Arquitecturas y Organización de Computadoras I
2° Cuatrimestre

TP N° 6 – Programación en lenguaje ensamblador MIPS con MIPSX

Facultad de Informática

Objetivo: Comprender la estructura de un programa en lenguaje ensamblador MIPS básico. Introducción a la interfaz mipsx y al conjunto de instrucción MIPS.

Recursos y Bibliografía:

Arq. MIPS Vol I, II and III.
Programa mipsx desarrollado por la cátedra.

1. Cree un programa que dado un número flotante de precisión simple extraiga los valores de signo, exponente y mantisa, guardando 1 si es positivo y -1 si es negativo en la variable `signo`, guardando el exponente traducido de notación en exceso a complemento a dos (es decir, habiéndose restado 127) en la variable `exponente`, y la mantisa (sin modificar, poner en cero todos los bits que no pertenezcan a la mantisa) en la variable `mantisa`.

```
.data
    memoria:
    flotante:
        .float 3.14
    signo:
        .byte 0 # -1 si es negativo, 1 si es positivo
    exponente:
        .byte 0 # En C2
    mantisa:
        .word 0 # Sin modificar del punto flotante,
                # es decir sin agregar el cero.
```

a. Cargue el programa anterior en **MIPSX** y, observando el código desensamblado, identifique cuales de las instrucciones de su programa son pseudo instrucciones y a que instrucciones máquina fueron traducidas.
b. Analizando el volcado de memoria, identifique la codificación expresada en hexadecimal de dos instrucciones aritméticas, una de salto, y dos de carga y almacenamiento ¿Cuál es la dirección del último byte del segmento de texto?
c. ¿Cuál es la dirección de memoria de los datos `flotante`, `signo` y `mantisa`? ¿Cuántos bytes se desperdician por alineamiento?

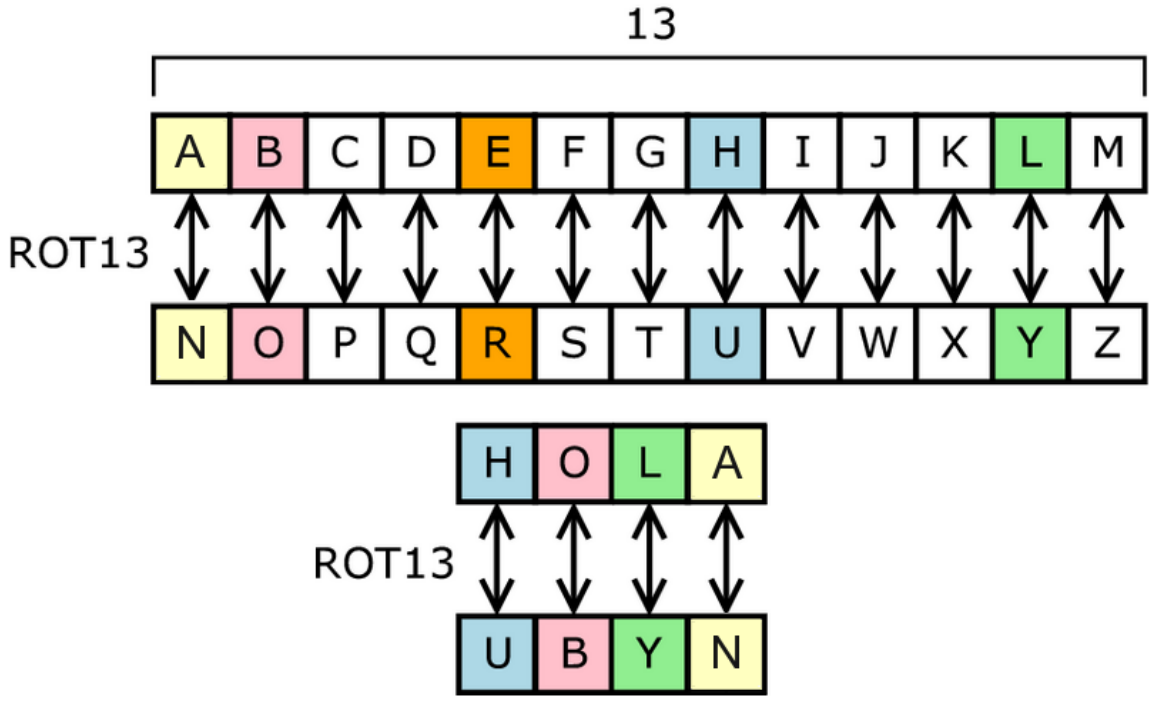
2. Dado el siguiente segmento de datos donde `Imagen` representa un a imagen de 9 pixeles con el formato **RGB** donde cada byte es la intensidad de color que compone cada pixel (reprecentada como un entero sin signo donde cero indica minima intencidad y 255 maxima intencidad). Realizar un programa que haga una conversión de colores a escala de grises, guardando el resultado en `destino` (la imagen destino también debe ser almacenada en formato **RGB**, con las tres componentes almacenando el mismo valor).

