



Instituto Tecnológico de Costa Rica

Arquitectura de Computadores II

**Test Plan
del
Simulador de coherencia de caché**

Grupo 4

David De la Hoz Aguirre
Kenichi Hayakawa Bolaños
Daniela Brenes Otárola
Oscar Méndez Granados

Tabla de Contenidos

Tabla de Contenidos.....	2
Pruebas de funcionalidad con la GUI.....	3
Ejecución MESI/MOESI.....	3
Selección de protocolo.....	3
Generación automática de instrucciones.....	3
Llenado de la memoria con valores aleatorios.....	4
Ejecución automática de instrucciones en un processing element.....	4
Ejecución manual de instrucciones en un processing element.....	5
Ejecución automática de instrucciones de todos los processing element.....	5
Generación de reportes.....	6
Unit Tests.....	7

Pruebas de funcionalidad con la GUI

A continuación se presentan pruebas de usuario donde se demuestra la funcionalidad de la aplicación según el protocolo, ya sea MESI o MOESI, además se presenta la funcionalidad de cada uno de los modos de ejecución, manual o automático para cada processing element, o bien, la ejecución de todas las instrucciones de todos los processing elements al mismo momento.

Ejecución MESI/MOESI

Selección de protocolo

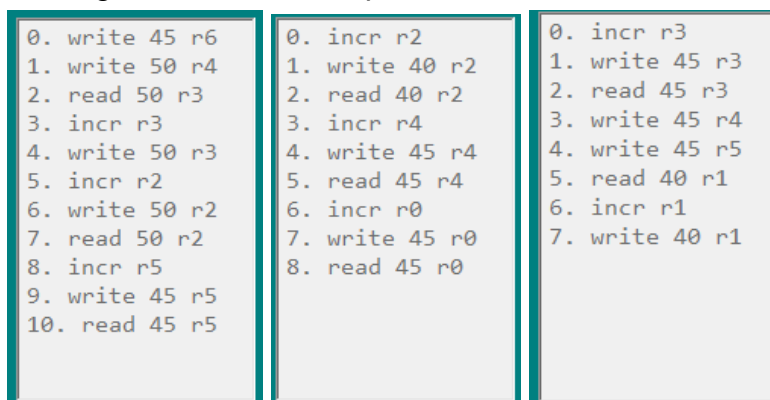
En esta prueba se buscaba seleccionar uno de los dos protocolos y demostrar su cambio en la interfaz, haciéndole entender al usuario final cuál protocolo va a ejecutar.



En el caso anterior se verificó que el cambio de un protocolo de coherencia de memoria, en este caso como se está evaluando el funcionamiento de MESI, se selecciona este.

Generación automática de instrucciones

Seguidamente se prueba la funcionalidad de generación automática de instrucciones para los tres processing element con los que cuenta el sistema.



En las capturas de pantalla anteriores se presenta como luego de hacer clic en el botón “Generate Instructions” se presentan las instrucciones generadas para cada uno de los processing element siendo estos numerados de 1 a 3 de izquierda a derecha.

Llenado de la memoria con valores aleatorios

Por otra parte, la aplicación tiene la capacidad de ejecutarse con valores aleatorios en memoria, o bien, ejecutarse con una memoria inicializada en ceros. En este caso se probará la funcionalidad de generar valores aleatorios para la memoria.

Memory				
Line	Block 0	Block 1	Block 2	Block 3
0	165	97	64	148
4	254	114	237	22
8	106	144	47	252
12	138	247	175	38
16	249	60	209	108
20	176	238	44	139
24	230	235	4	183
28	59	254	2	78
32	73	222	5	229
36	105	96	213	132
40	188	194	42	134
44	215	124	40	20
48	236	198	25	13
52	128	114	172	36
56	137	242	60	194
60	32	17	14	15

Como puede verse en la imagen anterior, la generación de valores aleatorios es exitosa.

Ejecución automática de instrucciones en un processing element

A continuación, se presenta la ejecución de instrucciones de forma automática para un processing element, en este caso para el processing element 1.

