PRÁCTICA 5 - RESOLUCIÓN

Pila, Subrutinas

Objetivos Familiarizarse con los conceptos que rodean a la escritura de subrutinas en una arquitectura RISC. Uso normalizado de los registros, pasaje de parámetros y retorno de resultados, generación y manejo de la pila y anidamiento de subrutinas.

Parte 1: Pila y Subrutinas.

1. Comprendiendo la primer subrutina: potencia ⭐

Muchas instrucciones que normalmente forman parte del repertorio de un procesador con arquitectura CISC no existen en el MIPS64. En particular, el soporte para la invocación a subrutinas es mucho más simple que el provisto en la arquitectura x86 (pero no por ello menos potente). El siguiente programa muestra un ejemplo de invocación a una subrutina llamada **potencia**:

```
.data
base: .word 5
exponente: .word 4
result: .word 0

.code
ld    $a0, base($zero)
ld    $a1, exponente($zero)
jal   potencia
sd    $v0, result($zero)
halt
```

```
potencia: daddi $v0, $zero, 1
    lazo: beqz  $a1, terminar
          daddi $a1, $a1, -1
          dmul  $v0, $v0, $a0
          j     lazo
terminar: jr    $ra
```

a) ¿Qué hace el programa? ¿Cómo está estructurado el código del mismo?

En las primeras dos líneas del programa principal, se cargan valores a dos registros para ser pasados a la subrutina "potencia", $a0 tiene el valor de la base, y $a1 el exponente.
La subrutina "potencia" calcula la potencia multiplicando $a0 por sí mismo tantas veces como lo indica el valor que contiene $a1
En el registro $v0 se le retorna al programa principal el valor de la potencia.

b) ¿Qué acciones produce la instrucción `jal`? ¿Y la instrucción `jr`?

La instrucción "jal" = jump and link salta a "potencia", que es la etiqueta (dirección) de la primera instrucción de la subrutina. Además, salva en $ra la dirección de retorno (dir de jal + 4)
La instrucción jr $ra carga en el PC la dirección salvada en $ra (equivalente al ret)

c) ¿Qué valor se almacena en el registro `$ra`? ¿Qué función cumplen los registros `$a0` y `$a1`? ¿Y el registro `$v0`? ¿Qué valores posibles puede recibir en **$a0** y **$a1** la subrutina **potencia**?

En el registro $ra se guarda la dirección de memoria de la instrucción

"sd $v0, result($zero)", instrucción siguiente al jal.
Los registros $a0 y $a1 son los usados como parámetros de entrada a la subrutina, mientras que $v0 es el usado para retornar el valor al programa principal (parámetro de salida). Los valores que se pasan son todos de tipo entero "WORD".

d) Supongamos que el WinMIPS no posee la instrucción **dmul** ¿Qué sucede si la subrutina **potencia** necesita invocar a otra subrutina para realizar la multiplicación en lugar de usar la instrucción **dmul**? ¿Cómo sabe cada una de las subrutinas a que dirección de memoria debe retornar?

La subrutina potencia debe guardar el valor que contiene $ra previo a invocar a otra subrutina, para no perderlo con ese próximo salto. Una vez que se retorna de la segunda subrutina se debe actualizar nuevamente el valor en $ra para que pueda retornar al programa principal.

e) Escriba un programa que utilice **potencia**. En el programa principal se solicitará el ingreso de la base y del exponente (ambos enteros) y se deberá utilizar la subrutina **potencia** para calcular el resultado pedido. Muestre el resultado numérico de la operación en pantalla.

```
            .data
mensajeBase: .asciiz "Ingrese un número para la base: "
mensajeExpo: .asciiz "Ingrese un número para el exponente: "
mensajeResu: .ascii "El resultado de la potencia: "
CONTROL:    .word32  0×10000
DATA:       .word32  0×10008

            .code
            lwu $t6, CONTROL($0)
            lwu $t7, DATA($0)

            daddi $t0, $0, 4
            daddi $t1, $0, mensajeBase  ; para imprimir mensajeBase
            sd $t1, 0($t7)              ; cargo mensajeBase
            sd $t0, 0($t6)              ;4 en control imprime msg
             daddi $t0, $0, 8
            sd $t0, 0($t6)              ;8 en control lee un numero
            lw $a0, 0($t7)              ;en $a0 el nro que esta en DATA
             daddi $t0, $0, 4           ;4 en control para imprimir msg
            daddi $t1, $0, mensajeExpo  ;para imprimir mensajeExpo
            sd $t1, 0($t7)              ;cargo mensajeExpo
            sd $t0, 0($t6)              ;4 en control para imprimir msg
            daddi $t0, $0, 8
            sd $t0, 0($t6)              ;8 en control lee un numero
            lwu $a1, 0($t7)             ;en $a1 el nro que esta en DATA

            jal potencia

             daddi $t0, $0, 4
             daddi $t1, $0, mensajeResu   ;para imprimir mensajeResul
            sd $t1, 0($t7)               ; mensajeResul
            sd $t0, 0($t6)               ;4 en control para imprimir msg
            daddi $t0, $0, 1
            sd $v0, 0($t7)               ;resultado en DATA
             sd $t0, 0($t6)             ;1 en control imprime entero
            halt

potencia:   daddi $v0, $zero, 1
lazo:       beqz $a1, terminar
            daddi $a1, $a1, -1
            dmul $v0, $v0, $a0
            j lazo
terminar:   jr $ra
```

