Memoria de prácticas de Estructura de Computadores

Por Luis Díaz del Río Delgado

Practica 1:

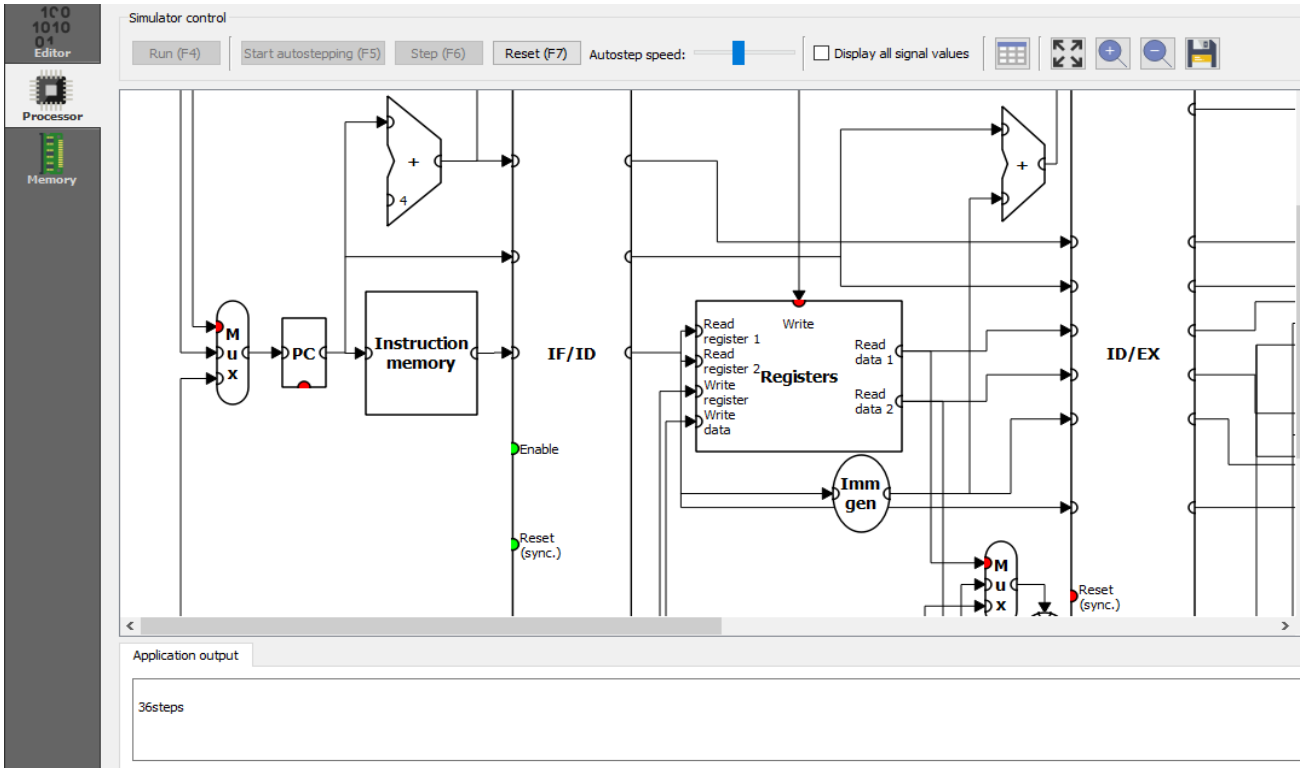
Ejercicio 1.a

```
.data

strl: .string "36steps"   #declaramos una variable string y la incializamos con -36Steps-
      .word 0

.text

la a1,strl #Guardamos la dirección de la variable en a1 para que posteriormente se pueda imprimir.
li a0,4    #Cargamos en a0 el valor 4.
ecall      #Al hacer ecall, el valor guardado en a0 llamará a la su función correspondiente.
           #En este caso, llama a print_string ya que le hemos pasado un 4, e imprimirá el string de a1.


```

En este ejercicio nos pedían que se imprimiera por pantalla la cadena de caracteres “36Steps”.

Para esto declaramos una variable, que nosotros hemos llamado str1, y la inicializamos con “36Steps”.

Tras eso, guardamos la dirección de nuestra variable en a1, y cargamos en a0 el valor 4. Esto se hace ya que en Ripes existen las llamadas Environment calls, dependiendo del valor cargado en a0, llamará a una función u otra. En nuestro caso la función a la que llamamos es print_string, que leerá nuestro string y lo imprimirá por pantalla tras hacer el ecall.



Aquí se muestra la salida de nuestro programa, con la cadena de caracteres “36Steps”

Ejercicio 1.b

En el apartado “b” del ejercicio nos pedían que hiciéramos lo mismo, pero esta vez imprimiendo “36” en una línea y “Steps” en otra, aprovechando que en código Ascii el número 10 corresponde al salto de línea.

```
.data
str1: .string "36"    #Declaramos dos cadenas de carácteres, por un lado -36- y por el otro -steps-
      .word 0
str2: .string "steps"
      .word 0
saltodelinea: .word 10   #Aquí declaramos una variable a la que útilizaremos en cada salto de línea.
                         #La incializamos a 10 ya que queremos hacer un salto de línea mediante Ascii.
.text
la a1,str1               #Como en ejercicio anterior, guardamos la dirección en a1 y cargamos 4 en a0.
li a0,4
ecall                    #Se imprime -36-

la a1, saltodelinea      #Guardamos en a1 la dirección de nuestra variable en a1.
li a7, 11                #En este caso, guardamos en a7 el valor 11, que llama a print_char.
ecall                    #Se imprime el salto de línea.

la a1, str2              #Como antes, se imprime el string de la misma manera.
li a0, 4
ecall                    #Se imprime -Steps-
```

Como antes no declaramos las cadenas de texto correspondientes, para este ejercicio solo necesitamos dos. Además, esta vez declaramos un int y lo inicializamos a 10.

Esto lo hemos hecho así ya que, en el caso de que hubiera que hacer más de un salto de línea, el código quedará algo más encapsulado. En este ejercicio no es necesario, pero es una buena práctica.

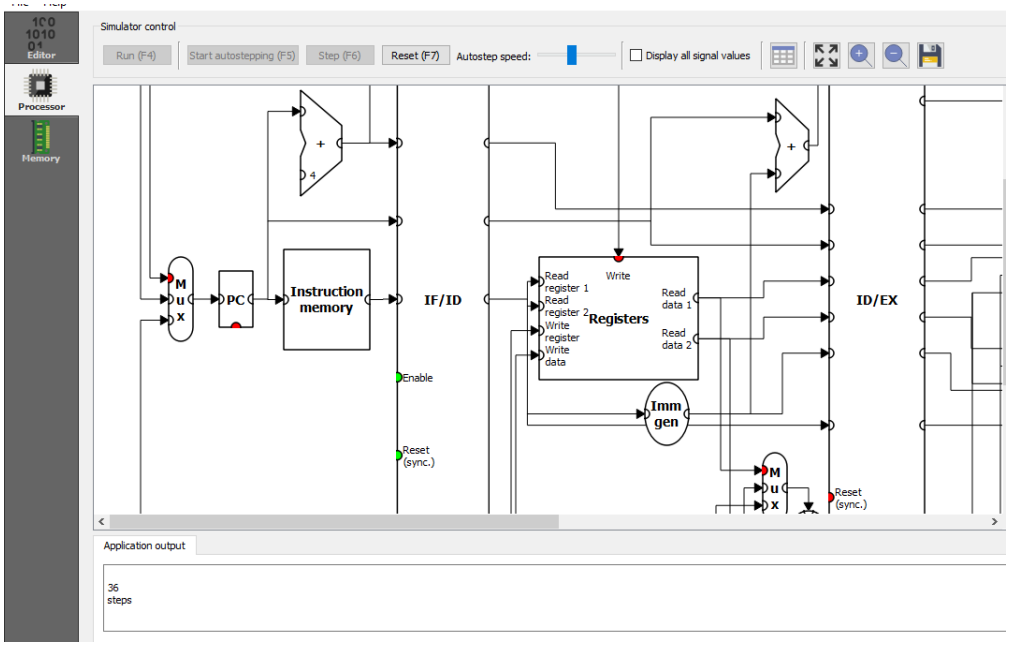
Una vez las variables están declaradas e inicializadas, imprimimos el primer string, de nombre str1 e inicializada como “36”, mediante el proceso descrito en el apartado anterior.

Tras eso, guardamos en a1 la dirección de memoria de nuestra variable. Como queremos imprimir un santo de línea mediante código Ascii, la variable está inicializada a 10 que es el número correspondiente a esta acción.

A continuación, cargamos en a7 el valor 11, esto llama a la función print_char, e imprime el símbolo Ascii correspondiente al valor guardado en a1.

Al hacer el ecall, se imprime el salto de línea.

Para finalizar, se imprime el segundo string (str2) que está inicializado a “Steps”.



