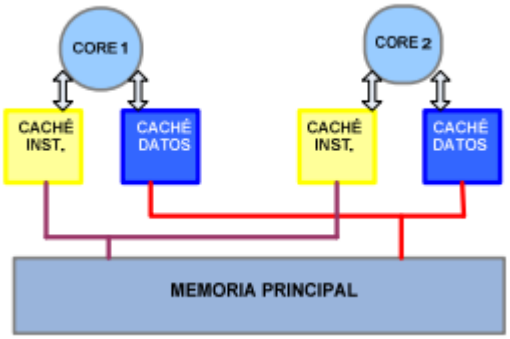
Ulises González
David Solano
Arquitectura de computadores

Situación a Simular

Se tratará de simular una computadora de arquitectura MIPS con dos procesadores. Cada uno de estos procesadores debe poseer una cache de datos y una de instrucciones. Las caches de instrucciones deben compartir un bus común para el acceso a memoria. De igual forma trabajan las caches de datos para acceder la memoria. Nótese que el sistema puede estar en simultáneo pidiéndole a memoria datos e instrucciones. El diagrama general del sistema a simular es el siguiente:



La simulación debe trabajar con un solo reloj sincronizado para los dos procesadores. De esta forma, cada procesador ejecuta una instrucción en cada ciclo del reloj. Las instrucciones que se van a procesar son instrucciones MIPS genéricas, junto con un conjunto de instrucciones que se añadieron para la implementación de semáforos. Las instrucciones que se van a ejecutar se codifican de la siguiente manera:

Subconjunto de instrucciones MIPS que se deben implementar			CODIFICACIÓN			
			1	2	3	4
Operación	**Operandos**	**Acción**	**Cód. Op.**	**Rf1**	**Rf2 ó Rd**	**Rd ó inmediato**
DADDI	RX, RY, #n	Rx <-- (Ry) + n	8	Y	X	n
DADD	RX, RY, RZ	Rx <-- (Ry) + (Rz)	32	Y	Z	X
DSUB	RX, RY, RZ	Rx <-- (Ry) - (Rz)	34	Y	Z	X
DMUL	RX, RY, RZ	Rx <-- (Ry) * (Rz)	12	Y	Z	X
DDIV	RX, RY, RZ	Rx <-- (Ry) / (Rz)	14	Y	Z	X
LW	RX, n(RY)	Rx <-- M(n + (Ry))	35	Y	X	n
SW	RX, n(RY)	M(n + (Ry)) <-- Rx	43	Y	X	n
BEQZ	RX, ETIQ	Si Rx = 0 SALTA	4	X	0	n
BNEZ	RX, ETIQ	Si Rx <> 0 SALTA	5	X	0	n
JAL	n	R31<--PC, PC<-- PC+n	3	0	0	n
JR	RX	PC <-- (Rx)	**2**	X	0	0
LL	RX, n(RY)	Rx <-- M(n + (Ry)) RL <-- n + (Ry)	**11**	Y	X	n
SC	RX, n(RY)	If RL = n + (Ry) then M(n + (Ry)) <-- Rx else Rx <-- 0	**12**	Y	X	n
FIN		Detiene el programa	**63**	0	0	0

Cada una de estas instrucciones vendrán en archivos de texto dentro de un directorio de hilos, que contendrá uno o más archivos que corresponden a hilos por ejecutar (cada archivo tiene las instrucciones línea por línea). Este directorio será proporcionado por el usuario.

