# Arquitectura de Computadores

# SEGMENTACIÓN DE CAUCE
# RIESGOS RAW DE DATOS

/home/vec/Dropbox/Arquitectura/Segmentacion/segmentacion_2_v3.odp

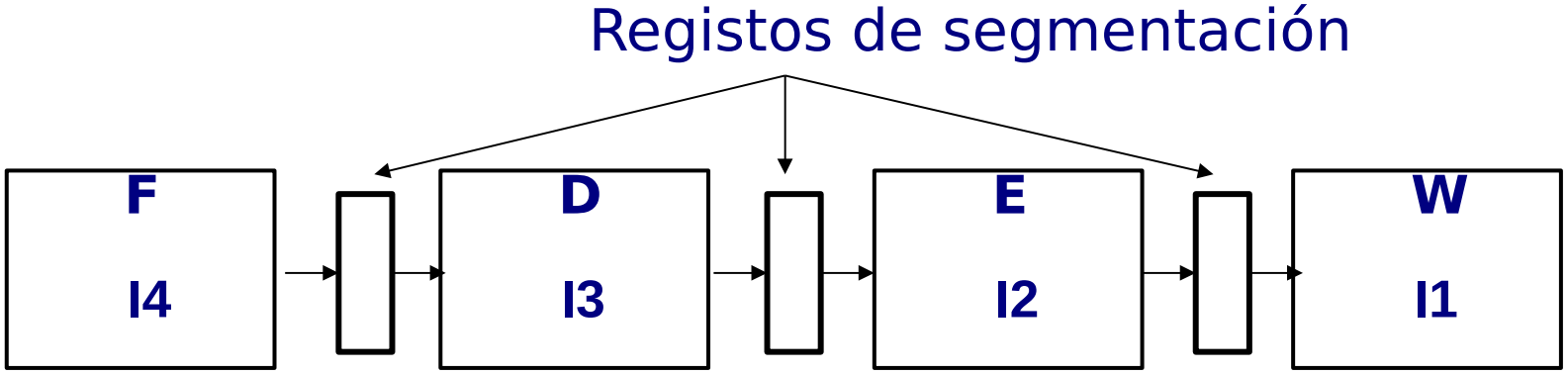
# Riesgos en Segmentación:

- El diseño de procesadores segmentados tiene gran dependencia del repertorio de instrucciones (RISC / CISC)
- Riesgos de Datos
  - Debidos a dependencias que existen entre los datos que manipulan las instrucciones
- Riesgos de Instrucciones o de Control
  - Debidos a cambios en el flujo de los programas
- Riesgos Estructurales
  - Debidos a conflictos de utilización de los elementos de la estructura de la máquina

# Etapas del cauce seg. profundidad 4

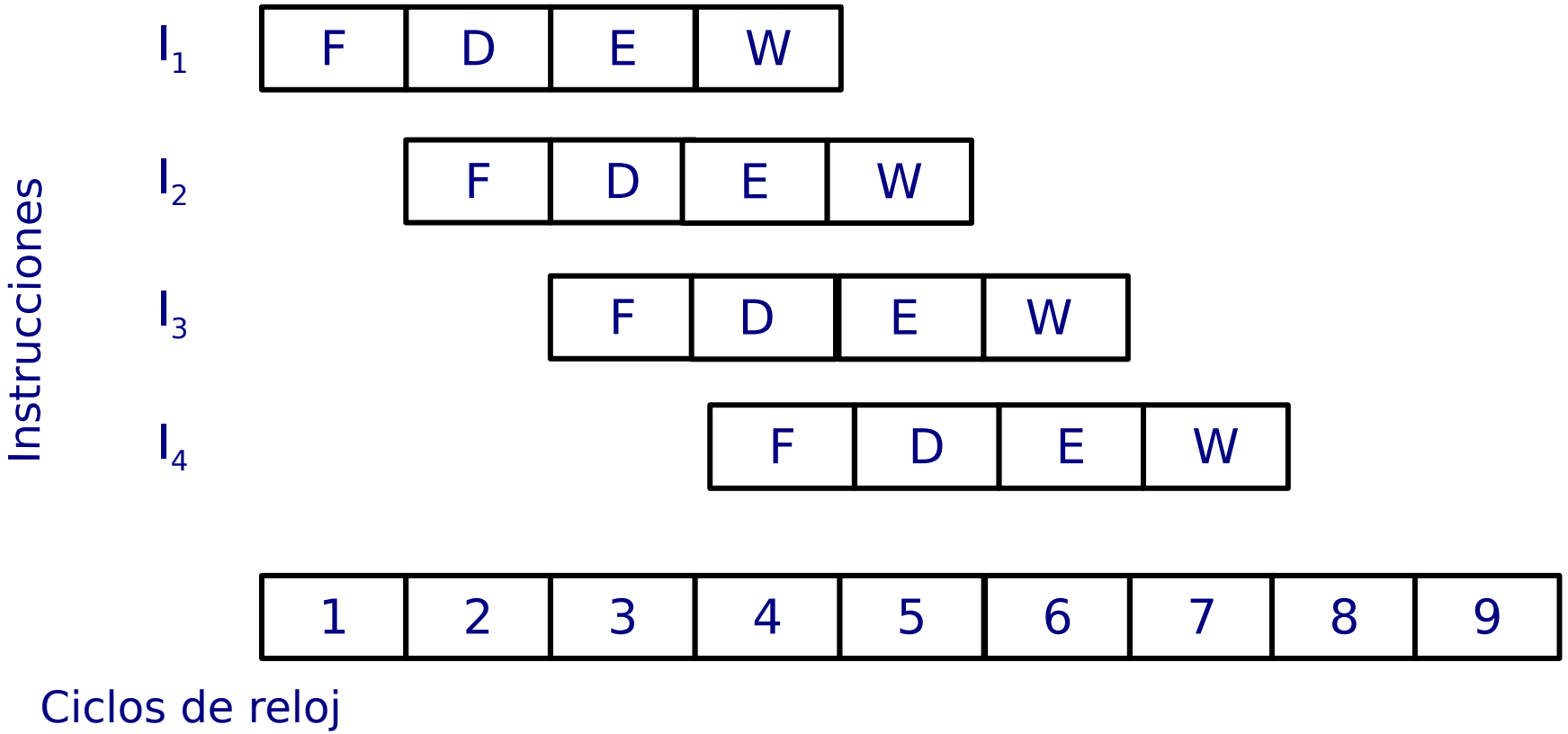
- Búsqueda (F) Fetch o Captación
  - Acceso a memoria cache (búsqueda de la instrucción)
  - Incremento del CP
- Decodificación (D)
  - Decodificación de la instrucción
  - Obtención de operandos
- Ejecución (E)
  - Procesamiento: ejecución en la ALU
  - Acceso a memoria: cálculo de dirección efectiva y/o lectura
  - Salto: cálculo del destino y decisión de salto (si/no)
- Actualización (W)
  - Almacenamiento del resultado (si lo hay)

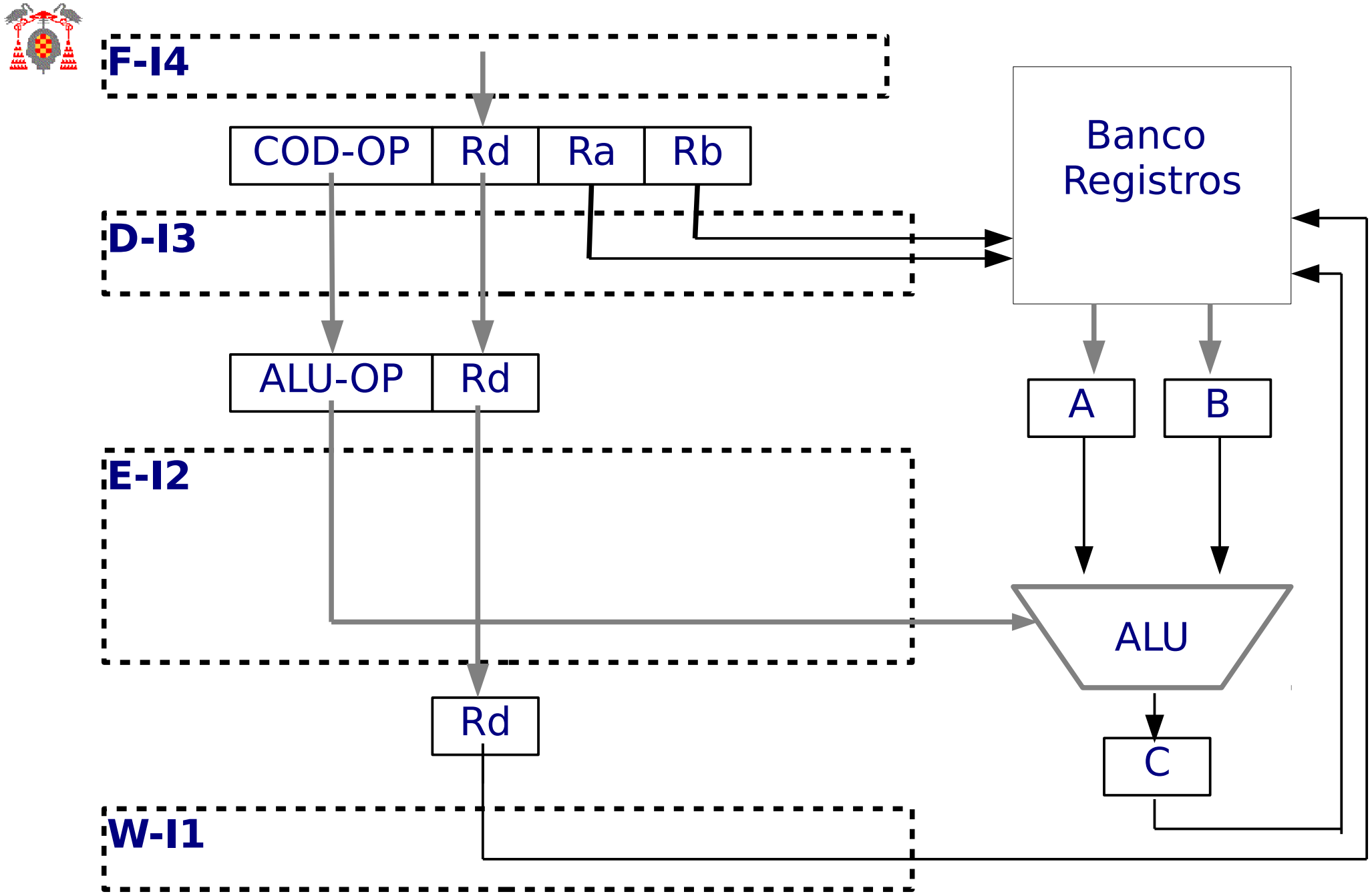
# Organización del Hardware




Los registros de segmentación contienen los datos de la instrucción que se encuentra en cada etapa

