Ejercicio 1

El siguiente multiprograma corre LEGv8 y usa la memoria **compartida** `0x20000`. Dar los valores posibles que puede contener esa dirección de la RAM al finalizar el multiprograma.

{ M[0x20000] = 0 }		
1: MOVZ X8,#2,LSL #16 2: LDUR X1,[X8,#0] 3: ORR X1,X1,#1 4: STUR X1,[X8,#0]	A: MOVZ X8,#2,LSL #16 B: LDUR X1,[X8,#0] C: ORR X1,X1,#2 D: STUR X1,[X8,#0]	a: MOVZ X8,#2,LSL #16 b: LDUR X1,[X8,#0] c: ORR X1,X1,#4 d: STUR X1,[X8,#0]
{ M[0x20000] = ?}		

alor final de m[0x20000] n decimal)	Un escenario de lleva a ese resultado	Valor final de Mem[0x20000] (en decimal)	Un escenario de lleva a ese resultado

Individualmente:

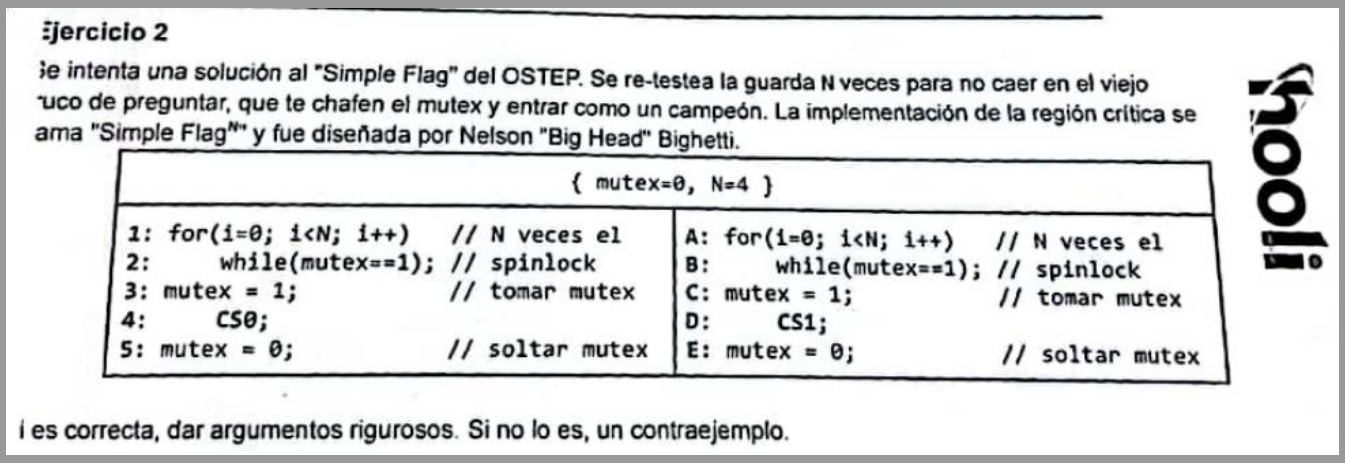
1: MOVZ X8, #2, LSL #16 // X8 = 0x20000 ≡ (2 * 2^16 = 131072)
2: LDUR X1, [X8, #0] // X1 = 0x0
3: ORR X1, X1, #1 // X1 = 0x1
4: STUR X1, [X8, #0] // X8 = 0x1

A: MOVZ X8, #2, LSL #16 // X8 = 0x20000 ≡ (2 * 2^16 = 131072)
B: LDUR X1, [X8, #0] // X1 = 0x0
C: ORR X1, X1, #2 // X1 = 0x2
D: STUR X1, [X8, #0] // X8 = 0x2

a: MOVZ X8, #2, LSL #16 // X8 = 0x20000 ≡ (2 * 2^16 = 131072)
b: LDUR X1, [X8, #0] // X1 = 0x0
c: ORR X1, X1, #4 // X1 = 0x4
d: STUR X1, [X8, #0] // X8 = 0x4

