

Guía de Solución Trabajo de Laboratorio 2

Sistemas con Microprocesadores

16/09/2026

Application Binary Interface (ABI)

Para permitir la interoperación de código generado por distintos compiladores es necesario definir una *interfaz binaria de aplicación*, la que tiene por objeto estandarizar el formato de los archivos objeto, los tipos de datos, uso de los registros, organización de la pila y el paso de parámetros a las funciones. De estos aspectos mencionados en este laboratorio nos interesa en particular las definiciones que afectan al comportamiento de las subrutinas definidos en el documento.

Registro	Nombre	Uso	Observaciones
x1	RA	Dirección de retorno	Cálculo de la dirección
x2	SP	Puntero de Pila	Apunta al primer lugar de la pila
x5 a x7	t0 a t2	Temporales	No conservan el valor entre llamadas
x8 a x9	s0 a s1	Variables	Conservan el valor entre llamadas
x10 a x11	a0 a a1	Argumentos	Retorno de valores
x12 a x17	a2 a a7	Argumentos	No conservan el valor entre llamadas
x18 a x27	s2 a s11	Variables	Conservan el valor entre llamadas
x28 a x31	t3 a t6	Temporales	No conservan el valor entre llamadas

Desarrollo de los problemas

En el presente laboratorio usted deberá desarrollar todos los problemas propuestos utilizando los conceptos de subrutinas transparentes y reutilizables. Esto implica que se deben respetar las convenciones de la interfaz binaria de aplicación previamente especificada.

En general en los problemas deberá también desarrollar un programa principal que prepare los parámetros, llame a la subrutina y reciba el resultado de la misma. En algunos problemas usted puede tener que desarrollar mas de un subrutina con diferente niveles de abstracción, en este caso deberá comenzar por las subrutinas mas simples y después implementar las más complejas utilizando las funciones ya desarrolladas.

Llamadas al Sistema operativo

Las llamadas al sistema operativo permiten pedirles servicios al mismo tales como interacción con el usuario mediante la consola o la lectura y escritura de archivos. Para realizar una llamada al sistema lo primero que debemos hacer es cargar el código del servicio que deseamos ejecutar en el registro a0. De acuerdo al servicio deseado, si el mismo requiere argumentos deberán cargarse los mismos en los registros correspondientes antes de ejecutar la instrucción ecall. Una vez ejecutada esta instrucción, podrá leerse la información recibida de los registros especificados (para los servicios que retornan un valor). La lista completa de los servicios disponibles se puede consultar en la Wiki del simulador en la dirección <https://github.com/ThaumicMekanism/venus/wiki>.

Problemas Propuestos

- 3) Escriba una subrutina que invierta una cadena de caracteres ASCII se encuentra almacenada en memoria. Esta subrutina recibe el puntero al inicio de la cadena en **a0** y un puntero al final en **a1**.
- a) Diseñe una subrutina **cambiar** que invierta el primer y el último carácter de la cadena. Esta subrutina recibe el puntero al primer elemento en **a0** y al último en **a1**.
 - b) Diseñe ahora una subrutina iterativa **invertirIT** que utiliza la rutina anterior para intercambiar los elementos de la cadena.
 - c) Diseñe ahora una subrutina recursiva **invertirRE** que utiliza la rutina del apartado a para intercambiar los elementos del extremo de la cadena y a sí misma para invertir el resto.
 - d) Mostrar por pantalla los resultados de cada subrutina con 2 cadenas distintas.

```
.data
cadenaIt: .asciiz "Hola Mundo"
cadenaRec: .asciiz "Mundo Hola"
.text
.globl main
main:
    la a0, cadenaIt      # Cargo la dirección del primer elemento de la cadena
    mv a1, a0            # Copio la dirección a a1

ultimoIt:
    addi a1, a1, 1       # Sumo 1 a la dirección
    lb t0, 0(a1)         # Cargo el elemento de la dirección
    bnez t0, ultimoIt    # Si no es el terminador, seguir sumando
    addi a1, a1, -1      # Si encontré el terminador, resto 1 para apuntar al último
    character

    jal invertirIt       # Llamado a la subrutina

    la a1, cadenaIt      # Cargo la dirección de la cadena
    li a0, 4             # Selección print_string
    ecall

    li a0, 11            # Print Char
    li a1, 10            # Cargo \n
    ecall

    la a0, cadenaRec      # Cargo la dirección del primer elemento de la cadena
    mv a1, a0            # Copio la dirección a a1

ultimoRec:
    addi a1, a1, 1       # Sumo 1 a la dirección
    lb t0, 0(a1)         # Cargo el elemento de la dirección
    bnez t0, ultimoRec   # Si no es el terminador, seguir sumando
    addi a1, a1, -1      # Si encontré el terminador, resto 1 para apuntar al último
    character

    jal invertirRec      # Llamado a la subrutina

    la a1, cadenaRec      # Cargo la dirección de la cadena
    li a0, 4             # Selección print_string
    ecall

    li a0, 10            # Selección del servicio terminar el programa sin error
    ecall                # Termina el programa y retorna al sistema operativo

cambiar:
    lb t0, 0(a0)          # Cargo el primer elemento
    lb t1, 0(a1)          # Cargo el último elemento
    sb t0, 0(a1)          # Cambio el primer elemento
    sb t1, 0(a0)          # Cambio el último elemento
    ret
```

```
invertirIt:
    addi sp, sp, -4      # Reduzco 4 el stack pointer para push
    sw ra, 0(sp)        # Push de ra
loopIt:
    ble a1, a0, finInvertirIt # Si la segunda direccion es igual o menor, no se cambia
    nada
    jal cambiar          # Llamo a cambiar
    addi a0, a0, 1       # Sumo uno a la menor
    addi a1, a1, -1      # Resto uno a la mayor
    j loopIt             # Salto incondicional
finInvertirIt:
    lw ra, 0(sp)         # Aumento 4 el stack pointer para pop
    addi sp, sp, 4       # Pop de ra
    ret                  # Retorno al programa principal

invertirRec:
    addi sp, sp, -4      # Reduzco 4 el stack pointer para push
    sw ra, 0(sp)        # Push de ra

    ble a1, a0, finInvertirRec # Si la segunda direccion es igual o menor, no hace falta
    cambiar nada
    jal cambiar          # Llamada a la funcion de cambiar
    addi a0, a0, 1       # Aumento 1 a la primera dirección
    addi a1, a1, -1      # Reduzco 1 a la última dirección
    jal invertirRec      # Llamada recursiva

finInvertirRec:
    lw ra, 0(sp)         # Aumento 4 el stack pointer para pop
    addi sp, sp, 4       # Pop de ra
    ret
```