Apellidos y Nombre: .....Grupo:.....DNI: .....

Examen Final, parte II

- Duración de esta parte: 2 horas. La solución de cada ejercicio se tiene que escribir en el espacio reservado para ello en el propio enunciado.
- No podéis utilizar calculadora, móvil, apuntes, etc. En hoja aparte se os da una "chuleta" con información útil para realizar los ejercicios.
- La solución se publicará mañana y las notas el próximo día 21. La revisión será el próximo lunes 25 a las 12:00 en el aula C6-E101.

Ejercicio 1 **(1 punto)**

Vamos a analizar diferentes aspectos de la ejecución, en el computador SISC von Neumann, del siguiente fragmento de código, suponiendo que la constante simbólica `N` vale 20 y la instrucción `MOVI R0,LO(N)` se encuentra almacenada en la palabra de memoria con dirección 0x0F9E:

```
MOVI   R0, LO(N)
MOVHI  R0, HI(N)
ADDI   R0, R0, -1
BNZ    R0, -2
```

a) ¿Cuánto valdrá el contenido de `R0` y del `PC` después de que se ejecute la instrucción `BNZ R0,-2` por quinta vez. (**0,1 puntos**)

R0 = 0x , PC = 0x ,

b) Sea Nc el número de ciclos que tarda en ejecutarse el código. ¿Cuánto vale Nc? (**0,1 puntos**)

Nc =

c) Completad las 4 filas de la siguiente tabla que hacen referencia a los ciclos de ejecución 9, 10, 11 y 12 (los ciclos de ejecución se numeran desde el 1 al Nc). Podéis ver los mnemotécnicos de salida de cada Nodo en el grafo en la chuleta. En la columna Acciones usad el lenguaje de transferencia de registros de la documentación de la asignatura (o una notación igual de precisa), que usa para su especificación símbolos generales como Rd, Ra, Rb, RX, RY, N6, N8... Recordad que en la Palabra de control compactada solo hay que expresar el valor de los campos que son significativos en ese ciclo (no hay que indicar los campos que valen x ni las señales de permiso de modificación del estado que valen 0). (**0,4 puntos**)

Ciclo	Nodo (Mnemo Salida)	Acciones	Palabra de control compactada
9			
10			
11			
12			

d) Completad la siguiente tabla, que es una extensión de la anterior con nuevas columnas y solo dos filas (ciclos 5 y 12), aunque ahora las filas son más estrechas. Repetimos las dos primeras columnas para que quede más claro. En la columna Instrucción en IR poned "No se sabe" cuando este sea el caso. La palabra de control es la que genera el SISC CONTROL UNIT y por lo tanto solo tenéis que poner x en los campos en los que no podáis saber su valor del campo porque no se sabe cómo se han implementado las x en la ROM_OUT. (**0,2 puntos**)

Ciclo	Nodo (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)
5																					
12																					

e) Completa siguiente tabla, que puede considerarse una continuación de las dos anteriores pero ahora con solo las filas de los ciclos 5 y 6 pero con nuevas columnas: la dirección (en decimal) de la ROM_OUT que se accede en cada ciclo y los 24 bits de su contenido (poned x siempre que un bit pueda valer tanto 0 como 1). (**0,2 puntos**)

Ciclo	Nodo (Mnemo Salida)	@ROM	Bnz	Bz	WrMem	RdIn	WrOut	WrD	Ldlr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0
5																										
6																										

Ejercicio 2 **(1,6 puntos)**

Completad el diseño del SISC Von Neumann para que pueda ejecutar, además de las 25 instrucciones originales SISA, la nueva instrucción

```
WAIT Rd
```

que al ejecutarse modifica el estado del computador dejando el contenido de `Rd` con el valor `0` (además de incrementar el `PC` en 2 como todas las instrucciones (excepto las de salto que pueden dejar otro valor en el `PC`). Pero, a diferencia de la instrucción `MOVI Rd,0` que tarda siempre el mismo número de ciclos, la nueva instrucción tarda un número de ciclos variable, que es función del valor contenido en `Rd` antes de ejecutarse la instrucción. En vez de especificar la semántica de la nueva instrucción como

```
Rd = 0;
```

lo vamos a hacer de forma que explique su implementación y muestre que el tiempo de ejecución es mayor cuanto mayor sea el contenido inicial de `Rd` (0<Rd<2^16). Como su ejecución modifica el `PC` de forma no convencional (aunque finalmente deja el `PC` incrementado en 2) especificamos el incremento del `PC` como si se tratara de una instrucción de salto, aunque no lo es. Su semántica es:

```
PC = PC + 2;
Rd = Rd - 1;
if (Rd != 0) PC = PC - 2;
```