Aquí se muestra la salida de nuestro programa, con la cadena de caracteres “36” seguido de un salto de línea y la cadena de caracteres “Steps”

Ejercicio 2

En este ejercicio nos pedían que realizáramos la operación matemática F = (A+B) - (C + D), siendo A=5, B=3, C=2, D=2.

```
.data
#Variables inicializadas con sus respectivos números
A: .word 5
B: .word 3
C: .word 2
D: .word 2

#Variable donde se almacena el resultado de la ecucación, inicializada a 0
F: .word 0

.text
begin:
lw a2,A  #Cargamos el contenido de nuestras variables A,B,C y D en a2, a3, a4 y a5 respectivamente
lw a3,B  #No hacemos esto ni en a0 ni el a1 ya que las vamos a necesitas más tarde para imprimir el resultad
lw a4,C
lw a5,D

add t0, a2,a3  #Hacemos la suma de a2 y a3(A+B) y guardamos el resultado en el registro temporal t0
add t1, a4,a5  #Hacemos la suma de a4 y a5(C+D) y guardamos el resultado en el registro temporal t1
sub t2, t0,t1  #Hacemos la resta de t0 y t1, y guardamos el resutado en t2.

la t3,F        #Guardamos en t3 la dirección de memoria de F.
sw a1, 0(t3)   #Almacenamos el valor de a1 en la palabra de memoria cuya dirección es t3+0

mv a1,t2       #Movemos el valor de t2 a a1.
li a0,1        #Cargamos en a0 el valor 1 para llamar a la función print_int
ecall          #Imprimimos el resultado.

end:
nop
```

Lo primero que hacemos es definir las variables A, B, C y D y las inicializamos con sus respectivos valores. A su vez, definimos la variable F y la inicializamos a 0, en esta variable será donde almacenemos el resultado.

Tras definir e inicializar las variables cargamos el contenido de las variables en los registros a2, a3, a4 y a5. La razón por la que no utilizamos a1 y a0 es debido a que, como hay que imprimir el resultado, necesitamos ambas libres.

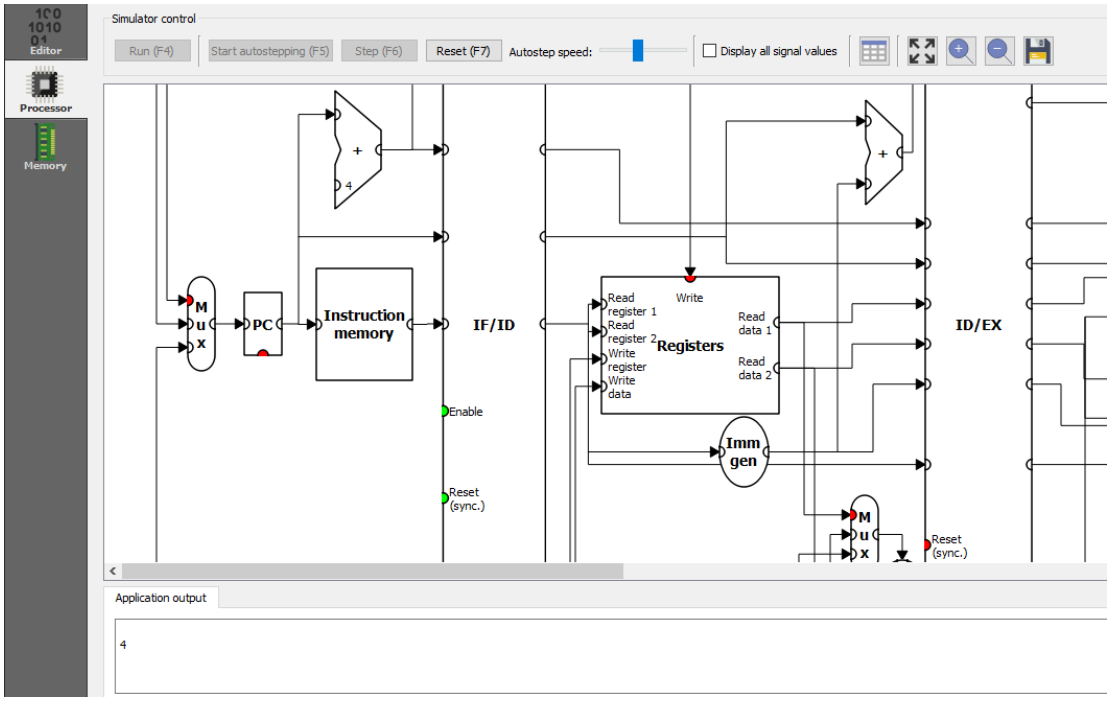
Acto seguido hacemos las operaciones, empezamos sumando a2 y a3 (A+B) y guardamos el resultado en un registro temporal(t0).

Hacemos la suma de a4 y a5(C+D) y lo guardamos de la misma manera en un registro temporal(t1).

Finalmente, hacemos la resta de t0 y t1, y lo guardamos en otro registro temporal(t2).

Una vez terminada la operación de (A+B) - (C + D), hay que guardar el resultado en F.

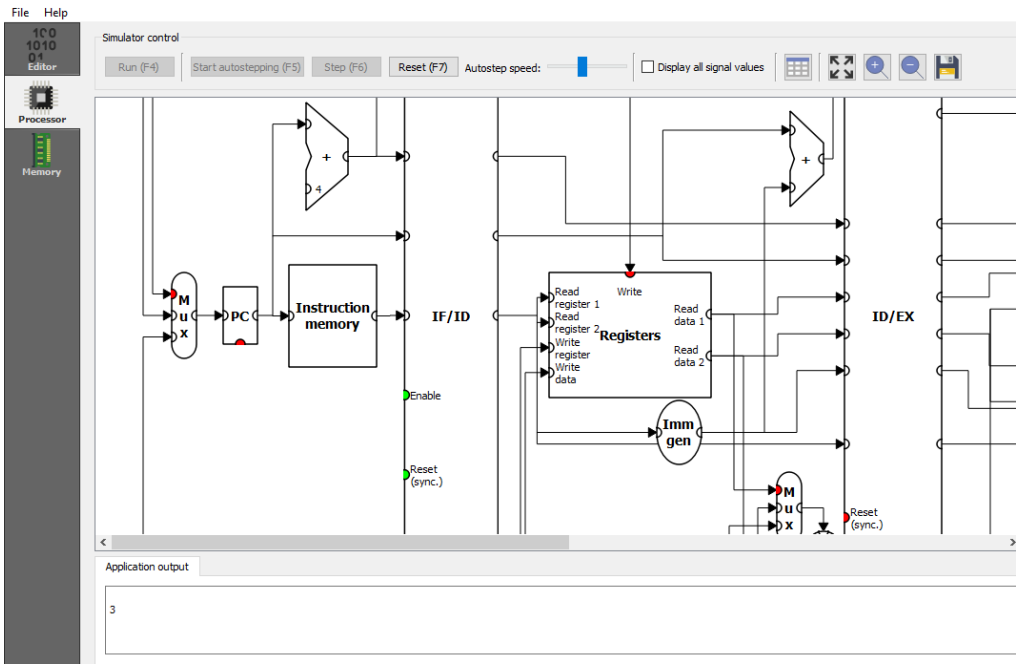
Para esto primero guardamos en el registro temporal t3 la dirección de memoria de F, y almacenamos el valor de a1 en t3. Movemos el valor de t2(nuestro resultado) en a1, y cargamos en a0 el valor 1 para llamar a la función print_int.



Aquí se ve el resultado de la ecuación con los números pedidos en el enunciado del ejercicio.

```
.data
#Variables inicializadas con sus respectivos números
A: .word 8
B: .word 3
C: .word 6
D: .word 2

#Variable donde se almacena el resultado de la ecucación, inicializada a 0
F: .word 0
.text
lw a2,A
lw a3,B
lw a4,C
lw a5,D

add t0, a2,a3
add t1, a4,a5
sub t2, t0,t1

la t3,F
sw a1, 0(t3)

mv a1,t2
li a0,1
ecall
```

Mismo ejercicio, pero con otros valores.



Resultado de la variación del código mostrada en la imagen anterior.

Ejercicio 3.a

En este ejercicio se pide que se calcule y se saque por pantalla el resto resultante de una división con valores a nuestra elección. Nosotros hemos escogido hacer la división 10/6.

