	**Fundamentos de Computadores** **4 de Septiembre de 2015. Examen Parcial 2º Cuatrimestre**
	Nombre:_____ DNI: _____ Apellidos: _____ Grupo: _____

Ejercicio 1 (3 puntos) Dados dos puntos P1(x1,y1) y P2(x2,y2), la distancia de Chebishev entre ellos puede calcularse con el siguiente algoritmo:

```
int chebishev(x1, y1, x2, y2)
{
  int d1, d2;
  d1 = abs(x1-x2)
  d2 = abs(y1-y2)
  if (d2 > d1)
      d1 = d2;
  return d1;
}
```

a) (1.5 puntos) Codificar en ensamblador la subrutina chebishev(x1, x2, y1, y2), que recibe cuatro parámetros enteros con signo que se corresponden con las coordenadas *(x1, x2)* e *(y1, y2)* de dos puntos P1 y P2 y devuelva la distancia de Chebyshev que separa P1 y P2. En su cuerpo, invocará a la subrutina abs (valor absoluto). Téngase en cuenta por tanto que la subrutina no es una subrutina hoja.

b) (1.5 puntos) Codificar en ensamblador un programa que almacene en un vector D la distancia de Chebishev desde un punto P a todos los puntos de un vector V de N puntos, dónde P, V y D serían variables globales. El vector V tendrá 2N enteros de forma que el i-ésimo punto tendrá coordenadas (x, y) = (V[2*i], V[2*i + 1]). Un código C equivalente sería:

```
#define N, ...

int Px, Py; //coordenadas x e y del punto P
int V[2N];  //Vector con N puntos V=[x0,y0,x1,y1,...]
int D[N];   //Vector de N distancias

void main(void)
{
      int i;
      for (i=0; i < N; i++)
            D[i] = chevishev(Px, Py, V[2*i], V[2*i + 1]);
}
```

* Nota: Se debe respetar el convenio visto en clase para llamadas a subrutinas

Ejercicio 2 (3 puntos) Sea un computador con una memoria principal de 1 MByte, una memoria cache con emplazamiento directo y con capacidad para albergar 16 bloques de 256 Bytes cada uno. Se pide:

a) (0.25 puntos) Indicar el formato de dirección para MP y MC si la memoria se direcciona tamaño byte.

b) (0.5 puntos) En un momento dado las etiquetas de algunos marcos de la cache son las indicadas en la tabla. Expresar en hexadecimal el rango de direcciones de memoria principal ubicadas en estos bloques de memoria cache.

Bloque MC (Hex.)	Etiqueta (Hex.)	Rango direcciones MP(Hex.)
6	33	-
A	77	-

c) (1.25 puntos) Partiendo del estado de cache del apartado anterior (el resto de marcos están vacíos), se ejecuta un programa con las siguientes referencias a memoria de tamaño byte. Calcular el número de fallos de cache que se producen:

```
Comentario              Dirección (hex)
Una vez                 33008
                        ...
                        347F7
Bucle 2 veces           347F8
                        ...
                        35FF7
```

d) (1 punto) La memoria principal tiene un tiempo de acceso de 10 ns, la memoria cache 1 ns y una penalización por un fallo de cache 500 ns. Calcula el tiempo de acceso a las referencias del programa anterior con y sin cache. Indica cuantas veces más rápido es el sistema con la cache.

Ejercicio 3 (1 punto) En un computador basado en un procesador de ARM cuya frecuencia es de 1GHz se ejecuta el siguiente código:

```
.data:
A:    .word M_valores_separados_por_comas
sum: .word 0
.text
start: ldr r0, =A
       mov r5, #0
       mov r4, #0
       mov r6, #M
L1:    cmp r4, r6
       bge L2
       ldr r1,[r0, r4 lsl #2]
       add r5, r5, r1
       add r4, r4, #1
       b L1
L2:    ldr r0, =sum
       str r5, [r0]
      .end
```

a) (0.5 puntos) Se ha comprobado que al ejecutar este código para un determinado valor de M se obtiene un CPI de 4.025. ¿Cuál ha sido el rendimiento del procesador medido en MIPS?

b) (0.5 puntos) Asumiendo que las instrucciones aritmético-lógicas tardan 4 ciclos, las instrucciones de load tardan 5 ciclos, las instrucciones de store tardan 4 ciclos, los saltos tomados 4 ciclos y los saltos no tomados 3 ciclos, obtener el valor de M.

