UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Organización y Arquitectura de Computadoras
Practica 6

PRESENTA

Carlos Emilio Castañon Maldonado 319053315
Pablo César Navarro Santana 317333091

PROFESOR

José de Jesús Galaviz Casas

AYUDANTES

Ricardo Enrique Pérez Villanueva
Sara Doris Montes Incin
María Ximena Lezama Hérnandez

Organización y Arquitectura de Computadoras

Practica 6

Preguntas

1 **En el ejercicio 5:**

- **¿A que valor tiende la serie?**
  Tiende al valor de π, entre mas iteraciones tiene el programa, mas nos acercamos al valor real de π.
- **¿Cuantos dígitos de la constante se pueden calcular con precisión sencilla?, es decir, considerando que existe precisión sencilla y precisión doble, ¿A cuantos dígitos estaría limitado nuestro resultado con precisión sencilla? Justifica tu respuesta.**
  El número de dígitos que se pueden calcular con precisión sencilla está limitado a 7 u 8 dígitos significativos, esto se debe a que la precisión sencilla de punto flotante utiliza 32 bits, que se divide en 1 bit para el signo, 8 bits para el exponente y 23 bits para la mantisa, como resultado, la precisión sencilla puede representar números con una precisión de aproximadamente 7 dígitos significativos.
- **¿Cuantas iteraciones son necesarias para calcular el mayor numero de dígitos?**
  Sea m las iteraciones, el numero de iteraciones para conseguir el mayor numero de dígitos es aproximadamente $m = 1000000$ ya que el programa en $MIPS$ nos arroja un valor de $\pi = 3.1415973$ y de intentarlo con un 0 mas, el programa reacciona de la siguiente manera con $m = 10000000 \quad \pi = 3.141597$ esto se debe a que el programa se quedo sin donde almacenar esa cantidad de datos por lo descrito anteriormente.

  Ahora, si hablamos del codigo en C descrito al final de este documento, la cantidad esta limitada a los recursos de nuestra computadora.

2 **Menciona 5 llamadas a sistema (syscall) que puedes usar en MIPS. Menciona su código de instrucción y que es lo que hace.**

- **syscall** 1: print integer (imprime un número entero)
  Esta llamada al sistema permite imprimir en la consola un número entero almacenado en un registro. El número entero se pasa en el registro $\$a0$.
- **syscall** 4: print string (imprime una cadena de caracteres)
  Esta llamada al sistema permite imprimir en la consola una cadena de caracteres almacenada en una dirección de memoria. La dirección de memoria se pasa en el registro $\$a0$.
- **syscall** 5: read integer (lee un número entero)
  Esta llamada al sistema permite leer un número entero desde la consola y almacenarlo en un registro. El número entero se pasa en el registro $\$v0$.
- **syscall** 8: read string (lee una cadena de caracteres)
  Esta llamada al sistema permite leer una cadena de caracteres desde la consola y almacenarla en una dirección de memoria. La dirección de memoria se pasa en el registro $\$a0$ y la longitud de la cadena se pasa en el registro $\$a1$.
- **syscall** 10: exit (salida del programa)
  Esta llamada al sistema permite finalizar la ejecución del programa. El código de salida del programa se pasa en el registro $\$a0$.

3 **¿Cuales son los 3 tipos principales de instrucciones? Menciona que comportamiento tiene cada tipo de instrucción.**

- **Instrucciones de carga/almacenamiento (load/store):**
  Estas instrucciones se utilizan para mover datos entre registros y memoria.
  Las instrucciones de carga mueven datos de la memoria a un registro, mientras que las instrucciones de almacenamiento mueven datos de un registro a la memoria.
- **Instrucciones aritméticas y lógicas (arithmetic/logical):**
  Estas instrucciones realizan operaciones matemáticas y lógicas en los datos.
  Las instrucciones aritméticas realizan operaciones como suma, resta, multiplicación y división, mientras que las instrucciones lógicas realizan operaciones como AND, OR, NOT y XOR.
- **Instrucciones de salto y de control (jump/control):**
  Estas instrucciones controlan el flujo de ejecución del programa.
  Las instrucciones de salto permiten saltar a una instrucción específica en el programa, mientras que las instrucciones de control permiten realizar operaciones como la comparación de valores y la toma de decisiones condicionales.

4 **¿Que hacen las instrucciones de tipo FR y de tipo FI? Da algunos ejemplos de instrucciones de este tipo y menciona porque están separadas de las otras 3 principales.**
Las instrucciones de tipo FR (floating-point register) y de tipo FI (floating-point immediate) son instrucciones que se utilizan específicamente para operar con números con el formato de punto flotante.

Las instrucciones de tipo FR operan con números en formato de punto flotante almacenados en registros de punto flotante (los registros $\$f0$ a $\$f31$ en MIPS), mientras que las instrucciones de tipo FI operan en números en formato de punto flotante almacenados en registros de propósito general (los registros $\$t0$ a $\$t9$ y $\$s0$ a $\$s7$ en MIPS), mediante la carga de una constante de punto flotante en la instrucción.

Estas instrucciones están separadas de las otras tres categorías principales (carga/almacenamiento, aritméticas/lógicas, y salto/control) *porque operan específicamente en números de punto flotante*, los cuales requieren un procesamiento diferente al de por ejemplo los números enteros, las instrucciones de punto flotante generalmente requieren más ciclos de reloj que las instrucciones que operan con números enteros, lo que puede tener un impacto en el rendimiento de la CPU.