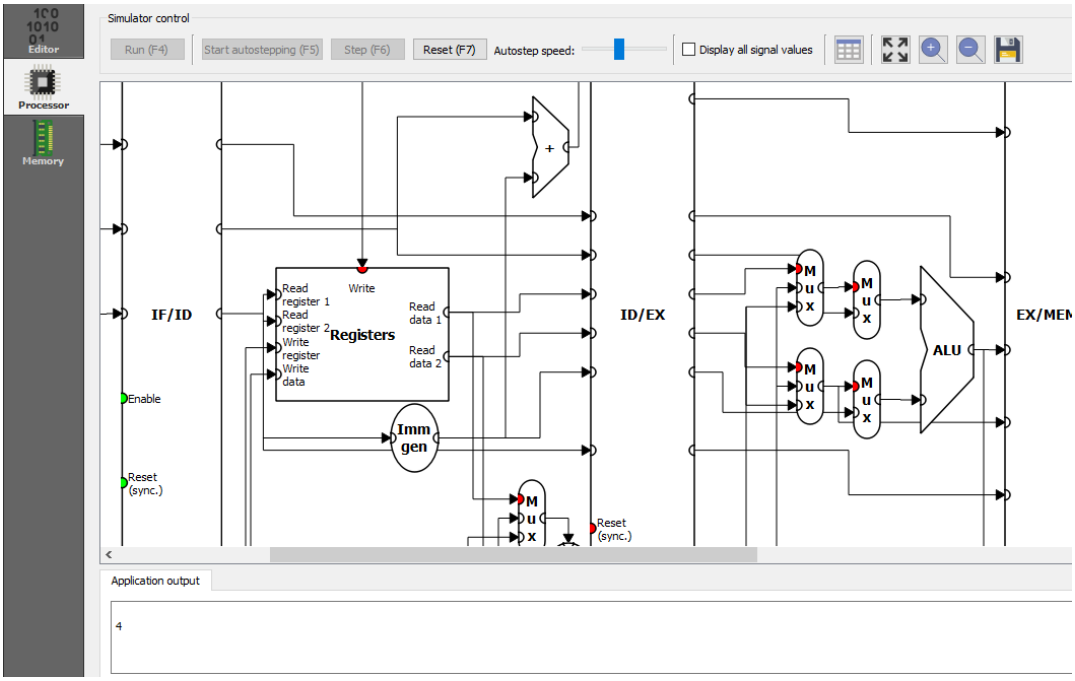
```
.data
A: .word 10  #Declaramos las variables A y B, con valores 10 y 6 respectivamente
B: .word 6

R: .word 0   #Variable R, donde se guarda el resultado, incialiado a 0.
.text
begin:

lw a2, A     #Cargamos las variables en a2 y a3. a1 y a0 están reservados para imprimir el resultado.
lw a3, B

rem t0, a2, a3  #Hacemos rem de a2 y a3, esto hace la división y guarda el resto en t0.

la t1, R        #Aquí seguimos el mismo procedimiento que en el ejercicio anterior.
sw a1, 0(t1)

mv a1, t0
li a0, 1
ecall

end:
nop
```

Lo primero, como siempre, es declararnos dos variables, A y B, y las inicializamos en 10 y 6 respectivamente. También declaramos una variable llamada R que inicializamos a 0.

Cargamos las variables A y B en a2 y a3.

Hacemos rem de a2 y a3, esto hace la división de a2 y a3 y guardan en t0 el resto de dicha división.

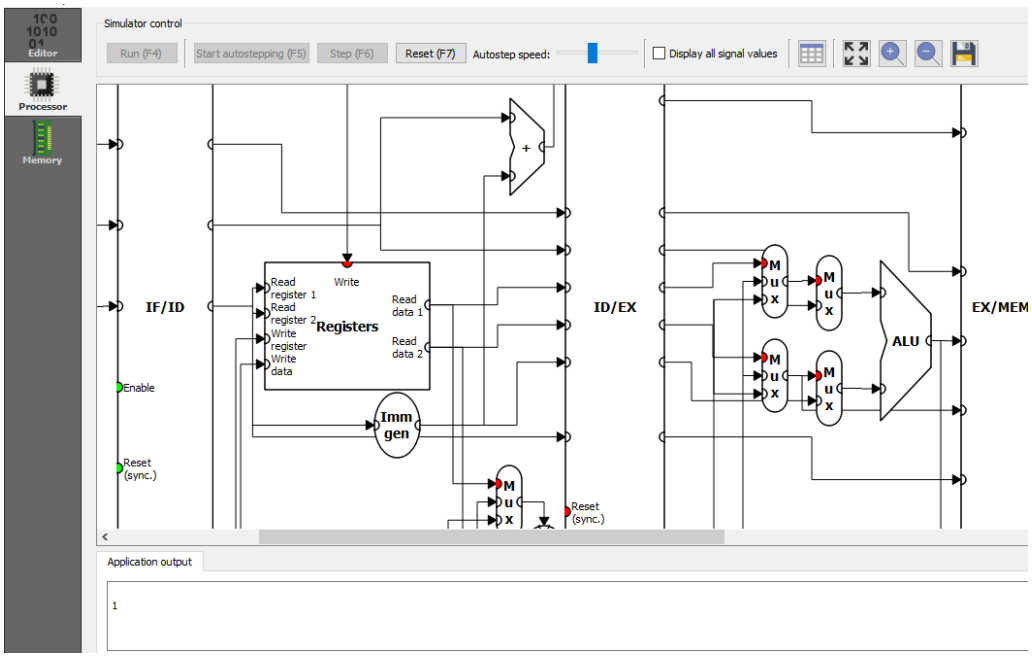
Para finalizar, seguimos el mismo procedimiento que hemos seguido en el ejercicio anterior para guardar el resultado en la variable F e imprimir el resultado.



Vemos que el resultado es el correcto.

```
.data

A: .word 19
B: .word 2

R: .word 0

.text
begin:

lw a2, A
lw a3, B

rem t0, a2, a3

la t1, R
sw a1, 0(t1)

mv a1, t0
li a0, 1
ecall

end:
nop

```

Aquí modificamos el valor de las variables.



El resultado tras la modificación del código.

Ejercicio 3.b

En este apartado nos piden que hagamos lo mismo que en el ejercicio anterior, pero esta vez sin utilizar la instrucción rem. Para hacer esto tenemos que utilizar la propiedad de la división en el que dividendo es el divisor multiplicado por el cociente más el resto.

Antes de meterse de cabeza en el programa, hay que despejar el resto. Una vez hecho esto nos queda que el resto = (dividendo/cociente) - divisor.

```
.data
Dividendo: .word 10 #Declaramos dos variables y las inicializamos con 10 y 6.
Divisor:   .word 6

Cociente:  .word 0  #Esta vez declaramos dos variables y las inicializamos a 0.
Resto:     .word 0

.text
begin:

lw a2, Dividendo    #Cargamos nuestras variables en a2, a3 y a4
lw a3, Divisor
lw a4, Cociente

div a4, a2, a3     #Como necesitamos el cociente, hacemos la división y guardamos en resultado en -Cociente-

div t0,a2,a4       #Se hace la división del dividendo partido del cociente. Guardamos el resultado en t0
sub t1, t0, a3     #Ahora a t0 se le resta el divisor y sacamos el resto. Lo guardamos en t1.

la t2, Resto       #Aquí seguimos el mismo procedimiento que en los anteriores ejercicios para guardar en
sw a1, 0(t2)       #la variable Resto el resultado e imprimir por pantalla el resultado

mv a1, t1
li a0,1
ecall

end:
nop
```

Declaramos nuestras variables, esta vez las hemos llamado Dividendo y Divisor para más claridad a la hora de hacer el ejercicio. A su vez nos declaramos dos variables, Resto y Cociente, los dos inicializados a 0. Ya que hay que calcular el cociente hemos decidido hacerlo así por simple claridad, se podría haber guardado simplemente en un temporal.

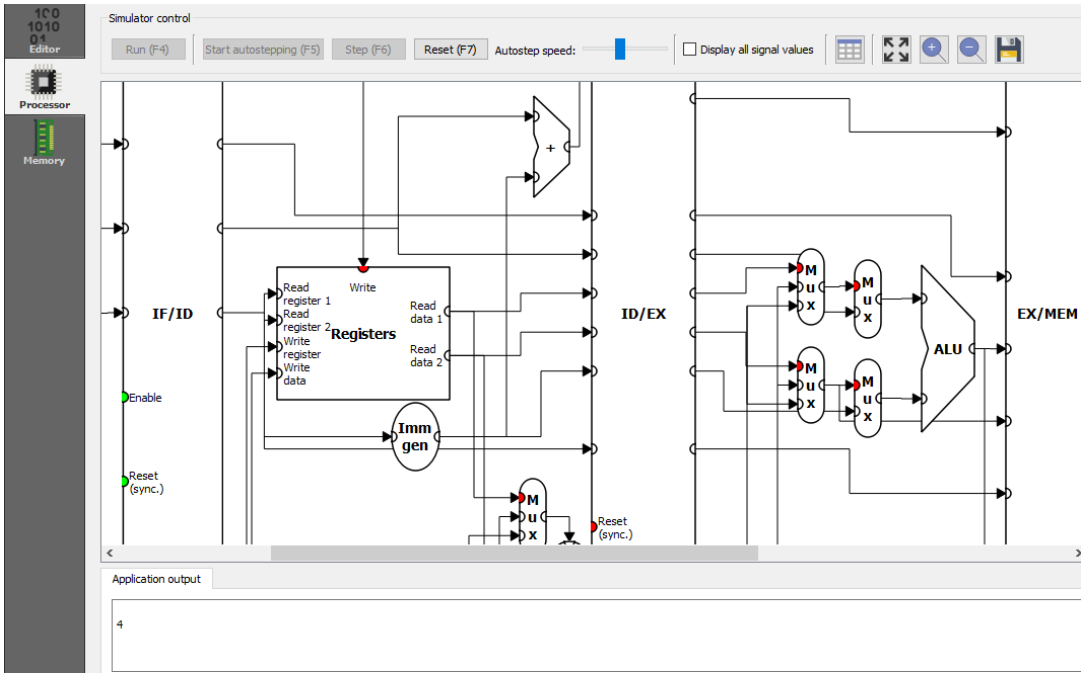
Cargamos las variables Dividendo, Divisor y Cociente en a2, a3 y a4 respectivamente.