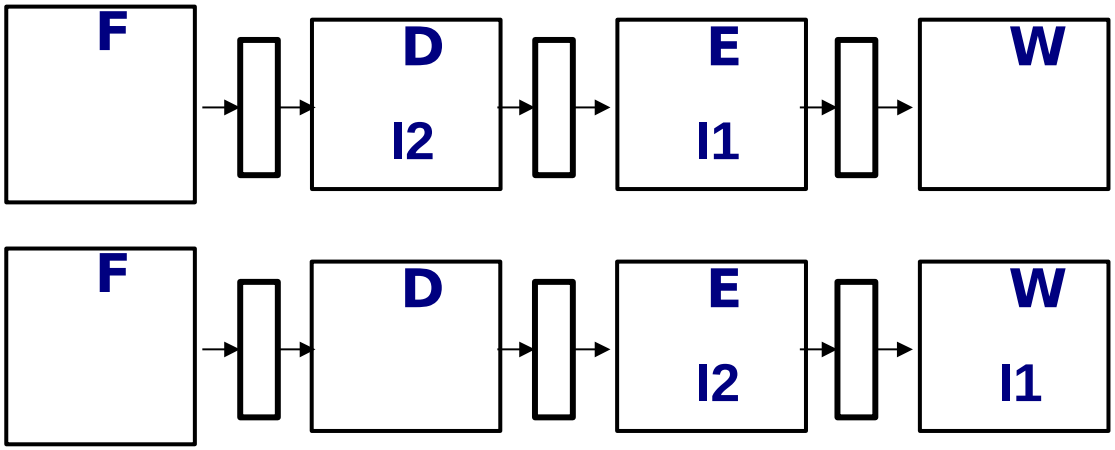
# Flujo en el cauce sin riesgos




F-I4
COD-OP
Rd
Ra
Rb
D-I3
ALU-OP
Rd
E-I2
Rd
W-I1
Banco
Registros
A
B
ALU
C

# RAW = Read After Write

I1: Add R1, R2, R3 R1 <-- R2 + R3 "**Write**" en R1 fase W
I2: Sub R4, R4, R1 R4 <-- R4 + R1 "**Read**" R1 fase D




Veamos cómo se produciría un resultado erróneo

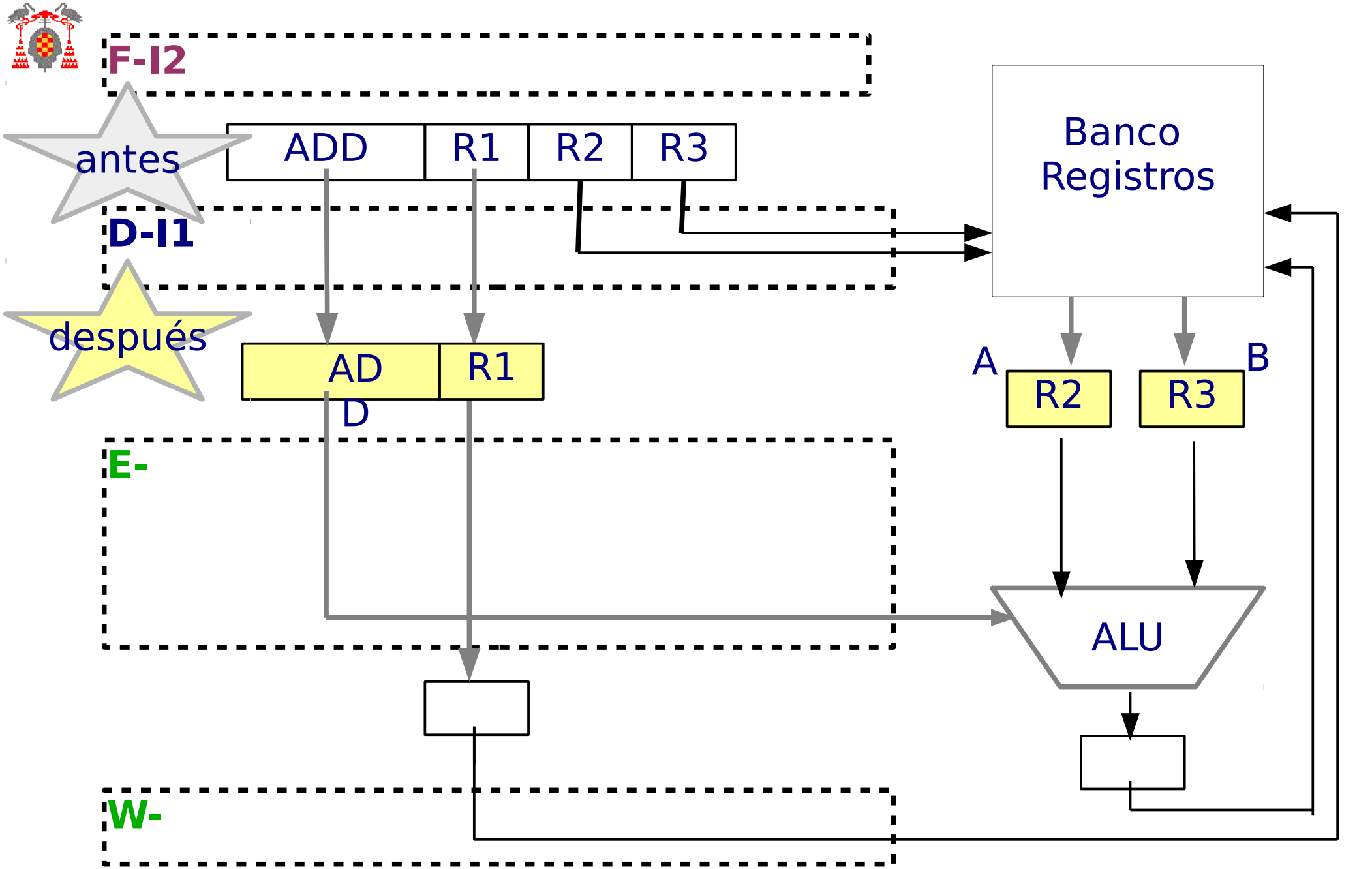
# Ejemplo

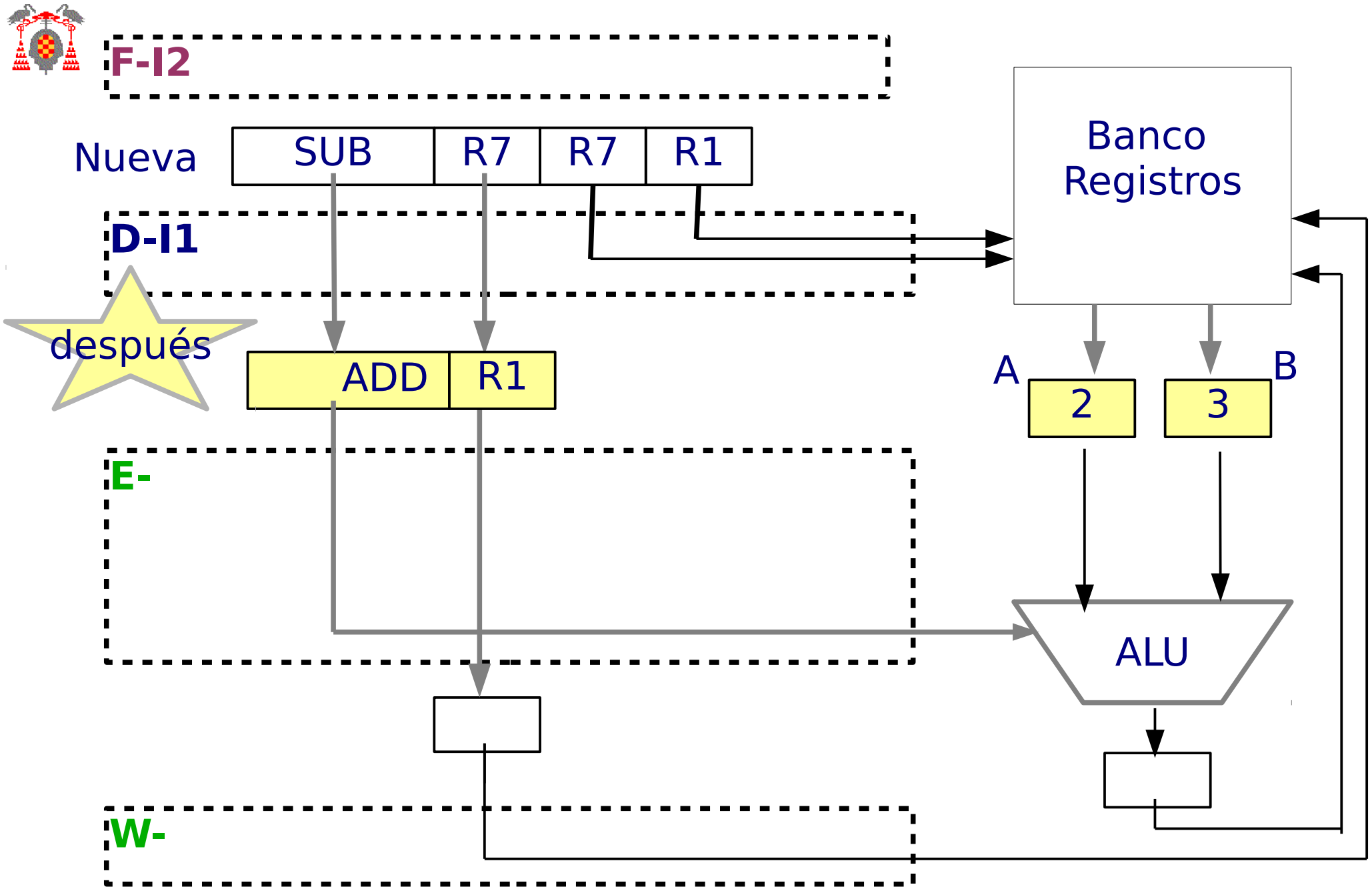
Supongamos que R1=1, R2=2, R3=3...

