

# Memoria de la práctica 2

Manel Lurbe Sempere

Manuel José Martínez Baños

## Contenido

Ejercicio 0.....	3
Cache config L1u L2u .....	3
Cache config L1s L2s.....	4
Ejercicio 1.....	5
Resultados obtenidos.....	5
Grafico obtenido .....	5
Comentario sobre los resultados obtenidos.....	6
Ejercicio 2.....	7
Cache config Dual Core .....	7
Cache config Dual Procesor .....	8
Script de simulación .....	9
Mezcla 1 .....	9
Mezcla 2 .....	10
Mezcla 3 .....	10

Mezcla 4 .....	10
Script de obtención de resultados .....	11
Resultados obtenidos .....	11
Grafico obtenido .....	12
Comentario sobre los resultados obtenidos.....	12
Ejercicio 3.....	13
Script de simulación .....	13
Script de obtención de resultados.....	14
Cache config FIFO .....	15
Cache config LRU .....	16
Cache config Random.....	17
Resultados obtenidos.....	18
Gráficos obtenidos .....	19
Comentario sobre los resultados obtenidos.....	20

## Ejercicio 0

### Cache config L1u L2u

```
cacheconfig.l1u_l2u
1  [ CacheGeometry l1topo ]
2  Sets = 16
3  Assoc = 2
4  BlockSize = 64
5  Latency = 2
6
7  [ CacheGeometry l2topo ]
8  Sets = 32
9  Assoc = 4
10 BlockSize = 64
11 Latency = 20
12
13 [ Net net-0 ]
14 Topology = Bus
15 LinkWidth = 32
16
17 [ Net net-1 ]
18 Topology = Bus
19 LinkWidth = 32
20
21 [ Cache l1 ]
22 Geometry = l1topo
23 LoNet = net-0
24
25 [ Cache l2 ]
26 Geometry = l2topo
27 HiNet = net-0
28 LoNet = net-1
29
30 [ MainMemory ]
31 HiNet = net-1
32 BlockSize = 64
33 Latency = 200
34
35 [ Node 0 ]
36 Core = 0
37 Thread = 0
38 DCache = l1
39 ICache = l1
40
```

