El examen está compuesto de dos bloques de 12 cuestiones
Para aprobar o compensar con el segundo parcial será necesario tener un mínimo de 4 cuestiones correctas en cada bloque
Las respuestas con valores no enteros deben tener 3 decimales
Todas las cuestiones tienen la misma puntuación ($10/24 = 0{,}4167$)

Bloque I

❑ Se tiene un procesador segmentado de 5 etapas con un CPI ideal de 1 y hueco de retardo de salto de 3 ciclos. Además, la dirección efectiva de salto no se conoce hasta el final del segundo ciclo del hueco. ¿Cual será el CPI real al ejecutar un programa con un 25 % de instrucciones de control, de las cuales un 60 % dan lugar a salto efectivo, si se sigue la estrategia de predicción de ...

A) ... salto NO efectivo?
B) ... salto efectivo?

A: 1,45 B: 1,6

Explicación: A: El 60 % de los saltos (25 %) penalizan con el hueco completo (3 ccilos). B: El 60 % de los saltos (25 %) penalizan solo 2 ciclos y el 40 % restante penalizan con el hueco completo (3 ciclos).

❑ Tenemos los siguientes diseños alternativos para un procesador segmentado:

A) cauce de 6 etapas, 5 de duración t y otra de duración 3t
B) como el anterior, pero segmentando totalmente la etapa que hace de cuello de botella
C) como el primero, pero añadiendo 2 replicas de la etapa que hace de cuello de botella, de modo que las 3 trabajen en paralelo

Completar la siguiente tabla con las respectivas ganancias ideales respecto a un procesador como el A, pero NO segmentado:

G(A)	2,667
G(B)	8
G(C)	8

Explicación: A: $G = 8t/3t$ (el cuello de botella limita G)
B: $G = 8t/t$ (el cuello de botella desaparece)
C: $G = 8t/t$ (el cuello de botella desaparece)

❑ Completa la siguiente tabla con Xs en función de las características de los diferentes niveles RAID indicados:

Característica	0	1	3	5
disco de paridad			X	
paridad distribuida				X
mirroring		X		
striping	X		X	X

❑ Completa la siguiente tabla con las palabras MAT o VEC en función del tipo de arquitectura (matricial o vectorial) a la que mejor corresponden las características indicadas.

Basada en segmentación	VEC
Rendimiento = f (tamaño de vectores o matrices)	VEC
Basada en paralelismo	MAT
Coste moderado	VEC

❑ Siendo T1P el tiempo empleado en ejecutar un programa sobre un procesador, TNP el tiempo empleado en ejecutar el mismo programa sobre NP procesadores, G la ganancia proporcionada por el multiprocesador y E la eficiencia del mismo, rellenar la siguiente tabla con las expresiones que correspondan.

G(real)	T1P/TNP	G(ideal)	NP
E(real)	G(real)/NP	E(ideal)	1

❑ Completa la siguiente tabla con Xs en función de si las características de fácil programación (FP) o elevada escalabilidad (EE) corresponden a cada uno de los tipos de computadores.

	FP	**EE**
Cluster de PCs		X
Multiprocesador simétrico (SMP)	X	
Constelación		X
Computador masivamente paralelo (MPP)		X

❑ En Windlx se simula la ejecución de un bucle de 100 iteraciones sin y con adelantamiento, obteniendo una ganancia al aplicar la tecnica hardware de 1.2. Si la simulación sin adelantamiento consumía 2400 ciclos ¿cuántos ciclos de detección por iteración logra eliminar el adelantamiento?

4

Explicación: Con la ganancia deducimos que el tiempo consumido con adelantamiento es 2000 ciclos y calculamos entonces los ciclos de detención por iteración como diferencia absoluta de tiempos dividida por el número de iteraciones.