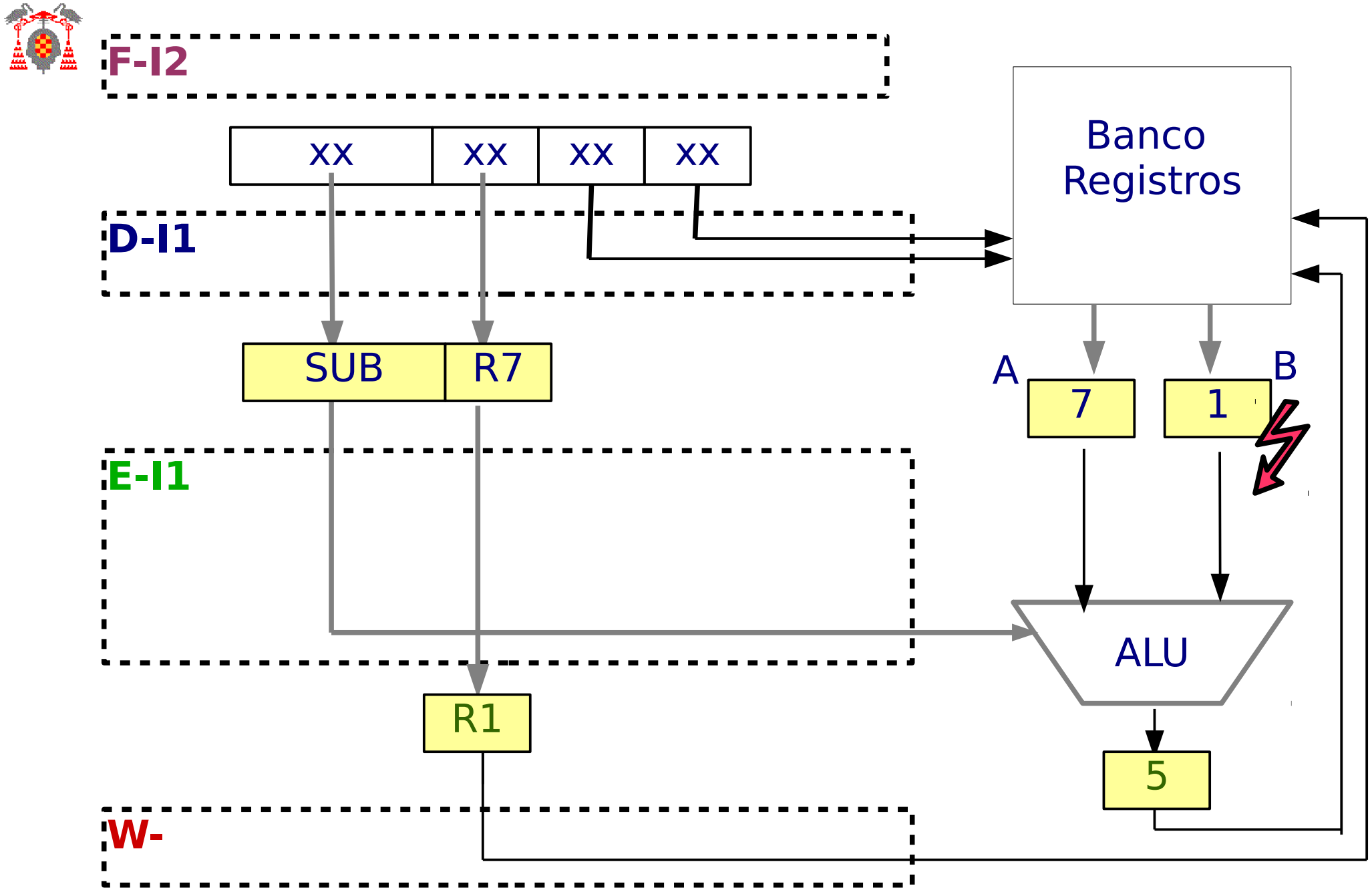
Ejecución errónea en el cauce

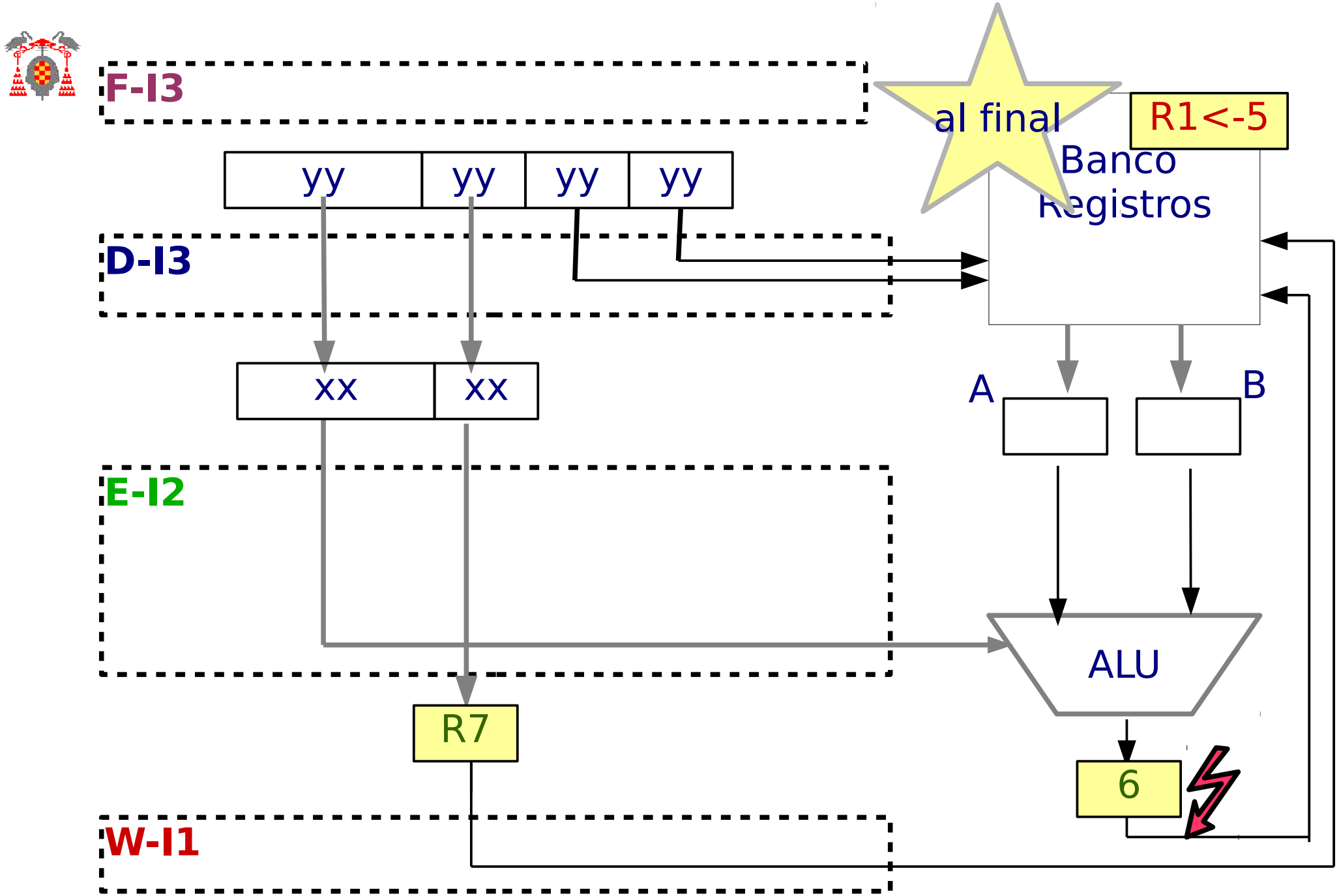
I1: ADD R1,R2,R3  (R1<--R2+R3  =5)

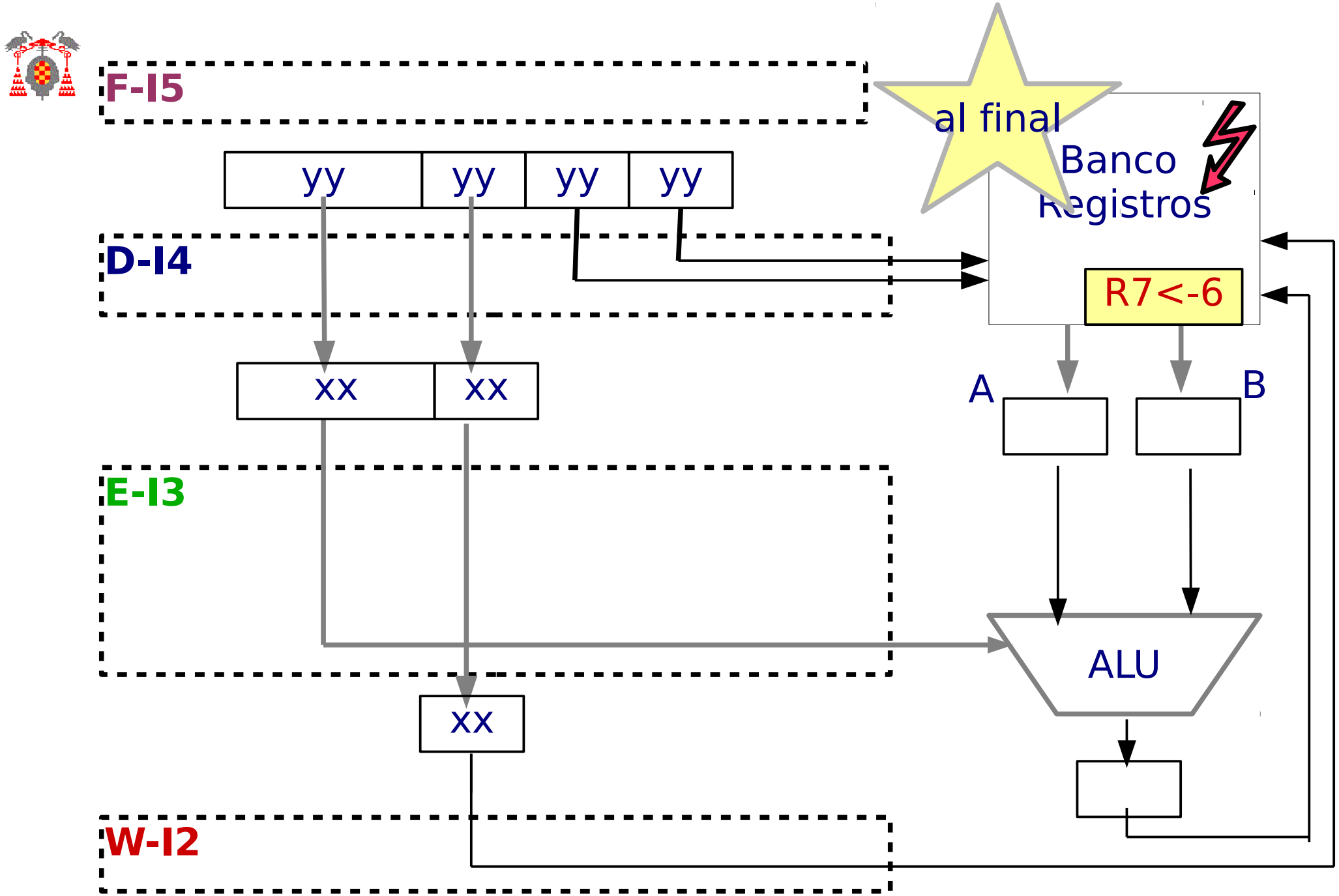
I2: SUB R7,R7,R1  (R7 <--R7-R1  = 6  vs 2 )


