

TRABAJO PRACTICO DE ARQUITECTURA DE COMPUTADORAS

ALGORITMO DE TOMASULO

Profesor: Ing. Orlando Micolini

Asignatura: Arquitectura de Computadoras

Integrantes:

Aguilar, Mauricio

Tarazi, Pedro Esequiel

22 de octubre de 2018

Contenido

INTRODUCCIÓN	3
DESARROLLO	4
DIAGRAMAS	5
CASOS DE PRUEBA.....	10
CONCLUSIÓN.....	15
BIBLIOGRAFÍA	16
ANEXO.....	17
PROGRAMA 1	17
PROGRAMA 2	17
PROGRAMA 3	18
PROGRAMA 4	18

INTRODUCCIÓN

El Algoritmo de Tomasulo es un algoritmo de planificación dinámica, diseñado para permitir a un procesador ejecutar instrucciones fuera de orden. Este algoritmo resuelve los riesgos WAW y WAR realizando un renombrado, lo que permite que las instrucciones se lancen. Además utiliza un bus de datos común en el que los valores calculados son enviados a todas las estaciones de reserva que lo necesiten. Esto permite mejorar la ejecución paralela de instrucciones en situaciones en las que otros algoritmos fallarían y provocarían una detención.

En la actualidad, gran parte de los procesadores hacen uso de variaciones de este algoritmo para la planificación dinámica de instrucciones.

En este trabajo se realizó la implementación del Algoritmo de Tomasulo en lenguaje de programación Java a fin de entender su funcionamiento. El algoritmo debe incluir un buffer de reordenamiento, conocido como ROB (Re-Order Buffer). Esto permitirá que las instrucciones ingresen en orden, se ejecuten fuera de orden, y salgan del sistema nuevamente en orden.

Se implementaron las siguientes partes:

- Unidad Funcional Sumador
- Estaciones de Reserva para el Sumador
- Unidad Funcional Multiplicador
- Estaciones de Reserva para el Multiplicado
- Unidad Funcional Cargador (Loader)
- Estaciones de Reserva para el Cargador
- Common Data Bus
- Re-Order Buffer
- Banco de Registros
- Banco de Memoria

DESARROLLO

El Algoritmo de Tomasulo con buffer de reordenamiento consta de las siguientes etapas:

1. Issue:

- Obtiene una instrucción de la cola.
- Despacha la instrucción si hay libre una estación de reserva y una entrada en el ROB.
- Envía los operandos a la estación de reserva si ellos están disponibles en los registros o en el ROB.
- El número de entrada del ROB utilizado es también enviado a la estación de reserva, el cual será usado cuando se necesite indicar el tag en el CDB.
- Si todas las estaciones de reserva y/o las entradas del ROB están llenas, entonces el despacho de la instrucción es detenido hasta que ambos tengan entradas disponibles.

2. Execute:

- Si uno o más de los operandos no está disponible, las estaciones de reservas monitorean el CDB en busca de valores que le puedan servir a esa estación.
- Cuando ambos operandos están disponibles en una estación de reserva, se ejecuta la operación.
- Las instrucciones pueden tener multiples ciclos de reloj. Por ejemplo:
 - ADD: 3 clocks
 - MUL: 5 clocks
 - LOAD: 2 clocks

