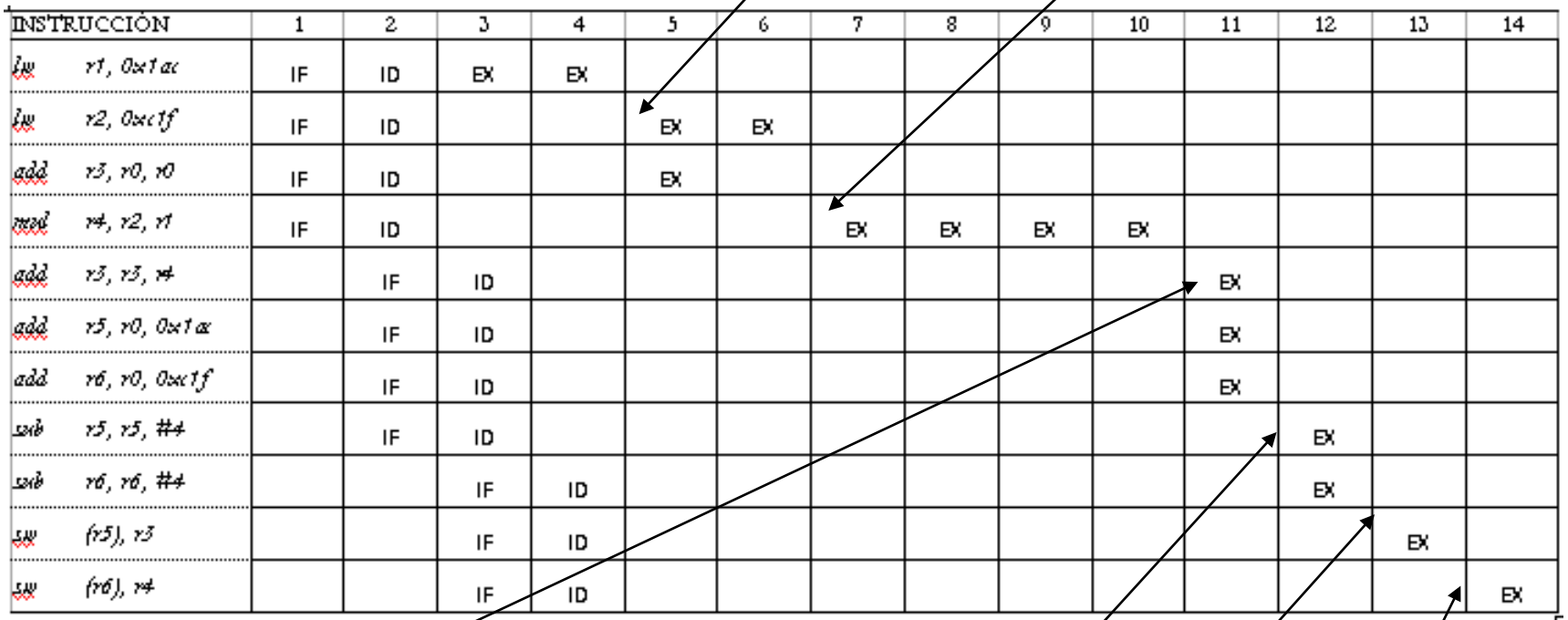
Tema 4. Problema 1

```
1.  lw r1,0x1ac ; r1 ← M(0x1ac)
2.  lw r2,0xc1f ; r2 ← M(0xc1f)
3.  add r3,r0,r0 ; r3 ← r0+r0
4.  mul r4,r2,r1 ; r4 ← r2*r1
5.  add r3,r3,r4 ; r3 ← r3+r4
6.  add r5,r0,0x1ac ; r5 ← r0+0x1ac
7.  add r6,r0,0xc1f ; r6 ← r0+0xc1f
8.  sub r5,r5,#4 ; r5 ← r5 - 4
9.  sub r6,r6,#4 ; r6 ← r6 - 4
10.  sw (r5),r3 ; M(r5) ← r3
11.  sw (r6),r4 ; M(r6) ← r4
```

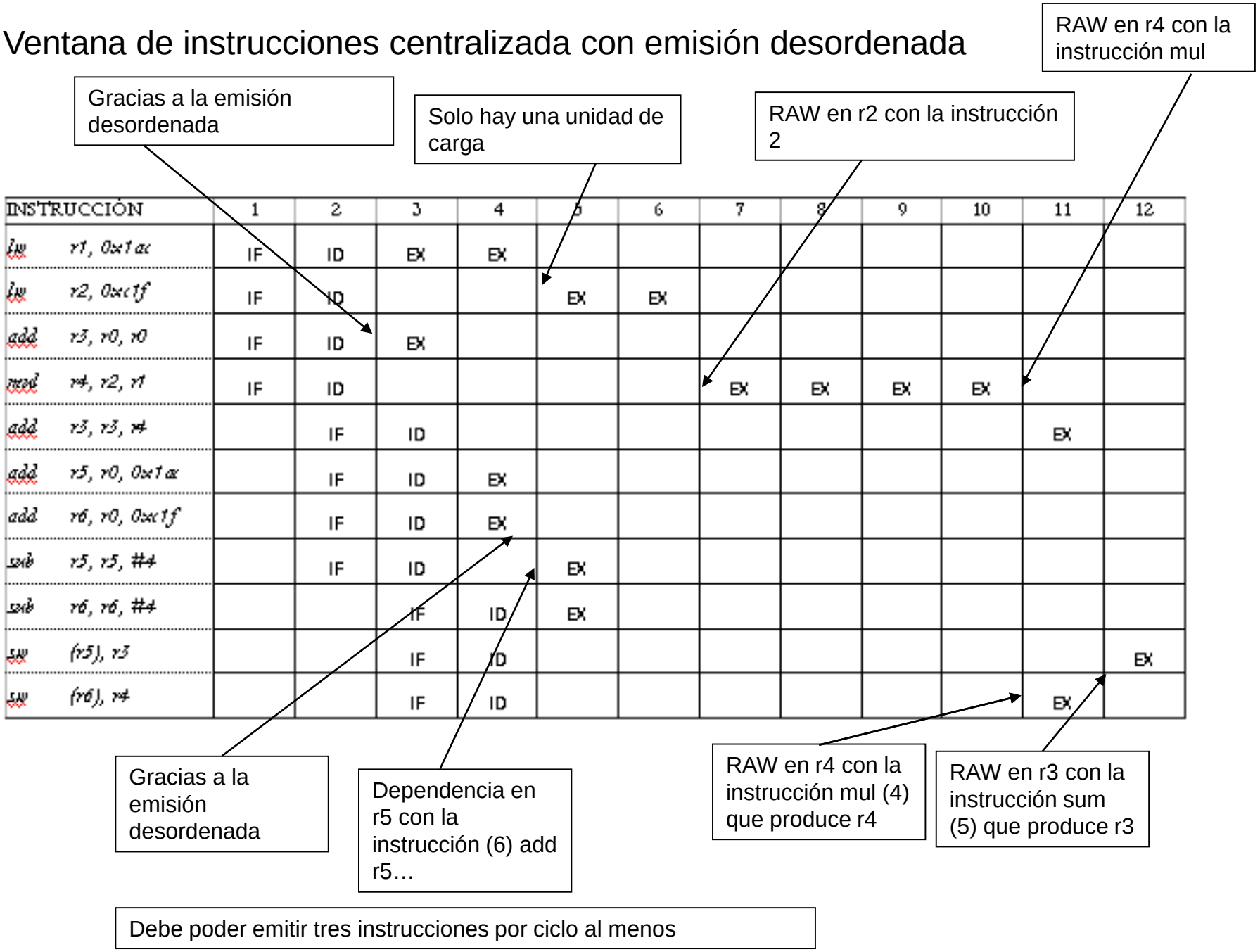
Ventana centralizada con emisión ordenada



