FUNDAMENTOS DE COMPUTADORES II
Examen convocatoria extraordinaria - 30 de junio de 2022

Nombre_____DNI_____

1) **(2 puntos)** Al traducir el siguiente código en lenguaje C a lenguaje ensamblador ARM se han producido una serie de fallos.

Código C

```
/* La función recibe un vector de enteros y un
índice como parámetros.
La función modifica el elemento de la posición
“indice” y devuelve el índice siguiente */

int actualiza(int V[], int indice) {
      v[indice] = v[indice] + 1;
      indice = indice + 1;
      return indice;
}
```

Código ensamblador ARM

```
 1: .global actualiza
 2:
 3: actualiza:
 4:   push {r0-r3, fp, lr}
 5:   add fp, sp, #6*4
 6:
 7:   mov r4, r1       @Asumo V está en r1.
 8:                    @ Copio a r4 para no perder su valor
 9:   mov r5, r2       @Asumo indice está en r2
10:                    @ Copio a r5 para no perder su valor
11:
12:   ldr r0, [r4, r5] @Cargo V[indice]
13:   add r0, r0, #1   @V[indice] + 1
14:   str r0, [r4, r5] @V[indice] = V[indice] + 1
15:
16:   add r5, r5, #1   @indice + 1
17:   mv r2, r5        @actualizo r2 para devolver el nuevo valor
18:
19:   pop {r0-r3,lr}
20:   mov pc, lr
21:
22: .end
```

a) (1 punto) Identificar razonadamente todos los fallos presentes en el código. Recuerda que la función debe seguir el convenio de llamadas visto en clase.
   Nota: Se pide una lista razonada de los fallos presentes en el código, no el código corregido.
b) (1 punto) Asumiendo que la función anterior estuviera correctamente codificada y en un fichero independiente, codificar la siguiente función en ensamblador ARM.
   Nota 1: No es necesario modificar la función anterior para que sea correcta. Es suficiente con asumir que se encuentra correctamente codificada.
   Nota 2: Se debe reservar el espacio para las variables e inicializarlas en las secciones correctas del programa.

Código en C

```
#define N 5
int V[N] = {1, 2, 3, 4, 5};

for(int i=0; i<N; i++)
   if( i == 0 || i == 2)
       actualiza(V, i);
```

2) (3.5 puntos) Se quiere modificar el repertorio de instrucciones del MIPS añadiendo dos nuevas instrucciones:
- RSB Rd,Rs,Rt donde BR(Rd)←BR(Rt)-BR(Rs)
- LWRDR Rd, Rs,Rt,#n donde BR(Rd)←M[BR(Rs)+BR(Rt)x2^n]

Para ambas instrucciones el formato que se utiliza es el tipo R, para la instrucción LWRDR el valor n se obtiene de campo shamt que hasta el momento no se utilizaba. Se pide:
a) (1.25 puntos) Camino de datos modificado
b) (1.25 puntos) Diagrama de estados modificado
c) (1 punto) Tablas de verdad modificadas

3) **(3.5 puntos)** Tenemos una máquina con una memoria cache de 1KB con bloques de 128B y política de emplazamiento directo. Además, sabemos:
   - El campo etiqueta de la dirección de cache ocupa 13 bits
   - El tiempo de acceso a la cache es de 1ns
   - El tiempo de acceso a memoria principal es de 10ns
   - El tiempo de penalización por fallo es de 100ns

   a) (0.75 puntos) Indicar el tamaño de la memoria principal en megabytes.
   b) (0.75 puntos) Calcular el número de bits necesarios para implementar la memoria cache (incluyendo el directorio).
   c) (1.25 puntos) Asumiendo que se accede consecutivamente a todas las direcciones de memoria comprendidas entre las direcciones 0x059604 y 0x059A08 (ambas inclusive) y teniendo en cuenta que el contenido inicial del directorio es el que se muestra a continuación. Indicar el número de fallos y aciertos que se producen, así como el contenido final del directorio.

	Validez	Etiqueta
0	1	0x0166
1	0	
2	1	0x1740
3	0	
4	1	0x0165
5	1	0x0175
6	0	
7	0	

   d) (0.75 puntos) Calcular la ganancia de velocidad al realizar los accesos de memoria del apartado c) utilizando la jerarquía de memoria presentada frente a los mismos accesos en un computador sin cache.

