Tema 06: Diseño de un CPU en pipelining

Arquitectura de Computadoras

Ing. Nicolás Majorel Padilla (npadilla@herrera.unt.edu.ar)

http://microprocesadores.unt.edu.ar/arqcom/

Temas que veremos

- Determinación de las etapas del procesador en pipeline.
- Análisis inicial de performance.
- Construcción del camino de datos.
- Señales de control.
- Riesgos del pipeline: tres tipos.
- Paradas del pipeline.
- Adelantamiento y Reordenamiento.
- Introducción a la predicción de saltos.

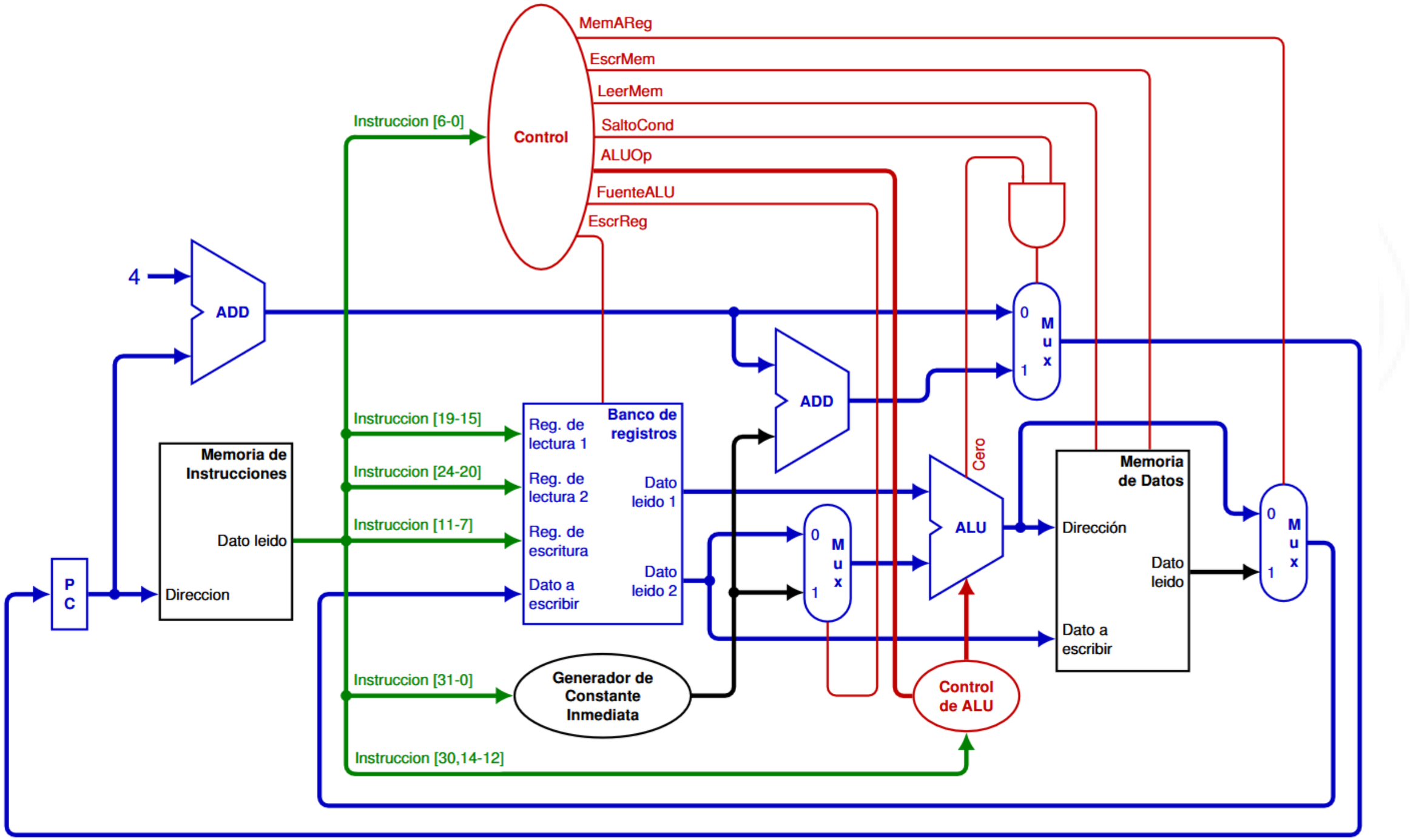
Lectura recomendada

- Computer Organization and Design, RISC-V Edition (2da ed, 2021)
  - Sección 4.6: *An Overview of Pipelining*
  - Sección 4.7: *Pipelined Datapath and Control*
  - Sección 4.8: *Data Hazards: Forwarding versus Stalling*
  - Sección 4.9: *Control Hazards*

Repaso: Pipelining

- Separar el camino de datos en etapas **independientes**.
  - Usando buffers intermedios, como en el ejemplo del sumador.
- **Objetivo**: pipeline ideal
  - Todas las etapas de la misma duración.
  - Todas las tareas pasan por todas las etapas.
  - A = m, en estado de régimen.

Repaso: Camino de datos ciclo único



Determinación de la cantidad de etapas

- El tiempo de ciclo está determinado por la instrucción que más demora, lw.
  - La que usa más unidades funcionales, de manera secuencial.
    - Memoria de instrucciones + banco de registros + operación en ALU + memoria de datos + escribir resultado en banco de registros.
      - 200 ps + 100 ps + 200 ps + 200 ps + 100 ps = 800 ps
- ***¿Cómo conviene hacer la separación?***
  - *¿Dos etapas? ¿Tres? ¿Más? ¿De qué duración?*

Determinación de la cantidad de etapas

- Consideraremos los siguientes criterios:
  - **Que en cada etapa se utilice una única unidad funcional.**
    - Para que de esa manera las etapas sean totalmente independientes.
  - **Que cada unidad funcional sea utilizada en una única etapa.**
    - Para buscar evitar conflictos con la utilización de las mismas.
    - Prestar atención al banco de registros.
  - **Que la duración del tiempo de ciclo sea mínima.**
    - Para maximizar la performance.
  - **Que sea lo más parecido posible a un pipeline ideal.**
    - Para maximizar la performance.

Determinación de la cantidad de etapas

- Para que la secuencia de timing de la instrucción lw cumpla con los criterios anteriores, **separaremos el camino de datos en cinco etapas**, de la siguiente manera:

1. **IF:** Búsqueda de la instrucción en la memoria de instrucciones.
2. **ID:** Decodificación de la instrucción y lectura de operandos en el banco de registros.
3. **EX:** Ejecución de la operación o cálculo de la dirección (de memoria o de salto) en la ALU.
4. **MEM:** Lectura o escritura en la memoria de datos.
5. **WB:** Escritura del resultado en el banco de registros.

Determinación de la cantidad de etapas

- Las cinco etapas anteriores fueron determinadas en base a la instrucción lw.
  - *¿Cómo sería para las demás instrucciones?*
  - Las mismas, porque buscamos que sea un pipeline ideal.
    - **Todas las instrucciones pasan por todas las etapas.**
    - sw pasa por la etapa 5 (WB) aunque no haga nada.
    - add pasa por la etapa 4 (MEM) aunque no haga nada.
    - beq pasa por las etapas 4 y 5 aunque no haga nada.
  - Empeora la latencia de algunas instrucciones, pero mejora la productividad total.

Determinación de la cantidad de etapas

- En las etapas determinadas, el banco de registros se utiliza dos veces.
  - En la etapa 2 (ID) para lectura y en la etapa 5 (WB) para escritura.
  - Esto no es un problema, ya que el banco de registros utilizado **posee puertos independientes para lectura y para escritura**.
  - En caso que un mismo registro se lea y se escriba en el mismo ciclo, es posible hacerlo.
    - Se escribe en la primera mitad del ciclo, y se lee en la segunda mitad del ciclo.

Determinación de la cantidad de etapas

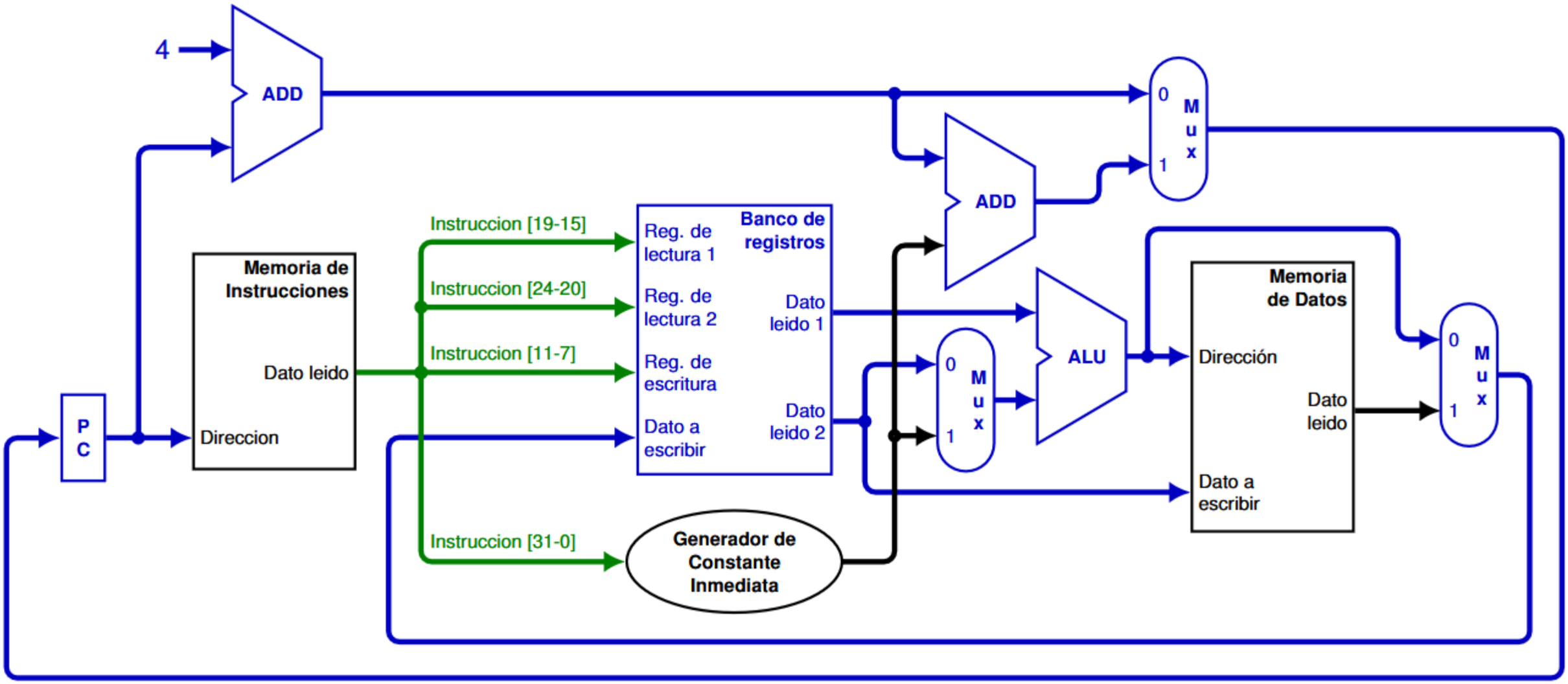
- En las cinco etapas determinadas, **la duración de las etapas queda bastante balanceada**:
  - 200 ps; 100 ps; 200 ps; 200 ps; 100 ps.
  - **No es ideal**, pero es lo más parecido posible.
- A su vez, **la duración del tiempo de ciclo estará determinada por la duración de la etapa más larga**:
  - T = max(200, 100, 200, 200, 100) = 200 ps
  - Es el mínimo posible.

Determinación de la cantidad de etapas

- La separación en cinco etapas es posible gracias al buen diseño del ISA de RISC-V:
  - Las instrucciones pueden ser buscadas de memoria en un solo ciclo.
    - Formatos de instrucción de longitud fija.
  - Las instrucciones pueden ser decodificadas en un solo ciclo.
    - Pocas instrucciones, simples. Formatos con mucha regularidad.
  - Arquitectura tipo Load-Store, con un único modo de direccionamiento para accesos a memoria (Indexado Base + Desplazamiento).
    - Se calcula la dirección de memoria en la etapa 3.
    - Se accede a memoria de datos en la etapa 4.

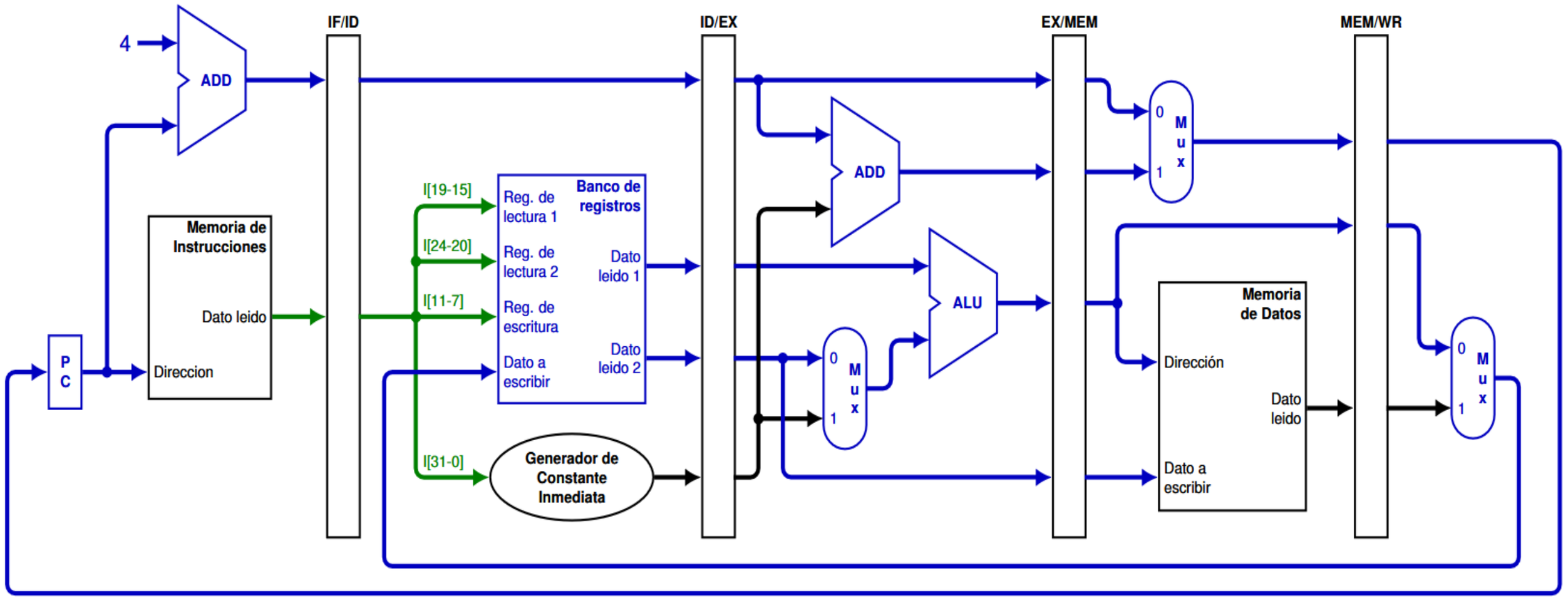
Repaso: Camino de datos ciclo único

- Volviendo a una versión previa…
  - Sin señales de control, ni el control de la ALU.