## Cache config L1s L2s

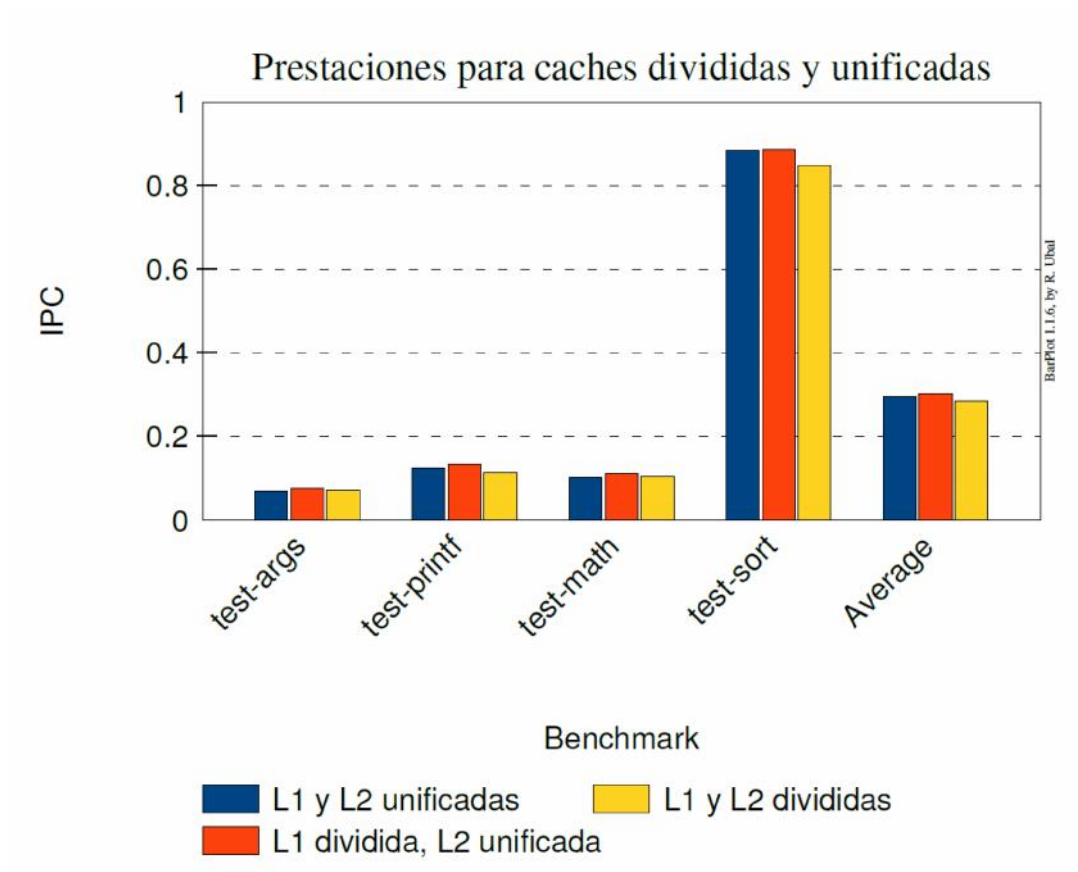
```
cacheconfig.l1s_l2s
1 [ CacheGeometry l1topo ]
2 Sets = 8
3 Assoc = 2
4 BlockSize = 64
5 Latency = 2
6
7 [ CacheGeometry l2topo ]
8 Sets = 16
9 Assoc = 4
10 BlockSize = 64
11 Latency = 20
12
13 [ Net net-0 ]
14 Topology = Bus
15 LinkWidth = 32
16
17 [ Net net-1 ]
18 Topology = Bus
19 LinkWidth = 32
20
21 [ Net net-2 ]
22 Topology = Bus
23 LinkWidth = 32
24
25 [ Cache dl1 ]
26 Geometry = l1topo
27 LoNet = net-0
28
29 [ Cache il1 ]
30 Geometry = l1topo
31 LoNet = net-1
32
33 [ Cache dl2 ]
34 Geometry = l2topo
35 HiNet = net-0
36 LoNet = net-2
37
38 [ Cache il2 ]
39 Geometry = l2topo
40 HiNet = net-1
41 LoNet = net-2
42
43 [ MainMemory ]
44 HiNet = net-2
45 BlockSize = 64
46 Latency = 200
47
48 [ Node 0 ]
49 Core = 0
50 Thread = 0
51 DCache = dl1
52 ICache = il1
```

# Ejercicio 1

## Resultados obtenidos

	cache.res			
1	0.06841	0.07484	0.07161	
2	0.1238	0.1337	0.1139	
3	0.1029	0.1109	0.105	
4	0.8832	0.8855	0.849	
5				

## Grafico obtenido



## *Comentario sobre los resultados obtenidos*

Viendo el grafico anterior vemos claramente que en todos los test la mejor configuración es L1 dividida y L2 unificada, seguidamente encontramos la configuración L1 y L2 unificadas, y por último tenemos la que resulta ser la que menos prestaciones nos ha demostrado, L1 y L2 divididas.

En el caso de las caches divididas L1 y L2 podemos decir que tienen el peor rendimiento de las 3 configuraciones porque tienen un espacio más limitado de cache que una unificada y si por ejemplo la cache de instrucciones no se llena y la de datos sí, la cache de datos no va a poder usar el espacio vacío de la caché de instrucciones, cosa que en una unificada sí que se podría hacer de tal forma que si son necesarios menos bloques de datos y más de código estos se reparten de forma eficiente y transparente.