Algunos ejemplos de instrucciones FR son:

$add.s\ f2, f4, f6$ # Suma los registros $f4$ y $f6$, y almacena el resultado en $f2$.
$mul.s\ f10, f12, f14$ # Multiplica los registros $f12$ y $f14$, y almacena el resultado en $f10$
$c.eq.s\ f8, f10$ # Compara los registros $f4$ y $f6$, y establece la bandera de igualdad

Algunos ejemplos de instrucciones FI son:

$l.s\ \$f2, 24(\$sp)$ # Carga el punto flotante en la posición de memoria $(\$sp + 24)$ en $f2$.
$addi\ \$t0, \$t0, 5.5$ # Suma 5.5 al valor de $\$t0$ y el resultado termina en $\$t0$.
$li.s\ \$f10, 3.1416$ # Carga la constante 3.1416 en el registro de punto flotante $f10$.

5 **¿Cuales son algunos de los desafíos de la optimización del código ensamblador para diferentes arquitecturas y plataformas de hardware?**
Alguno de estos desafíos es la adaptabilidad de la arquitectura, ya que si tenemos dos arquitecturas diferentes, al querer adaptar un programa de una a otra, se necesita ver la equivalencia de instrucciones, y puede que estas sean muchísimo mas costosas de una a otra arquitectura.
También hay que contemplar que hay casos donde ciertas arquitecturas están creadas para soportar de mejor forma ciertas tareas. Un ejemplo claro sería comparar un celular con una computadora, ambos tienen diferentes set de instrucciones, pero la arquitectura de un celular están hecho para ser lo mas reducido posible y con el mayor aprovechamiento de energía posible, en cambió, una computadora esta hecha para poder dar el mayor rendimiento posible.

6 **¿Existe alguna diferencia en escribir programas en lenguaje ensamblador comparado con escribir programas en lenguajes de alto nivel?**
Si, la gran diferencia radica que en lenguaje ensamblador, las instrucciones son a un nivel muy bajo, mientras que en lenguajes de nivel alto, puede hacerse una abstracción de estas instrucciones a unas mas generales y así no tener que preocuparse por cosas como los registros de memoria.
En si, podríamos decir que la gran diferencia radica en que las instrucciones en ensamblador, podrían decirse que son instrucciones a nivel de hardware y pueden ser muy técnicas de entender, mientras que en lenguajes de alto nivel, las instrucciones son menos técnicas y se asemejan mucho mas al lenguaje humano

7 **¿Cuales son algunas de las ventajas y desventajas de usar el lenguaje ensamblador en comparación con los lenguajes de programación de nivel superior?**
¿Como impactan estas ventajas/desventajas en el proceso de desarrollo, el rendimiento y la capacidad de mantenimiento del software?
La gran ventaja del lenguaje ensamblador, radica en su gran optimización de los programas, ya que como estamos casi manejando la computadora a nivel hardware, entonces podemos usar de la forma mas óptima, las herramientas de esta. Pero, esto viene a un gran precio, y es el tiempo, ya es muy tardado y tedioso programar en ensamblador, y la mayoría de veces, la diferencia de rendimiento entre un programa en C y un programa en ensamblador, es imperceptible. También el mantenimiento de un programa en ensamblador, puede resultar muy complejo y tomaría mucho mas tiempo que uno en un lenguaje de programación mas alto. En conclusión, obtendrías una gran optimización de un programa, tanto en espacio como en tiempo, pero con la desventaja de pasar mucho mas tiempo haciendo este programa.

Ejercicios

1 En el lenguaje de programación de tu elección, escribe un programa que calcule la serie

$$4 \times \sum_{n=0}^{m} \frac{(-1)^n}{2n+1}$$

donde m es el numero de iteraciones que se ejecutara el programa. NO es necesario que mandes el código de este ejercicio, solo manda una screenshot de tu código.
Comenta que hace cada linea y que es lo que hace el código en general.
Prueba poniendo $m = 10$, $m = 100$, después $m = 1000$, $m = 10000$, etc.

En un archivo llamado `Practica6.C` :

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Programa que calcula la suma de leibnz con valores arbitrarios de m
double leibniz(int m) {
    double resultado = 0;  // Inicializamos el resultado a 0
    for (int n = 0; n < m; n++) {   // Iteramos hasta m
        int arriba = (n % 2 == 0) ? 1 : -1; // (-1)^n
        double division = arriba / (2.0*n+1.0); // 1/(2n+1)
        resultado += division;  // Sumamos el resultado
    }
    return resultado * 4;  // Multiplicamos por 4 el resultado
}

int main() {
    // Pedimos un valor de m al usuario
    int m;
    printf("Introduce un valor de m: ");
    scanf("%d", &m);

    double pi = leibniz(m);
    printf("Pi: %.15f\n", pi);
    return 0;
}
```

Compilar con:

```
gcc -o Practica6 Practica6.c -lm
```

Resultados:

```
> ./Practica6
Introduce un valor de m: 10
Pi: 3.041839618929403

> ./Practica6
Introduce un valor de m: 100
Pi: 3.131592903558554

> ./Practica6
Introduce un valor de m: 1000
Pi: 3.140592653839794

> ./Practica6
Introduce un valor de m: 10000
Pi: 3.141492653590035

> ./Practica6
Introduce un valor de m: 100000
Pi: 3.141582653589720

> ./Practica6
Introduce un valor de m: 1000000
Pi: 3.141591653589774

> ./Practica6
Introduce un valor de m: 10000000
Pi: 3.141592553589792

> ./Practica6
Introduce un valor de m: 100000000
Pi: 3.141592643589326

> ./Practica6
Introduce un valor de m: 1000000000
Pi: 3.141592652588050

> ./Practica6
Introduce un valor de m: 10000000000
Pi: 3.141592652878837
```