❑ El alto consumo de cafés de Autobar por parte de algún profesor ha provocado que se tengan que hacer recortes en el presupuesto destinado a adquirir un nuevo servidor. Inicialmente se pensaba en una máquina con procesador MIPS R3000 (como la vista en clase) pero se ha tenido que optar por una versión similar, algo más barata, que presenta las siguientes deficiencias:

- No presenta cachés independientes para datos y código
- No implementa la técnica de adelantamiento

Se sabe que un programa que contiene el siguiente fragmento de código tarda en ejecutarse 150 ciclos de reloj.

```
1       addi  r30,  r0,   10
2  bucle:
3       sub   r2,   r3,   r4
4       and   r5,   r6,   r7
5       and   r8,   r9,   r10
6       or    r11,  r12,  r13
7       subui r30,  r30,  1
8       bnez  r30,  bucle
```

Nota: El registro r0 siempre contiene la constante 0.

— Si la instrucción 3 se sustituyese por **lw r2, (r20)** ¿cuántos ciclos tardaría en ejecutarse el citado fragmento?

160

Explicación: Se perdería un ciclo por iteración al no poder entrar en la etapa IF una instrucción nueva cuando la instrucción de carga esté accediendo a memoria. 1 ciclo perdido X 10 iteraciones = 10 ciclos adicionales de ejecución.

— Si sustituimos el segundo operando de la instrucción **or** por **r2**, aparece una dependencia de datos entre dicha instrucción y la primera del bucle. ¿Cuántos ciclos se tardaría ahora en ejecutar el programa?

150

Explicación: Las instrucciones 'problemáticas' están suficientemente alejadas como para que no sea necesario detener el cauce. Al no producirse ninguna detención, el número de ciclos necesarios para ejecutar el programa no varía.

❑ En las gráficas del anexo se simula el comportamiento de un programa que rellena un array con los 12 primeros elementos de la secuencia de Fibonacci. Los 2 primeros son inicializados al declarar el array y un bucle calcula los 10 restantes. En función del comportamiento observado, responder a las tres cuestiones que siguen:

— Indica el nombre de la técnica o técnicas implícitas en dos de las simulaciones que justifican las mejora de sus tiempos de ejecución respecto a la tercera.

	Técnica 1	Técnica 2
Simulación A	Adelantamiento	Reordenación
Simulación B		
Simulación C	Adelantamiento	

Explicación: Aplicando lo estudiado en las practicas realizadas con el simulador Windlx y con las imágenes de cada simulación, se pueden deducir las técnicas que se usan en cada una de ellas.

— ¿Cuántos ciclos se ahorran en cada iteración del bucle debido exclusivamente a la técnica hardware?

3

Explicación: El adelantamiento se aplica de manera exclusiva en la simulación C. El número de ciclos que se pierden por iteración en ella es 1, mientras que en la B, donde no se aplica ninguna técnica, es de 4. La diferencia es de 3 ciclos es pues lo que se ahorra con la técnica hardware.

— Considerando que las gráficas representan la simulación de la última iteración ejecutada y que el último ciclo de la instrucción de salto es el ciclo final del programa ¿qué ganancia se consigue debido exclusivamente a la técnica software?

1,094

Explicación: La simulación más rapida, que incorpora técnicas tanto hardware como software, es la A y la que incorpora solo la técnica hardware es la C. La ganancia solicitada será por tanto la relación inversa de tiempos de ambas simulaciones (106 y 116 ciclos respectivamente).

Bloque II

❑ El Servicio Nacional de Metereología ha implantado un acceso a sus previsiones metereológicas utilizando las RPC de Sun Microsystem. La API consiste en dos procedimientos, uno de los cuales es de uso libre por los usuarios y el otro es de acceso restringido a los administradores.

