	**Fundamentos de Computadores** **12 de Septiembre de 2016. Examen Parcial 2º Cuatrimestre**
	Nombre:_____ DNI: _____ Apellidos: _____ Grupo: _____

Ejercicio 1 (3 puntos) Un procesador cuenta con una jerarquía de memoria que está compuesta por una memoria principal de 16MB y una cache con emplazamiento de 256B con bloques de 32B. En dicho procesador se ejecuta un código que genera accesos a memoria consecutivos a todas las direcciones entre la 0x000108 y la 0x000270.

a) (0.25 puntos) Muestra el formato de la dirección para la Mp y la cache

b) (0.75 puntos) Indicar razonadamente los bloques de memoria accedidos por el código, señalando para cada uno el bloque o marco de cache y la etiqueta que le corresponderían.

c) (0.75 puntos) Suponiendo que el contenido inicial del directorio es el mostrado en la tabla siguiente, indicar los fallos de cache que provocará la ejecución del código y el contenido final que tendrá el directorio.

V	Etiqueta
0	
0	
0	
0	
1	0x0001
1	0x0001
1	0x0001
1	0x0001

Supongamos que a continuación vuelve a ejecutarse el mismo código una segunda vez:

d) (0.5 puntos) Indica razonadamente cuántos fallos de cache se producirían la segunda vez

e) (0.75 puntos) Si el tiempo de acceso a la memoria principal es de 30 ns, el tiempo de acceso a la cache es de 1 ns y la penalización por fallo es de 300 ns, indica cuál sería el tiempo medio de acceso a memoria y la ganancia que se obtiene respecto a un sistema sin cache (teniendo en cuenta las dos ejecuciones del código).

Ejercicio 2 (1 punto) En un determinado computador con procesador ARM a una frecuencia de reloj de 1GHz se mide el tiempo que tarda en ejecutarse el siguiente fragmento de un programa:

```
        mov r4, #0
        ldr r5, =A
        mov r6, #0
        mov r4, #127
L1:     ldr r0, [r5, r4, lsl #2]
        add r6, r6, r0
        sub r4, r4, #1
        cmp r4, #0
        bge L1
L2:
        ... (resto del programa)
```

a) (0.5 puntos) Sabiendo que dicho fragmento de código ha tardado 2898 ns en ejecutarse, obtenga el CPI para este código.
b) (0.5 puntos) ¿Cuántos MIPS podríamos decir que tiene el procesador a partir de esta prueba?

Ejercicio 3 (3 puntos) Dato un vector **A** de **N** enteros de 32bit sin signo, se desea calcular su Código de Redundancia Cíclico o **CRC** siguiendo el algoritmo de Fletcher. Dicho algoritmo genera como salida dos enteros sin signo que servirán para comprobar la integridad de los datos. El algoritmo de Fletcher de 64bits se puede implementar a partir del siguiente código:

```
void Fletcher64( unsigned int data[], int count, unsigned int crc[] ) {
    unsigned int sum1 = 0;
    unsigned int sum2 = 0;
    int index;

    for ( index = 0; index < count; ++index ) {
        sum1 = sum_mod64(sum1, data[index]);
        sum2 = sum_mod64(sum1 , sum2);
    }
    crc[0] = sum1;
    crc[1] = sum2;
}
```

donde la rutina unsigned int sum_mod64(unsigned int A, unsigned int B) está ya implementada.

a) (1.75 puntos) Escribir el código ensamblador de la rutina Fletcher64
b) (1.25 puntos) Escriba un programa en lenguaje ensamblador del ARM que, invocando a la rutina Fletcher64, calcule el CRC de los siguientes vectores:
    a. V={0x12340000, 0x00005678}
    b. W={0xAB000000, 0x00CD0000, 0x0000EF00, 0x00000011}