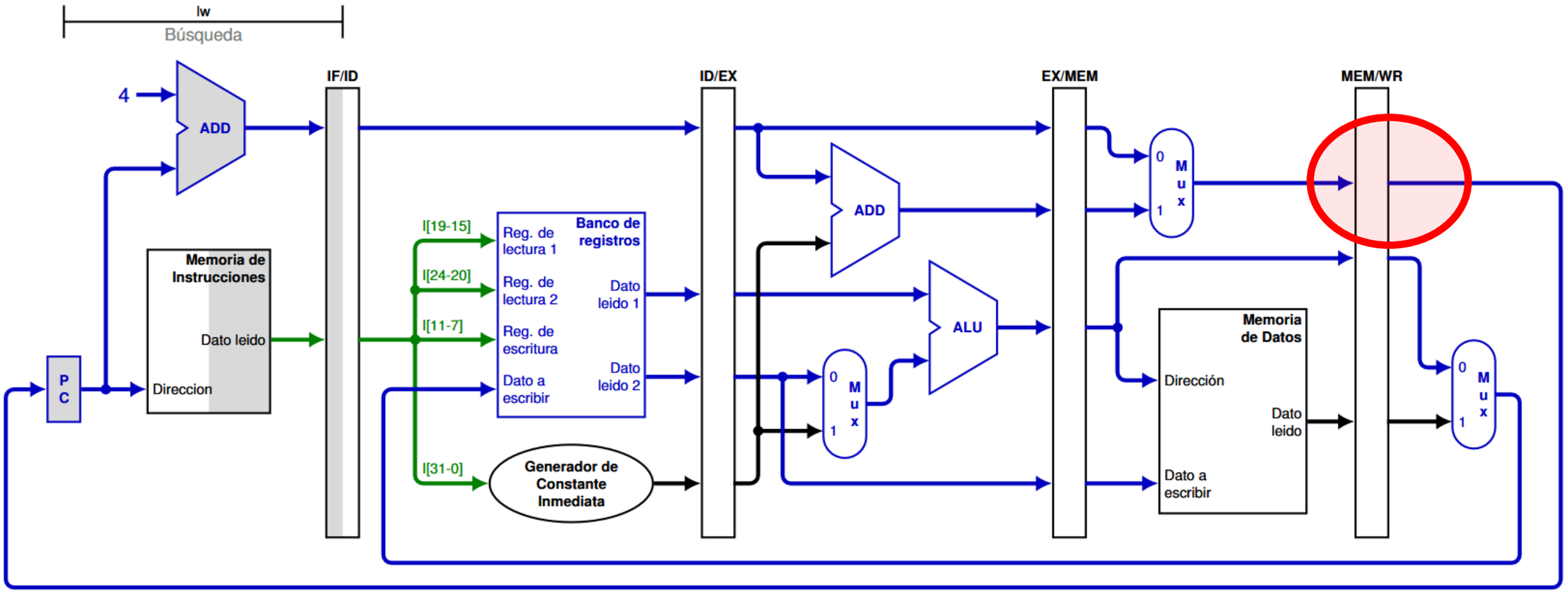


Camino de datos en pipelining

- Para separar el camino de datos en cinco etapas utilizamos cuatro registros intermedios.
  - Los componentes toman la información del registro inmediatamente anterior, y guardan los resultados en el posterior.

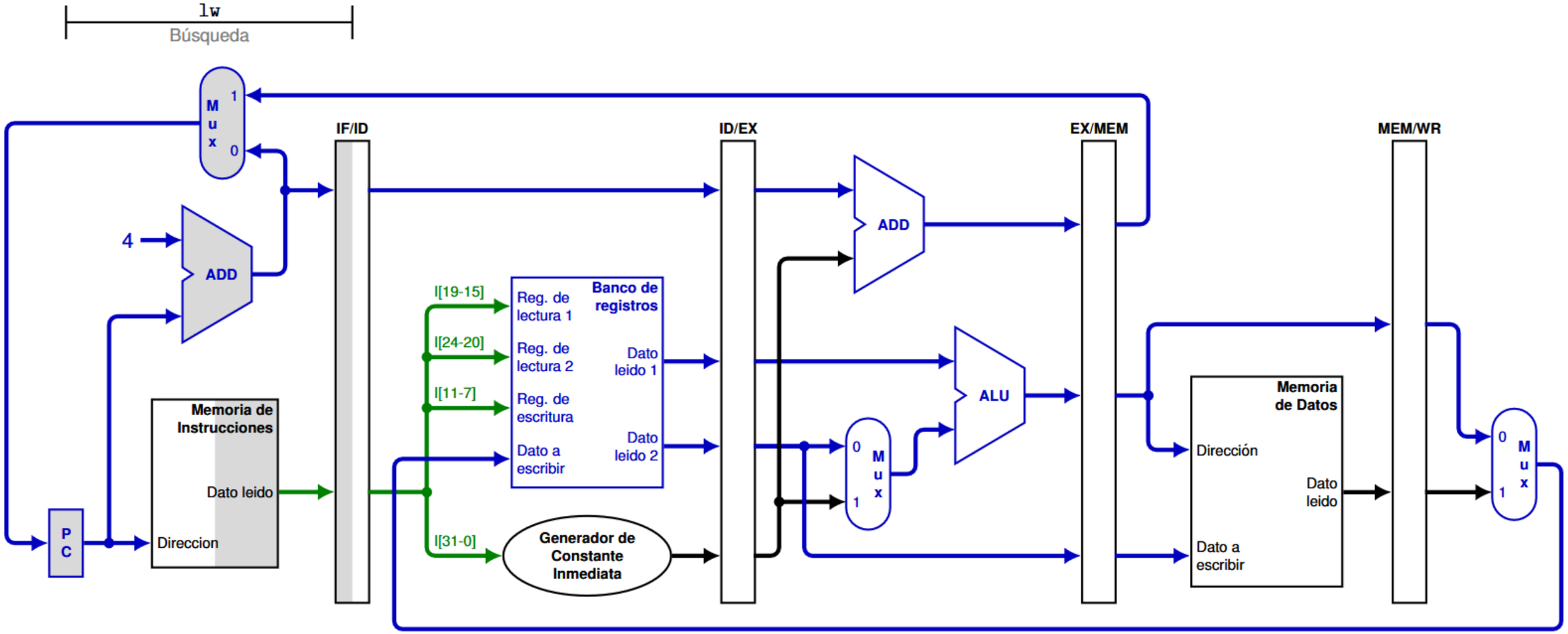


Ejemplo Ejecución lw: Etapa 1 (IF)



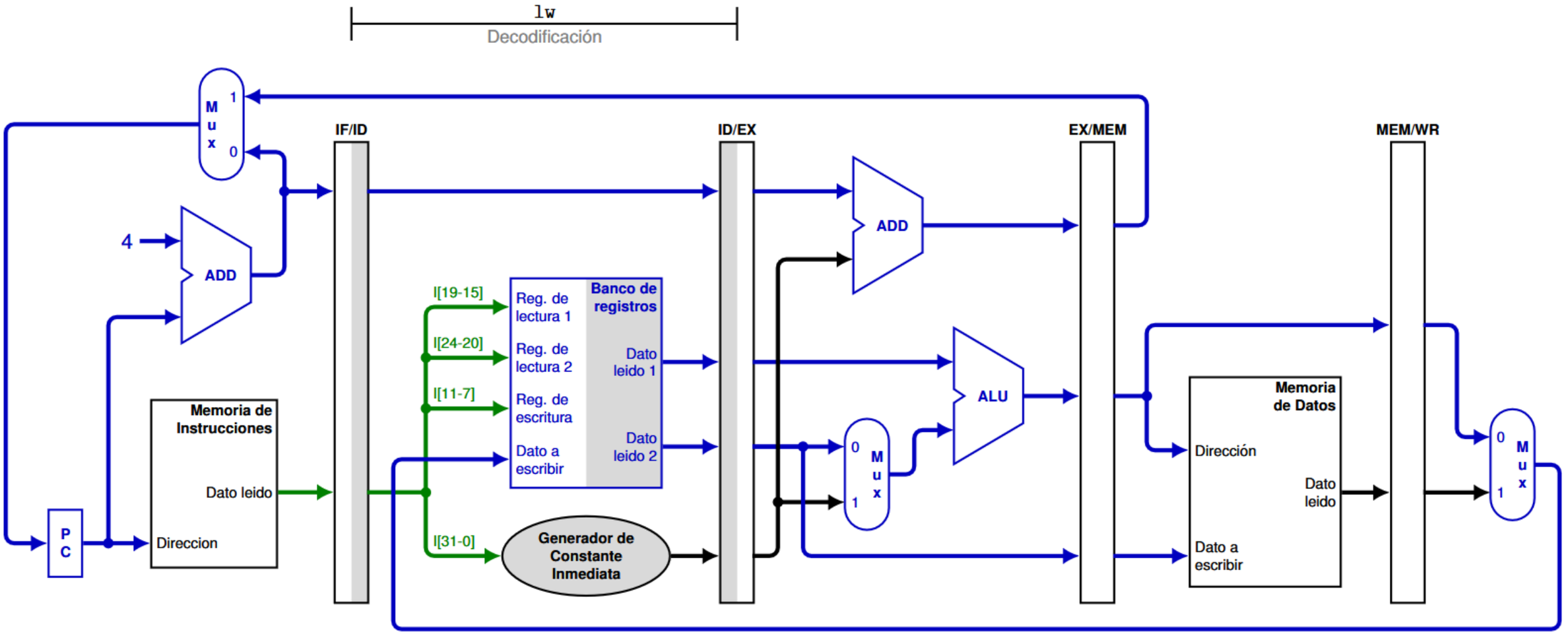
- ¡Error! No podemos actualizar PC tomando el valor de PC+4 del registro MEM/WB.
  - Porque el valor de PC+4 será el de otra instrucción.

Ejemplo Ejecución lw: Etapa 1 (IF)

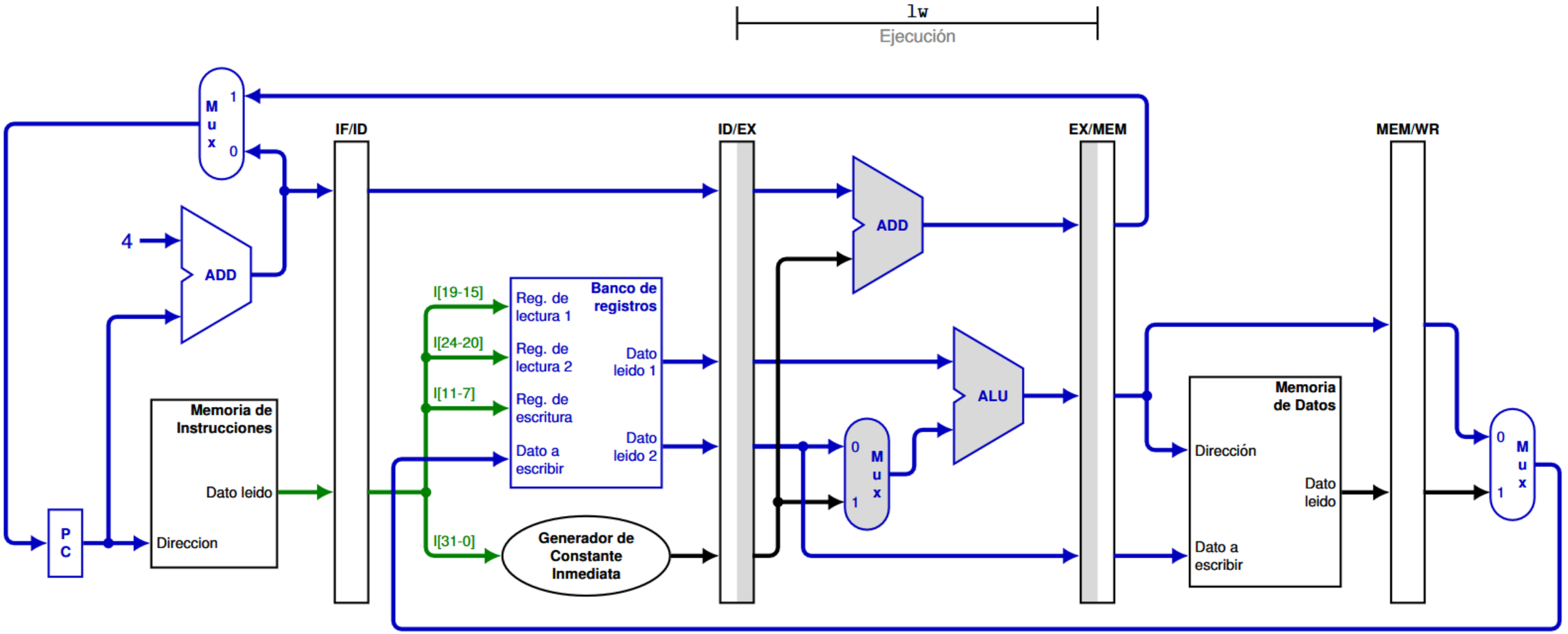


- Adelantamos el multiplexor FuentePC a la etapa 1.
  - La dirección de salto sigue siendo desde la etapa 4.

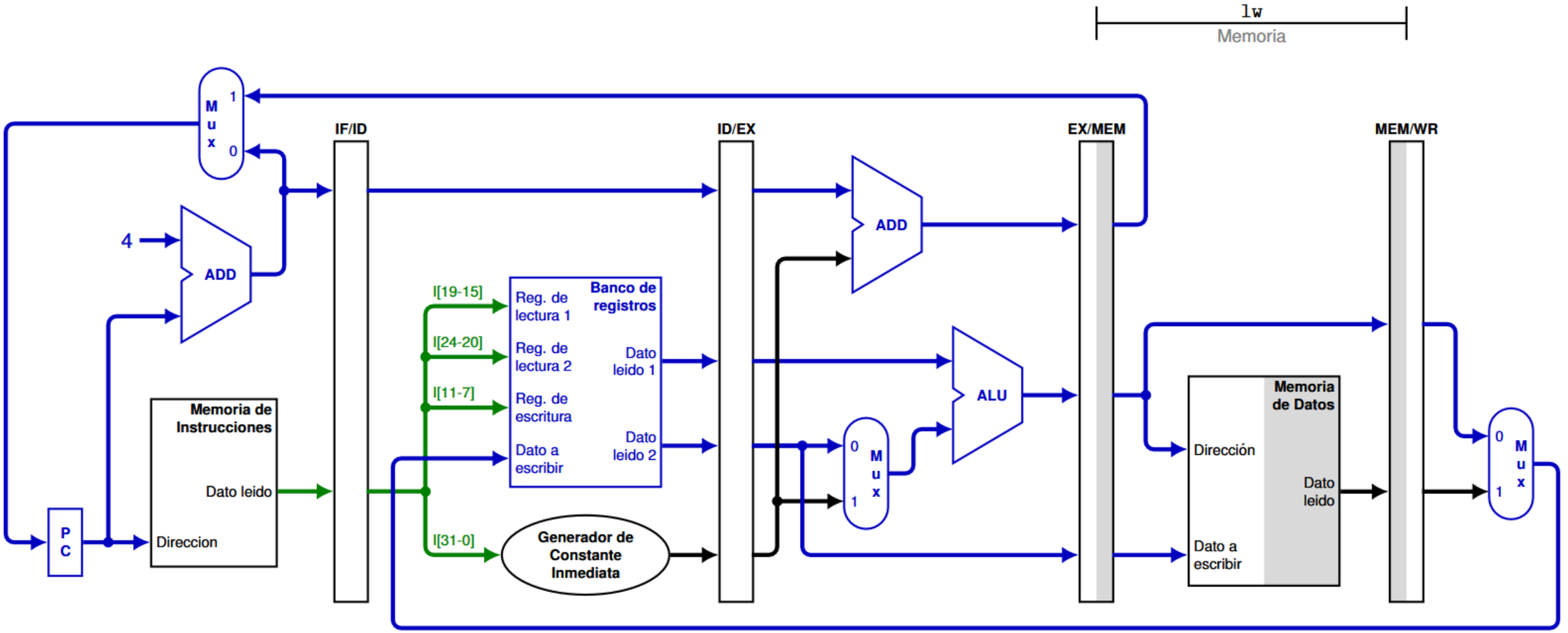
Ejemplo Ejecución lw: Etapa 2 (ID)