3. Write Result:

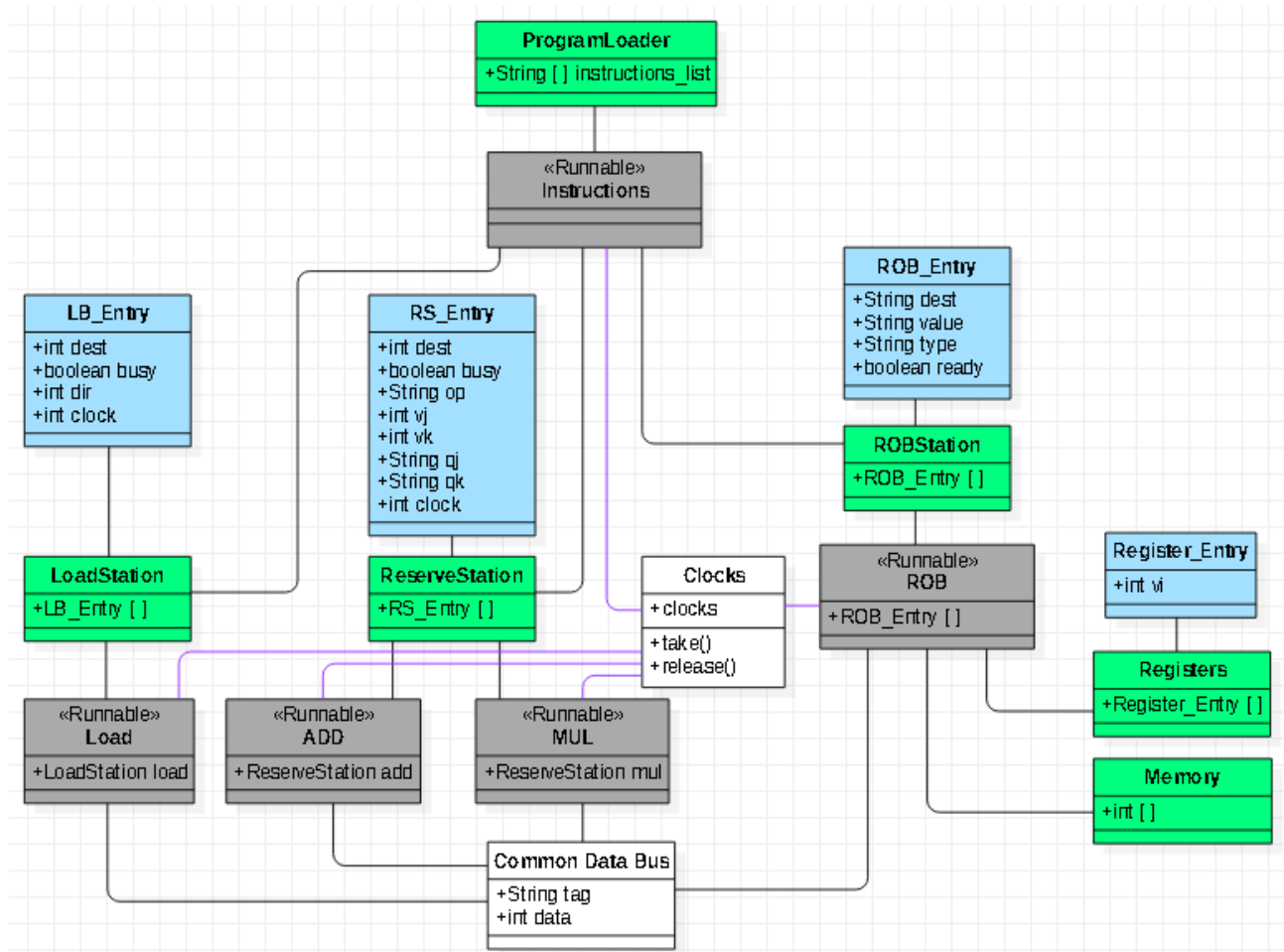
- Cuando el resultado está disponible, escribe en el CDB para el ROB.
- El ROB lee el CDB y determina a que entrada pertenece ese dato.
- Se marca esa entrada del ROB como Ready, para indicar que ya puede escribirse en memoria o en los registros.

