Práctica 2: Mejoras al microprocesador segmentado

El objetivo de esta práctica es implementar una serie de mejoras sobre el microprocesador RISC-V (visto en clase de teoría y desarrollado en la práctica 1). En esta práctica se resolverán los riesgos (*"hazards"*) generados al segmentar el procesador RISC-V.

Para realizar esta práctica se recomienda consultar la sección 4.1 (Data Hazards: Forwarding versus Stalling) del libro de Patterson & Hennessy y las transparencias de la teoría del Tema 2.

Ejercicio 1: Riesgos de datos (7 puntos)

En este ejercicio se modifica el hardware del microprocesador para lograr una ejecución correcta en situaciones en las que la ejecución de una instrucción depende del resultado de otra anterior aún presente en el pipeline, debido al uso de un mismo registro (Read After Write –*RAW*– data hazard). Se implementarán 3 mecanismos:

1) ***Forwarding de datos hacia la ALU.***

Sobre el diseño de microprocesador anterior se añadirán los caminos de adelantamiento de datos ("forwarding") mostrados en la siguiente figura, que adelantarán el resultado de la instrucción anterior (desde MEM) o desde dos anteriores (desde WR) para su uso en la etapa "EX".

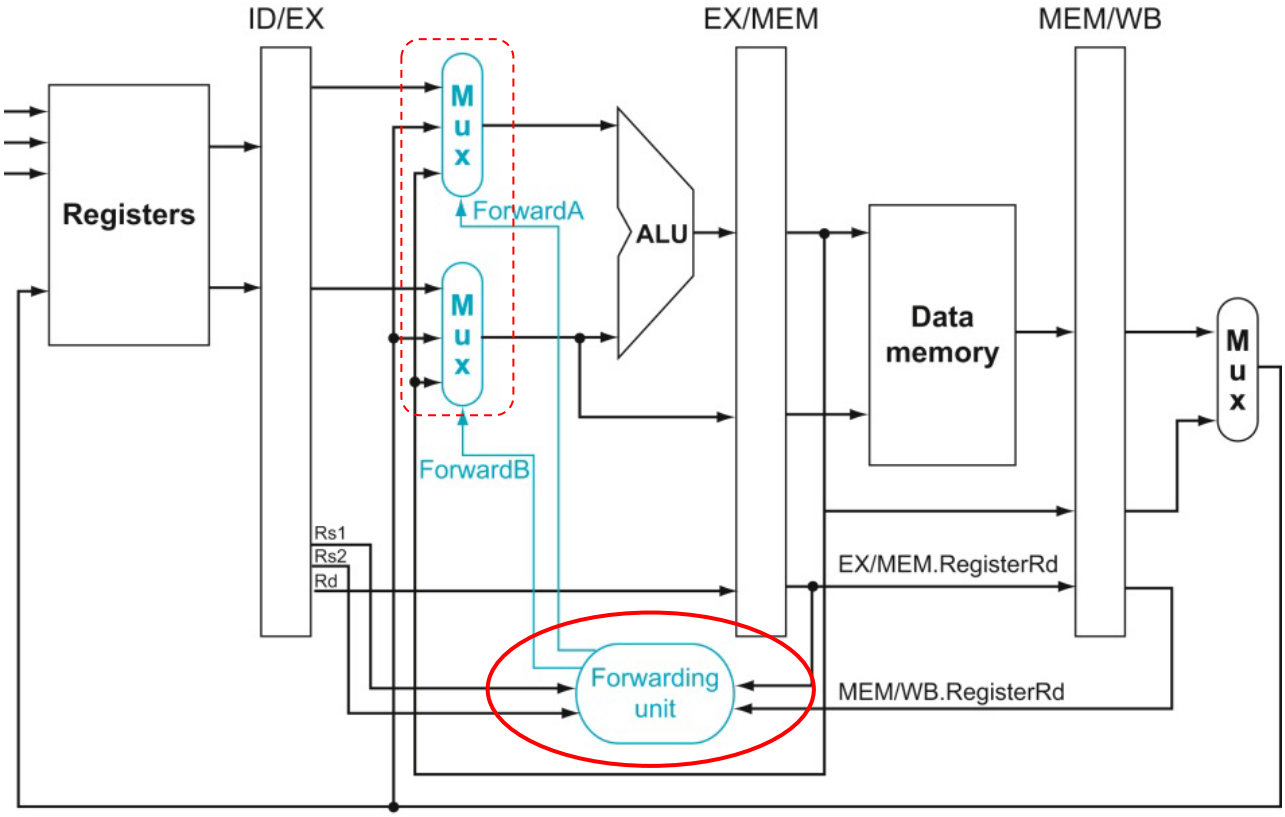

Fig 1. Detalle de la ruta de datos (datapath) para el adelantamiento de datos

En la presentación de Teoría de la asignatura y en el libro pueden encontrarse detalladas las condiciones para activar las nuevas rutas en los multiplexores de "forwarding" (adelantamiento), ha de suceder que en cualquiera de las dos