Execute All PEs

```

0. write 45 r6
1. write 50 r4
2. read 50 r3
3. incr r3
4. write 50 r3
5. incr r2
6. write 50 r2
7. read 50 r2
8. incr r5
9. write 45 r5
10. read 45 r5

```

Start

Step

Registers

r0	r1	r2	r3	r4	r5	r6	r7	r8
0	0	1	1	0	1	0	0	0

Cache

Block 0	Block 1	Block 2	Block 3	State
236	198	1	13	Modified
0	0	0	0	Invalid
0	0	0	0	Invalid
215	1	40	20	Modified

Current Instruction # 10

Ejecución manual de instrucciones en un processing element

A continuación, se presenta la ejecución de forma manual por medio del botón “Step”. Para este caso se utilizará el processing element dos.

0. incr r2
1. write 40 r2
2. read 40 r2
3. incr r4
4. write 45 r4
5. read 45 r4
6. incr r0
7. write 45 r0
8. read 45 r0

Current Instruction # 2

Start

Step

0. incr r2
1. write 40 r2
2. read 40 r2
3. incr r4
4. write 45 r4
5. read 45 r4
6. incr r0
7. write 45 r0
8. read 45 r0

Current Instruction # 3

Start

Step

Como puede evidenciarse luego de presionar el botón de en 3 ocasiones el “Current Instruction” es 2, luego al presionar el botón “Step” nuevamente la el “Current Instruction es 3” demostrando que la ejecución de la instrucción fue exitosa, ya que, este valor se actualiza una vez la instrucción fue ejecutada.

Ejecución automática de instrucciones de todos los processing element

Por otra parte, la ejecución automática para todos los processing elements consiste en la ejecución de todas las instrucciones de los 3 PEs de forma simultánea.

Execute All PEs

0. write 45 r6
1. write 50 r4
2. read 50 r3
3. incr r3
4. write 50 r3
5. incr r2
6. write 50 r2
7. read 50 r2
8. incr r5
9. write 45 r5
10. read 45 r5

Current Instruction #

Start

Step

0. incr r2
1. write 40 r2
2. read 40 r2
3. incr r4
4. write 45 r4
5. read 45 r4
6. incr r0
7. write 45 r0
8. read 45 r0

Current Instruction #

Start

Step

0. incr r3
1. write 45 r3
2. read 45 r3
3. write 45 r4
4. write 45 r5
5. read 40 r1
6. incr r1
7. write 40 r1

Current Instruction #

Start

Step

Registers

r0	r1	r2	r3	r4	r5	r6	r7	r8
0	0	1	1	0	1	0	0	0

Cache

Block	Block	Block	Block	State
0	1	2	3	
248	130	1	68	Modified
0	0	0	0	Invalid
0	0	0	0	Invalid
0	1	166	182	Modified

Registers

r0	r1	r2	r3	r4	r5	r6	r7	r8
1	0	1	0	0	0	0	0	0

Cache

Block	Block	Block	Block	State
0	1	2	3	
0	0	0	0	Invalid
0	0	0	0	Invalid
2	182	183	219	Invalid
0	1	166	182	Invalid

Registers

r0	r1	r2	r3	r4	r5	r6	r7	r8
0	2	0	1	0	0	0	0	0

Cache

Block	Block	Block	Block	State
0	1	2	3	
0	0	0	0	Invalid
0	0	0	0	Invalid
2	182	183	219	Modified
0	1	166	182	Invalid