La nueva instrucción se tiene que poder ejecutar **sin efectuar ningún cambio en el computador** excepto el contenido de la ROM_Q+ y el de la ROM_OUT y si se codifica adecuadamente solo se requiere **un nuevo estado** en el grafo de la UC, que llamaremos Wait. La ejecución de las 25 instrucciones SISA originales se hace exactamente igual que antes de introducir la nueva instrucción. Se pide (os recomendamos que hagáis a la vez los apartados a, b y c, ya que están estrechamente relacionados entre sí):

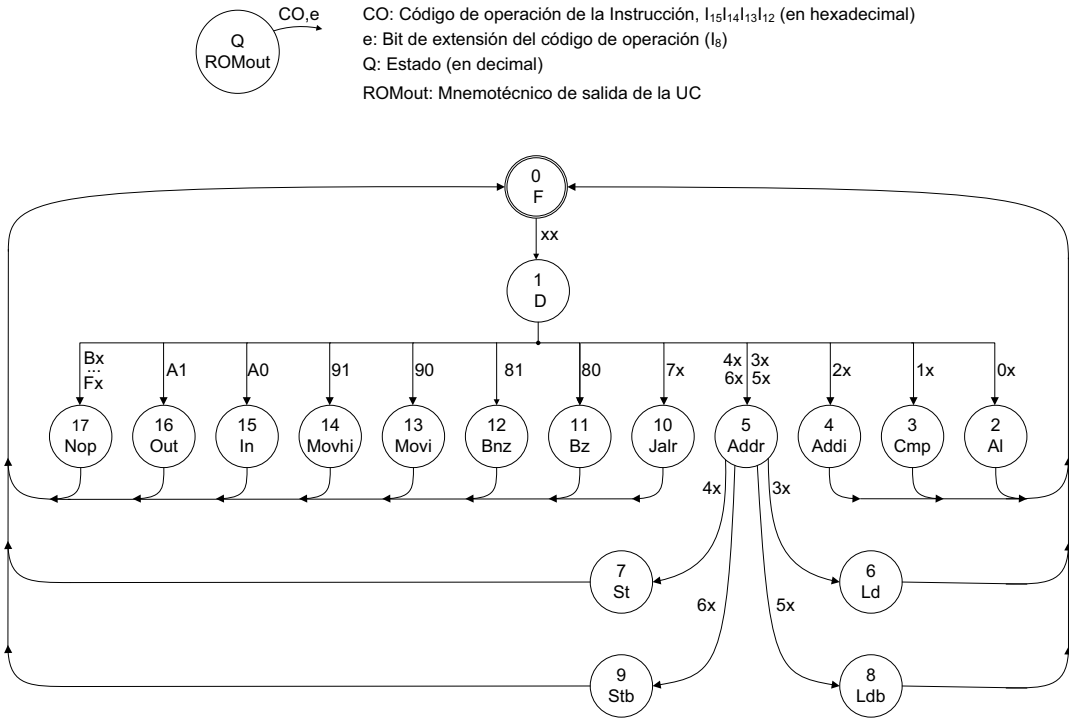
a) Proponed el formato y la codificación de la nueva instrucción (cada bit puede valer `0`, `1`, `a`, `b`, `d`, `n`, `x`) sabiendo que el campo del código de operación es: `1111` (que en el SISA original no se usaba). Solo hay una posible codificación que permite implementar esta instrucción con las restricciones enunciadas: es la clave para poder hacer bien el resto de apartados. (**0,2 puntos**)

`Codificación:`

b) Completad la tabla para el nuevo estado Wait. (**0,5 puntos**)

Nodo		Acciones	Palabra de control compactada
Estado (decimal)	Mnemo salida		
18	Wait		

c) Modificad el grafo de estados de la unidad de control (añadiendo nodos y arcos y/o modificando los ya existentes) dibujando sobre y/o al lado del siguiente grafo original. (**0,2 puntos**)



d) Indicad la dirección o las direcciones (en binario, con x cuando sea posible para referirnos a más de una dirección) de la ROM_Q+ y su contenido (en hexadecimal) para implementar correctamente el arco que va del nodo/estado 1 (con mnemotécnico de salida D) al nuevo nodo Wait usado para implementar la nueva instrucción. (**0,2 puntos**)

Arco del nodo 1 (D) al nodo	**Dirección o direcciones** (en binario)	**Contenido** (en hexa)

e) Completad (poniendo 0, 1 o x en cada bit) la fila de la tabla que especifica para el nuevo nodo Wait la dirección (en decimal) de la ROM_OUT y su contenido para que se ejecute correctamente la nueva instrucción, poniendo el máximo número de x posibles. (**0,3 puntos**)

@ROM	Bnz	Bz	WrMem	RdIn	WrOut	WrD	LdIr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0	**Nodo (Mnemo Salida)**
																									Wait

f) ¿Cuánto debería valer N en el siguiente código, que usa la nueva instrucción, para que tarde en ejecutarse lo mismo que el código del *Ejercicio 1* para N=20? (**0,1 puntos**)

```
MOVI   R0, LO(N)
MOVHI  R0, HI(N)
WAIT   R0
```