f) Escriba un programa que lea un exponente **x** y calcule **2^x + 3^x** utilizando dos llamadas a **potencia**. Muestre en pantalla el resultado. ¿Funciona correctamente? Si no lo hace, revise su implementación del programa ¿Qué sucede cuando realiza una segunda llamada a **potencia**? **Pista**: Como caso de prueba, intente calcular $2^3+3^3 = 8+27 = 35$..

```
           .data
mensajeExpo: .asciiz "Ingrese un número para el exponente: "
mensajeResu: .ascii "El resultado es: "
    CONTROL: .word32  0×10000
       DATA: .word32  0×10008

      .code
      lwu $t6, CONTROL($0)
      lwu $t7, DATA($0)

      daddi $t0, $0, 4            ;4 en control para imprimir msg
      daddi $t1, $0, mensajeExpo  ;para imprimir mensajeExpo
      sd $t1, 0($t7)              ;mensajeExpo
      sd $t0, 0($t6)              ;4 en control para imprimir msg
      daddi $t0, $0, 8
      sd $t0, 0($t6)              ;8 a control lee nro
      lwu $a1, 0($t7)             ;en $a1 nro que esta en DATA,

      daddi $a0, $0, 2            ;2 a base

      jal potencia

      dadd $v1, $0, $v0           ;$v0 en $v1 p/2da llamada

      lwu $a1, 0($t7)             ;en $a1 nro que esta en DATA,
      daddi $a0, $0, 3            ;3 a base

      jal potencia

      dadd $v1, $v1, $v0          ;suma terminos

      daddi $t0, $0, 4
      daddi $t1, $0, mensajeResu
      sd $t1, 0($t7)              ;mensajResu
      sd $t0, 0($t6)              ;4 en control imprime msg
      daddi $t0, $0, 1
      sd $v1, 0($t7)              ;pongo el resultado en DATA
      sd $t0, 0($t6)              ;1 en control imprime un entero
      halt

potencia:   daddi $v0, $zero, 1
lazo:       beqz $a1, terminar
            daddi $a1, $a1, -1
            dmul $v0, $v0, $a0
            j lazo
terminar:   jr $r
```

2. Salvado de registros ⭐

Los siguientes programas tienen errores en el uso de la convención de registros. Indicar qué registros cuál es el error y cómo se podría arreglar el problema en cada caso.

.

A)

```
   .code

   daddi $t0, $0, 5
   daddi $t1, $0, 7
   jal subrutina
   sd $t2, variable ($0)
   halt

   subrutina: daddi $t4, $0, 2
              dmul $t0, $t0, $t4
              dmul $t1,$t1,$t4
              dadd $t2,$t1,$t0
              jr $ra
```

B)

```
   .code
   daddi $a0, $0, tabla
   jal subrutina
   daddi $t0, $0, 10
         daddi $t1, $0, 0
   loop: bnez $t0, fin
         ld $t2, 0($a0)
         dadd $t1, $t1, $t2
         daddi $t0, $t0, -1
         dadd $a0, $a0, 8
         j loop
   fin: halt
```

C)

```
   .code
   daddi $a0, $0, 5
   daddi $a1, $0, 7
   jal subrutina
   dmul $t2, $a0, $v0
   sd $t2, variable ($0)
   halt
```

D)

```
   .code
         daddi $t0, $0, 10 # dimension
         daddi $t1, $0, 0 # contador
         daddi $t2, $0, 0 # desplazamiento
   loop: bnez $t0, fin
         ld $a0, tabla ($t2)
         jal espar
         bnez $v0, seguir
         dadd1 $t1, $t1, 1
  seguir:daddi $t2, $t2, 1
         daddi $t0, $t0, -1
         dadd $t2, $t2, 8
         j loop
        sd $t2, resultado($0)
   fin: halt
```

