Apellidos: ..... Nombre: ..... Grup: .....

Examen final IC (*Parte 2)*

- Duración del examen: 2:30 horas.
- No podéis utilizar calculadora, móvil, apuntes, etc.
- La solución del examen se publicará en Atenea mañana por la mañana.

Nota: La indicación de la puntuación de los ejercicios es sobre 10 puntos, pero esta parte del examen final solo representa 5 puntos de la nota del examen final.

Ejercicio 1 *(1.9 puntos)*

El programa en ensamblador SISA que se muestra a continuación se ha traducido a lenguaje máquina para ser ejecutado en el SISC Von Neumann, situando la sección de datos (`.data`) a partir de la dirección **`0x6000`** y la sección de código (`.text`) a partir de la dirección **`0x9500`**.

```
.data
        long=6
        A: .byte 0X06, 0x88, 0x82, 0xFA, 0x25, 0xBA
           .even
        B: .word 0x000F, 0x9A01, 0x0102, 0xFFFE, 0x0010, 0XF2A3
        C: .space 24,10
        D: .word -1
.text
        MOVI  R3, long
        MOVI  R1, lo(A)
        MOVHI R1, hi(A)
        MOVI  R2, lo(B)
        MOVHI R2, hi(B)
        MOVI  R7, 1
loop:   LDB   R5, 0(R1)
        LD    R4, 0(R2)
        SUB   R6, R4, R5
        AND   R0, R6, R7
        BNZ   R0, impar
par:    ST    16(R2), R6
        BZ    R0, cont
impar:  ST    16(R2), R5
cont:   ADDI  R1, R1, 1
        ADDI  R2, R2, 2
        ADDI  R3, R3, -2
        BNZ   R3, loop
.end
```

a) Una vez ensamblado y cargado el programa en la memoria, ¿A qué direcciones de memoria corresponden las etiquetas o direcciones simbólicas siguientes? *(0.4 puntos)*

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? *(0.6 puntos)*

Instrucción	@ memoria y contenido
`SUB R6, R4, R5`	Mem_w[0x] = 0x
`BNZ R0, impar`	Mem_w[0x] = 0x
`ST 16(R2), R5`	Mem_w[0x] = 0x

c) Una vez ejecutado el programa en el SISC von Neumann, completa la siguiente tabla con el contenido de la memoria en hexadecimal (no es necesario poner 0x del valor). Las posiciones de memoria que no se pueda saber su contenido dejadlas en blanco. *(0.6 puntos)*

@memoria	Valor
0x6000	
0x6001	
0x6002	
0x6003	
0x6004	
0x6005	
0x6006	
0x6007	

@memoria	Valor
0x6008	
0x6009	
0x600A	
0x600B	
0x600C	
0x600D	
0x600E	
0x600F	

@memoria	Valor
0x6010	
0x6011	
0x6012	
0x6013	
0x6014	
0x6015	
0x6016	
0x6017	

@memoria	Valor
0x6018	
0x6019	
0x601A	
0x601B	
0x601C	
0x601D	
0x601E	
0x601F	

d) Si se ejecutase el programa en el computador en la SISC Harvard multiciclo, indicad el número total de instrucciones que ejecutaría, así como cuántas son lentas y cuántas son rápidas. *(0.3 puntos)*

N_{total} = ____ N_{lentas} = ____ $N_{rápidas}$= ____

Ejercicio 2 *(2 puntos)*

Cada uno de los apartados pregunta sobre un ciclo concreto de la ejecución de varias instrucciones en el SISC Von Neumann. Escribid el contenido del registro IR, el contenido de la ROM_OUT y el valor de los bits de la **palabra de control** que genera el bloque **SISC CONTROL UNIT** durante el ciclo a que hace referencia cada apartado**.** Para cada apartado/fila se indica el nodo/estado de la UC en ese ciclo y la instrucción (en ensamblador) que está almacenada en el IR en ese ciclo. Podéis ver el grafo de estados de Moore de la UCG en el anexo. Suponed que el contenido de todos los registros, `Rk` para k=0,...7, antes de ejecutarse cada instrucción es 0.

