	Fundamentos de Computadores **16 de Junio 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 64MB y una cache con emplazamiento directo de 4KB con 16 bloques. En dicho procesador se ejecuta un código que genera accesos de lectura a memoria consecutivos a todas las direcciones entre 0x0300800 y 0x03017FF.

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

b) (0.75 puntos) Indica los fallos de cache que provocará la ejecución del código indicando los bloques sobre los que se produce el fallo, asumiendo que la cache está inicialmente vacía

c) (0.75 puntos) Indica cuál será el contenido del directorio (array de etiquetas o *tags*) tras finalizar la ejecución

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

d) (0.5 puntos) ¿Cuántos fallos de cache se producirían la segunda vez?

e) (0.75 puntos) Si el tiempo de acceso a la memoria principal es de 20 ns, el tiempo de acceso a la cache es de 1 ns y la penalización por fallo es de 400 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 1.5GHz se ejecuta el siguiente programa:

```
       mov r4, #0
       ldr r5, =A
       mov r6, #0
L1:    cmp r4, #128
       bge L2
       ldr r0, [r5, r4, lsl #2]
       add r6, r6, r0
       add r4, r4, #1
       b L1
L2:
       b .
```

a) Suponiendo que las instrucciones aritmético lógicas tardan 4 ciclos, las instrucciones de carga tardan 5 ciclos, los saltos tomados 4 ciclos y los no tomados 3 ciclos, obtener el número medio de ciclos por instrucción para este programa (CPI) (hasta la primera vez que se ejecuta la instrucción `b .`).

b) En el apartado anterior se suponía un comportamiento ideal en el que todos los accesos a memoria se realizan en 1 ciclo de reloj, en cuyo caso las instrucciones de carga tardan los 5 ciclos de reloj indicados. Supongamos ahora que el sistema tiene una memoria cache en la que se produce un fallo por cada 16 ejecuciones de la instrucción ldr r0, [r5, r4, lsl #2] y que la penalización por fallo es de 100 ciclos. Teniendo en cuenta el tiempo adicional debido a las penalizaciones, calcular de nuevo el CPI.

Ejercicio 3 (3 puntos) Dado un vector, A, de 3*N componentes, se desea obtener un nuevo vector, B, de N componentes donde cada componente de B es la suma módulo 32 de una tripleta de elementos consecutivos de A. Es decir:

B[0] = (A[0]+A[1]+A[2]) mod 32, B[1] = (A[3]+A[4]+A[5]) mod 32, etc

Se pide:

a) (1.75 puntos) Escribir un programa en lenguaje ensamblador del ARM que implemente el cálculo descrito de acuerdo con el siguiente código C equivalente:

```
#define N 4
int A[3*N] = {una lista de 3*N valores},
int B[N];
int i, j=0;
void main (void)
{
       for (i=0; i<N; i++){
              B[i] = sum_mod_32(A, j, 3);
              j=j+3
       }
}
```

donde la subrutina sum_mod_32(V,p,m) devuelve como resultado la suma módulo 32 de m elementos consecutivos del vector V tomados a partir de la posición p.

b) (1.25 puntos) Escribir el código ensamblador de la subrutina sum_mod_32, de acuerdo al siguiente código C equivalente:

```
sum_mod_32 (int A[ ], int j, int len)
{
       int i, sum=0;
       for (i=0; i<len; i++)
              sum = sum + A[j+i];
       sum=mod_power_of_2(sum,5);
       return sum;
}
```

donde la subrutina mod_power_of_2(num, exp), siendo num un entero positivo y exp un entero mayor que 0 y menor que 32, devuelve como resultado el valor de (num mod 2^exp). Por ejemplo, si invocamos a la función como: mod_power_of_2(34, 5), la función devolvería como salida: ((34) mod (2^5))=(34 mod 32)=2.

Nota.- En todos los apartados se debe respetar el estándar de llamadas a subrutinas estudiado en clase, y las variables pueden ubicarse en registros o en memoria (global o pila según corresponda).

Ejercicio 4. (3 puntos) Se desea añadir al procesador MIPS estudiado en clase la instrucción *Load Upper Immediate*, cuyo nemotécnico sería:

```
LUI Rx, Immediate
```

codificada como una instrucción Tipo-I:

6	5	5	16
110011	not used	Rx	Immediate

y cuya descripción se correspondería con el siguiente movimiento registro a registro:

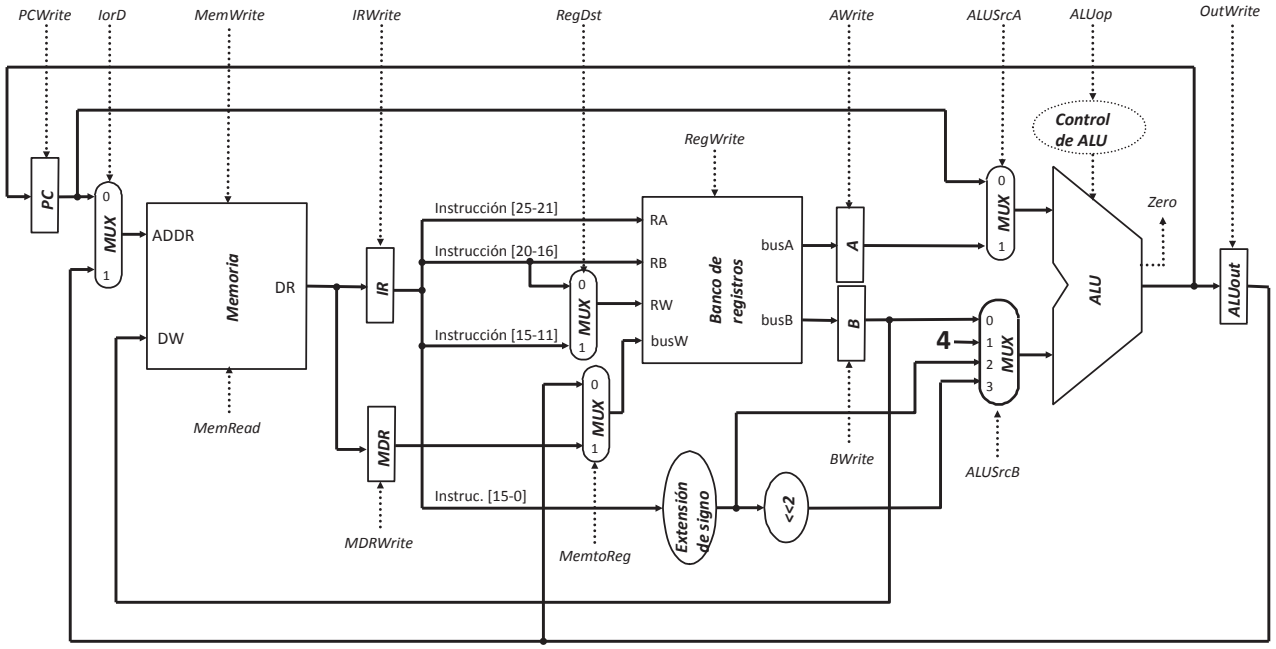
```
BR(Rx) <- (Immediate << 16)   ,   PC <- PC + 4
```

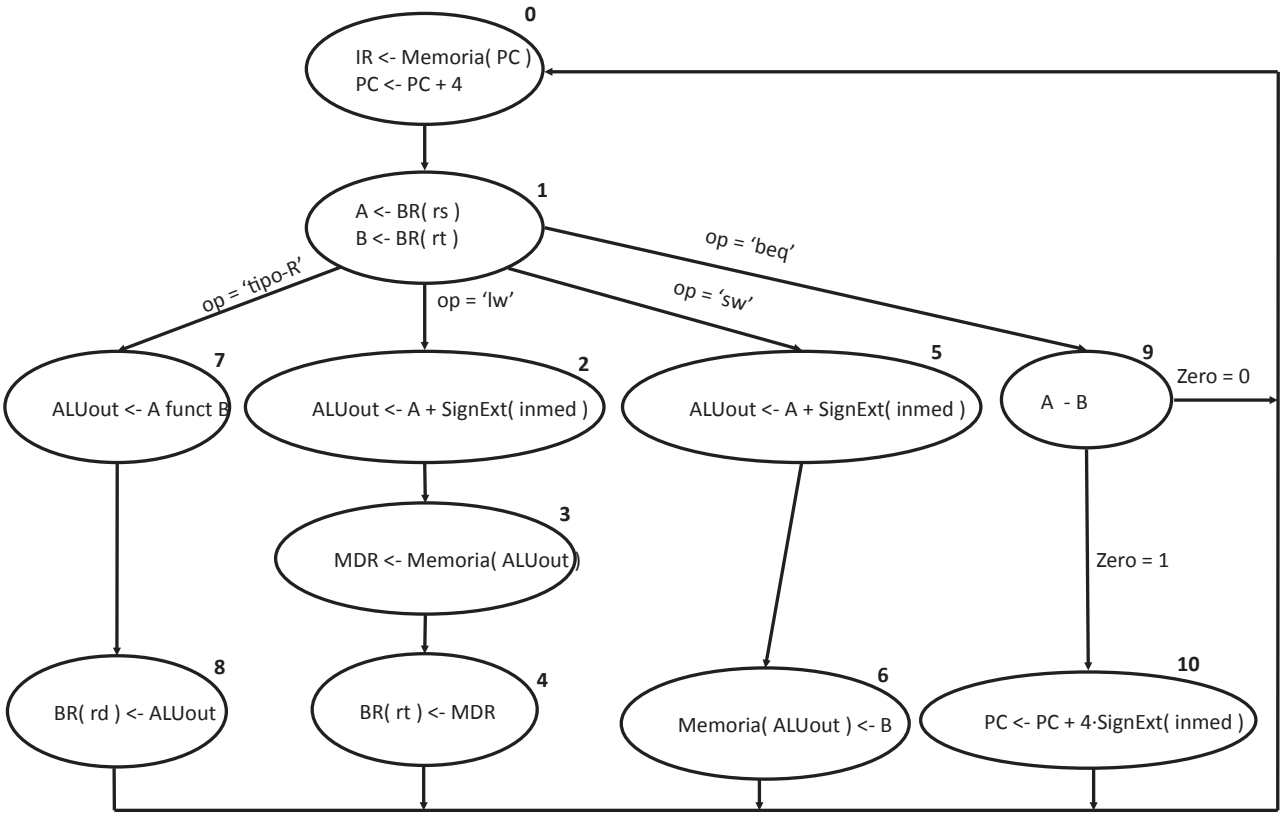
es decir, el contenido del campo Immediate de la instrucción se desplaza 16 posiciones a la izquierda, rellenando con ceros por la derecha, y se almacena en el registro destino.

a) (1 punto) Indicar los cambios necesarios en la ruta de datos del MIPS estudiada en clase.

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.


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