Exercicis Tema 14

S'ha d'entregar un únic fitxer PDF que inclogui la solució que vosaltres proposeu als problemes plantejats. El fitxer PDF no te que ser necessàriament una solució feta per ordinador, pot ser una solució escrita a ma i digitalitzada. El PDF ha d'incloure una capçalera on s'indiqui el vostre nom i cognoms, i l'enunciat de cada pregunta abans de la vostra resposta.

Exercici 1

Especificad el **camino crítico** (indicando la suma ordenada de los tiempos de propagación de los bloques por los que pasa) y calculad el **tiempo de ciclo mínimo** para que el computador **SISC Von Neumann** pueda ejecutar correctamente el tipo de instrucción SISA que se indica en cada apartado (este sería el tiempo de ciclo mínimo del computador si solo ejecutara instrucciones como la indicada u otras que requieran menor tiempo). No tenéis que añadir ningún porcentaje de seguridad en el cálculo del tiempo de ciclo mínimo. Suponed que los tiempos de propagación de los bloques que forman el computador son los siguientes:

T_p(ROM_Q+) = 70 u.t.

T_p(ROM_OUT) = 90 u.t.

T_p(MUX-2-1) = 60 u.t.

T_p(MUX-4-1) = 110 u.t.

T_p(REG) = 100 u.t. // Tiempo de propagación de un registro.

T_p(REGFILE) = 200 u.t. // Tiempo de lectura del banco de registros

T_p(ALU-slow) = 900 u.t. // Tp de la ALU para las operaciones/funciones lentas: `ADD`, `SUB`, `CMP*`.

T_p(ALU-quick) = 450 u.t. // Tp de la ALU para las operaciones/funciones rápidas: cualquier otra distinta de `ADD`, `SUB`, `CMP*`.

T_{acc}(MEMORY) = 1100 u.t. // Tiempo de acceso (para la lectura o escritura) a la memoria

T_p(AND-2) = T_p(OR-2) = 20 u.t.

T_p(NOT) = 10 u.t.

El tiempo de propagación de un bloque combinacional (T_p) y el tiempo de acceso a memoria para realizar una lectura (T_{acc}) es el tiempo desde que están estables todas las entradas necesarias hasta que se estabilizan las salidas requeridas al valor correcto para las entradas aplicadas. Desconocemos como se han implementado internamente los bloques (y podría ser de forma diferente a los vistos en clase). Recordad que un registro con señal de carga (Ld), REGwLd, está construido con un REG y un MUX-2-1 (no os damos el esquema interno del REGwLd, porque lo tenéis que saber).

a) T_c correspondiente al nodo de **D** (decode).

b) T_c correspondiente al nodo de **Addi**.

c) T_c correspondiente al nodo de **Ldb**.

d) T_c correspondiente al nodo de **Bnz**.

Exercici 2

El fragmento de programa en ensamblador SISA que se muestra a continuación se ha traducido a lenguaje máquina situando la sección de datos (.data) a partir de la dirección `0x0100` y la sección de código (.text) a partir de la dirección `0xC000`. Antes de la primera instrucción que se muestra (`MOVI R5,0`) hay 150 instrucciones que no se muestran y después de la última instrucción que se muestra (`ST 0(R2),R5`) hay 20 instrucciones que tampoco se muestran.

```
.data
        long=6
        A: .word 1,-4, 15, 0, 12
        B: .word 13, 2, 14, -2, 9
        C: .space A
        D: .space 1
.text
        . . .
        MOVI  R5, 0
        MOVI  R1, lo(A)
```

```
        MOVHI R1, hi(A)
        MOVI  R2, lo(C)
        MOVHI R2, hi(C)
        ADDI  R0, R1, long
loop:   LD    R3, 0(R1)
        ADDI  R1, R1, 2
        LD    R4, long(R1)
        ADD   R7, R3, R4
        ADD   R3, R3, R7
        ST    0(R2), R3
        ADDI  R2, R2, 2
        ADD   R5, R5, R3
        CMPLEU  R7, R1, R0
        BNZ   R7, loop
endlp:  ST    0(R2), R5
        . . .
.end
```

Una vez ensamblado y cargado el programa en la memoria y antes de comenzar su ejecución:

a) ¿A qué direcciones de memoria corresponden las etiquetas o direcciones simbólicas siguientes?

D=0x loop=0x

b) ¿Cuál es la dirección de memoria y su contenido donde han quedado almacenadas las siguientes instrucciones una vez cargado el programa?

Instrucción	@ memoria y contenido
`MOVHI R2, hi(C)`	Mem_w[0x] = 0x
`LD R4, long(R1)`	Mem_w[0x] = 0x
`BNZ R7, loop`	Mem_w[0x] = 0x

c) Una vez ejecutado el programa en el SISC von Neumann ¿Cuál es la dirección de la memoria donde ha escrito la última instrucción (`ST 0(R2), R5`) y cuál es su contenido? Suponed que las instrucciones que hay antes de la primera instrucción del fragmento de programa (`MOVI R5,0`) no modifican el estado del computador excepto el PC.

Mem_w[0x] = 0x

d) ¿Cuántas instrucciones se ejecutan del código que se muestra, cuántas son lentas y cuántas son rápidas en la Harvard multiciclo y cuantos ciclos tarda desde que se ejecuta la primera instrucción del código (`MOVI R5,0`) hasta que se ejecuta la última instrucción del código (`ST 0(R2),R5`), ambas incluidas? ¿Cuánto tarda en ejecutarse el código en el Harvard uniciclo, en el Hardard multiciclo y en el Von Neumann suponiendo que los tiempos de ciclo son 3.000, 2000 y 1.000 u.t. respectivamente?