Las caches totalmente unificadas tampoco son la mejor opción ya que puede darse el caso de que haya muchos reemplazos en L1 y L2 por parte de las instrucciones sobre los datos y viceversa, por eso la mejor opción es buscar el punto intermedio que es L1 dividida y L2 unificada, así podemos ofrecer más almacenamiento a la caché que lo necesite (L2) y aseguremos un espacio reservado por lo menos para instrucciones y para datos (L1), de este modo en L1 no hay tanta “interferencias” de tipos (mezcla de instrucciones y datos ).

En resumen, podemos decir, que la memoria en L1 es una zona más crítica que en L2, por tanto, interesa, tener una L1 dividida para instrucciones y datos. Pero una L2 unificada, para que esta sea más grande y pueda almacenar diferentes tipos de datos, que pueden ser necesitados por otros recursos. Evitar de este modo lo máximo posibles fallos en la L2.

Así logramos, la mayoría de las veces, que en caso de fallo en L1 , acceda a L2 y aquí haya un “acierto” por tanto se produzca un ciclo de para.

## Ejercicio 2

### Cache config Dual Core

```
cachedualcore x
1 [ CacheGeometry l1topo ]
2 Sets = 8
3 Assoc = 2
4 BlockSize = 64
5 Latency = 2
6
7 [ CacheGeometry l2topo ]
8 Sets = 64
9 Assoc = 4
10 BlockSize = 64
11 Latency = 20
12
13 [ Net net-0 ]
14 Topology = Bus
15 Linkwidth = 32
16
17 [ Net net-1 ]
18 Topology = Bus
19 Linkwidth = 32
20
21 [ Cache d11-0 ]
22 Geometry = l1topo
23 LoNet = net-0
24
25 [ Cache i11-0 ]
26 Geometry = l1topo
27 LoNet = net-0
28
29 [ Cache d11-1 ]
30 Geometry = l1topo
31 LoNet = net-0
32
33 [ Cache i11-1 ]
34 Geometry = l1topo
35 LoNet = net-0
36
37 [ Cache l2 ]
38 Geometry = l2topo
39 HiNet = net-0
40 LoNet = net-1
41
42 [ MainMemory ]
43 HiNet = net-1
44 BlockSize = 64
45 Latency = 200
46
47 [ Node 0 ]
48 Core = 0
49 Thread = 0
50 DCache = d11-0
51 ICache = i11-0
52
53 [ Node 1 ]
54 Core = 1
55 Thread = 0
56 DCache = d11-1
57 ICache = i11-1
```

## Cache config Dual Procesor

```
cachedualprocesor
1  [ CacheGeometry l1topo ]
2  Sets = 8
3  Assoc = 2
4  BlockSize = 64
5  Latency = 2
6
7  [ CacheGeometry l2topo ]
8  Sets = 64
9  Assoc = 4
10 BlockSize = 64
11 Latency = 20
12
13 [ Net net-0 ]
14 Topology = Bus
15 Linkwidth = 32
16
17 [ Net net-1 ]
18 Topology = Bus
19 Linkwidth = 32
20
21 [ Net net-2 ]
22 Topology = Bus
23 Linkwidth = 32
24
25 [ Cache dl1-0 ]
26 Geometry = l1topo
27 LoNet = net-0
28
29 [ Cache il1-0 ]
30 Geometry = l1topo
31 LoNet = net-0
32
33 [ Cache dl1-1 ]
34 Geometry = l1topo
35 LoNet = net-1
36
37 [ Cache il1-1 ]
38 Geometry = l1topo
39 LoNet = net-1
40
41 [ Cache l2-0 ]
42 Geometry = l2topo
43 HiNet = net-0
44 LoNet = net-2
45
46 [ Cache l2-1 ]
47 Geometry = l2topo
48 HiNet = net-1
49 LoNet = net-2
50
51 [ MainMemory ]
52 HiNet = net-2
53 BlockSize = 64
54 Latency = 200
55
56 [ Node 0 ]
57 Core = 0
58 Thread = 0
59 DCache = dl1-0
60 ICache = il1-0
61
62 [ Node 1 ]
63 Core = 1
64 Thread = 0
65 DCache = dl1-1
66 ICache = il1-1
```