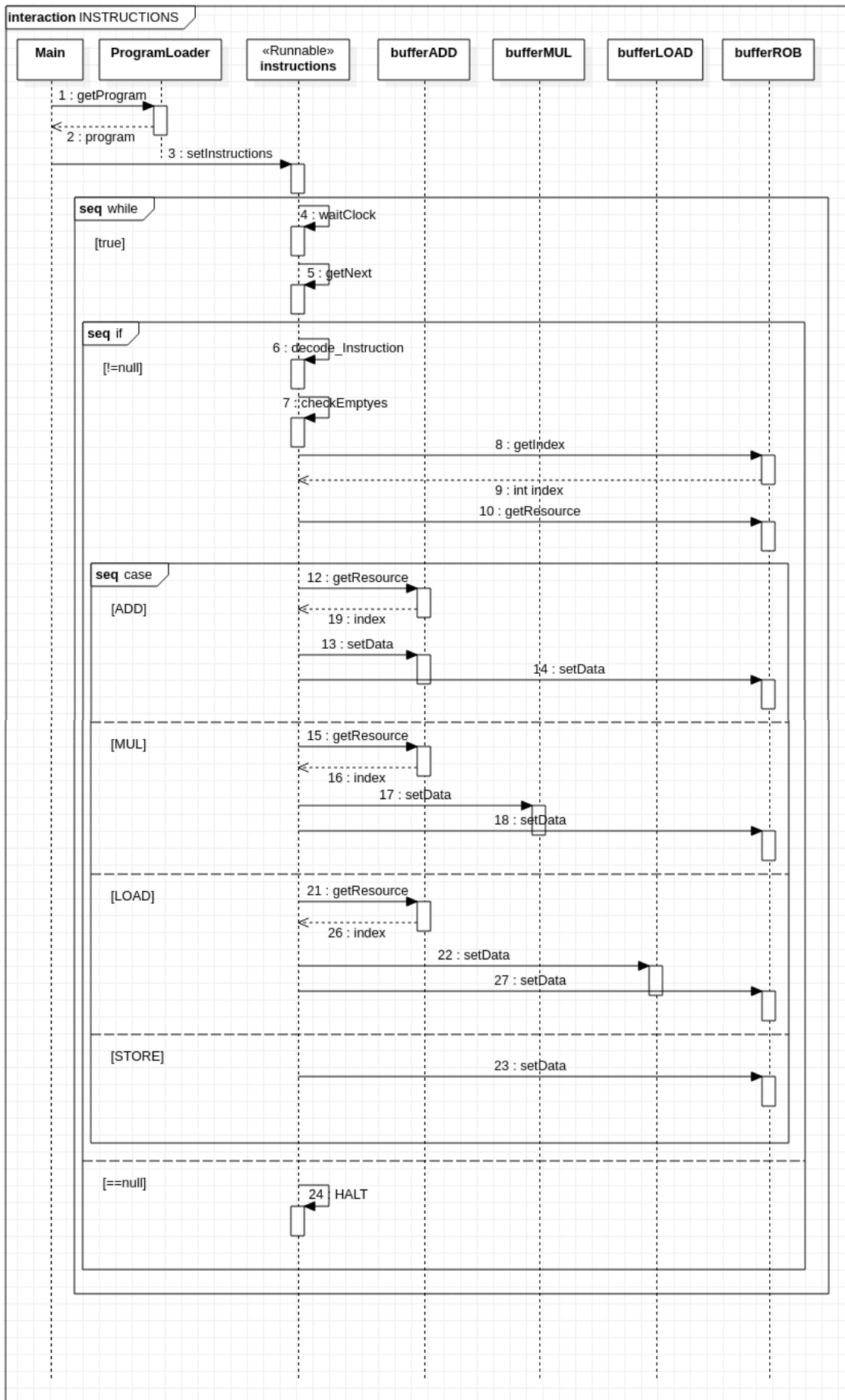
4. Commit:

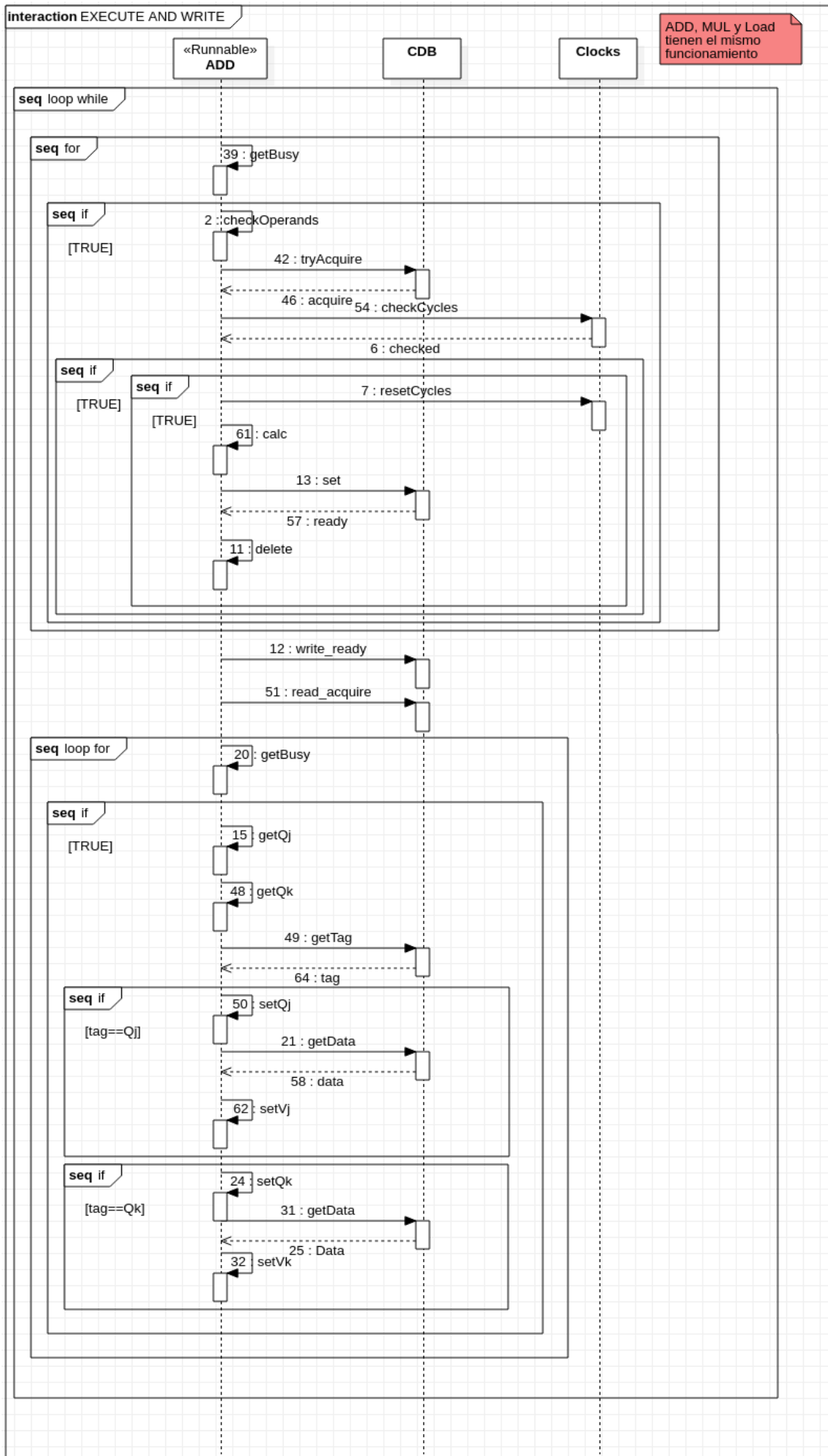
- Cuando la instrucción en la cabeza del ROB tiene su resultado presente (Ready), entonces esta instrucción esta lista para escribirse en memoria o registros.
- Se escribe su resultado, se limpia la entrada y la cabecera avanza un lugar.

