Tema 08: 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.

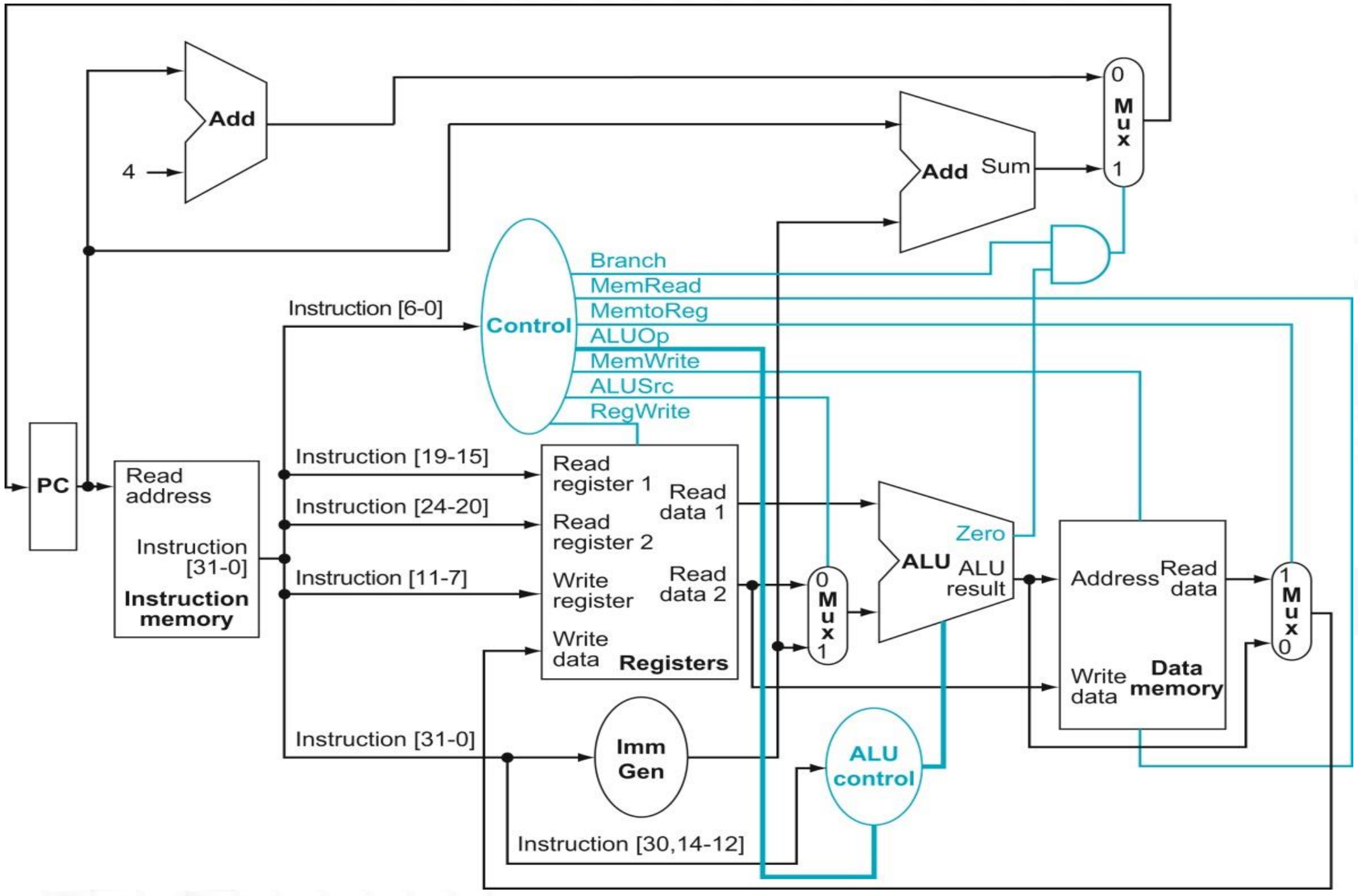
Lectura recomendada

- 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.
- **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



Secuencia de la instrucción LW

- Analizando el timing de la instrucción LW en el procesador de ciclo único, determinamos la siguiente secuencia:

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).
4. ME: Acceso a memoria.
5. WB: Escritura del resultado en el banco de registros.

- *¿Cómo serían las secuencias de las demás instrucciones?*

Etapas del procesador

- A cada paso de la diapositiva anterior lo ejecutamos en una etapa.
  - **Sólo se utiliza una única unidad funcional en cada etapa, independiente de las demás.**
  - Cada unidad funcional es utilizada una única vez por instrucción.
    - Quedan bastante balanceadas las etapas.
- Cada etapa se ejecutará en 1 ciclo.
  - **La duración del ciclo quedará determinada por la duración de la etapa más larga.**
- Nótese que el Banco de Registros se lee en la etapa 2 y se escribe en la etapa 5.
  - No es un problema, porque se usan puertos independientes.

Etapas del procesador

- La separación anterior en 5 etapas es posible gracias a que el ISA de RISC-V fue diseñado pensando en pipelining:
  - Las instrucciones pueden ser buscadas de memoria en un solo ciclo.
  - Las instrucciones pueden ser decodificadas en un solo ciclo.
  - Tipo Load/Store, con un único modo de direccionamiento (indexado).
    - Se calcula la dirección de memoria en la etapa 3.
    - Se accede a memoria en la etapa 4.

Performance pipelining vs ciclo único

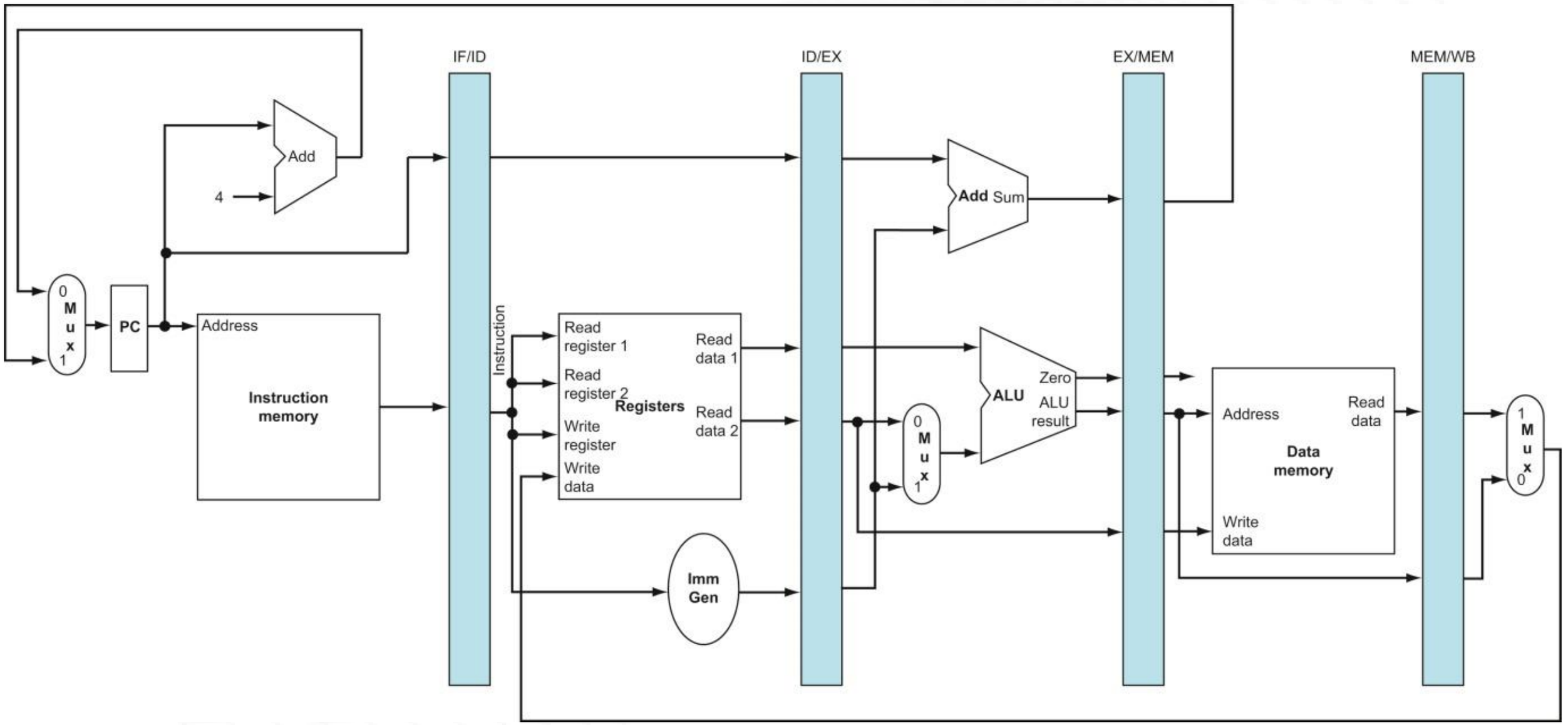
- Suponiendo los mismos retardos vistos en el procesador de ciclo único.
  - Memorias, ALU y sumadores 200 ps, Banco de registros 100 ps, todo lo demás despreciable.
  - LW en uniciclo = 800 ps = $T_{uniciclo}$
  - $T_{pipeline}$ = 200 ps
- Suponiendo que ejecutamos 3 LW seguidos:
  - *¿Cuánto demoran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

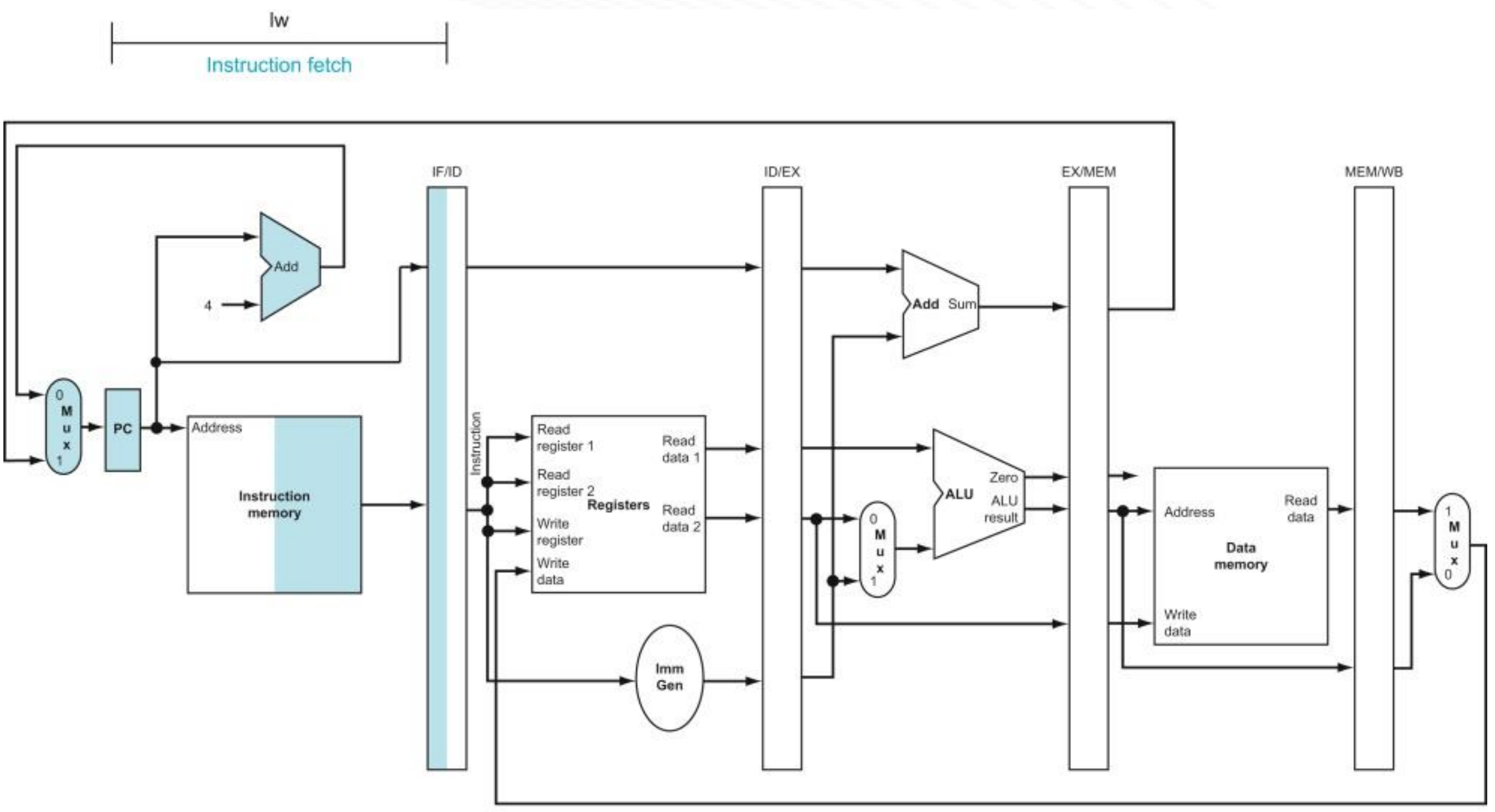
- En estado de régimen, **conseguimos que se termine una instrucción en cada ciclo**.
  - CPI = 1, con un tiempo de ciclo más corto.
- Pero la aceleración no es igual a la cantidad de etapas.
- Y la latencia de una instrucción aumentó.
  - No es relevante, ya que nos interesa mejorar la productividad.
- Y además es poco realista tener todas instrucciones LW.
  - No representa un problema si hacemos que todas las instrucciones pasen siempre por todas las etapas.