El procedimiento `ObtenPrev`, de acceso libre, recibe como parámetro el número de días para los que se desea la previsión (0 = hoy, 1 = hoy y mañana, etc) y devuelve una lista encadenada de estructuras, cada una de las cuales contiene la previsión para cada uno de los días. Internamente, `ObtenPrev` llama al procedimiento `PrevisionDia` que recibe como parámetro el día para el que se desea la previsión (0 = hoy, 1= mañana, 2=pasado mañana, etc) y devuelve un puntero a una estructura `infomet` con la previsión metereológica para ese día. La propia función reserva el espacio necesario para esa estructura.

El procedimiento `ActualizarPrev` es un procemiento administrativo restringido que únicamente se puede ejecutar por clientes desde una determinada máquina. El procedimiento actualiza las bases de datos con la previsión metereológica para un día determinado, para ello recibe como parámetro una pareja <día, información metereológica> y retorna cero si todo ha ido bien. Internamente, `ActualizarPrev` invoca el precedimiento `actualiza` que tiene el siguiente interface:

```
int actualiza( int dia, char *prevision,
               int precip, int tmax, int tmin)
```

dónde `dia` es el número del día a actualizar, `prevision` es una cadena de texto con la previsión del tiempo, `precip` es la probabilidad de lluvia y `tmax` y `tmin` son las temperaturas máxima y mínima previstas. Así mismo, en el caso de que hubiera previsión de nieve, tendría también que invocar el procedimiento `cotadenieve` que tiene el interface:

```
int cotadenieve( int dia, int cota)
```

dónde `dia` es el número del día a actualizar y `cota` la altura mínima a la que se prevé nieve. Tanto `actualiza` como `cotadenieve` retornan un cero si todo ha salido bien.

El listado 1 muestra el fichero de definición del interface necesario para acceder a los servicios y el listado 2 la implementación de uno de los servicios de la aplicación.

```
1  union CotaNieve switch (int hay) {
2    case 0:
3      void;
4    case 1:
5      int cota;
6  };
7
8  struct infomet {
9    string prev<20>;
10   int ProbPrec;
11   CotaNieve nieve;
12   int TMax;
13   int TMin;
14 };
15
16 struct parInfoDia {
17   int dia;
18   infomet info;
19 };
20
21 struct listPrev {
22   infomet prevision;
23   listPrev *sigdia;
24 };
25
26 program PrevMeteo {
27   version MeteoVER {
28     listPrev ObtenPrev( int ndias) = 1;
29     int ActualizarPrev ( parInfoDia info) = 2;
30   } = [redacted];
31 }= 0x20501501;
32
33
```

Listado 1. Interface.x

```
1  int * actualizarprev_4 (parInfoDia *dat,
2                          struct svc_req *r)
3  {
4    static int result;
5    authunix_parms *ucred;
6
7    result=1;
8    if ( r->rq_cred.oa_flavor == AUTH_UNIX ) {
9      ucred = ( struct authunix_parms *)r->rq_clntcred;
10   if ([redacted],MIMAQ)) {
11    result= actualiza([redacted]
12                      [redacted]
13                      [redacted]);
14     if (dat->info.nieve.hay ==1 )
15       result= cotadenieve([redacted]
16                           [redacted]);
17     }
18   }
19   return &result;
20 }
21
22
```

Listado 2.

— ¿Qué falta en la línea 10 del listado 2?

```
strcmp(ucred->aup_machname},MIMAQ)
```

Explicación: Como la invocación del servicio actualiza sólo se puede hacer desde una máquina determinada, hay que comprobarlo mirando el campo `machname` de las credenciales UNIX del cliente.