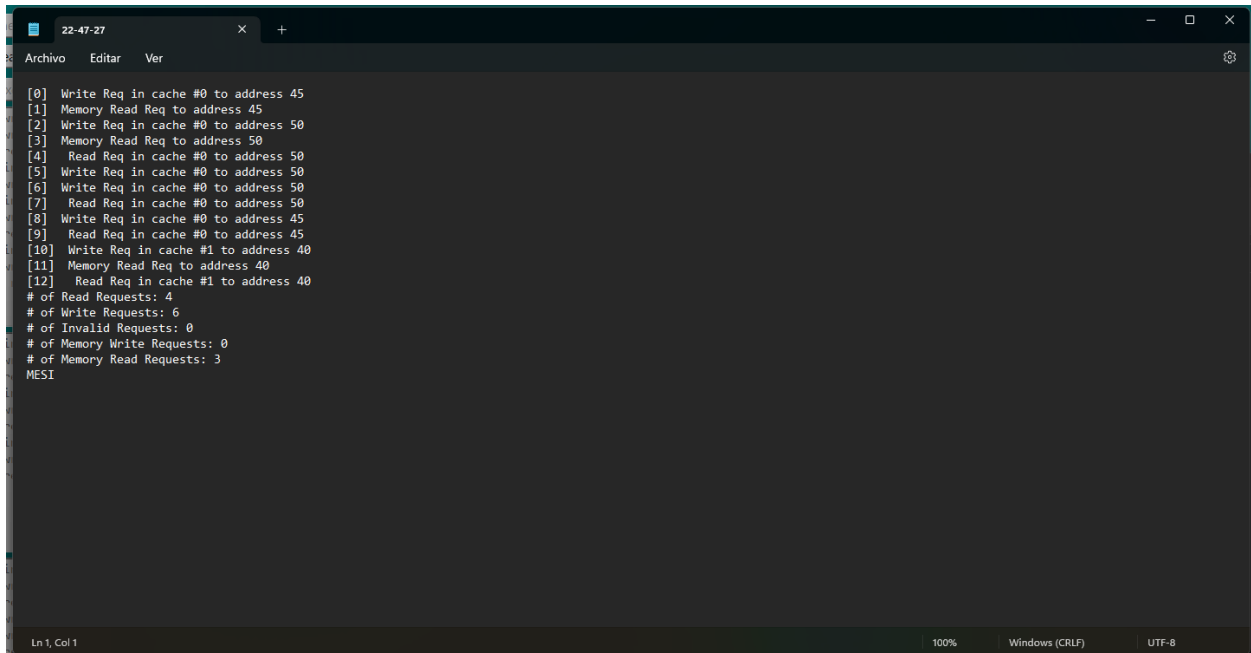
Memory

Line	Block	Block	Block	Block
	0	1	2	3
0	147	142	210	103
4	122	79	199	75
8	25	20	0	196
12	12	239	250	107
16	95	90	35	64
20	113	165	15	54
24	206	140	140	37
28	252	87	1	37
32	90	21	162	108
36	212	156	78	195
40	2	182	183	219
44	0	1	166	182
48	248	130	1	68
52	222	208	115	71
56	221	128	88	13
60	13	219	65	70

Como se puede observar, en la imagen anterior se muestra la ejecución finalizada de todas las instrucciones de los tres processing elements, además se puede observar como las tres cachés cuentan con valores en sus líneas y que además el estado de estas no en todos es el mismo inicial, por lo cual se concluye que el funcionamiento de este modo es exitoso.