Valor final de Mem[0x20000]	Un escenario que lleva a ese resultado	Valor final de Mem[0x20000]	Un escenario que lleva a ese resultado

(en decimal)		(en decimal)	
0x7	abcdABCD1234	0x6	12ABC34Dabcd
0x1	12ABCDabcd34	0x5	ABabcCDd1234
0x2	AB1234abcdCD	0X3	abABCcdD1234
0x4	abABCD1234cd	—	—



Ejercicio 2

Se intenta una solución al "Simple Flag" del OSTEP. Se re-testea la guarda N veces para no caer en el viejo truco de preguntar, que te chafen el mutex y entrar como un campeón. La implementación de la región crítica se llama "Simple Flag^N" y fue diseñada por Nelson "Big Head" Bighetti.

{ mutex=0, N=4 }

```
1: for(i=0; i<N; i++)    // N veces el
2:     while(mutex==1); // spinlock
3: mutex = 1;           // tomar mutex
4:     CS0;
5: mutex = 0;           // soltar mutex
```

```
A: for(i=0; i<N; i++)    // N veces el
B:     while(mutex==1); // spinlock
C: mutex = 1;           // tomar mutex
D:     CS1;
E: mutex = 0;           // soltar mutex
```

Si es correcta, dar argumentos rigurosos. Si no lo es, un contraejemplo.

AB 12 AB 12 AB 12 AB 12 345 CDE

Supongamos que el mutex está inicializado en 0, luego se evalúan las 4 guardas del for, dando siempre en el while interno false. Luego cuando no entra más al for, no vuelve a evaluar el while y los dos programas entran a la zona crítica.

Ejercicio 3

Al siguiente multiprograma se lo denomina *Concurrent Vector Writing*. **Suponga atomicidad linea a linea**. Agregar **semáforos entre lineas** para que el resultado final sea `{1,1,1,1}`. Hay varias soluciones posibles. Cuanto más concurrente, es decir más escenarios de ejecución tenga, mejor.

{ a[4] = {2,2,2,2} }	
1: i = 0; 2: while (i<4) { 3: a[i] = 0; 4: i++; 5: }	A: j = 0; B: while (j<4) { C: a[j] = 1; D: j++; E: }

sem_t full, empty;
sem_init(full,1);
sem_init(empty,0);

{ a[4] = {2, 2, 2, 2} }	
1: i = 0; 2: while(i<4) { 3: sem_down(full) 4: a[i] = 0; 5: i++; 6: sem_up(empty) 7: }	A: j = 0; B: while(j<4) { C: sem_down(empty) D: a[j] = 1; E: j++; F: sem_up(full) G: }

Se podrían poner fuera del while, pero creo que es más concurrente de esta forma. Si es que entendí bien el enunciado xd.

Ejercicio 4

El disco duro rotacional[1] **Seagate BarraCuda de 1 TB** e interfaz SATA3 (ST1000DM010) tiene las características de la fila central de la tabla y **cuesta 70 USD**. Se calcula y presenta la $R_{i/O}$ velocidad de lectura al azar para bloques de 4 KiB.
Se está evaluando **duplicar la velocidad de rotación** "Ver.UpRPM" que incrementa el precio final de venta en 50 USD, o bien **reducir a la mitad el tiempo promedio de búsqueda** "Ver.DownSeek" y esto incrementa el costo en 45 USD.
El cociente $R_{i/O}$/Precio mide el **costo del disco para lecturas al azar** y se mide en KiB/s por USD.
Calcularlo y ordenar los discos del más conveniente al menos conveniente.