A) El programa está usando $t0 y $t1 como parámetros de entrada, y $t2 como parámetro de retorno. Se deberían usar $a0 y $a1 y devolver el resultado en $v0.
B) Se está usando $a0 (registro no preservado) después de llamado a subrutina. Se soluciona utilizado por ejemplo el registro $s0.
C) Idem B), se está usando $a0 luego del llamado a subrutina.
D) En cada iteración se llama a subrutina y luego se utilizan $t0, $t1 y $t2, valores que pueden ser modificados en la subrutina. Utilizar por ejemplo $s0, $s1 y $s2.

3. Uso de la pila ⭐

En WinMIPS no existen las instrucciones **PUSH** y **POP**. Por ese motivo, deben implementarse utilizando otras instrucciones existentes. No solo eso, sino que el registro SP es en realidad un registro usual, r29, que con la convención se puede llamar por otro nombre, **$sp**. El siguiente programa debería intercambiar los valores de $t0 y $t1 utilizando la pila. No obstante, así como está no va a funcionar porque push y pop no son instrucciones válidas. Implementar la funcionalidad que tendrían estas operaciones utilizando instrucciones daddi, sd y ld para que el programa funcione correctamente. Recordar que los registros ocupan 8 bytes, y por ende el push y el pop deberán modificar a $sp con ese valor.

```
.code
daddi $sp, $0, 0×400
daddi $t0, $0, 5
daddi $t1, $0, 8
push $t0
push $t1
pop $t0
pop $t1
halt
```

```
       .code
daddi $sp, $0, 0×400
daddi $t0, $0, 5
daddi $t1, $0, 8

;push $t0
daddi $sp, $sp, -8
sd $t0, 0($sp)

;push $t1
daddi $sp, $sp, -8
sd $t1, 0($sp)

;pop $t0
ld $t0, 0($sp)
daddi $sp, $sp, 8

;pop $t1
ld $t1, 0($sp)
daddi $sp, $sp, 8

halt
```

4. Variantes de la subrutina potencia ⭐⭐

La versión anterior de **potencia** utiliza pasaje por registros y por valor. Escribir distintas versiones de la subrutina **potencia** en base términos del pasaje de parámetros

a) **Referencia y Registro** Pasando los parámetros por referencia desde el programa principal a través de registros, y devolviendo el resultado a través de un registro por valor.

```
          .data
     base:.word 5
exponente:.word 4
   result:.word 0

          .code
          daddi $a0, $0, base        ;dirección de memoria de base
          daddi $a1, $0, exponente   ;dirección de memoria de exponente

          jal potencia

          sd $v0, result($zero)
          halt

potencia: daddi $v0, $zero, 1
          ld $t1, 0($a1)      ;contenido dirección de memoria exponente
          ld $t0, 0($a0)      ;contenido dirección de memoria base
    lazo: beqz $t1, terminar
          daddi $t1, $t1, -1
          dmul $v0, $v0, $t0
          j lazo
terminar: jr $ra
```

b) **Valor y Pila** Pasando los parámetros por valor desde el programa principal a través de la pila, y devolviendo el resultado a través de un registro por valor.

```
           .data
     base:.word 5
exponente:.word 4
   result:.word 0

          .code
          daddi $sp, $0, 0×0400
          ld $a0, base($zero)
          ld $a1, exponente($zero)
; almaceno en pila base y exponente
; se podría usar el desplazamiento con -8 y -16 sin ir restando a $sp

           daddi $sp, $sp, -8       ;push $a0
           sd $a0, 0($sp)

           daddi $sp, $sp, -8       ; push $a1
           sd $a1, 0($sp)

           jal potencia
           sd $v0, result($zero)
           halt

potencia: daddi $v0, $zero, 1
          ld $t1, 0($sp)      ; pop $a1
          daddi $sp, $sp, 8
          ld $t0, 0($sp)      ; pop $a0
          daddi $sp, $sp, 8
    lazo: beqz $t1, terminar
          daddi $t1, $t1, -1
          dmul $v0, $v0, $t0
          j lazo
terminar: jr $ra
```