## Script de simulación

```
ej2_proceso.simul
1  #!/bin/sh
2
3  resfile="resultados_ejer2.res"
4  tempfile=$(mktemp)
5  rm -f $resfile
6  for workload in mezcla1 mezcla2 mezcla3 mezcla4; do
7      for config in cachedualcore cachedualprocesor; do
8          ./m2s -cores 2 -cacheconfig $config -ctxconfig $workload >/dev/null 2>$tempfile
9          line=$(cat $tempfile | grep sim.ipc)
10         ipc=$(echo $line | awk '{print $2}')
11         echo -n "$ipc " >> $resfile
12         echo -n "."
13     done
14     echo >> $resfile
15 done
16 rm -f $tempfile
17 echo
18
```

## Mezcla 1

```
mezcla1
1  [ Context 0 ]
2  Exe = test-args.i386
3  Args = arg1 arg2
4
5  [ Context 1 ]
6  Exe = test-printf.i386
```

### Mezcla 2

```
mezcla2
1  [ Context 0 ]
2  Exe = test-math.i386
3
4  [ Context 1 ]
5  Exe = test-printf.i386
6
```

### Mezcla 3

```
mezcla3
1  [ Context 0 ]
2  Exe = test-sort.i386
3
4  [ Context 1 ]
5  Exe = test-args.i386
6  Args = arg1 arg2
7
```

### Mezcla 4

```
mezcla4
1  [ Context 0 ]
2  Exe = test-sort.i386
3
4  [ Context 1 ]
5  Exe = test-math.i386
6
```

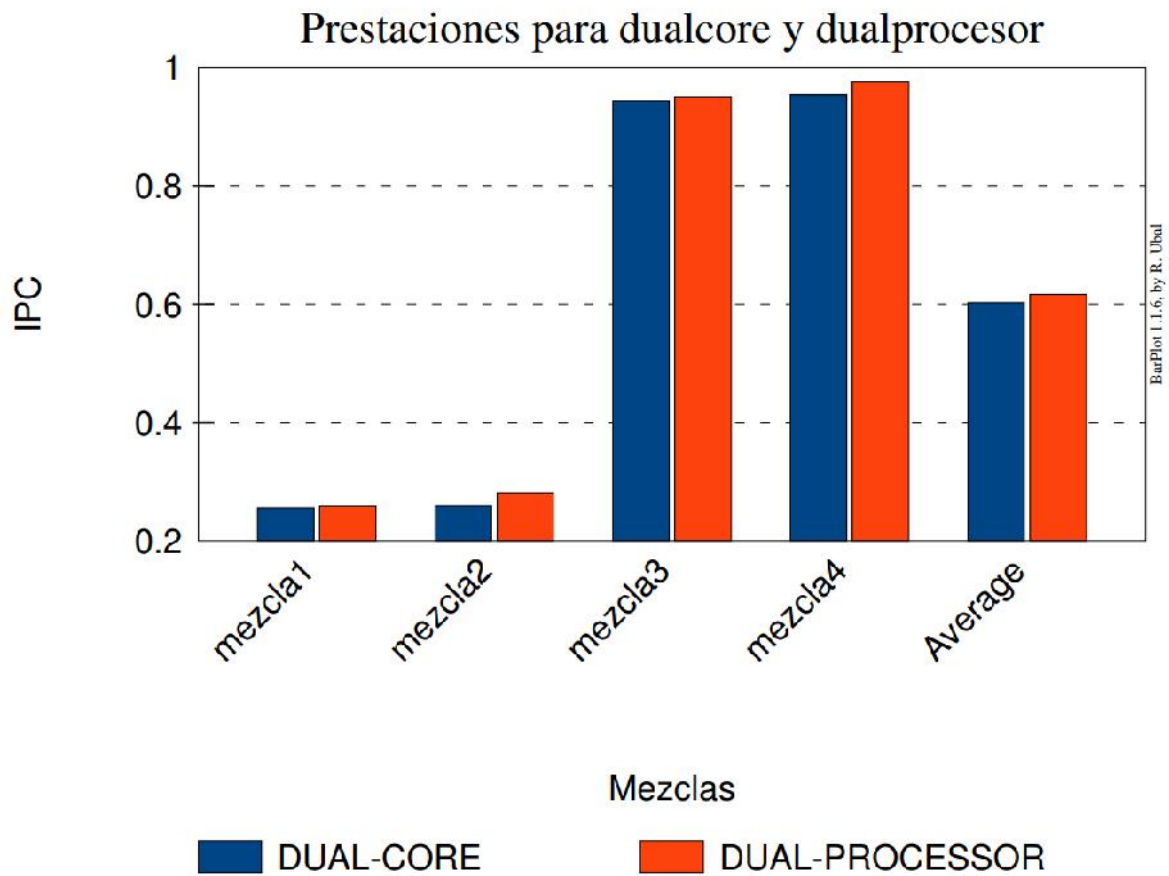
### Script de obtención de resultados

```
ejer2_proceso.plot
1  #!/bin/sh
2  plotfile=$(mktemp)
3  cat << EOF >> $plotfile
4  XTics = 'mezcla1' 'mezcla2' 'mezcla3' 'mezcla4'
5  Key = 'DUAL-CORE' 'DUAL-PROCESSOR'
6  Averages = True
7  XLabel = 'Mezclas'
8  YLabel = 'IPC'
9  Title = 'Prestaciones para dualcore y dualprocesor'
10 EOF
11 cat resultados_ejer2.res >> $plotfile
12 ./barplot.py $plotfile resultados_ejer2.eps
13 rm -f $plotfile
```