N=

g) ¿Cuántos ciclos tardaría en ejecutarse el código del apartado f) si, por error, se usara N=0 (ya se ha dicho que Rd debe ser >0)? (**0,1 puntos**)

Expresión del número de ciclos =

Ejercicio 3 **(1,4 puntos)**

El fragmento de programa en ensamblador, una vez completado, se traducirá a lenguaje máquina para ser ejecutado en el SISC Von Neumann situando la sección `.data` a partir de la dirección 1024 de memoria y a continuación la sección `.text`. Se sabe que en la sección `.text` antes de la instrucción que se encuentra en la dirección simbólica `L0` hay 256 instrucciones.

a) Completad la escritura del fragmento de código para que la ejecución de las instrucciones (desde la `L0` a la `L3`, ambas incluidas) procese **un elemento concreto** del vector `V` y deje el resultado en un elemento del vector `W`. Esto es, se procesa `V[i]` y se deja el resultado en `W[i]`, para un valor de `i` concreto, que se entra por teclado. Se realizan los siguientes pasos: (**0,5 puntos**)

- Primero, después de cargar valores iniciales en registros, se entra desde el teclado un número natural `i` (que podrá ser cualquier número del 0 al 9). Este número es el índice de los vectores `V` y `W` (de 10 elementos cada vector, siendo cada elemento un número natural codificado en binario en una palabra de memoria).
- Después, se encuentra el máximo número natural `k` (en el rango de 0 a 16 ambos incluidos) tal que `V[i]` sea divisible por 2^k (2 elevado a `k`). Un número natural codificado en binarios es divisible por 2^k si tiene los `k` bits de menor peso con valor 0. La búsqueda del `k` máximo se realiza en un bucle que itera desde `k=16` descendiendo hasta `k=1`. En cada iteración comprueba si `V[i]` es divisible por 2^k. La primera vez que la comprobación es verdadera sale del bucle porque ha encontrado el `k`. Si en ninguna de las iteraciones se cumple la comprobación se termina con `k=0`, ya que todos los números son divisibles por 1 = 2^0.
- Por último, se escribe el valor `k` encontrado en el elemento `i` del vector `W` (haciendo `W[i]=k`) y continúa con el código que no se muestra.

Nota: Si se sale en la primera iteración del bucle que comprueba cada valor de `k`, se sale con `k=16`, es porque `V[i]` codifica el 0, que es divisible por cualquier número. Si se sale con `k=15` es porque `V[i]` codifica el 2^15 que es divisible por 2^15 (además de ser divisible por 2^14, 2^13...). Si se sale con `k=14` es porque `V[i]` codifica el 1, 2 o 3 por 2^14 (estos tres números son divisibles por 2^14, además de 2^13...) etc.

Nota: R6=Mascara, R5=Constante, R4=k, R1= vale 0 o vale 1.

```
.data
W:      .space 20
        .even
V:      .word 127,128,512,63,64
        .word 72,250,29,1024,56
.text
        ...
L0:     MOVI    R7,   LO(V)
        MOVHI   R7,
        MOVI    R6,
        MOVI    R5,
        M       R4,
L1:     IN      R1,  KEY
        B       R1,
        IN      R2,  KEY
        SH      R2,   R2,
        A       R7,     ,   R2
        L       R2,    (R7)
L2:     A       R2,   R2,
        B       R2,
        A       R4,   R4,
        SH      R6,     ,   R5
        BNZ     R6,
L3:     S       (R7),   R4
        ...
.end
```

b) Una vez ensamblado y cargado el programa en memoria:

¿A qué dirección de memoria corresponde cada una de las etiquetas, direcciones simbólicas, y qué palabra (word) contiene? (**0,3 puntos**)

`V:`	=>	Mem_w[0x	] = 0x
`L0:`	=>	Mem_w[0x	] = 0x

c) Una vez ejecutado el fragmento de código de la etiqueta `L0` a la `L3` ambas incluidas y suponiendo que antes de ejecutarse el puerto/registro de estado de la impresora contiene un 1 y el de datos un 4:

- ¿Cuál es la dirección de memoria donde ha escrito la instrucción con etiqueta `L3` y cuál es su contenido? (**0,3 puntos**)

Mem_w[0x] = 0x

- ¿Cuántas instrucciones se ejecutan? ¿Cuánto tardaría en ejecutarse el código en el Harvard uniciclo y en el Von Neumann suponiendo que los tiempos de ciclo son 4.000 y 1.000 u.t. respectivamente? ¿Cuánto vale x para que sea cierta la siguiente afirmación? "El computador Von Neumann es un x% **más rápido** que el Harvard uniciclo, ejecutando este código. (**0,3 puntos**).

InstrucEjec = ; Tejec(Harvard uniciclo) = ; Tejec(V.Neumann) = ; x =

