HOJA DE PROBLEMAS DEL TEMA 4.
PROGRAMACIÓN EN ENSAMBLADOR DE MIPS
VECTORES DE NÚMEROS ENTEROS

---

1. Realizar un programa en ensamblador de MIPS que copie un vector de 16 números enteros de 32 bits en otro vector de igual tamaño. Se puede partir de la siguiente versión en C:

```
#define N 16
int x[N], y[N];
register int i;
int main (void) {
      for (i = 0; i < N; i++)
            y[i] = x[i];
      return 0;
}
```

En la solución los vectores ***x*** e ***y*** existirán con el mismo nombre, y tendrán un contenido inicial desconocido. Excepcionalmente, para realizar las pruebas se podrá dar un contenido inicial conocido a los elementos del vector ***x***. Escribir la solución en el fichero **vector_int_copia.asm**.

---

2. Realizar un programa en ensamblador de MIPS que sume dos vectores de 10 números enteros con signo de 32 bits. El vector suma se guardará en el primer vector, destruyendo su valor previo. Se puede partir de la siguiente versión en C:

```
int v1[10], v2[10];
register int i;
int main (void) {
      i = 0;
      do {
            v1[i] = v1[i] + v2[i];
            i = i + 1;
      } while (i <> 10);
      return 0;
}
```

En la solución los vectores ***v1*** y ***v2*** existirán con el mismo nombre, y tendrán un contenido inicial desconocido. Excepcionalmente, para realizar las pruebas se podrá dar un contenido inicial conocido a los elementos del vector ***v1***. Escribir la solución en el fichero **vector_int_suma.asm**.

---

3. Diseñar un programa en ensamblador de MIPS que calcule los valores máximo y mínimo de un vector de enteros. Se puede partir del siguiente código en C:

```
#define N 16
int v[N], maximo, minimo;
register int i, aux;
int main (void) {
      maximo = v[0];
      minimo = maximo;
      for (i = 1; i < N; i++) {
            aux = v[i];
            if (aux < minimo)
                  minimo = aux;
            else if (aux > maximo)
                  maximo = aux;
      }
      return 0;
}
```

En la solución el vector ***v*** y las variables ***maximo*** y ***minimo*** existirán con el mismo nombre, y tendrán un contenido inicial desconocido. Excepcionalmente, para realizar las pruebas se podrá dar un contenido inicial conocido a los elementos del vector ***v***. Escribir la solución en el fichero **vector_int_minmax.asm**.

---

4. Diseñar un fragmento de código en ensamblador de MIPS que, dado un vector de 16 números enteros, calcule la suma de los elementos con valor par. Se puede partir de la siguiente versión en C:

```
#define N 16
int vector[N], suma;
register int i, temp;
int main (void) {
     suma = 0;
     i = 0;
     while (i < N) {
            temp = vector[i];
            if (temp % 2 == 0)
                  suma = suma + temp;
            i++;
      }
      return 0;
}
```

En la solución las variables **vector** y **suma** existirán con el mismo nombre, y tendrán un contenido inicial desconocido. Excepcionalmente, para realizar las pruebas se podrá dar un contenido inicial conocido a los elementos de **vector**. Escribir la solución en el fichero **vector_int_suma_pares.asm**.

---

5. Diseñar un fragmento de código en ensamblador de MIPS que, dados dos vectores de 16 elementos, copie los elementos de x en y en orden inverso. Se puede partir de la siguiente versión en C:

```
#define N 16
int x[N], y[N];
register int i, j;
int main (void) {
     suma = 0;
    i = 0;
    j = N-1;
    while (i < N) {
          y[j] = x[i];
          i++;
          j--;
    }
    return 0;
}
```

En la solución los vectores **x** e **y** existirán con el mismo nombre, y tendrán un contenido inicial desconocido. Excepcionalmente, para realizar las pruebas se podrá dar un contenido inicial conocido a los elementos del vector **x**. Escribir la solución en el fichero **vector_int_copia_inv.asm**.

---

6. Diseñar un fragmento de código en ensamblador de MIPS que, dado un vector de 16 enteros, genere otro vector Y donde el elemento i-ésimo del vector Y sea la suma de todos los elementos del vector X con índice menor o igual que i. Es decir: Y[0]= X[0]; Y[1]=X[0]+X[1]; Y[2]=X[0]+X[1]+X[2] y así sucesivamente. Se puede partir de la siguiente versión en C:

```
#define N 16
int X[N], Y[N];
register int i;
int main (void) {
     Y[0] = X[0];
     i = 1;
     while (i < N)
          Y[i] = Y[i-1] + X[i];
     return 0;
}
```

En la solución los vectores **X** e **Y** existirán con el mismo nombre, y tendrán un contenido inicial desconocido. Excepcionalmente, para realizar las pruebas se podrá dar un contenido inicial conocido a los elementos del vector **X**. Escribir la solución en el fichero **vector_int_suma_acum.asm**.

---

7. Diseñar un fragmento de código en ensamblador de MIPS que, dados tres vectores de números enteros (**X**, **Y** y **Z**), copie en cada elemento del vector **Z** el valor mínimo entre los elementos del mismo índice de los vectores **X** e **Y**. Es decir,

Z[i] = mínimo(X[i],Y[i])

Se puede partir de la siguiente versión en C:

```
#define N 16
int X[N], Y[N], Z[N];
register int i, temp1, temp2;
int main (void) {
      i = 0;
      while (i < N) {
            temp1 = X[i];
            temp2 = Y[i];
            if (temp1 < temp2)
                  Z[i]=temp1;
            else
                  Z[i]=temp2;
            i++;
      }
      return 0;
}
```

En la solución los vectores **X**, **Y** y **Z,** existirán con el mismo nombre, y tendrán un contenido inicial desconocido. Excepcionalmente, para realizar las pruebas se podrá dar un contenido inicial conocido a los elementos de los vectores **X** e **Y**. Escribir la solución en el fichero **vector_int_copia_min.asm**.