### Resultados obtenidos

	resultados_ejer2.res	
1	0.2556	0.2584
2	0.2588	0.281
3	0.9423	0.9497
4	0.9528	0.9757
5		

### *Grafico obtenido*



### *Comentario sobre los resultados obtenidos*

En el grafico anterior vemos claramente que una configuración Dual-Processor saca mejores prestaciones que una configuración Dual-Core, ya que en la primera configuración tenemos caches individuales para cada procesador y en la segunda configuración tenemos una jerarquía de caches compartida para ambos procesadores. Esto provoca que haya reemplazos de un procesador sobre el otro derivando en una pérdida de prestaciones.

Como hemos visto en el apartado anterior test-sort tiene más IPC por eso en la mezcla 3 y 4, donde aparece test-sort, hace aumentar considerablemente el IPC en estas dos mezclas.

## Ejercicio 3

### Script de simulación

```
resfile="resultados_ejer3.res"
tempfile=$(mktemp)
rm -f $resfile

resfile2="resultados_ejer3_INS.res"
rm -f $resfile2

resfile3="resultados_ejer3_REMP.res"
rm -f $resfile3
#test-printf.i386 test-math.i386 test-sort.i386
for workload in test-args.i386 test-printf.i386 test-math.i386 test-sort.i386; do
  for config in ejer3LRU_cacheconfig.llu_l2u ejer3FIFO_cacheconfig.llu_l2u ejer3Random_cacheconfig.llu_l2u; do
    ./m2s -cacheconfig $config -report:cache report_cache $workload >/dev/null 2>$tempfile

    line=$(cat $tempfile | grep l1.hitratio)
    hitratio=$(echo $line | awk '{print $2}')
    echo -n "$hitratio " >> $resfile
    echo -n "."

    line2=$(cat $tempfile | grep sim.inst)
    num_instruccions=$(echo $line2 | awk '{print $2}')
    echo -n "$num_instruccions|" >> $resfile2
    echo -n "."

    #cat report_cache.txt | grep -m1 "Evictions ="
    #Filtrem la primera linea del report_cache que corresponent al paramete Evictions de l1
    remplazos=$(cat report_cache | grep -m1 "Evictions =" | awk '{print $3}')
    echo $remplazos
    echo -n "$remplazos " >> $resfile3
    echo -n "."

    rm -f rc.$config
  done
done
echo >> $resfile
echo >> $resfile2
echo >> $resfile3

done
rm -f $tempfile

echo
```