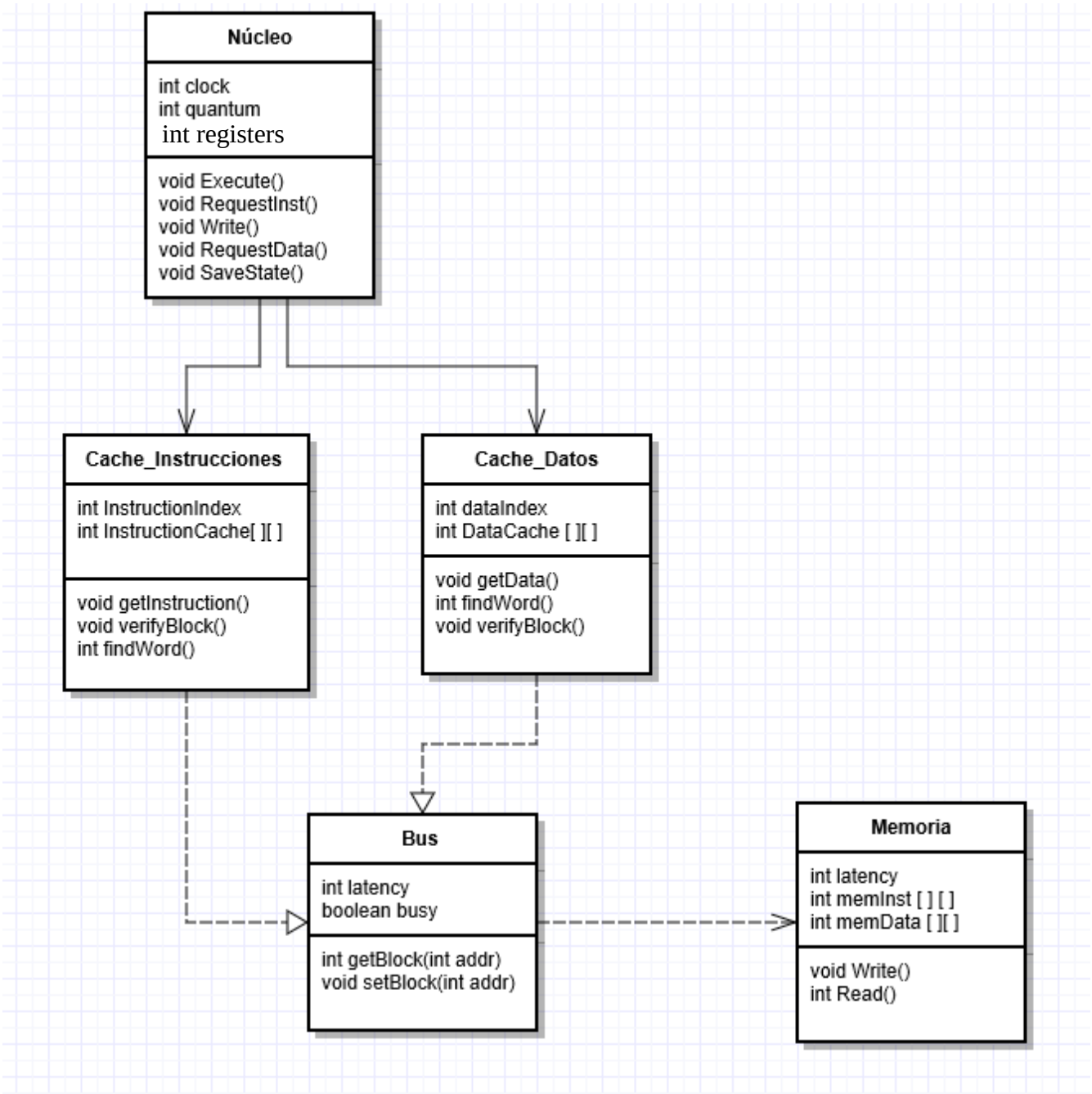
Adicionalmente, el usuario debe ser capaz de configurar varios parámetros en cada corrida. Estos son:

- Número de ciclos que se tardan leyendo o escribiendo una palabra en memoria principal.
- Número de ciclos de reloj que tarda cada bus transfiriendo una palabra.
- Valor del quantum.

Implementación de la Simulación

Para implementar esta simulación se crearon una serie de clases y estructuras de datos