INSTRUCCIÓN	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw r1, 0x1ac	IF	ID	EX	EX										
lw r2, 0xc1f	IF	ID			EX	EX								
add r3, r0, r0	IF	ID			EX									
mul r4, r2, r1	IF	ID					EX	EX	EX	EX				
add r3, r3, r4		IF	ID								EX			
add r5, r0, 0x1ac		IF	ID								EX			
add r6, r0, 0xc1f		IF	ID								EX			
sub r5, r5, #4		IF	ID									EX		
sub r6, r6, #4			IF	ID								EX		
sw (r5), r3			IF	ID									EX	
sw (r6), r4			IF	ID										EX

Comprobar que no se emiten más instrucciones de las posibles y que es ordenada

Ventana de instrucciones centralizada con emisión desordenada

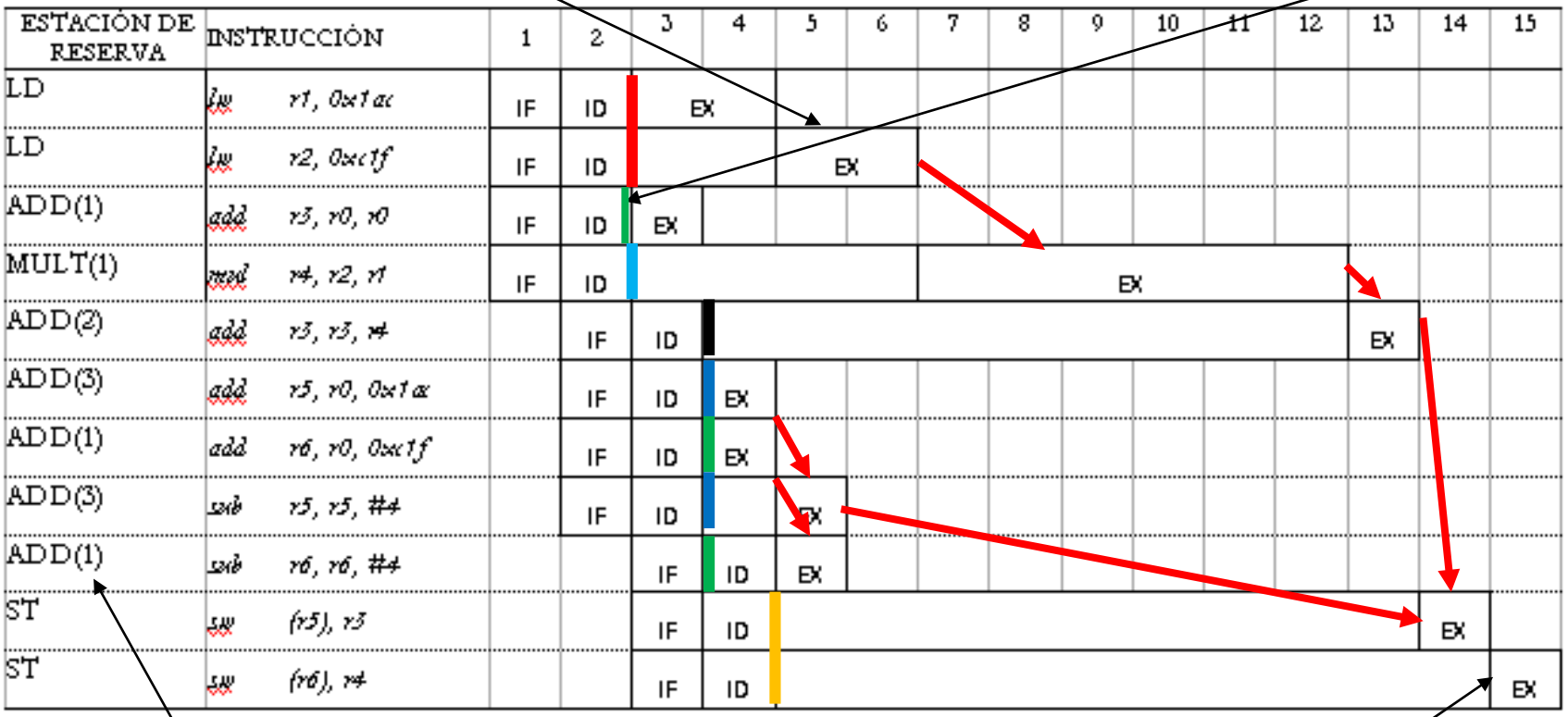


INSTRUCCIÓN	1	2	3	4	5	6	7	8	9	10	11	12
lw r1, 0x1ac	IF	ID	EX	EX								
lw r2, 0xc1f	IF	ID			EX	EX						
add r3, r0, r0	IF	ID	EX									
mul r4, r2, r1	IF	ID					EX	EX	EX	EX		
add r3, r3, r4		IF	ID								EX	
add r5, r0, 0x1ac		IF	ID	EX								
add r6, r0, 0xc1f		IF	ID	EX								
sub r5, r5, #4		IF	ID		EX							
sub r6, r6, #4			IF	ID	EX							
sw (r5), r3			IF	ID								EX