## Script de obtención de resultados

```
ejer3_reemplazo.plot
1  #!/bin/sh
2  plotfile=$(mktemp)
3  cat << EOF >> $plotfile
4  XTics = 'test-args' 'test-printf' 'test-math' 'test-sort'
5  Key = 'LRU' 'FIFO' 'Random'
6  Averages = True
7  XLabel = 'Benchmark'
8  YLabel = 'HIT RATE'
9  Title = 'Prestaciones para diferentes tipos de reemplazo'
10 EOF
11 cat resultados_ejer3.res >> $plotfile
12 ./barplot.py $plotfile resultados_ejer3.eps
13 rm -f $plotfile
14
15 plotfile=$(mktemp)
16 cat << EOF >> $plotfile
17 XTics = 'test-args' 'test-printf' 'test-math' 'test-sort'
18 Key = 'LRU' 'FIFO' 'Random'
19 Averages = True
20 XLabel = 'Benchmark'
21 YLabel = 'Num.Ins'
22 Title = 'Num. instrucciones realizadas por los programas'
23 EOF
24 cat resultados_ejer3_INS.res >> $plotfile
25 ./barplot.py $plotfile resultados_ejer3_INS.eps
26 rm -f $plotfile
27
28 plotfile=$(mktemp)
29 cat << EOF >> $plotfile
30 XTics = 'test-args' 'test-printf' 'test-math' 'test-sort'
31 Key = 'LRU' 'FIFO' 'Random'
32 Averages = True
33 XLabel = 'Benchmark'
34 YLabel = 'Num.Reemplazos'
35 Title = 'Para Diferentes Politicas de Reemplazo, Num.Reemplazos realizados'
36 EOF
37 cat resultados_ejer3_REMP.res >> $plotfile
38 ./barplot.py $plotfile resultados_ejer3_REMP.eps
39 rm -f $plotfile
```

## Cache config FIFO

```
ejer3FIFO_cacheconfig.l1u_l2u
1 [ CacheGeometry l1topo ]
2 Sets = 16
3 Assoc = 2
4 BlockSize = 64
5 Latency = 2
6 Policy = FIFO
7
8 [ CacheGeometry l2topo ]
9 Sets = 32
10 Assoc = 4
11 BlockSize = 64
12 Latency = 20
13
14 [ Net net-0 ]
15 Topology = Bus
16 LinkWidth = 32
17
18 [ Net net-1 ]
19 Topology = Bus
20 LinkWidth = 32
21
22 [ Cache l1 ]
23 Geometry = l1topo
24 LoNet = net-0
25
26 [ Cache l2 ]
27 Geometry = l2topo
28 HiNet = net-0
29 LoNet = net-1
30
31 [ MainMemory ]
32 HiNet = net-1
33 BlockSize = 64
34 Latency = 200
35
36 [ Node 0 ]
37 Core = 0
38 Thread = 0
39 DCache = l1
40 ICache = l1
41
```



## Cache config LRU

```
ejer3LRU_cacheconfig.l1u_l2u
1 [ CacheGeometry l1topo ]
2 Sets = 16
3 Assoc = 2
4 BlockSize = 64
5 Latency = 2
6 Policy = LRU
7
8 [ CacheGeometry l2topo ]
9 Sets = 32
10 Assoc = 4
11 BlockSize = 64
12 Latency = 20
13
14 [ Net net-0 ]
15 Topology = Bus
16 LinkWidth = 32
17
18 [ Net net-1 ]
19 Topology = Bus
20 LinkWidth = 32
21
22 [ Cache l1 ]
23 Geometry = l1topo
24 LoNet = net-0
25
26 [ Cache l2 ]
27 Geometry = l2topo
28 HiNet = net-0
29 LoNet = net-1
30
31 [ MainMemory ]
32 HiNet = net-1
33 BlockSize = 64
34 Latency = 200
35
36 [ Node 0 ]
37 Core = 0
38 Thread = 0
39 DCache = l1
40 ICache = l1
```