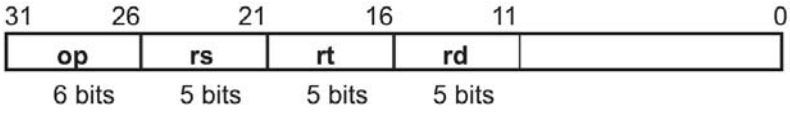
Ejercicio 4 (3 puntos) Se desea añadir al procesador MIPS visto en clase las siguientes instrucciones:

- MULT Rs, Rt: HI <- $(Rs*Rt)_{63..32}$ y LO <- $(Rs*Rt)_{31..0}$ y PC <- PC + 4
- MFHI Rd: Rd <- HI y PC <- PC + 4
- MFLO Rd: Rd <- LO y PC <- PC + 4

La primera de ellas multiplica los enteros de 32 bits almacenados en Rs y Rt generando un entero de 64 bits. Los 32 bits más significativos los almacena en un nuevo registro HI, mientras que los 32 bits menos significativos los almacena en otro nuevo registro LO. Las otras dos instrucciones permiten copiar el contenido de los nuevos registros HI y LO en algún registro del banco de registros del MIPS. Así, por ejemplo, la secuencia

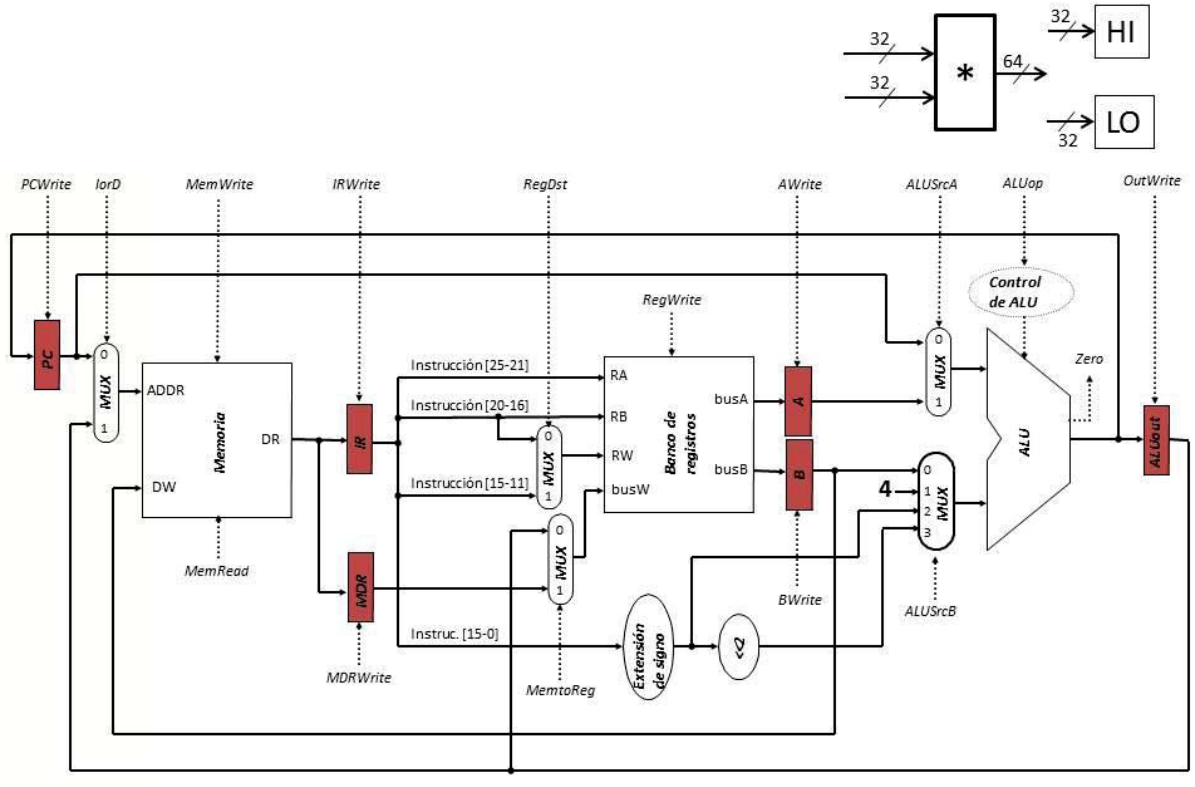
MULT r1,r2
MFHI r1
MFLO r2

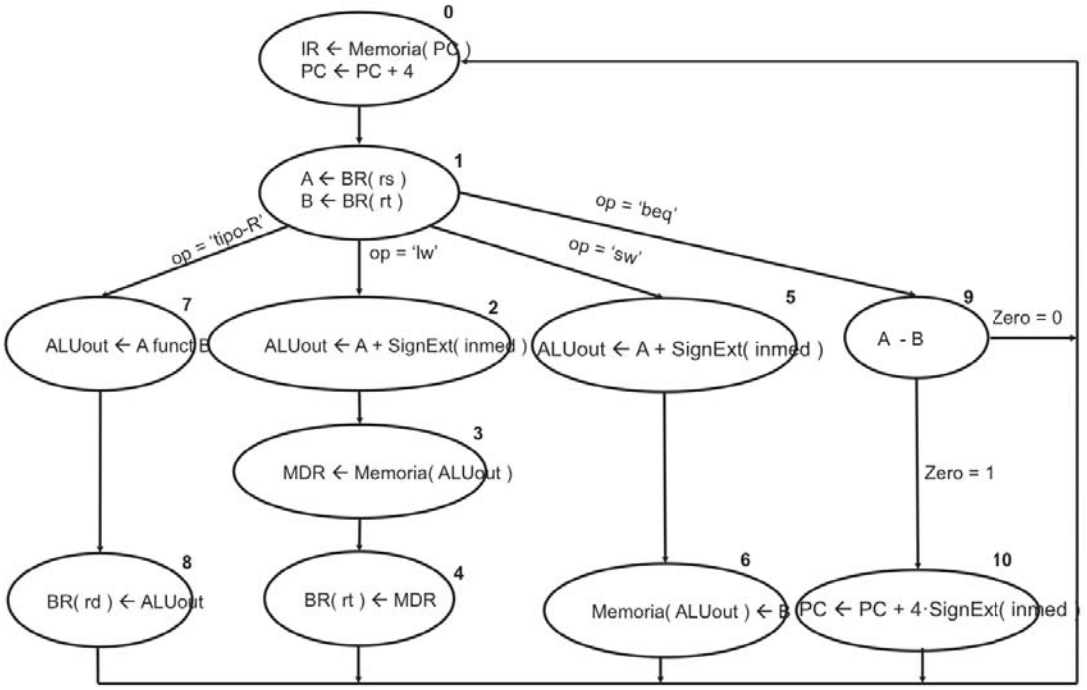
almacena en el registro r1 la parte más significativa del producto r1*r2 y en el registro r2 la parte menos significativa. El formato de las tres instrucciones sería el siguiente:



con los códigos de operación (OP) 001010, 001011 y 001100 respectivamente. Se pide:

a) (1 punto) Modifica la ruta de datos para que puedan ejecutarse estas 3 instrucciones. Utilizando los dos nuevos registros HI y LO, un multiplicador que reciba 2 números de 32 bits y devuelva 1 número de 64 bits, y todos los multiplexores y señales de control que necesites.
b) (1 punto) Indica las modificaciones necesarias en la máquina de estados del controlador para que puedan ejecutarse estas instrucciones.
c) (1 punto) Indica los cambios que serían necesarios en las tablas de verdad del controlador.





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