4) **(1 punto)** Se ejecuta el código ARM mostrado a continuación en un computador cuyas instrucciones requieren los siguientes ciclos:
   - Operaciones aritmético-lógicas, almacenamiento (store) y comparaciones: 4 ciclos.
   - Carga: 5 ciclos.
   - Saltos: tomados 4 ciclos, no tomados 3 ciclos.

```
     ldr r1, [r0]
     ldr r2, [r0, r4]
     add r2, r1, r2
     cmp r2, r3
     beq fin
     sub r3, r3, r5
fin: str r2, [r0, r4]
```

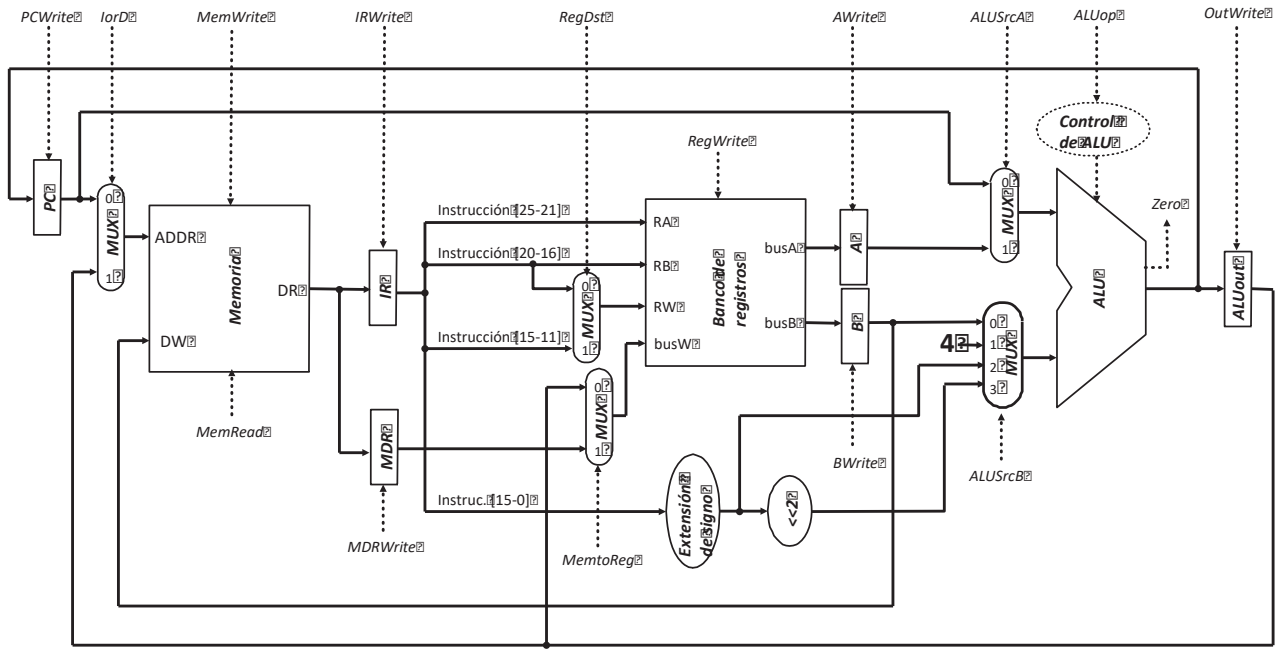
   Sabiendo que la frecuencia de reloj es de 1.2 GHz y que el estado inicial es el mostrado a continuación, calcular el CPI y el tiempo de ejecución del código.