sw (r6), r4			IF	ID							EX	

Estación de reserva con tres líneas para cada unidad funcional y envío ordenado

Una sola unidad de Load

Aunque el envío sea ordenado la ejecución puede ser desordenada porque hay varias estaciones de reserva



Entre paréntesis, la estación de reserva de la unidad funcional en la que se ejecuta la instrucción
El orden en el que se asignan depende de la política que utilice la unidad de ID (también denominada ID/ISS)

Solo hay una unidad de almacenamiento y el envío es ordenado

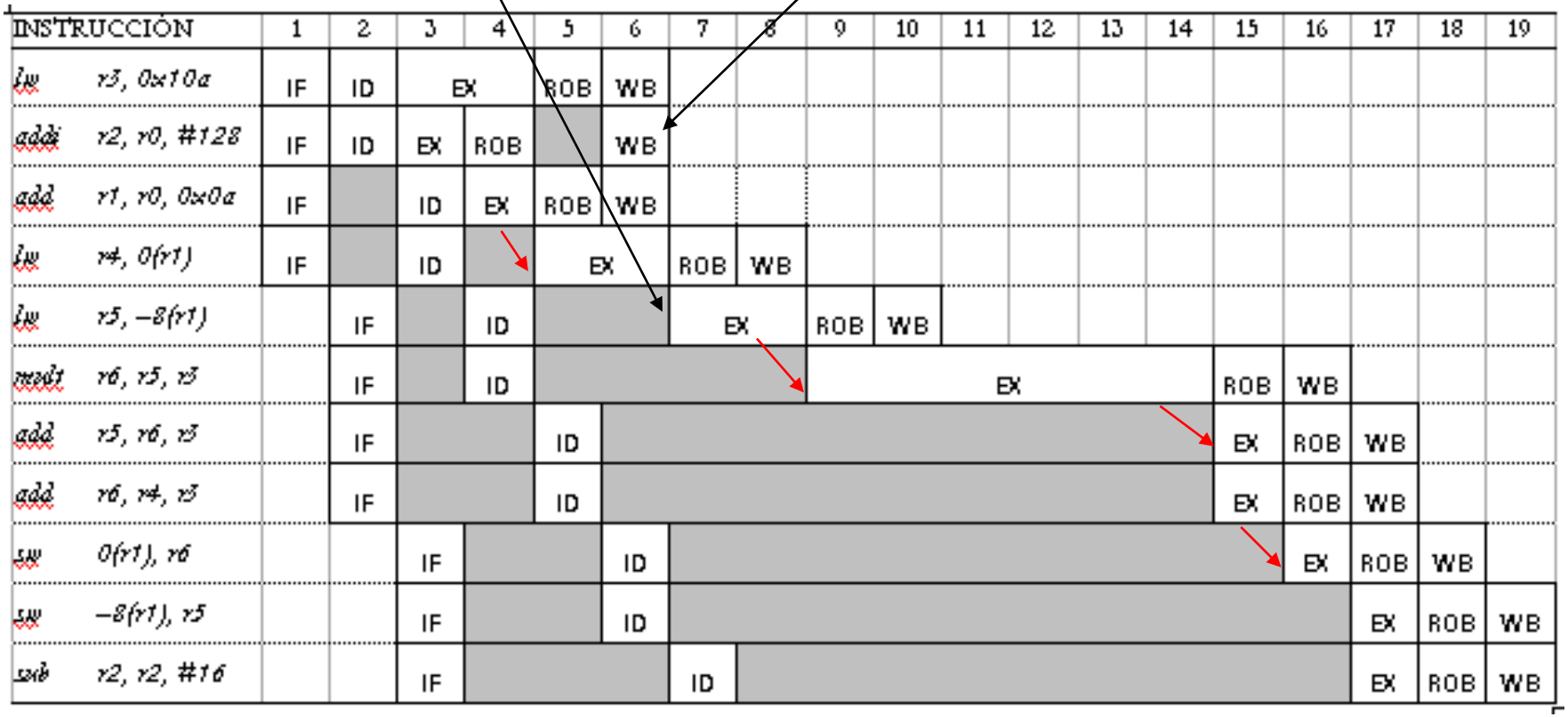
Problema 2

```
1.    lw     r3,0x10a      ; r3 ← M(0x10a)
2.    addi   r2,r0,#128    ; r2 ← r0+128
3.    add    r1,r0,0x0a    ; r1 ← r0+0x0a
4.    lw     r4,0(r1)      ; r4 ← M(r1)
5.    lw     r5,-8(r1)     ; r5 ← M(r1-8)
6.    mult   r6,r5,r3      ; r6 ← r5*r3
7     add    r5,r6,r3      ; r5 ← r6+r3
8     add    r6,r4,r3      ; r6 ← r4+r3
9     sw     0(r1),r6      ; M(r1) ← r5
10.   sw     -8(r1),r5     ; M(r1-8) ← r5
11.   sub    r2,r2,#16     ; r2 ← r2-16
```