Camino de datos en pipelining

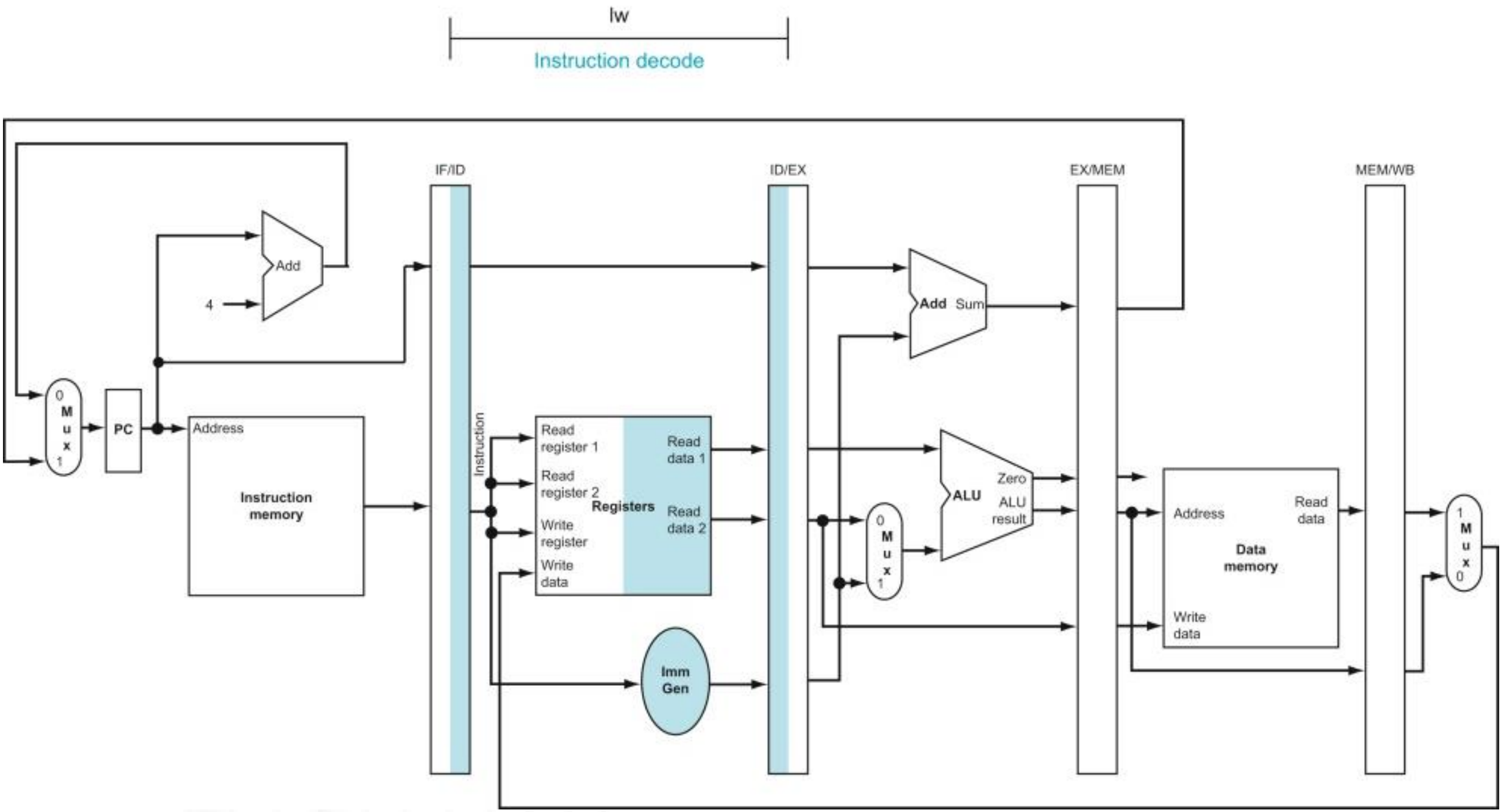
- Separamos las etapas mediante registros intermedios.
  - Mantienen la información producida en el ciclo previo.



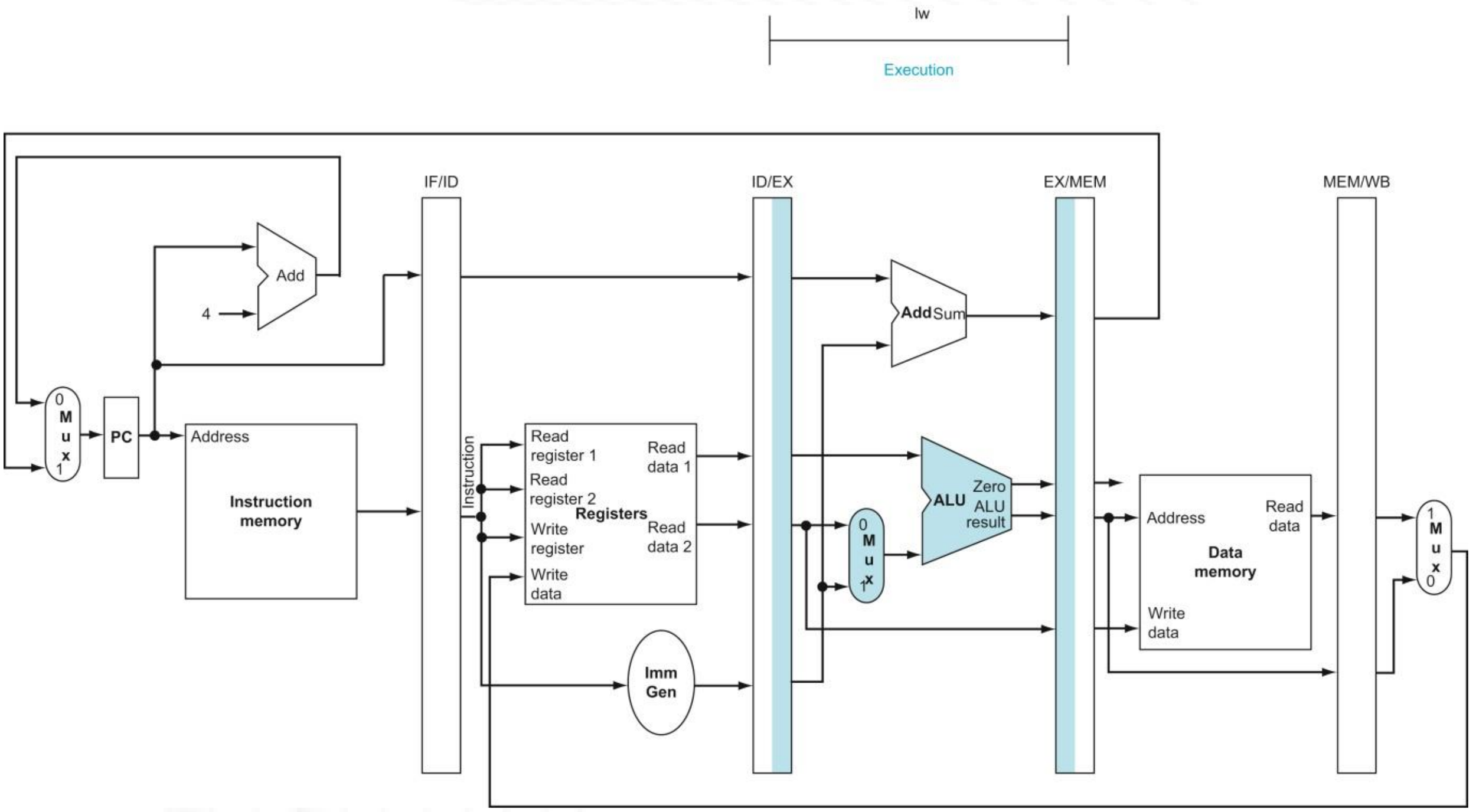
Ejemplo para LW: Etapa 1 (IF)



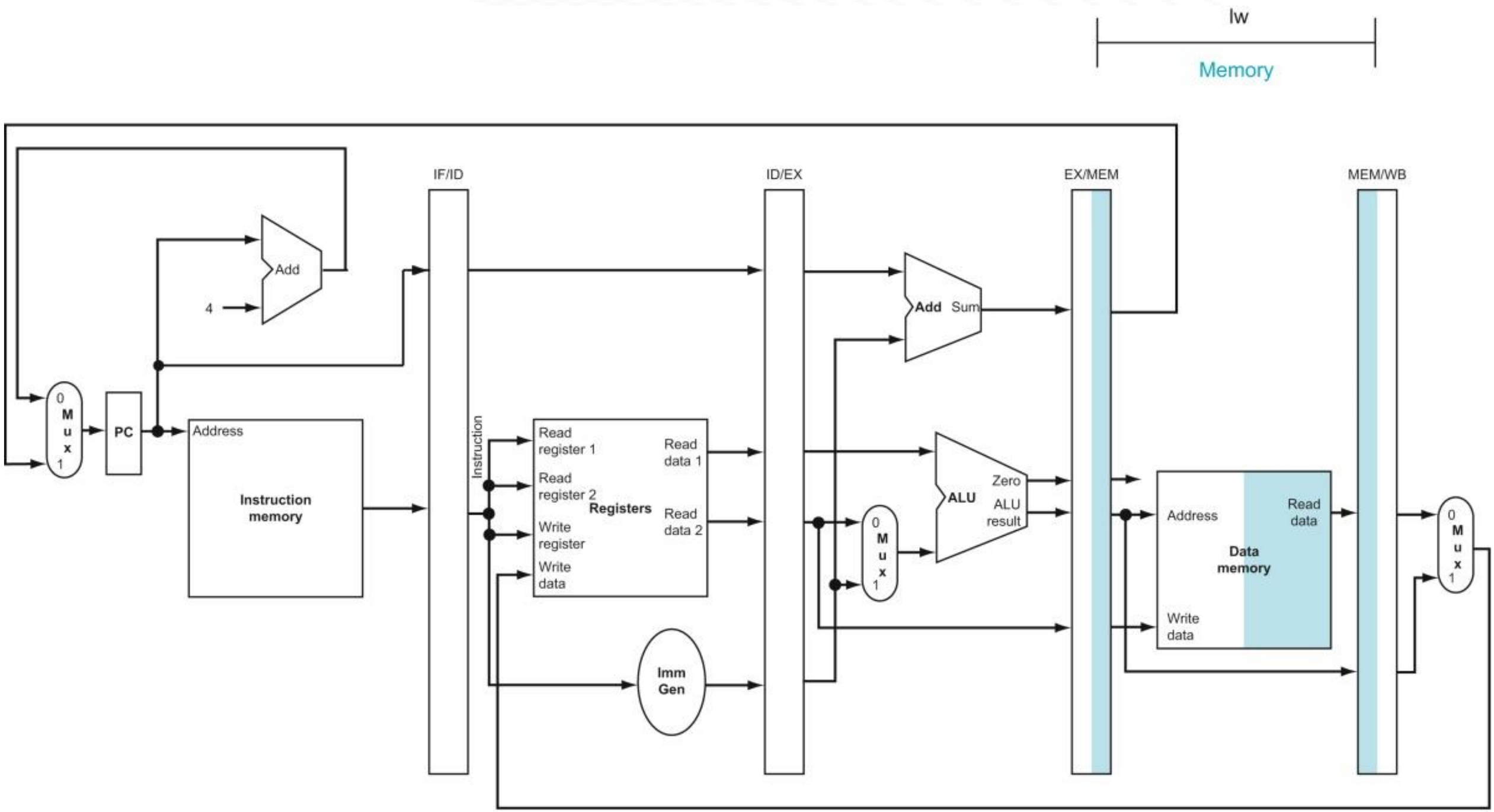
Ejemplo para LW: Etapa 2 (ID)



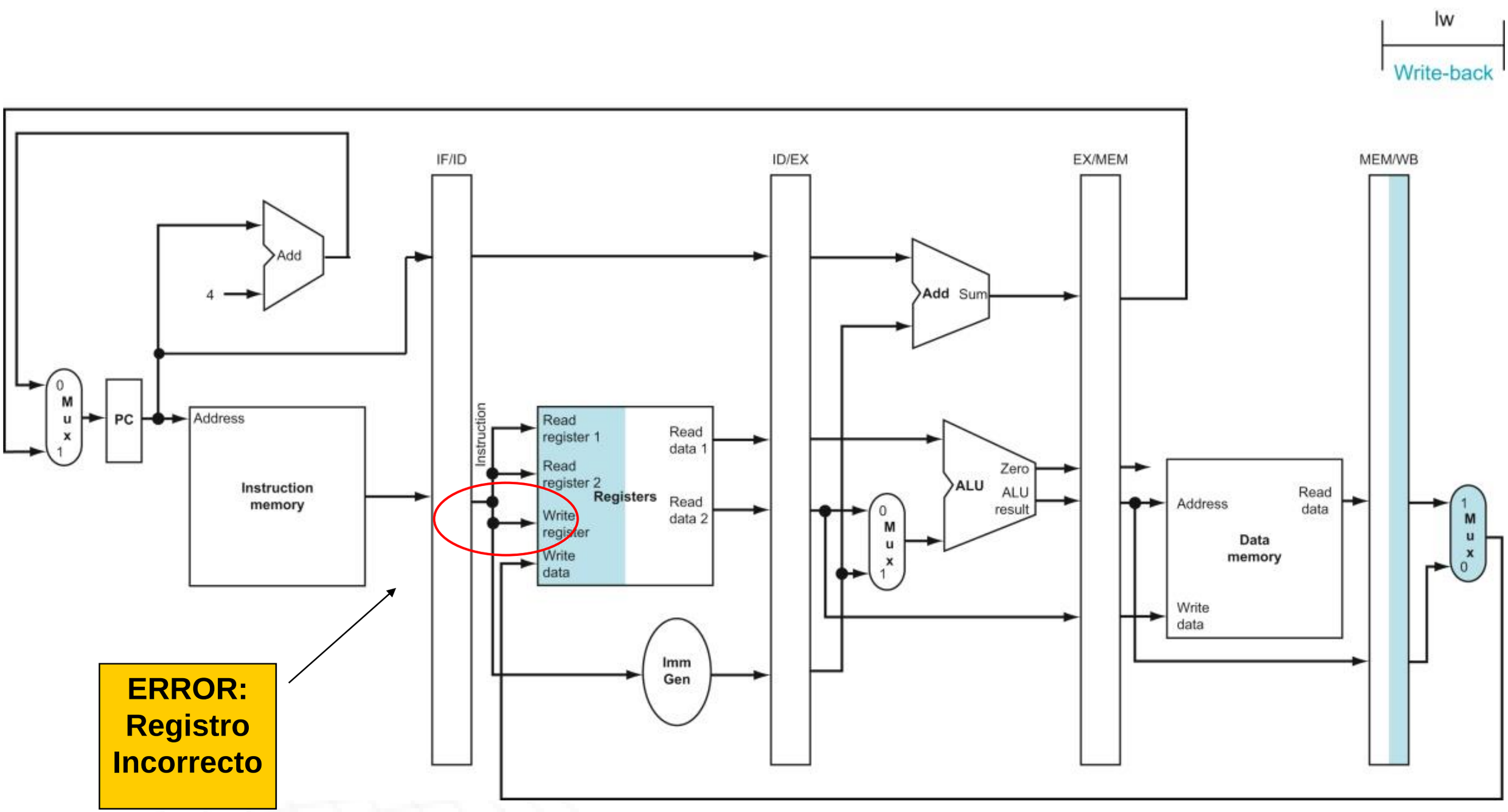
Ejemplo para LW: Etapa 3 (EX)