En la primera operación calculamos el cociente mediante la división del Dividendo(a2) entre el divisor(a3) utilizando la instrucción div.

Después calcular el cociente, se divide, utilizando nuevamente la instrucción div, el dividendo y el cociente(a2/a4). Se guarda en t0.

Para finalizar, se hace la resta de t0 y a3(Divisor) y se guarda en el temporal t1.

Una vez calculado esto, guardamos el resultado en R e imprimimos el mismo de la misma manera que en el apartado anterior y el ejercicio 2.



Nuestro resto.

Practica 2:

Ejercicio1:

En este ejercicio se nos pide crear una cadena de texto e iniciarla con una palabra de seis caracteres en minúsculas, y convertir la palabra a mayúsculas sin utilizar saltos.

```
.data
str: .string "qwerty" #Creamos una variable de tipo string y la inicializamos.
     .word 0

newline: .word 10    #Creamos una variable de tipo entero y la inicializamos a 10 ya que es para el salto de linea.
.text
begin:
li t6, 32 #Cargamos en t6 el valor 32(Distancia entre mayusculas y minusculas en ASCII).

la a1, str #Imprimimos nuestro string sin modificar.
li a0, 4
ecall

la a1, newline #Imprimimos el salto de linea.
li a7, 11
ecall

#convertir la palabra a mayúsculas.
la a1, str #Metemos en a1 la dirección de la variable de nuestra cadena de caracteres.

lb t1, 0(a1) #Cargamos en t1 el primer byte de nuestro string. En este caso -q-.
sub t1, t1, t6 #Le restamos a nuestro byte 32, para convertirlo en mayúscuulas.
sb t1, 0(a1)   #Cuardamos el byte modificado en el string inicial. Qwerty

lb t1, 1(a1)   #Repetimos esto todo el rato, hasta modificar todo nuestro string.
sub t1, t1, t6
sb t1, 1(a1) #QWerty

lb t1, 2(a1)
sub t1, t1, t6
sb t1, 2(a1)   #QWErty

lb t1, 3(a1)
sub t1, t1, t6
sb t1, 3(a1) #QWERty

lb t1, 4(a1)
sub t1, t1, t6
sb t1, 4(a1)   #QWERTy

lb t1, 5(a1)
sub t1, t1, t6
sb t1, 5(a1) #Con este sb, ya tenemos todo nuestro string en mayusculas. QWERTY

li a0, 4 #Imprimimos QWERTY.
ecall

end:
nop
```

Primero declaramos e inicializamos una cadena de texto, de seis caracteres, en este caso “qwerty”.

Además, creamos una variable de tipo entero inicializada a diez, para hacer los saltos de línea.

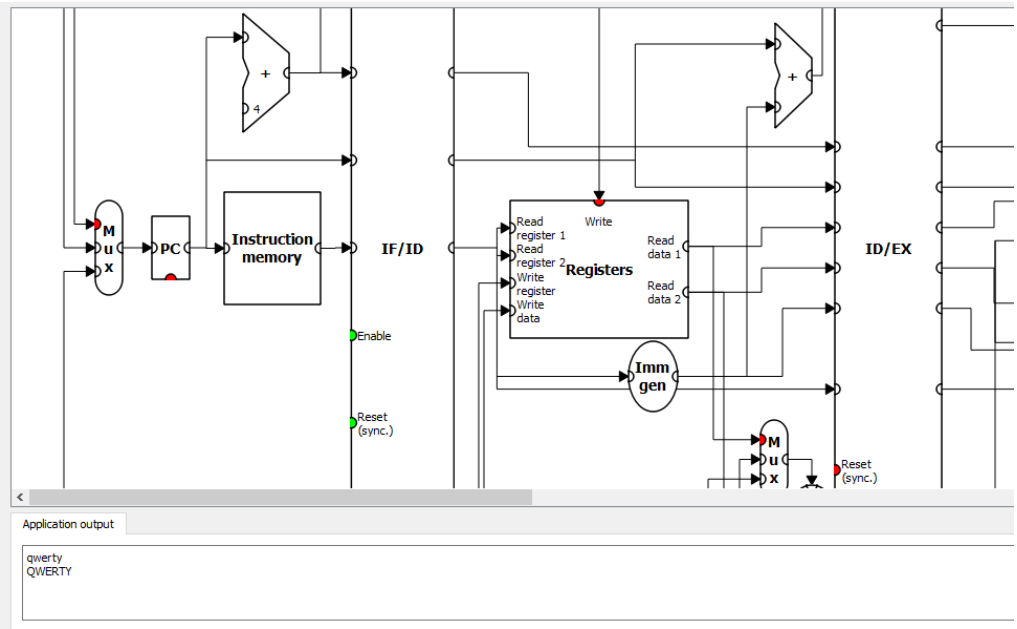
Tras esto, cargamos en t6 el valor 32, que es la diferencia entre mayúsculas y minúsculas en ASCII.

Imprimimos de la misma forma que siempre tanto el string original como el salto de línea y empezamos a transformar el string a mayusculas.

Cargamos la dirección de memoria del string en a1. Después de eso, guardamos en t1 el primer byte de nuestro string, restamos el valor de la letra almacenada en t1 con t6, es decir t1 –32, y guardamos el resultado en el mismo t1. Para acabar, guardamos el carácter, ahora siendo mayúsculas, en su posición correspondiente.

Hacemos esto de la misma manera para cada byte, hasta llegar al último byte.

Para finalizar, imprimimos el string modificado.



Nuestro string en minúsculas y tras un salto de línea en mayúsculas.

Ejercicio 2:

En este ejercicio nos piden que creemos una cadena de texto, de tamaño indeterminado, que tenga cualquier combinación de minúsculas, espacios y finalizada en un punto. Al igual que en el ejercicio anterior, convertir la cadena a mayúsculas, parando cuando llegue a un punto, pero esta vez utilizando saltos.

```
.data
str: .string "hola buenos dias." #Creamos una variable y la inicializamos.
     .word 0

newline: .word 10
.text
begin:
li t6, 32 #Diferencia entre letras y espacio en ASCII.

la a1, str  #Imprimimos el string sin modificar.
li a0, 4
ecall

la a1, newline #Imprimimos el salto de linea.
li a7, 11
ecall

la a1, str #Metemos en a1 la dirección de memoria del string.

#Bucles y excepción de salida:

loop:                #Creamos una etiqueta llamada loop. Aqui estará nuesto bucle.
li t5, 46            #Cargamos en t5 el valor 46, que es ASCII corresponde a un punto.
lb t1, 0(a1)
beq t1, t5, end_loop  #Si nuestro valor t1 = t5, es decir, igual a un punto, se imprime el string.

beq t1, t6, continue  #Este bucle es para evitar que al restar al string 32, el espacio se combierta en un NULL.

sub t5, t1, t6        #Nuestro caracter -32, guardada en t5 para ver correctamente el auto-stepping
sb t5, 0(a1)          #Guardamos el byte modificado en el string original.
addi a1, a1, 1        #Por cada loop se suma 1 a a1, esto recorre el string.

j loop                #Llama al loop para que se repita hasta que salte la condición de salida.

continue:             #Una excepción para evitar que el espacio se combierta en null al ser restado.

sb t6, 0(a1)          #Guardamos directamente el espacio sin restarlo.
addi a1, a1, 1        #Pasamos al siguiente caracter del string.
j loop                #Llamamos al loop para que siga hasta que se cumpla la expceción de salida.

end_loop:             #Una vez llegado al punto, se imprime el string.

la a1, str            #Imprimimos el string modificado.
li a0, 4
ecall

```

Primero nos creamos un string y lo inicializamos, en este caso, como “hola buenos dias.”

Además, creamos una variable de tipo entero, inicializada a diez, para el salto de línea.

Después de crear las variables e inicializarlas, lo primero que hacemos es guardar en t6 el valor 32, que es la diferencia entre las mayúsculas y las minúsculas en ASCII, así como el valor que tiene el espacio en ASCII.

Imprimimos el string original en minúsculas y el salto de línea y pasamos a convertir el string a mayúsculas.

