	Fundamentos de Computadores 2ºC. 27 de mayo de 2019. Examen 2º Parcial (todos los apartados, 10 puntos). **Examen Final (solo apartados en negrita, 5.5 puntos).**
	Nombre:_____ DNI: _____ Apellidos: _____FINAL/PARCIAL:_____

Ejercicio 1. Dado un vector A de MxN enteros, escriba un programa en ensamblador del ARM que ordene A de tal manera que cada fragmento M de N enteros esté ordenado de manera creciente:

(A[0] ≤ A[1] ≤ … ≤ [N-1]); (A[N] ≤ A[N+1] ≤ … ≤ A[2N-1]); etc.

Para realizar la tarea:

a) (1.25 pts) Escriba un programa en ensamblador que implemente la solución en pseudocódigo de alto nivel mostrada en el cuadro de la izquierda. Para realizar la tarea, este código invoca a la subrutina `part_sort(int V[], int pos, int len)` encargada de ordenar de forma creciente un fragmento de `len` elementos a partir de `V[len]`.

b) (1.75 pts) Escriba el código ensamblador correspondiente a la subrutina `part_sort` siguiendo el pseudocódigo mostrado en la parte derecha del siguiente cuadro. Nótese que el primer argumento de la subrutina es la dirección de comienzo del vector V.

```
#define N,5
#define M,4

int A[M*N]={a list of M*N integers};
int i;

int main(){
  for (i = 0; i<M; i++)
    part_sort(A,i*N,N);
}
```

```
void part_sort(int V[], int pos, int len){
  int i,j,temp1,temp2;
  for (i=pos; i<pos+len-1; i++){
    for (j=i+1; j<pos+len; j++){
      if (V[i]>V[j]){
        temp1=V[i];
        temp2=V[j];
        V[i]=temp2;
        V[j]=temp1;
      }
    }
  }
  return;
}
```

Ejercicio 2. Un programa tarda en ejecutarse 4.5ns en un computador que trabaja a una frecuencia de 1.2GHz obteniéndose además los siguientes resultados en cuanto a instrucciones ejecutadas:

	Aritméticas	**Load**	**Store**	**Salto**
Porcentaje	55	20	10	15
Ciclos	5	6	5	3.7

a) (0.5 pts) Calcule el CPI del programa, indique cuántas instrucciones ejecuta el procesador y exprese el rendimiento en MIPS.

b) (0.5 pts) Sabiendo que el número de ciclos empleados por el procesador para los saltos tomados es de 4 y en caso no tomarlos emplea 3, ¿qué porcentaje del total de instrucciones corresponde a cada tipo de salto?