Emisión Ordenada

Una sola unidad de carga

Se tiene que mantener el orden al retirar las instrucciones

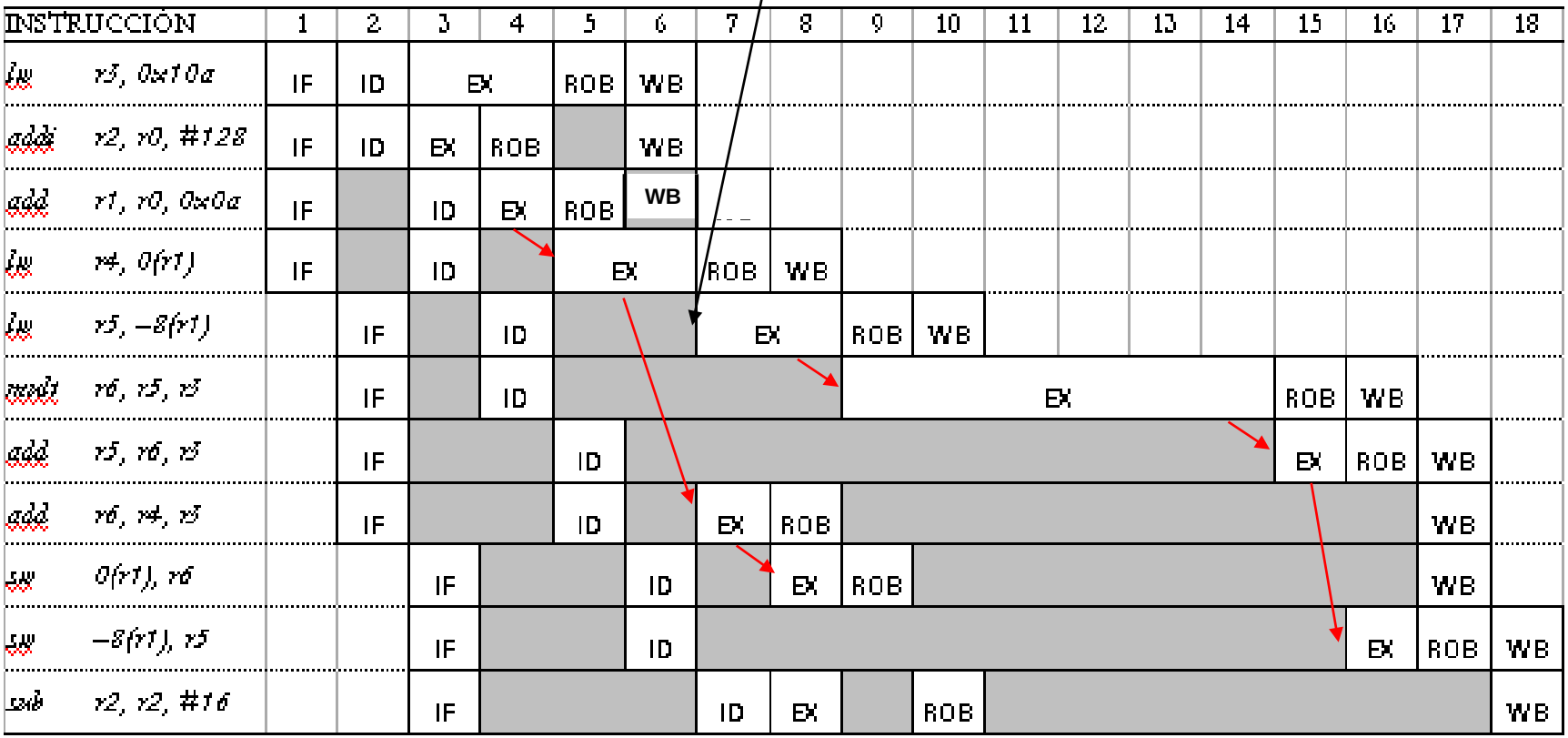


Tiene que comprobarse que **no se decodifican, emiten, ni escriben en el ROB más de dos instrucciones por ciclo, ni se retiran más de tres instrucciones por ciclo**.

Se podría prescindir de la etapa ROB (la etapa EX finaliza con la escritura en el ROB además de en la ventana de instrucciones) y en las instrucciones SW se podría prescindir de la etapa ROB