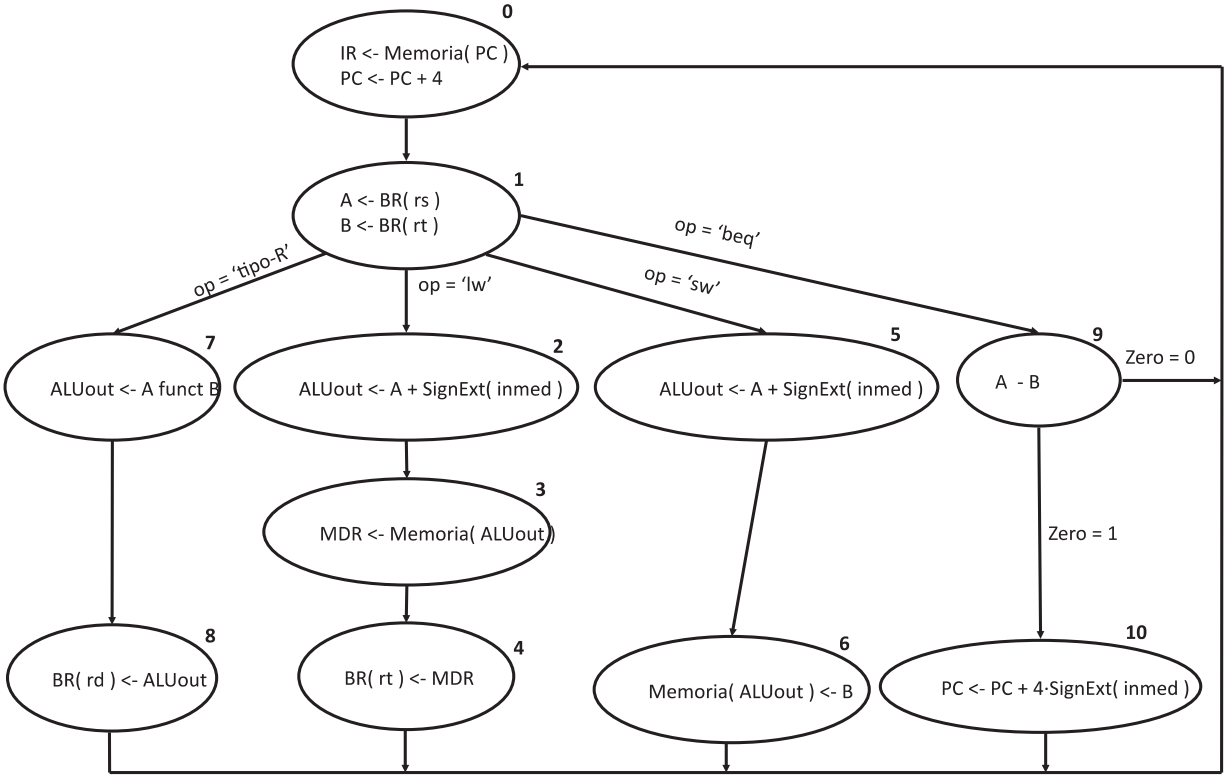
Banco de registros

R0	0x0000C000
R1	0x00000006
R2	0x00000001
R3	0x0000000D
R4	0x00000010
R5	0x00000002

Memoria

0x0000C000	0x00000006
0x0000C004	0x0FFFF004
0x0000C008	0x00000001
0x0000C00C	0x0004200D
0x0000C010	0x00000003
0x0000C014	0x0C000002


PCWrite
IorD
MemWrite
IRWrite
RegDst
RegWrite
AWrite
ALUSrcA
ALUop
OutWrite
PC
MUX
ADDR
Memoria
DR
DW
MemRead
IR
MDR
MDRWrite
Instrucción[25-21]
Instrucción[20-16]
Instrucción[15-11]
Instruc.[15-0]
RA
RB
RW
busW
Banco de registros
busA
busB
A
B
Control de ALU
Zero
ALU
ALUout
4
BWrite
ALUSrcB
MemtoReg
Extensión de signo
<<2


0
IR <- Memoria(PC)
PC <- PC + 4
1
A <- BR(rs)
B <- BR(rt)
op = 'tipo-R'
op = 'lw'
op = 'sw'
op = 'beq'
7
ALUout <- A funct B
2
ALUout <- A + SignExt(inmed)
5
ALUout <- A + SignExt(inmed)
9
A - B
Zero = 0
Zero = 1
3
MDR <- Memoria(ALUout)
8
BR(rd) <- ALUout
4
BR(rt) <- MDR
6
Memoria(ALUout) <- B
10
PC <- PC + 4·SignExt(inmed)

Estado actual	op	Zero	Estado siguiente	IRWrite	PCWrite	AWrite	BWrite	ALUSrcA	ALUScrB	ALUOp	OutWrite	MemWrite	MemRead	IorD	MDRWrite	MemtoReg	RegDest	RegWrite
0000	**XXXXXX**	X	0001	1	1			0	01	00 (add)		0	1	0				0
0001	100011 (lw)	X	0010	0	0	1	1					0	0					0
0001	101011 (sw)	X	0101															
0001	000000 (tipo-R)	X	0111															
0001	000100 (beq)	X	1001															
0010	XXXXXX	X	0011	0	0			1	10	00 (add)	1	0	0					0
0011	XXXXXX	X	0100	0	0							0	1	1	1			0
0100	XXXXXX	X	0000	0	0							0	0			1	0	1
0101	XXXXXX	X	0110	0	0		0	1	10	00 (add)	1	0	0					0
0110	XXXXXX	X	0000	0	0							1	0	1				0
0111	XXXXXX	X	1000	0	0			1	00	10 (funct)	1	0	0					0
1000	XXXXXX	X	0000	0	0							0	0			0	1	1
1001	XXXXXX	0	0000	0	0			1	00	01 (sub)		0	0					0
1001	XXXXXX	1	1010															
1010	XXXXXX	X	0000	0	1			0	11	00 (add)		0	0					0