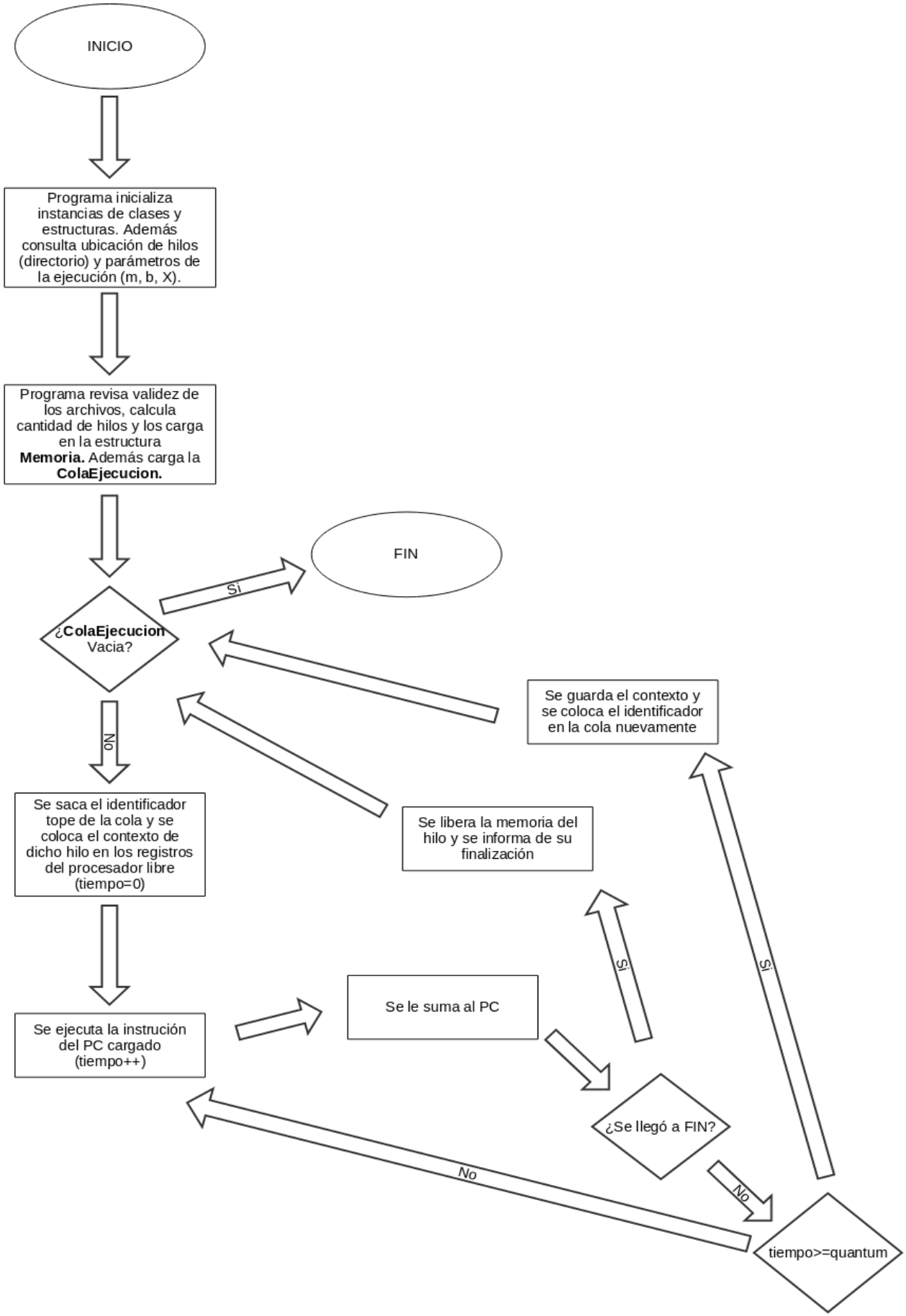
que contienen y abstraen todo lo necesario para la corrida de la simulación. Estas clases se pueden ver en el siguiente diagrama de clases:



El diagrama de clases muestra la relación jerárquica que existe dentro de la simulación. Las instancias de los núcleos se mantendrán en sincronía por medio de un reloj y manejarán el tiempo habilitado entre hilos por medio del quantum. Se cuenta con una serie de registros para almacenar las operaciones y operandos y una serie de comando que le permiten pedir más instrucciones a la caché, escribir en memoria y ejecutar instrucciones. Ambas cachés trabajan

de manera independiente; dentro de cada una se encuentra una estructura que almacena tanto instrucciones como datos para proveerlos al núcleo cuando éste los pide por medio de sus métodos. Pueden además verificar bloques dentro de su estructura y buscar palabras por medio de un índice. Las dos cachés acceden a diferentes partes de la memoria por medio del bus, el cual mantiene una bandera para señalar a quien lo pide si se encuentra o no ocupado. Tanto la memoria como el bus cuentan con una latencia para indicar el tiempo que duran realizando los accesos a direcciones, además de escribir o extraer bloques.

El diagrama de flujo básico se muestra en la siguiente página.