Emisión Desordenada

Una sola unidad de carga

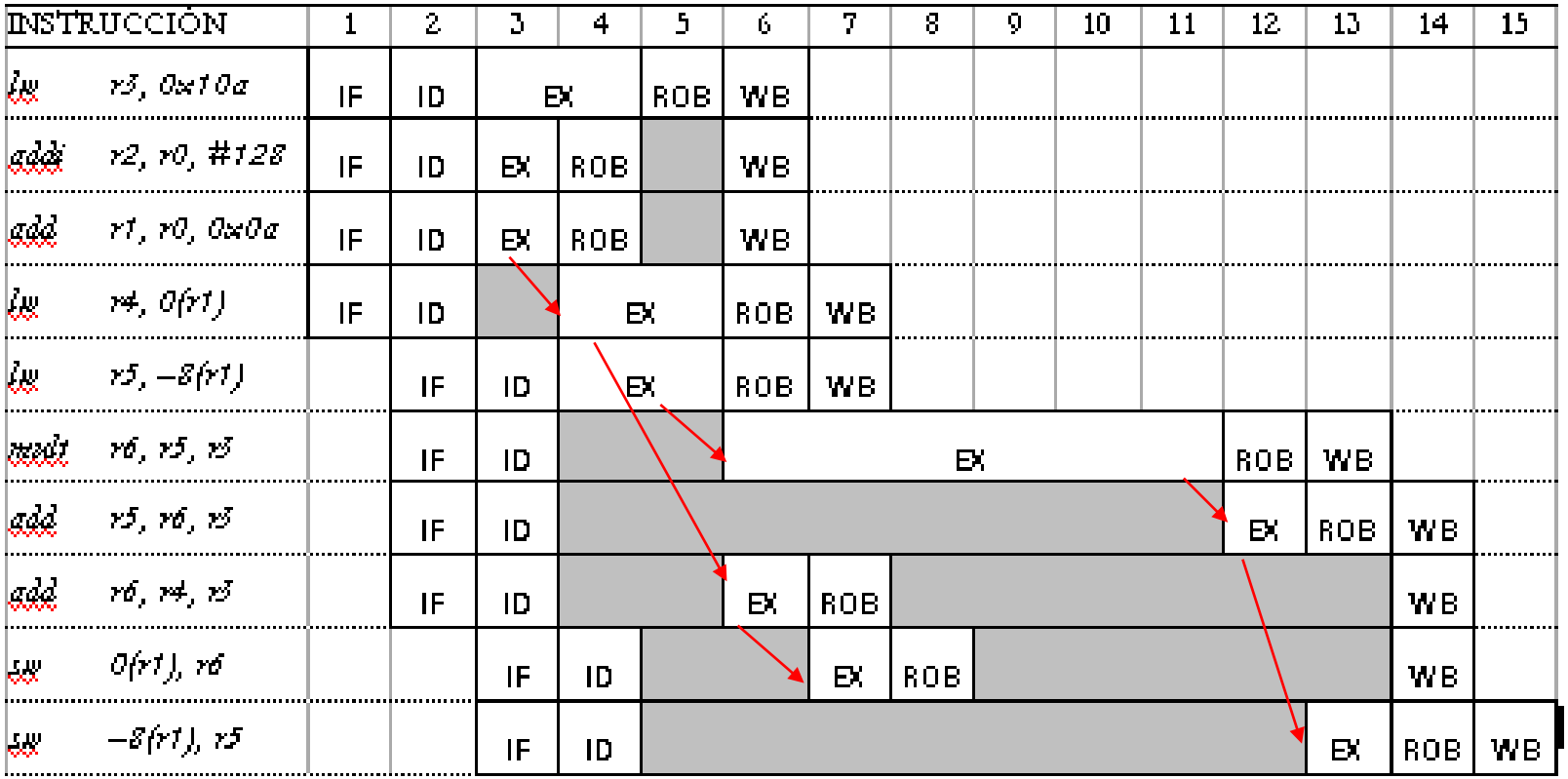


INSTRUCCIÓN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
lw r3, 0x10a	IF	ID	EX		ROB	WB												
addi r2, r0, #128	IF	ID	EX	ROB		WB												
add r1, r0, 0x0a	IF		ID	EX	ROB	WB												
lw r4, 0(r1)	IF		ID		EX		ROB	WB										
lw r5, −8(r1)		IF		ID			EX		ROB	WB								
mult r6, r5, r3		IF		ID					EX						ROB	WB		
add r5, r6, r3		IF			ID										EX	ROB	WB	
add r6, r4, r3		IF			ID		EX	ROB									WB	
sw 0(r1), r6			IF			ID		EX	ROB								WB	
sw −8(r1), r5			IF			ID										EX	ROB	WB
sub r2, r2, #16			IF				ID	EX		ROB								WB

También en este caso hay que tener en cuenta que no se pueden decodificar, emitir, ni escribir en el ROB, más de dos instrucciones por ciclo (obsérvese que la instrucción sub r2,r2,#16 debe esperar un ciclo para su etapa ROB por esta razón), ni se pueden retirar más de tres instrucciones por ciclo.

Se podría prescindir de ROB, y de WB en las instrucciones de almacenamiento SW

3. a)



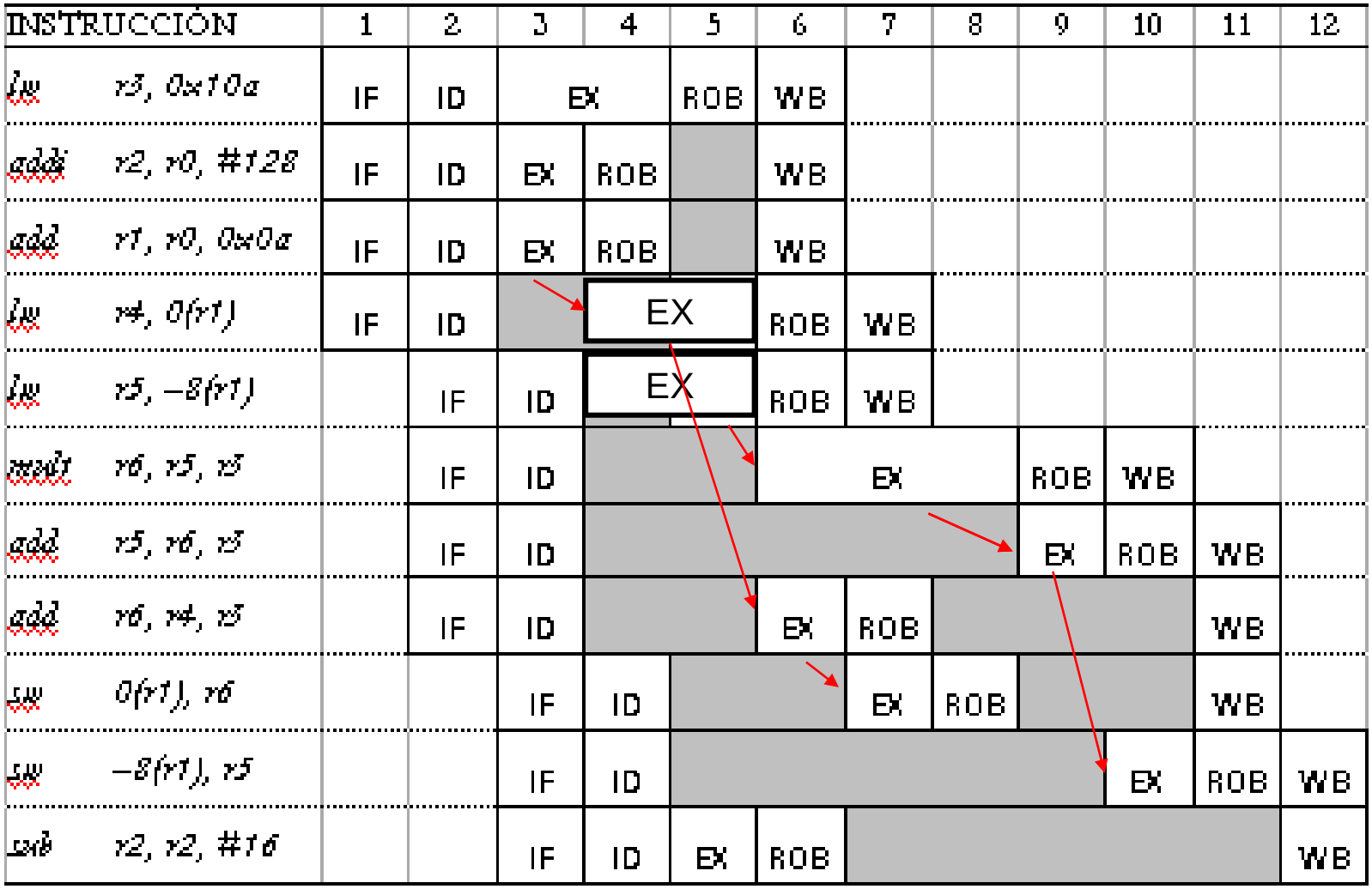
Se decodifican el mismo número de instrucciones que se captan.