Nombre	RPM	Seektime [ms]	Transf. [MiB/s]	$R_{i/O}$ [KiB/s]	$R_{i/O}$/Precio
Ver.UpRPM	15000	8.5	210	380.28	
Barracuda	7200	8.5	210	315.33	
Ver.DownSeek	7200	4.25	210	474.20	

Orden de conveniencia. Poner el nombre:

#1 _____ #2 _____ #3 _____

Nombre	RPM	Seektime[ms]	Transf. [MiB/s]	$R_{i/o}$ [KiB/s]	$R_{i/o}$ / Precio
Ver.UpRPM	15000	8.5	210	380.28	
Barracuda	7200	8.5	210	315.33	
Ver.DownSeek	7200	4.25	210	474.20	

Rendimiento por dólar (KiB/s por USD) = Rendimiento en KiB/s / Precio en USD

Barracuda $R_{i/o}$precio = 315.33 KiB/s / (70 USD) = 4,50 KiB/s por USD

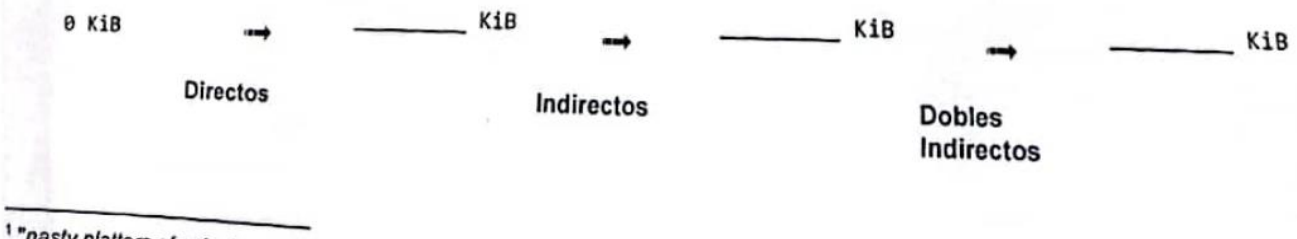
Ver.UpRPM $R_{i/o}$precio = 380.28 KiB/s / (70 + 50) USD = 3,169 KiB/s por USD

Ver.DownSeek $R_{i/o}$precio = 474.20 KiB/s / (70 + 45) USD = 4.12 KiB/s por USD

#1 Barracuda #2 Ver.DownSeek #3 Ver.UpRPM

Ejercicio 5

Se tiene un Sistema de Archivos *UNIX-like* con un esquema diferente, pues tiene **varios indirectos y varios dobles indirectos**. Tamaño de bloque, 1 KiB. **Tamaño de índice de bloque, 24 bits**. Bloques directos, indirectos, doble indirectos: 8, 8, 8. Dar la **secuencia estrictamente creciente** de tamaños de archivos que van marcando el paso de Directos a Indirectos y a Doble Indirectos, hasta el máximo tamaño de archivo posible.



[1] "nasty platters of spin...

Tamaño de bloque = 1 KiB
Índice de bloque = 24 bits = 3 bytes
Bloques directos = 8
Bloques indirectos = 8
Bloques doble indirectos = 8

tamaño de archivo en bloques directos = Tamaño de bloque * Bloques directos = 1 KiB * 8 = 8 KiB

Cant.bloques indirectos por bloque = 1 KiB / 3 bytes = 341
Tam. de archivo por bloque indirecto = 341 * 1 KiB = 341 KiB
tamaño de archivo en 8 bloques indirectos = 341 KiB * 8 = 2,728 KiB

tamaño de archivo en Directos + Indirectos = 8 KiB + 2,728 KiB = 2,736 KiB

Bloques doble indirectos = 8

Un bloque doble indirecto tiene 341.33 bloques, que apuntan cada uno a un bloque que tiene 341.33 bloques. Entonces tenemos 341^2 = 116281 bloques en un bloque doble indirecto. Esto es 116281 * 1 KiB = 116281 KiB Por bloque doble indirecto. Como tenemos 8 de esos, en total es
116281 KiB * 8 = 930248 KiB

tamaño de archivo en Directos + Indirectos + Doble indirectos = 8 KiB + 2,736 KiB + 930248 KiB
= 932992 KiB

0 KiB (Directos)→ 8 KiB (indirectos)→ 2,736 KiB (Dobles Indirectos)→ 932992 KiB