F-I2
antes
ADD
R1
R2
R3
D-I1
después
AD
R1
D
Banco
Registros
A
B
R2
R3
E-
ALU
W-


F-I2
Nueva
SUB
R7
R7
R1
Banco Registros
D-I1
después
ADD
R1
A
B
2
3
E-
ALU
W-


F-I2
xx
xx
xx
xx
Banco
Registros
D-I1
SUB
R7
A
B
7
1
E-I1
ALU
R1
5
W-

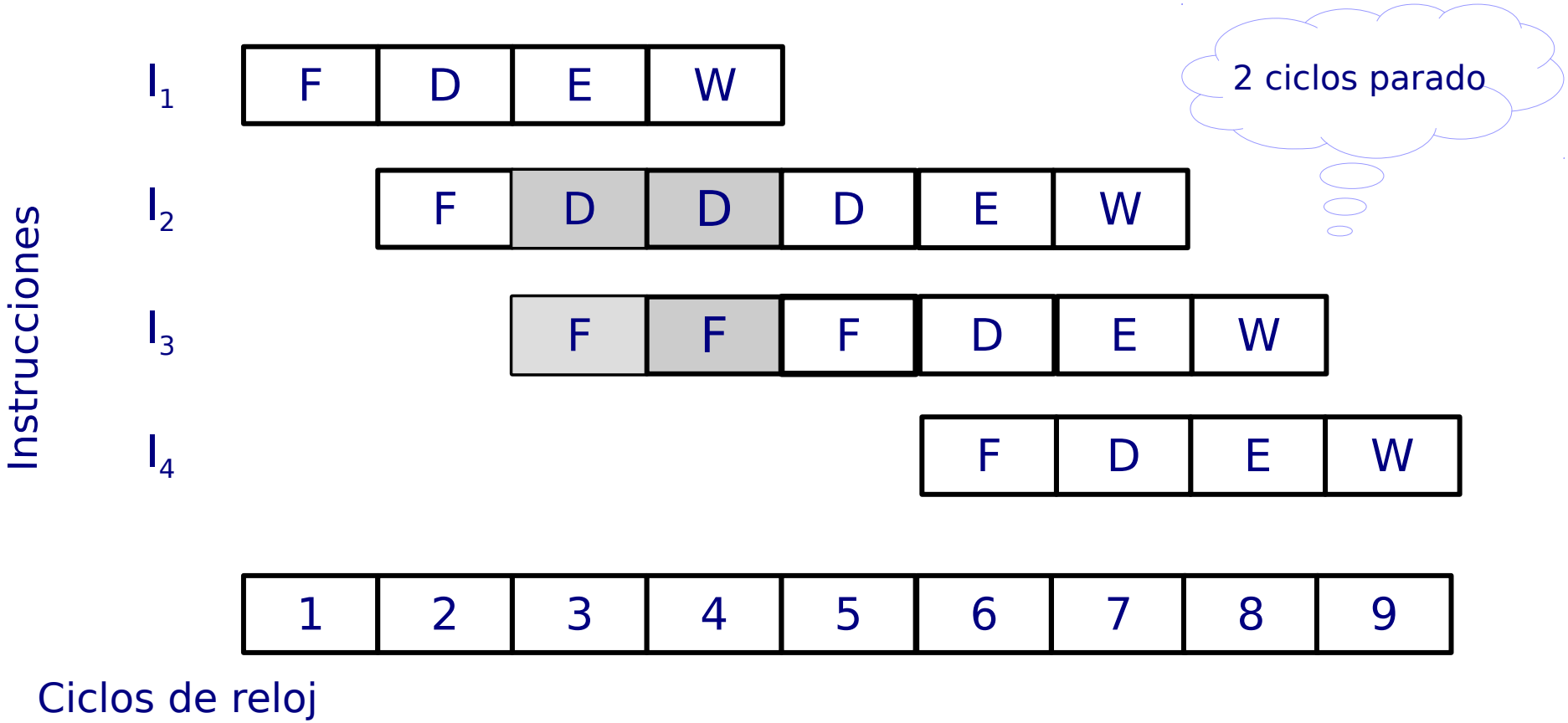

F-I3
yy
yy
yy
yy
D-I3
xx
xx
E-I2
R7
W-I1
al final
R1<-5
Banco
Registros
A
B
ALU
6


F-I5
al final
Banco
Registros
yy
yy
yy
yy
D-I4
R7<-6
xx
xx
A
B
E-I3
ALU
xx
W-I2

# Inhibir la EMISIÓN

- Cuando una instrucción supera la etapa de DECODIFICACIÓN decimos que se ha EMITIDO
- Si una instrucción que esta en decodificación tiene una dependencia de otra que está en el cauce, NO SE EMITE
- De este modo se evita una ejecución errónea

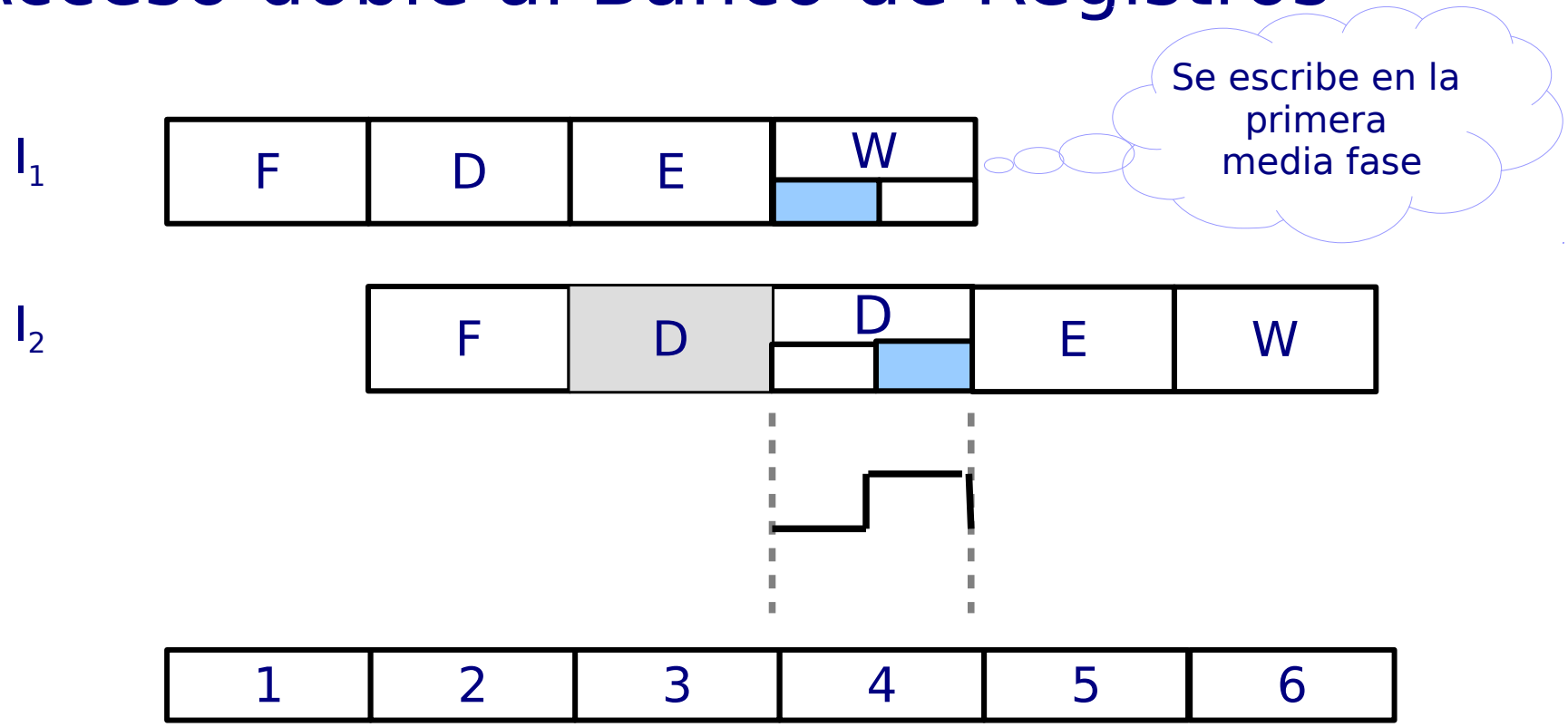
# Las dependencias de datos producen paradas