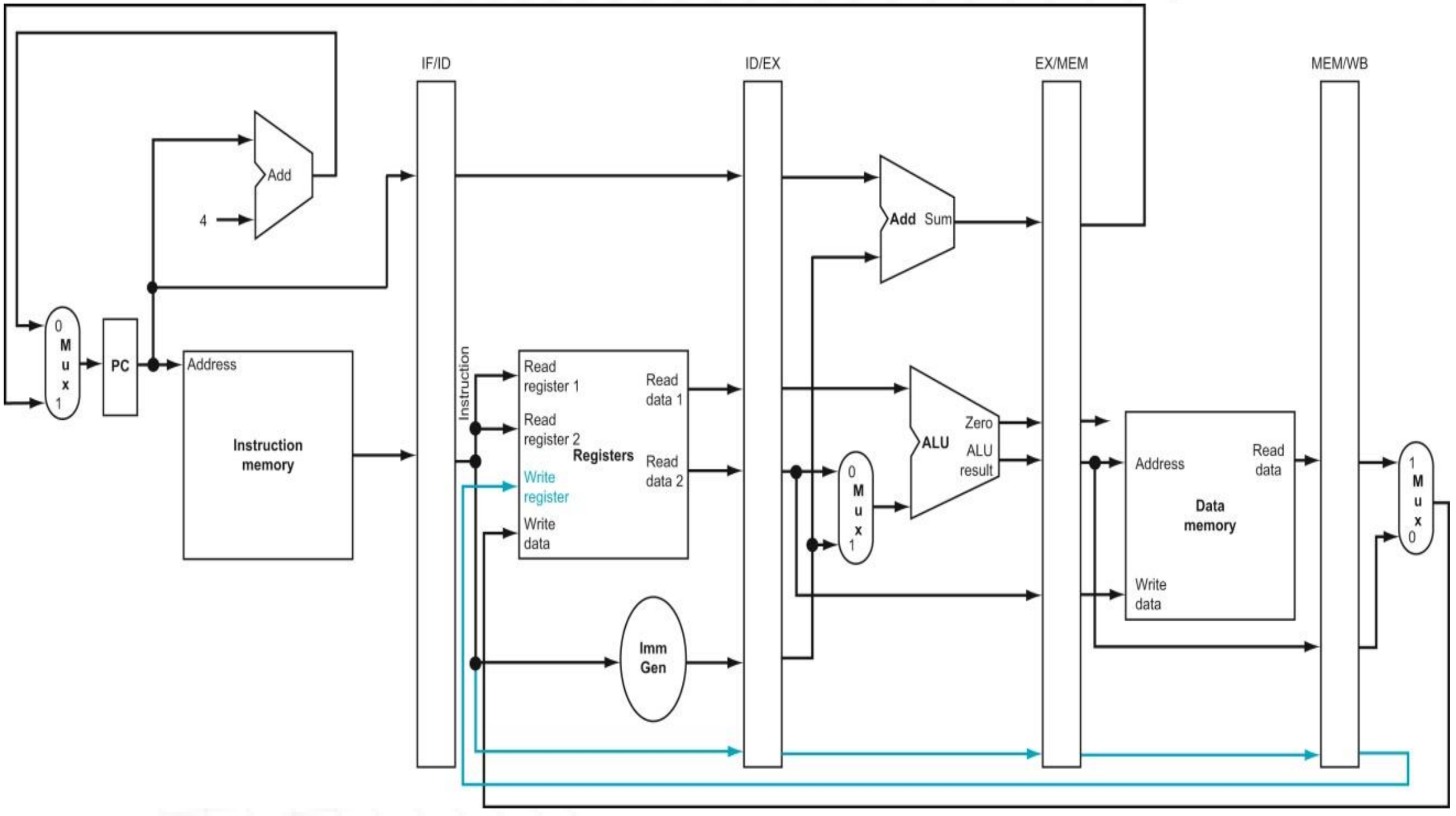
Ejemplo para LW: Etapa 4 (ME)



Ejemplo para LW: Etapa 5 (WB)