— Completa las llamadas a `actualiza` y `cotadenieve` de las líneas 11 y 15 del listado 2.

```
actualiza(dat->dia, dat->info.prev,
     dat->info.ProbPrec, dat->info.TMax,
       dat->info.TMin);
cotadenieve(dat->dia,
   dat->info.nieve.CotaNieve_u.cota);
```

Explicación: Para actualizar la previsión hay que llamar a `actualiza` con los prámetros adecuados y que se han recibido como parámetro de la RPC en la estructura `dat`. El primer parámetro es el día a actualizar y es el primer campo de la estructura `parInfoDia` y se accede con la expresión: `dat->dia`. El resto de información necesaria se encuentra en la estructura `info` por lo que tendremos que acceder a esos campos de la siguiente manera: `dat->info.prev`, `dat->info.ProbPrec`, `dat->info.TMax` y `dat->info.TMin`. Nos queda la llamada al procedimiento `actualiza` de la siguiente manera:

```
actualiza(dat->dia, dat->info.prev,
     dat->info.ProbPrec, dat->info.TMax,
       dat->info.TMin);
```

Además, en el caso de que hubiera previsión de nieve, habría que hacer una llamada al procedimiento `cotadenieve` obteniendo la información también de la estructura `info`, pero de la union discriminada `CotaNieve`, en concreto del campo `cota`. En este caso, el acceso al dato nos quedaría en la forma `dat->info.nieve.CotaNieve_u.cota` quedando la llamada al procedimiento:

```
cotadenieve(dat->dia,
  dat->info.nieve.CotaNieve_u.cota);
```

— Escribe la implementación del servicio `ObtenPrev` utilizando el interface simplificado de las RPC de Sun.

```
listPrev *
obtenprev(int *ar, struct svc_req *rqstp)
{
  static listPrev  result;
  listPrev *plstPrev;
  int i;
  infomet d;
  plistPrev=&result;

  for (i=0; i<= ar; i++)
  {
    plistPrev->prevision=PrevisionDia(i);
    if (i != ar ) {
     plistPrev=plistPrev->sigdia;
    }
  }
  plistPrev->sigdia=null;
  return &result;
}
```

Explicación: Para implementar el procedimiento tenemos que tener en cuenta que la variable en la que retornemos el resultado tiene que ser declarada de tipo `static`. Además, la función `PrevisionDia` que hay que llamar desde dentro del procedimiento reserva el espacio necesario para la estructura `infomet` que retorna. Con toda esa información sólo hay que realizar llamadas sucesivas a `PrevisionDia`, ir almacenando el resultado obtenido en la estructura adecuada y hacer avanzar el puntero que nos recorre la lista resultado.

— ¿Qué falta en el hueco de la línea 30 del listado 1?.

4

Explicación: Falta el valor asignado al número de versión del programa. Ese número aparece siempre como postfijo del nombre de las funciones que implementan los servicios. En nuestro programa tenemos en el listado 2 que el nombre de la función tiene de nombre `actualizarprev_4` por lo que el número asignado a la versión es 4.

— El comando `ls` nos ha generado la siguiente salida en la máquina `sirio.edv.uniovi.es` (la misma que hemos usado en las prácticas durante el curso) y que sabemos que tiene un sistema operativo SunOS. En esa máquina se pretende generar los archivos ejecutables del cliente y del servidor de la aplicación, `cliente.exe` y `servidor.exe` respectivamente. ¿Cuáles son las instrucciones de compilación necesarias para generar en esa máquina esos ficheros correctamente?

```
client.c    inter.h     inter.x  inter_clnt.c
inter_svc.c inter_xdr.c server.c
```

```
gcc -o servidor.exe server.c
          inter_svc.c inter_xdr.c -lnsl
gcc -o cliente.exe cliente.c
        inter_clnt.c inter_xdr.c -lnsl
```

Explicación: En las máquinas SunOS hay que tener en cuenta que hay que incluir la librería nsl en la línea de compilación. Y se debe tener cuidado de no incluir el archivo `inter.h` en dicha línea.