INICIO
Programa inicializa instancias de clases y estructuras. Además consulta ubicación de hilos (directorio) y parámetros de la ejecución (m, b, X).
Programa revisa validez de los archivos, calcula cantidad de hilos y los carga en la estructura **Memoria.** Además carga la **ColaEjecucion.**
FIN
Si
¿**ColaEjecucion** Vacia?
No
Se saca el identificador tope de la cola y se coloca el contexto de dicho hilo en los registros del procesador libre (tiempo=0)
Se guarda el contexto y se coloca el identificador en la cola nuevamente
Se libera la memoria del hilo y se informa de su finalización
Se ejecuta la instrución del PC cargado (tiempo++)
Se le suma al PC
Si
Si
¿Se llegó a FIN?
No
No
tiempo>=quantum

El flujo básico del programa consiste en pedir los datos y simplemente empezar a ejecutar los hilos del directorio (archivos), asignándole a cada hilo el quantum de tiempo de procesador, dependiendo de cada procesador y aplicando round robin sobre la totalidad de hilos, hasta que estos vayan terminando de ejecutarse. Una vez que todos los hilos se terminan de ejecutar, el programa muestra el resumen de los resultados de la corrida y termina. Nótese que el diagrama anterior ejecuta un hilo principal o de control, pero luego del paso #3, empieza a utilizar paralelismo para hacer la simulación más real (simula paralelismo de ambos procesadores). De esta forma, el programa debería bifurcar la ejecución de cada instrucción que se vaya ejecutando. Además de realizar una sincronización de los dos hilos que se ejecutan en cada instante (la arquitectura tiene un solo reloj para los dos procesadores).

Es importante destacar que inicialmente no se va a trabajar con los retrasos ocasionados por ir a memoria principal. Esto hace que ambos procesadores sincronizar cada instrucción MIPS, estos en realidad vayan ejecutando la misma cantidad de instrucciones. Más adelante se deberá tomar en cuenta que los accesos a memoria generan retrasos en los que el procesador no puede ejecutar otra cosa ya que debe esperar un resultado de su pedido a memoria.

En la siguiente página se muestra un cuadro que condensa los casos en los que incurren las operaciones de acceso a la caché de datos y a la memoria principal. Cada uno de los casos desglosa a su derecha la acción que se toma dentro del programa en cuestión, los elementos que utiliza, el momento en que se realiza la acción, su consecuencia en la simulación y las implicaciones de la sincronización.

NOTA: El cuadro debe ser considerado en conjunto con los diagramas de flujo presentados luego del mismo, ya que en él se especifican los momentos en donde se llevan a cabo las implementaciones de los casos dentro del programa. Además, cabe destacar que los diagramas presentados son las versiones corregidas a aquellas presentadas en la segunda entrega del trabajo.

	Caso	Acción	Elementos	Momento	Consecuencia	Sincronización
LW	Caché hit Compartido	Lee el dato	Caché propia	Mismo ciclo	NA	Bloquea la propia caché
LW	Caché hit Modificado	Lee el dato	Caché propia	Mismo ciclo	NA	Bloquea la propia caché
LW	Caché fallo	Pone mensaje de fallo de lectura en bus	Bus, caché propia, caché del otro núcleo	En el final del ciclo, puede esperar	Revisa si la otra caché tiene el bloque, si no busca en memoria	Bloquea la otra caché y bloquea el bus. Puede tener que esperar al siguiente ciclo por recursos
LW	No tiene caché	Espera a que la caché se desocupe	Caché propia	Mismo ciclo, puede esperar	NA	NA
SW	Caché hit Compartido	Pide bus e invalida el bloque de la otra caché (de ser necesario), modifica su bloque y lo marca como M	Caché propia, bus, otras cachés que contienen el bloque	Al final del ciclo para evitar lecturas sucias	El otro núcleo no tiene acceso al bloque modificado (se invalida)	Esperar por disponibilidad del bus y cachés, puede tener que intentarlo de nuevo al ciclo siguiente.
SW	Caché hit	Escribe en	Caché	Mismo	NA	NA

	Modificado	el bloque libremente	propia	ciclo		
SW	Caché fallo	Pone mensaje de fallo de escritura en bus	Bus, caché propia, caché de otros núcleos	Mismo ciclo, puede esperar por recursos.	Si lo tenía la otra caché, lee el bloque, lo invalida en las demás cachés y lo coloca modificado en su caché, sino lo lee de memoria.	Esperar por disponibilidad de bus y/o cachés. Puede tener que intentarlo al siguiente ciclo.
SW	No tiene caché	Espera	Caché propia	Mismo ciclo	NA	NA
LL		Pone en RL la dirección	Caché propia	Mismo ciclo	NA	NA
SC		Lee de RL la dirección, si es la misma coloca un 1 en memoria, sino un 0.	Caché propia	Mismo ciclo	Bloquea el candado	El candado debe de estar libre para poderse ejecutar