Guardamos en a1 la dirección de memoria del string y empezamos con los bucles.

Nos creamos la etiqueta “loop:” en la que estará nuestro algoritmo. Primero guardamos en t5 el valor 46, correspondiente al punto en ASCII.

Tras eso, guardamos en t1 el primer byte del string, se hace así ya que queremos que recorra todo el string desde el principio.

Una vez guardado en t1 el byte correspondiente a la iteración, y antes de hacer ningún cambio al carácter almacenado, hacemos las comprobaciones necesarias. Primero comprobamos si es el final de nuestro string es igual a un punto. Para esto utilizamos “beq”, de tal forma que, si t1 = t5, se salte a la excepción “end_loop”.

Esta condición será nuestra salida del bucle e imprimirá el string resultante.

En el caso de que no sea un punto, se comprueba si es un espacio. Esto es así ya que, al tener el espacio el mismo valor que le vamos a restar a cada uno de los bytes, es decir 32, al hacer la resta pasa a ser un NULL, por lo que solo se imprime todo aquello que esté antes del primer espacio.

Para hacer tal comprobación, utilizamos otro “beq”, en este caso, comparado si t1 = t6. Si esto es así, saltamos a “continue”. En el que guardamos directamente t6, es decir 32, espacio en ASCII, en el byte correspondiente, sin manipular, ya que como he comentado antes, no queremos que se convierta en un NULL.

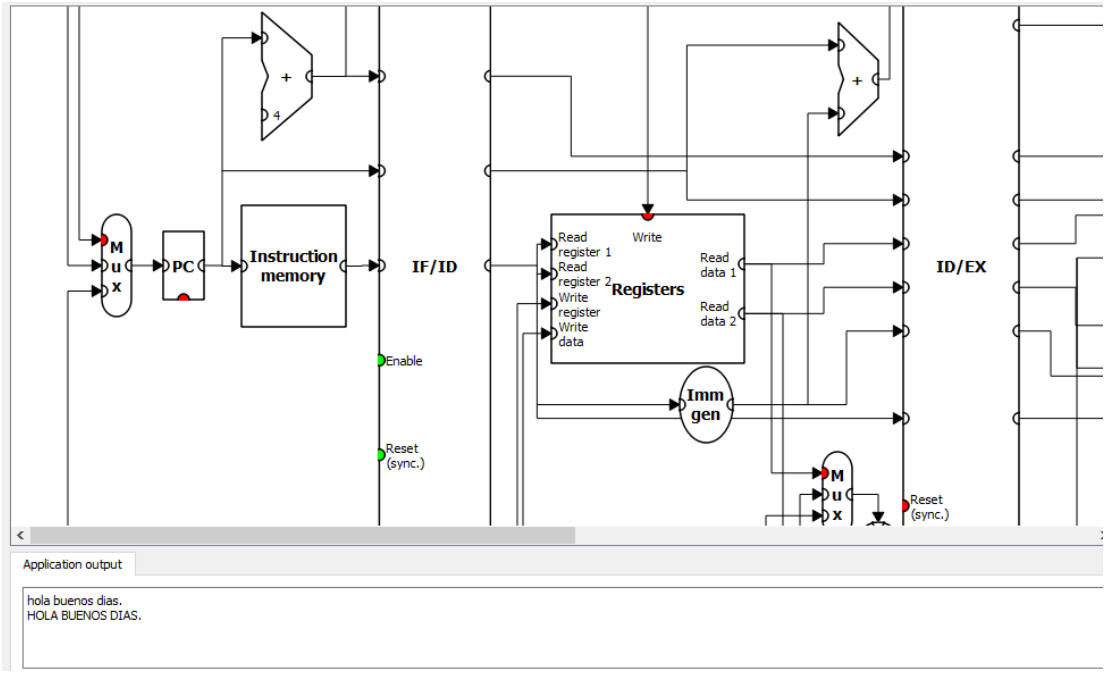
Además, se la hace un addi (add inmediato), de a1 + 1. Esto es así ya que, al no ejecutar el resto del código en “loop” tras el salto, necesitamos apuntar al siguiente byte de la cadena antes de volver a la etiqueta “loop” mediante “j loop”.

De vuelta a nuestra etiqueta “loop”, si el carácter no cumple ninguna de nuestras excepciones, se convierte a mayúsculas mediante la resta de t1 con t6, es decir, el valor numérico de nuestro carácter -32. El resultado es guardado en t5, la razón por la que se guarda el resultado en t5 específicamente es por pura facilidad a la hora de hacer la comprobación de que el algoritmo funciona correctamente mediante el “auto-stepping” de Ripes

Una vez guardado en t5 el carácter modificado, ahora en mayúscula, se guarda en su posición correspondiente. Acabado esto, simplemente guardamos el carácter modificado en su posición.

Se hace, el addi de a1 + 1, para pasar a la siguiente posición de nuestro string.

Para finalizar, se vuelve a llamar a “loop” mediante “j loop” para que se repita el bucle hasta que se cumpla la condición de salida.



Nuestro resultado.

Practica 3:

Ejercicio1:

En este ejercicio se pide que se cree un programa que se capaz de calcular si el año es bisiesto o no.

```
.data
year: .word 2020                    #creamos nuestas variables correspondientes
bisiesto: .string "Es bisiesto"
nobisiesto: .string "No es bisiesto"
dospuntos: .word 58                 #Estas dos variables son simplemente para que quede bonito al imprimir
espacio: .word 32

.text
begin:
la a2, bisiesto       #Cargamos en a2 y a3 la dirección de nuestras cadenas de carácteres
la a3, nobisiesto
addi t0, t0, 4       #Un año es bisiesto si es divisible por 4.

lw a1,year           #Imprimimos nuestro año
li a0, 1
ecall

la a1, dospuntos     #Imprimimos dos puntos.
li a0, 4
ecall

la a1, espacio       #Imprimimos un espacio.
li a0, 4
ecall

lw a1, year          #Cargamos en a1 nuestro año.

rem t2, a1, t0       #Calculamos el resto de la división de nuestro año entre 4.

beq t2, zero, leap   #Si el resto es cero, es bisiesto.

mv a1, a3            #En el caso de que el resto no sea igual a cero, no es bisiesto.

jal end              #Saltamos a end para imprimir si es bisiesto o no.
leap:
mv a1, a2           #En el caso de que se cumpla que es bisiesto, movemos a2 a a1 para imprimir.
end:
li a0, 4            #Imprimimos si es bisiesto o no.
ecall
```

Primero creamos nuestras variables. Una variable de tipo entero, llamada year, donde la inicializaremos con el año que queremos saber si es bisiesto y no. Otras dos variables de tipo string donde guardaremos las cadenas de texto “Es bisiesto” y “No es bisiesto”.

Las otras dos variables, dospuntos y espacio, son simplemente para que quede bien el resultado.

Guardamos en a2 y a3 las direcciones de memoria de nuestras cadenas de texto. Además, haremos un addi para guardar en t0 un 4.

Se hace esto ya que un año es bisiesto si es divisible entre 4.

Imprimimos nuestro año y los caracteres “:” y “ “(Espacio).

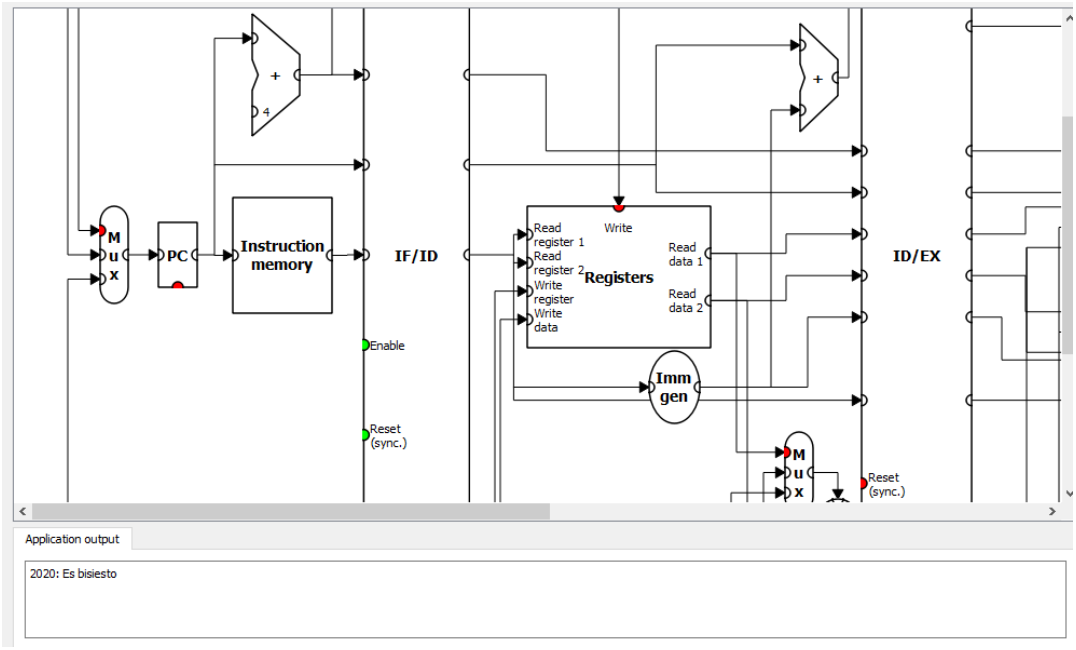
Tras eso cargamos en a1 nuestro año.