a) Indica el contenido del registro IR. Utilizad el valor 0 para los bits que sean x (solo para este apartado). *(0.4 puntos)*

Nodo / Estado (Mnemo Salida)	**Instrucción en IR** (en ensamblador)	**Valor del IR** (en hexadecimal)
Cmp	`CPMLTU R1, R2, R3`	0x
D	`BNZ R4, -16`	0x
In	`IN R5, 50`	0x
Movi	`MOVI R6, 0xAC`	0x

b) Indica el contenido de la ROM_OUT en hexadecimal usando las conexiones en el orden que están en el anexo. Utilizad el valor 0 para los bits que sean x (solo para este apartado). *(0.8 puntos)*

Nodo / Estado (Mnemo Salida)	**Instrucción en IR** (en ensamblador)	**Contenido ROM_OUT** (en hexadecimal)
Cmp	`CPMLTU R1, R2, R3`	0x
D	`BNZ R4, -16`	0x
In	`IN R5, 50`	0x
Movi	`MOVI R6, 0xAC`	0x

c) Indica el valor de los bits de la **palabra de control** que genera el bloque **SISC CONTROL UNIT** durante el ciclo a que hace referencia cada apartado**. Poned x siempre que no se pueda saber el valor de un bit** (ya que no podemos suponer cómo se han implementado las x en la ROM_OUT). *(0.8 puntos)*

Nodo / Estado (Mnemo Salida)	**Instrucción en IR** (en ensamblador)	**Palabra de Control**																		
		@A	@B	Pc/Rx	Ry/N	OP	F	P/I/L/A	@D	WrD	Wr-Out	Rd-In	Wr-Mem	Ldlr	LdPc	Byte	Alu/R@	R@/Pc	N (hexa)	ADDR-IO (hexa)
Cmp	`CPMLTU R1,R2,R3`																			
D	`BNZ R4, -16`																			
In	`IN R5, 50`																			
Movi	`MOVI R6, 0xAC`																			

Apellidos: ..... Nombre: ..... Grup: .....

Ejercicio 3 *(0.9 puntos)*

Indicad qué cambios se producen en el estado del computador SISC Von Neumann después de ejecutar cada una de las instrucciones de la tabla suponiendo que antes de ejecutarse cada una de ellas el PC vale `0x86AC`, el contenido de todos los registros pares es `0x6789` y el de los registros impares es `0x0123`, el byte contenido en todas las direcciones pares de la memoria es `0x64` y el de todas las impares es `0x32` y el contenido de todos los puertos de entrada es `1` y el de los de salida es `2`. Utilizad la notación MEM_b[0x…]=0x... y/o MEM_w[0x…]=0x... para indicar cualquier cambio en la memoria y PORTIN[0x...]=0x... para los puertos de entrada y PORTOUT[0x...]=0x... para los de salida.

Instrucción a ejecutar	**Cambios en el estado del computador**
`STB -11(R2), R4`	
`JALR R2, R4`	
`LD R6, 17(R3)`	

Ejercicio 4 *(1.8 puntos)*

Se ha conectado a la UPG un dispositivo externo de entrada que nos envía valores y que tiene el registro de status en la dirección 9 del espacio de direccionamiento de entrada y el de datos en la 12. También se ha conectado un dispositivo externo de salida que tiene el registro de status en la dirección 3 del espacio de direccionamiento de entrada y el de datos en la dirección 6 del espacio de direccionamiento de salida. Ambos dispositivos tienen un efecto lateral en la lectura/escritura del dato sobre su registro de estado.

Se desea implementar un algoritmo en lenguaje SISA para que reciba 50 valores (words) desde el dispositivo de entrada y los envíe al dispositivo de salida en el orden inverso en el que se han recibido. Escribid el programa en código ensamblador SISA para que realice esta función. Se valorará la claridad y brevedad del código. El código no debe ocupar más de 20 instrucciones.