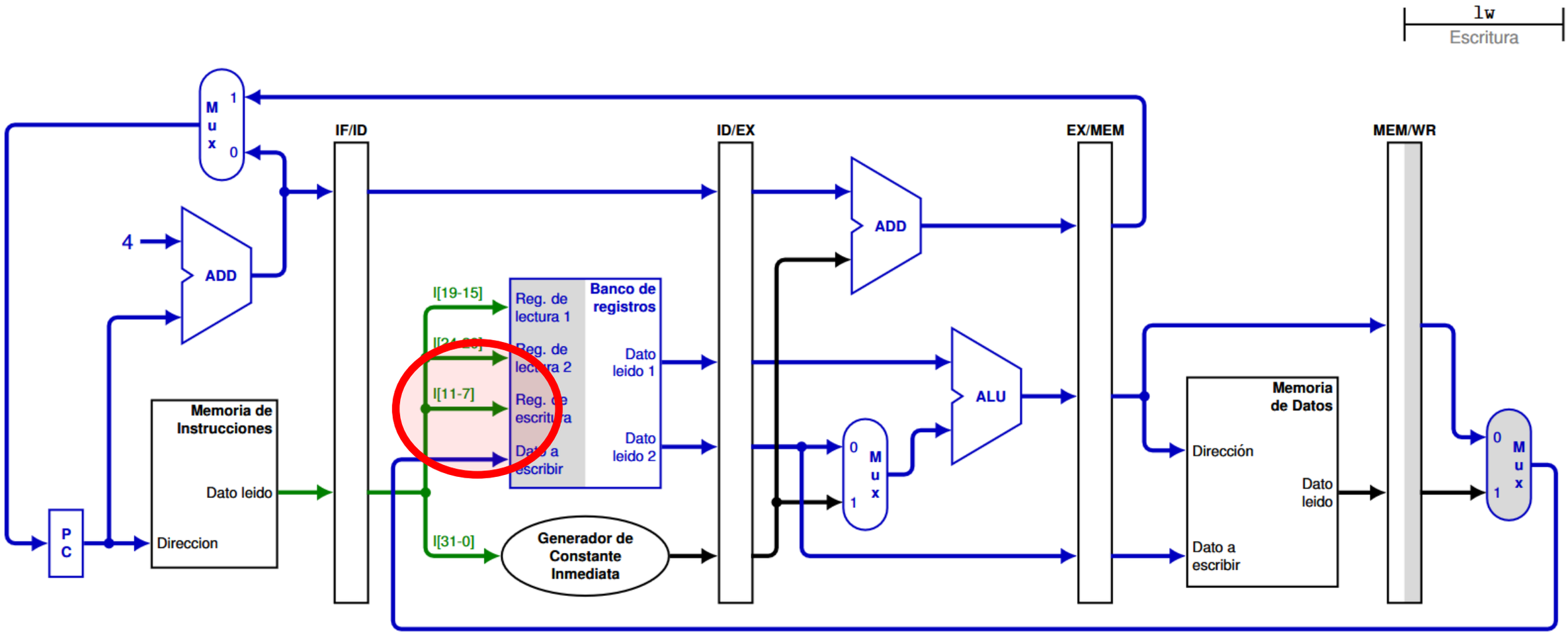
Ejemplo Ejecución lw: Etapa 3 (EX)



Ejemplo Ejecución lw: Etapa 4 (ME)

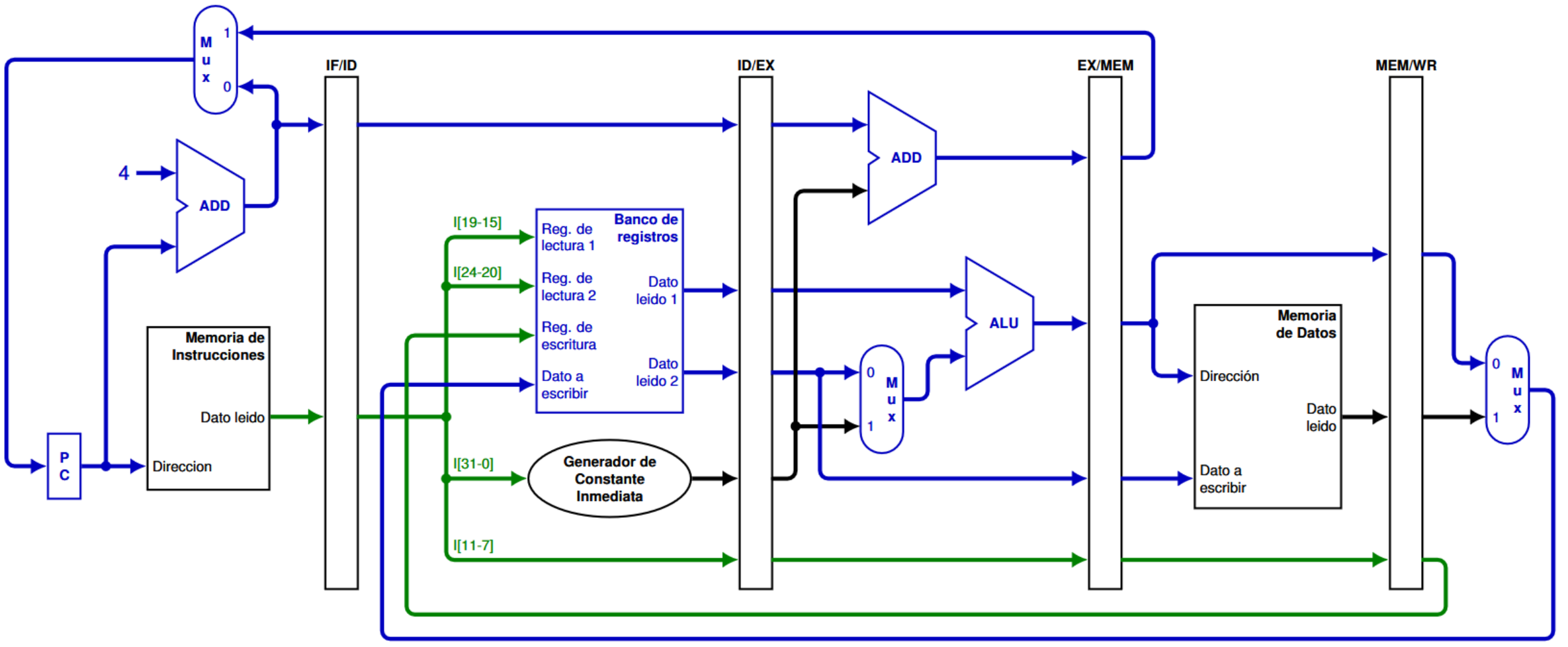


Ejemplo Ejecución lw: Etapa 5 (WB)



- ¡Error! El registro destino no proviene del registro MEM/WB, sino de IF/ID.
  - El registro destino será de otra instrucción.

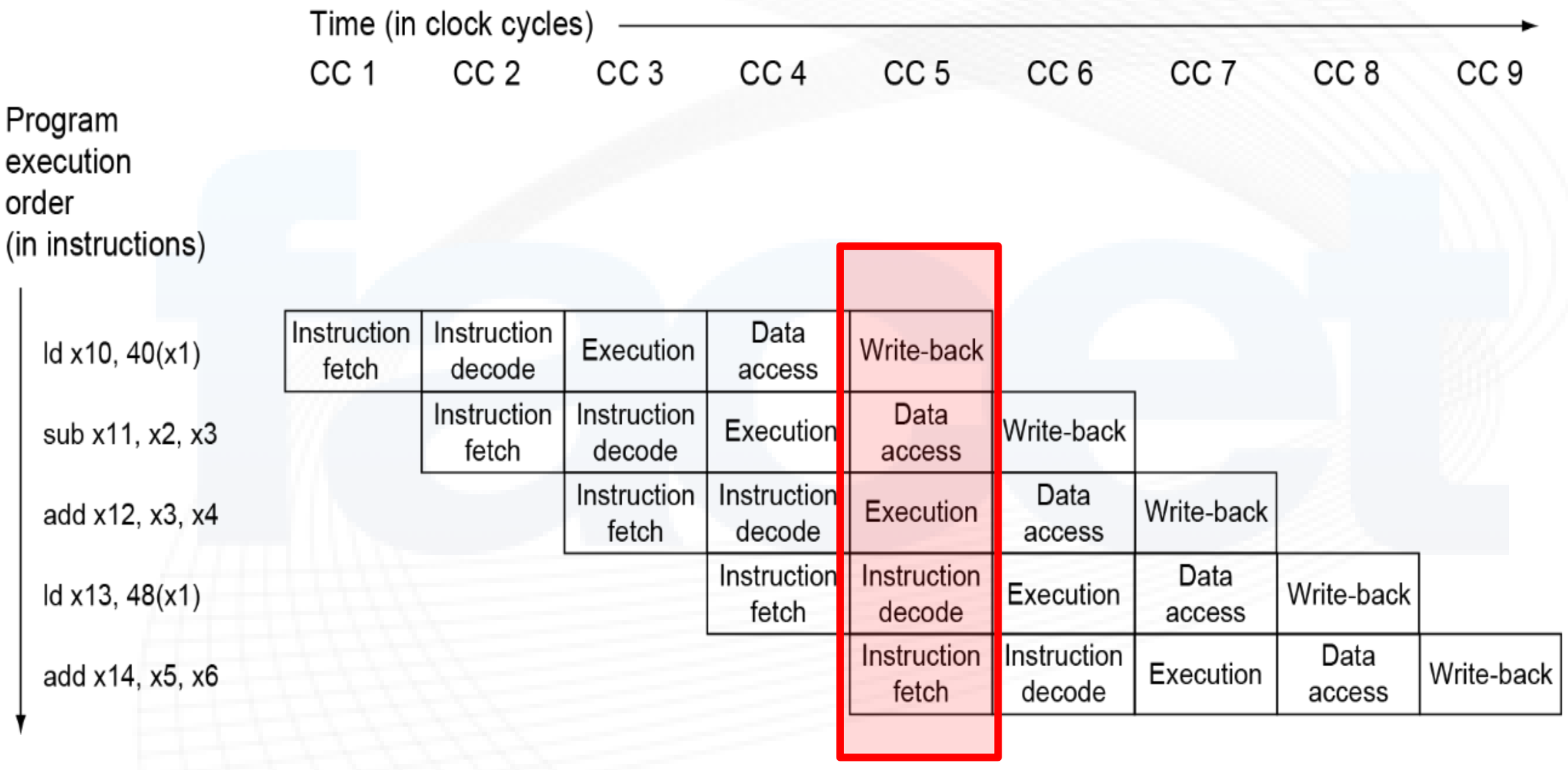
Ejemplo Ejecución lw: Etapa 5 (WB)



- Es necesario **propagar** el registro destino desde donde se genera (etapa 2) hasta donde se lo usa (etapa 5).
  - Pasando obviamente por todos los registros intermedios.