Nº de instruc. ejecutadas = Nº instr. lentas (H. multiciclo)= Nº instr. rápidas (H. multiciclo)=

Nº de ciclos (H. uniciclo)= Nº de ciclos (H. multiciclo)= Nº de ciclos (Von Neumann)=

Tejec(Harvard uniciclo) = Tejec(Harvard multiciclo) = Tejec(Von Neumann) =

e) Supongamos que el código ha cambiado y entre la primera instrucción del bucle y la instrucción de salto hay más de 200 instrucciones.

```
.text
        . . .
loop:   LD    R3, 0(R1)
        . . .               ; suponed que hay 200 instrucciones
        CMPLEU  R7, R1, R0
        BNZ   R7, loop
endlp:  ST    0(R2), R5
        . . .
.end
```

Ahora la instrucción de salto ya no puede ser directamente un BNZ ya que esta fuera del alcance de la constante que se puede codificar en el formato de instrucción de los saltos. Suponed que el PC de la instrucción BNZ vale 0x3500, ¿Cuál sería la dirección de la instrucción anterior y posterior más alejadas a las que podríamos acceder con la instrucción de salto?

@Mem instrucción anterior = 0x

@Mem instrucción posterior = 0x

f) Una posible solución a este inconveniente es la que se presenta incompleta a continuación. Completa los huecos que faltan para que el código siga funcionando aunque haya más de 200 instrucciones entre origen y el destino del salto.

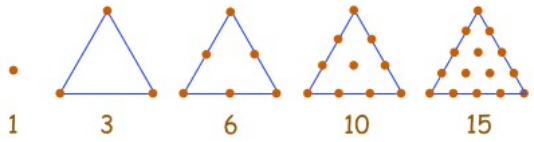
Recordad que la semántica del JALR es:

Sintaxis ensamblador:	`JALR Rd, Ra`
Semántica:	`PC = PC+2; tmp=Ra&(~1); Rd=PC; PC=tmp;`
Lenguaje máquina:	`0111 aaa ddd xxxxxx`

```
loop:  LD    R3, 0(R1)
       . . .            ;suponed que hay 200 instrucciones y que no usan registros adicionales
       CMPLEU  R7, R1, R0
       BZ    R7, endlp

       _______________

       _______________

       JALR   ________
endlp: ST    0(R2), R5
```

Exercici 3

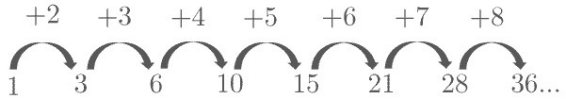
Una de las sucesiones más antiguas estudiadas es la correspondiente a los números triangulares. Un número triangular es aquel que puede recomponerse en la forma de un triángulo equilátero (por convención, el primer número triangular es el 1). Los números triangulares, junto con otros números figurados, fueron objeto de estudio por Pitágoras y los Pitagóricos, quienes consideraban sagrado el 10 escrito en forma triangular, y al que llamaban Tetraktys. Esta sucesión se genera con un patrón de puntos que forma un triángulo. Añadiendo otra fila de puntos y contando el total se encuentra el siguiente número de la sucesión.



Numéricamente la sucesión se escribe como 1, 3, 6, 10, 15, 21, 28, 36, 45, ... Donde cada número triangular T_n está definido por la siguiente fórmula: $T_n = \frac{n(n+1)}{2}$

a) Completad el código SISA para que calcule *k* primeros elementos de la sucesión de números triangulares y los almacene en memoria a partir de la posición 23456. En la posición 23456 deberá almacenarse el primer valor de la sucesión (T_0), en la 23458 el segundo valor de la sucesión (T_1) y así sucesivamente hasta *k* elementos. El código debe rellenar las *k* primeras posiciones de memoria a partir de la posición 23456 con los valores de la sucesión. Los valores son números naturales de 16 bits. En caso de que el valor de la sucesión calculado exceda los 16 bits el programa se queda con los 16 bits de menor peso. El número de valores de la sucesión a calcular (*k*) nos vendrá indicado mediante un dispositivo de teclado con efecto lateral en la lectura que tiene el registro de status en la posición 9 y el de datos en la 4. Y se nos garantiza que este valor de *k* siempre estará comprendido entre 1 y 5000.

La regla de formación de los números triangulares es que cada término se obtiene sumando al anterior la cantidad correspondiente a su número de orden, como podemos observar en la siguiente figura.



@Mem				
0x0000	.data			
0x0001	.text	MO___	R5, _____	; R5=primera posición de memoria
0x0002		MO___	R5, _____	
0x0003		MO___	R6, _____	; R6=primer valor
0x0004		MO___	R4, _____	; R4=número de orden
0x0005	espera:	_____	R7, _____	; hay dato?
0x0006		_____	R7, _____	
0x0007		_____	R1, _____	; numero elementos
0x0008	bucle:	_____	0(____), ___	
0x0009		_____	R5,____, ___	
0x000A		_____	R4,____, ___	
0x000B		_____	R6,____, ___	
0x000C		_____	R1,____, ___	
0x000D		_____	R1, _____	
0x000E	.end			

b) ¿Cuántos ciclos y unidades de tiempo tarda en ejecutarse el código anterior en Harvard uniciclo y en el Von Neumann, considerando que el número de elementos recibidos es β, que el valor β ya está disponible en los puertos de entrada/salida en el momento de empezar la ejecución del programa y que el tiempo de ciclo del Harvard uniciclo es de 4000 u.t. y el del Von Neumann es de 1000 u.t.?

Código en el Harvard uniciclo: Número de ciclos= Tejec=

Código en el Von Neumann: Número de ciclos= Tejec=