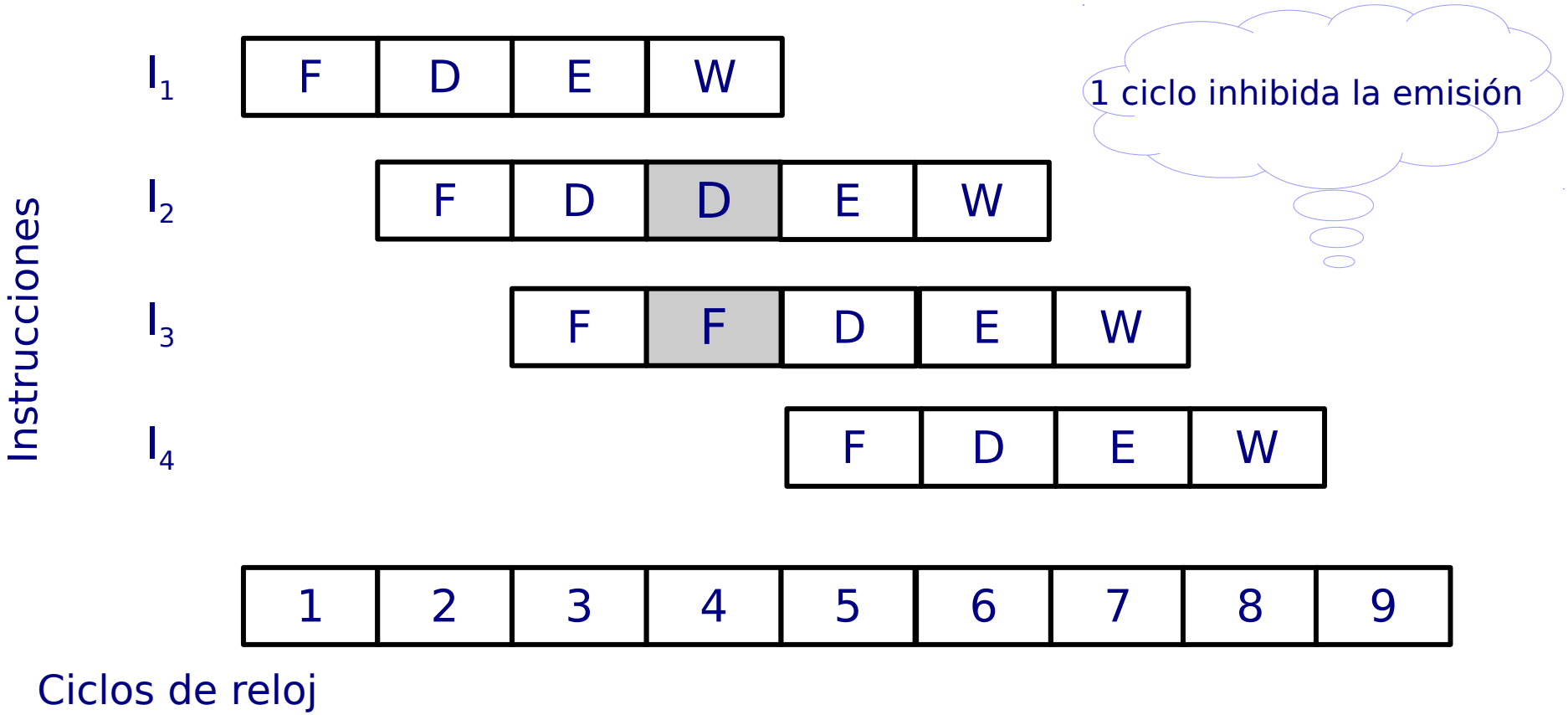
# Doble acceso al banco de Registros

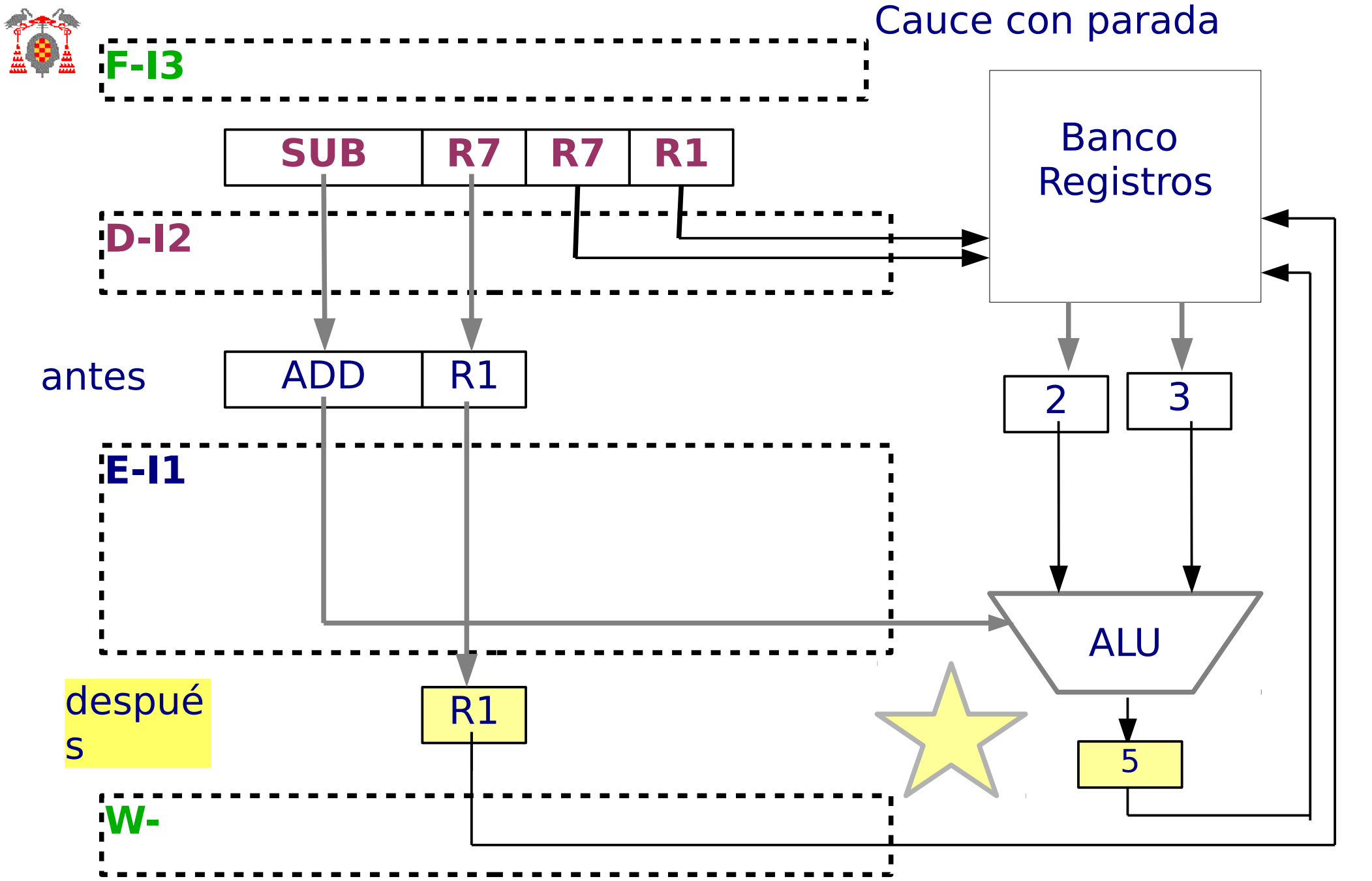
- Podemos hacer que el banco de registros se accedido simultáneamente en el mismo ciclo de reloj
  - Escrituras en la primera mitad de W
  - Lecturas en la segunda mitad de W
- Esto permite decrementar un ciclo de parada en el cauce
- --> Si una instrucción en decodificación tiene dependencia con la instrucción anterior no se emite durante 1 ciclo

# Acceso doble al Banco de Registros

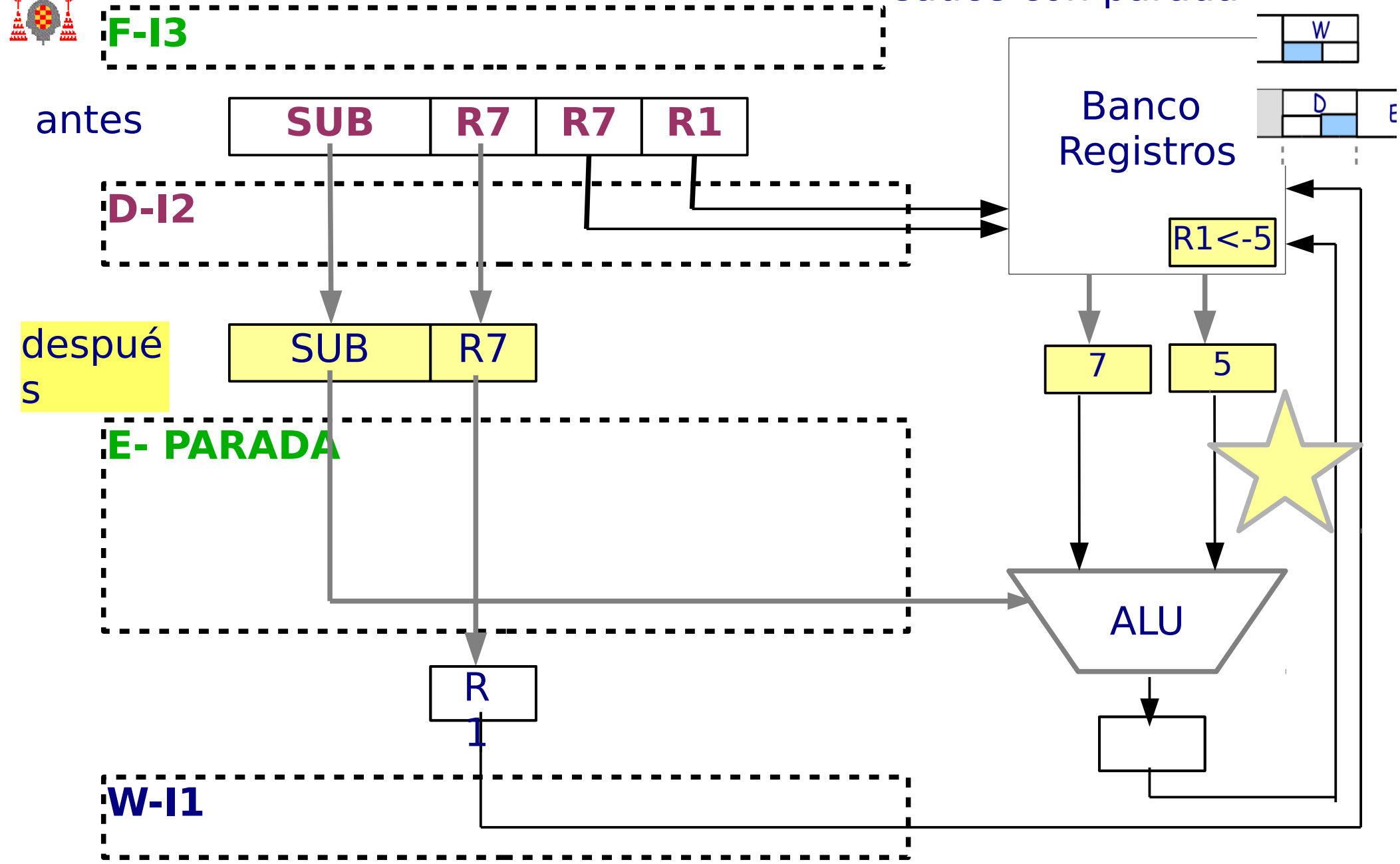


# Cauce de 4 etapas con una parada

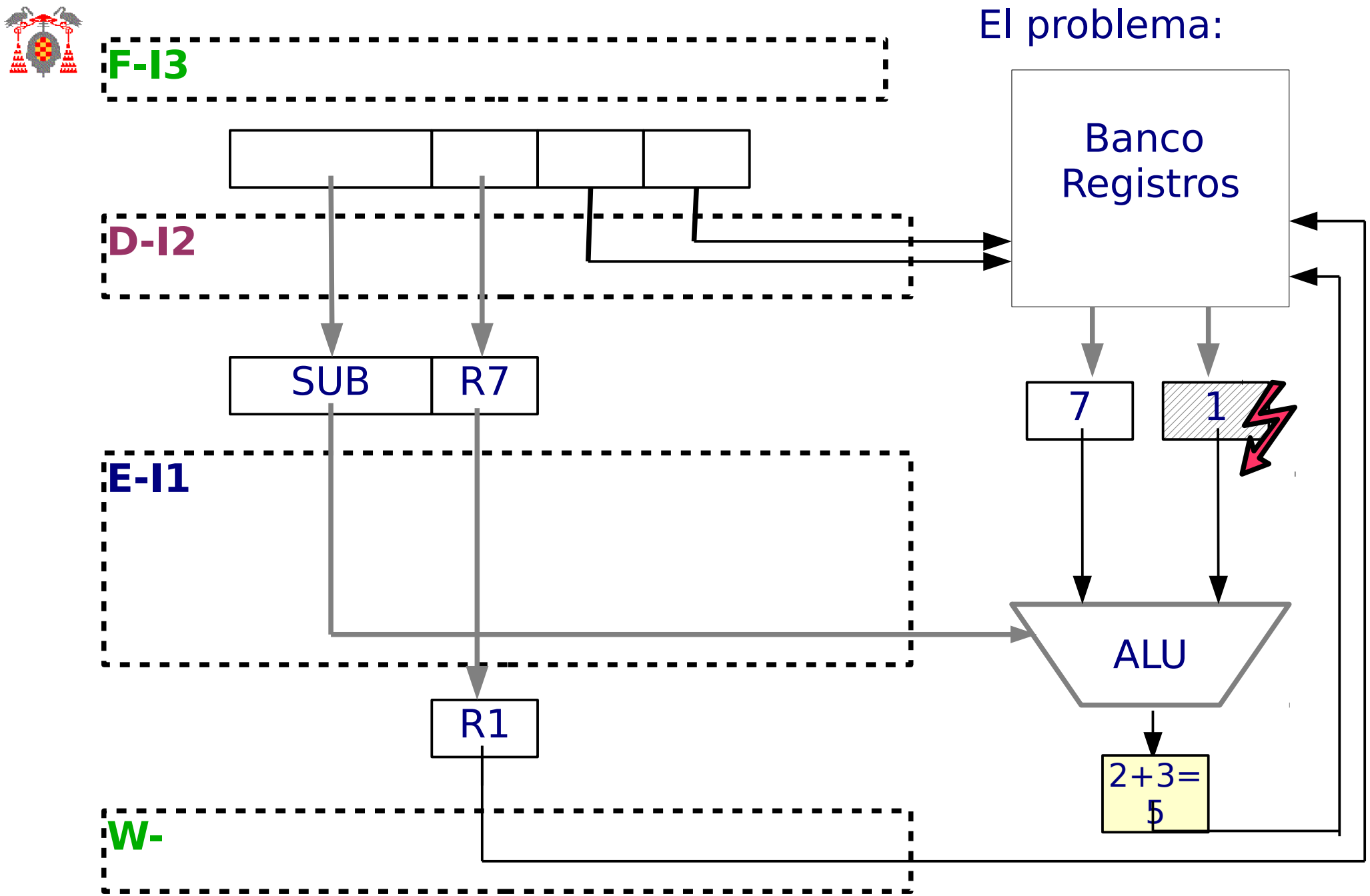



Cauce con parada
F-I3
SUB
R7
R7
R1
Banco Registros
D-I2
antes
ADD
R1
2
3
E-I1
ALU
después
R1
5
W-