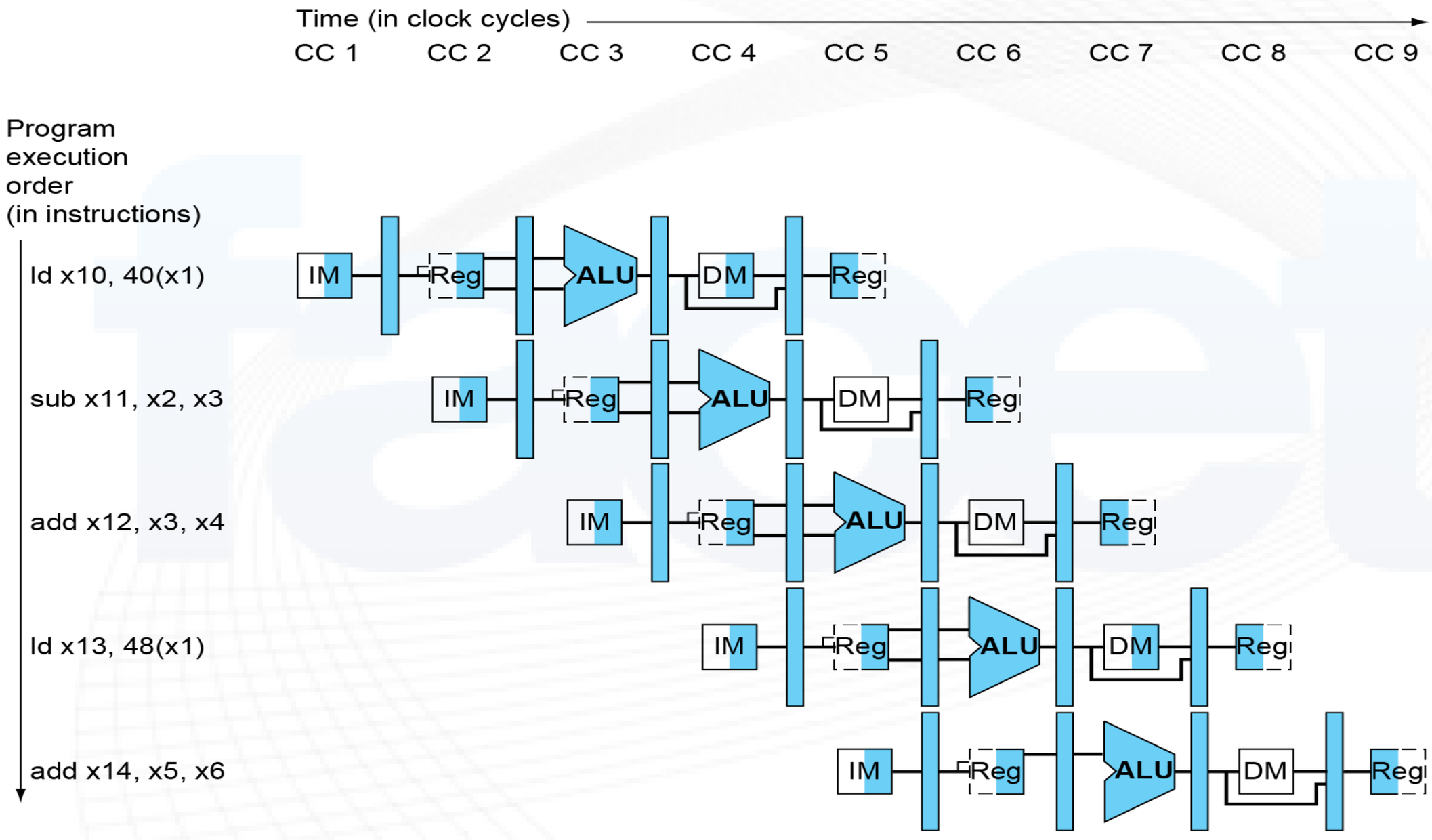
Visualización del pipeline

- Forma tradicional



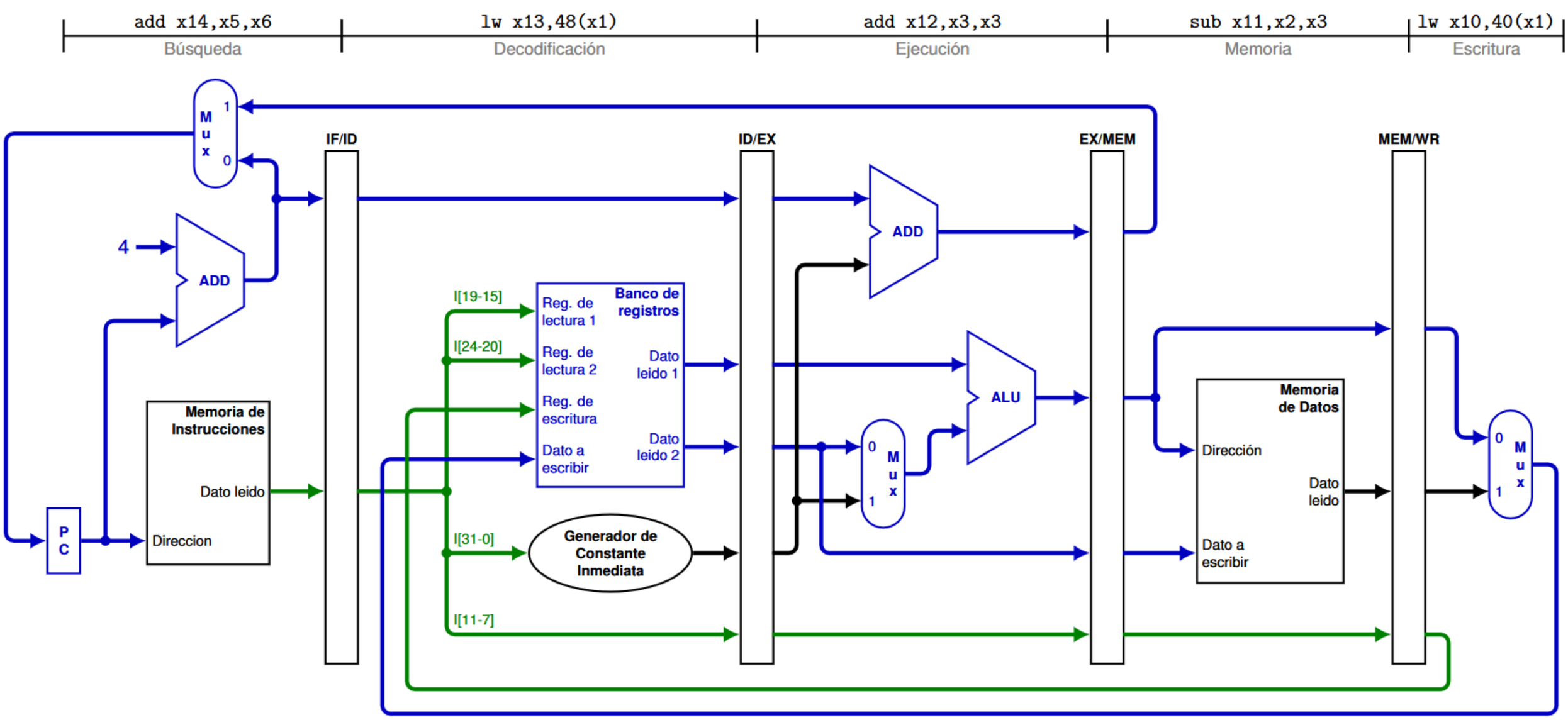
Visualización del pipeline

- Uso de recursos

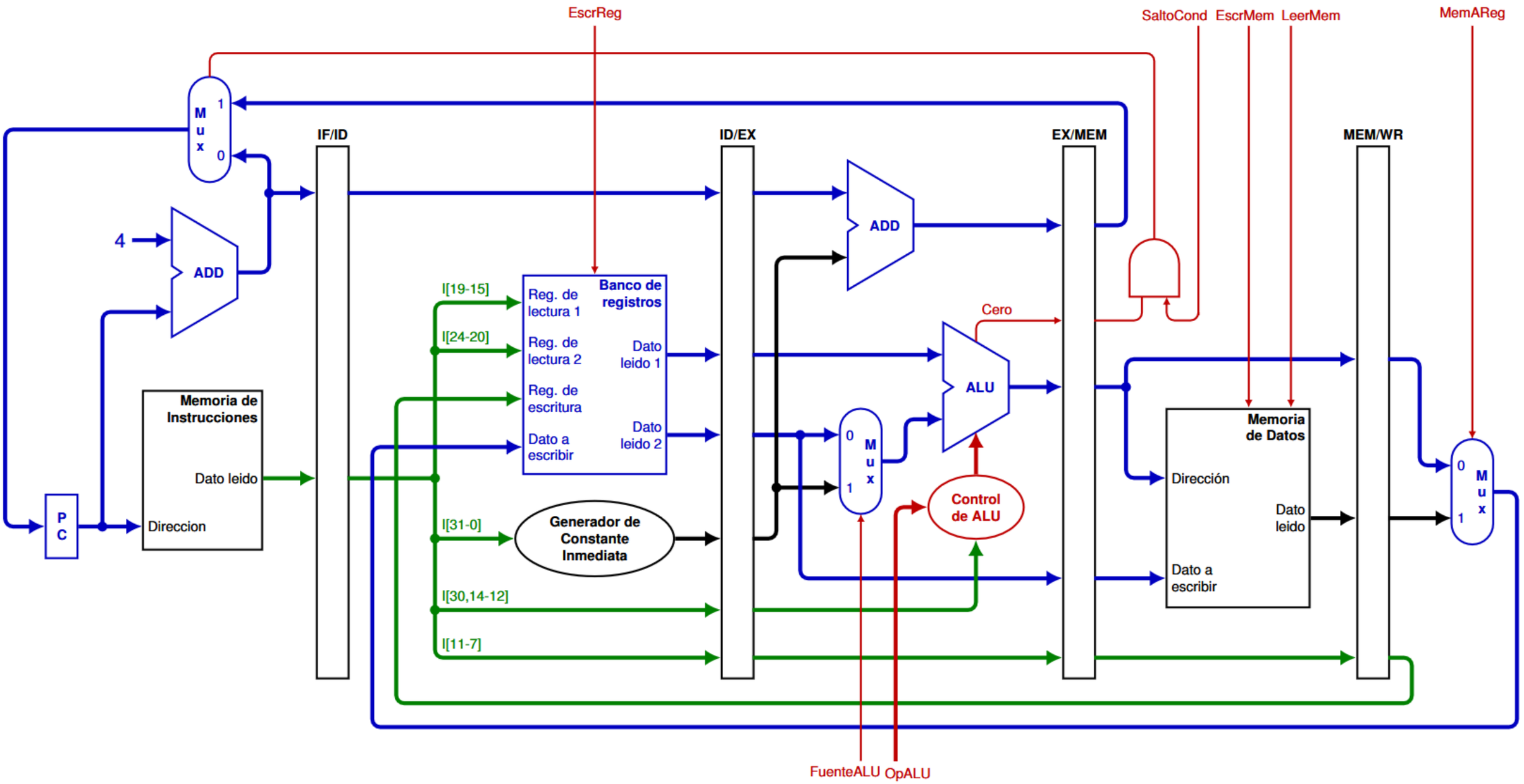


Visualización del pipeline

- Instantánea de un ciclo en particular.
- ¡Hay cinco instrucciones ejecutándose simultáneamente!



Señales de control en pipelining



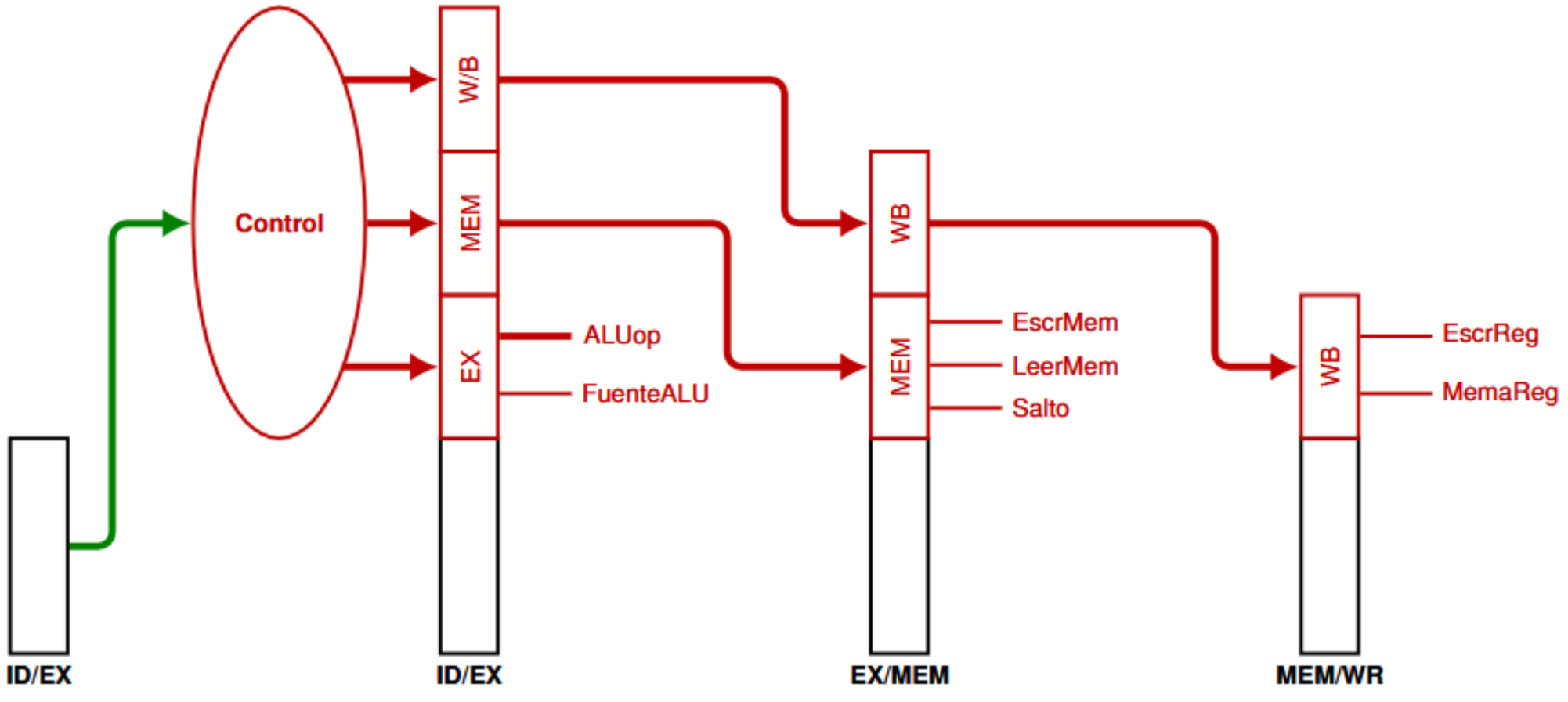
- Son las mismas siete que en el diseño de ciclo único.

Unidad de Control en Pipeline

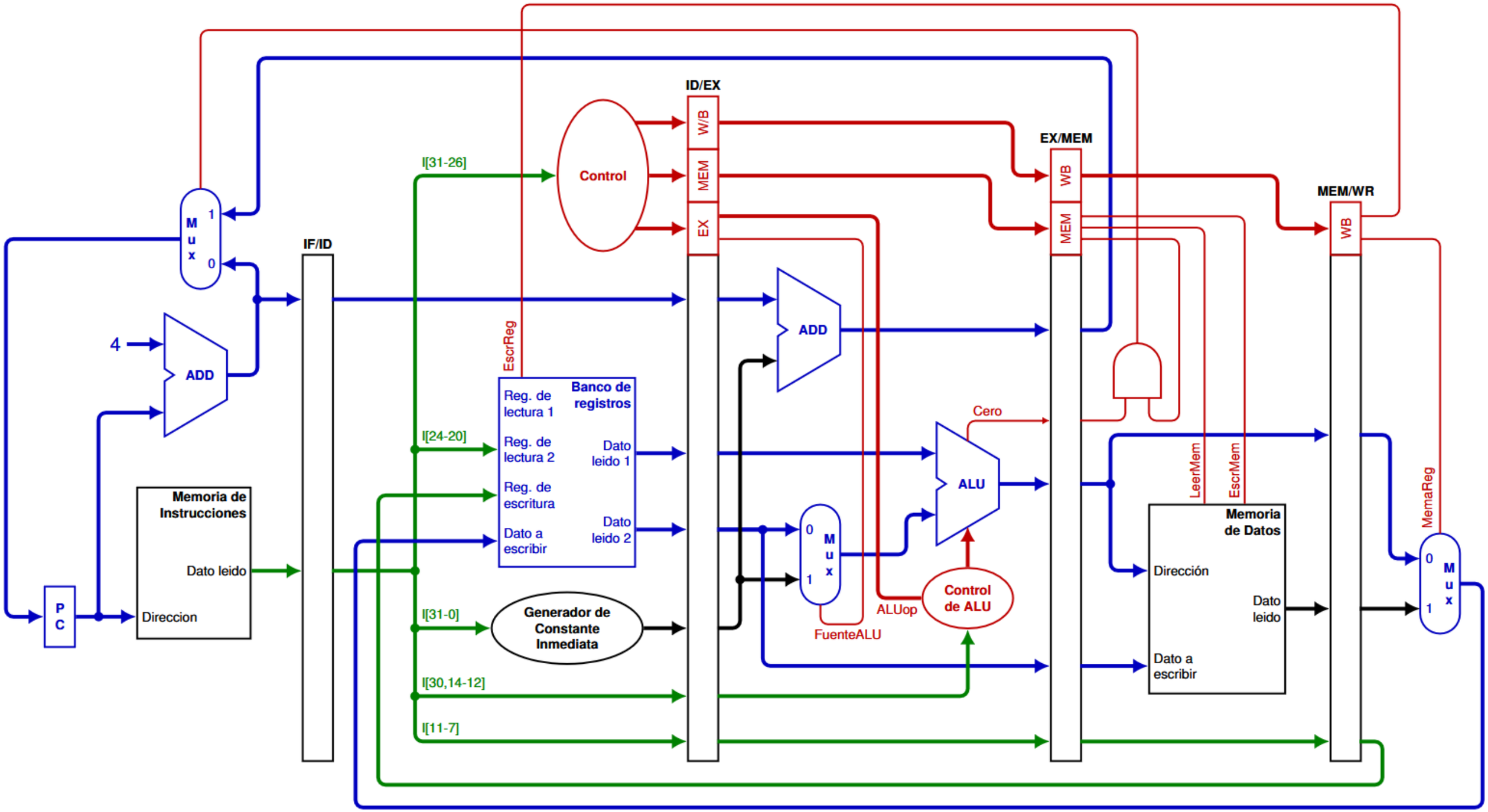
- **¡La Unidad de Control es la misma que la del ciclo único!**
  - Se generan las señales de control durante la etapa 2 (ID).
  - Pero hay un problema:
    - Algunas de las señales se usan en la etapa 3, otras en la etapa 4 y otras en la 5.

Unidad de Control en Pipeline

- Solución: que las señales de control también se escriban en los registros intermedios, y **se propaguen junto con la instrucción**.
  - Igual que se hizo con el registro destino.



Camino de datos en pipeline completo



Performance pipelining vs ciclo único

- Suponiendo los mismos retardos vistos en el procesador de ciclo único.
  - Memorias, ALU y sumadores 200 ps, Banco de registros 100 ps, todo lo demás despreciable.
  - $T_{uniciclo}$ = 800 ps; $T_{pipeline}$ = 200 ps
- Suponiendo que ejecutamos 3 LW seguidos:
  - *¿Cuánto demoran en ciclo único? ¿Cuánto en pipelining? ¿Cuánto es la aceleración?*
- Suponiendo que ejecutamos muchísimos LW seguidos:
  - *¿Cuánto demoran en ciclo único? ¿Cuánto en pipelining? ¿Cuánto es la aceleración?*

Performance pipelining vs ciclo único

- En estado de régimen, se completa una instrucción en cada ciclo.
  - **CPI = 1**, igual que la implementación de ciclo único.
  - Pero **con un tiempo de ciclo más corto**, igual que en la implementación multiciclo.
- Sin embargo, la aceleración no es igual a la cantidad de etapas.
- También aumenta la latencia de las instrucciones.
  - No es relevante, ya que nos interesa mejorar la productividad.
- Es poco realista tener todas instrucciones lw.
  - No representa un problema si todas las instrucciones pasan siempre por todas las etapas.

Riesgos en el pipeline

- Hasta aquí hicimos una suposición implícita:
  - En cada ciclo se puede ingresar una nueva instrucción al pipeline.
- Los problemas que pueden ocurrir se llaman **riesgos** (*hazards*), y pueden ser de tres tipos:
  - **Estructurales:** un recurso requerido está siendo ocupado por otra instrucción.
  - **De Datos:** hay que esperar que una instrucción previa se complete (dependencias).
  - **De Control:** la decisión de qué acción tomar depende de una instrucción previa (saltos).

