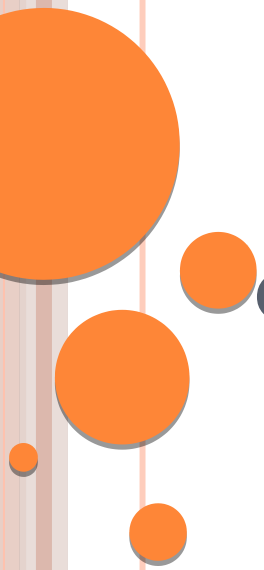

CONSTRUYENDO PROGRAMAS EN MIPS

Principio de Computadores

Tutoría Académica nº 4

INTRODUCCIÓN: PROGRAMANDO EN ENSAMBLADOR

- En las tutorías académicas 2 y 3 se introdujo los elementos básicos de programación en MIPS.
  - Cómo representar los datos en los registros.
  - Las principales instrucciones con enteros
  - Cómo realizar en ensamblador las estructuras de control comunes de los programas: secuenciales, alternativas y iterativas
  - Como manejar datos en punto flotante con el coprocesador
  - Cómo realizar llamadas al sistema

- En esta tutoría académica te presentamos una estrategia para abordar los problemas de programación en MIPS, elaborando previamente el pseudocódigo en un lenguaje de alto nivel (C, C++).
  - Para ello te presentaremos un ejemplo práctico.

PROBLEMA: PROGRAMA PARA REPASAR LAS TABLAS DE MULTIPLICAR

Se desea realizar un programa para repasar las tablas de multiplicar.

El usuario deberá introducir por teclado la tabla de multiplicar que desea comprobar. Si introduce un cero el programa deberá finalizar, y en caso contrario el programa preguntará al usuario por todas las multiplicaciones del número introducido por los número del 1 al 10 (su tabla de multiplicar). Al final de cada tabla se visualizará el porcentaje de aciertos.

PROBLEMA: PROGRAMA PARA REPASAR LAS TABLAS DE MULTIPLICAR

- No es competencia de esta asignatura el desarrollo de algoritmos por lo que no entraremos a explicar técnicas de programación. Se entiende que tienes los conocimientos para plantear un pseudocódigo, diagrama de flujo o programa en un lenguaje de alto nivel para resolver un problema de complejidad similar al planteado.
- Para este caso el programa deberá realizar las siguientes acciones:

1. Preguntar por el número cuya tabla quieres repasar
2. Preguntar por el resultado de la multiplicación de ese número por todos los número comprendidos entre 1 y 10.
3. Contar el número de aciertos.
4. Escribir el porcentaje de aciertos.

- Parece evidente que los pasos 2 y 3 se deberán realizar en un bucle que itere entre 1 y 10.
- Como probablemente hayas deducido estos 4 pasos deberán realizarse en un bucle hasta que el número introducido sea un cero (tal y como se planteó el problema).

CONSTRUYE EL PSEUDOCÓDIGO Y SOLUCIÓN EN LENGUAJE DE ALTO NIVEL

```cpp
/*
 * File:   repasotablas.cpp
 * Authores: Carlos y Alberto
 *
 * Descripción: El usuario introducira por teclado la tabla de multiplicar que
 * quiere repasar. Si introduce un 0 el programa finalizara. Si introduce un
 * numero mayor que 0 debera pregutar al usuario por todas las multiplicaciones
 * del numero introducido por lo numero del 1 al 10. Al final de cada tabla se
 * visualizara el porcentaje de aciertos.
 */
#include <iostream>

int main(int argc, char** argv) {

  std::cout << "Programa para repasar las tablas de multiplicar.\n";
  int n; // numero del que quiero repasar la tabla
  do {
    std::cout<<"¿Qué tabla deseas repasar? Introduce un número (0 para salir): ";
    std::cin >> n;
    if (n == 0) break; // si introduce un 0 sale del bucle
    int aciertos = 0; // incializamos el numero de aciertos
    for (int i = 1; i <= 10; i++) {
      std::cout << i << " x " << n << " ? ";
      int resultado;
      std::cin >> resultado;
      if (resultado == (i * n)) {
        aciertos++;
      }
    }
    int porcentaje = aciertos * 10;
    std::cout << "Tu porcenaje de aciertos es del " << porcentaje << "%\n";
  } while (n != 0);
  std::cout << "Termina el programa.\n";
}
```