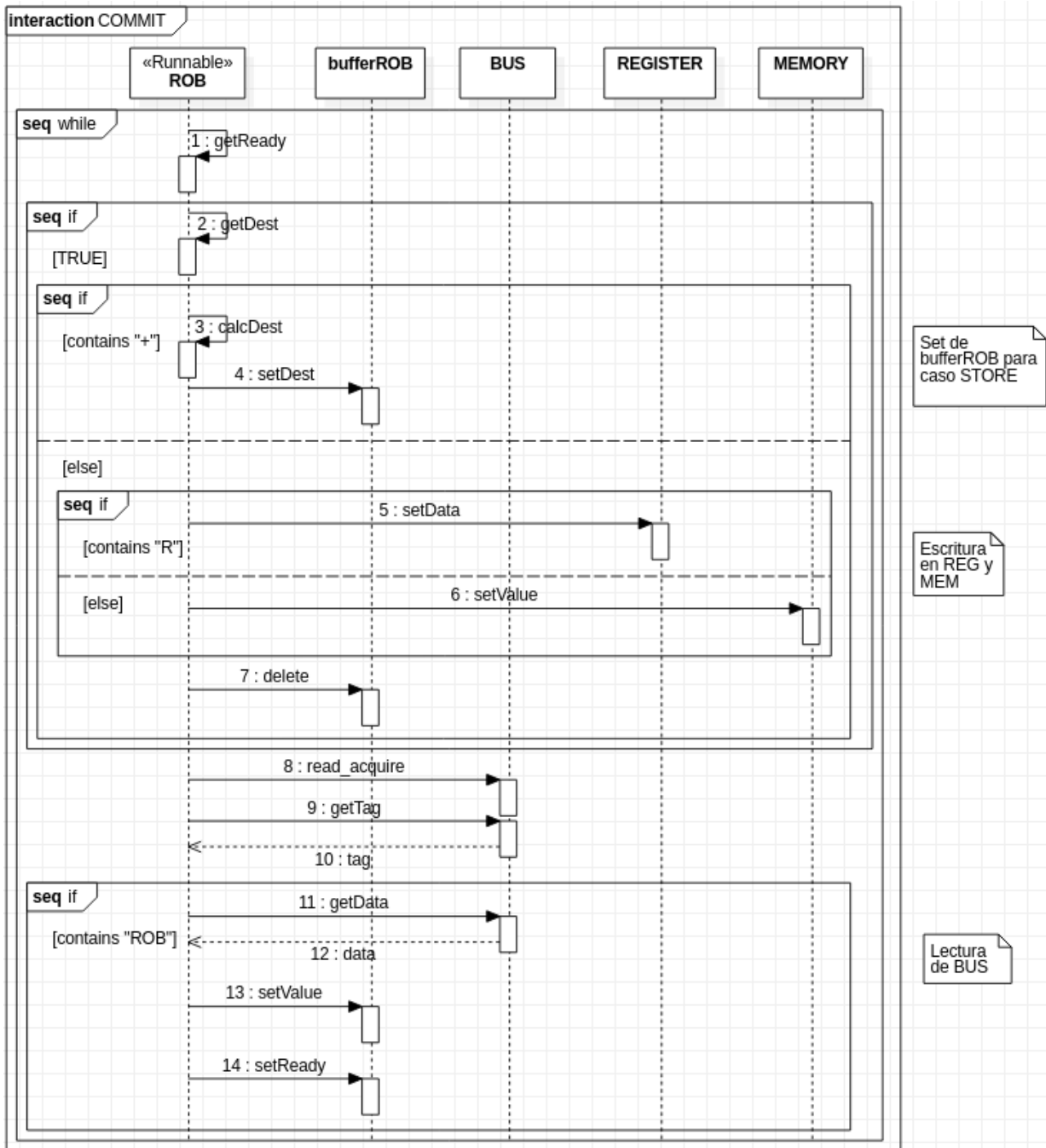
DIAGRAMAS

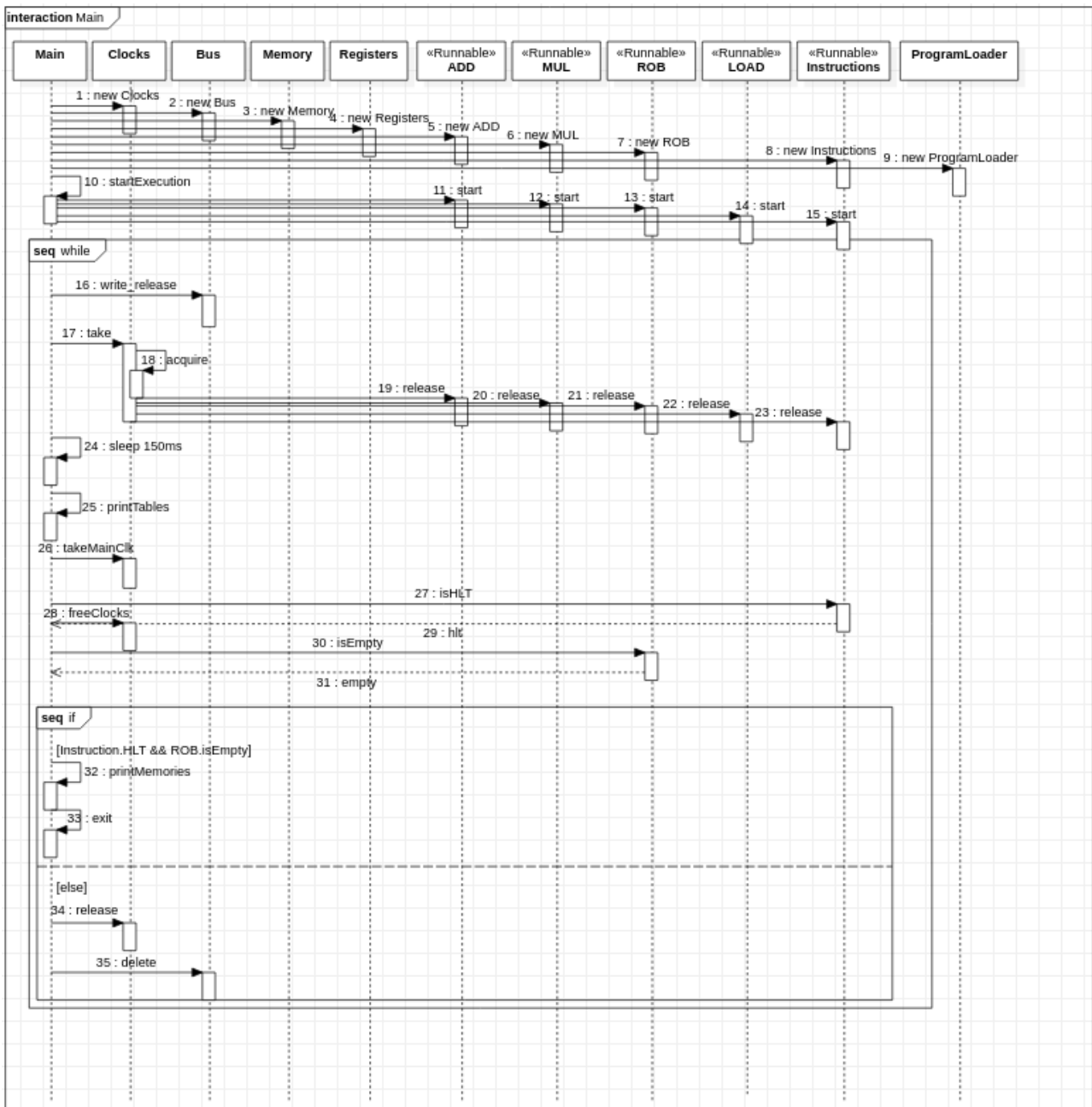
Luego de estudiar su funcionamiento, se procedió a elaborar un diagrama de clases y un diagrama de secuencias iniciales, como bosquejo inicial a modo de entender cuál es la mejor manera de implementarlo, y descubrir cuestiones que no aparecían en primera instancia.