Riesgos en el pipeline

- A los riesgos se intentará evitarlos.
- Si no se pueden evitar, se intentará resolverlos.
- Si no se pueden solucionar, entonces hay que **parar el pipeline (*stall*)** hasta que se resuelva el riesgo.
  - Equivale a insertar una o más “burbujas” o “huecos” en el pipeline.
  - Es una solución sencilla, pero que baja la performance, porque son ciclos desperdiciados.
- Los riesgos modifican el CPI:
  - **$CPI_{pipeline} = CPI_{ideal}$ + paradas por riesgos**
  - **paradas por riesgos = estructurales + de datos + de control**

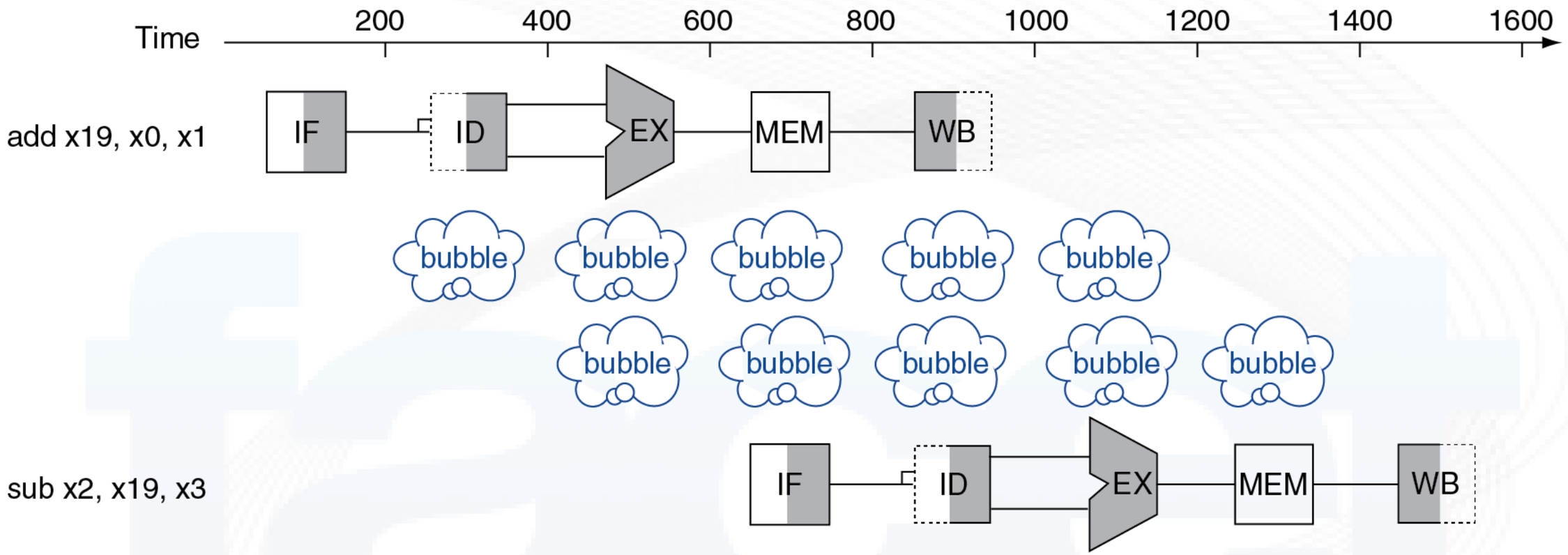
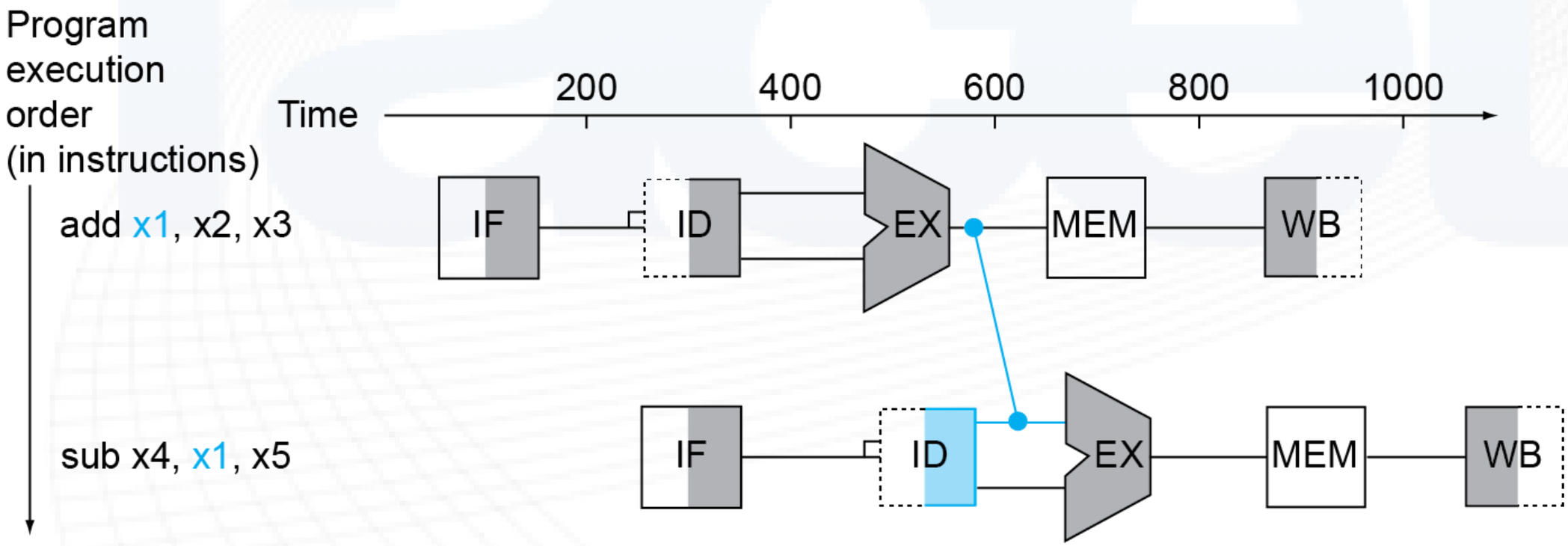
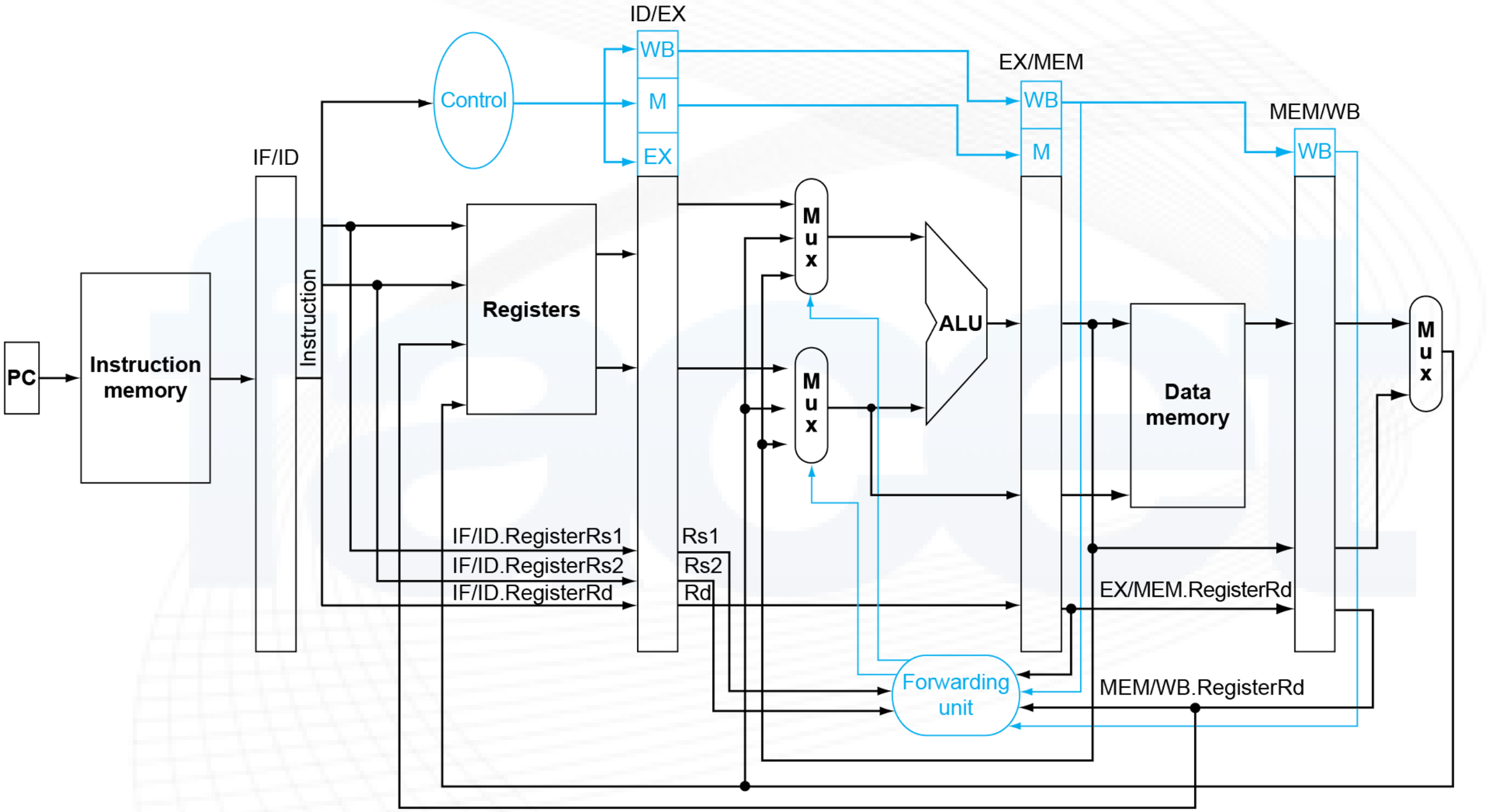
Riesgos Estructurales

- Ocurren cuando dos instrucciones, en dos etapas distintas, quieren utilizar el mismo recurso en el mismo instante.
  - Por ejemplo, si hubiera una única memoria, se intentaría acceder a la misma en IF y en ME.
  - O si las instrucciones Tipo R no pasaran por todas las etapas, podrían intentar escribir en el Banco de Registros al mismo tiempo que una instrucción lw.

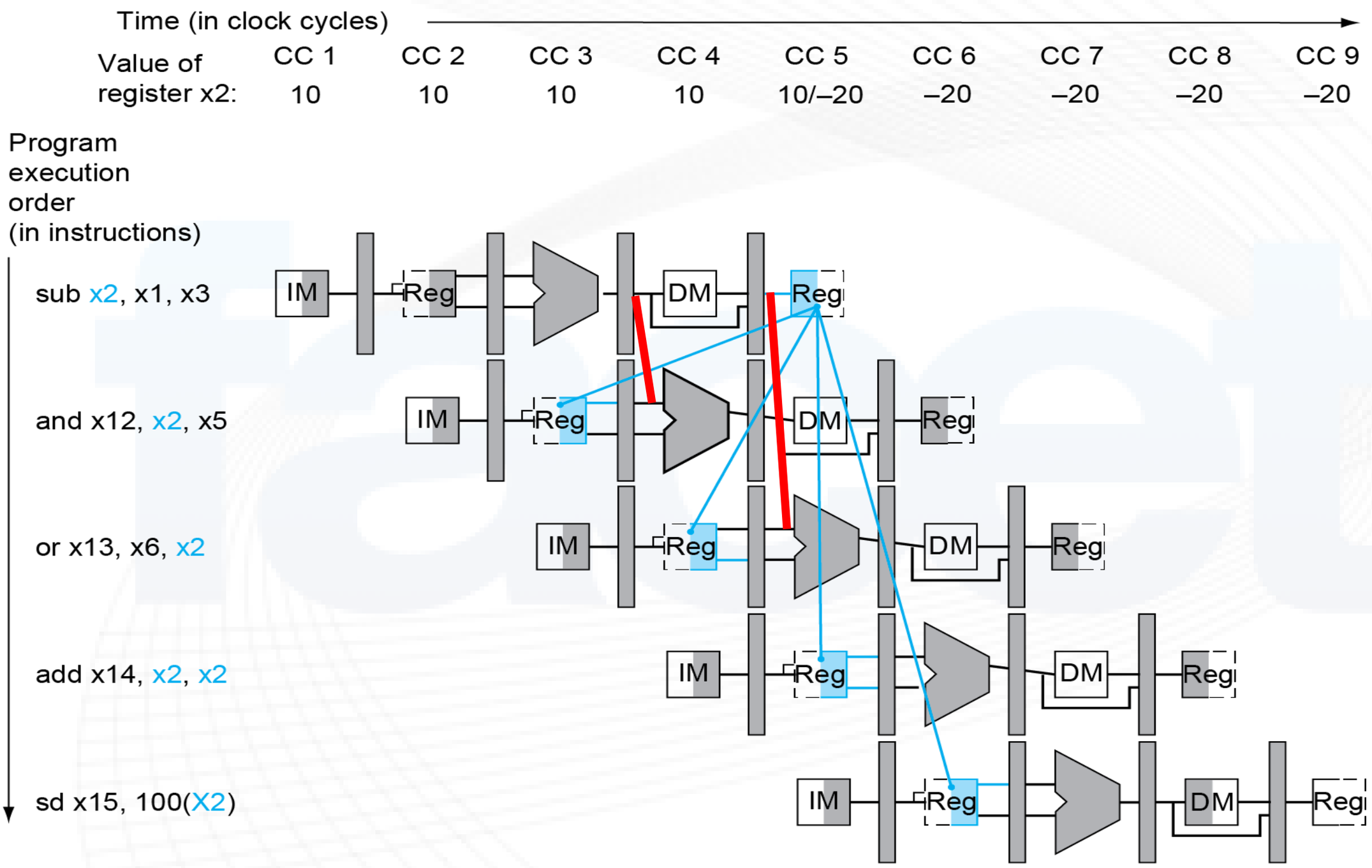
Riesgos Estructurales en RISC-V

- Nuestro diseño **no posee riesgos estructurales**.
  - Memorias separadas y recursos duplicados.
    - Aumentan el costo, pero mejoran la performance.
  - Todas las instrucciones pasan por todas las etapas.
    - Sólo se utiliza una unidad funcional por etapa.
    - Todas las instrucciones usan la misma unidad funcional en la misma etapa.
- Entonces:
  - **paradas por riesgos = de datos + de control**