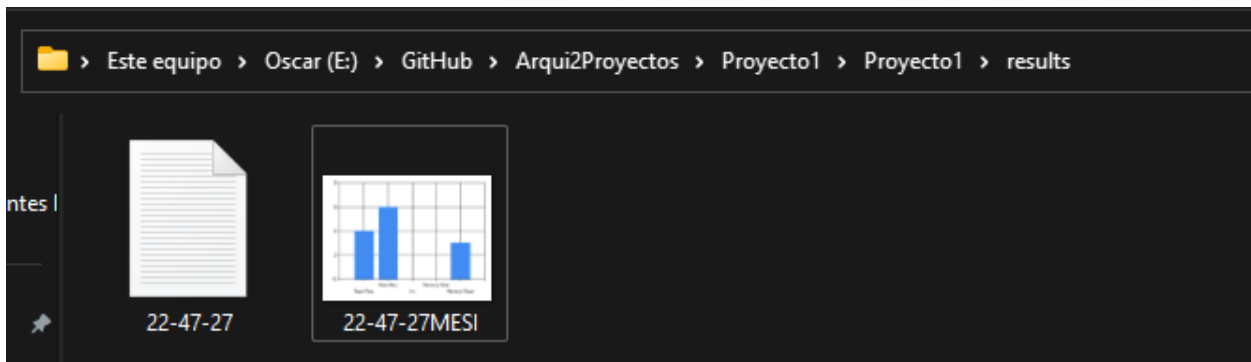
Generación de reportes

Finalmente, como parte de los requerimientos de usuario se debía tener un control de los read request, write request, invalidates, entre otros, es por esto que se buscó generar un archivo .log como el que se presenta en la siguiente imagen.



```
[0] Write Req in cache #0 to address 45
[1] Memory Read Req to address 45
[2] Write Req in cache #0 to address 50
[3] Memory Read Req to address 50
[4] Read Req in cache #0 to address 50
[5] Write Req in cache #0 to address 50
[6] Write Req in cache #0 to address 50
[7] Read Req in cache #0 to address 50
[8] Write Req in cache #0 to address 45
[9] Read Req in cache #0 to address 45
[10] Write Req in cache #1 to address 40
[11] Memory Read Req to address 40
[12] Read Req in cache #1 to address 40
# of Read Requests: 4
# of Write Requests: 6
# of Invalid Requests: 0
# of Memory Write Requests: 0
# of Memory Read Requests: 3
MESI
```

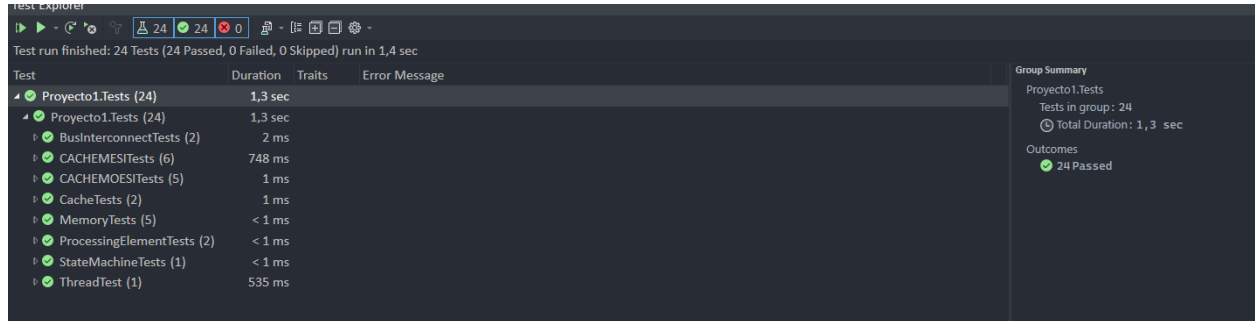
Además, como se desea que el usuario cuente con el mayor contexto posible de los resultados de la ejecución, se crea una gráfica en la cual se muestra un recuento del logfile, haciendo que la interpretación de los resultados sea más interactiva.



Finalmente, como parte de los requerimientos de usuario, se requería que los resultados fueran almacenados en una carpeta “results” como se muestra en la imagen anterior.

Unit Tests

En la siguiente captura de pantalla se muestra un resumen de las pruebas unitarias ejecutadas para la verificación del buen funcionamiento del proyecto



Test	Duration	Traits	Error Message
Projecto1.Tests (24)	1,3 sec		
Projecto1.Tests (24)	1,3 sec		
BusInterconnectTests (2)	2 ms		
CACHEMESITests (6)	748 ms		
CACHEMOESITests (5)	1 ms		
CacheTests (2)	1 ms		
MemoryTests (5)	< 1 ms		
ProcessingElementTests (2)	< 1 ms		
StateMachineTests (1)	< 1 ms		
ThreadTest (1)	535 ms		

Group Summary
Projecto1.Tests
Tests in group: 24
Total Duration: 1,3 sec
Outcomes
24 Passed

En cada una de estas pruebas, se comprobaron casos tanto típicos como atípicos que el grupo quiso probar y asegurar de su buen funcionamiento, incluyendo casos como comunicaciones entre procesos, casos posibles del flujo de datos, transiciones apropiadas en los diagramas de estado de los protocolos de coherencia de caché MESI y MOESI, pruebas de la memoria principal, de los PEs, y el funcionamiento con hilos.

Muchas de estas pruebas resultaron provechosas, dado que permitió que el grupo se enterara de bugs que habían en el código y funcionamientos no esperados que fueron detectados, y arreglados. En particular, el tema más sensible era el flujo de datos entre módulos, dado que se presentaba un efecto de cascada de si había una comunicación errónea o mal implementada, todo lo que seguía después de eso también fallaba, y era importante revisar errores desde el punto más alto posible.