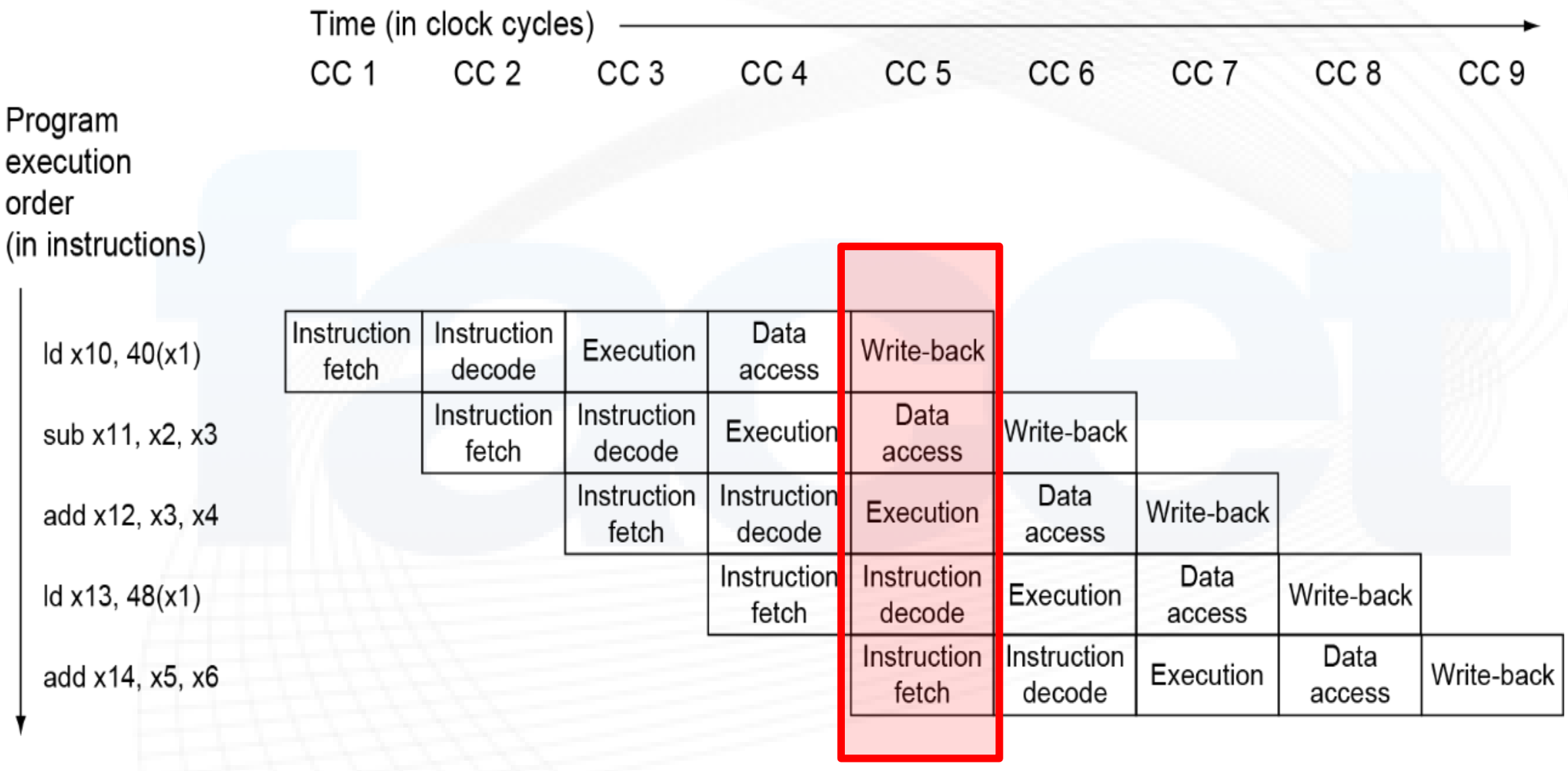


Camino de datos corregido en WB



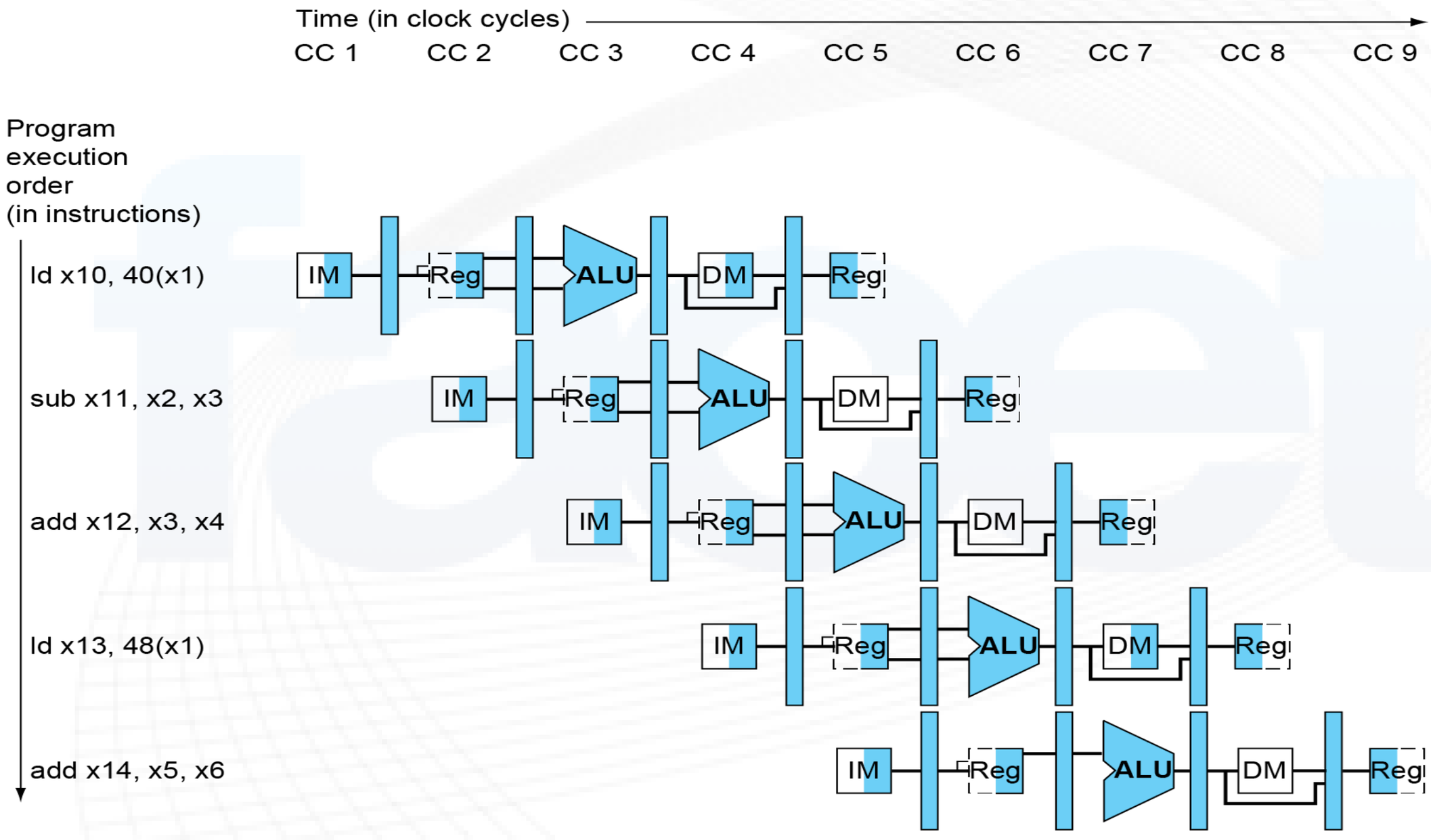
Visualización del pipeline

- Forma tradicional



Visualización del pipeline

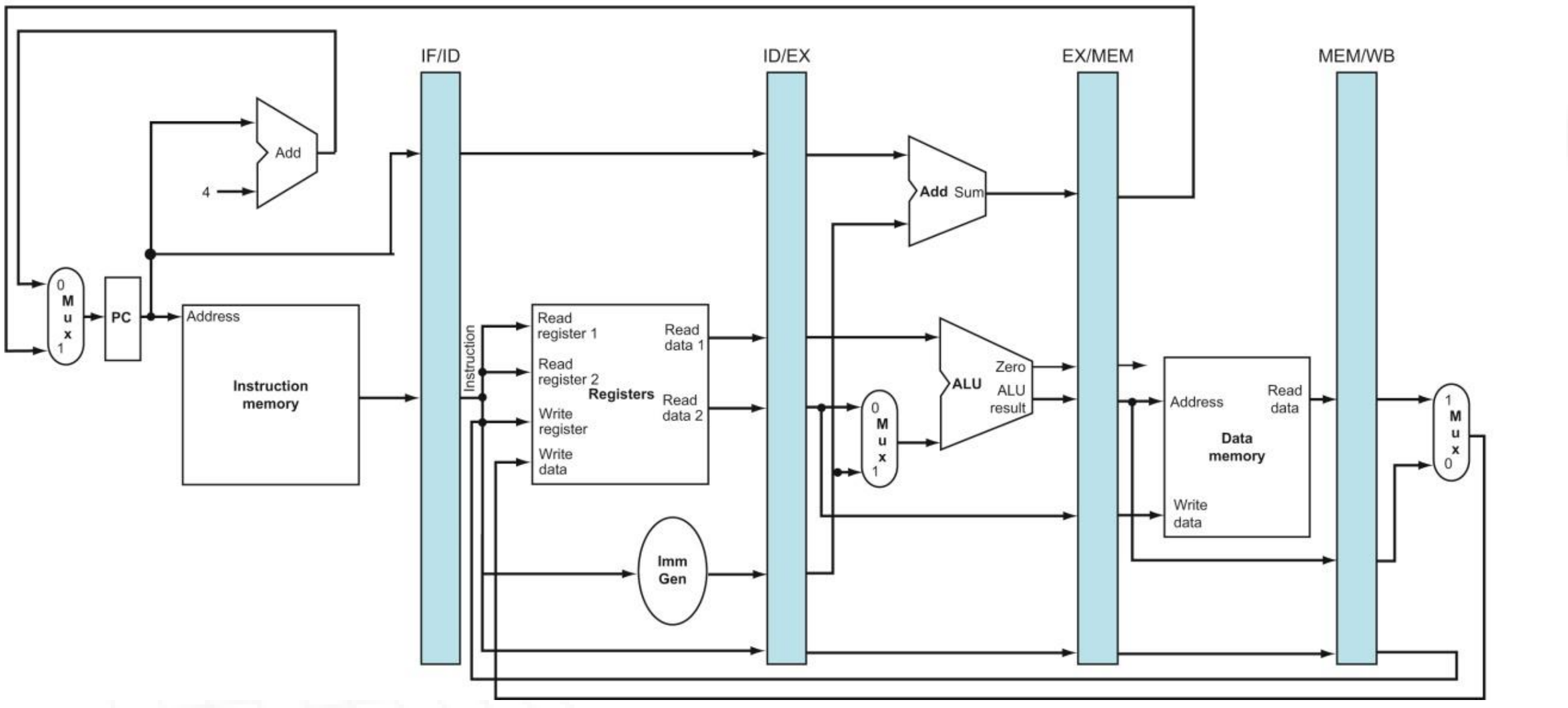
- Uso de recursos



Visualización del pipeline

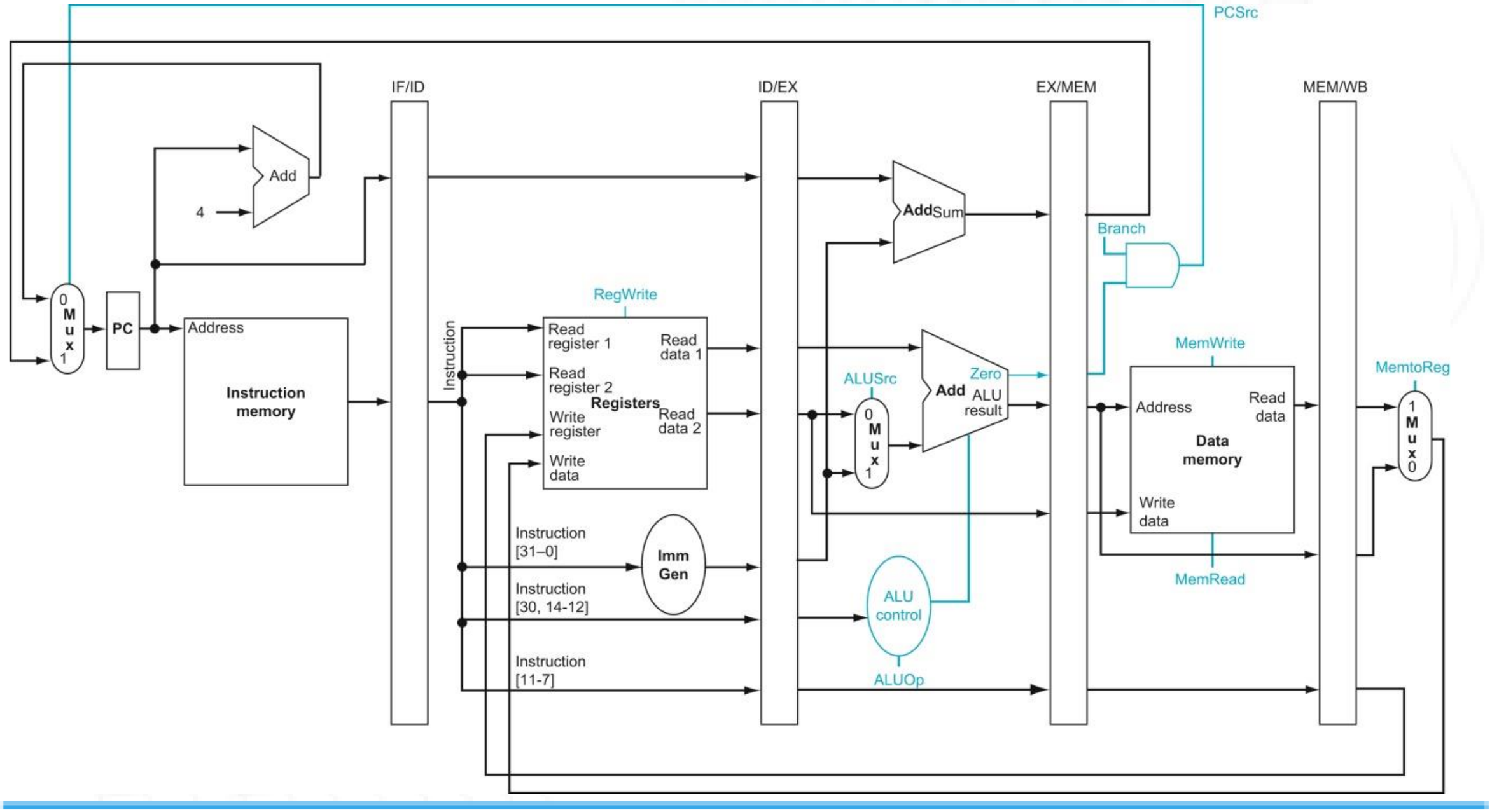
- Instantánea de un ciclo en particular

add x14, x5, x6	lw x13, 48(x1)	add x12, x3, x4	sub x11, x2, x3	lw x10, 40(x1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back



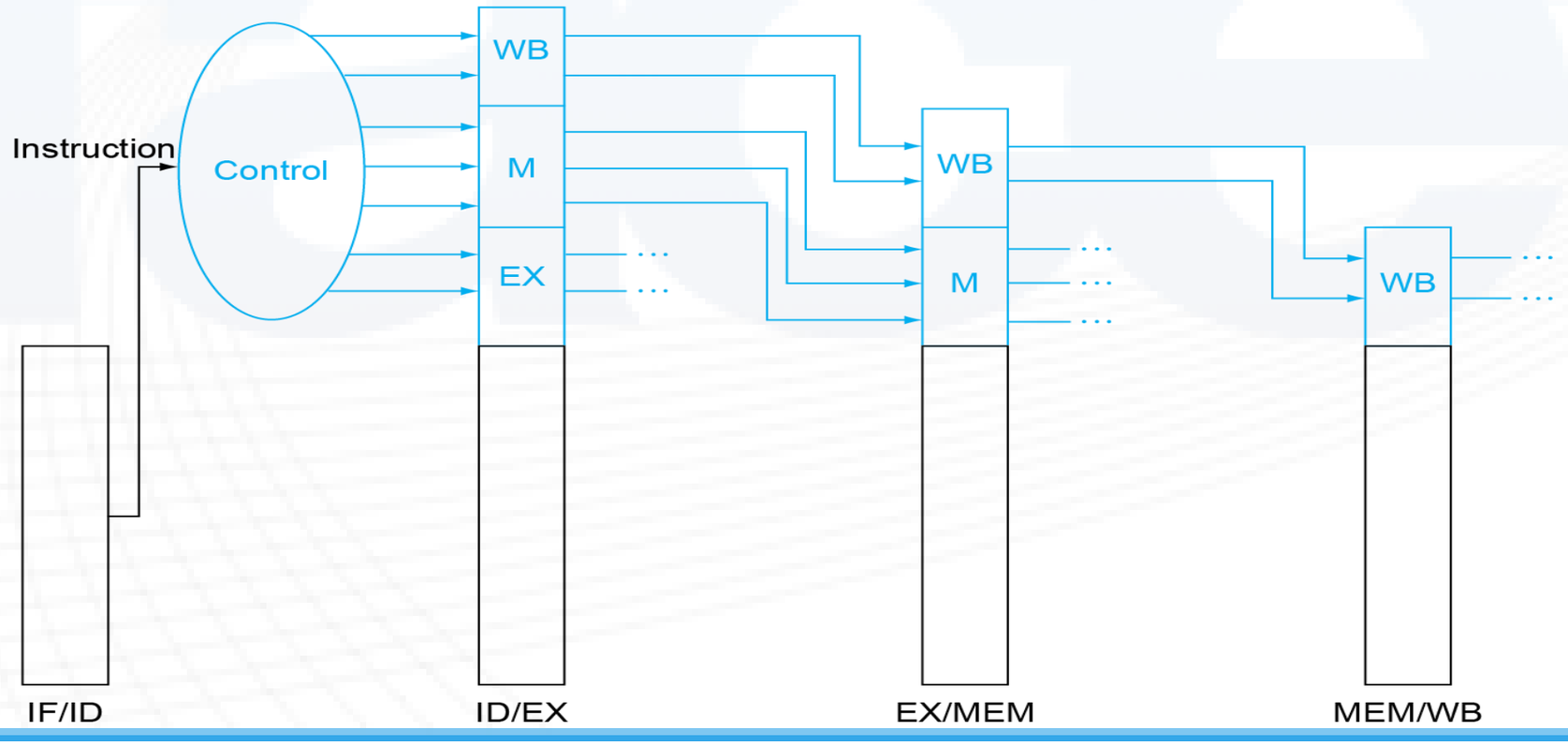
Señales de control del camino de datos

- Son las mismas 7 que en el diseño de ciclo único.

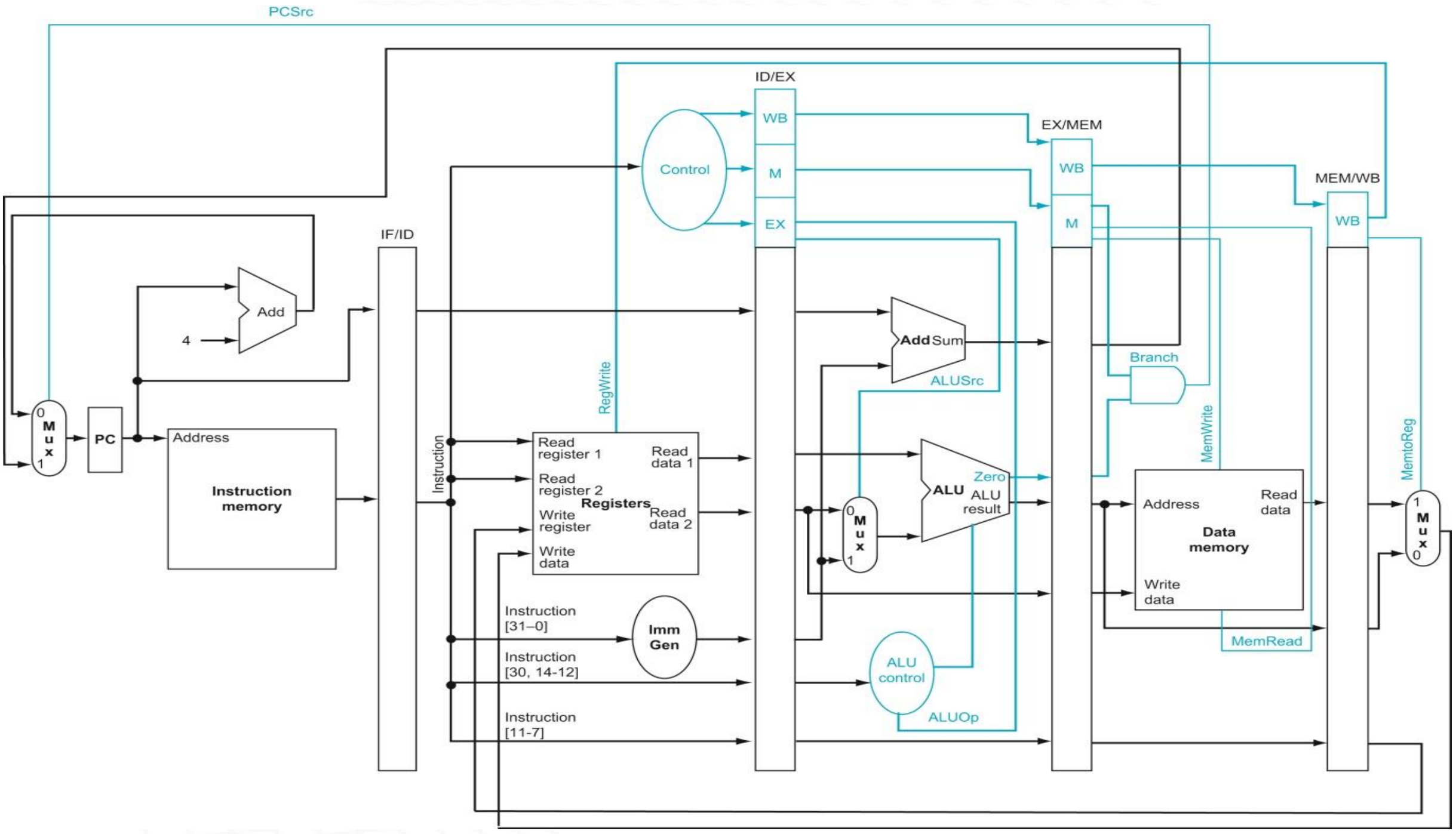


Control en Pipeline

- **¡La Unidad de Control es igual a la del ciclo único!**
  - Se generan las señales de control durante la etapa 2 (ID).
  - Problema: hay señales que se usan en la etapa 3, y otras en la etapa 4, y otras en la 5.
  - Solución: que las señales de control también se escriban en los registros intermedios, y se propaguen junto con la instrucción.



Camino de datos en pipeline completo



Problemas del procesador en pipeline

- Hasta aquí mantuvimos CPI = 1, y redujimos T.
- Pero con la suposición que siempre podemos ingresar una nueva instrucción al pipeline, lo cual no es tan así.
- Los problemas que pueden ocurrir se llaman **riesgos** (*hazards*), y pueden ser:
  - **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).

Cuando hay que parar el pipeline

- Si un riesgo no se soluciona, hay que **parar el pipeline (*stall*)** hasta que se resuelva el riesgo.
  - Se dice que se insertan una o más “burbujas” en el pipeline.
  - Es una solución sencilla, pero que baja la performance, porque son ciclos desperdiciados.
- Una “burbuja” se inserta realizando dos tareas:
  - Poner todas las señales de control del registro ID/EX en cero
    - Hace que en las etapas EX, ME y WB no se haga nada.
  - Evitar la actualización de PC y del registro IF/ID.
    - Se mantienen las instrucciones en etapa IF e ID.
- Los riesgos modifican el CPI:
  - **$CPI_{pipeline} = CPI_{ideal}$ + paradas por riesgos**
  - **$CPI_{pipeline} = CPI_{ideal}$ + paradas por riesgos estructurales + paradas por riesgos de datos + paradas por riesgos de control**

