Practica de diseño de un procesador monociclo

El objetivo de esta práctica de la asignatura de Tecnología de Computadores es la implementación de un procesador MIPS reducido (nombre del procesador **TG00MIPS**).

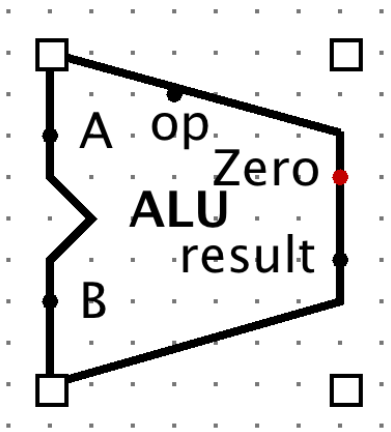
Las características hardware principales del procesador TG00MIPS son:

- Palabra de 32 bits.
- Banco de 32 registros de propósito general de 32 bits. **El registro $0 será de solo lectura y contendrá el valor 0**.
- Unidad aritmético-lógica (ALU) de 32 bits con las siguientes características:
  - Suma y resta de operandos enteros representados en convenio complemento a 2 (C2)
  - Operación AND lógica, OR lógica y NOR lógica.
  - Operación de comparación “menor que”.
  - Un flag de estado: Z indica que el resultado de la ALU es igual a 0).
- Memoria de instrucciones:
  - Bus de dirección y de datos de 32 bits
  - Direccionable a nivel de byte.
- Memoria de datos:
  - Bus de dirección y de datos de 32 bits
  - Direccionable a nivel de byte.

Para el diseño del procesador se proporcionan ya diseñados los siguientes elementos: una ALU, un banco de 32 registros de 32 bits y las memorias de instrucciones y de datos.

En las siguientes figuras y tablas se muestra la forma, interfaz y funcionamiento de los elementos proporcionados.

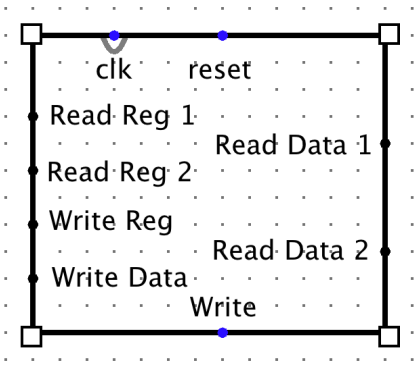
ALU (ALU)



Señales	
A[31:0]	Dato de entrada
B[31:0]	Dato de entrada
op[3:0]	Operación a realizar por la ALU
result[31:0]	Dato de salida
Zero	Se activa (1) cuando el resultado es cero

op[3:0]	Operación ALU	
0000	result = A & B	And (bit a bit)
0001	result = A | B	Or (bit a bit)
0010	result = A + B	Suma
0110	result = A – B	Resta
0111	If (A<B) result = 1 else result = 0	Activar si A menor que B
1100	result = $\overline{A \mid B}$	Nor (bit a bit)

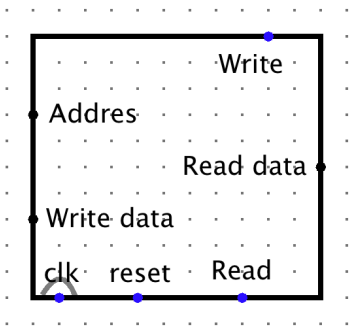
BANCO DE REGISTROS (Register file)



Señales	
clk	Señal de reloj para sincronizar la escritura
Write	Señal de escritura (1)
Read Data 1[31:0]	Dato leído del registro indicado en Read Reg 1
Read Data 2[31:0]	Dato leído del registro indicado en Read Reg 2
Write Data[31:0]	Dato a escribir en el registro indicado en Write Reg
Read Reg 1[5:0]	Código del registro a leer por salida Read Data 1
Read Reg 2[5:0]	Código del registro a leer por salida Read Data 2
Write Reg[5:0]	Código del registro a escribir

Write	Operación del registro
0	Lectura: Read Data 1[31:0] = Contenido almacenado en el registro indicado por Read Reg 1[5:0] Read Data 2[31:0] = Contenido almacenado en el registro indicado por Read Reg 2[5:0]
1	Escritura por flanco de bajada de CLK: Registro indicado por Write Reg[5:0] se carga con Write Data[31:0] Además se realizan las dos lecturas al igual que con Write=0

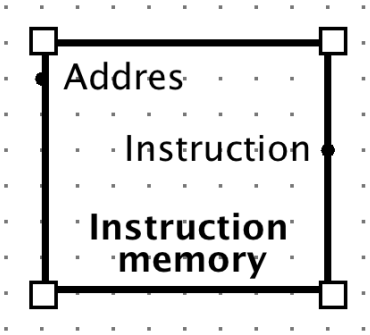
Memoria de datos (Data Memory 32)