No existen limitaciones para el número de instrucciones por ciclo que se emiten, escriben el ROB, y se retiran.

Están disponibles todas las unidades funcionales que se necesiten para que no haya colisiones (riesgos estructurales).

Se podría prescindir de WB en las instrucciones de almacenamiento SW, y también de ROB si se supone que la escritura finaliza con la escritua en el ROB al mismo tiempo que en la ventana de instrucciones

Se reduce el tiempo del multiplicador



$$T(n) = 12 = \mathrm{TLI} + (n-1) \times \mathrm{CPI} = 6 + (11-1) \times \mathrm{CPI}$$

CPI=0.6

Ejercicio 8

En un programa, una instrucción de salto condicional (a una dirección de salto anterior) dada tiene el siguiente comportamiento en una ejecución de dicho programa:

SSNNN SSNSN SNSSS SSN

donde S indica que se produce el salto y N que no. Indique la penalización efectiva que se introduce si se utiliza:

a) Predicción fija (siempre se considera que se no se va a producir el salto)
b) Predicción estática (si el desplazamiento es negativo se toma y si es positivo no)

c) Predicción dinámica con dos bits, inicialmente en el estado (11).
d) Predicción dinámica con tres bits, inicialmente en el estado (111).

Nota: La penalización por saltos incorrectamente predichos es de 5 ciclos y para los saltos correctamente predichos es 0 ciclos.

Ejercicio 8

Solución:

En el caso de usar predicción fija, se produciría un fallo del predictor cada vez que se tome el salto, tal y como muestra la Tabla siguiente. Por tanto, la penalización total sería de:

$$P_{\text{fijo}} = F_{\text{fijo}} \times P = 11 \times 5 = 55 \text{ ciclos}$$

PREDICCIÓN FIJA	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*	*N*
EJECUCIÓN	*S*	*S*	*N*	*N*	*N*	*S*	*S*	*N*	*S*	*N*	*S*	*N*	*S*	*S*	*S*	*S*	*S*	*N*
PENALIZACIÓN	*P*	*P*				*P*	*P*		*P*		*P*		*P*	*P*	*P*	*P*	*P*	

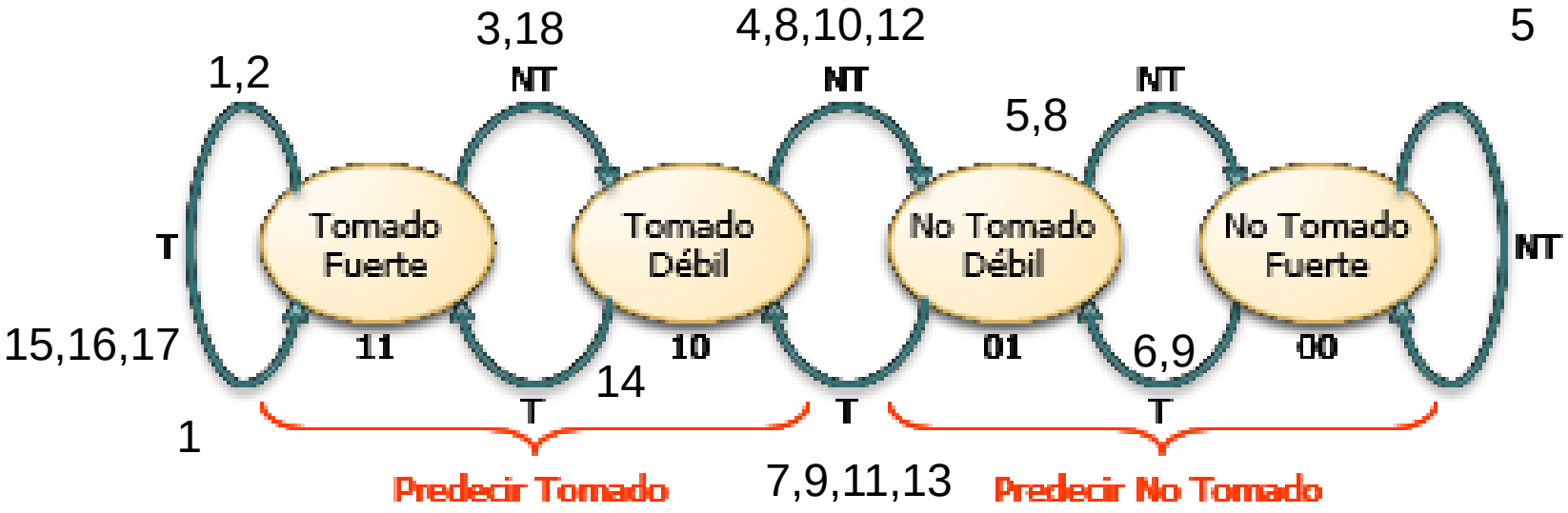
$$P_{\text{estático}} = N_{\text{estático}} \times P = 7 \times 5 = 35 \text{ ciclos}$$