Ejercicio 4. (3 puntos) En su última revisión del repertorio de instrucciones, MIPS incluyó algunas instrucciones nuevas que realizan cálculos relativos al PC. Una de ellas es una suma relativa al PC, cuyo nemotécnico es:

```
ADDIUPC Rs, Immediate
```

codificada con el siguiente formato:

6	5	21
111011	Rs	Immediate

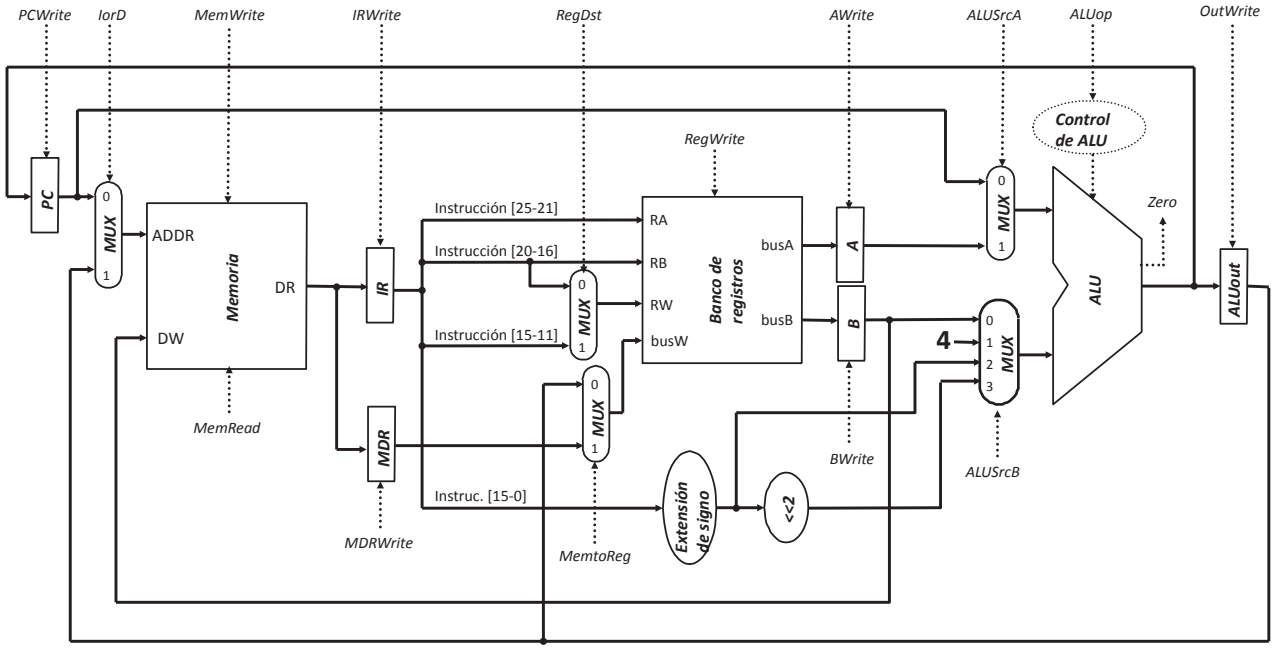
y cuya descripción es la siguiente:

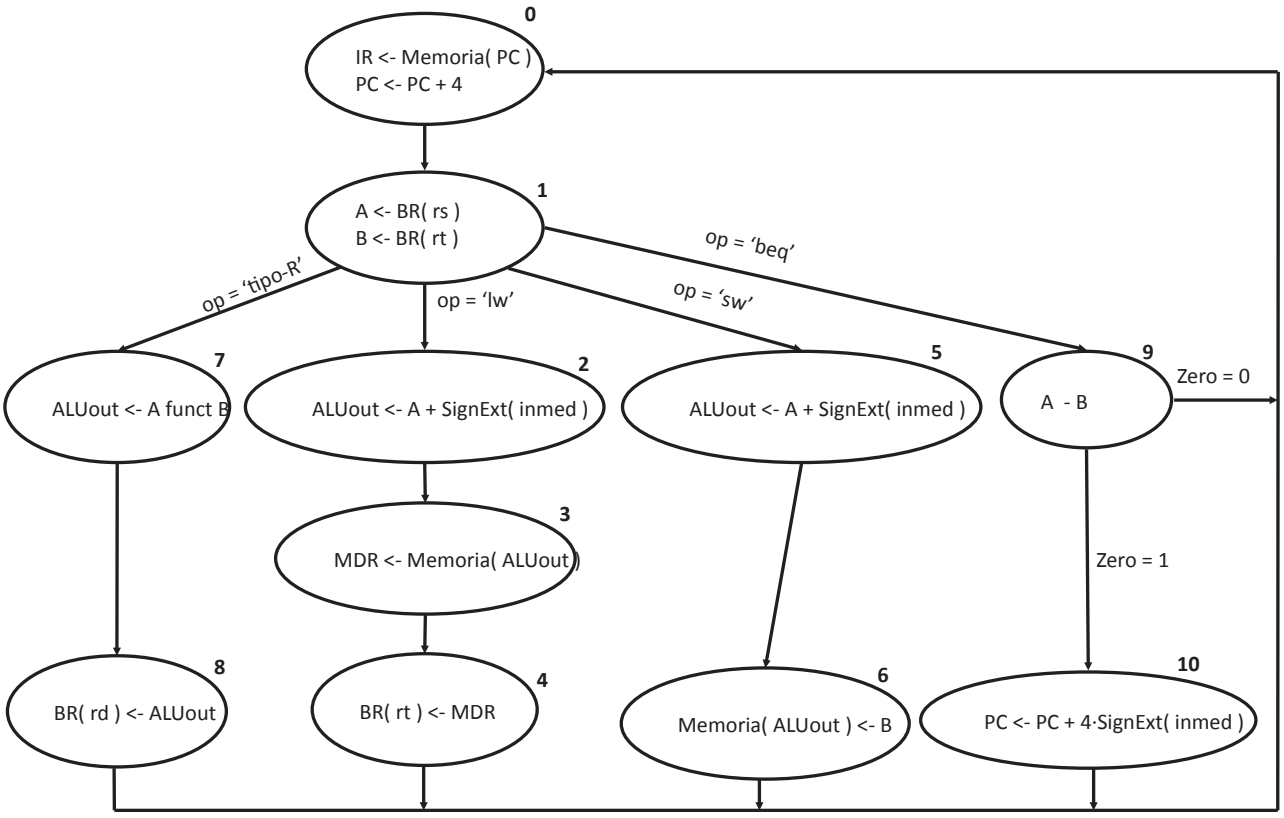
```
BR(Rs) <- (PC+4) + (SignExt(Immediate))   ,   PC <- PC + 4
```

es decir, el contenido del campo Immediate de la instrucción se extiende en signo hasta los 32 bits, se suma con el PC de la instrucción incrementado en 4 unidades, y se almacena en Rs.

Queremos incorporar esta instrucción al procesador estudiado en clase, se pide:
a) (1 punto) Indicar los cambios necesarios en la ruta de datos.
b) (1 punto) Indicar los cambios necesarios en el diagrama de transición de estados del controlador.
c) (1 punto) Indicar los cambios necesarios en la tabla de verdad del controlador.

Nota: Fíjate que el formato empleado para esta instrucción no corresponde a ninguno de los vistos en clase (Tipo-R, Tipo-I). Ello no debe suponer ninguna dificultad a la hora de realizar el ejercicio: simplemente debes incluir el HW necesario para cumplir la descripción explicada arriba de acuerdo a la organización de los campos de la instrucción en el nuevo formato.


PCWrite
IorD
MemWrite
IRWrite
RegDst
AWrite
ALUSrcA
ALUop
OutWrite
RegWrite
Control de ALU
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
BWrite
Extensión de signo
<<2
MemtoReg
4
ALU
Zero
ALUout
ALUSrcB


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