PROBLEMAS TEMA 2

ACCESO A MEMORIA

Algunos de los siguientes problemas consisten en traducir lenguaje C a ensamblador MIPS. Se recomienda emplear el siguiente convenio sobre uso de los registros:

- Cuando usemos registros para guardar el contenido de una variable o bien un puntero, usaremos los registros `$s0` a `$s7`.
- Cuando usemos registros para cálculos intermedios, emplearemos los registros `$t0` a `$t7`.

Problema 1.- (Ejemplo en Patterson 4ª ed., capítulo 2.3) Traducir la siguiente sentencia de lenguaje C a ensamblador MIPS:

```
A[12] = h + A[8];
```

Deben resultar **3 instrucciones** de código ensamblador.

Supóngase que:

- El contenido de la variable ***h*** está ya almacenado en `$s2`.
- La dirección base de la matriz ***A*** está ya almacenada en `$s3`.
- Los datos de ***A*** son de tamaño `word`.

Pistas/consejos:

- Es importante elegir bien el modo de direccionamiento que se emplea para acceder a memoria. Elegir un modo que no sea el más indicado puede llevar a necesitar más instrucciones para hacer el mismo trabajo. **En este problema NO se permite usar etiquetas**, se debe acceder a la memoria empleando las direcciones que ya están guardadas en registro.
- También es importante el orden de las operaciones. Es fácil establecer el orden correcto si se tiene en cuenta que para realizar cada operación es necesario haber calculado antes sus operandos.

Problema 2.- Para hacer más legible el código del problema anterior, suponer que esta vez sí tenemos definida una etiqueta A que indica la dirección base de la matriz `A`. Cambiar el modo de direccionamiento de los accesos a memoria, y emplear uno que haga uso de la etiqueta `A`.

Problema 3.- Traducir los siguientes fragmentos de código C a ensamblador MIPS:

```
f = g - A[B[4]];
```

```
f = A[B[g]+1];
```

```
f = A[g] - (B[g-1] + B[g+1]);
```

Supóngase que:

- Las variables `f` y `g` se han asignado a los registros `$s0` y `$s1`.
- Los punteros a las matrices `A` y `B` se han asignado a los registros `$s6` y `$s7`.

Problema 4.- Traducir el siguiente fragmento de código C a ensamblador MIPS:

```
f = A[B[g]+1];
```

- Las variables `f` y `g` se han asignado a los registros `$s0` y `$s1`.
- Los punteros a las matrices `A` y `B` se han asignado a los registros `$s6` y `$s7`.

BUCLES

Problema 5.- El siguiente fragmento de código C calcula el sumatorio de números naturales hasta un cierto N. Traducir a ensamblador MIPS:

```
int i=1;
int j=0;
int N=1000;
do {
       j = j + i;
       i++;
} while(i <= N);
```

- Las variables `i`, `j` y `N` se deberán almacenar en los registros `$s0`, `$s1` y `$s7`.
- Prestar especial atención a las inicializaciones de variables.
- Para asegurarse de si los límites del bucle están bien programados, indicar qué valor debe tener la variable `i` antes y después de ejecutar la última iteración del bucle. Verificar así que las condiciones están bien ajustadas.
- Ejecutar en MARS y comprobar que el resultado es el esperado.

Problema 6.- En el código del problema anterior, ¿qué ocurriría si N==0?
Para evitar errores como el que se produce en ese caso, cambiar el bucle `do-while` por un bucle `while`.

Ojo a la estructura del bucle while en ensamblador:

- Debe tener al final del bucle un salto **in**condicional al principio.
- La primera instrucción del bucle debe ser un salto condicional hacia fuera del bucle (*condición de salida*).

Problema 7.- Escribir en ensamblador MIPS un programa que utilice una llamada al sistema (*syscall*) para pedir números al usuario. Después de 10 iteraciones, el programa debe imprimir por pantalla la suma de los números introducidos.

- Emplear la hoja de referencia o la ayuda de MARS para localizar qué funciones del sistema se deben utilizar.
- Si se van a imprimir cadenas, declararlas como `.asciiz`

EJECUCIÓN CONDICIONAL

Problema 8.- Modificar el programa anterior: si el usuario introduce un cero, salir del bucle y presentar por pantalla la suma de los números introducidos hasta el momento.

Problema 9.- Escribir un programa en el que se declaran dos vectores `A` y `B`, cada uno con 10 elementos numéricos tamaño `word`, que serán los números del 1 al 10 en desorden (los elementos de A deben estar en un orden diferente de los de B). Se reserva también espacio para un vector `C`, también de 10 elementos de tamaño palabra.

El programa debe almacenar en cada posición de `C` el elemento que sea mayor entre los correspondientes de `A` y `B`. Es decir:

```
Para i desde 1 hasta 10:
  C[i] = max(A[i], B[i])
```

- Para acceder a datos en memoria, utilizar un índice que se vaya incrementando de uno en uno. Para calcular el desplazamiento correspondiente a cada índice se puede utilizar cualquiera de las variantes conocidas.

Problema 10.- Modificar el programa anterior: en lugar de un índice utilizar un puntero a A, otro a B y otro a C. A cada iteración se deberán incrementar los punteros en tantas unidades como sea necesario para apuntar al siguiente elemento.

Problema 11.- *Coming soon...*