PREDICCIÓN ESTÁTICA	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*
EJECUCIÓN	*S*	*S*	*N*	*N*	*N*	*S*	*S*	*N*	*S*	*N*	*S*	*N*	*S*	*S*	*S*	*S*	*S*	*N*
PENALIZACIÓN			*P*	*P*	*P*			*P*		*P*		*P*						*P*

Ejercicio 8

$$P_{2\text{ bits}} = F_{2\text{ bits}} \times P = 11 \times 5 = 55 \text{ ciclos}$$

ESTADO	11	11	11	10	01	00	01	10	01	10	01	10	01	10	11	11	11	11
PREDICCIÓN DINÁMICA (2 BITS)	S	S	S	S	N	N	N	S	N	S	N	S	N	S	S	S	S	S
EJECUCIÓN	S	S	N	N	N	S	S	N	S	N	S	N	S	S	S	S	S	N
PENALIZACIÓN			P	P		P	P	P	P	P	P	P	P					P



Ejercicio 8

$$P_{3\,\text{bits}} = F_{3\,\text{bits}} \times P = 10 \times 5 = 50\ \text{ciclos}$$

ESTADO	*111*	*111*	*111*	*011*	*001*	*000*	*100*	*110*	*011*	*101*	*010*	*101*	*010*	*101*	*110*	*111*	*111*	*111*
PREDICCIÓN DINÁMICA (3 BITS)	*S*	*S*	*S*	*S*	*N*	*N*	*N*	*S*	*S*	*S*	*N*	*S*	*N*	*S*	*S*	*S*	*S*	*S*
EJECUCIÓN	*S*	*S*	*N*	*N*	*N*	*S*	*S*	*N*	*S*	*N*	*S*	*N*	*S*	*S*	*S*	*S*	*S*	*N*
PENALIZACIÓN			*P*	*P*		*P*	*P*	*P*		*P*	*P*	*P*	*P*					*P*

Ejercicio 10

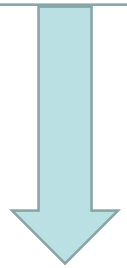
if (A>B) then { X=1;}
else {if (C<D) then X=2; else X=3;}

```
                lw      r1, a       ; r1 = A
                lw      r2, b       ; r2 = B
        p1, p2  cmp.gt  r1,r2       ; Si A > B  p1 = 1 y p2 = 0 (si no, p1 = 0 y p2 = 1)
(p1)            addi    r5, r0, #1
        p3      cmp.ne  r0, r0      ; Inicializamos p3 a 0
        p4      cmp.ne  r0, r0      ; Inicializamos p4 a 0
(p2)            lw      r3, c       ; r3 = C
(p2)            lw      r4, d       ; r4 = D
(p2)    p3, p4  cmp.lt  r3, r4      ; Sólo si p2 = 1 p3 o p4 pueden ser 1
(p3)            addi    r5, r0, #2  ; Se ejecuta si p3 = 1 (y p2 = 1)
(p4)            addi    r5, r0, #3  ; Se ejecuta si p4 = 1 (y p2 = 1)
                sw      (x),r5      ; Almacenamos el resultado
```

