ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

ESTRUCTURA DE COMPUTADORES

GRADO EN INGENIERÍA INFORMÁTICA

ACTIVIDADES PRÁCTICAS

ACTIVIDAD 4.1:
TRAZAS DE PROGRAMAS EN ENSAMBLADOR

CURSO 2021-22

INTRODUCCIÓN

En esta actividad se propone la realización de una serie de trazas de programas en ensamblador de **MIPS**, útiles de cara a la preparación del examen de prácticas de la asignatura.

Las trazas se realizarán mediante el simulador **MARS**, de Peter Sanderson y Kenneth Vollmar, de la Missouri State University, y se grabarán en el fichero **trazas.xlsx**.

EJERCICIO 1: INTERPOLACIÓN LINEAL CON ENTEROS

Se pretende realizar una interpolación lineal del valor de un punto de una recta situado entre otros dos puntos dados, mediante la expresión siguiente:

$$f(x) = \frac{(x - x_1) \cdot (y_2 - y_1)}{(x_2 - x_1)} + y_1$$

donde **(x_1, y_1)** y **(x_2, y_2)** son números enteros que contienen las coordenadas cartesianas de 2 puntos, mientras que ***x*** es la abscisa de un punto intermedio situado entre los dos anteriores.

Una solución parcial al problema anterior escrita en lenguaje C es la siguiente (obviando la parte de entrada/salida por consola):

```
int x,y,x1,y1,x2,y2;
...
y = ((x-x1)*(y2-y1)) / (x2-x1) + y1;
...
```

A continuación se muestra una posible solución en lenguaje ensamblador, contenida en el fichero **interpolación_lineal.asm** (se obvian la parte de reserva de espacio para datos en memoria y la entrada/salida por consola):

```
# y = ((x-x1)*(y2-y1)) / (x2-x1) + y1;
        lw      $s0,x           # Carga de x en $s0
        lw      $s1,x1          # Carga de x1 en $s1
        lw      $s2,y1          # Carga de y1 en $s2
        lw      $s3,x2          # Carga de x2 en $s3
        lw      $s4,y2          # Carga de y2 en $s4
        sub     $t0,$s0,$s1     # t0 = x - x1
        sub     $t1,$s4,$s2     # t1 = y2 - y1
        mul     $t0,$t0,$t1     # t0 = t0 * t1
        sub     $t1,$s3,$s1     # t1 = x2 - x1
        div     $t0,$t0,$t1     # t0 = t0 / t1
        add     $t0,$t0,$s2     # t0 = t0 + y1
        sw      $t0,y           # Almacenamiento del resultado en y
```

Se pide realizar varias trazas del programa pedido. obviando la parte de E/S por consola. Incluir las trazas en el fichero **trazas.xlsx**.

Juegos de datos de entrada:

Datos	Juego 1	Juego 2
x1	2	-3
y1	0	6
x2	6	3
y2	8	14
x	4	0

EJERCICIO 2: TRIÁNGULO

Se pretende saber si tres segmentos s_1, s_2 y s_3 de longitudes dadas l_1, l_2 y l_3 pueden formar triángulo, lo cual será cierto si y sólo si se cumplen simultáneamente las siguientes desigualdades:

$$l1 < l2 + l3$$

$$l2 < l1 + l3$$

$$l3 < l1 + l2$$

A continuación se muestra una posible solución en C:

```
void entrada (void);
void salida (void);
int lon1, lon2, lon3;
register int tmp1, tmp2, tmp3; // Temporales
int triang;
int main (void) {
   entrada();
   tmp1 = lon2 + lon3;
   tmp2 = lon1 + lon3;
   tmp3 = lon1 + lon2;
   if ( (lon1 < tmp1) && (lon2 < tmp2) && (lon3 < tmp3) )
      triang = 1;
   else
      triang = 0;
   salida();
   return 0;
}
```

En el fichero **triang.asm** se proporciona un esqueleto de la solución.

Se pide:

a) Programar la solución completando el código del fichero.

b) Realizar varias trazas de ejecución del programa, obviando la entrada/salida por consola. Incluir las trazas en el fichero **trazas.xlsx**.