Se utiliza la instrucción “rem” con la que calcularemos el resto correspondiente a la división de nuestro año entre cuatro.

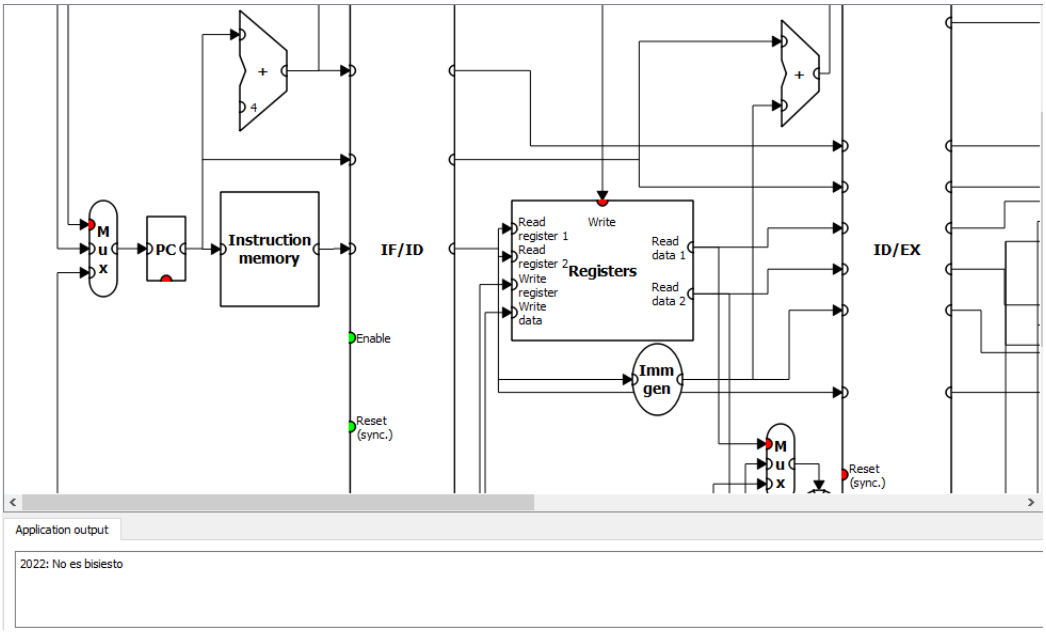
Si t2(el resto de nuestra división) es igual a cero, entonces el año es bisiesto, por lo que salta a “leap:” donde se mueve a2(bisiesto) a a1, para imprimir por pantalla, por ejemplo, “2020: Es bisiesto”.

En el caso de que no sea así, se mueve a3(nobisiesto) a a1, para imprimir por pantalla, por ejemplo, “2021: No es bisiesto”.

Para que no interfiera en el resultado la etiqueta “leap” hacemos un jal que salta directamente a end, donde está nuestro ecall.



Este resultado corresponde al año 2020, que sí es un año bisiesto



En este caso he metido el año 2022, que no es bisiesto, y este es el resultado.

Ejercicio 2:

En este ejercicio nos piden que creemos tres procedimientos, uno que imprima un entero, otro que imprima un string y uno que imprima un salto de línea.

```
.data
entero: .word 5               #Creamos nuestras tres variables correspondientes.
newline: .word 10
str: .string "Hello World!"

.text
begin:
jal print_str                 #Saltamos al procedimiento que imprime el string.
jal print_newline             #Saltamos al procedimiento que imprime el salto de línea.
jal print_entero              #Saltamos al procedimiento que imprime el entero.

end:
li a0, 10
ecall

print_entero:                 #Procedimiento que imprime el entero.
lw a1, entero
li a0, 1
ecall

ret

print_newline:                #Procedimiento que imprime el salto de línea.
la a1, newline
li a0, 4
ecall

ret

print_str:                    #Procedimiento que imprime el string-
la a1, str
li a0, 4
ecall

ret
```

Nos definimos nuestras variables y las inicializamos. En este caso he elegido para el entero el número 5 y para el string la mítica frase “Hello World!”. Además, he creado una variable para el salto de línea.

Al principio llamamos a todos nuestros procedimientos de tal manera que imprima lo siguiente:

Hello World!

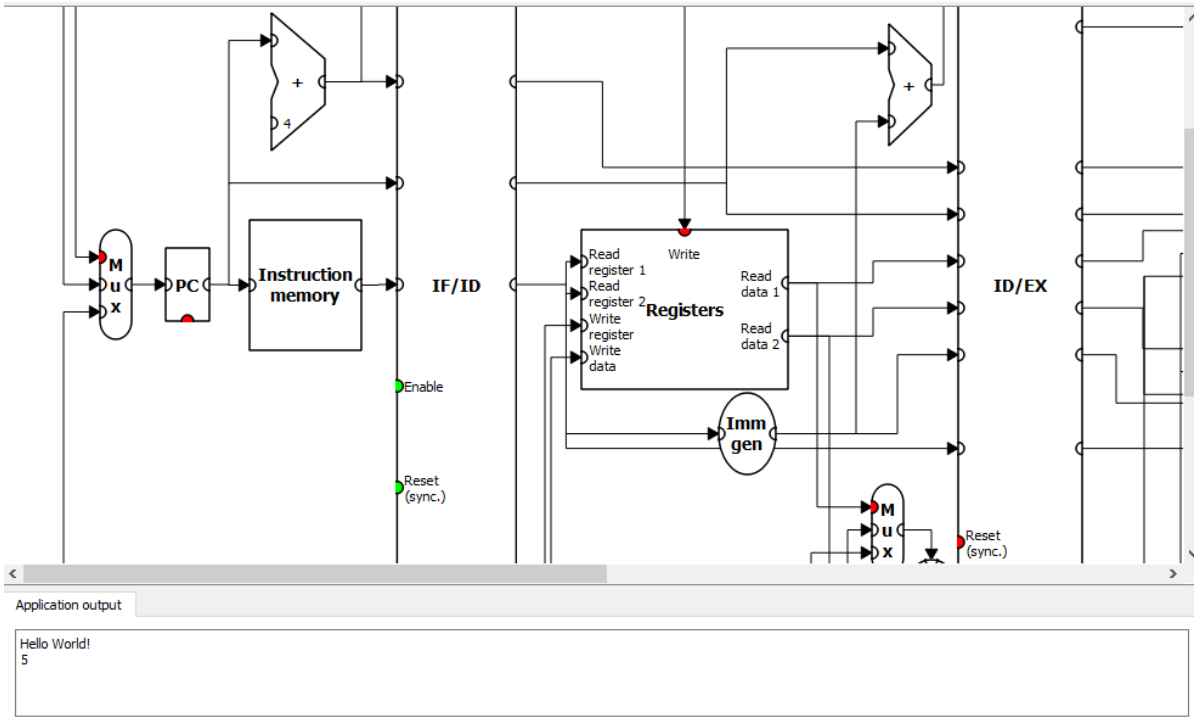
5

Ponemos inmediatamente después nuestro “End” que terminará nuestro programa

En “print_entero:” imprimimos nuestro entero como siempre que queremos imprimir un entero.

En “print_newline” imprimimos como de costumbre los saltos de línea o cualquier símbolo ASCII.

En “print_str:” imprimimos de la misma manera que en el resto de los ejercicios.



Vemos que el resultado es el esperado.

Ejercicio 3:

En este ejercicio se nos pide que hagamos un programa similar al del ejercicio 1, pero esta vez utilizando procedimientos.

```
.data
year: .word 2020                    #Definimos nuestras variables y las inicializamos.
bisiesto: .string "Es bisiesto"
nobisiesto: .string "No es bisiesto"

.text
begin:
la a2, bisiesto                     #Guardamos en a2 y en a3 las direcciones de memoria de nuestros strings.
la a3, nobisiesto

addi t0, t0, 4                      #Guardamos en t0 el valor 4, el cual utilizaremos para calcular si es bisiesto o no.

jal print_year                      #Imprimimos nuestro año.
jal calc_leap                       #Calculamos si es bisiesto o no.
jal print_leap_noleap               #Imprimimos si es o no es bisiesto,

end:                                #Paramos el programa.
li a0, 10
ecall

print_leap_noleap:                  #Desde aquí imprimimos si es o no es bisiesto
li a0, 4
ecall

ret
calc_leap:                          #Calculamos si es bisiesto o no.
rem t1, a1, t0                      #Sacamos el resto de la división de nuestro año entre 4.
beq t1, x0, leap                    #Si el resto es igual a cero saltamos a -leap-

mv a1, a3                           #En el caso de que no sea igual a cero movemos a3 a a1(Para imprimir -No es bisiesto-)

ret

leap:                               #Si es resto es igual a cero, movemos a2 a a1(Para imprimir -Es bisiesto-)
mv a1, a2

ret

print_year:                         #Con esto se imprime nuestro año.
lw a1, year
li a0, 1
ecall

ret
```

Definimos nuestras variables, igual que en el primer ejercicio.