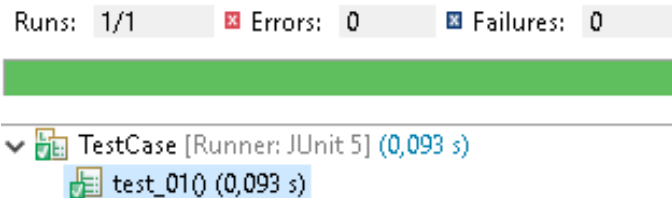






CASOS DE PRUEBA


Se realizaron tres test de comprobación del funcionamiento del Re-Order Buffer (ROB):


ID_Test	Test_01
Nombre del Test	Carga de instrucción en ROB
Requerimiento	La instrucción debe cargarse en el ROB siempre y cuando haya lugar disponible en él y en la estación de reserva (ER).
Precondiciones	Se debe crear un archivo el cual contiene las instrucciones del programa a ejecutar. El mismo debe llamarse “program#.txt”. El archivo debe contener una instrucción por línea. Las instrucciones son del tipo: ADD R1 R2 R3
Secuencia	Ciclo 1: “Instructions” lee la primera instrucción del buffer. Luego verifica si hay lugar en el ROB y en la ER correspondiente. Si hay lugar, se carga la instrucción.
Resultado Esperado	La instrucción debe estar cargada en el buffer de ROB y en el buffer de la ER correspondiente.
Resultado Obtenido	La instrucción se cargó en el ROB y en la ER
Estado del TEST	EXITOSO




ID_Test	Test_02
Nombre del Test	Bloqueo de carga en ROB por buffer lleno
Requerimiento	Si no hay lugar en el buffer del ROB, se debe detener la carga de instrucciones hasta que se desocupe un lugar en él.
Precondiciones	Se debe crear un archivo el cual contiene las instrucciones del programa a ejecutar. El mismo debe llamarse "program#.txt". El archivo debe contener una instrucción por línea. Las instrucciones son del tipo: ADD R1 R2 R3
Secuencia	Ciclo 1. "Instructions" lee la primera instrucción del buffer. Es un ADD. Verifica si hay lugar en el ROB y en ADD. Tiene lugar, carga la instrucción y toma la siguiente. Ciclo 2: "Instructions" lee la segunda instrucción del buffer. Es un LD. Como ROB no tiene lugar, detiene la carga hasta que se libere. Ciclos 2,3 y 4: ADD se ejecuta. Ciclo 5: Luego de que la primera instrucción se escribió en el registro correspondiente, se libera el lugar en ROB. Ciclo 6: Se carga la instrucción LD en ROB y en LOAD.
Resultado Esperado	La carga de instrucciones en debe bloquearse a la espera de lugar disponible.
Resultado Obtenido	La carga de instrucciones se bloqueó.
Estado del TEST	Exitoso

