HOJA DE PROBLEMAS DEL TEMA 4.
PROGRAMACIÓN EN ENSAMBLADOR DE MIPS
IMPLEMENTACIÓN DE SENTENCIAS DE CONTROL

---

1. Realizar dos versiones de un programa en ensamblador que, dados dos números enteros, calcule cuál de ellos contiene el valor máximo. La primera se basará en el siguiente código en C:

```
int x, y, maximo;
int main (void) {
      maximo = x;
      if (maximo < y)
            maximo = y;
      return 0;
}
```

La segunda versión se basará en el siguiente código en C:

```
int x, y, maximo;
int main (void) {
      if (x >= y)
            maximo = x;
      else
            maximo = y;
      return 0;
}
```

En ambas soluciones las variables ***x***, ***y*** y ***maximo*** existirán en la solución con el mismo nombre, y tendrán un valor inicial desconocido. Para realizar las pruebas se introducirán manualmente los valores necesarios en las variables. Escribir las soluciones en los ficheros **maximo1.asm** y **maximo2.asm**.

---

2. Tres segmentos s_1, s_2 y s_3 de longitudes dadas l_1, l_2 y l_3 pueden formar triángulo si y sólo si se cumplen simultáneamente las siguientes desigualdades:

$$l_1 < l_2 + l_3 \qquad l_2 < l_1 + l_3 \qquad l_3 < l_1 + l_2$$

Diseñar un programa ensamblador de MIPS que compruebe si tres segmentos de longitud conocida pueden formar triángulo. Se puede partir de la siguiente versión en C:

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

Las variables ***lon1***, ***lon2***, ***lon3*** y ***triang*** deben existir en la solución con el mismo nombre, y tendrán un valor inicial desconocido. Para realizar las pruebas se introducirán manualmente los valores necesarios en las variables. Escribir la solución en un fichero llamado **triangulo.asm**.

---

3. Realizar un programa en ensamblador de MIPS que calcule el término enésimo de la serie de Fibonacci. Esta serie se define del siguiente modo:

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

Se puede partir de la siguiente versión en C:

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

Las variables ***n*** y ***f*** deben existir en la solución con el mismo nombre, y tendrán un valor inicial desconocido. Para realizar las pruebas se introducirán manualmente los valores necesarios en las variables. Escribir la solución en un fichero llamado **fibonacci.asm**.

---

4. Se pretende diseñar un programa en ensamblador de MIPS que, dado un entero que se supone positivo, inspeccione los bits que lo componen y contabilice cuántos de entre ellos se encuentran a 1. Se parte de la siguiente versión del programa en C:

```
int dato, unos;
register int tmp, mascara;
int main (void) {
      unos = 0;
      tmp = dato;
      mascara = 0x80000000;
      while (tmp != 0) {
            if (tmp & mascara != 0)
                  unos = unos + 1;
      tmp = tmp << 1;}
      return 0;
}
```

Las variables ***dato*** y ***unos*** deben existir en la solución con el mismo nombre, y tendrán un valor inicial desconocido. Para realizar las pruebas se introducirán manualmente los valores necesarios en las variables. Escribir la solución en un fichero llamado **unos.asm**.

---

5. Realizar un programa en ensamblador que calcule el término T(n) de la sucesión definida por:

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

Se puede partir de la siguiente versión en C:

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

Las variables ***n*** y ***tn*** deben existir en la solución con el mismo nombre, y tendrán un valor inicial desconocido. Para realizar las pruebas se introducirán manualmente los valores necesarios en las variables. Escribir la solución en un fichero llamado **termino_serie.asm**.

---

6. 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.

Realizar un programa en ensamblador que, dado un dato de tipo entero positivo (32 bits), indique si se trata de un número perfecto. Se puede partir de la siguiente solución en C:

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

Las variables ***n*** y ***perfecto*** deben existir en la solución con el mismo nombre, y tendrán un valor inicial desconocido. Para realizar las pruebas se introducirán manualmente los valores necesarios en las variables. Escribir la solución en un fichero llamado **perfecto.asm**.

---

7. Diseñar un programa en ensamblador de MIPS que calcule el máximo común divisor y el mínimo común múltiplo de dos números mediante el algoritmo de Euclides. Se puede partir de la siguiente versión en C:

```
int x, y, mcd, mcm;
register int tmp, resto;
int main(void) {
      tmp = x;
      mcd = y;
      do {
            resto = tmp % mcd;
            if (resto <> 0) {
                  tmp = mcd;
                  mcd = resto;
            }
      } while (resto != 0);
      mcm = x * y / mcd;
      return 0;
}
```

Las variables ***x***, ***y***, ***mcd*** y ***mcm*** deben existir en la solución con el mismo nombre, y tendrán un valor inicial desconocido. Para realizar las pruebas se introducirán manualmente los valores necesarios en las variables. Escribir la solución en un fichero llamado **mcd_mcm.asm**.