CONSTRUYE EL PSEUDOCÓDIGO Y SOLUCIÓN EN LENGUAJE DE ALTO NIVEL

- Teniendo la solución en lenguaje de alto nivel nos permitirá ejecutarlo y comprobar que el algoritmo funciona como se desea.

```
matrix:EscritorioESIT/TA4$ g++ -Wall -o repasotablas repasotablas.cpp
matrix:EscritorioESIT/TA4$ ./repasotablas
Programa para repasar las tablas de multiplicar.
¿Qué tabla deseas repasar? Introduce un número (0 para salir): 0
Termina el programa.
matrix:EscritorioESIT/TA4$ ./repasotablas
Programa para repasar las tablas de multiplicar.
¿Qué tabla deseas repasar? Introduce un número (0 para salir): 3
1 x 3 ? 3
2 x 3 ? 6
3 x 3 ? 4
4 x 3 ? 5
5 x 3 ? 15
6 x 3 ? 20
7 x 3 ? 20
8 x 3 ? 4
9 x 3 ? 1
10 x 3 ? 30
Tu porcenaje de aciertos es del 40%
¿Qué tabla deseas repasar? Introduce un número (0 para salir): 0
Termina el programa.
matrix:EscritorioESIT/TA4$
```

- A partir del este código podemos hacer la traducción a lenguaje ensamblador, de manera sistemática, siguiendo los pasos que se indican en las siguientes transparencias.

LOS DATOS …

- Como seguro ya sabes los programas son la combinación de algoritmos y datos.
- Lo primero que vamos a hacer es estudiar cómo podemos representar los datos.
- En nuestro caso todos los datos son enteros, por lo que podremos usar registros de la CPU para almacenar las variables del programa
  - en caso que fuera necesario también podríamos utilizar registros flotantes o zonas de memoria
- Debemos elegir entre registros salvados o temporales. Debemos preguntarnos *¿Durante la vida de la variable hay llamadas a funciones o al sistema (*`syscall`*)?*:
  - SI → usar registros salvado (`$s#`) para la variable
  - NO → usar registro temporal (`$t#`) o salvado (`$s#`) para la variable.
- En los comentarios **debe aparecer** la asignación de variables a registros. Mejor en forma de tabla.

LAS CADENAS DE CARACTERES …

- Como se vio en la TA3, es la función 4 del `syscall` la que nos va a permitir imprimir cadenas de caracteres por la consola. Esta función recibe (a través del registro `$a0`) la dirección de memoria donde comienza la cadena, que debe estar terminada por un 0.
- Por tanto, las cadenas de caracteres que queramos imprimir deberán estar definidas en la zona `.data` con la directiva `.asciiz` y tener asignada una etiqueta.
- Antes de empezar a codificar reuniremos todas las cadenas de caracteres de nuestro programa y las definiremos en la zona `.data` , poniendo una etiqueta con nombre adecuado.

```
        .data
strTitulo:      .asciiz "Programa para repasar las tablas de multiplicar.\n"
strQueTabla:    .asciiz "¿Qué tabla deseas repasar? Introduce un número (0 para salir): "
strX:           .asciiz " x "
strInterroga:   .asciiz " ? "
strTuPorcen:    .asciiz "Tu porcenaje de aciertos es del "
strPorcent:     .asciiz "%\n"
strTermina:     .asciiz "Termina el programa.\n"
```

AHORA LAS SENTENCIAS ….

- Con lo anterior definido, iremos traduciendo:
  - Cada sentencia de alto nivel → en una o varias instrucciones en ensamblador
  - Cada estructura de control → en su equivalente en etiquetas, y saltos condicionales o saltos incondicionales.
  - Las instrucciones de alto nivel son el mejor comentario de lo que hace cada conjunto de instrucciones.
- Este procedimiento se puede (se debe) hacer de manera sistemática, sin necesidad de improvisar.
- No es conveniente hacer toda la traducción de golpe. Es preferible ir probando nuestro código que cada vez hayamos traducido unas pocas sentencias o estructuras.

¿TE ATREVES A MÁS?

- Cuando llegues a casa teclea el programa y pruébalo en el QtSPIM
- Ejecútalo paso a paso para que veas la evolución
- Modifica el programa: haz primero el programa en lenguaje de alto nivel hasta que tengas más práctica.
- Cambio propuesto:
  - Haz que el usuario, en el caso de que falle, tenga la oportunidad de volver a introducir otro solución a la multiplicación, antes de contabilizarla como fallo.