c) **Referencia y Pila** Pasando los parámetros por referencia desde el programa principal a través de la pila, y devolviendo el resultado a través de un registro por valor.

```
          .data
     base:.word 5
exponente:.word 4
   result:.word 0

          .code
          daddi $sp, $0, 0×0400
          daddi $a0, $0, base
          daddi $a1, $0, exponente
; almaceno en pila direcciones de memoria, base y exponente
; se podría usar el desplazamiento con -8 y -16 sin ir restando a $sp,

          daddi $sp, $sp, -8  ;push $a0
          sd $a0, 0($sp)


          daddi $sp, $sp, -8   ; push $a1
          sd $a1, 0($sp)

          jal potencia
          sd $v0, result($zero)
          halt

potencia: daddi $v0, $zero, 1
          ld $t2, 0($sp)      ; pop la dirección de exponente
          daddi $sp, $sp, 8
          ld $t3, 0($sp)      ; pop la dirección de base
          daddi $sp, $sp, 8
          ld $t1, 0($t2)
          ld $t0, 0($t3)

    lazo: beqz $t1, terminar
          daddi $t1, $t1, -1
          dmul $v0, $v0, $t0
          j lazo
terminar: jr $ra
```

5. Salvado de registros en subrutinas anidadas ⭐⭐

Las siguientes subrutinas anidadas tienen errores en el uso de la convención de los registros, en especial con respecto a cuales tienen que salvarse y cuáles no, y también cuándo y en qué caso debe hacerse. Indicar los errores y corregir el código para que las subrutinas usen la convención correctamente.

A)
```
#v0: devuelve 1 si a0 es impar y 0 dlc
#a0: número entero cualquiera
esimpar: andi $v0, $a0, 1
         jr $ra

#v0: devuelve 1 si a0 es par y 0 dlc
#a0: número entero cualquiera
espar: jal esimpar
       #truco: espar = 1 - esimpar
       daddi $s0, $0, 1
       dsub $v0, $s0, $v0
       jr $ra
```

B)
```
#v0: devuelve la cantidad de bits 0
que tiene un número de 64 bits
#a0: número entero cualquiera
cant0: daddi $t0, $0, 0
       daddi $t1, $0, 64
  loop:jal espar
       dadd $t0,$t0,$v0
       #desplazo a la derecha
       #para quitar el último bit
       dsrl $a0, $a0, 1
       daddi $t1, $t1, -1
       bnez $t1, loop
       jr $ra
```

C)
```
#v0: volumen de un cubo
#a0: long del lado lado del cubo
vol: daddi $sp, $sp, -16
     sd $ra, 0($sp)
     sd $s0, 8($sp)
     dadd $s0, $0, $a0
     dmul $s0,$a0,$a0
     dmul $s0,$s0,$a0
     daddi $v0,$s0,0
     ld $ra, 0($sp)
     ld $s0, 8($sp)
     daddi $sp, $sp, 16
     jr $ra

#v0: diferencia de volumen de los cubos
#a0: long del lado del cubo más grande
#a1: long del lado del cubo más chico
diffvol: jal vol
         daddi $t0,$v0,0
         daddi $a0,$a1,0
         jal vol
         dsub $v0,$t0,$v0
         jr $ra
```

A)En la subrutina espar se modifica $s0, el programa principal esperaría que se conserve su valor.Además la subrutina espar llama a esimpar sin salvar dirección de retorno $ra.
B)$t0, $t1 y $a0 se actualizan dentro del lazo y pueden volver modificados de la subrutina espar. Si se quiere conservar su valor los debería mover a otros registros conservados, o apilarlos. Pero hay un error más grave, el programa principal llama a subrutina espar y ésta a su vez llama a la subrutina esimpar sin salvar la dirección de retorno $ra.
C)La subrutina diffvol es quien llama a la subrutina vol, por lo cual en diffvol se debería salvar el valor de $ra. Guardar el valor en vol no tiene sentido. Además no se puede asumir en diffvol que la subrutina no modificó $a1, si quería usar su valor lo debería haber preservado antes. Por último, no tiene sentido usar $s0 en vol (si bien se preserva correctamente) si no se hacen llamados desde ella a otra subrutina. En diffvol sí se hay llamados, y se debería usar un $s.