## Cache config Random

```
ejer3Random_cacheconfig.l1u_l2u
1 [ CacheGeometry 11topo ]
2 Sets = 16
3 Assoc = 2
4 BlockSize = 64
5 Latency = 2
6 Policy = Random
7
8 [ CacheGeometry 12topo ]
9 Sets = 32
10 Assoc = 4
11 BlockSize = 64
12 Latency = 20
13
14 [ Net net-0 ]
15 Topology = Bus
16 LinkWidth = 32
17
18 [ Net net-1 ]
19 Topology = Bus
20 LinkWidth = 32
21
22 [ Cache 11 ]
23 Geometry = 11topo
24 LoNet = net-0
25
26 [ Cache 12 ]
27 Geometry = 12topo
28 HiNet = net-0
29 LoNet = net-1
30
31 [ MainMemory ]
32 HiNet = net-1
33 BlockSize = 64
34 Latency = 200
35
36 [ Node 0 ]
37 Core = 0
38 Thread = 0
39 DCache = 11
40 ICache = 11
```

## Resultados obtenidos

resultados_ejer3.res			
1	0.2935	0.2707	0.2689
2	0.3053	0.2712	0.2866
3	0.3829	0.3417	0.3428
4	0.8293	0.8002	0.8067
5			

1 Hit Rate

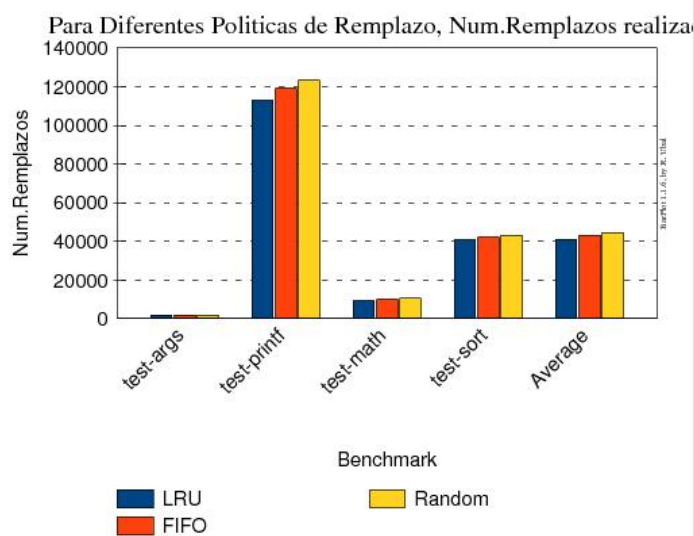
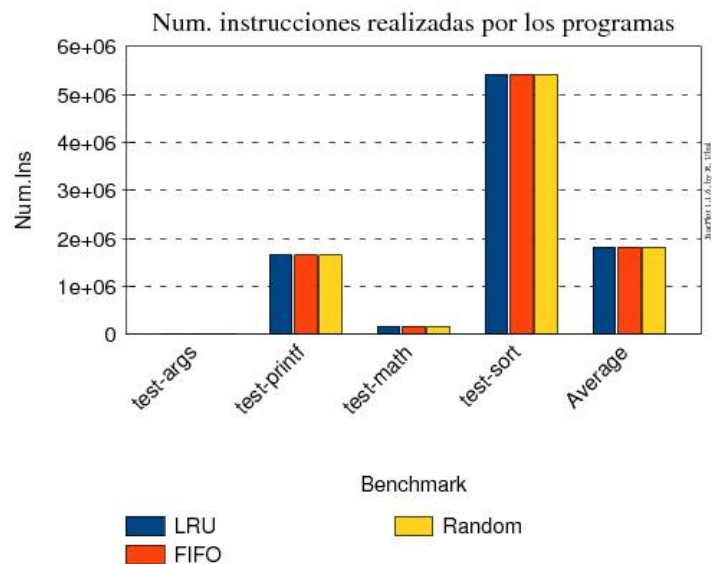
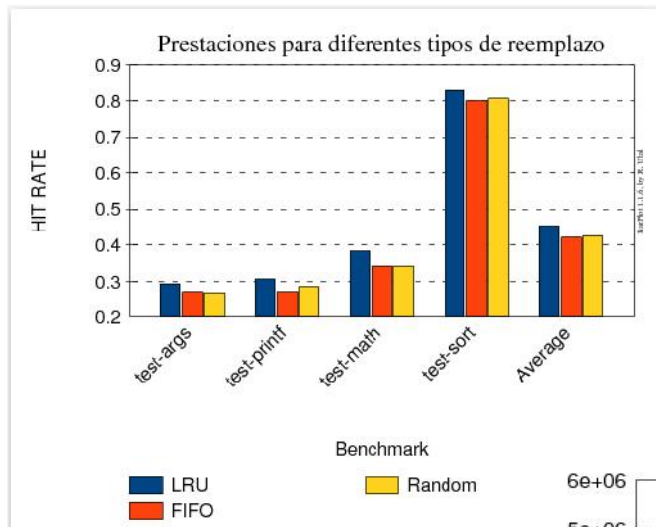
resultados_ejer3_REMP.res			
1	1732	1803	1890
2	112769	119147	123339
3	9363	9988	10417
4	40834	42173	43091
5			