Guardamos en a2 y a3 la dirección de memoria de nuestras cadenas de caracteres.

Guardamos en t0 el valor 4 mediante addi.

Llamamos a nuestros procedimientos, primero “print_year” para imprimir nuestro año, luego “calc_leap” para calcular si es bisiesto o no y finalmente “print_leap_noleap” que imprime si es o no es bisiesto nuestro año.

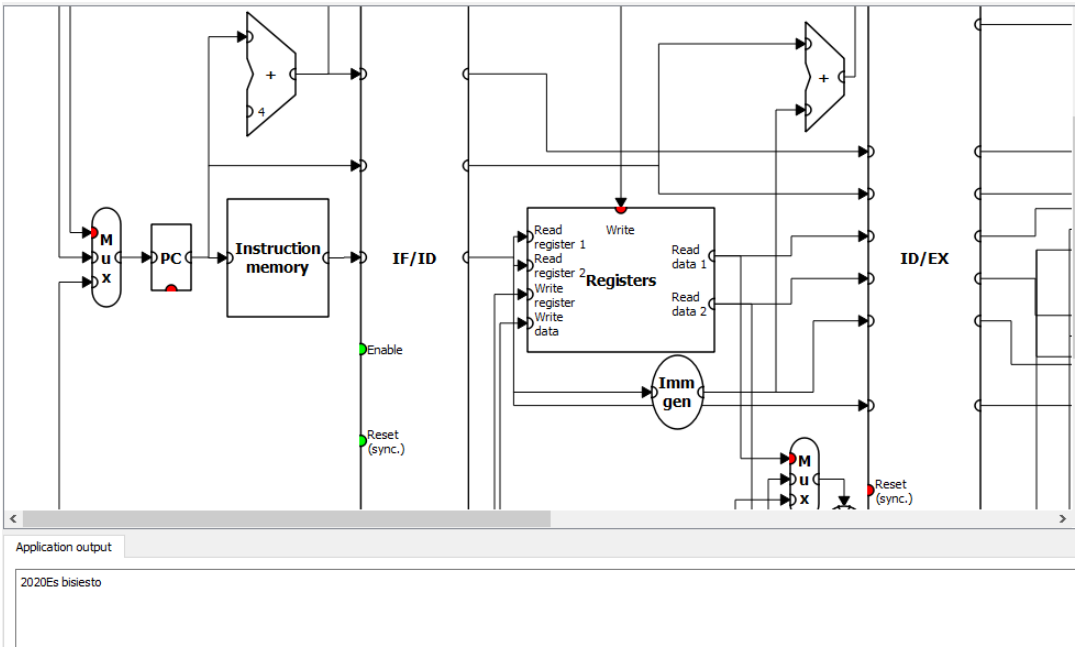
Con “end” paramos el programa.

En “print_leap_noleap” imprimimos si es o no es binario.

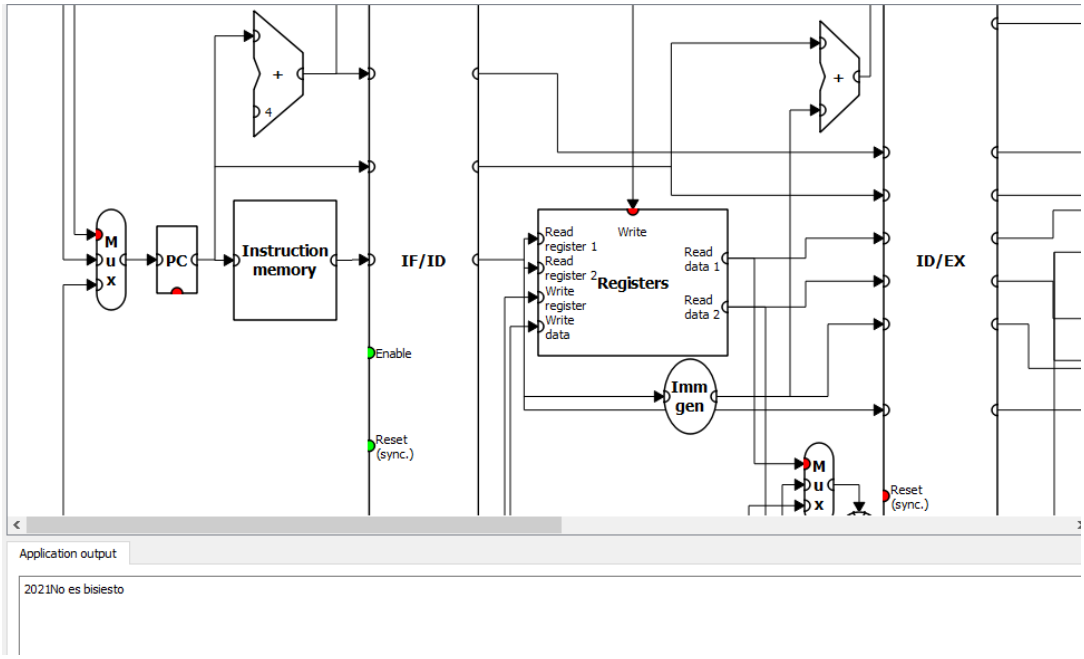
En “calc_leap” calculamos el resto de nuestro año dividido entre 4 con la instrucción “rem”. Con nuestro resto calculado, comparamos si el resto es igual a cero. Se esto se cumple, saltamos a “leap:” que moverá a2 a a1(Imprimirá “Es binario”).

En el caso de que no se cumpla la condición, movemos a3 a a1(Imprimirá “No es binario”).

Finalmente, en “print_year:” imprimimos nuestro año.



El resultado para el año 2020.



El resultado para el año 2021.

Practica 4:

Ejercicio 1:

En este ejercicio se nos pide que se imprima un array de números especificados en el ejercicio, mediante recursividad.

```
.data
space: .word 32    #Variable de tipo entero para imprimir espacios.

list:   .word 7     #Nuestro array de números.
            .word 5
            .word -4
            .word 100
            .word -250
            .word 73
            .word -50

.text

begin:
la a2 list      #Cargamos la dirección de memoria de nuestro array.
lw t0 0(a2)     #Metemeos en t0 nuestra primera palabra.

loop:
add t2, zero, zero   #Inicializamos a cero t2
mv t2, t0            #Movemos t0 a t2
lw t0 4(a2)          #Cargamos en t0 nuestra siguiente palabra.
addi a2, a2, 4       #Apuntamos a la siguiente palabra del array.
beq t2, zero, end    #Si nuestro t2 = 0, deja de imprimir.
jal print            #Hacemos doble recursividad, llamamos a print.


print:

addi a0 x0 1       #Guardamos en a0 un 1 mediante un addi.
addi a1 t2 0       #Guardamos en el número de t2 en a1 mediante un addi.
ecall              #Imprimimos el número.

la a1 space        #Imprimimos el espacio.
li a0, 4
ecall

jal loop           #Volvemos a loop.

end:
li a0, 10
ecall
```

Creamos una variable llamada “space”, con la que se imprimirá los especiados. A su vez creamos el array de números.

Guardamos en a2 la dirección de memoria del array, y cargamos la primera palabra del array en t0.

Empezamos con el bucle.