6. Subrutinas anidadas ⭐⭐

a) **Factorial:** Implemente la subrutina **factorial**, que dado un número **N** devuelve **factorial(N) = N! = N * (N-1) * (N-2) * ... * 2 * 1.** Por ejemplo, el factorial de 4 es 4! = 4*3*2*1. Recordá también que el factorial de 0 también existe, y es **factorial(0) =0! = 1**

```
        .data
Valor: .word 6
Resul: .word 0
```

```
            .code
            ld $a0, Valor($0)
            jal factorial
            sd $v0, Resul($0)
            halt

factorial: daddi $v0, $0, 1
     otro: beqz $a0, fin
           dmul $v0, $v0, $a0
           daddi $a0, $a0, -1
           j otro
      fin: jr $ra
```

b) **Número combinatorio:** Utilizando **factorial**, implementa la subrutina **comb** que calcula el **número combinatorio** (también llamado **coeficiente binomial**) **comb(m,n) = m! / (n! * (n-m)!)**. Asumir que **n > m**.

```
            .data
   ValorM: .word 6
   ValorN: .word 3
    Resul: .word 0

           .code
           daddi $sp, $0, 0×400
           ld $a0, ValorM($0)
           ld $a1, ValorN($0)
           jal comb
           sd $v0, Resul($0)
           halt

    comb: daddi $sp, $sp, -8
          sd $ra, 0($sp)
          dsub $t0, $a0, $a1
          jal factorial
          dadd $t1, $0, $v0
          dadd $a0, $0, $a1
          jal factorial
          dadd $t2, $0, $v0
          dadd $a0, $0, $t0
          jal factorial
          dmul $t0, $t2, $v0
          ddiv $v0, $t1, $t0
          ld $ra, 0($sp)
          daddi $sp, $sp, 8
          jr $ra

factorial:daddi $v0, $0, 1
     otro:beqz $a0, fin
          dmul $v0, $v0, $a0
          daddi $a0, $a0, -1
```

```
          j otro
      fin:jr $ra
```

7. Subrutinas de Strings ⭐⭐

Implemente subrutinas típicas para manipular strings, de modo de tener construir una pequeña librería de código reutilizable.

a) **longitud:** Recibe en $a0 la dirección de un string y retorna su longitud en $v0

```
 longitud: daddi $v0, $0 ,0
siguiente: lbu $t1, 0($a0)
           beqz $t1, fin
           daddi $v0, $v0, 1
           daddi $a0, $a0, 1
           j siguiente
      fin:  jr $ra
```

b) **contiene:** Recibe en $a0 la dirección de un string y en $a1 un carácter (código ascii) y devuelve en $v0 1 si el string contiene el carácter $a1 y 0 de lo contrario.

```
 contiene: daddi $v0, $0 ,0
siguiente: lbu $t1, 0($a0)
           beqz $t1, fin
           beq $t1, $a1, iguales
           daddi $a0, $a0, 1
           j siguiente
  iguales: daddi $v0, $0, 1
      fin: jr $ra
```

c) **es_vocal:** Determina si un carácter es vocal o no, ya sea mayúscula o minúscula. La rutina debe recibir el carácter y debe retornar el valor 1 si es una vocal ó 0 en caso contrario. Para implementar **es_vocal**, utilizar la subrutina **contiene**, de modo que para preguntar si un carácter es una vocal, se pregunte si un string con todas las vocales posibles contiene a este carácter.

```
           .data
  vocales: .asciiz "AEIOUaeiou"

          .code
           daddi  $sp, $0, 0×400      ; $a1 caracter a verificar
es_vocal:  daddi $a0, $0, vocales     ; $a0 dirección de vocales
           daddi $sp ,$sp, -8
           sd $ra, 0($sp)
           jal contiene
           ld $ra, 0($sp)
```

```
           daddi $sp, $sp, 8
           jr $ra

contiene:  daddi $v0, $0 ,0
siguiente: lbu $t1, 0($a0)
           beqz $t1, fin
           beq $t1, $a1, iguales
           daddi $a0, $a0, 1
           j siguiente
  iguales: daddi $v0, $0, 1
     fin: jr $ra
```

d) **cant_vocales** Usando la subrutina escrita en el ejercicio anterior, **cant_vocales** recibe una cadena terminada en cero y devuelve la cantidad de vocales que tiene esa cadena.

```
               .data
      cadena: .asciiz "Una cadena para contar"
     vocales: .asciiz "AEIOUaeiou"

               .code
               daddi  $sp, $0, 0×400

cant_vocales: daddi $t0, $0, 0
              daddi $sp ,$sp, -8
              sd $ra, 0($sp)
  otra_letra: lbu $a0, 0($a2)
              beqz $a0, fin_cadena
              jal es_vocal
              dadd $t0, $t0, $v0
              daddi $a2, $a2, 1
              j otra_letra
  fin_cadena: daddi $v0, $t0, 0
              ldu $ra, 0($sp)
              daddi $sp, $sp, 8
              jr $ra

    es_vocal: daddi $a1, $0, vocales   ;en $a0 la direccion de vocales
              daddi $sp ,$sp, -8
              sd $ra, 0($sp)
              jal contiene
              ldu $ra, 0($sp)
              daddi $sp, $sp, 8
              jr $ra

    contiene: daddi $v0, $0 ,0
   siguiente: lbu $t1, 0($a1)
              beqz $t1, fin
              beq $t1, $a0, iguales
              daddi $a1, $a1, 1
              j siguiente
     iguales: daddi $v0, $0, 1
         fin: jr $ra
```