siguientes etapas ("MEM" y "WB") se esté escribiendo el mismo registro que cualquiera de los dos que fueron leídos del banco de registros en la etapa anterior para su uso en esta etapa "EX", dando precedencia a un posible resultado presente en "MEM" sobre otro en "WB" (por ser más reciente) y considerando el caso especial del registro 0.

Lo que en el dibujo se muestra como "Forwarding unit" es la generación de las señales de selección de los nuevos multiplexores (los que están dentro de la línea roja discontinua en la figura anterior), a partir de las condiciones antes comentadas. Se deberá implementar esta lógica en la misma arquitectura (esto es, sin crear ningún bloque jerárquico adicional; tanto para esta "Forwarding unit" como para los propios multiplexores pueden utilizarse procesos combinacionales explícitos o asignaciones concurrentes).

2) *Forwarding* interno en el banco de registros.

Los adelantamientos de datos implementados con el esquema anterior contemplan la ejecución en la ALU de una operación con un nuevo valor de registro disponible en los registros de pipeline EX/MEM y MEM/WB (realmente en este último caso tras un multiplexor adicional), correspondientes a una y dos instrucciones anteriores respectivamente. Un caso adicional se da cuando la dependencia es respecto a la instrucción que entró en el pipeline 3 ciclos antes. En este caso la solución es crear un adelantamiento de datos dentro del propio banco de registros (es decir en la etapa ID en lugar de en la etapa EX). Las relaciones de datos para el esquema anterior y para este caso adicional se muestran en la siguiente figura:

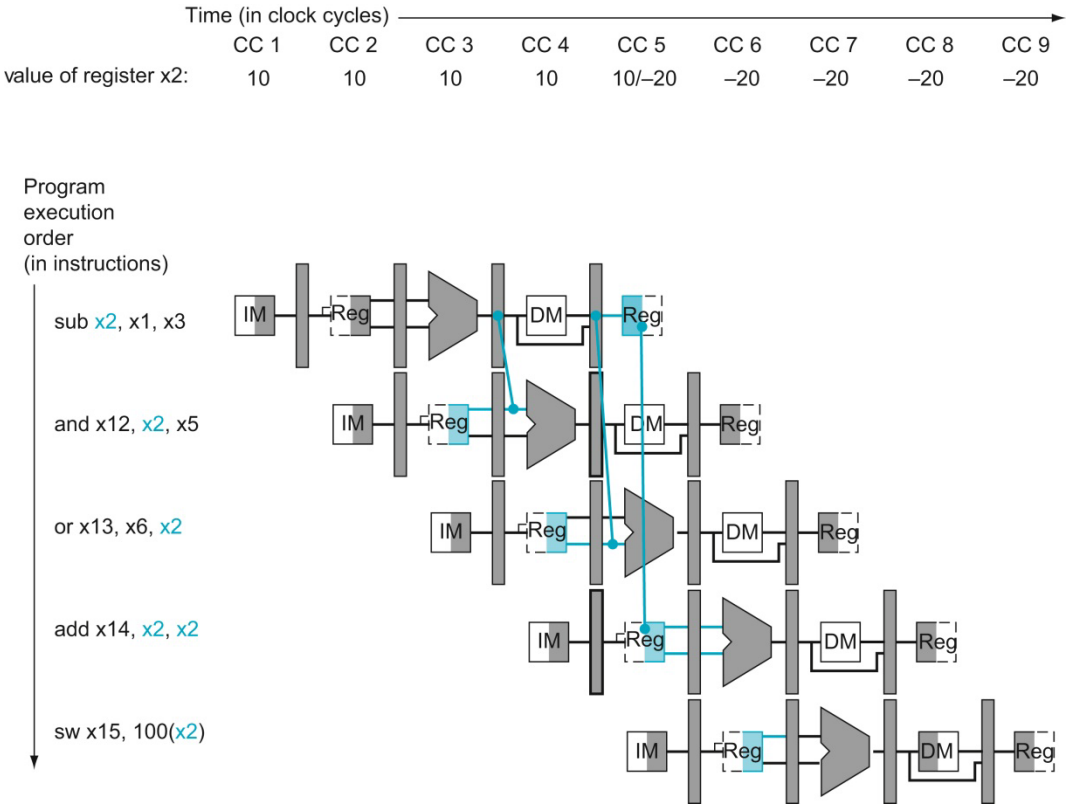

Fig 2. Posibles adelantamientos de datos desde el banco de registros

Deberá modificarse el banco de registros entregado en el material de partida de la Práctica 1 en una de las dos formas posibles:

- Añadiendo un path combinacional, de forma que, cuando se lea el mismo registro que se esté escribiendo, el banco entregue el valor que se está escribiendo en lugar del que había en el registro accedido. Además, habrá que tener en cuenta el comportamiento especial del registro 0.
- Haciendo que el banco de registros funciones en flanco de bajada. En esta alternativa debe ser capaz de explicar porque ha de funcionar.

3) Detección del caso en que una instrucción LW carga un registro que es utilizado por la instrucción que le sigue.

En este caso no es posible adelantar datos. Debe generarse un ciclo de detención ("stall"), repitiendo las etapas IF e ID actuales e insertando una "burbuja" (a modo de instrucción *nop*) en las etapas siguientes. La "Hazard detection unit" mostrada en la siguiente figura detectará la condición mencionada, evitando la actualización del contador de programa (PC - Program Counter) y del registro de pipeline IF/ID y poniendo a cero en el registro ID/EX las señales de control necesarias para crear la "burbuja":

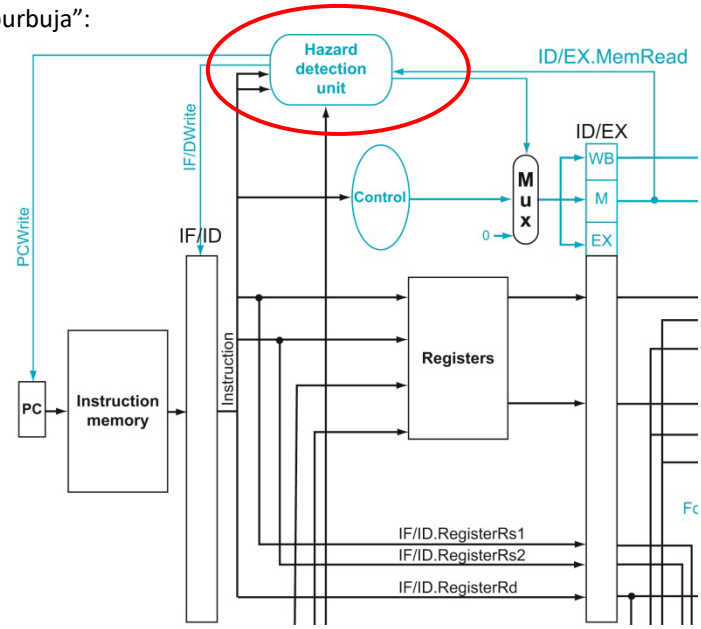

Fig 3. Ruta de datos (datapath) para la detección de riesgos LW-use

Nota 0: para determinar la condición de *load-use* hazard se usará lo visto en Teoría y expuesto en el libro de referencia, es decir, bastará observar los campos *rs1* y *rs2* de la nueva instrucción respecto al destino *rd* de una instrucción LW en el ciclo anterior. Esta solución no es óptima pues genera burbujas innecesarias para las instrucciones LUI y SW. No es necesario considerar esta peculiaridad.

Nota 1: El procesador debe ser capaz de ejecutar correctamente cualquier instrucción, excepto las instrucciones de tipo *branch* cuando se vean implicadas en riesgo de datos.