Riesgos Estructurales

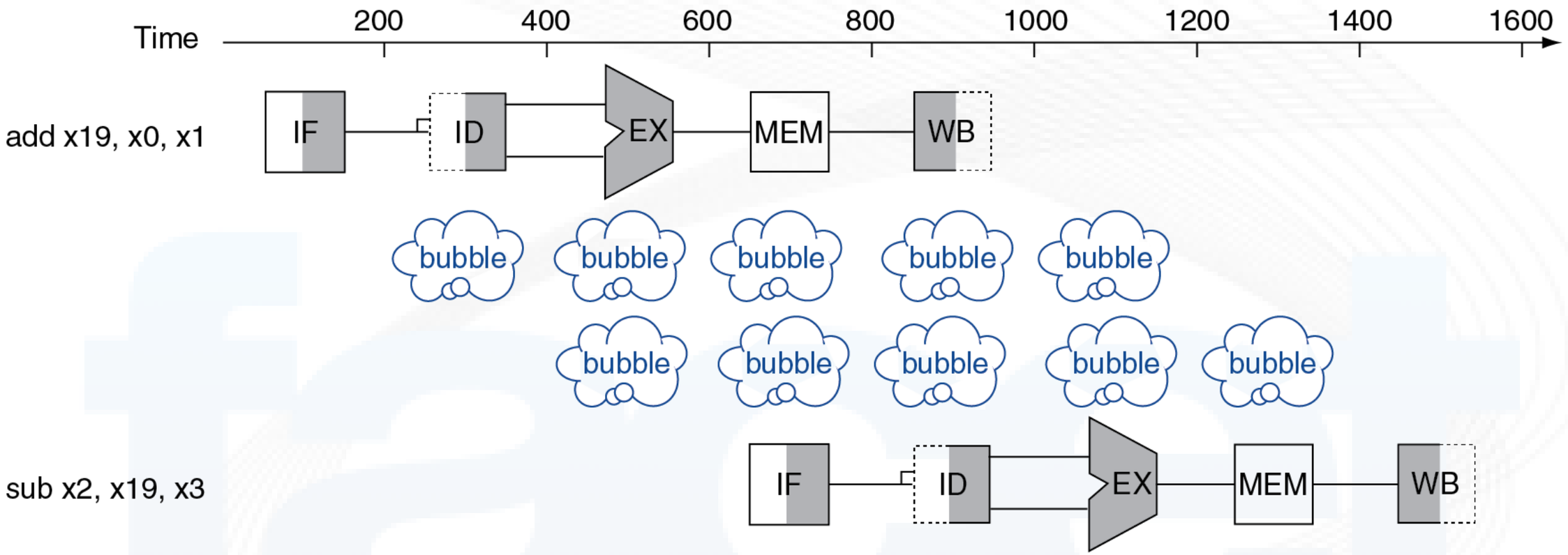
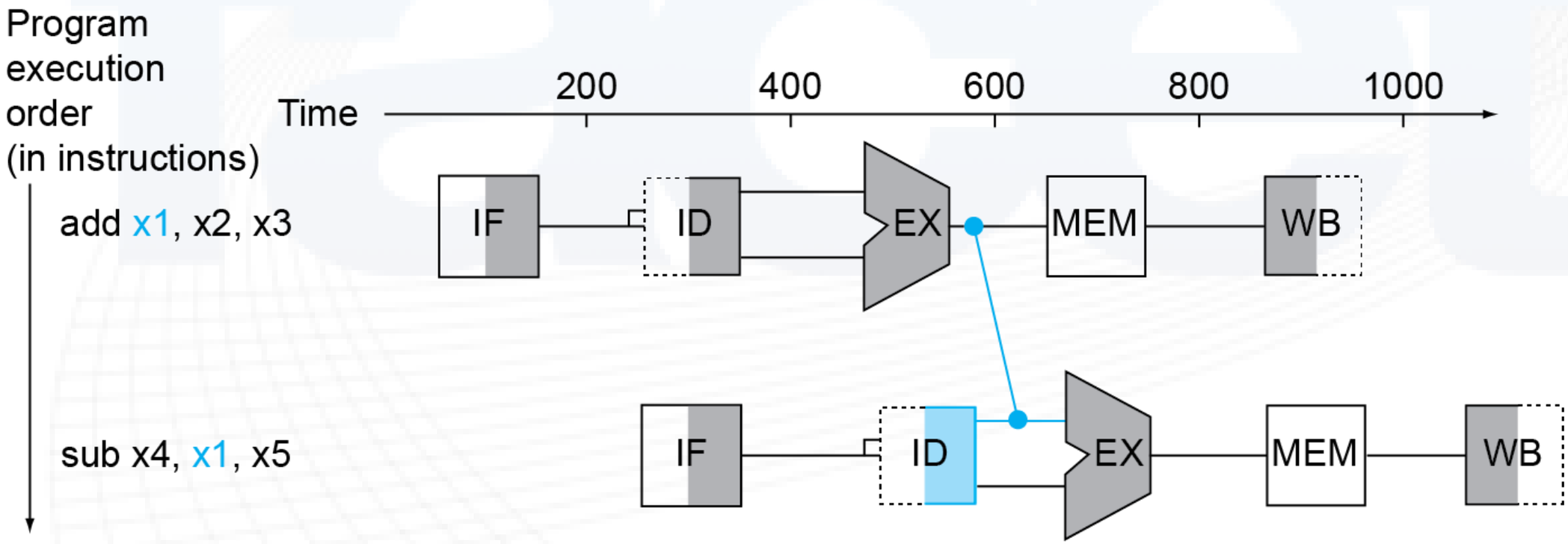
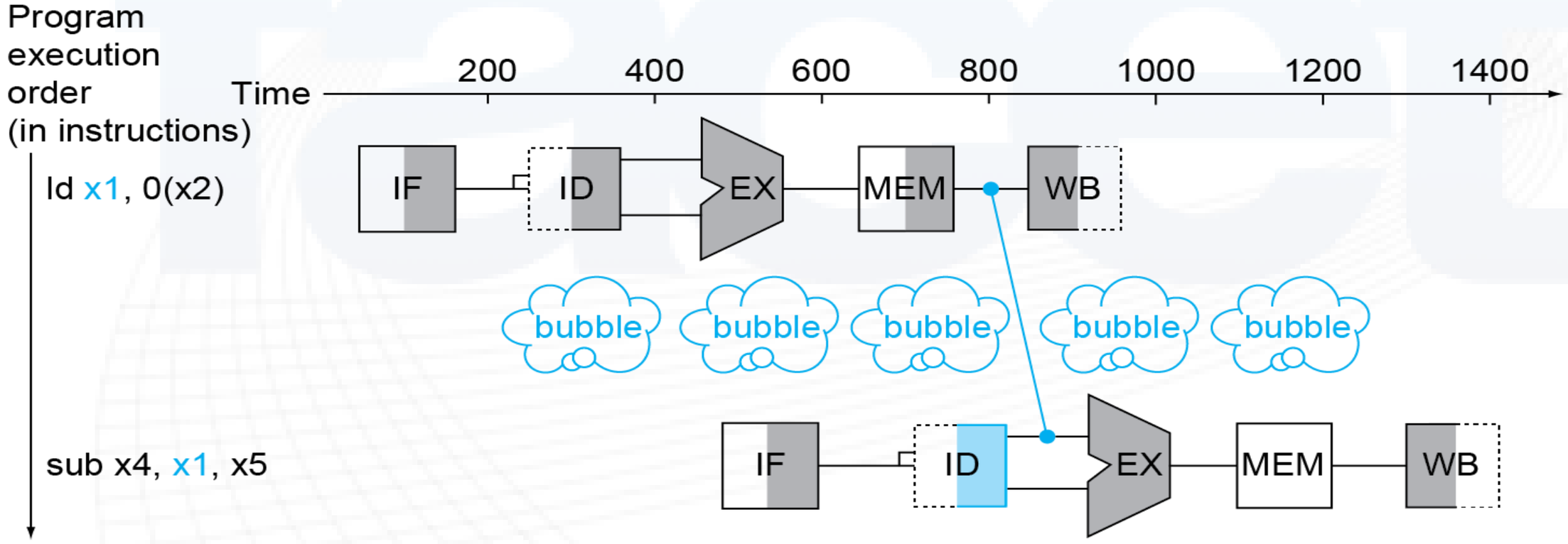
- Dos instrucciones, en dos etapas distintas, quieren utilizar el mismo recurso en el mismo instante.
  - Por ejemplo, si hubiera una única memoria, se intentaría acceder a la misma en IF y en ME.
  - Otro ejemplo, si las instrucciones Tipo R no pasaran por todas las etapas, podrían intentar escribir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

Riesgos Estructurales en RISC-V

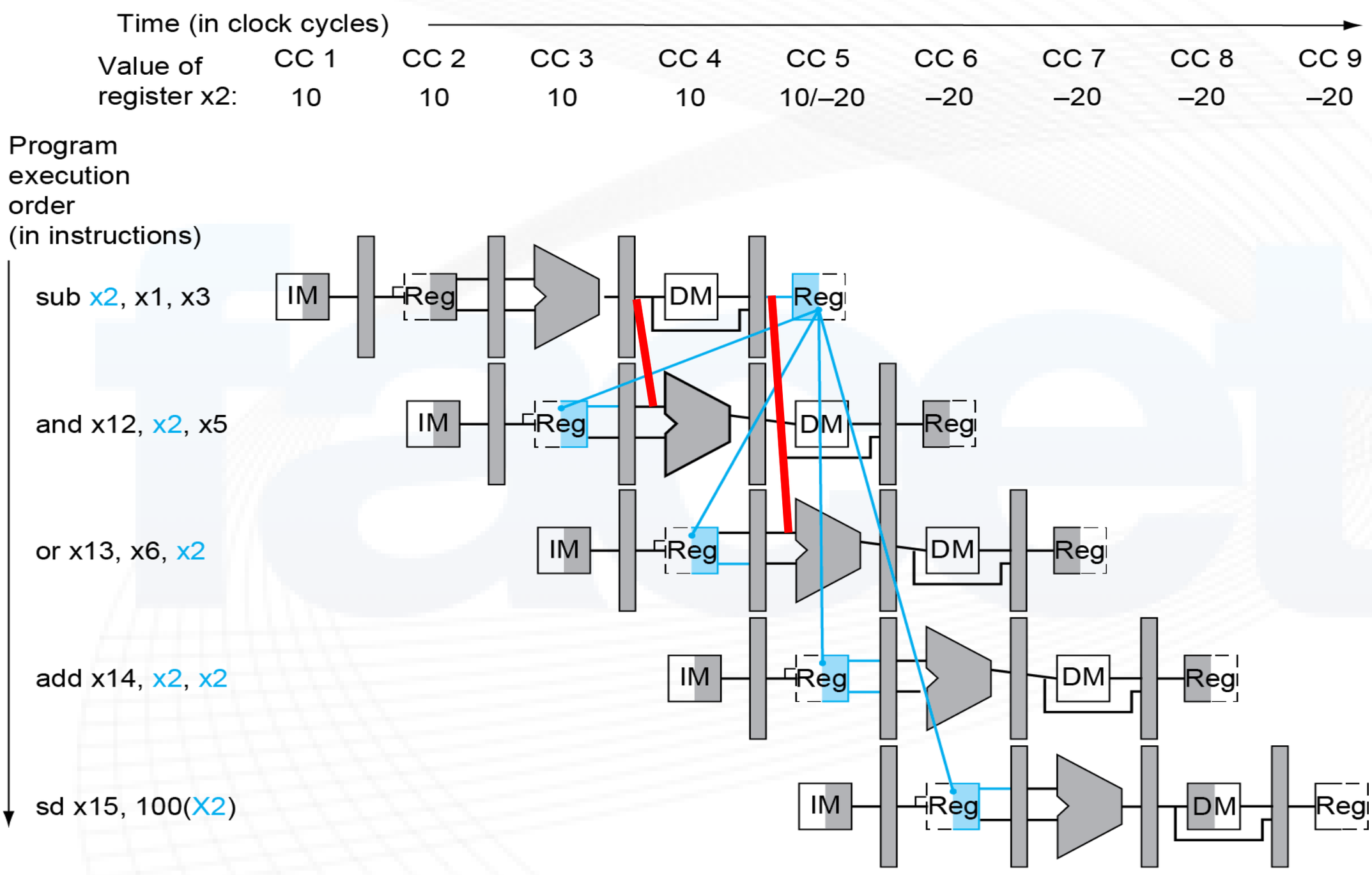
- Nuestro diseño **no posee riesgos estructurales**.
  - Memorias separadas y recursos duplicados.
    - Aumentan el costo, pero mejoran la performance.
  - Todas las instrucciones pasan por todas las etapas.
  - Sólo se utiliza una unidad funcional por etapa.
  - Todas las instrucciones usan la misma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