Runs: 1/1  Errors: 0  Failures: 0




▼  TestCase [Runner: JUnit 5] (0,653 s)


 test_02() (0,653 s)

ID_Test	Test_03
Nombre del Test	ROB escribe en REG
Requerimiento	Cuando ROB lee una de sus entradas como Ready, escribe en el registro correspondiente.
Precondiciones	Se debe crear un archivo el cual contiene las instrucciones del programa a ejecutar. El mismo debe llamarse "program#.txt". El archivo debe contener una instrucción por línea. Las instrucciones son del tipo: ADD R1 R2 R3
Secuencia	Ciclo 1: Se lee una instrucción. Si hay lugar, se carga en ROB y en la ER correspondiente. Ciclo 2, 3 y 4: se ejecuta la instrucción. Ciclo 5: Se escribe en CDB. ROB lee CDB, carga valor en la entrada correspondiente, pone flag Ready a True. Ciclo 6: Si la entrada se encuentra en la cabeza de la cola, se escribe en Registro que corresponda.
Resultado Esperado	R0 = 3
Resultado Obtenido	R0 = 3
Estado del TEST	Exitoso

Runs: 1/1 ✖ Errors: 0 ❌ Failures: 0



██

▼  TestCase [Runner: JUnit 5] (0,763 s)

 test_03() (0,763 s)

ID_Test	Test_04
Nombre del Test	ROB escribe en orden
Requerimiento	ROB debe escribir las instrucciones en Registro y/o Memoria en el orden en el que fueron despachadas.
Precondiciones	Se debe crear un archivo el cual contiene las instrucciones del programa a ejecutar. El mismo debe llamarse "program#.txt". El archivo debe contener una instrucción por línea. Las instrucciones son del tipo: ADD R1 R2 R3
Secuencia	<p>Ciclo 1: Se lee una instrucción ADD. Si hay lugar, se carga en ROB y en la ER correspondiente.</p> <p>Ciclo 2: Se ejecuta un ciclo de ADD. Se lee una instrucción MUL, la cual depende del resultado de la instrucción anterior.</p> <p>Ciclo 3: Se ejecuta un ciclo de ADD. Se lee una instrucción LOAD, que no depende de ninguna de las anteriores.</p> <p>Ciclo 4: Se ejecuta un ciclo de ADD y uno de LD.</p> <p>Ciclo 5: ADD finaliza su ejecución y se escribe en CDB. ROB y MUL leen el bus y actualizan sus tablas con el valor de ADD. LD ejecuta su segundo ciclo. ADD listo para escribirse en Registers.</p> <p>Ciclo 6: MUL ejecuta un ciclo. LD escribe en CDB. ROB escribe resultado de ADD en Registers y elimina la entrada. Luego lee el bus y actualiza su tabla. LD listo para escribirse en Registers.</p> <p>Ciclo 7, 8, 9, y 10: MUL ejecuta un ciclo.</p> <p>Ciclo 11: MUL escribe en CDB. ROB lee CDB y actualiza su tabla.</p> <p>Ciclo 12: El resultado de MUL es escrito en Registers, que se encuentra en la cabecera del ROB.</p> <p>Ciclo 13: LD finalmente llega a la cabecera de ROB, el cual escribe el resultado en Registers.</p>
Resultado Esperado	Las instrucciones finalicen en orden.
Resultado Obtenido	Las instrucciones finalizaron en orden.
Estado del TEST	Exitoso

Runs: 1/1  Errors: 0  Failures: 0

 TestCase [Runner: JUnit 5] (1,513 s)
 test_040 (1,513 s)

ID_Test	Test_05
Nombre del Test	Resultados correctos
Requerimiento	Los programas deben finalizar con resultados correctos.
Precondiciones	Se debe crear un archivo el cual contiene las instrucciones del programa a ejecutar. El mismo debe llamarse "program#.txt". El archivo debe contener una instrucción por línea. Las instrucciones son del tipo: ADD R1 R2 R3
Secuencia	Se carga un programa completo y se ejecuta hasta finalizar.
Resultado Esperado	Los registros y memorias deben tener resultado correctos
Resultado Obtenido	Los registros y memorias tienen resultados correctos.
Estado del TEST	Exitoso

Runs: 1/1 Errors: 0 Failures: 0

▼ TestCase [Runner: JUnit 5] (5,293 s)

 test_05() (5,293 s)