Para pasar a escala de grises deben realizar el promedio entre las tres componentes que forman el píxel, donde cada componente es un entero sin signo que va de `0..255`.

```
.data
```

```
memoria:
Imagen:
    .byte 123,0,33,44,55,2,56,78,66
    .byte 0,33,44,55,2,56,78,66,123
    .byte 33,44,55,2,56,78,66,23,55
destino: .space 27
```

3. Cree un programa que aplique el cifrado `ROT13` a una cadena de texto `ASCII`. Este cifrado sustituye las letras de la A a la M por aquellas de la N a la Z, y viceversa.



4. Dado el siguiente programa escrito en código ensamblador de mips, y suponiendo que el compilador traduce cada pseudo instrucción a 3 instrucciones máquina ¿Cuantos bytes ocupa el segmento de texto? Si la dirección de la primera instrucción es `0x400B0` ¿Cuál es la dirección de la última instrucción? ¿Cuál es la dirección del último byte del segmento de texto?

```
li $t0, 5
add $t3, $t0, $t0
move $t4, $t0
la $t0, memoria
lw $t3, 11($t0)
ori $t3, $t4, 1
j start
blt $t3, $7, menor
lw $t3, valor
```

5. Desarrolle un programa calculadora. Este programa lee una cadena de texto de 3 letras, donde se encuentran dos operandos de un dígito y una operación, como se presenta en el segmento de datos de ejemplo, en la etiqueta **`calcular`**.

```
.data

memoria:
calcular:  .ascii "3+6"
resultado_ascii:  .byte 0
resultado: .byte 0
```

La calculadora soporta la operación de suma o resta.
El programa debe:

1. Convertir los valores ascii a su valor numérico.
2. Decodificar la operación a realizar, que está también en representación de código de caracteres ascii, entre los operandos.
3. Realizar la operación.
4. Almacenar el resultado numérico en la posición de memoria **`resultado`**.
   El resultado también deberá almacenarse en la posición de memoria **`resultado_ascii`**, pero en representación de código de caracteres ascii.

El resultado posible es de un sólo dígito, en el rango de 0 a 9. Si el resultado excede este rango de representación coloque en **`resultado_ascii`** la letra "**`N`**".

6. Trabaje con el siguiente código.

```
unsigned char  distancias[20] = { 123, 43, 83, 9, 210, 0, 22, 94, 174, 19, 5,
12, 6, 8, 9, 0, 6, 2, 10, 200 };
unsigned char  inc[20] = { 1, 10, 9, 19, 2, 87, 9, 8, 7, 6, 5, 4, 3, 56, 56,
57, 58, 59, 100, 23};
int i;
int res = 0;

main ()
{
      for (i=0; i<20; i++) {
            distancias[i] = distancias[i] + inc[i];
            res = res + distancias[i];
      }
}
```

a. Traduzca el programa a lenguaje ensamblador MIPS, como lo haría un compilador. El tipo de datos unsigned char tiene un rango de representación de [0 a 255]. El tipo de dato int puede almacenar una palabra de la arquitectura. Ejecute su programa en mipsx.

b. Si el procesador MIPS tiene un reloj de 1Mhz, ¿cuanto se demora en ejecutar este programa?

c. En esta computadora se cuenta con una memoria caché de 8 entradas. En cada línea de caché cabe una palabra. Un acceso a caché demora 1 microsegundo. Un acceso a memoria principal demora 100 microsegundos.

   Utilizando las direcciones de memoria proporcionadas en la ejecución en mipsx responda:
   - ¿Cuántos fallos de caché tuvo la ejecución del programa?
   - ¿Cuál es el tiempo de acceso medio luego de ejecutar el programa?
   - ¿Cuáles son las direcciones efectivas que genera la CPU cuando debe acceder a distancias[4], distancias[19], inc[5], inc[18], res?
   - ¿En qué líneas de caché se ubican los datos para esas direcciones efectivas?
   - ¿Se genera fallo de caché en el primer acceso a esas direcciones efectivas?
   - ¿Cuál es el tiempo de ejecución del programa si se tiene en cuenta esta caché?
   - ¿Y si no existiese esta caché?