Juegos de datos de entrada para realizar las trazas:

Datos	Juego 1	Juego 2
l1	2	3
l2	3	4
l3	6	5

EJERCICIO 3: SERIE DE FIBONACCI

Se pretende calcular el término n-ésimo de la serie de Fibonacci siendo n>1. La fórmula de recurrencia para calcularlo es la siguiente:

$$f_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f_{n-1} + f_{n-2} & n > 1 \end{cases}$$

A continuación se muestra una posible solución en C:

```
void entrada (void);
void salida (void);
int n, f;
register int f1, f2;
register int i;
int main (void) {
   entrada();
   f2 = 0;
   f1 = 1;
   for (i = 2; i <= n; i++) {
      f = f1 + f2;
      f2 = f1;
      f1 = f;
   }
   salida();
   return 0;
}
```

En el fichero **fibonacci.asm** se proporciona un esqueleto de la solución.

Se pide:

a) Programar la solución completando el código del fichero.

b) Realizar varias trazas de ejecución del programa, escribiendo los valores de los registros involucrados únicamente en los siguientes puntos:

   1. Justo antes de entrar en el bucle (una única vez).
   2. En la ramificación condicional correspondiente a la evaluación de la condición de continuación en el bucle (una vez en cada iteración).
   3. Tras ejecutar dicha ramificación condicional indicada en el paso 2 (una vez en cada iteración).
   4. En la instrucción siguiente al bucle (una única vez).

Se pide realizar una traza del programa pedido. obviando la parte de E/S por consola. Incluir la traza en el fichero **trazas.xlsx**.

Juego de datos de entrada para realizar la traza:

Datos	Juego 1
n	5

EJERCICIO 4: TÉRMINO N-ÉSIMO DE UNA SERIE

Se pretende calcular el término n-ésimo de una serie que tiene la siguiente fórmula de recurrencia:

$$T(i) = \begin{cases} 1 & i = 0 \\ 2 \cdot T(i-1) & i\ impar \\ 2 \cdot T(i-1) - 1 & i\ par \end{cases}$$

A continuación se muestra una posible solución en C:

```
void entrada (void);
void salida (void);
int n, tn;
register int i;
int main (void) {
   entrada();
   tn = 1;
   for (i = 1; i <= n; i++) {
      tn = 2 * tn;
      if (i % 2 == 0)
         tn = tn - 1;
   }
   salida();
   return 0;
}
```

En el fichero **termino_serie.asm** se proporciona un esqueleto de la solución. Se pide:

a) Programar la solución completando el código del fichero.

b) Realizar varias trazas de ejecución del programa, escribiendo los valores de los registros involucrados únicamente en los siguientes puntos:

   1. Justo antes de entrar en el bucle (una única vez).
   2. En la ramificación condicional correspondiente a la evaluación de la condición del if (una vez en cada iteración).
   3. En la instrucción siguiente a la indicada en el paso 2 (una vez en cada iteración).
   4. En la instrucción siguiente al bucle (una única vez).

Se pide realizar varias trazas del programa pedido. obviando la parte de E/S por consola. Incluir las trazas en el fichero **trazas.xlsx**.

Juegos de datos de entrada para realizar las trazas:

Datos	Juego 1	Juego 2
n	4	7

EJERCICIO 5: NÚMEROS PERFECTOS

Un número natural es perfecto si la suma de sus divisores propios (distintos de él mismo) es igual a sí mismo. Por ejemplo, el número 6 es perfecto, ya que sus divisores propios son 1, 2 y 3, que sumados dan 1+2+3=6.

A continuación se muestra una posible solución en C a un programa que calcula si un número natural de 32 bits es perfecto o no:

```
void entrada (void);
void salida (void);
int n, perfecto;
register int i, suma;
int main (void) {
   entrada();
   suma = 1;
   i = 2;
   while (i <= n/2) {
      if (n % i == 0)
         suma = suma + i;
         i++;
   }
   perfecto = (n == suma);
   salida();
   return 0;
}
```

En el fichero **perfecto.asm** se proporciona un esqueleto de la solución.

Se pide:

a) Programar la solución completando el código del fichero.

b) Realizar varias trazas de ejecución del programa, escribiendo los valores de los registros involucrados únicamente en los siguientes puntos:
   1. Justo antes de entrar en el bucle (una única vez).
   2. En la ramificación condicional correspondiente a la evaluación de la condición del if (una vez en cada iteración).
   3. En la instrucción siguiente a la indicada en el paso 2 (una vez en cada iteración).
   4. En la instrucción siguiente al bucle (una única vez).

Se pide realizar varias trazas del programa pedido. obviando la parte de E/S por consola. Incluir las trazas en el fichero **trazas.xlsx**.

Juegos de datos de entrada para realizar las trazas:

Datos	Juego 1	Juego 2
n	6	15