— Antes de la invocación del servicio `ActualizarPrev` (utilizando el interface que facilita `rpcgen`) ¿Qué función/es del API de las RPC de Sun Microsystem ha tenido necesariamente que ejecutar el cliente? Escribe el código de la llamada a esa/s función/es declarando todas las variables que utilices.

```
CLIENT *clnt;
char *maq="maquina.servidor.es";
  clnt=clnt_create(maquina,PrevMeteo,
                          MeteoVER,"udp");
  if (clnt== NULL) {
    clnt_pcreateerror(maquina);
    exit(-1);
   }
   clnt->cl_auth=authunix_create_default();
```

Explicación: Para poder invocar un servicio es necesario inicializar una estructura `CLIENT` con los parámetros del servidor que lo contiene. Para ello utilizamos `clnt_create`. Además, como el servicio que queremos ejecutar es un servicio administrativo que no puede ser ejecutado por cualquier cliente, es necesario incluir las credenciales UNIX del cliente en cada invocación, para lo cual tenemos que invocar la macro `authunix_create_default()`.

❑ En la siguiente tabla se muestran las acciones que pueden realizar tanto el cliente como el servidor respecto a la tolerancia a fallos de una RPC. Cada una de esas acciones dará lugar a una semántica concreta. Rellena los huecos de la tabla de manera adecuada.

Cliente	**Servidor**		
Reintenta RPC	**Filtra duplicados**	**Acción**	**Semántica RPC**
No	No aplicable	No aplicable	Quizás
SI	No	Reejecución Servicio	Al menos una vez
SI	SI	Retransmite respuesta	A lo más una vez

❑ Se desea crear un par de claves RSA para poder realizar criptografía de clave pública. Para ello se han escogido como generadores los números primos $P = 79$ y $Q = 47$. Durante el cálculo de la pareja de claves, para la clave pública se escogió el número más pequeño de los posibles, y para la secreta la que resulta del algoritmo RSA.

— ¿Cuál es la pareja de claves $\langle K_P, N\rangle$ y $\langle K_S, N\rangle$ calculada?

$\langle 5, 3713\rangle$ y $\langle 2153, 3713\rangle$

Explicación: Para calcular la pareja de claves aplicamos el algoritmo RSA a partir de los dos números primos dado. Primero calculamos $N = P * Q = 3713$ y $Z = (P-1)(Q-1) = 3588$. Se factoriza Z y nos queda que $Z = 2^2 * 3 * 13 * 23$. En el enunciado se dice que como clave pública se ha escogido el número más pequeño de los posibles por lo que tenemos que escoger el número primo más pequeño que no sea factor de Z. Vemos que ese número es 5 por lo que ya tenemos el valor de la clave pública que será el par $\langle 5, 3713\rangle$. Para encontrar la clave secreta tenemos que encontrar el número K_s tal que $K_p * K_s \bmod Z = 1$ o, dicho de otro modo, el número K_s que verifique que $K_p * K_s = n * Z + 1$. Haciendo variar n comprobamos que para $n = 3$ obtenemos un valor en la serie $n * Z + 1$ que es múltiplo de 5. En este caso tenemos que $5 * K_s = 10765$ lo que nos da el valor de $K_s = 2153$, siendo la clave secreta el par $\langle 2153, 3713\rangle$.

— Se quieren utilizar las claves calculadas para firmar un mensaje. Si el mensaje está compuesto por los siguientes bytes (en hexadecimal):

```
23 7F A0 DD E6
```

¿Cuál ha sido la primera fórmula matemática que se ha aplicado para realizar la firma? (se supone que se firma el mensaje directamente, no su *hash*) Contestar con todos los elementos de la fórmula en **base diez**.

$283^{2153} \bmod 3713$

Explicación: Las fórmulas que permiten encriptar y desencriptar la información con el algorítmo RSA son: $C = M^{K_p} \bmod N$ y $M = C^{K_s} \bmod N$ siendo M un trozo de texto claro de k bits. El valor de k tiene que verificar que $2^k < N$ para garantizar la unicidad de los resultados. En nuestro caso, $k = 11$ ya que $2^{11} < 3713$, $N = 3713$, $K_s = 2153$ y $K_p = 5$. Para firmar el mensaje que se envía, es necesario que utilicemos la clave secreta ya que el destinatario tiene que comprobar la autenticidad usando nuestra clave pública.