e) **comparar:** Recibe como parámetros las direcciones del comienzo de dos cadenas terminadas en cero y retorna la posición en la que las dos cadenas difieren. En caso de que las dos cadenas sean idénticas, debe retornar -1.

```
          .data
 cadena1: .asciiz "hola"
 cadena2: .asciiz "hola"
  result: .word 0

          .code
          daddi $a0, $0, cadena1
          daddi $a1, $0, cadena2
          jal comparar
          sd $v0, result($zero)
          halt

comparar: dadd $v0, $0, $0
    loop: lbu $t0, 0($a0)
          lbu $t1, 0($a1)
          beqz $t0, fin_a0
          beqz $t1, final
          bne $t0, $t1, final
          daddi $v0, $v0, 1
          daddi $a0, $a0, 1
          daddi $a1, $a1, 1
          j loop
  fin_a0: bnez $t1, final
          daddi $v0, $0, -1
    final:jr $r
```

8. Subrutinas con vectores ⭐⭐

a) suma Escribir una subrutina que reciba como argumento en $a0 la dirección de una tabla de números enteros de 64 bits, y en $a1 la cantidad de números de la tabla. La subrutina debe devolver en $v0 la suma de los números de la tabla

```
            .data
   vector: .word 2,3,4,5,1,2,12,0,2
     cant: .word 9
   result: .word 0

            .code
            daddi $a0, $0, vector
            ld $a1, cant($0)
            jal Suma
            sd $v0, result($0)
            halt

     Suma: addi $v0, $0, 0
     otro: beqz $a1, fin
           ld $t0, 0($a0)
           dadd $v0, $v0, $t0
           daddi $a0, $a0, 8
           daddi $a1, $a1, -1
           j otro
      fin: jr $ra
```

b) positivos Idem a), pero la subrutina debe contar la cantidad de números positivos. Para ello, implementar y usar otra subrutina llamada **es_positivo**, que reciba un número en $a0 y devuelva en $v0 1 si es positivo y 0 de lo contrario.

```
           .data
    vector:.word 2, -3, 4, -5, 1, -2, 12, 0, 2
      cant:.word 9
    result:.word 0

            .code
            daddi $sp, $0, 0×0300
            daddi $a0, $0, vector
            ld $a1, cant($0)
            jal positivos
            sd $v1, result($0)
            halt

 positivos:daddi $sp, $sp, -8
           sd $ra, 0($sp)
           daddi $v1, $0, 0
           daddi $t0, $a0, 0
      otro:beqz $a1, fin
           ld $a0, 0($t0)
           jal es_pos
           daddi $t0, $t0, 8
           daddi $a1, $a1, -1
           dadd $v1, $v1, $v0  ; suma valor retornado de es_pos
           j otro
fin:       ld $ra, 0($sp)
           addi $sp, $sp, 8
           jr $ra
```

```
es_pos:    slt $v0, $0, $a0
           jr $ra
```

9. Subrutina recursiva ⭐⭐⭐

En un ejercicio anterior se implementó la subrutina **factorial** de forma iterativa. Implementar ahora la subrutina pero de forma **recursiva.**

La definición recursiva de **factorial** es:

Caso base: **factorial**(0) = 1

Caso recursivo: **factorial**(n) = n * **factorial**(n-1)

En términos de pseudocódigo, su implementación es:

```
subrutina factorial(n):
if n = 0:
   retornar 1
else:
   m = factorial(n-1)
   retornar n * m
```

a) Implemente la subrutina `factorial` definida en forma recursiva.

Pista 1: El caso base puede codificarse directamente

Pista 2: En el caso recursivo hay una llamada a otra subrutina, con lo cual deberá preservar el registro $ra

Pista 3: En el caso recursivo deberá primero llamar a **factorial** de forma recursiva, pero si el valor **n** original está guardado en un registro temporal, se perderá ese valor.

```
          .data
    valor:.word 10
   result:.word 0

          .code
          daddi $sp, $0, 0x400
          ld $a0, valor($0)
          jal factorial
          sd $v0, result($0)
          halt

factorial:daddi $sp, $sp, -16
          sd $ra, 0($sp)
          sd $s0, 8($sp)
          beqz $a0, fin_rec
          dadd $s0, $0, $a0
          daddi $a0, $a0, -1
          jal factorial
          dmul $v0, $v0, $s0
          j fin
 fin_rec: daddi $v0, $0, 1
     fin: ld $s0, 8($sp)
          ld $ra, 0($sp)
          daddi $sp, $sp, 16
          jr $ra
```

b)¿Es posible escribir la subrutina `factorial` sin utilizar una pila? Justifique.