CONCLUSIÓN

Se desarrolló el Algoritmo de Tomasulo con Buffer de Reordenamiento. La ejecución fuera de orden es una de sus principales ventajas, ya que permite que el procesador pueda seguir ejecutando aquellas instrucciones que no sean dependientes de sus antecesoras y que no se encuentren en la ventana de instrucciones, es decir, actualmente en ejecución en el procesador. Entonces puede intercalar instrucciones y aprovechar casi al máximo al procesador. El procesamiento de instrucciones solo se detiene si no hay una estación de reserva o una entrada en el ROB disponible para la próxima instrucción. Por lo tanto haciendo ajustes en el tamaño del ROB podemos ver que con una cantidad de entrada igual a la suma de las cantidades de estaciones de reserva, el sistema funciona sin detenciones.

En cuanto al Common Data Bus, se ha implementado con un semáforo, y por tanto su política es la siguiente: Cualquiera de las unidades funciones que necesite escribir en el CDB, compite por el semáforo, y aquella que lo gane podrá escribir. Para evitar que haya inanición sobre las estaciones de reserva de una misma unidad funcional, se implementó la ejecución de modo de darle prioridad a estas en un orden Round-Robin. No se ha implementado una política para evitar la inanición de una unidad funcional, ya que solo son tres, sus instrucciones se ejecutan en distintas cantidades de clocks y el sistema de renombrado hace que haya un orden implícito entre ciertas instrucciones por lo cual sería lógico pensar que si solo hay tres estaciones de reserva en las unidades de suma y multiplicación, estas quedarán ociosas rápidamente luego de algunas escrituras en el buffer.

Mediante la realización de este trabajo logramos comprender el funcionamiento de un procesador con el Algoritmo de Tomasulo, así como sus ventajas y desventajas.

BIBLIOGRAFÍA

- Computer Architecture: A Quantitative Approach 5th Edition by John L. Hennessy (Author), David A. Patterson (Author)
- Tema 3 – ILP, Planificación Dinámica, Predicción de Saltos, Especulación - Arquitectura de Computadoras – Orlando Micolini

ANEXO

PROGRAMA 1

ADD R0 R1 R2

LD R1 1 R2

ADD R2 R1 R4

ST 1 R4 R1

MUL R4 R6 R2

ADD R3 R4 R5

Resultado:

R0 = 3 M0 = 0

R1 = 3 M1 = 1

R2 = 7 M2 = 2

R3 = 47 M3 = 3

R4 = 42 M4 = 4

R5 = 5 M5 = 3

R6 = 6 M6 = 6

R7 = 7 M7 = 7

R8 = 8 M8 = 8

PROGRAMA 2

ADD R0 R1 R2

ADD R1 R2 R3

ADD R2 R3 R4

ADD R4 R5 R6

ADD R5 R6 R7

Resultado:

R0 = 3 M0 = 0

R1 = 5 M1 = 1

R2 = 7 M2 = 2

R3 = 3 M3 = 3

R4 = 11 M4 = 4

R5 = 13 M5 = 5

R6 = 6 M6 = 6

R7 = 7 M7 = 7

R8 = 8 M8 = 8

PROGRAMA 3

```
ADD R0 R1 R2
LD R1 1 R0
ADD R2 R1 R4
ST 1 R4 R1
MUL R4 R6 R2
ADD R3 R4 R5
```

Resultado:

R0 = 3	M0 = 0
R1 = 4	M1 = 1
R2 = 8	M2 = 2
R3 = 53	M3 = 3
R4 = 48	M4 = 4
R5 = 5	M5 = 4
R6 = 6	M6 = 6
R7 = 7	M7 = 7
R8 = 8	M8 = 8

PROGRAMA 4

```
LD R1 1 R4
LD R2 0 R3
ADD R4 R0 R2
LD R0 0 R1
ADD R5 R0 R2
MUL R5 R1 R2
MUL R6 R2 R3
MUL R7 R3 R4
MUL R8 R4 R5
ST 0 R0 R7
ST 0 R2 R8
```

Resultado:

R0 = 5	R6 = 9	M0 = 0	M6 = 6
R1 = 5	R7 = 9	M1 = 1	M7 = 7
R2 = 3	R8 = 45	M2 = 2	M8 = 8
R3 = 3		M3 = 45	
R4 = 3		M4 = 4	
R5 = 15		M5 = 9	