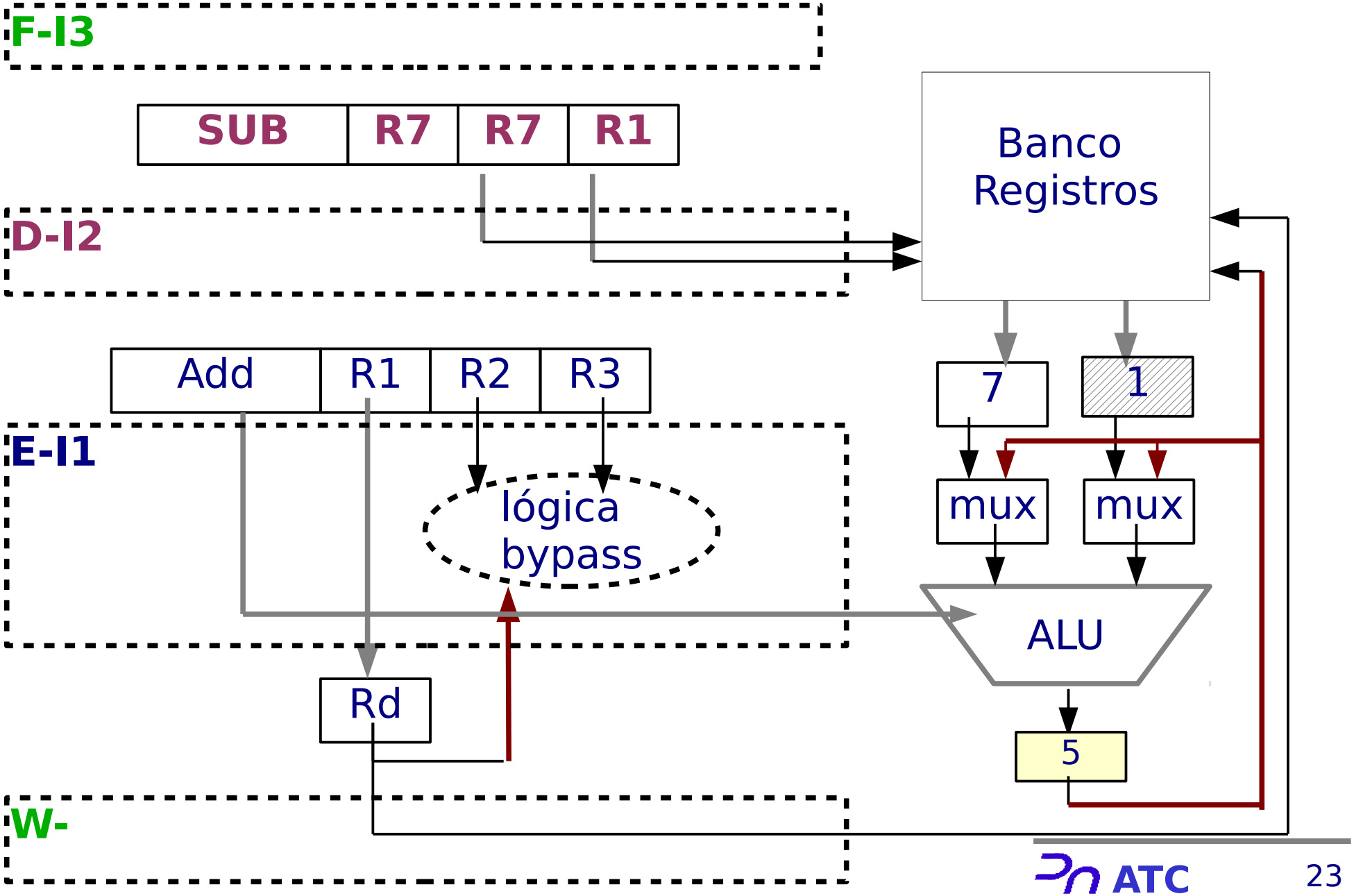
# Cauce con parada



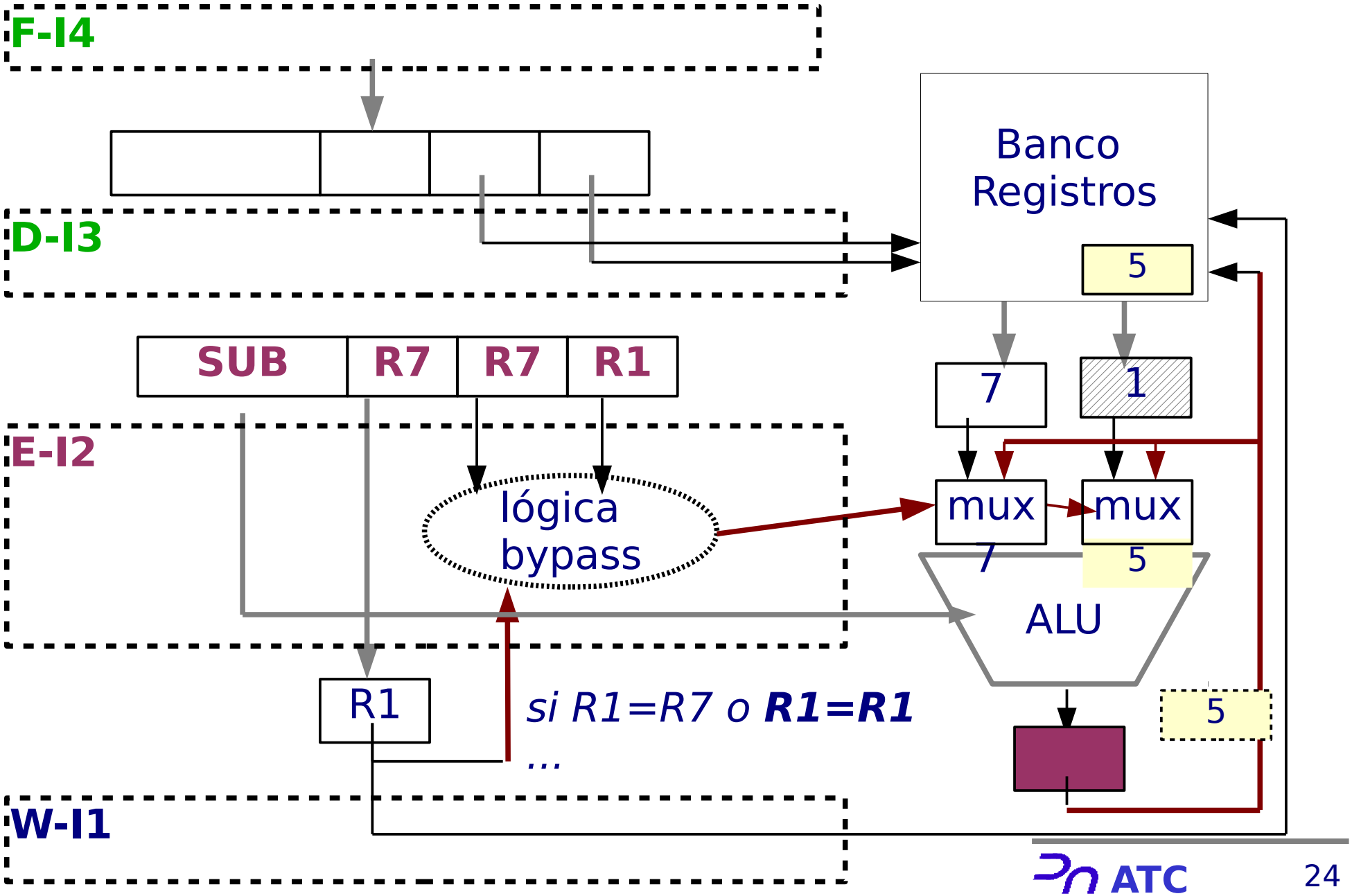
# Una nueva mejora:
# Circuito de adelantamiento