Riesgos de Datos

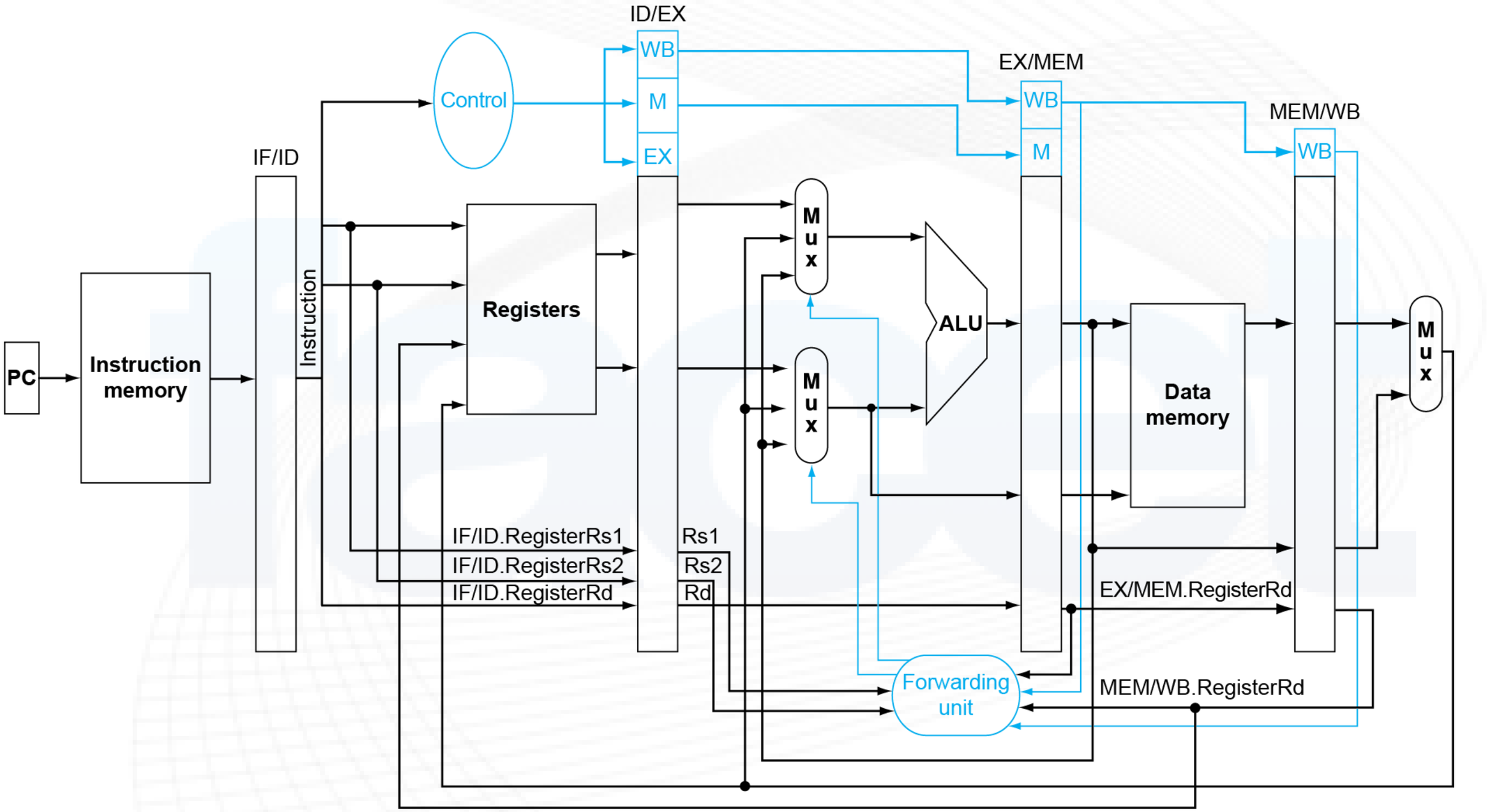
- Una instrucción depende del resultado de una instrucción previa que aún está dentro del pipeline.
  - ```
    add     x19, x0, x1
    sub     x2, x19, x3
    ```

- 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.
    ```

Riesgos de Datos



- Son sólo dos burbujas, porque recordemos que el Banco de registros escribe con el flanco que inicia el 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.
  - Se conoce como ***Forwarding***, o *Bypassing*.



Riesgos de Datos – Adelantamiento

- Usar adelantamiento requiere agregar muchas interconexiones nuevas, más nuevos multiplexores.
  - Ahora un dato puede provenir de nuevas fuentes.
- Más una unidad nueva de Adelantamiento.
  - Que detecte la ocurrencia del riesgo, y genere las señales de control necesarias.
- Agrega complejidad y costo al diseño.
  - Pero soluciona casi todos los riesgos de datos.
  - *¿Casi?*

Riesgos de Datos – Problema con LW

- 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.
  - Se lo conoce como ***Load-to-use Penalty***.



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?*

Riesgos de Datos – Dependencias

- 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).
  - Y ambos se pueden solucionar por adelantamiento.
    ```

Riesgos de Datos – Dependencias



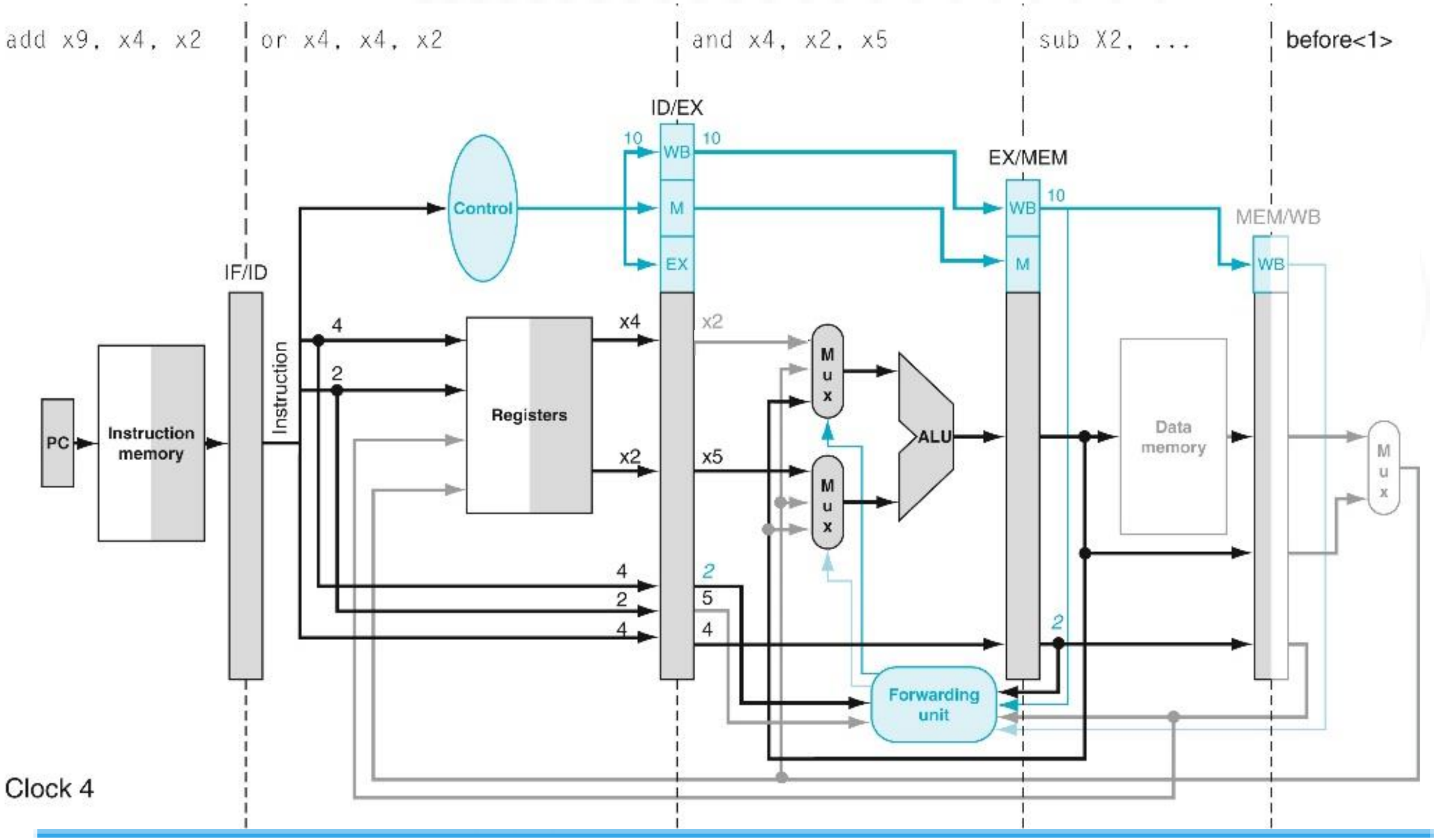
Pipeline con adelantamiento



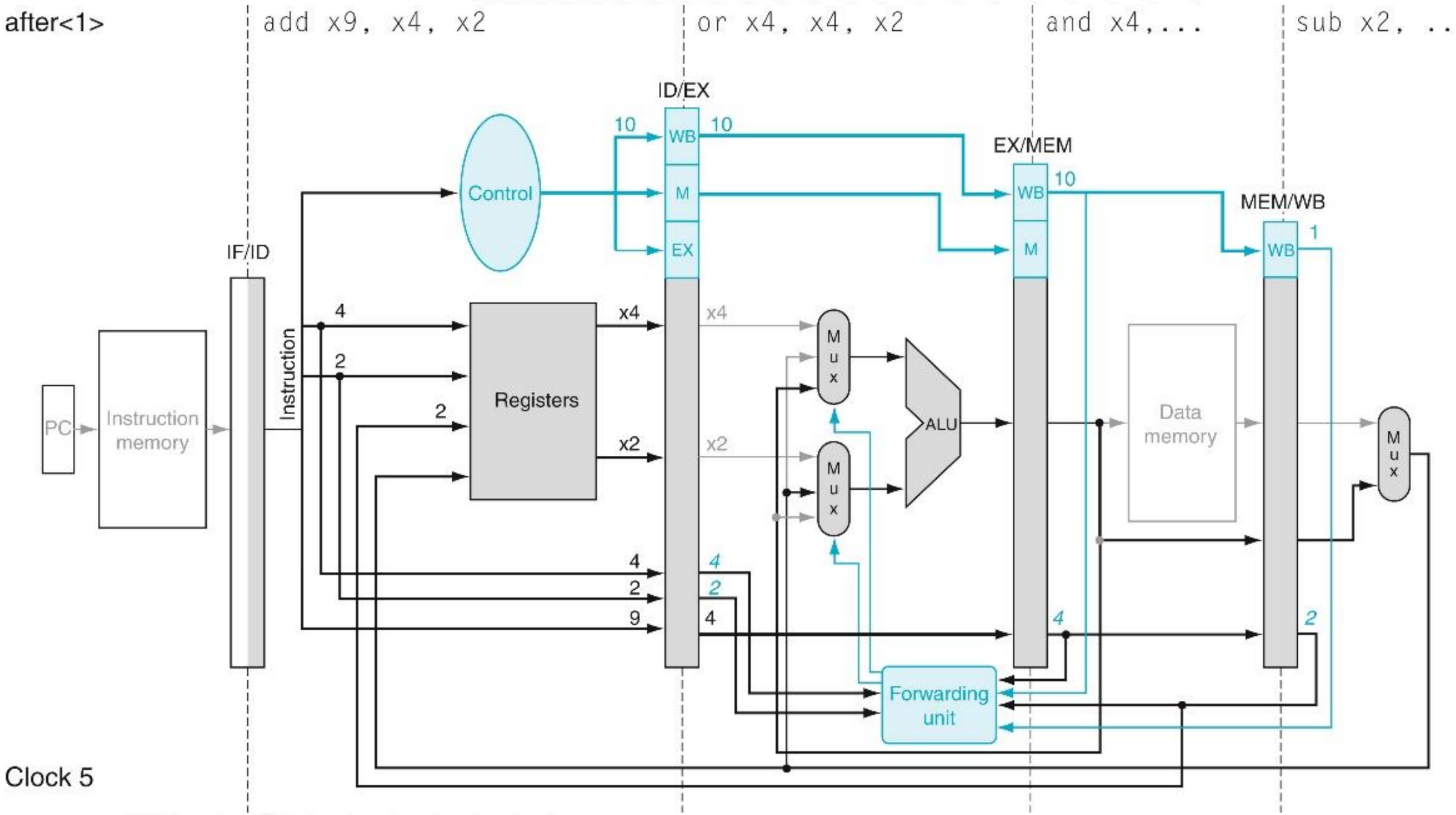
Unidad de adelantamiento

- La unidad de adelantamiento necesita detectar el riesgo.
  - Comparar si alguno de los dos registros que se leen en la etapa 2 (ID) es igual al registro que se escribe de la instrucción en la etapa 3 (EX) o en la etapa 4 (ME).
  - Y si es que las instrucciones en la etapa 3 o 4 escriben en un registro (RegWrite).