Diagrama de flujo de la operación LoadWord:

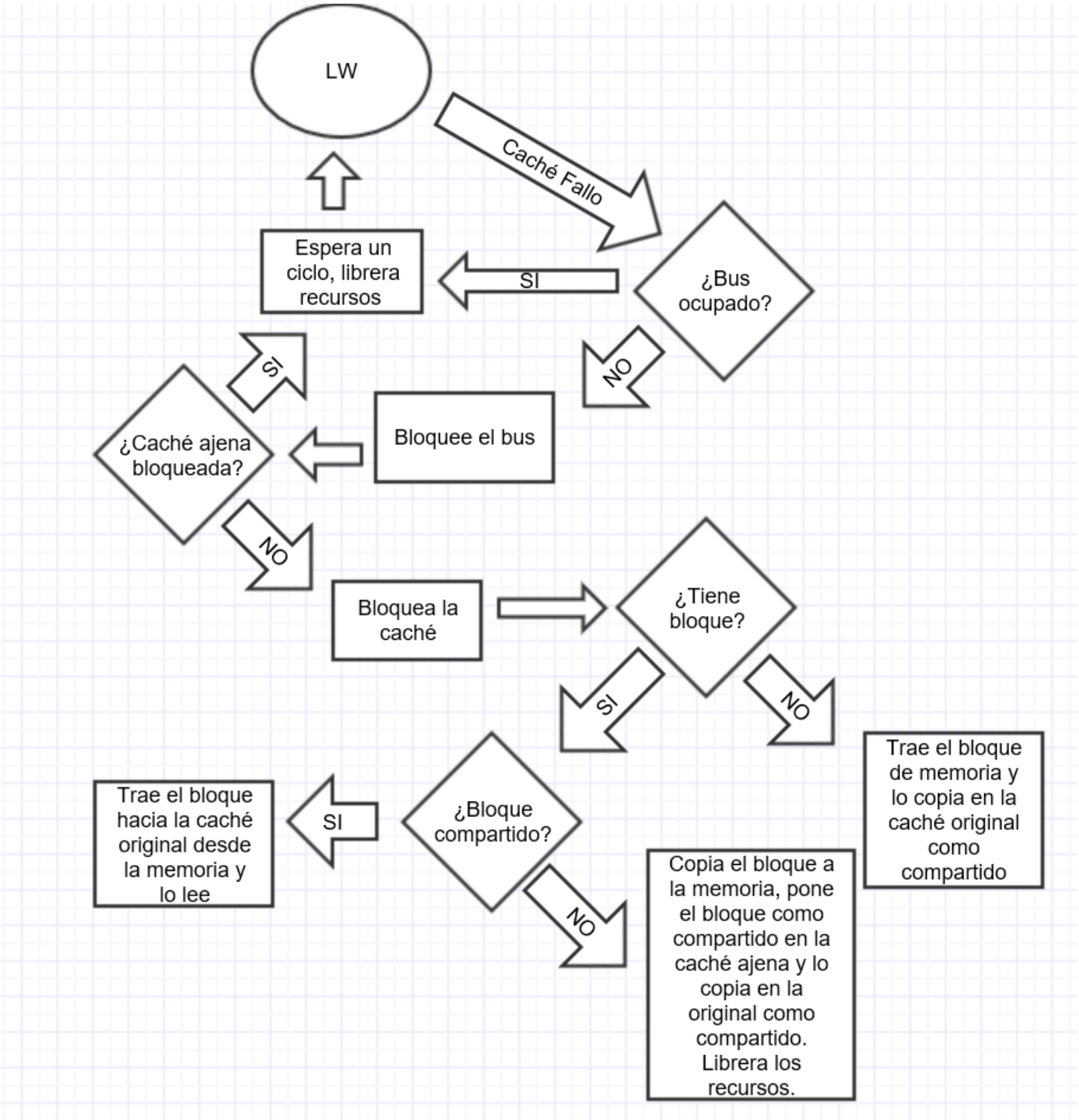
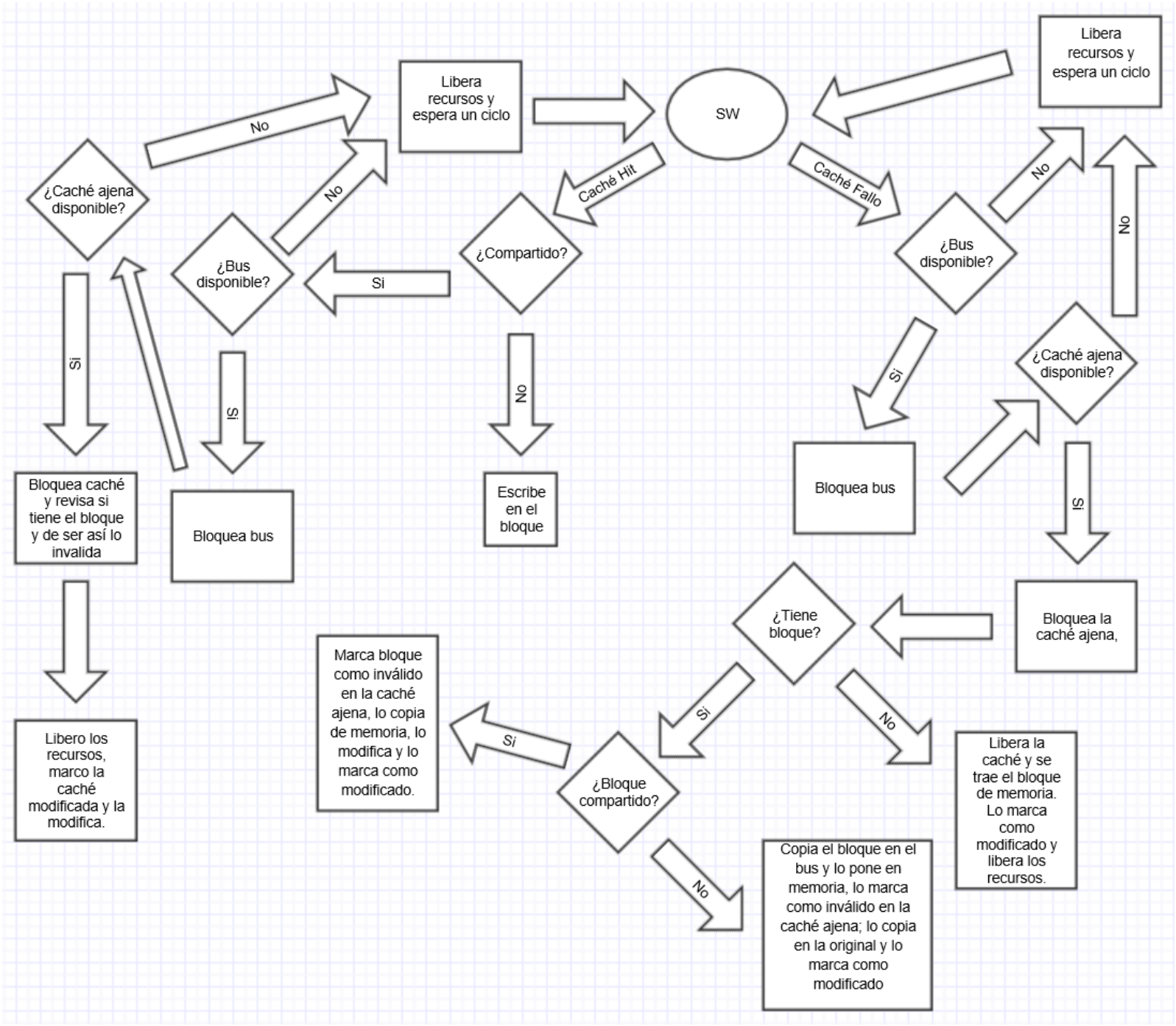


Diagrama de flujo de la operación StoreWord:



Problemas sin resolver

En un número aleatorio y no constante de veces en las que se ejecuta la simulación, es posible que el programa irrespete las latencias del bus y memoria indicadas al correr al inicio de la misma. Es importante destacar que este problema no ocurre siempre, pero en caso de darse, los núcleos no respetan la espera indicada y actúan prematuramente, ignorando en cierta medida los valores indicados.

Posibles soluciones: verificar la latencia en cada tick de la simulación desde el *main* para obligar a que ésta sea respetada.