2 Reemplazo

resultados_ejer3_INS.res			
1	20636	20636	20636
2	1660412	1660412	1660412
3	171496	171496	171496
4	5426759	5426759	5426759
5			

3 Instrucciones

## Gráficos obtenidos



## *Comentario sobre los resultados obtenidos*

En el ejercicio 3, si visualizamos la gráfica donde tenemos en cuenta el hit-rate, vemos que, en todo el uso de la política, LRU sale ganadora y muy ajustadas entre ellas la random y FIFO después.

LRU es la política con mejores prestaciones porque reemplaza los bloques que más antiguos que han sido usados. Este mecanismo se basa en pura heurística. Depende de un contador para poder conocer cuando han sido usados. Para después descartar aquellos con el contador menor.

LRU por tanto, tiene mayor ventaja que el resto, porque esos bloques reemplazados, tiene una probabilidad mayor de que no se vuelvan a necesitar.

Pero LRU tiene el inconveniente de que es más difícil de implementar, a parte, de que en L2 su uso es bastante mas ineficiente.

Si nos vamos más al detalle. Tenemos dos gráficas más, la que nos muestra el Número de instrucciones por cada programa y otra el número de reemplazamos.

Como vemos la relación está clara, a mayor cantidad de número de instrucciones, su número de reemplazos es mayor y por tanto los aciertos crecen, pero no siempre es así.

Hay que tener en cuenta que si un programa tiene muy pocas instrucciones, como es el caso de test-args, sus reemplazos serán muy escasos y su hit-rate también. No le da tiempo a acertar, porque de primeras la mayoría de veces falla al principio al no tener los valores por tanto al tener pocas instrucciones no necesitará cargar más “bloques” que contienen un conjunto de instrucciones.

En el caso de test-print el número de reemplazos es muy alto por lo que su tasa de acierto es muy baja, debido a la cantidad de veces que falla y debe reemplazar. Además, vemos que test-sort tiene un numero de instrucciones mayor que por ejemplo test-print y en cambio, el número de reemplazos es menor. Podemos concluir, que no siempre el número de instrucciones determina un número de reemplazos mayor y por tanto peor hit-rate.