    - Y sólo si el registro destino es distinto de cero.
- Y luego actuar en consecuencia
  - No adelantar nada, porque no hay riesgo.
  - Adelantar desde la etapa ME o desde la etapa WB.
- Si ocurre un riesgo doble, se adelanta el más reciente.
- Además, hay que detectar el riesgo extra del LW.

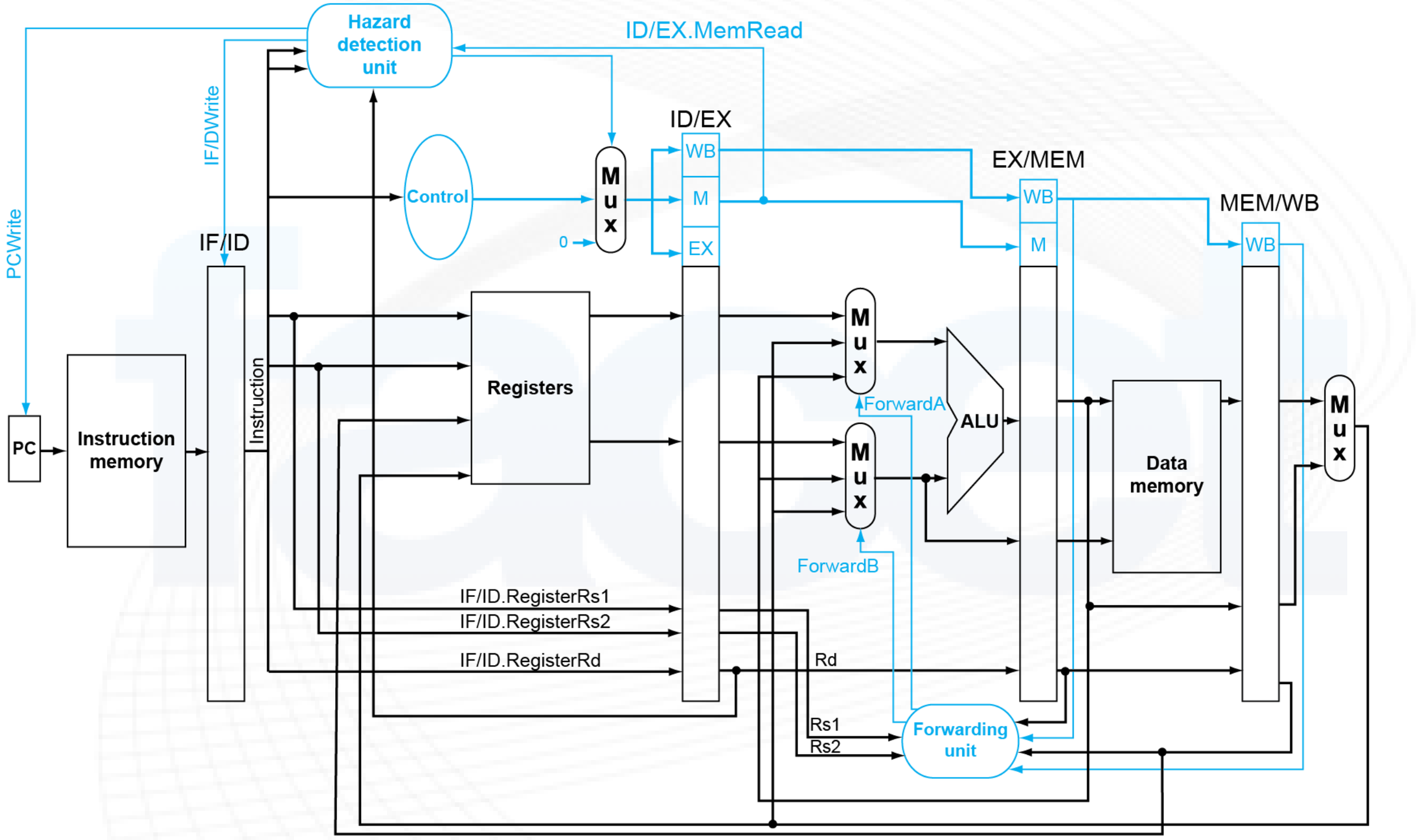
Unidad de adelantamiento – Ejemplo



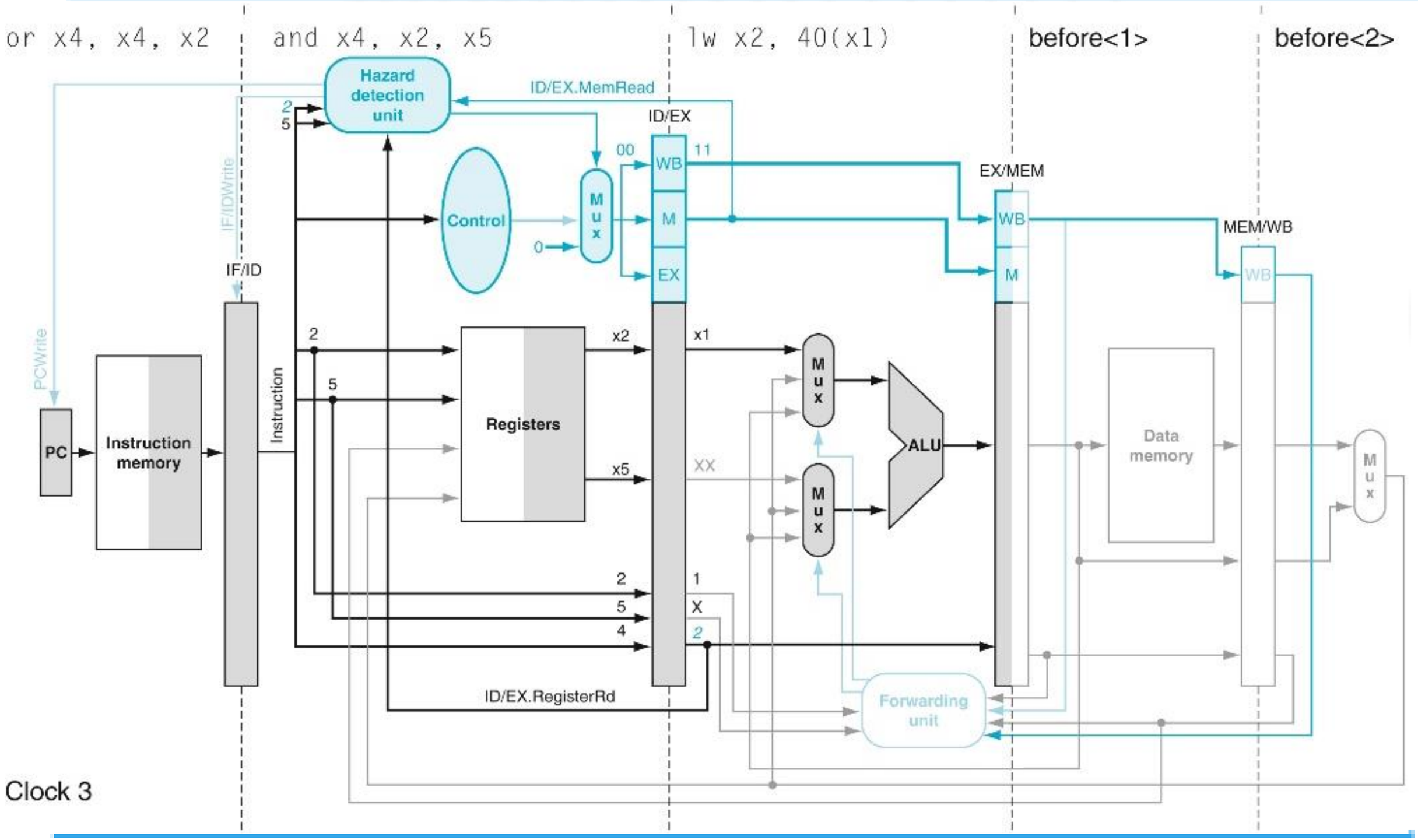
Unidad de adelantamiento – Ejemplo



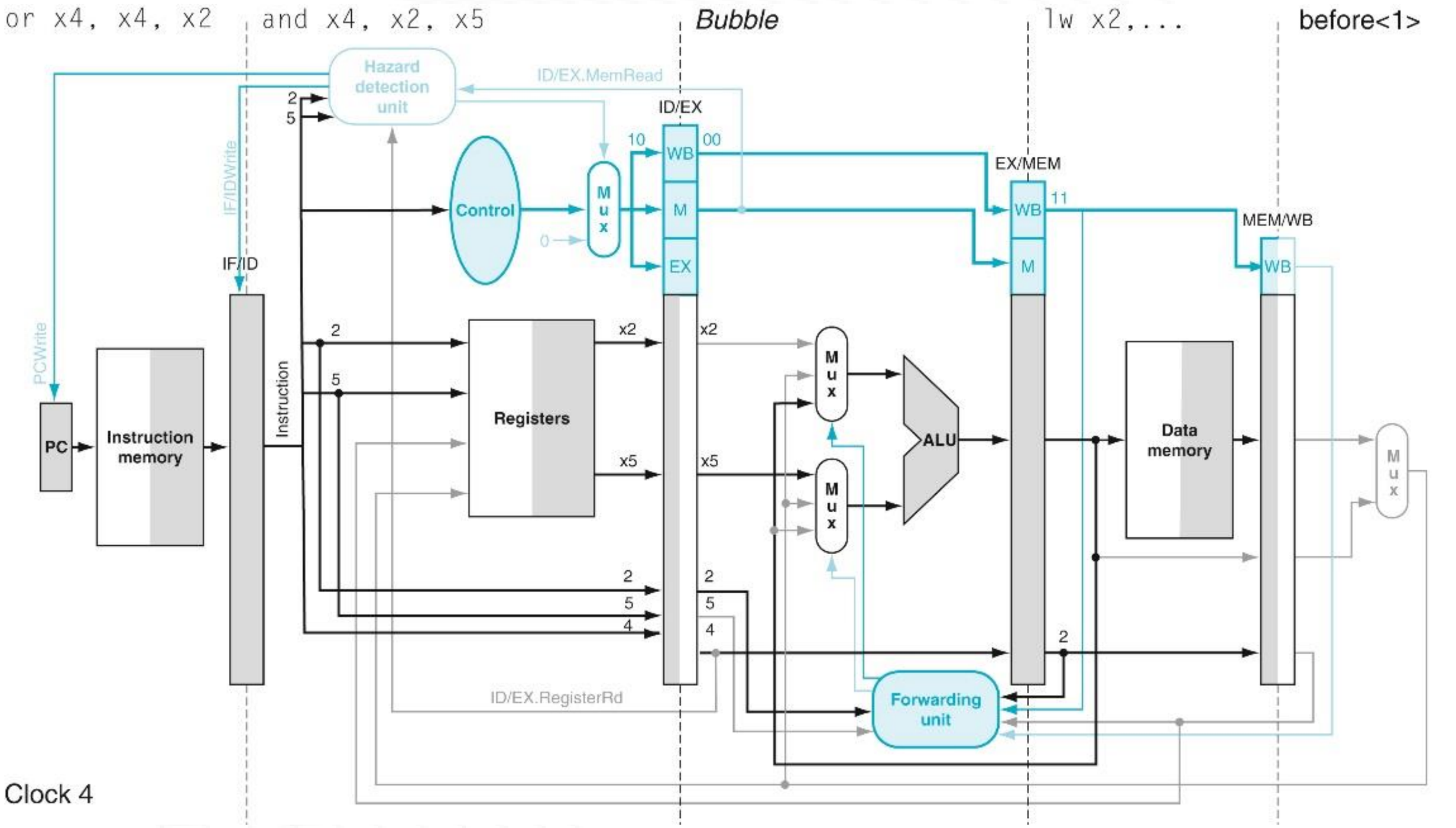
Pipeline más detección de riesgos



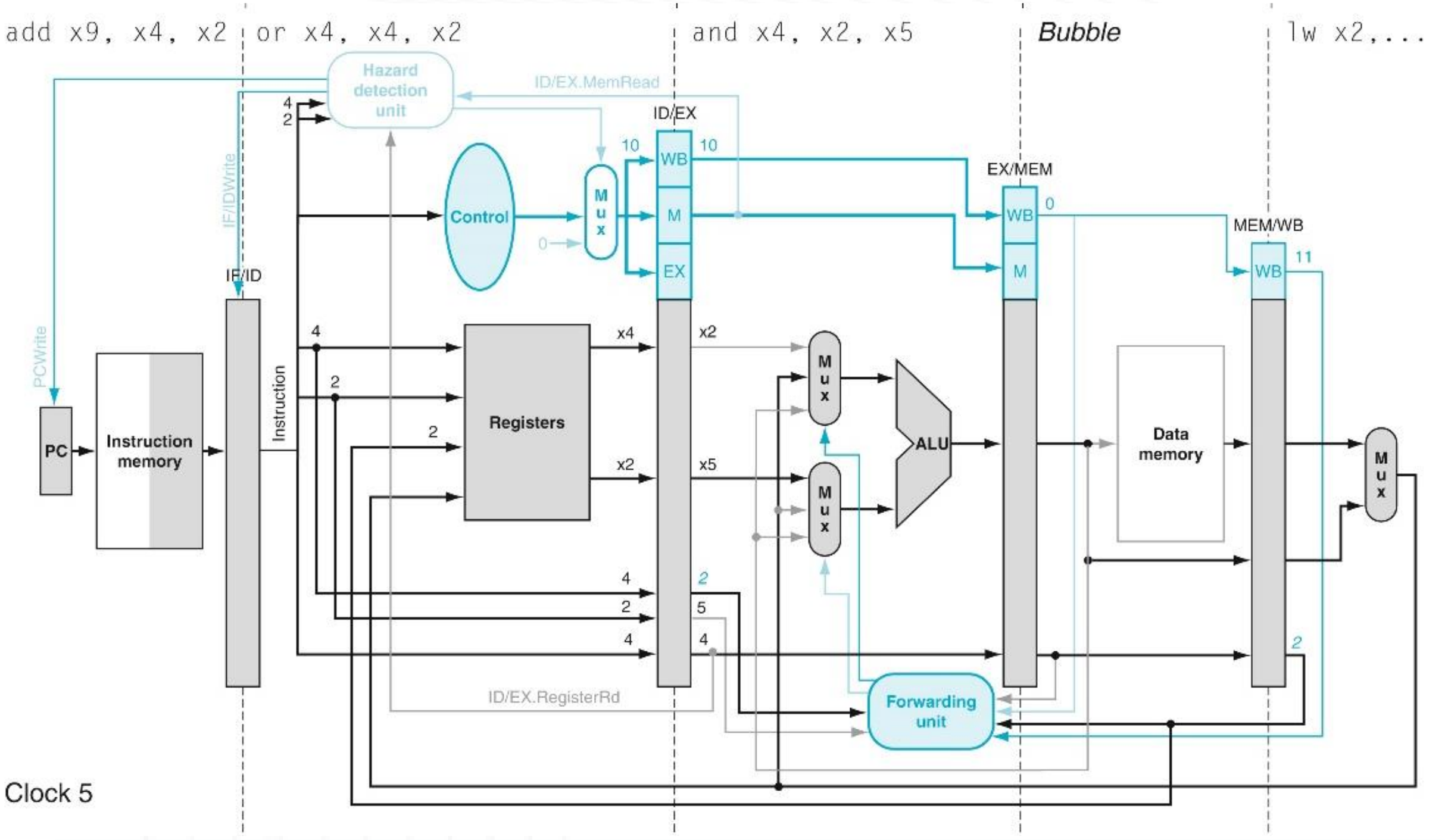
Inserción de burbujas – Ejemplo



Inserción de burbujas – Ejemplo



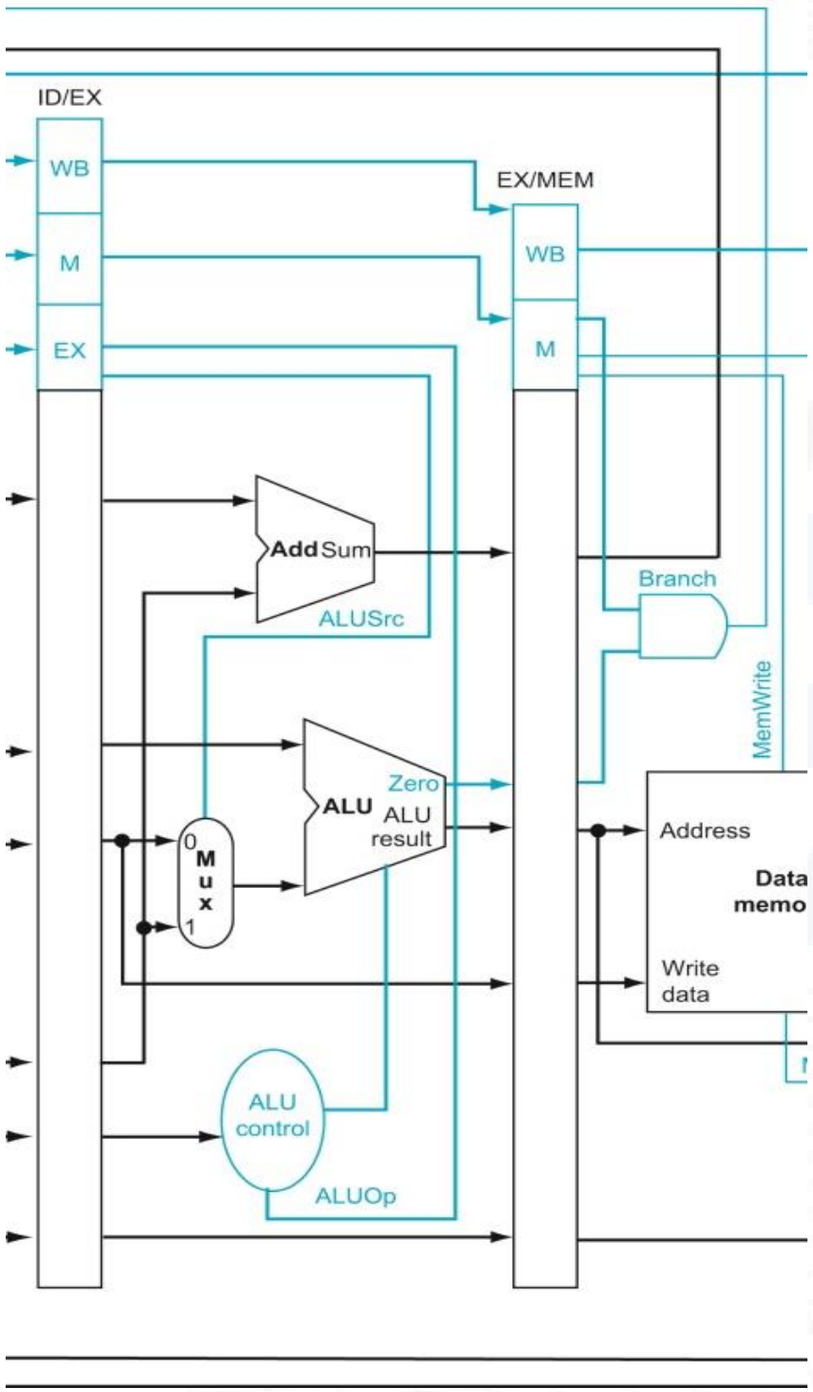
Inserción de burbujas – Ejemplo



Riesgos de Control

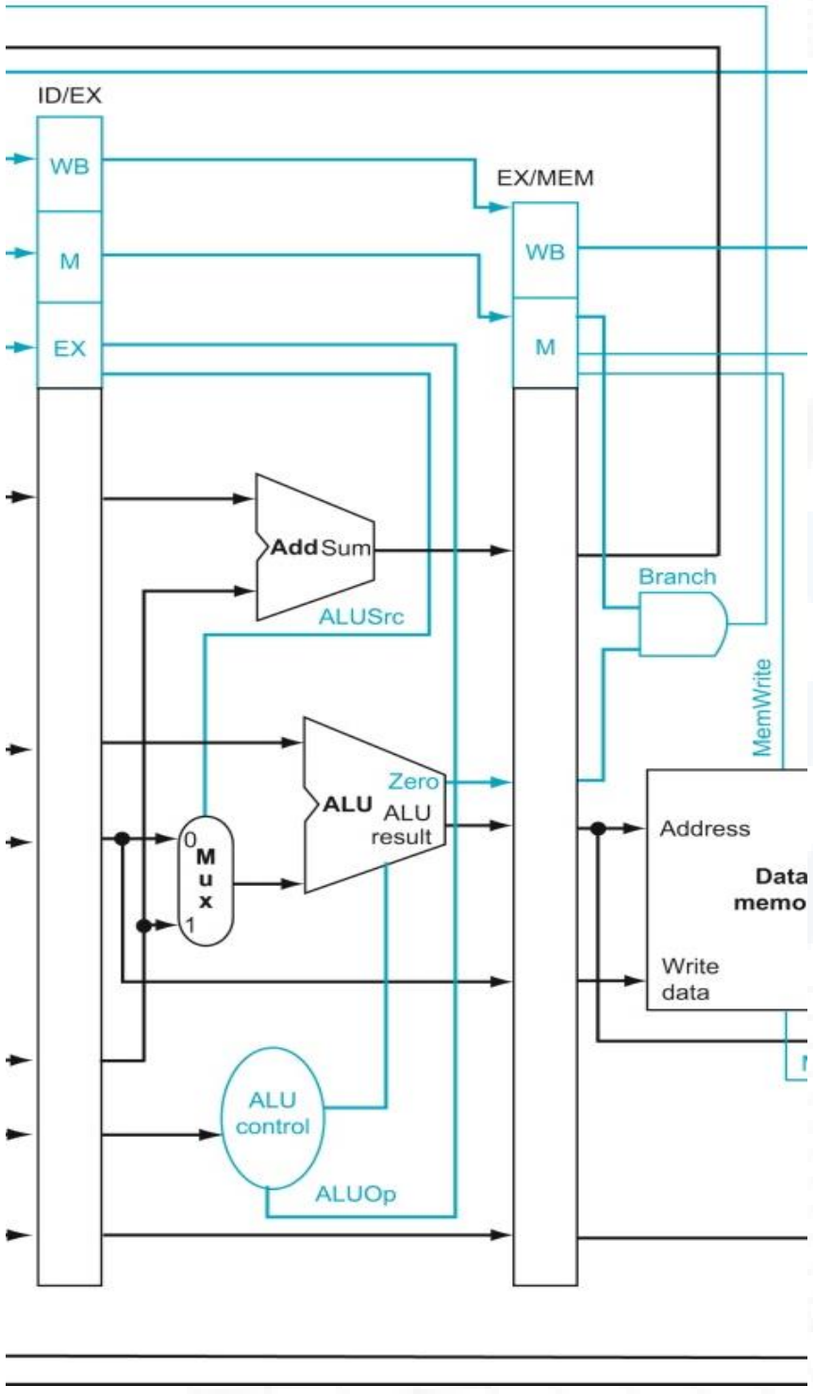
- Los saltos son los que determinan el flujo de control de un programa.
  - La búsqueda de la siguiente instrucción depende del resultado de un salto.
  - En un pipeline, no siempre se puede hacer el fetch de la instrucción correcta.
    - Porque el salto se resuelve completamente en la etapa 4 (ME).
    - ¡Esto implica que habría que agregar tres burbujas por cada salto!
- La resolución de un salto implica dos tareas:
  - Determinar si realmente se efectuará o no.
  - Determinar cuál es la dirección destino del salto.

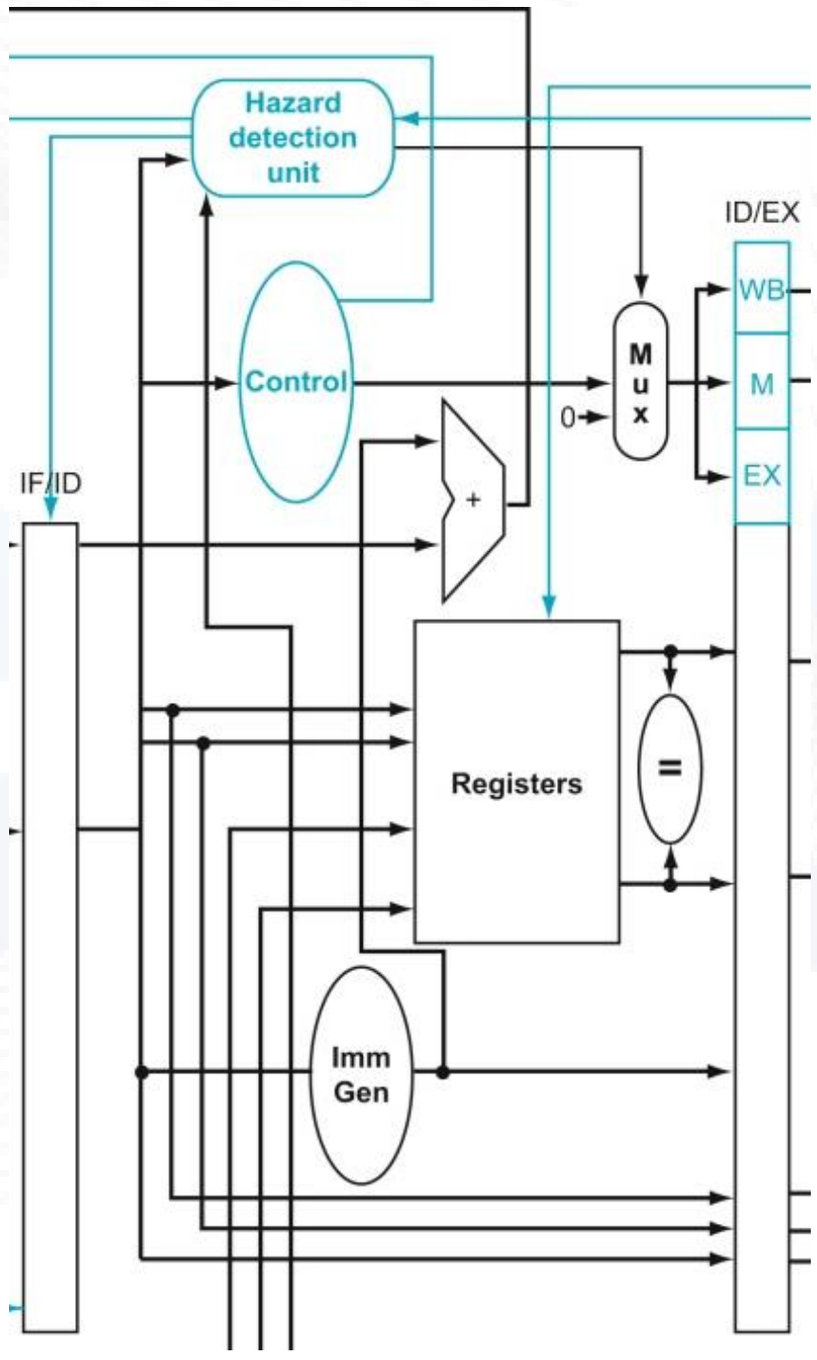
Riesgos de Control – Primera alternativa



- Para determinar si el salto se efectuará o no, en la etapa EX la ALU compara los registros.
  - Podríamos agregar un comparador a la salida del Banco de Registros para realizar esta tarea.
  - Liberamos a la ALU de esta tarea.
  - Movemos la determinación del salto a la etapa ID.
- Para calcular la dirección de salto, en la etapa EX hay un sumador.