Por lo tanto, el mensaje que tenemos que enviar esta compuesto por los bytes $237FA0...$ que en binario tienen la siguiente representación: 001000110111111110100000.... Agrupamos ahora de 11 en 11 bits (00100011011111111010000...), convertimos a decimal (2832024...) y aplicamos la fórmula: $283^{2153} \bmod 3713$

❑ Como parte de un algorimo de cifrado, es necesario obtener el resultado de $53^{34} \bmod 77$. Para ello se aplica el algoritmo de la exponenciación binaria aplicando el módulo tras cada paso, para mantener los números manejables. ¿Cuál es el resultado final de la operación y cuántas multiplicaciones se han tenido que realizar?

Resultado: 60 Multiplicaciones: 8

Explicación: El exponente en binario es `100010`, que tiene 6 bits. El algoritmo de la exponenciación binaria usa tantos pasos como bits tenga el exponente. En cada paso se eleva el resultado del paso anterior al cuadrado (lo cual nos da una multiplicación en cada paso) y además en los pasos que corresponden a los bits de valor `1`, se multiplica también ese resultado por la base (lo que nos daría una multiplicación más para estos pasos). El total de multiplicaciones por tanto será igual al número total de bits, más el número de `1` en el exponente. En este caso, 8 multiplicaciones.

La aplicación del algoritmo va dando los siguientes resultados intermedios:

Paso	bit	Resultado
1	1	53
2	0	37
3	0	60
4	0	58
5	1	37
6	0	60

El resultado final es por tanto 60.

❑ El siguiente código pretende poner en hora el reloj de un computador que forma parte de un sistema distribuído. Para ello hace uso de las siguientes funciones:

- `time(NULL)` Esta función devuelve el contador de segundos de la máquina local, con origen de tiempos en el 1 de Enero de 1970.
- `servicio_time(maquina)` Esta función le pide la hora a la máquina que recibe como parámetro, usando el protocolo `time`. La función devuelve el valor respondido por la máquina (ya con el orden de bytes correcto), que tiene como origen de tiempos el 1 de Enero de 1900.
- `poner_hora(contador)` Cambia la hora local de la máquina, de forma que la nueva hora sea la especificada en `contador`, que debe tener origen de tiempos en 1970. En esta implementación no nos preocupamos por evitar cambios bruscos o retrocesos en dicho reloj local.

```
1 #define OFFSET 2208988800LU
2 long int hora1, hora2, lahora;
3 hora1  = time(NULL);
4 lahora = servicio_time("hora.uniovi.es");
5 hora2  = time(NULL);
6 poner_hora(                                   );
```

— Para estimar la hora correcta a partir de la respuesta del servidor, utiliza el algoritmo de Christian. ¿Qué falta entonces en el hueco?

```
lahora - OFFSET + (hora2 - hora1)/2
```

Explicación: El algoritmo de Christian trata de tener en cuenta los retardos en la red. Es decir, entre el instante en que se pide la hora (instante recogido en `hora1` y el instante en que se recibe (recogido en `hora2`), ha pasado un tiempo debido al retardo de la red. La respuesta que envió el servidor (recogida en `lahora`) está ya anticuada cuando llega al cliente, pues ya es más tarde. Pero ¿cuánto más tarde? El algoritmo de Christian consiste en suponer que la respuesta del servidor se originó en el punto medio entre `hora1` y `hora2`. Por tanto cuando se recibe la hora, la hora real es la que marcaba el servidor más `(hora2-hora1)/2`.
Aparte del ajuste anterior debido al algoritmo de Christian, es necesario otro ajuste debido al diferente origen de tiempos que usa el protocolo time. La respuesta del servidor está medida desde 1900, pero los restantes tiempos que se manejan van medidos desde 1970. Es necesario restar a la respuesta del servidor el "número mágico" recogido en la constante `OFFSET` (que es la cuenta de segundos transcurridos entre 1900 y 1970).