Señales	
clk	Señal de reloj para sincronizar las escrituras en memoria
Write	Señal de escritura
Read	Señal de lectura
Read data[31:0]	Bus de datos de salida (lectura)
Write data[31:0]	Bus de datos de entrada (escritura)
Addres[31:0]	Bus de direcciones

Write	Read	Operación de la memoria
0	1	Lectura: Read data[31:0] = M(Addres[31:0])
1	0	Escritura por flanco de bajada de clk: M(Addres[31:0]) ← Read data[31:0]

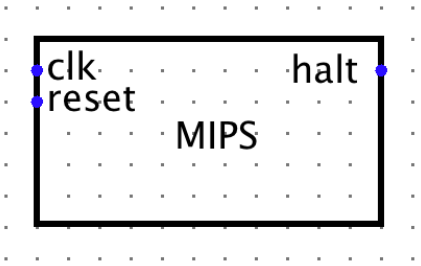
Memoria de Instrucciones (Instruction memory)



Señales	
Instruction[31:0]	Bus de datos de salida (lectura)
Addres[31:0]	Bus de direcciones

Operación de la memoria
Continuamente: Instruction[31:0] = M(Addres[31:0])

El interfaz del procesador a diseñar es el siguiente:



Entradas:

- clk: reloj del sistema que sincronizará todas las instrucciones
- reset: puesta a cero de todos los elementos de memoria

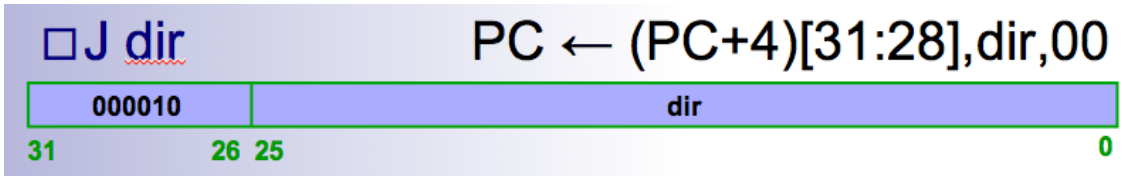
Salida:

- halt: señal que se activará cuando se ejecute una instrucción especial denominada “halt”.

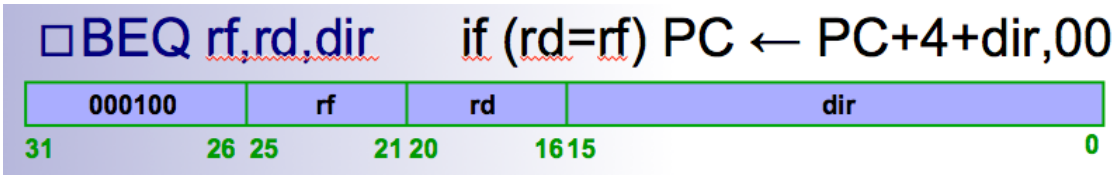
INSTRUCCIONES

El conjunto mínimo de instrucciones que debe soportar el procesador son:

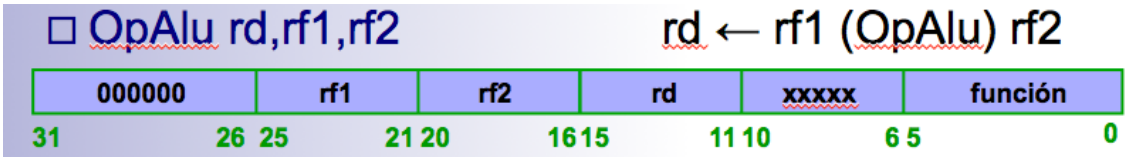
- **Instrucciones de Bifurcación:**
  - Salto incondicional: **J dir**



  - Salto condicional: **BEQ rf, rd, dir**

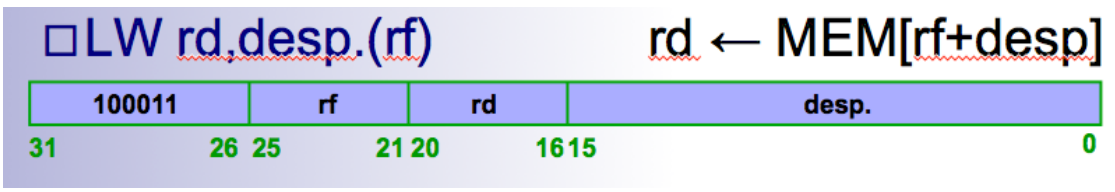


- **Instrucciones de transformación de información:**
  - Operaciones con la ALU: **OpAlu rd, rf1, rf2**