Ejercicio 5 *(1.2 puntos)*

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+) = 80 u.t.

T_p(ROM_OUT) = 110 u.t.

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

T_p(MUX-4-1) = 120 u.t.

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

T_p(REGFILE) = 300 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) = 500 u.t. // Tp de la ALU para las operaciones/funciones rápidas: cualquier otra distinta de ADD, SUB, CMP*.

T_{acc}(64KB MEMORY) = 1000 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).

T_c=

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

T_c=

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

T_c=

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

T_c=

Apellidos: ..... Nombre: ..... Grup: .....

Ejercicio 6 *(2.2 puntos)*

Una manera bastante eficiente de recorrer completamente en orden un vector de elementos que se encuentra en memoria consiste en inicializar un registro con la dirección del primer elemento del vector, calcular la dirección siguiente al último elemento del vector (@inicial + longitud_vector * tamaño_elemento), y crear un bucle para recorrer el vector, donde en cada iteración se incremente el registro que contiene la dirección inicial del vector en un elemento. Este bucle se repite mientras la dirección almacenada en el registro no coincide con la final. Se ha decidido crear una nueva instrucción (**ADD2BNE**) que haga la mayor parte de este trabajo.

Veamos un ejemplo. Supongamos que tenemos un vector A de 10 words que queremos inicializar con el valor -8. El código SISA original y con la nueva instrucción para realizar la tarea serían los siguientes:

Código original

```
        MOVI   R0,-8
        MOVI   R1,lo(A)
        MOVHI  R1,hi(A)
        ADDI   R2, R1, 20
bucle:  ST     0(R1),R0
        ADDI   R1, R1, 2
        CMPEQ  R7, R1, R2
        BZ     R7, bucle
```

Código con la nueva instrucción

```
        MOVI   R0,-8
        MOVI   R1,lo(A)
        MOVHI  R1,hi(A)
        ADDI   R2, R1, 20
bucle:  ST     0(R1),R0
        ADD2BNE R1, R2, bucle
```

Completad el diseño del SISC Von Neumann para que pueda ejecutar, además de las 25 instrucciones originales SISA, una nueva instrucción **ADD2BNE**. Que tiene el formato y codificación, la sintaxis ensamblador y la semántica siguientes:

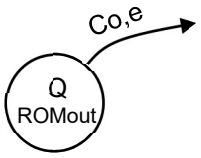
Codificación: `1111 aaa bbb nnnnnn`
Sintaxis: `ADD2BNE Ra, Rb, N6`
Semántica: `PC=PC+2; Ra=Ra+2; if (Ra!=Rb) {PC=PC+SE(N6)*2}`

La instrucción incrementa el valor del registro `Ra`, lo compara con el de `Rb` y en caso que sean distintos, entonces rompe el secuenciamento implícito saltando, a partir de la siguiente instrucción, tantas instrucciones como indique la constante N6 (similar a los saltos `BZ` y `BNZ`).

Para añadir esta nueva instrucción no es necesario modificar el hardware del procesador SISC Von Neumann. Completad el diseño del computador para que ejecute, además de las instrucciones originales, la nueva instrucción **ADD2BNE** sin modificar el hardware y sólo modificando el contenido de las ROM's. Se pide:

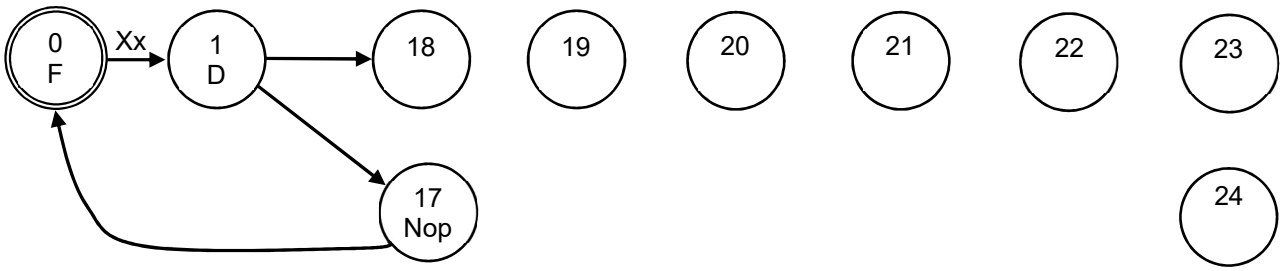
a) Completad el fragmento del grafo de estados de la figura correspondiente al circuito secuencial de la unidad de control. Se da la leyenda del grafo y los nodos necesarios para ejecutar la nueva instrucción, pero faltan arcos y etiquetas. Dibujad todos los arcos que faltan y todas las etiquetas. En número de nodos necesarios para ejecutar la nueva instrucción dependerá de la solución que propongáis, pero no serán más de 7 y deberá ser coherente con las acciones que pongáis en el apartado b). Los nodos que os sobren podéis dejarlos sin conectar. *(0.2 puntos)*

Leyenda:



CO: Código de operación de la Instrucción, I_{15} I_{14} I_{13} I_{12} (en hexadecimal)
e: Bit de extensión del código de operación (I_8)
Q: Estado (en decimal)
ROMout: Mnemotécnico de salida

Grafo:



b) Completad el contenido de la siguiente tabla que indica, mediante una fila para cada nodo del grafo de estados de la unidad de control, la acción (o acciones en paralelo) que se realiza en el computador en cada uno de los ciclos/nodos que requiere la ejecución de la nueva instrucción (Fetch, Decode, y los ciclos/nodos de la ejecución propiamente dicha). Para especificar las acciones usad el mismo lenguaje de transferencia de registros que en la documentación. En número de filas/nodos de la tabla a rellenar dependerá de la solución que propongáis y deberá ser coherente con el apartado a) y c). Nota: Existe una solución solo con 6 nodos (F, D y 4 nodos nuevos). *(0.8 puntos)*

Nodo	Mnemotécnico	Acciones
E0	F	IR ← Mem_w[PC] // PC ← PC+2
E1	D	RX ← Ra // RY ← Rb // R@ ← PC+SE(N8)*2
E18		
E19		
E20		
E21		
E22		
E23		
E24		

Si la solución propuesta en el apartado b) no permite ejecutar correctamente la nueva instrucción, no se corregirán los apartados a) ni c).

c) Completad (poniendo 0, 1 o x en cada bit) la siguiente tabla que especifica, mediante una fila por cada nuevo nodo añadido al grafo de estados de la unidad de control, la dirección (en decimal) de la ROM_OUT y su contenido para que se ejecute correctamente la nueva instrucción. **Poned x siempre que el valor de un bit no importe**. En la columna de la derecha poned el mnemotécnico que habéis usado para nombrar cada nuevo nodo en el grafo del apartado b). *(0.8 puntos)*

@ROM	Bnz	Bz	WrMem	RdIn	WrOut	WrD	LdIr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A_1 P/I/L/A_0	OP_1 OP_0	MxN_1 MxN_0	MxF	F_2 F_1 F_0	Mx@D_1 Mx@D_0	**Nodo (Mnemo)**
18																			
19																			
20																			
21																			
22																			
23																			
24																			

d) Completad la dirección o contenido, según corresponda, de la ROM_Q+ del SISC Von Neumann. *(0.2 puntos)*

`ROM_Q+[0x0A8]=` `0x____` `ROM_Q+[` `0x____` `]=0x10`

e) Si ejecutásemos el código de ejemplo en el procesador original y en el que tiene la nueva instrucción, suponiendo que se ha implementado usando 6 nodos (F, D y 4 nodos nuevos). ¿Cuántos ciclos tardaría en cada caso? *(0.2 puntos)*

Nciclos Original = _____ Nciclos Nueva instrucción = _____