TEMA 4

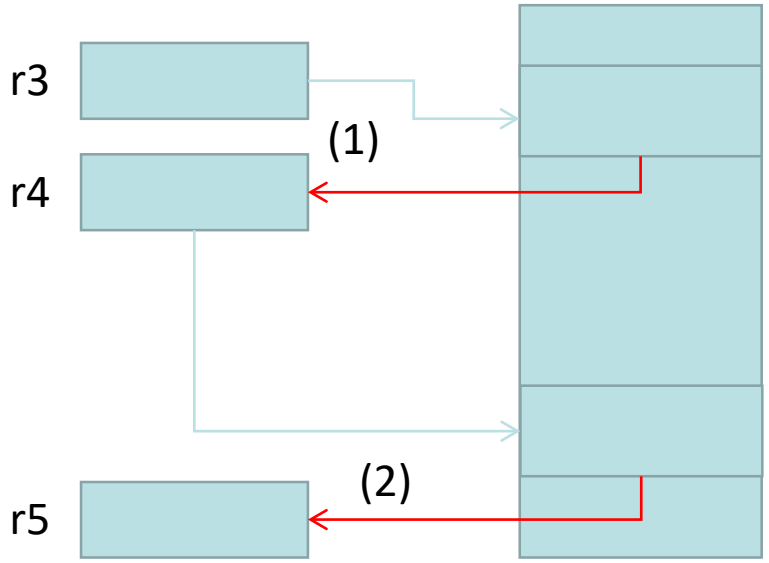
Ejercicio 9

#	OP1	OP2
1	lw r1, x(r2)	add r10, r11, r12
2		add r13, r10, r14
3	beqz r3, direc	
4	lw r4, 0(r3)	
5	lw r5, 0(r4)	



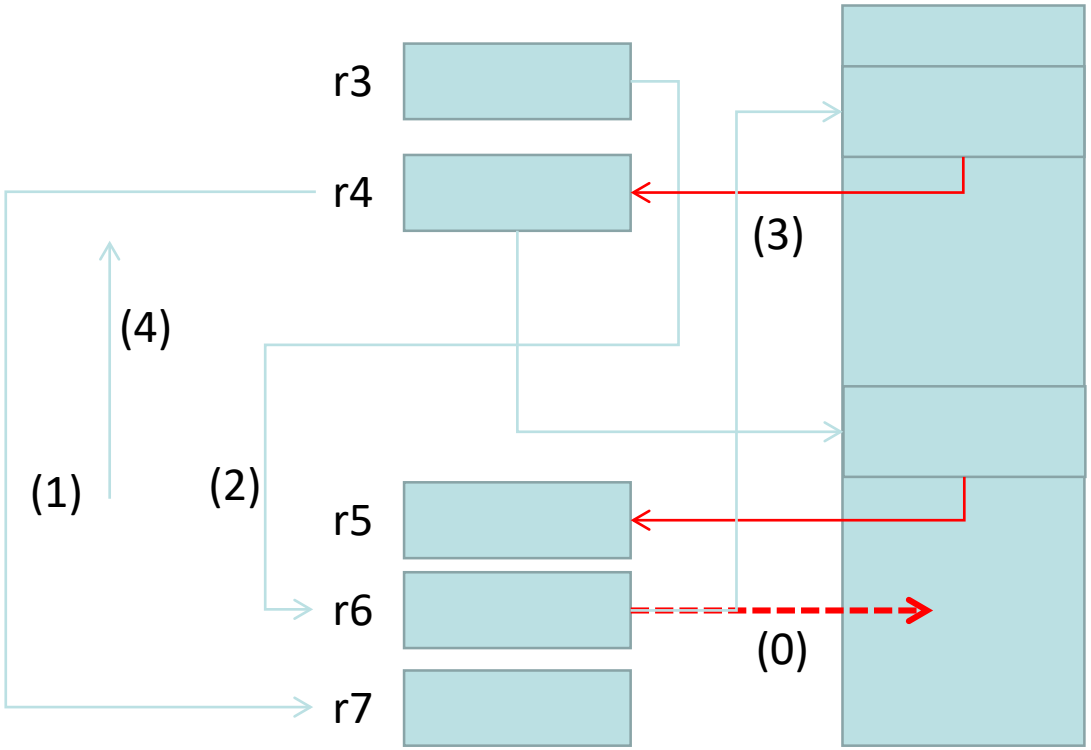
```
lw      r1, x(r2)   ; (1)
nop                 ; (2)
beqz    r3, direc   ; (3)
lw      r4, 0(r3)   ; (4)
lw      r5, 0(r4)   ; (5)
```

```
      lw      r1, x(r2)   ; (1)
      nop                 ; (2)
      beqz    r3, direc   ; (3)
(1)   lw      r4, 0(r3)   ; (4)
(2)   lw      r5, 0(r4)   ; (5)
```



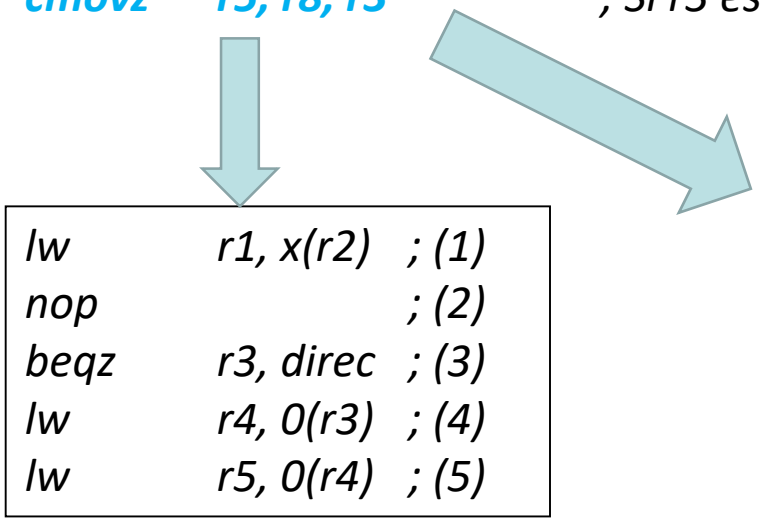

```
addi    r6, r0, #1000     ; Fijamos r6 a una dirección segura
lw      r1, x(r2)
mov     r7, r4            ; Guardamos el contenido original de r4 en r7
cmovnz  r6, r3, r3        ; Movemos r3 a r6 si r3 es distinto de cero
lw      r4, 0(r6)         ; Carga especulativa
cmovz   r4, r7, r3        ; Si r3 es 0, hay que hacer que r4 recupere su valor
beqz    r3, direc
lw      r5, 0(r4)   ; Si r3 no es cero, hay que cargar r5
```

```
(0)  addi   r6, r0, #1000   ; Fijamos r6 a una dirección segura
     lw     r1, x(r2)
(1)  mov    r7, r4          ; Guardamos el contenido original de r4 en r7
(2)  cmovnz r6, r3, r3      ; Movemos r3 a r6 si r3 es distinto de cero
(3)  lw     r4, 0(r6)       ; Carga especulativa
(4)  cmovz  r4, r7, r3      ; Si r3 es 0, hay que hacer que r4 recupere su valor
     beqz   r3, direc
     lw     r5, 0(r4)       ; Si r3 no es cero, hay que cargar r5
```



Si r6 no se carga con r3 en **cmovnz** como luego se hace lw hay que asegurarse que esa dirección no genera excepción

```
addi    r6, r0, #1000   ; Fijamos r6 a una dirección segura
lw      r1, x(r2)
mov     r7, r4          ; Guardamos el contenido original de r4 en r7
mov     r8, r5          ; Guardamos r5 en otro registro temporal r8
cmovnz  r6, r3, r3      ; Movemos r3 a r6 si r3 es distinto de cero
lw      r4, 0(r6)       ; Carga especulativa
lw      r5, 0(r4)       ; Esta carga también es especulativa
cmovz   r4, r7, r3      ; Si r3 es 0, hay que hacer que r4 recupere su valor
cmovz   r5, r8, r3      ; Si r3 es 0 hay que hacer que r5 recupere su valor
```



```
lw      r1, x(r2)   ; (1)
nop                 ; (2)
beqz    r3, direc   ; (3)
lw      r4, 0(r3)   ; (4)
lw      r5, 0(r4)   ; (5)
```

#	OP1	OP2
1	addi r6,r0,#1000	add r10, r11, r12
2	lw r1,x(r2)	add r13, r10, r14
3	cmovnz r6,r3,r3	mov r7,r4
4	lw r4, 0(r3)	mov r8,r5
5	lw r5, 0(r4)	
6	cmovz r4,r7,r3	cmovz r5,r8,r3