Nota 2: Es conveniente generar un código que pruebe exhaustivamente los adelantamientos. El código fuente debe ser compilado por el compilador RARS

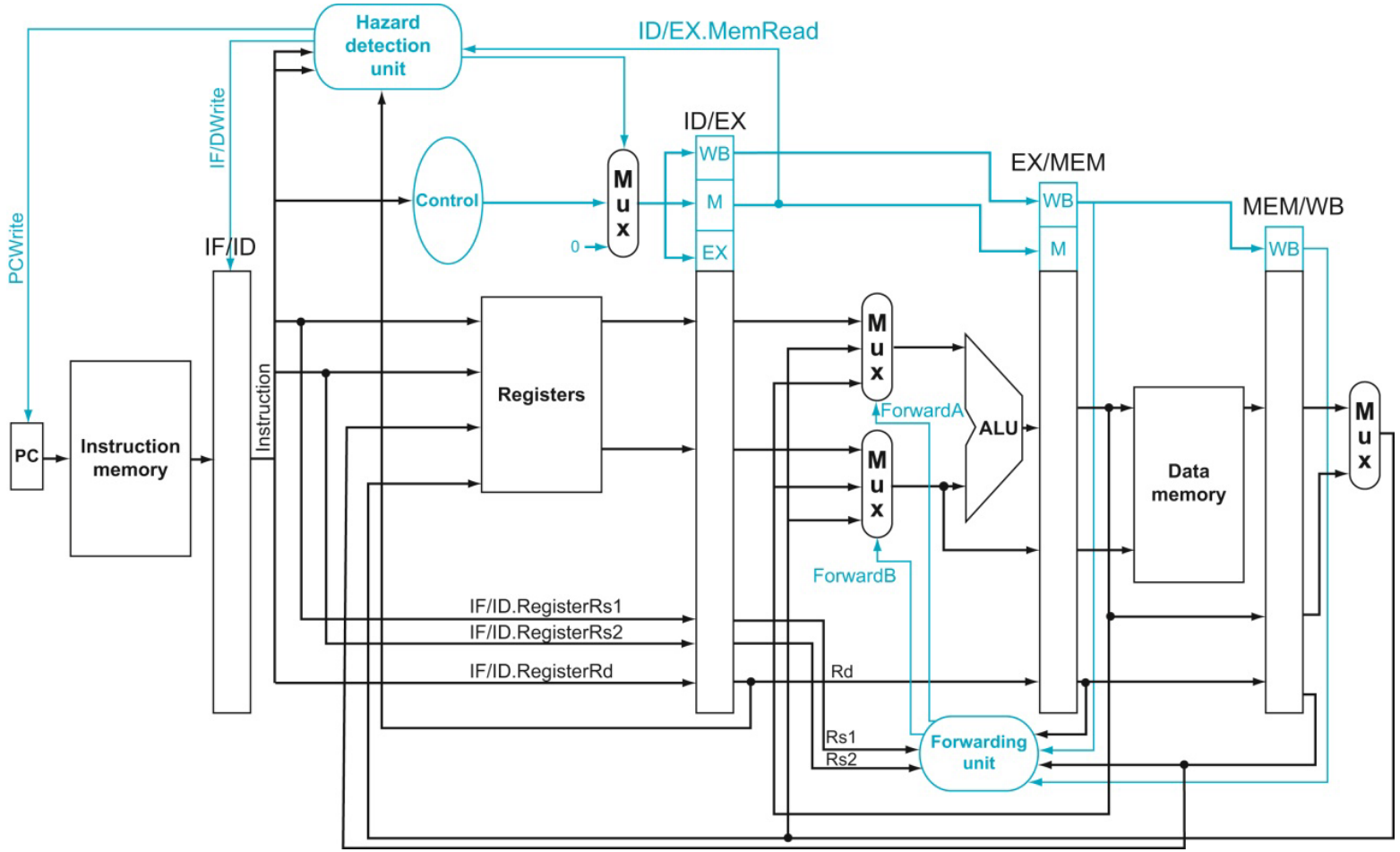

Fig 4. Datapath completo para el soporte de riesgos tipo RAW (Forwarding Unit y Hazard Detection Unit)

Ejercicio 2: Riesgos en Control (en saltos condicionales) (3 puntos)

En el ejercicio 1 no se han tenido en cuenta los riesgos producidos por las instrucciones de salto (*branch*). En este ejercicio el procesador debe ser modificado para ejecutar correctamente la instrucción *branch* ante cualquier condición previa del programa. En particular, deben realizarse modificaciones en procesador para que la instrucción *branch* se ejecute correctamente después de una operación de tipo ALU (instrucción que cargue un registro con un resultado de la ALU) así como después de una carga de memoria de datos.