El problema:
F-I3
D-I2
E-I1
W-
Banco
Registros
SUB
R7
R1
7
1
ALU
2+3=
5

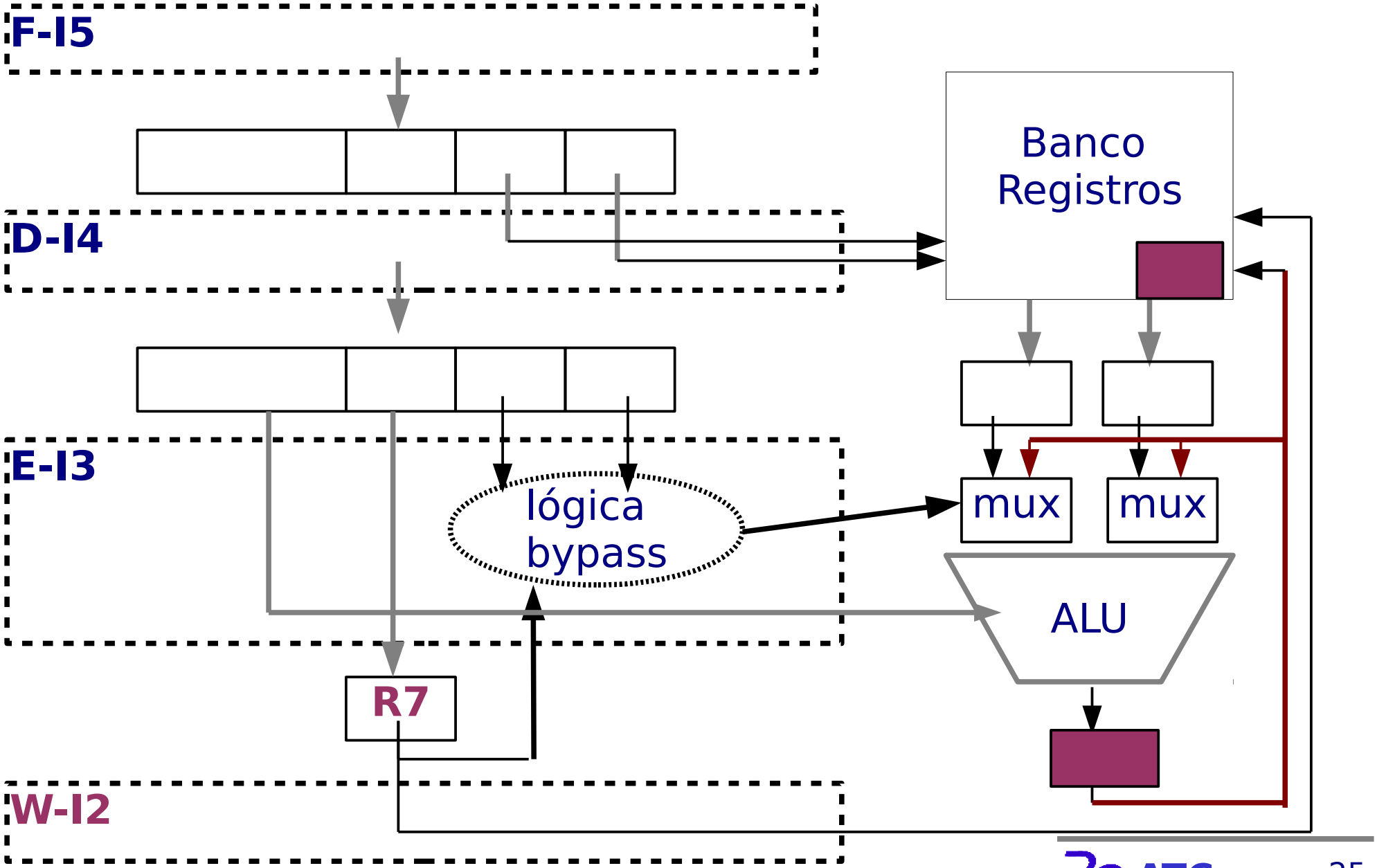
# Cauce con circuito de adelantamiento



# Cauce con circuito de adelantamiento

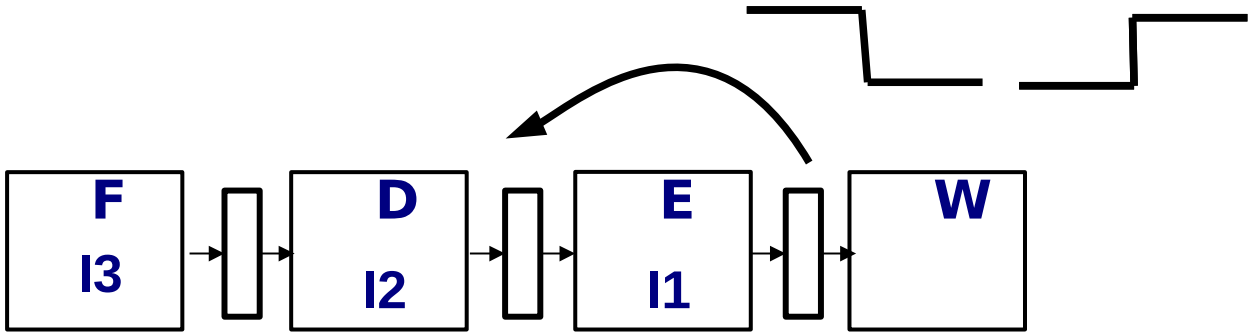


# Cauce con circuito de adelantamiento

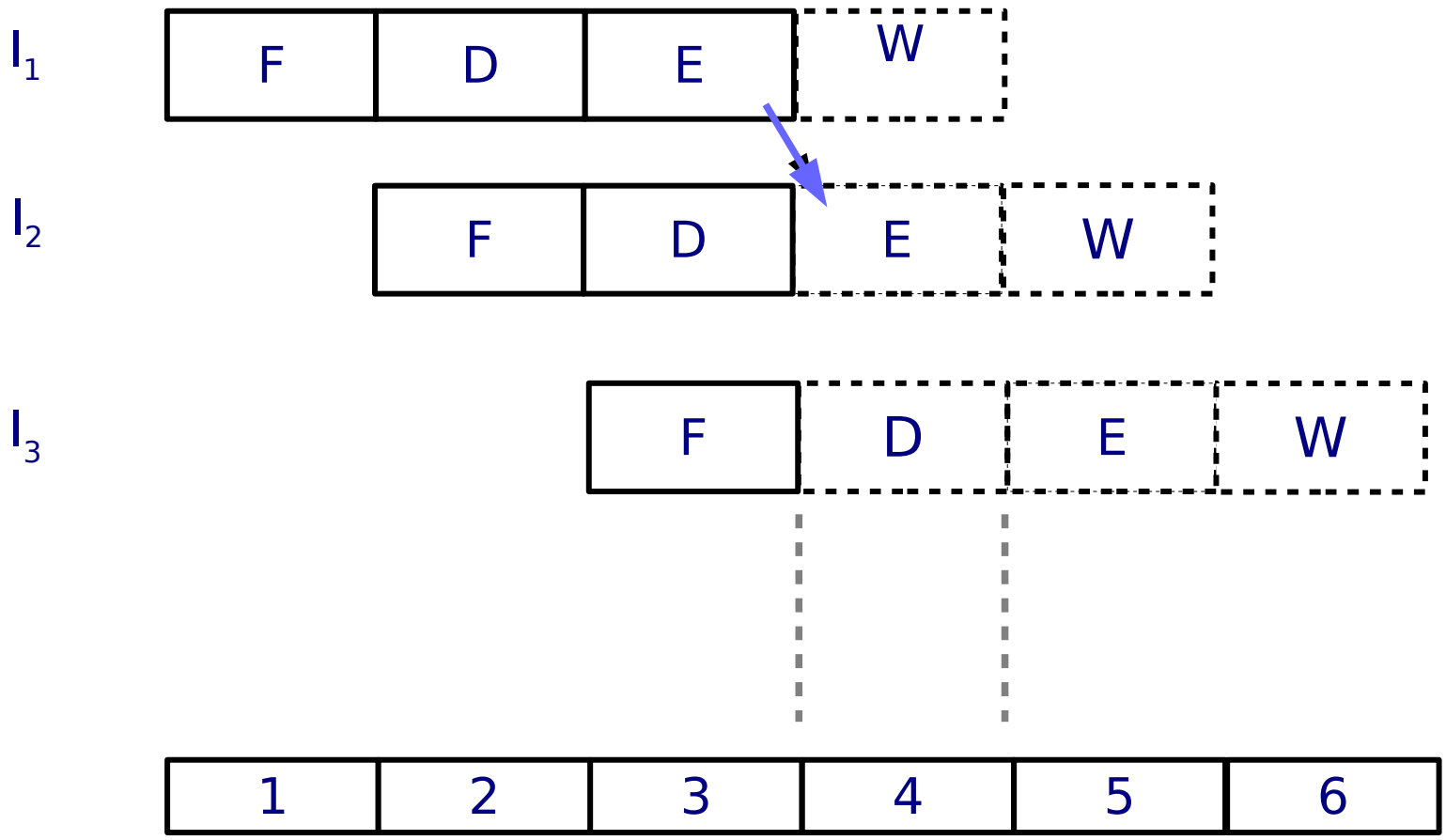


# Se evita la parada que queda

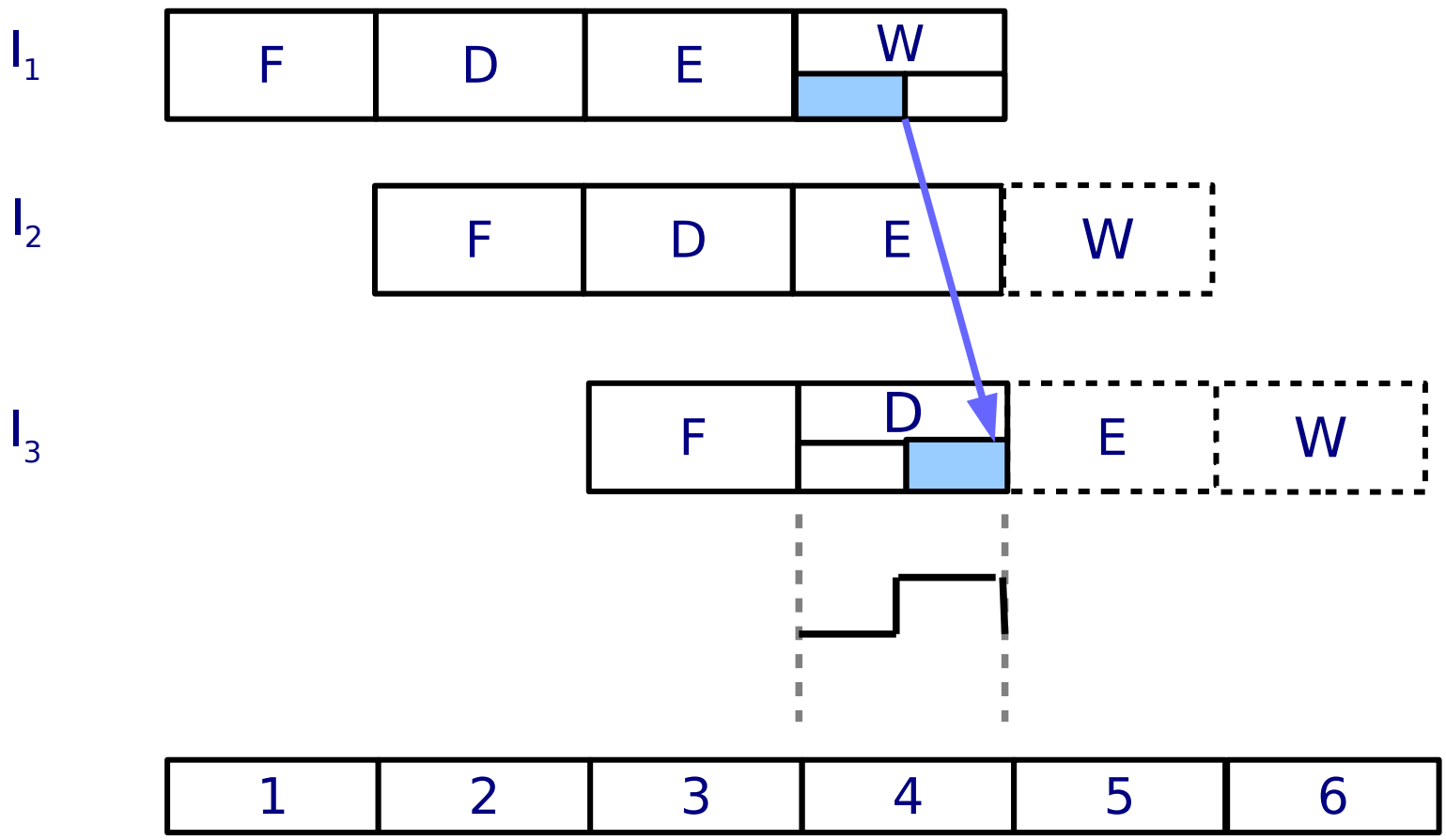
I1: Add **R1**, R2, R3 R1 <-- R2 + R3 "**Write**" en Reg-Seg @ salida E
I2: Sub R7, R7, **R1** R7 <-- R7 + R1 "**Read**" en MUX @ entrada E desde Reg-Seg @ salida E
I3: Xor R5, R5, **R1** R3 <-- R5 | R1 Bco Registros accesos @ E/D