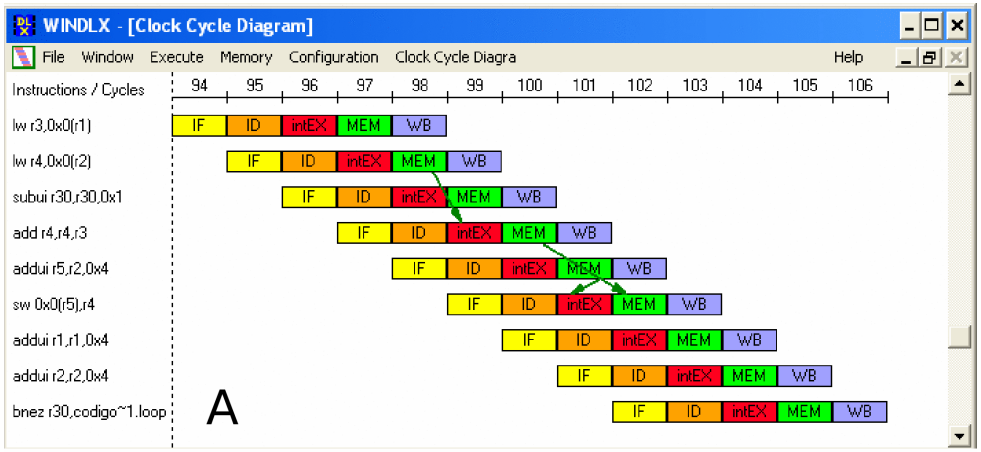
— Si en lugar de una sincronización de relojes físicos se deseara simplemente una sincronización de relojes lógicos con el algoritmo de Lamport ¿que habría que poner en vez de la última línea del programa?

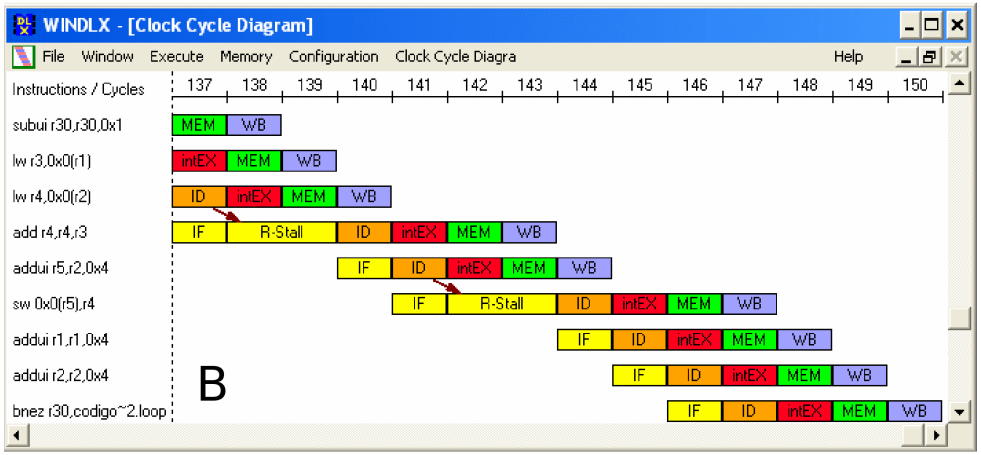
```
if (lahora - OFFSET > hora2)
   poner_hora(lahora - OFFSET + 1);
```