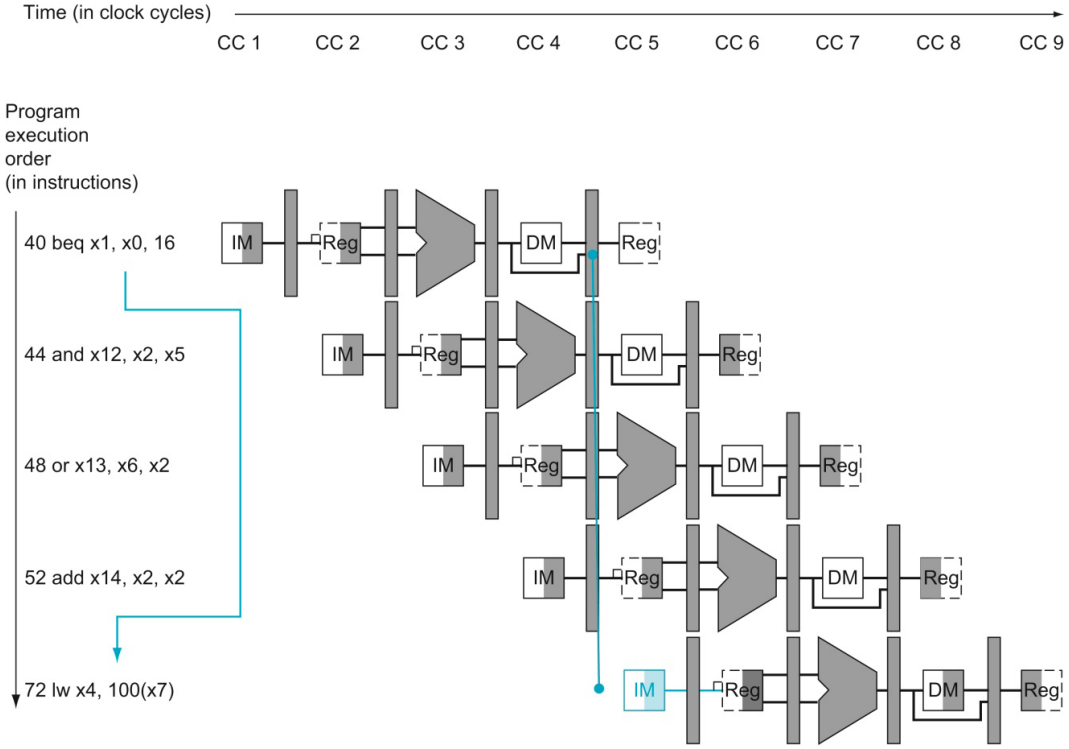

Fig 5. El impacto de las instrucciones de salto en el pipeline (cuando el salto se resuelve en MEM)

Asimismo, se solicita la entrega del código fuente de un programa (fichero en ensamblador) que sirva como comprobación del correcto funcionamiento de los saltos *branch* en estos supuestos de riesgo:

- Una instrucción tipo-R previa modifica un registro y este se usa en un *branch.* Hacer que el salto sea **efectivo**, y que sea **no efectivo**.
- Una lectura de memoria se guarda en un registro y a continuación se ejecuta un salto condiciona (branch). Hacer que el salto sea **efectivo**, y que sea **no efectivo**.

Nota del ejercicio. La implementación deberá resolver el salto en la etapa MEM (cuarta etapa). El libro de referencia sugiere adelantar el salto a la etapa de decodificación (segunda etapa), pero esto requiere realizar adelantamientos (forwarding) a la etapa DE. Usted puede debe resolverlo en la cuarta etapa reseteado los registros correspondientes.
De forma opcional se puede adelantar a la etapa DE pero como ejercicio opcional y adicional que se tendrá en cuenta para la nota, puede guiarse del simulador https://webriscv.dii.unisi.it/ para entender las condiciones del adelantamiento.

MATERIAL A ENTREGAR

- Los ficheros VHDL de los ejercicios 1 y 2.
- El código fuente del programa ensamblador de prueba del ejercicio 1 y 2.

Notas:

- La entrega se realizará a través de moodle.
- La fecha límite son las 23:59 del día viernes previo al comienzo de la siguiente práctica. Verificar la fecha en la planificación de la asignatura.
- El profesor de la práctica puede solicitar la exposición/defensa del trabajo solicitado durante la clase posterior a la entrega. Esta defensa es parte de la nota de la práctica.