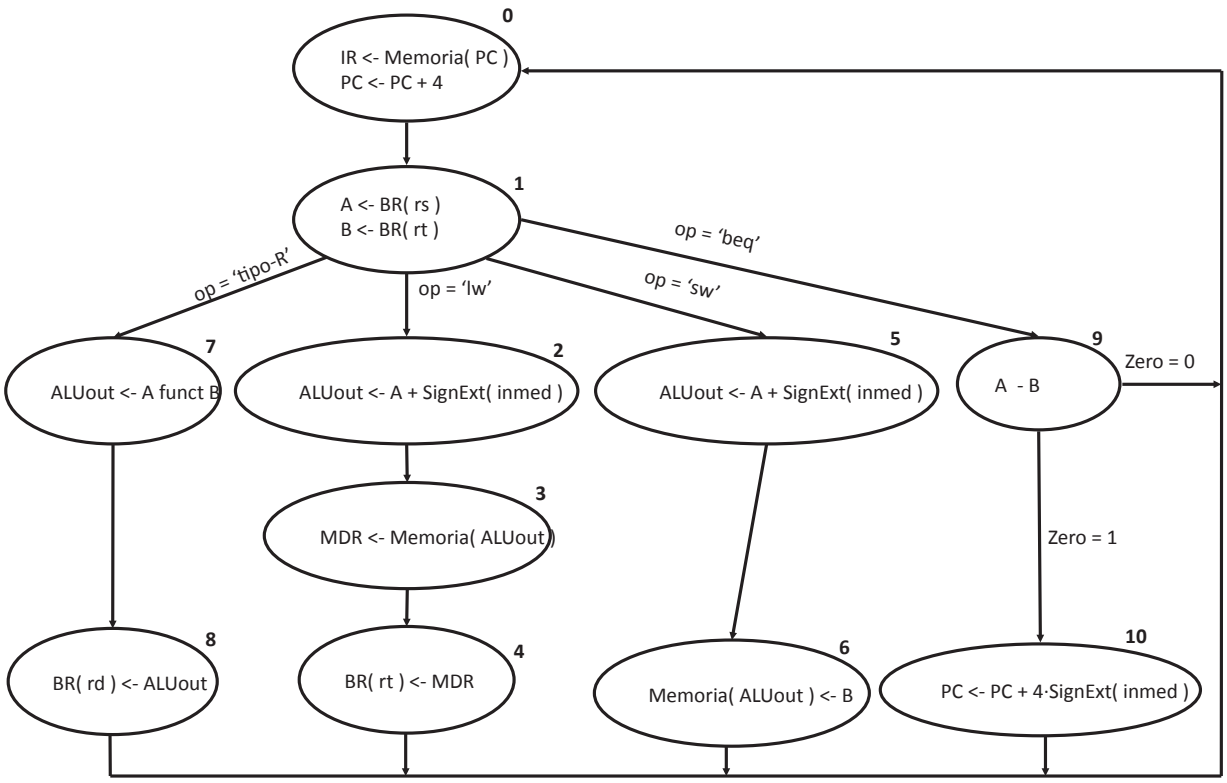
Ejercicio 3. Interesa incluir una nueva funcionalidad en el procesador MIPS visto en clase. Concretamente, se desea especificar qué debe hacer el procesador si se lee una instrucción no válida (esto es, que el campo *OP* de la instrucción que esté IR no se corresponde con ninguno de los vistos en clase). Si esto ocurre se debe guardar el PC en el registro 31 y saltar a la dirección 0x00000004. Indica y justifica:

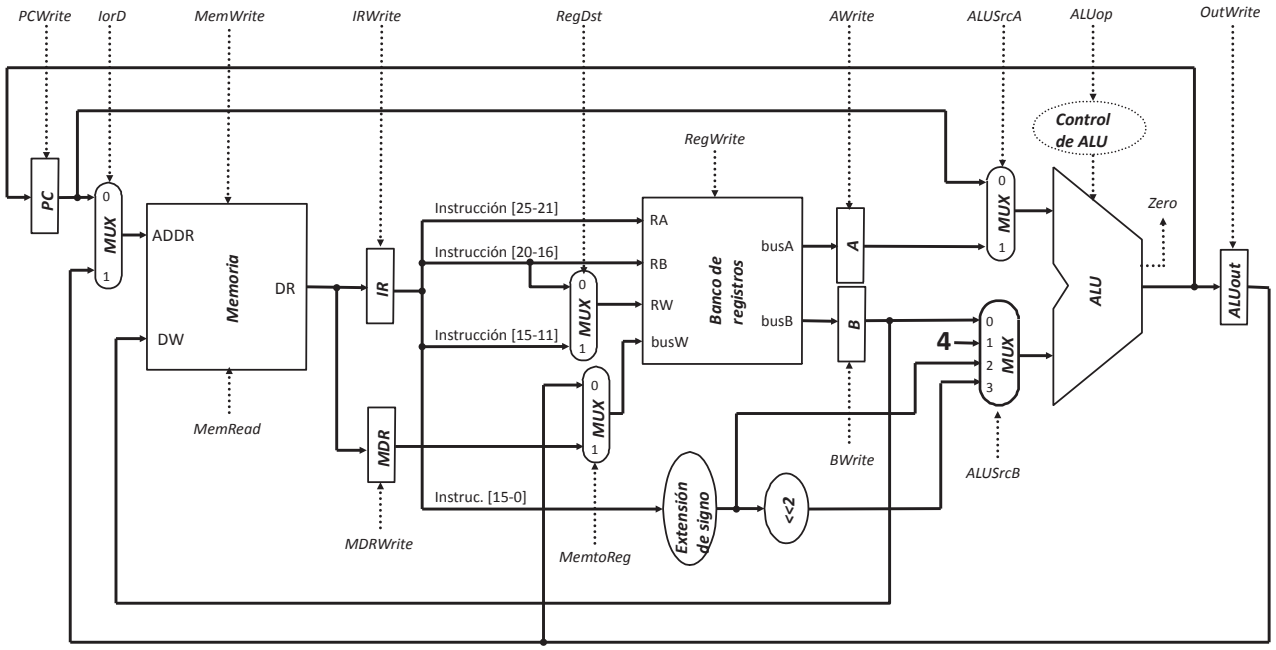
a) (1 pt) Modificar la ruta de datos para implementar esta nueva funcionalidad.