# Adelanto a I2: sin paradas



# Acceso doble al BR para I3: sin paradas

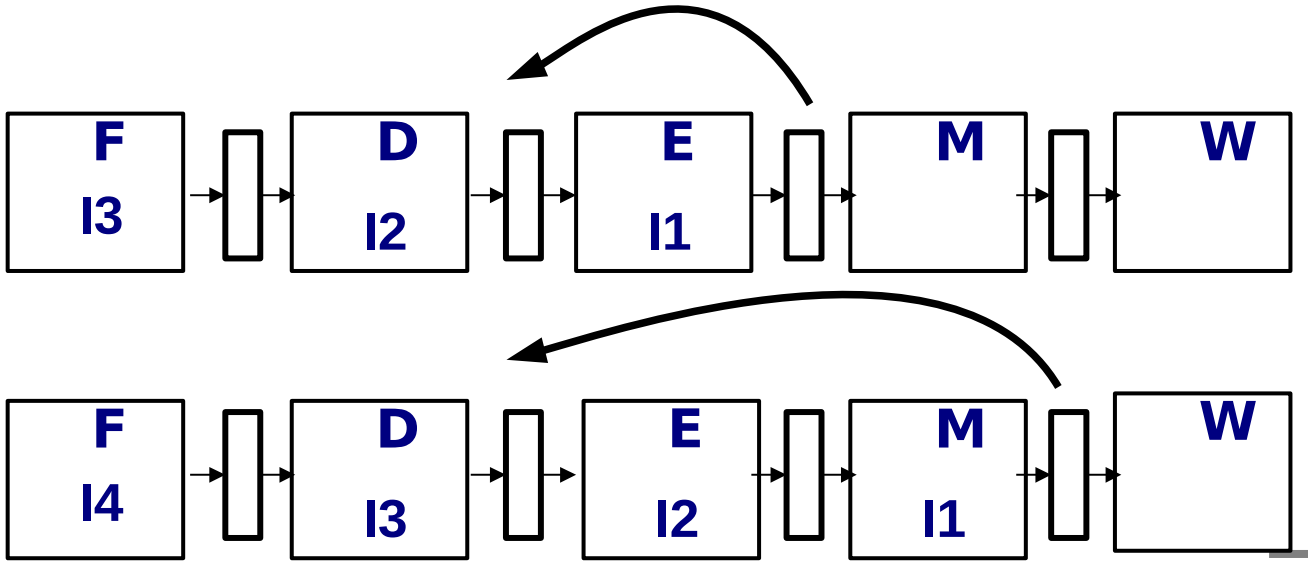


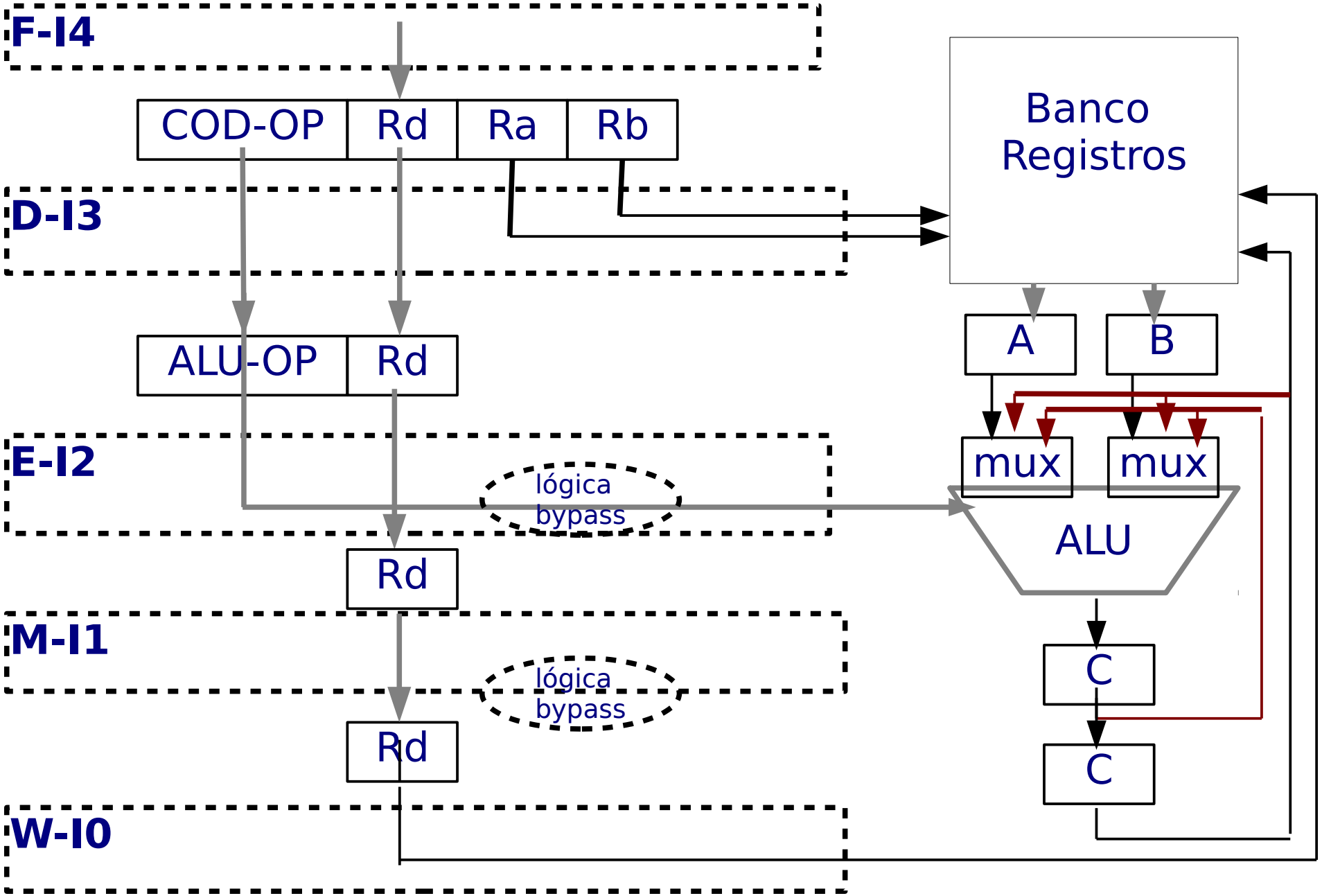
# Cauce de 5 etapas

- F=IF Búsqueda de la instrucción (F)
- D=ID Decodificación, lectura de Registros fuente (D)
- X=EX: (E)
  - ALU: Operación en la ALU
  - L/S: cálculo de direcciones efectivas
  - SALTO: carga del PC
- M=ME Acceso a memoria en instrucciones L/S (resto: nada)
- W=WB Escritura de Registros (W)

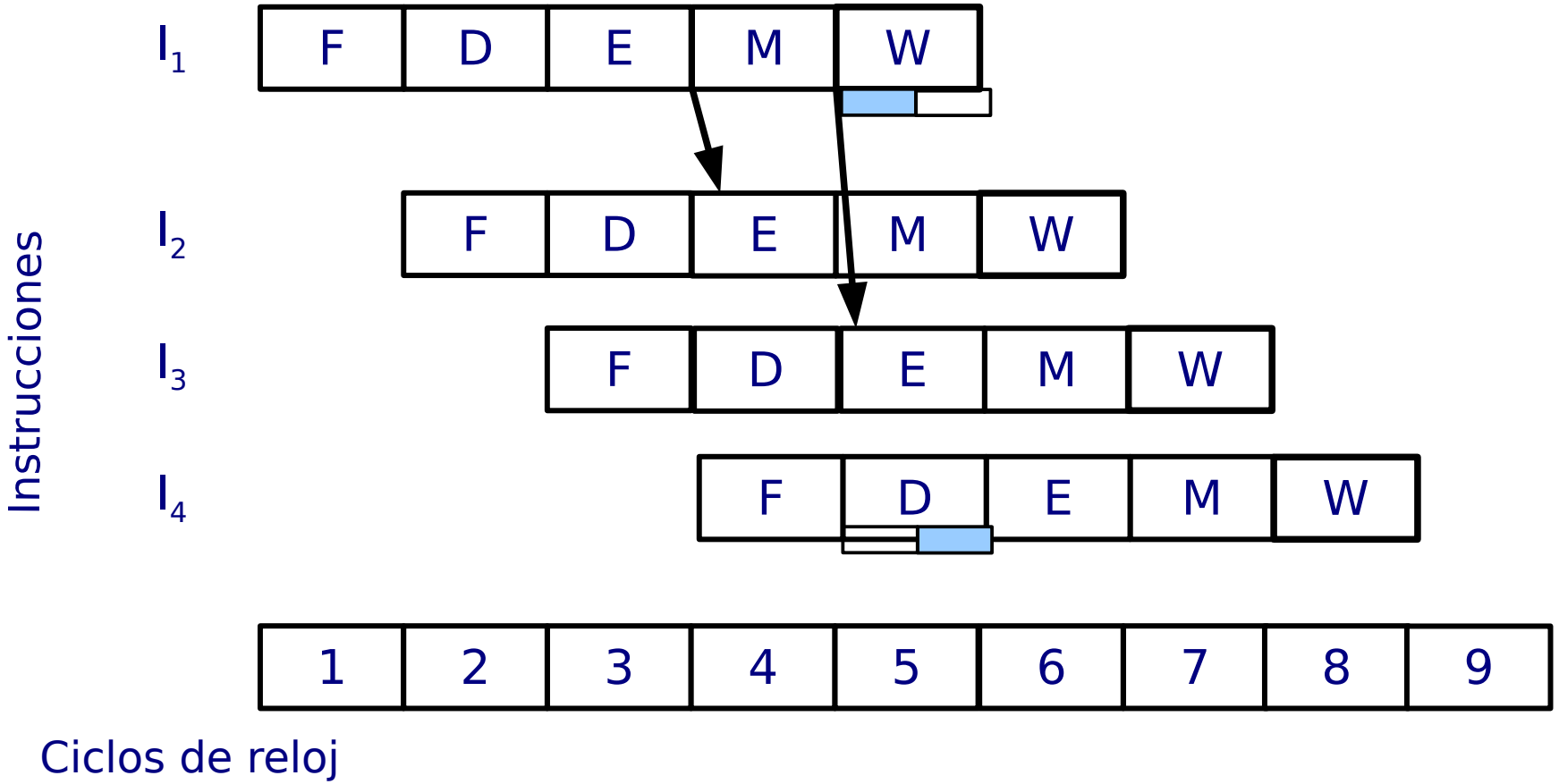
# Circuito de adelantamiento doble

| | | | |
|---|---|---|---|
| I1: Add | **R1**, R2, R3 | R1 <-- R2 + R3 | "**Write**" en Reg-Seg @ salida E |
| I2: Sub | R4, R4, **R1** | R4 <-- R4 + R1 | "**Read**" en MUX @ entrada E desde Reg-Seg @ salida E |
| I3: Xor | R5, R5, **R1** | R3 <-- R5 \| R1 | "**Read**" en MUX @ entrada E desde Reg-Seg @ salida M |
| I4: And | R6, R2, **R1** | R6 <-- R2 & R1 | Bco Registros **accesos @ E/D** |




F-I4
COD-OP
Rd
Ra
Rb
D-I3
ALU-OP
Rd
E-I2
lógica
bypass
Rd
M-I1
lógica
bypass
Rd
W-I0
Banco
Registros
A
B
mux
mux
ALU
C
C

# Sin parada con C. Adelantamiento 5 etapas



# Detenciones en 5 etapas

- Si una instrucción (como I3) tiene dependencia de una anterior
  - si depende del resultado de EX de la inmediatamente anterior se emite (ALU-ALU, SW-ALU)
  - si depende del resultado de ME de la inmediatamente anterior NO se emite (LD-ALU)
  - en DLX: si es un salto y depende del resultado de EX de la inmediatamente anterior NO se emite pues el salto se resuelve en DE
  - Estudiar cada caso realizando diagramas de cauce como los vistos