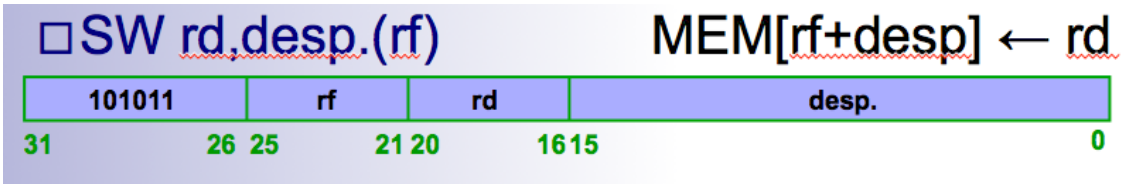


OpAlu	función	OpAlu	función
ADD	100000	OR	100101
SUB	100010	SLT	101010
AND	100100		

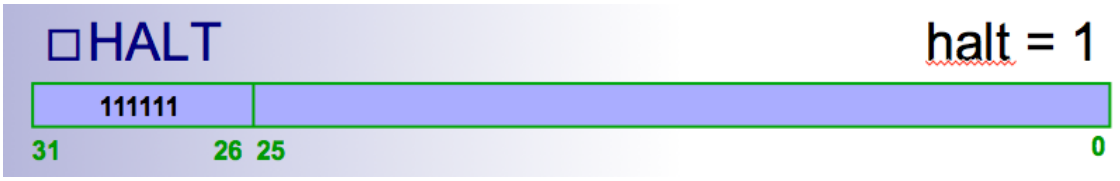
- **Instrucciones de transferencia:**
  - Carga: **LW rd, desp.(rf)**



- Almacenamiento: **SW rd, desp.(rf)**



- **Instrucción especial:**
  - **HALT**



Tareas a desarrollar por el alumno

1.- Diseño e implementación de la unidad de datos en Logisim: Para la implementación de la unidad de datos puedes utilizar como bloques básicos de diseño: ALU, banco de registros, registros, sumador/restador, biestables, multiplexores, codificadores, decodificadores y puertas lógicas (todos estos elementos del ancho de bits que necesites).

Debes bajarte de la plataforma de enseñanza virtual un prototipo de proyecto sobre el cual vas a diseñar tu procesador. El nombre del prototipo es TG00MIPS.ZIP. Dicho nombre deberá ser cambiado antes de empezar a trabajar con él. Deberás cambiar los 4 primeros caracteres (**TG00**) por los que correspondan:

- **T**: Titulación del alumno (**I** Ing. Informática, **S** Ing. del Software, **C** Ing. de Computadores)
- **G**: Grupo (**A, B**)
- **00**: Número de equipo del alumno con dos dígitos (**01, 02, …**)

Así por ejemplo el equipo 13 del grupo B de la ingeniería informática, deberá trabajar sobre el proyecto con nombre: **IB13MIPS.**

2.- Implementación de la unidad de control: Diseña una unidad de control que implemente el ciclo de instrucción diseñado para el procesador y active las señales de control que van a la unidad de datos de forma adecuada.

3.- Cálculo del tiempo de ciclo: Calcula el tiempo de ciclo (duración en “ticks” de simulación del ciclo de reloj) para tu implementación del procesador MIPS, y

modifica el reloj principal del procesador (en el circuito TG00MIPS) de acuerdo a la duración calculada.

Tiempo de Ciclo:	ticks

El retardo (en ticks) de las unidades funcionales que se te han proporcionado es el siguiente (considera 0 el retraso de el resto de elementos que hayas añadido tu en el diseño):

- **Memoria de instrucción: 4** (desde que se pone una dirección válida en la entrada de direcciones, hasta que aparece el dato correcto de dicha dirección hay que esperar 4 ticks)
- **Memoria de datos: 4** (tanto de lectura como de escritura. Para lectura hay que esperar 4 ticks desde que se pone la dirección hasta que está disponible el dato. En escritura lo mismo, hay que esperar 4 ticks desde que se pone la dirección y el dato a escribir, hasta que se ha llevado a cabo la escritura)
- **ALU: 4** (desde que se cambia alguna entrada hasta que aparece el resultado correcto en la salida)
- **Banco de registros: 2** (tanto para lectura como para escritura).

4.- Programación del sistema: Introduce el siguiente programa en tu procesador para comprobar el correcto funcionamiento del mismo. Se trata de un código que calcula la suma de los elementos de un vector (similar al realizado en las prácticas de programación en ensamblador MIPS).

Pasa a código máquina la instrucciones del segmento de código (.text) utilizando el formato de instrucción mostrado al principio de este documento. Introdúcelo a partir de la posición 0 de memoria (la etiqueta “main” se corresponde con esa posición 0 de memoria) en el fichero asociado a la memoria de instrucciones (TG00rom.txt).