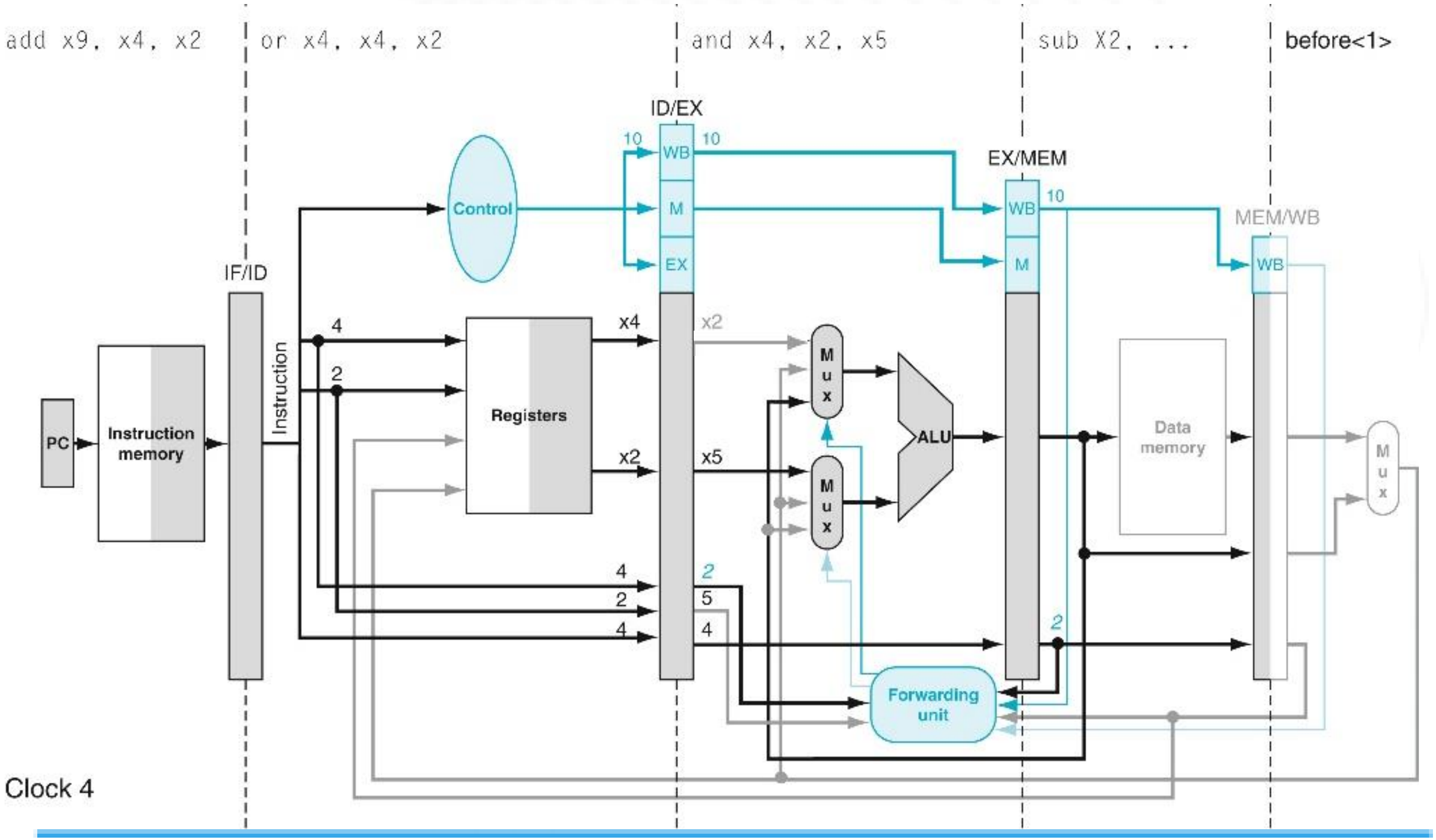
Riesgos de Datos

- Una instrucción depende del resultado de una instrucción previa (dependencia tipo RAW) que aún está dentro del pipeline.
  - ```
    add     x19, x0, x1
    sub     x2, x19, x3
    ```
  - La instrucción add escribe el registro x19 en la etapa 5 (WB).
  - La instrucción sub lee el registro x19 en la etapa 2 (ID).
  - Para que no ocurran errores, debemos parar el pipeline durante dos ciclos.
    - O lo que es lo mismo, insertar dos burbujas, para que ocupen las etapas 3 y 4.
    ```

Riesgos de Datos



- Son sólo dos burbujas, porque recordemos que el Banco de registros escribe en la primera mitad del ciclo.
  - Así que primero escribe el valor deseado, y luego lo lee.

Riesgos de Datos – Adelantamiento

- En realidad, no es necesario esperar que el registro se escriba en la etapa 5.
  - ¡El resultado ya está disponible en la etapa 3!
  - Entonces podemos “adelantarlo” desde el registro intermedio donde está guardado, hasta donde se lo necesite usar.
  - Esta técnica se conoce como **Adelantamiento** (*Forwarding* o *Bypassing)*.



Riesgos de Datos – Adelantamiento

- Usar adelantamiento requiere agregar muchas interconexiones nuevas, más nuevos multiplexores.
  - Porque ahora un dato puede provenir de más de un registro intermedio.
- También implica incorporar una nueva **Unidad de Adelantamiento** (*Forwarding Unit*).
  - Encargada de detectar la ocurrencia del riesgo, y generar las señales de control necesarias para los nuevos multiplexores.
- Agrega complejidad y costo al diseño.

Riesgos de Datos – Adelantamiento

- Para **detectar el riesgo** es necesario:
  - Comparar si alguno de los dos registros fuente de la instrucción en la etapa 2 (ID) es igual al registro destino de la instrucción en la etapa 3 (EX) o en la etapa 4 (MEM).
  - Y si es que las instrucciones en la etapa 3 o 4 escriben en un registro (EscrReg).
    - Y sólo si el registro destino es distinto de cero.
- Si se detecta un riesgo:
  - Adelantar el o los datos necesarios desde la etapa MEM, o desde la etapa WB, o desde ambas.
- Si ocurre un riesgo doble, se adelanta el más reciente.

Camino de datos con Adelantamiento



Riesgos de Datos – Adelantamiento

- No todas las dependencias constituyen un riesgo de datos.
  - ```
    sub  x2, x1,x3
    and  x12,x2,x5
    or   x13,x6,x2
    add  x14,x2,x2
    sd   x15,100(x2)
    ```

- Sólo and y or constituyen realmente un riesgo.
  - Porque cuando add lee x2 en la etapa 2 (ID), sub ya se encuentra en la etapa 5 (WB).
- Ambos se pueden solucionar por adelantamiento.
    ```

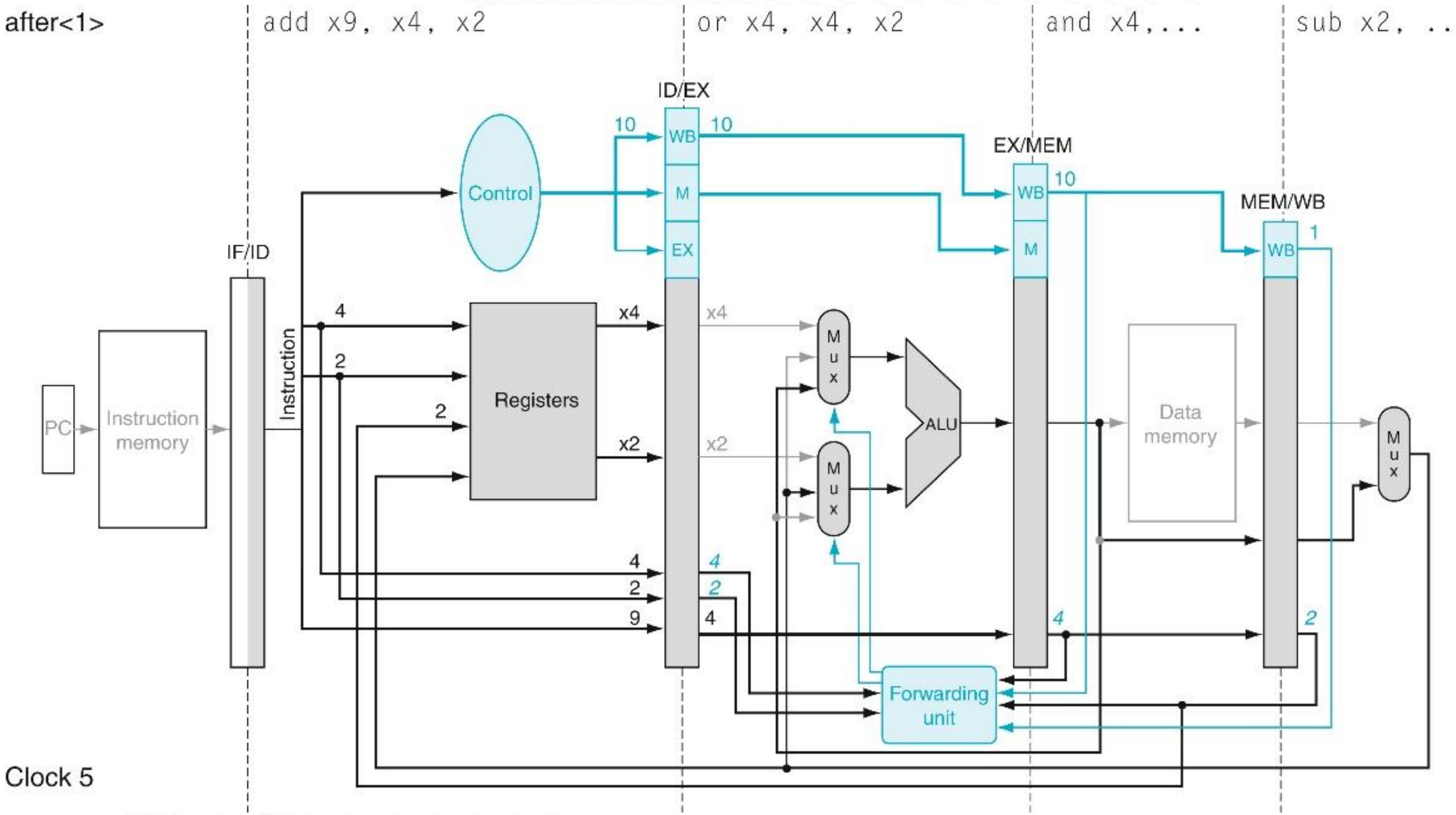
Ejemplo de Adelantamiento



Ejemplo de Adelantamiento

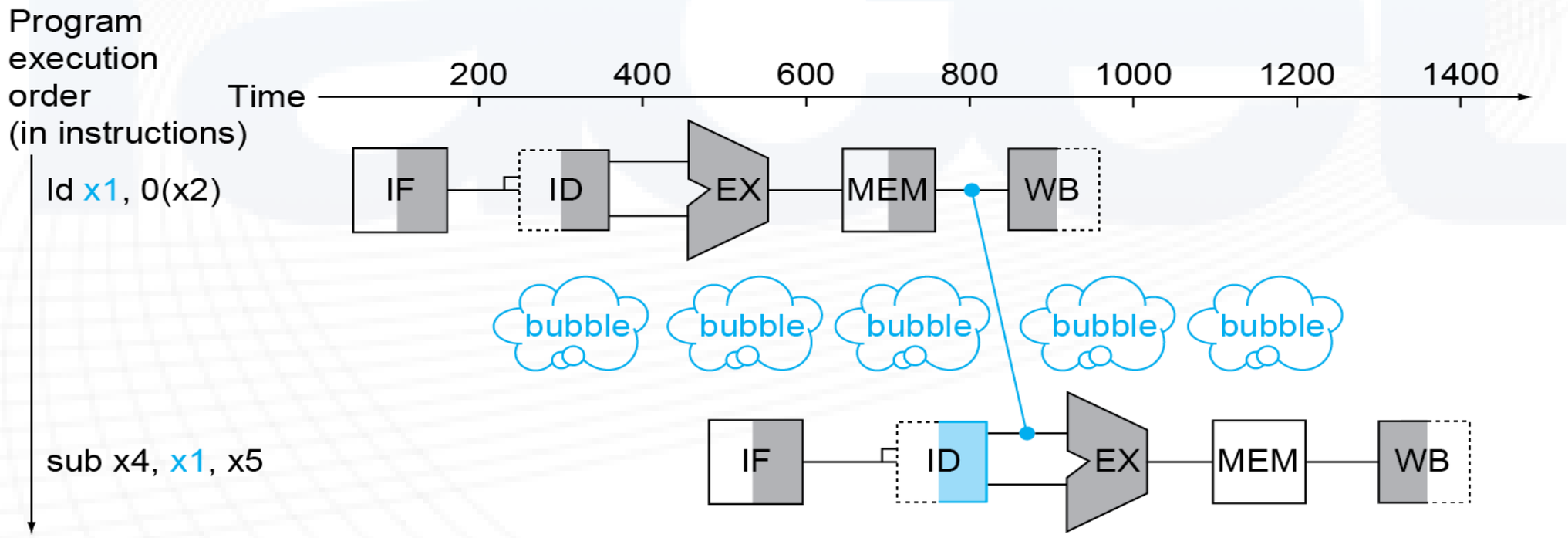


Ejemplo de Adelantamiento



Riesgos de Datos – Problema con lw

- A pesar de que agrega complejidad y costo al diseño, es muy necesario.
- Porque soluciona casi todos los riesgos de datos.
  - *¿Casi?*
- Si el dato no está disponible cuando es necesitado, no hay manera de adelantarlo.
- Esto sucede cuando una instrucción lw es seguida inmediatamente por una instrucción que use el registro destino del lw.



Riesgos de Datos – Reordenamiento

- La solución más simple para evitar el problema con la dependencia de lw, es **cambiar el orden de las instrucciones** en el código.

```
lw    x1, 0(x0)
lw    x2, 8(x0)
add   x3, x1, x2
sw    x3, 24(x0)
lw    x4, 16(x0)
add   x5, x1, x4
sw    x5, 32(x0)
```

```
lw    x1, 0(x0)
lw    x2, 8(x0)
lw    x4, 16(x0)
add   x3, x1, x2
sw    x3, 24(x0)
add   x5, x1, x4
sw    x5, 32(x0)
```

- *¿Cuánto demoran en ejecutarse los códigos?*

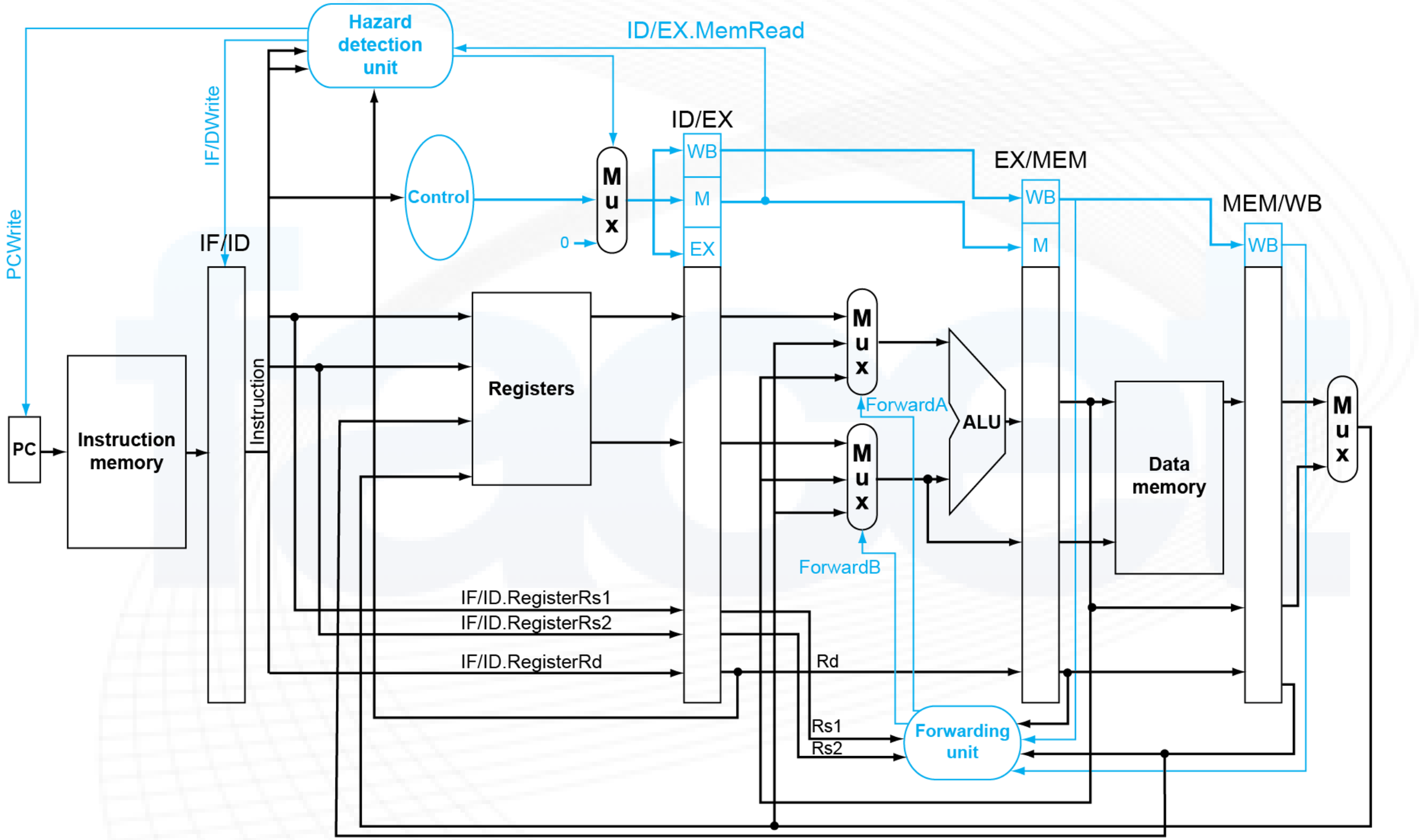
Riesgo de Datos – Paradas

- La Unidad de Adelantamiento puede detectar y solucionar todos los riesgos de datos, menos el de la dependencia del lw.
- Este riesgo de datos podría evitarse mediante Reordenamiento de las instrucciones.
- Pero si no es posible reordenar, para garantizar el correcto funcionamiento habrá que parar el pipeline.
  - Se lo conoce como ***Load-to-use Penalty***.
- En nuestra implementación actual, esta penalidad implica agregar una sola burbuja.
  - En otras implementaciones puede ser mayor.
- Para insertar una burbuja en el pipeline se agrega al camino de datos una **Unidad de Inserción de Burbuja** (*Hazard Detection Unit*).