  - Sus datos ya están disponibles en la etapa ID, así que podemos cambiarlo de lugar.
- **Esto nos permite resolver todo el salto en la etapa ID.**
  - Siempre y cuando no haya problemas de timing en la duración de la etapa.

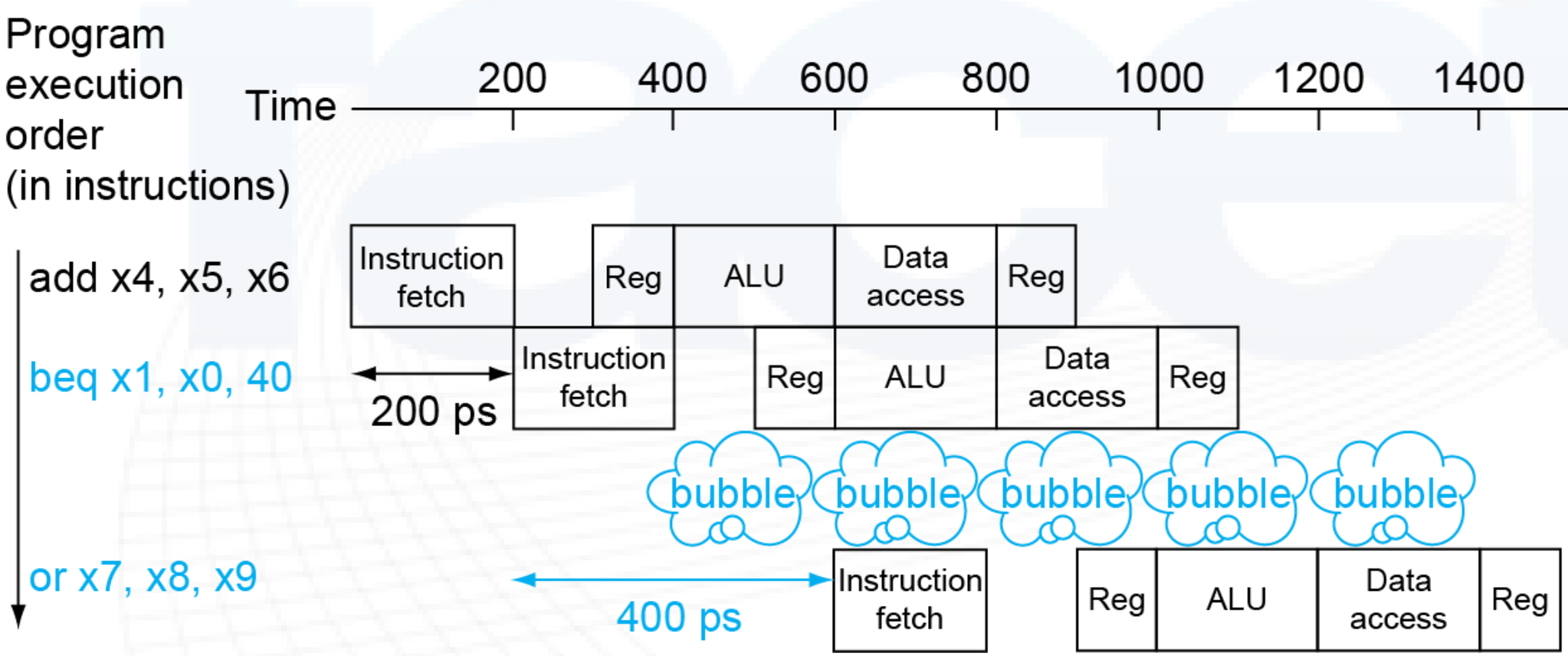
Riesgos de Control – Primera alternativa





Riesgos de Control – Primera alternativa

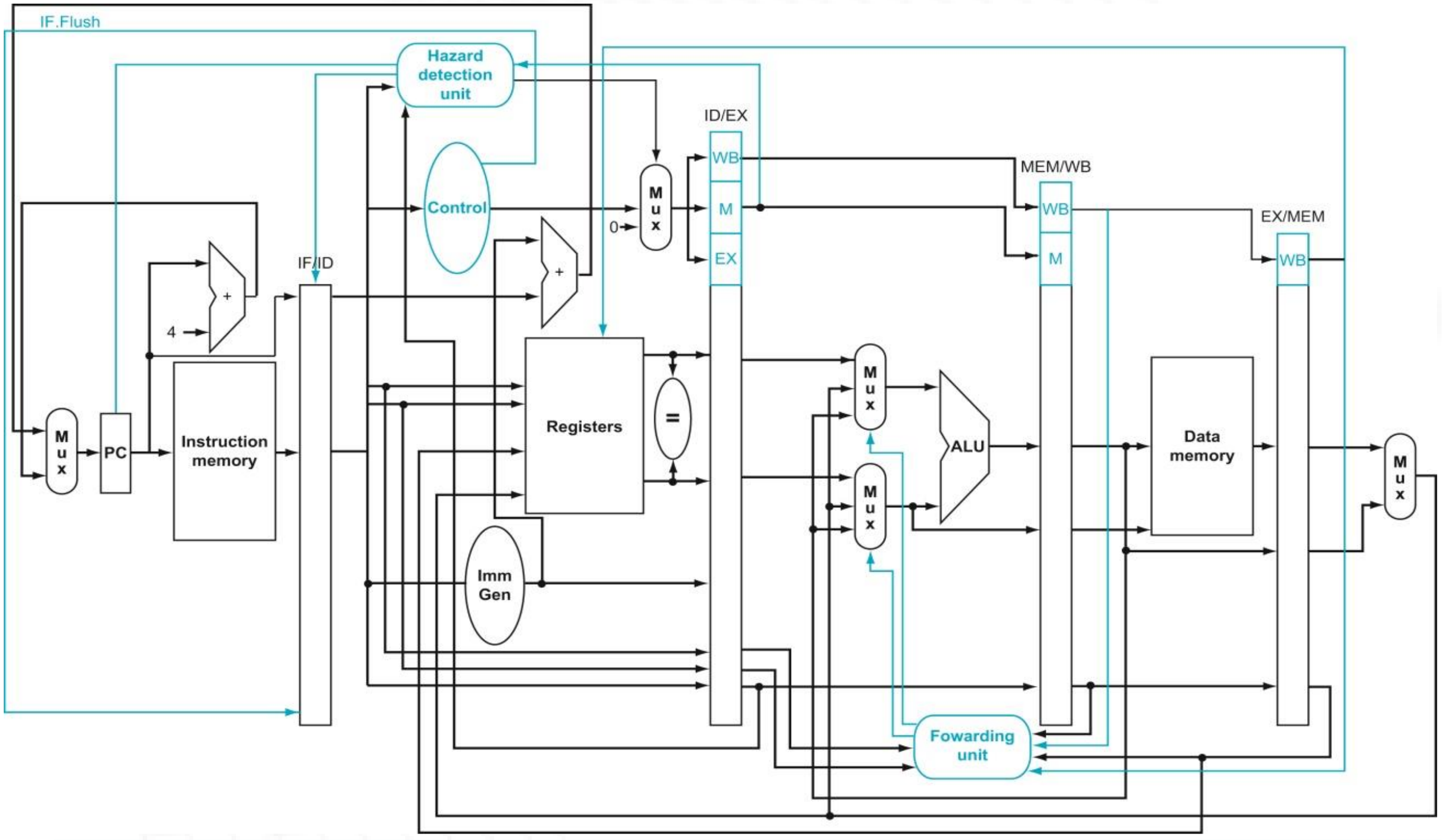
- Para que esta alternativa funcione, la condición a evaluar en los saltos debe ser **simple**.
- Además, genera nuevos riesgos de datos, que necesitan nuevos caminos de adelantamiento.
  - En el caso que la instrucción previa al salto escriba en un registro que el salto lee.
- E inclusive, aún debemos insertar una burbuja.



Riesgos de Control – Predicción de Saltos

- El problema de la alternativa anterior, es que en pipelines más largos, es muy difícil adelantar toda la resolución del salto.
  - Y por lo tanto, las burbujas a insertar son inaceptables.
- Idealmente, deberíamos intentar hacer toda la resolución del salto en la etapa IF.
  - Lo cual suena bastante ilógico, porque ni siquiera se sabe si la instrucción que se busca es un salto.
- Entonces se usa **predicción de saltos**.
  - Intentar adivinar si el salto se efectúa o no.
  - Y solamente parar el pipeline cuando la predicción falla.
    - Se la conoce como ***Branch-misprediction Penalty***.
  - Dos tipos de predicción: estática y dinámica.
  - Más detalles… en el tema siguiente.

Pipeline 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.

Agradecimientos

- Las diapositivas de este tema fueron basadas en las realizadas por el Ing. Daniel Cohen.
- A su vez inspiradas en las clases del curso CS152 de la Universidad de Berkeley, California, USA.
  - Realizadas por los Prof. D. A. Patterson, John Lazzaro, Krste Asanovic.