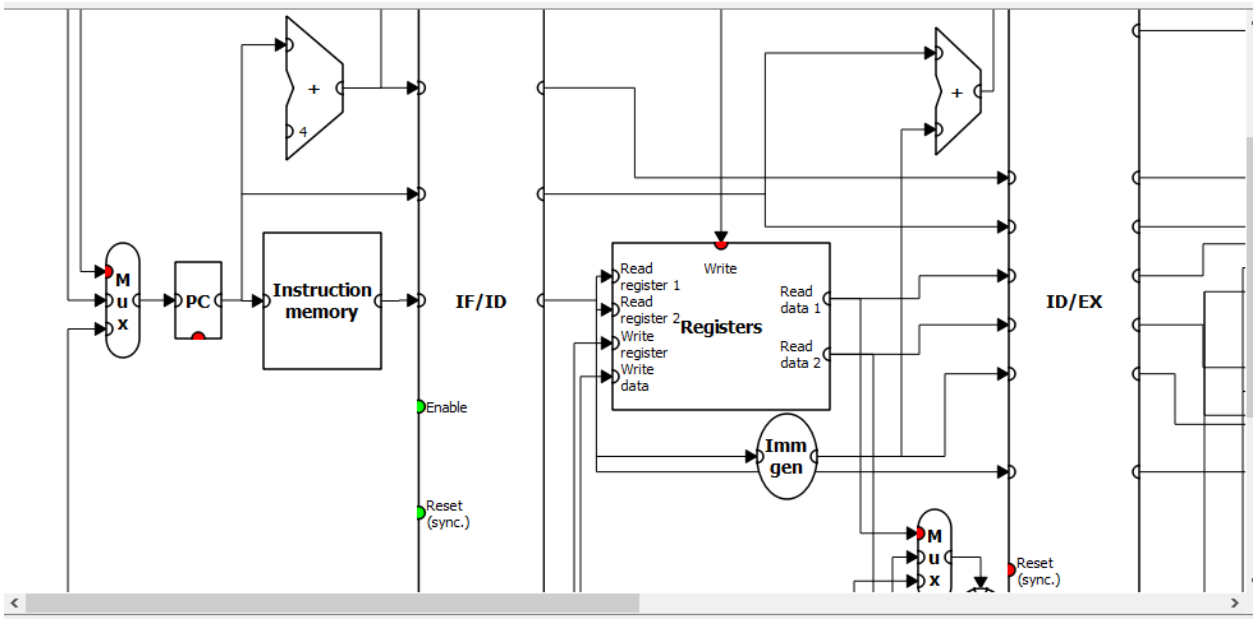
Primero inicializamos t2 a 0, nos servirá como un auxiliar donde guardaremos t0. Después en t0 cargamos la siguiente palabra del array. Con un “addi” apuntamos a la siguiente posición del array, para ir recorriendo todo el array de números. Hacemos la condición de salida.

Para la condición de salida hacemos un beq en el que si t2 es igual a 0 pare el programa. Esto se comprueba así ya que una vez impreso el –50, los siguientes números son 0 ya que no hay nada en las posiciones siguientes al último número del array.

Saltamos a” print”.

En print primero guardamos en a0 el valor 1, para imprimir enteros. Después guarden a0 el valor 1, para imprimir enteros. Después guardamos en a1 el valor de t2, para que cada vez imprima en número siguiente.

Imprimimos el espacio y llamamos a jal loop para que sea recursivo.



Ejercicio 2:

Se nos pide hacer el algoritmo de la burbuja de forma recursiva utilizando los datos del apartado anterior.

```
.data
length:     .word 7

space:  .word 32
break:  .word 10

list:   .word 7
            .word 5
            .word -4
            .word 100
            .word -250
            .word 73
            .word -50
.text
begin:
la a2 list          #Guardamos el espacio de memoria del array en a2
lw a3 length        #Guardamos en a3 el tamaño del array
jal bubbleSort      #Llamamos a bubbleSort
jal end             #Llamamos a end.

bubbleSort:
addi sp, sp -16
sw ra, 12(sp)
jal printNums               #Llamamos a printNums.
la a2 list                  #Guardamos en a2 la dirección de memoria de la list.
li t1 1                     #Guardamos en t1 el valor 1, esto nos servirá de contador.
lw a3 length                #Guardamos en a3 el tamaño del array
li t4 0                     #Inicializamos t4 a 0
jal checkPair               #Saltamos a checkPair para comprobar si un número es mayor que otro.
bgt t4, zero, bubbleSort    #Si t4 es mayor que 0 saltamos a bubbleSort
lw ra, 12(sp)
addi sp, sp, 16
ret

checkPair:
addi sp, sp -16
sw ra, 12(sp)
lw t2, 0(a2)
lw t3, 4(a2)
bgt t2, t3, swap_a          #Si t2 es mayor que t3, saltamos a swap_a para cambiarlos de posición.
addi a2, a2, 4              #Apuntamos al siguiente elemento del array
addi t1, t1, 1              #Aumentamos t1 en 1.
bgt a3, t1, checkPair       #Si a3 es mayor que t1, empezamos checkPair de nuevo.
lw ra, 12(sp)
addi sp, sp, 16
ret

swap_a:
sw t3, 0(a2)                #Cambiamos un número por el otro
sw t2, 4(a2)
addi t4, t4, 1
ret
```

```
printNums:
addi sp, sp -16
sw ra, 12(sp)
li t5 0                     #Inicializamos t5
la a2 list                  #Guardamos en a2 el espacio de memoria del array.
jal loop                    #Llamamos a loop
la a1 break                 #Imprimimos el salto de linea.
li a0, 4
ecall
lw ra, 12(sp)
addi sp, sp, 16
ret

loop:
addi sp, sp -16
sw ra, 12(sp)
lw t2 0(a2)
jal print                   #Llamamos a print.
addi a2, a2, 4              #Apuntamos al siguiente elemento del array.
addi t5, t5, 1              #Aumentamos t5 en 1
bgt a3, t5, loop            #Si a3 es mayor que t5, repetimos loop.
lw ra, 12(sp)
addi sp, sp, 16
ret

# print t2
print:
addi a0 x0 1          #Imprimimos como en el ejercicio anterior.
addi a1 t2 0
ecall
la a1 space
li a0, 4
ecall
ret

end:
li a0, 10
ecall
```

Creamos las variables correspondientes, entre las que están “length”, el tamaño del array y nuestro array.

Guardamos en a2 el espacio de memoria del array y en a3 el tamaño del array.

Llamamos a “bubbleSort” para empezar el “algorito”. Cuando finaliza “bubbleSort” saltamos a “end”.

En “bubbleSort” primero nos hacemos la pila, tras eso llamamos a printNums para que imprima los números del array. Guardamos en a2 la dirección de memoria del array, inicializamos t1 a 1, esto será el contador.

Inicializamos t4 a 0 y llamamos a” checkPair” para que compruebe si debe o no cambiar de posición los números.

Después, utilizando “bgt”, hacemos que llame a “bubbleSort” mientras t4>0.

En “checkPair” comprobamos si debemos cambiar o no los números que están siendo comparados. Utilizamos otra vez “bgt”, en este caso si t2 > t3, hacemos swap.

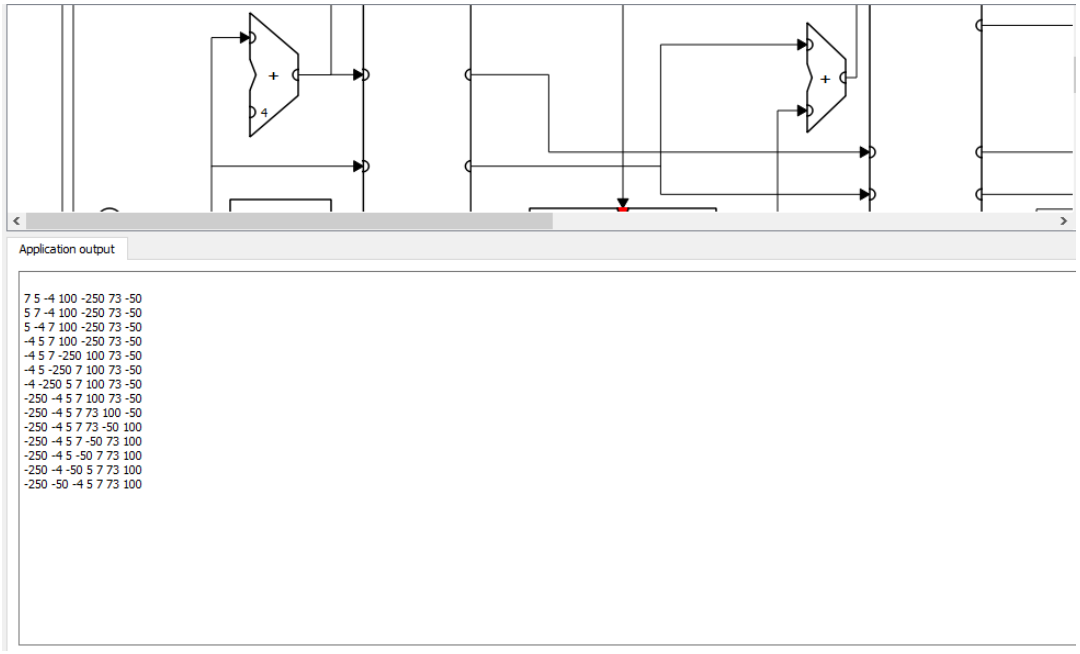
Apuntamos al siguiente elemento del array con “addi a2, a2, 4 “y aumentamos el valor de t1 en 1.

Esto es para utilizarlo otra vez en un “bgt” en el que, si nuestro tamaño del array es más grande que el contador de t1, sigua haciendo las comprobaciones.

En “swap_a” simplemente cambiamos de posición los números y aumentados t4 en 1.

Con “printNums” imprimimos el array por cada pasada completa del algoritmo.

Con “loop” y ”print” se hace una impresión de los números como en el ejercicio anterior.



El resultado. La primera línea es el original y la última es el array ya ordenado.