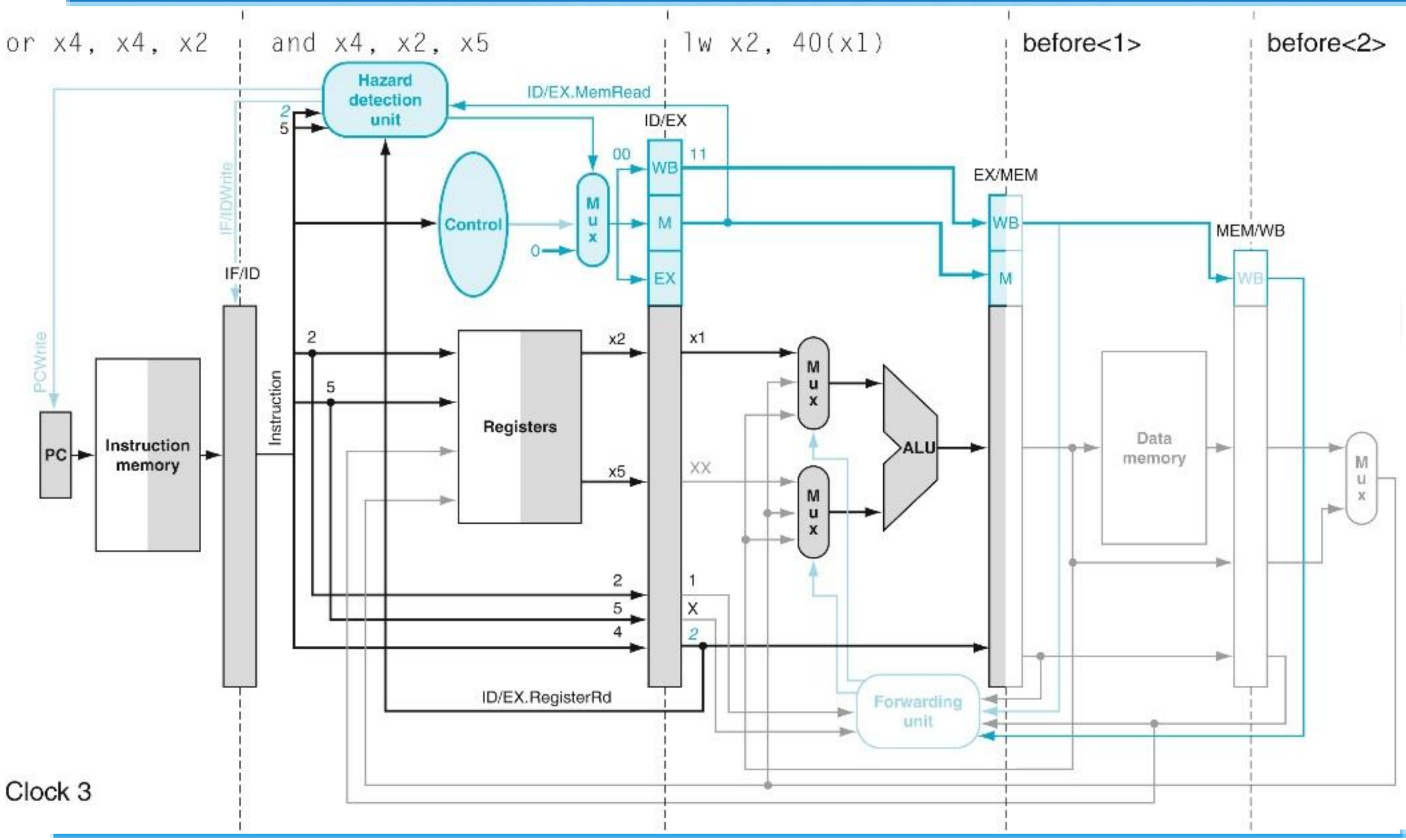
Riesgo de Datos – Paradas

- La Unidad de Inserción de Burbuja primero debe detectar el riesgo:
  - Controlar que en la etapa 3 (EX) haya una instrucción lw (mediante la señal de control LeerMem).
  - Controlar que el registro destino de la instrucción en la etapa 3 (EX) coincide con alguno de los registros fuente de la instrucción en la etapa 2 (ID).
- Si se detecta el riesgo, una burbuja se inserta realizando dos acciones:
  - Evitar la actualización de PC y del registro IF/ID mediante una nueva señal de control.
    - Para que se mantengan las instrucciones en las etapas IF e ID.
  - Poner todas las señales de control del registro ID/EX en cero.
    - Mediante un nuevo multiplexor.
    - Para que en las etapas EX, MEM y WB posteriores no se haga nada.

