

EXERCICIS PAPER 14

①

a) Decode: càlcul adreça destí de salt.

Cami crític: $REG_Q \rightarrow M \times N \rightarrow N \rightarrow R_y / N \rightarrow ALU \rightarrow RQ$.

$$T_p = 100 (REG_Q) + 90 (ROM_OUT) + 110 (MUX_{4-1} M \times N \text{ selecció}) + 60 (MUX R_y / N) + 900 (ALU ADD) = \boxed{1260 \text{ u.t}}$$

b) Addi:

Cami crític: $REG_Q \rightarrow ROM_OUT \rightarrow M \times N \rightarrow N \rightarrow R_y / N \rightarrow ALU \rightarrow M \times 4-1 \rightarrow M \times 2-1$.

$$T_p = 100 (REG_Q) + 90 (ROM_OUT) + 110 (MUX_{4-1} M \times N \text{ selecció}) + 60 (MUX R_y / N) + 900 (ALU ADD) + 110 (MUX_{4-2} P.I.L.A) + 60 (MUX_{2-1} Ld REG_w) = \boxed{1430 \text{ u.t}}$$

(REGFILE)

c) Ldb:

Cami crític: $REG_Q \rightarrow RQ / PC \rightarrow ADDR_MEM \rightarrow RD_MEM \rightarrow P/I/L/A \rightarrow REGFILE$

$$T_p = 100 (REG_Q) + 90 (ROM_OUT) + 60 (MUX_{2-1} RQ / PC) + 1100 (RAM Ldb) + 110 (MUX_{4-1} P/I/L/A) + 60 (dada registre càrrega Rd) = \boxed{1210 \text{ u.t}}$$

d) Bnz:

Cami crític: $REG_Q \rightarrow ROM_OUT \rightarrow PC/Rx \rightarrow ALU \rightarrow AND-2 \rightarrow OR-2 \rightarrow REG_PC$

$$T_p = 100 (REG_Q) + 90 (ROM_OUT) + 60 (MUX PC/Rx) + 150 (ALU-quick) + 20 + 20 = \boxed{740 \text{ u.t}}$$

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:

$T_p(\text{ROM_Q+}) = 70 \text{ u.t.}$

$T_p(\text{ROM_OUT}) = 90 \text{ u.t.}$

$T_p(\text{MUX-2-1}) = 60 \text{ u.t.}$

$T_p(\text{MUX-4-1}) = 110 \text{ u.t.}$

$T_p(\text{REG}) = 100 \text{ u.t.}$ // Tiempo de propagación de un registro.

$T_p(\text{REGFILE}) = 200 \text{ u.t.}$ // Tiempo de lectura del banco de registros

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

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

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

$T_p(\text{AND-2}) = T_p(\text{OR-2}) = 20 \text{ u.t.}$

$T_p(\text{NOT}) = 10 \text{ u.t.}$

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 0×0100 y la sección de código (.text) a partir de la dirección $0 \times C000$. 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
    0x0114C: .space A
    D: .space 1
.text
    ...
    0x4A2C MOVI R5, 0
    0x4A2E MOVI R1, 10(A)
```



```

C130 MOVHI R1, hi(A)
C132 MOVI R2, lo(C)
C134 MOVHI R2, hi(C)
C136 ADDI R0, R1, long
C138 loop: LD R3, 0(R1)
C13A ADDI R1, R1, 2
C13C LD R4, long(R1)
C13E ADD R7, R3, R4
C140 ADD R3, R3, R7
C142 ST 0(R2), R3
C144 ADDI R2, R2, 2
C146 ADD R5, R5, R3
C148 CMPLU R7, R1, R0
C14A BNZ R7, loop
C14C 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=0x0115 loop=0xC138

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 [0xC134] = 0x950A
LD R4, long(R1)	Mem _w [0xC13C] = 0x3306
BNZ R7, loop	Mem _w [0xC14A] = 0x8FF6

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[0xC14C] = 0x4540

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 = 47 Nº instr. lentas (H. multiciclo) = 13 Nº instr. rápidas (H. multiciclo) = 34

Nº de ciclos (H. uniciclo) = 47 Nº de ciclos (H. multiciclo) = 154 Nº de ciclos (Von Neumann) = 154

Tejec(Harvard uniciclo) = 141.000 u.t Tejec(Harvard multiciclo) = 308.000 u.t Tejec(Von Neumann) = 154.000 u.t

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
    CMPLU 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 = 0x348A

@Mem instrucción posterior = 0x3580

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

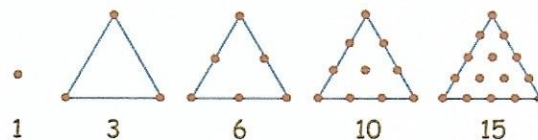
Semántica: PC = PC+2; tmp=Ra&(~1); Rd=PC; PC=tmp;

Lenguaje máquina: 0111 aaa ddd xxxxxx

```
loop: LD    R3, 0(R1)
      . . . ;suponed que hay 200 instrucciones y que no usan registros adicionales
      CMPLU R7, R1, R0
      BZ    R7, endlp
      MOVI R7, lo[loop]
      MOVHI R7, hi[loop]
      JALR  Rd, R7
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.



Númericamente 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.