NO, ya que tenemos que ir guardando la dirección de retorno por cada llamado a factorial, y podrían no alcanzar los registros, o tendríamos que ir guardando la dirección en un arreglo que es lo mismo que hace la pila.

Parte 2: Ejercicios tipo parcial

1. Lectura y procesamiento de números ⭐⭐⭐

Implementar una subrutina INGRESAR_NUMERO. La misma deberá solicitar el ingreso por teclado de un número entero del 1 al 9. Si el número ingresado es un número válido entre 1 y 9 la subrutina deberá imprimir por pantalla el número ingresado y retornar dicho valor. En caso contrario, la subrutina deberá imprimir por pantalla “Debe ingresar un número” y devolver el valor 0. Para ello, implementar y usar una subrutina ENTRE que reciba un número N y otros dos números B y A, y devuelva 1 si B<N<A o 0 de lo contrario.

Usando la subrutina INGRESAR_NUMERO implementar un programa que invoque a dicha subrutina y genere una tabla llamada NUMEROS con los valores ingresados. La generación de la tabla finaliza cuando la suma de los resultados obtenidos sea mayor o igual a el valor almacenado en la dirección MAX.

Al finalizar la generación de la tabla, deberá invocar a la subrutina PROCESAR_NUMEROS, que debe recibir como parámetro la dirección de la tabla NUMEROS y la cantidad de elementos y contar la cantidad de números impares ingresados. Se debe mostrar por pantalla el valor calculado, con el texto "Cantidad de Valores Impares: “ y el valor. Para ello, utilizar la subrutina ES_IMPAR codificada anteriormente.

```
          .data

mensaje:  .asciiz "Ingrese un número entre 1 y 9 "
msg_error:.asciiz "Debe ingresar un número! "
msg_final:.asciiz "Cantidad de Impares: "
      MAX:.word 020
resultado: .word 0

CONTROL: .word32 0x10000
   DATA: .word32 0x10008
numeros: .byte 0


         .code

         daddi $sp, $0, 0x300
         daddi $a0, $0, numeros     ;direccion de números
         daddi $a1, $0, 0           ; veces que ingresó un número
         daddi $s0, $0, 0           ;suma de los números ingresados
         ld $s1, MAX($0)            ;valor a alcanzar

 seguir: jal ingresar_numero
         dadd $s0, $s0, $v0         ; sumo el valor obtenido
         sb $v0, numeros($a1)       ;guardo en la tabla
         daddi $a1, $a1, 1          ;incremento numeros ingresados
         slt $s2, $s0, $s1          ;s2 = 1 si no alcance el total
         bnez $s2, seguir
```

```
         daddi $a0, $0, numeros     ;direccion de números
         jal procesar_numeros
         sd $v0, resultado($0)
         halt


ingresar_numero: daddi $sp, $sp, -8
                 sd $ra, 0($sp)
                 dadd  $t4, $0, $a0   ; resguardo valores
                 dadd  $t5, $0, $a1

                 lwu $t6, CONTROL($0)
                 lwu $t7, DATA($0)
                 daddi $t0, $0, 4    ; imprimo mensaje
                 daddi $t1, $0, mensaje
                 sd $t1, 0($t7)
                 sd $t0, 0($t6)
                 daddi $t0, $0, 9    ; leo un caracter
                 sd $t0, 0($t6)
                 ld $t2, 0($t7)

                  daddi $t2, $t2, -48  ; t2 tiene ascci del caracter.
                                       ; Le resto 48 (código del 0)

                  dadd  $a0, $0, $t2   ; oarametros para sub entre
                  daddi $a1, $0, 1
                  daddi $a2, $0, 9

                  jal entre

                  beqz $v0, error

                  daddi $vo, $t2, 0 ; devuelve $v0 = el número ingresado

                  daddi $t0, $0, 1  ; lo imprimo

                  sd $t2, 0($t7)
                  sd $t0, 0($t6)
                  j terminar

            error: daddi $t0, $0, 4  ;  imprime “Debe ingresar un número”
                   daddi $t1, $0, msg_error
                   sd $t1, 0($t7)
                   sd $t0, 0($t6)
                   daddi $vo, $0, 0  ;devuelve $v0 = 0

          terminar: dadd  $a0, $0, $t4
                    dadd  $a1, $0, $t5
              ld $ra, 0($sp)
              daddi $sp, $sp, 8
              jr $ra
```

```
          entre:  daddi $v0, $0, 0
                  slt $t9, $a0, $a1 ; si $a0 <1 ⇒ $t9 = 1  sino $t9 = 0
                  bnez $t9, no_es
                  daddi $a2, $a2, 1

                  slt $t9, $a0, $a2 ; si $a0 <10 ⇒ $t9 = 1  sino $t9 = 0
                  beqz $t9, no_es

                  daddi $v0, $0, 1
            no_es: jr $ra


procesar_numeros: daddi $sp, $sp, -8
                  sd $ra, 0($sp)
                  dadd  $t4, $0, $a0

                  daddi $t2, $0, 0        ;cantidad de números impares
             lazo: lb $t0, 0($t4)         ; leo de la tabla

                  dadd  $a0, $0, $t0
                  jal es_impar
                  beqz $v0, es_par

                  daddi $t2, $t2, 1     ;incremento cantidad de impares
           es_par: daddi $a1, $a1, -1    ;un elemento menos para procesar
                  daddi $t4, $t4, 1     ;nuevo elemento de la tabla
                  bnez $a1, lazo        ;si no llegue a 0, sigo

                  daddi $t0, $0, 4      ; imprimo mensaje
                  daddi $t1, $0, msg_final
                  sd $t1, 0($t7)
                  sd $t0, 0($t6)
                  daddi $t0, $0, 1      ;muestro cantidad impares
                  sd $t2, 0($t7)
                  sd $t0, 0($t6)
                  dadd $v0, $0, $t2     ; devuelvo en v0 la cantidad

                  ld $ra, 0($sp)
                  daddi $sp, $sp, 8

                  jr $ra

          es_impar: andi $v0, $a0, 1
                  jr $ra
```