Introduce los valores del segmento de datos (.data) en el fichero asociado a la memoria de datos (TG00ram.txt). Ten en cuenta que la etiqueta “datos” sería la posición 0 de memoria, “tam” la 32, “res” la 36, etc.

Una vez metido todo el contenido en las dos memorias, simula el procesador y comprueba que el resultado de la simulación es correcto.

Calcula el tiempo total de ejecución del programa en tu procesador, sabiendo el tiempo de ciclo calculado anteriormente y el número de instrucciones que se deben ejecutar:

Tiempo de ejecución:	ticks

Comprueba que el tiempo que has calculado se corresponde con el obtenido en la simulación.

Programa para la simulación

```
.data
datos: .word 2, 4, 6, 8, -2 -4, -6 -7
tam:   .word 8
res:   .word 0
uno:   .word 1
cuatro: .word 4

.text
main:  lw $8, tam($0)          # $8 contador de elementos a leer
       add $9, $0, $0          # $9 dirección del vector “datos” a leer
       lw $1, uno($0)          # $1 valor 1
       lw $4, cuatro($0)       # $4 valor 4
       sub $11, $11, $11       # $11 acumulador de la suma
loop:  lw $10, 0($9)
       add $11, $11, $10
       add $9, $9, $4
       sub $8, $8, $1
       beq $8, $0, salir
       j loop
salir: sw $11, res($0)
       halt
```

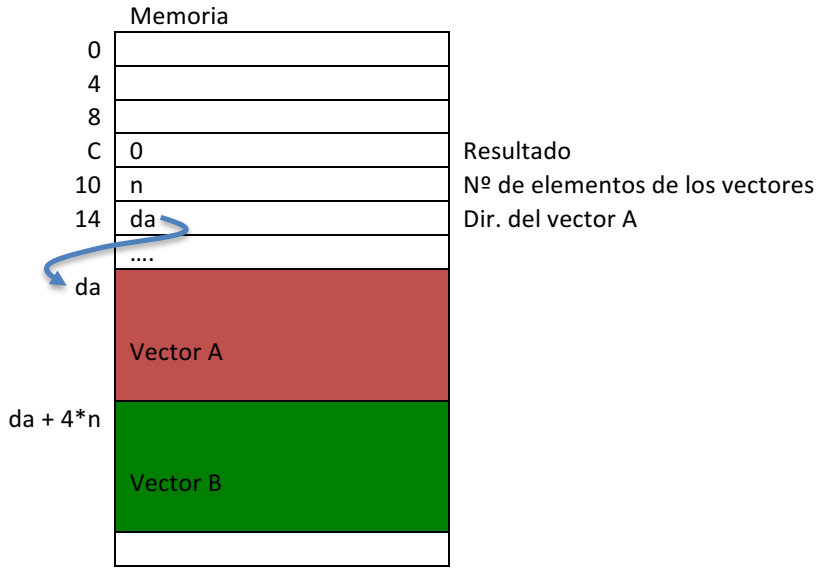
Práctica de programación del procesador

El objetivo de esta práctica es realizar un programa que se ejecute en el procesador diseñado.

El programa a realizar debe calcular la suma de las diferencias en valor absoluto de los elementos de dos vectores (SAD), operación que se utiliza para medir cuanto se parecen dos vectores A y B.

$$SAD(A,B) = \sum_i |A[i] - B[i]|$$

Estos dos vectores estarán formados cada uno por un número (n) de valores enteros (32 bits en C2) especificados en la posición $10_{(16}$ y almacenados consecutivamente a partir de la posición de memoria indicada en la dirección $14_{(16}$. Se realizará la resta elemento a elemento de los vectores (el primero de A menos el primero de B, el segundo menos el segundo, …) acumulando la suma total del valor absoluto de dichas restas. Debe almacenar el resultado en la posición $C_{(16}$ (no hay que tener en cuenta si se produce desbordamiento al calcular el SAD). Las posiciones de memoria $0_{(16}$, $4_{(16}$, y $8_{(16}$ se pueden usar para variables que necesite el programa. El esquema de la memoria de datos para este programa sería:



Tareas a desarrollar por el alumno

1.- Diseño del programa para la implementación monociclo: Diseña el programa propuesto utilizando las instrucciones disponibles en la implementación monociclo del procesador. Introdúcelo a partir de la posición 0 de la memoria de instrucciones (fichero TG00rom.txt).

Introduce los siguientes valores decimales (enteros de 32 bits) a partir de la posición $C_{(16}$ de la memoria de datos (fichero TG00ram.txt): 0, 4, 24, 25, -3, -6, 1, 23, -1, -4,-1. Esto corresponde a un ejemplo de vectores de 4 elementos.

2.- Simulación del procesador monociclo con el programa anterior: Simula la ejecución del programa en el procesador monociclo y comprueba que el resultado es correcto.

Anota el tiempo total de simulación del programa:

Tiempo de ejecución: ticks