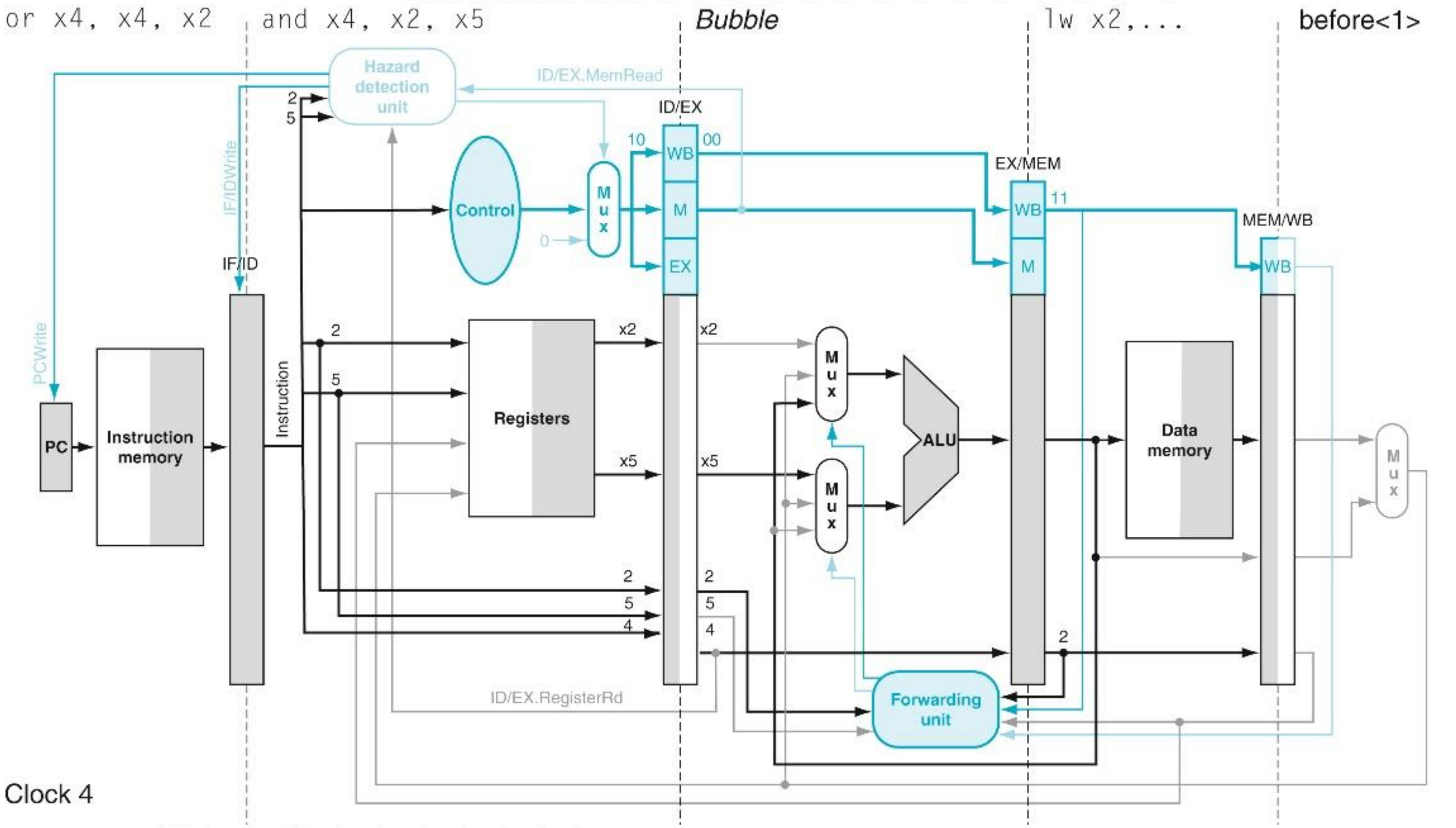
Pipeline con inserción de burbujas



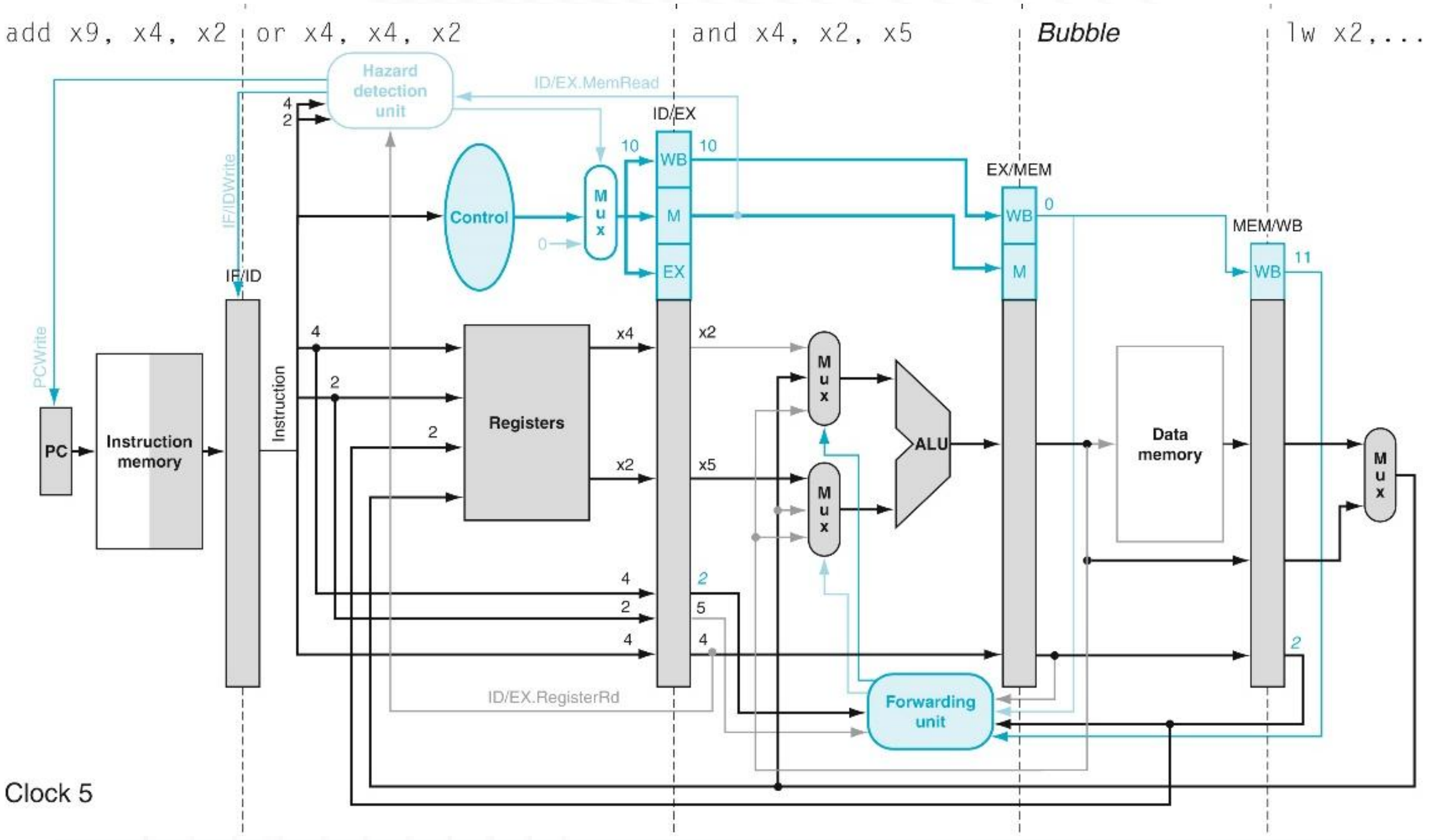
Ejemplo de Inserción de burbujas



Ejemplo de Inserción de burbujas



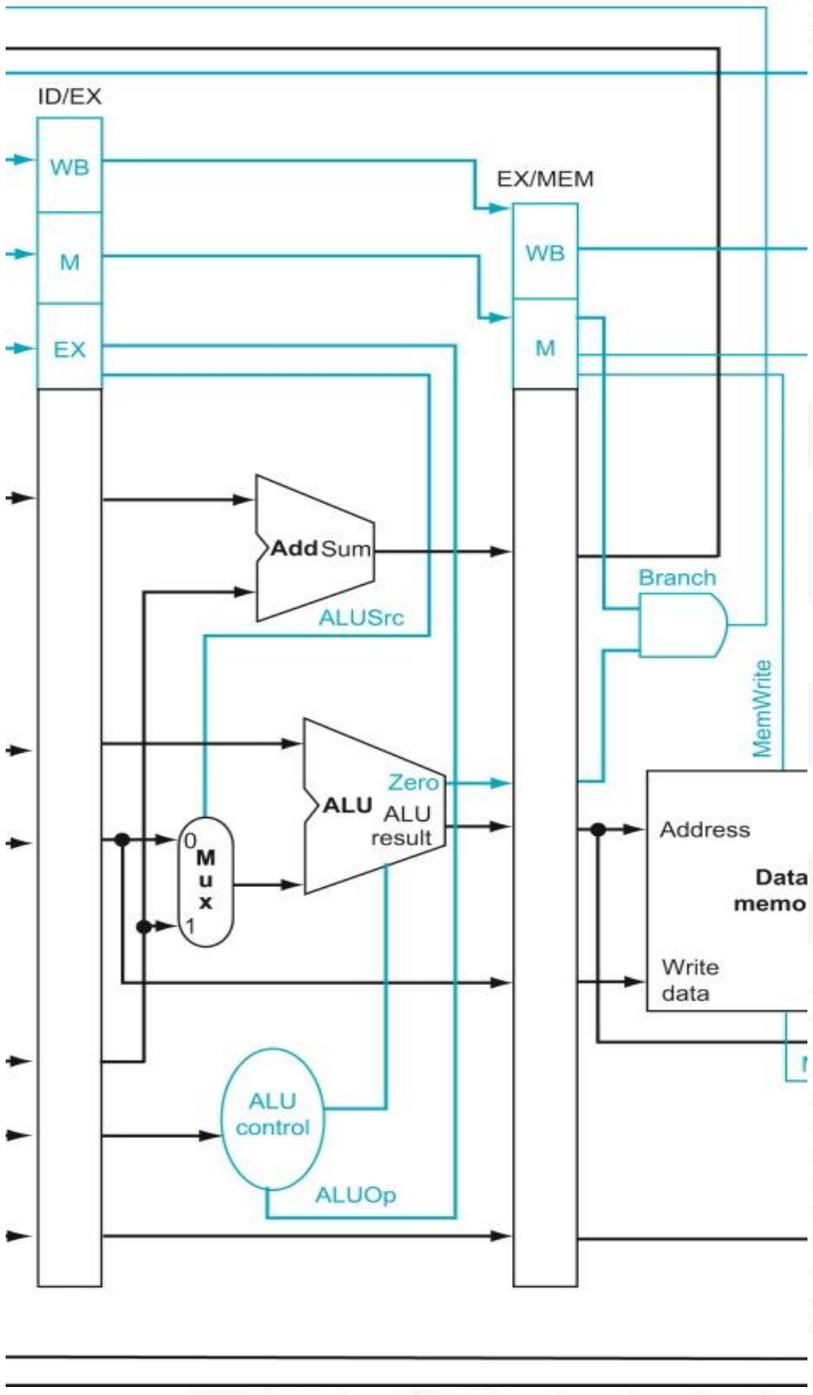
Ejemplo de Inserción de burbujas



Riesgos de Control

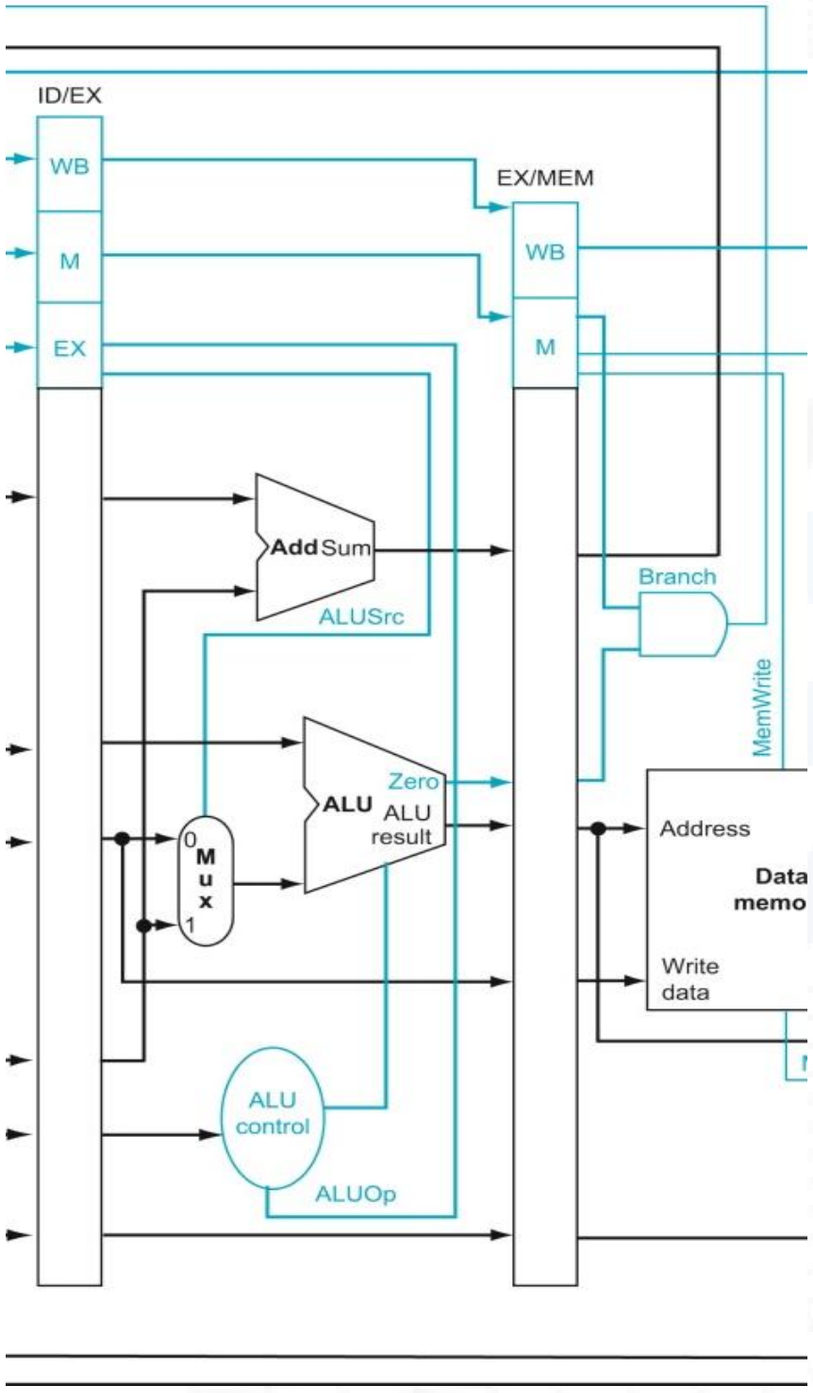
- Los saltos determinan el flujo de control de un programa.
  - La búsqueda de la siguiente instrucción depende de la resolución de un salto.
- La resolución de un salto implica dos tareas:
  - Determinar si realmente se efectuará o no.
  - Determinar cuál es la dirección destino del salto.
- En un pipeline, no siempre se puede hacer el fetch de la instrucción correcta.
  - En nuestra implementación, los saltos se resuelven completamente en la etapa 4 (MEM).
  - En el ciclo siguiente, se buscará la instrucción correcta en la etapa 1 (IF) y el salto estará en la etapa 5 (WB).
  - ¡Esto implica que habría que agregar tres burbujas por cada salto!

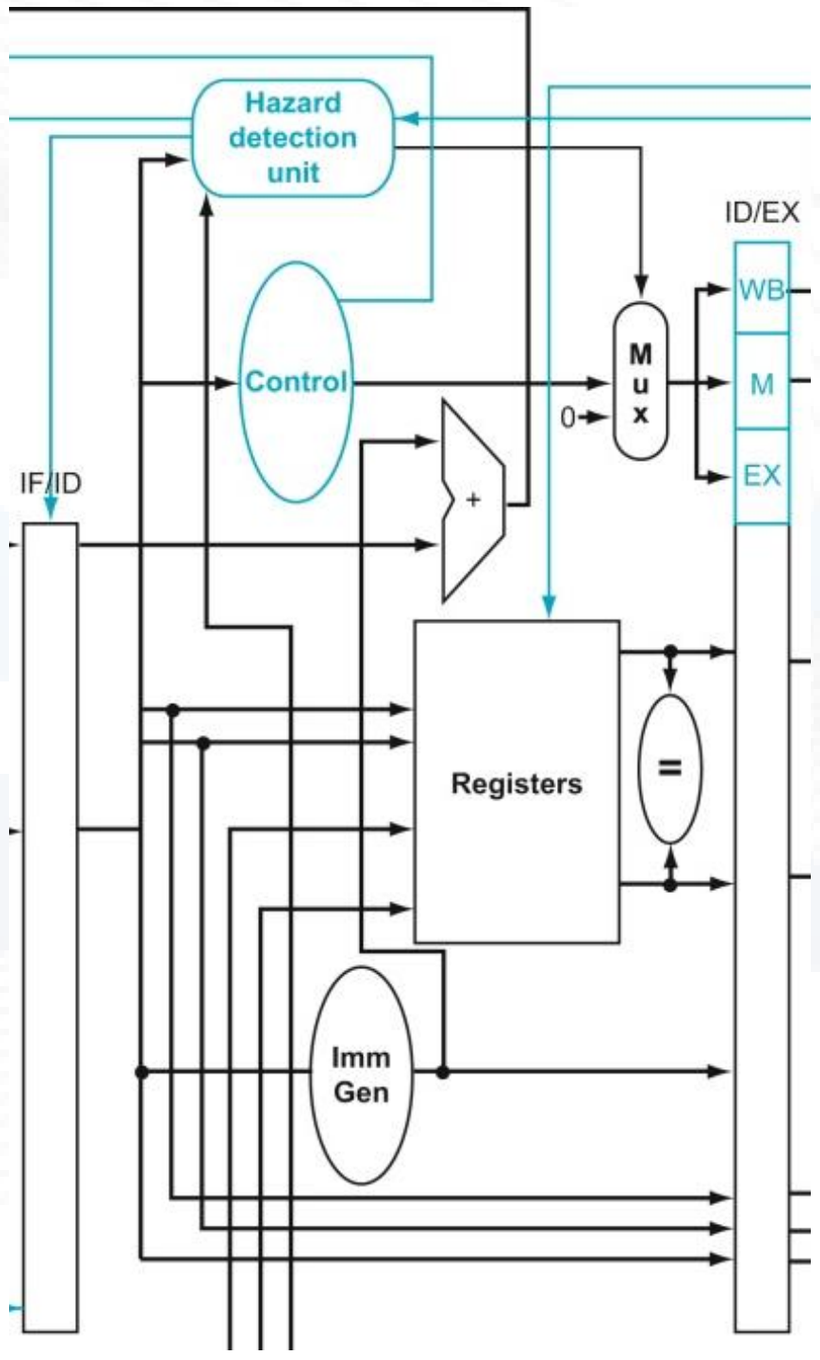
Riesgos de Control – Primera alternativa



- Para determinar si el salto se efectuará o no, en la etapa EX la ALU compara los registros.
  - Podemos agregar un comparador a la salida del Banco de Registros para realizar esta tarea.
  - Liberamos la ALU, moviendo la determinación del salto a la etapa 2 (ID).
- Para calcular la dirección de salto, en la etapa EX hay un sumador.
  - Sus datos ya están disponibles en la etapa 2 (ID), así que podemos cambiarlo de lugar.
  - No modifica la duración de la etapa, porque no depende del acceso al Banco de Registros.
- **Se puede resolver todo el salto en la etapa 2 (ID).**

Riesgos de Control – Primera alternativa





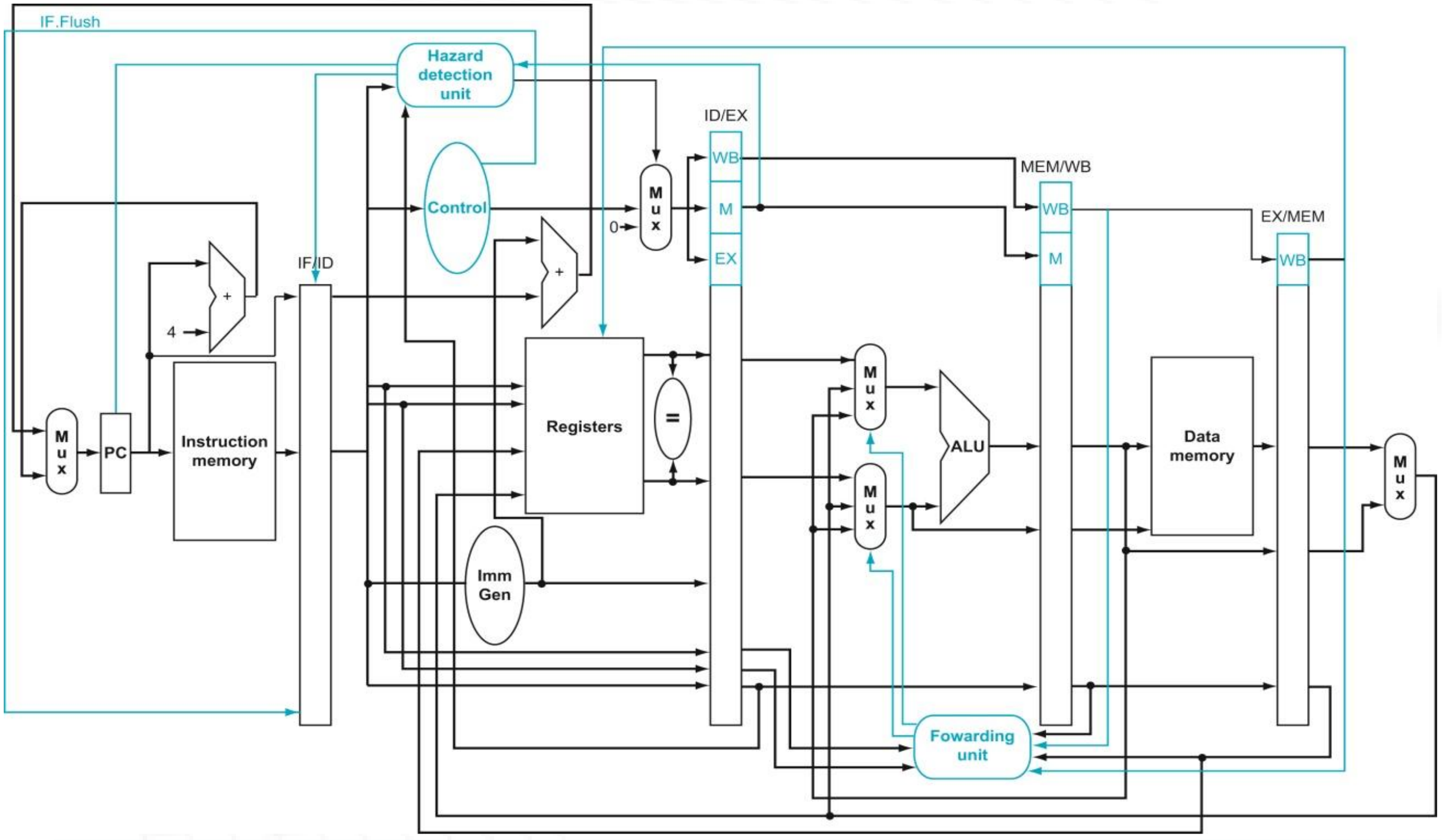
Riesgos de Control – Primera alternativa

- Para que esta alternativa funcione, la condición a evaluar en los saltos debe ser **simple**.
  - El diseño del ISA ayuda.
- Además, genera nuevos riesgos de datos, que necesitan nuevos caminos de adelantamiento.
  - En el caso que la instrucción previa al salto escriba en un registro que el salto lee.
  - Habrá que adelantar a la etapa 2 (ID) en vez de a las entradas de la ALU de la etapa 3 (EX).
- E inclusive, aún debemos insertar una burbuja.
  - Porque si el salto se resuelve completamente en la etapa 2 (ID), hay que descartar lo que esté en la etapa 1 (IF).
  - Esta burbuja equivale a “vaciar el pipeline”, y se la denomina *flush*, a diferencia de los *stalls* por datos.

Riesgos de Control – Segunda alternativa

- El problema de la alternativa anterior, es que en pipelines más largos, es muy difícil adelantar tanto toda la resolución del salto.
  - Por lo tanto, la cantidad de burbujas a insertar puede hacerse grande.
- Idealmente, deberíamos intentar que toda la resolución del salto sea en la etapa 1 (IF).
  - Es ilógico, porque ni siquiera se sabe si la instrucción que se busca es un salto.
- Entonces se usa **predicción de saltos**.
  - Gran Idea en Arquitectura de Computadoras.
  - Intentar adivinar si el salto se efectúa o no, y solamente vaciar el pipeline cuando la predicción falla.
    - Se la conoce como ***Branch-misprediction Penalty***.
  - Más detalles… en el tema siguiente.

Camino de datos completo



Resumen final

- ¡Diseñamos un procesador en pipelining!
  - Cinco etapas independientes. Cada una utiliza una única unidad funcional.
  - Separadas por registros de segmentación.
  - Todas las instrucciones pasan por todas las etapas.
    - Ventajas: CPI = 1, como el uniciclo, pero con T pequeño.
    - Desventaja: más componentes, mayor área, mayor costo.
- El control es simple, combinacional.
  - Las señales de control se propagan junto con la instrucción.

Resumen final

- Problemas del pipelining: Riesgos
  - Solucionarlos parando el pipe, ¡nunca es una buena opción!
- Riesgos Estructurales resueltos por diseño.
- Riesgos de Datos, causados por dependencias.
  - Hay dos cuestiones: detectarlos y solucionarlos.
  - Solucionados por Adelantamiento, excepto un caso solucionado por Reordenamiento (si se puede).
    - Tratar de evitar la penalidad *load-to-use*.
- Riesgos de control
  - Intentamos minimizarlos adelantando la resolución del salto.
  - Intentamos evitarlos mediante predicción.
    - Si no se puede, se paga la penalidad por fallo en la predicción.