b) (1 pt) Modificar la máquina de estados del controlador para contemplar esta posibilidad.

c) (1 pt) Completa la tabla de verdad del controlador. Ya se incluye la transición del estado '0001' al estado '1011' en caso de instrucción no válida. Sólo debes incluir una fila por cada estado que hayas añadido en el apartado *b)* y una columna por cada señal de control que hayas creado en el apartado *a)*.

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															
0001	101011 (sw)	X	0101	0	0	1	1					0	0					0
0001	000000 (tipo-R)	X	0111															
0001	000100 (beq)	X	1001															
0001	**NO VÁLIDO**	**X**	**1011**															
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


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)


PCWrite
IorD
MemWrite
IRWrite
RegDst
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]
RegWrite
RA
RB
RW
busW
Banco de registros
busA
busB
A
B
BWrite
Control de ALU
Zero
ALU
ALUout
ALUSrcB
4
Extensión de signo
<<2
MemtoReg

Ejercicio 4. A continuación, se muestra la estructura de un programa que dado un vector *A* de 1024 componentes obtiene un vector *B* con el doble de la suma de cada una de las componentes del vector *A*.

```
        .global start
        .data
A:      .word 0x00000006, 0x00000008, 0x00000005, 0xFFFFFFFF …
        .bss
B:      .space  …               @ a contestar en apartado a)
        .text
start:  …
        …
        …
FIN:    b.
        .end
```

Se pide:

a. Suponiendo que dicho programa se carga en la dirección de memoria 0X00001000 indica:

1. **(1 pt) La dirección que corresponde a cada uno de los vectores y al comienzo del programa (e.d., el valor de las etiquetas *A*, *B* y *start*).**
2. (0.25 pts) El número de palabras que se deben reservar para el vector *B.*

b. Se ejecutan dos versiones (v1 y v2) de dicho programa en un computador en el que las instrucciones aritméticas, de almacenamiento, y las de salto necesitan 4 ciclos para su ejecución, las de carga necesitan 5, y las de salto condicional si el salto no se produce necesitan 3. Entendiendo la ejecución hasta la etiqueta fin, dónde hay un punto de ruptura, cal:

1. (0.75 pts) Calcula el CPI para cada uno de los programas.
2. **(0.75 pts) Calcula el tiempo de ejecución de ambos códigos para un reloj de 1 Ghercios.**
3. (0.25 pts) Calcula la mejora del tiempo de ejecución (eficiencia) del código B frente al A.

```
      @ CÓDIGO v1

       mov r0,#1          @ i=0
       lsl r1,r0,#10      @ r1=1024
       ldr r2,=A
       ldr r3,=B
Bucle: cmp r0,r1
       bge fin
       ldr r4,[r2,r0,lsl #2] @A[i]
       lsl r4,r4,#2
       str r4,[r3,r0,lsl #2] @B[i]
       add r0,r0,r1
       b Bucle
fin:   b.
```

```
      @ CÓDIGO v2

       mov r0,#1          @ i=0
       lsl r0,r0,#10      @ r1=1024
       sub r0,r0,#1
       ldr r2,=A
       ldr r3,=B
Bucle: ldr r4,[r2,r0,lsl #2] @A[i]
       lsl r4,r4,#2
       str r4,[r3,r0,lsl #2] @B[i]
       subs r0,r0,r1
       bge Bucle
fin:   b.
```