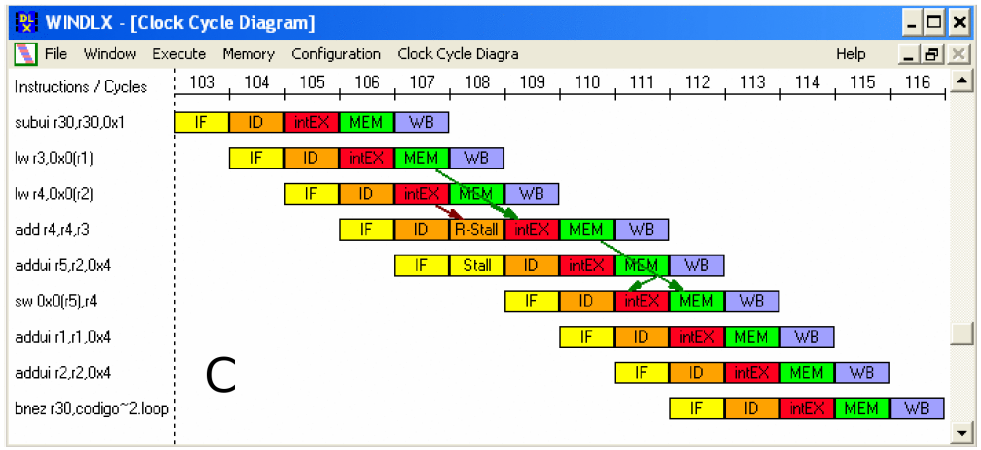
Explicación: Según el algoritmo de Lamport, el reloj lógico se sincroniza sólo en la recepción de un mensaje que contenga una marca de tiempo mayor que la máquina que lo recibe. El mensaje en este caso es la respuesta a la petición de hora, y la marca de

tiempo del mensaje es el valor retornado (menos el `OFFSET` para ajustar el origen de tiempos). Por tanto si esta marca es mayor que `hora2`, el reloj local debe actualizarse a un valor que sea mayor que dicha marca. Simplemente añadiendo 1 a la marca serviría. Esto se pasaría como parámetro a la función `poner_hora` para que el reloj local quede actualizado. De estas consideraciones se deduce el código de la respuesta.

Anexo


WINDLX - [Clock Cycle Diagram]
File Window Execute Memory Configuration Clock Cycle Diagra Help
Instructions / Cycles
94 95 96 97 98 99 100 101 102 103 104 105 106
lw r3,0x0(r1)
lw r4,0x0(r2)
subui r30,r30,0x1
add r4,r4,r3
addui r5,r2,0x4
sw 0x0(r5),r4
addui r1,r1,0x4
addui r2,r2,0x4
bnez r30,codigo~1.loop
IF ID intEX MEM WB
A


WINDLX - [Clock Cycle Diagram]
File Window Execute Memory Configuration Clock Cycle Diagra Help
Instructions / Cycles
137 138 139 140 141 142 143 144 145 146 147 148 149 150
subui r30,r30,0x1
lw r3,0x0(r1)
lw r4,0x0(r2)
add r4,r4,r3
addui r5,r2,0x4
sw 0x0(r5),r4
addui r1,r1,0x4
addui r2,r2,0x4
bnez r30,codigo~2.loop
MEM WB
intEX MEM WB
ID intEX MEM WB
IF R-Stall ID intEX MEM WB
IF ID intEX MEM WB
IF R-Stall ID intEX MEM WB
IF ID intEX MEM WB
IF ID intEX MEM WB
IF ID intEX MEM WB
B


WINDLX - [Clock Cycle Diagram]
File Window Execute Memory Configuration Clock Cycle Diagra Help
Instructions / Cycles
103 104 105 106 107 108 109 110 111 112 113 114 115 116
subui r30,r30,0x1
lw r3,0x0(r1)
lw r4,0x0(r2)
add r4,r4,r3
addui r5,r2,0x4
sw 0x0(r5),r4
addui r1,r1,0x4
addui r2,r2,0x4
bnez r30,codigo~2.loop
IF ID intEX MEM WB
IF ID intEX MEM WB
IF ID intEX MEM WB
IF ID R-Stall intEX MEM WB
IF Stall ID intEX MEM WB
IF ID intEX MEM WB
IF ID intEX MEM WB
IF ID intEX MEM WB
IF ID intEX MEM WB
C