2. Colores alternativos⭐⭐⭐

Escribir un programa que imprime alternativamente un punto rojo y uno azul en la pantalla gráfica, y llena toda la pantalla de esta forma.
Para ello, implementar una subrutina **fila_alternativa** que recibe un número de fila y dos colores, y llena toda la fila con pixeles de los dos colores de forma alternativa. Utilizando **fila_alternativa**, escriba un programa que pinte toda la pantalla de rojo y azul, de forma tal que en la primera fila se comience con el color rojo, en la 2da con azul, y así sucesivamente.

```
           .data
     pos_X:.byte 0
     pos_Y:.byte 0
color_rojo:.byte 255, 0, 0, 0
color_azul:.byte 0, 0, 255, 0
   CONTROL:.word32 0×10000
      DATA:.word32 0×10008

          .code
          lbu $s0, pos_Y(r0)
          lbu $s1, pos_X(r0)
          lwu $s2, color_rojo(r0)
          lwu $s3, color_azul(r0)

          daddi $s4, $0, 1
   otra_f:dadd $a0, $0, $s0
          beqz $s4 , invertir
          daddi $a1, $s2, 0     ; color rojo en $a1
          daddi $a2, $s3, 0     ; color azul en $a2
          j salteo
invertir: daddi $a2, $s2, 0     ; color rojo en $a2
          daddi $a1, $s3, 0     ; color rojo en $a1

 salteo: jal fila_alternativa

         daddi $s0, $s0, 1   ; incremento coord y
         lbu $s1, pos_X(r0)  ; reinicio coord x
         xori $s4, $s4, 1
         slti $s5, $s0, 50
         bnez $s5, otra_f
         halt

fila_alternativa: lwu $t6, CONTROL(r0)
                  lwu $t7, DATA(r0)
                  daddi $t0, $0, 0
                  daddi $t1, $0, 1
                  sb $a0, 4($t7)      ; fila fija
           seguir:sb $t0, 5($t7)
                  beqz $t1 , color
                  daddi $t4, $a1, 0   ;color 1
                  j salto
            color:daddi $t4, $a2, 0   ;color 2
            salto:sw $t4, 0($t7)
                  xori $t1, $t1, 1    ;invierte valor
                  daddi $t2, $0, 5
                  sd $t2, 0($t6)      ; 5 en control
                  daddi $t0, $t0,1    ; incremento coord x
```

```
                slti $t9, $t0, 50
                bnez $t9, seguir
                jr $ra
```