Práctica 5: Punto flotante, pila y subrutinas

Parte 1: Punto Flotante

Ejercicio 1

Instrucción	Descripción
mtc1 r_f, f_d	Copia los 64 bits del registro entero r_f al registro de punto flotante f_d
cvt.l.d f_d, f_f	Convierte a entero el valor en punto flotante contenido en f_f, escribiéndolo en f_d
cvt.d.l f_d, f_f	Convierte a punto flotante el valor entero copiado al registro f_f, escribiéndolo en f_d
mfc1 r_d, f_f	Copia los 64 bits del registro de punto flotante f_f al registro entero r_d

Para convertir un valor entero almacenado en cualquiera de los registros enteros (r0 a r31) a su representación equivalente en punto flotante y escribir esta última en uno de los registros de punto flotante (f0 a f31):

- Copiar los 64 bits del registro entero “rf” al registro de punto flotante “fd” ejecutando la instrucción “mtc1 rf, fd”.
- Convertir a punto flotante el valor entero copiado al registro “ff” y escribirlo en “fd” ejecutando la instrucción “cvt.d.l fd, ff”.
- Importante: los números muy grandes serán redondeados en su mejor representación de punto flotante.

Para convertir un valor en punto flotante almacenado en cualquiera de los registros de punto flotante (f0 a f31) a su representación equivalente en entero (punto fijo) y escribir esta última en uno de los registros enteros (r1 a r31):

- Convertir a entero el valor en punto flotante contenido en el registro “ff” y escribirlo en “fd” ejecutando la instrucción “cvt.l.d fd, ff”.
- Copiar los 64 bits del registro de punto flotante “ff” al registro entero “rd” ejecutando la instrucción “mfc1 rd, ff”.
- Importante: el número no se redondea, sino que se trunca.

Ejercicio 2

b)

El presente programa realiza una tarea ligeramente diferente a aquélla solicitada en la consigna ya que imprime en pantalla el valor tanto entero como en punto flotante de la superficie del triángulo, así como también la operación aritmética completa realizada para calcularlo.

```
            .data
control:    .word32 0x10000
data:       .word32 0x10008
msj_ing_1: .asciiz "Ingrese el valor de la base (punto flotante): "
msj_ing_2: .asciiz "Ingrese el valor de la altura (punto flotante): "
msj_calc:  .asciiz "Calculo de la superficie del triangulo:"
msj_op_1:  .asciiz "*"
msj_op_2:  .asciiz "/"
msj_igu:   .asciiz "----"
msj_lf:    .asciiz "\n"
sup:       .word 0

            .code
            lwu $t0, control($0)
            lwu $t1, data($0)
            daddi $t2, $0, 6
            sd $t2, 0($t0)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_ing_1
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 8
            sd $t2, 0($t0)
            l.d f0, 0($t1)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_ing_2
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 8
            sd $t2, 0($t0)
            l.d f1, 0($t1)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_calc
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 3
            s.d f0, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_op_1
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 3
            s.d f1, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_op_2
            sd $t3, 0($t1)
            sd $t2, 0($t0)
```

```
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t4, $0, 2
            daddi $t2, $0, 1
            sd $t4, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 4
            daddi $t3, $0, msj_igu
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            mtc1 $t4, f2
            cvt.d.l f2, f2
            mul.d f0, f0, f1
            div.d f0, f0, f2
            daddi $t2, $0, 3
            s.d f0, 0($t1)
            sd $t2, 0($t0)
            cvt.l.d f0, f0
            mfc1 $t4, f0
            daddi $t2, $0, 4
            daddi $t3, $0, msj_igu
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t3, $0, msj_lf
            sd $t3, 0($t1)
            sd $t2, 0($t0)
            daddi $t2, $0, 1
            sd $t4, 0($t1)
            sd $t2, 0($t0)
            sd $t4, sup($0)
            halt
```

Ejercicio 3

```
            .data
v:          .double -5, 4, -7.5, 12, 6.2
cant_pos:   .word 0
suma_pos:   .double 0
suma_v:     .double 0

            .code
            daddi $t0, $0, 0
            daddi $t1, $0, 5
            daddi $t2, $0, 0
            mtc1 $t0, f1
            cvt.d.l f1, f1
            mov.d f2, f1
            mov.d f3, f1
bucle:      l.d f0, v($t0)
            add.d f3, f3, f0
            c.le.d f0, f1
            bc1t sig_num
```

```
            add.d f2, f2, f0
            daddi $t2, $t2, 1
sig_num:    daddi $t0, $t0, 8
            daddi $t1, $t1, -1
            bnez $t1, bucle
            sd $t2, cant_pos($0)
            s.d f2, suma_pos($0)
            s.d f3, suma_v($0)
            halt
```

Parte 2: Pila y Subrutinas

Ejercicio 1

Para que el programa planteado en la consigna funcione correctamente, en la subrutina "potencia" debe reemplazarse la instrucción "bnez $a1, terminar", es decir, la primera del bucle "lazo", por la instrucción "beqz $a1, terminar".

a)

Básicamente, el programa propuesto en el presente ejercicio realiza la operación aritmética conocida como potencia, utilizando como base el valor almacenado en la dirección en memoria de datos etiquetada "base" y, como exponente, el número contenido en la dirección "exponente". Por su parte, el resultado obtenido es eventualmente guardado en la dirección "result". Por lo tanto, la fórmula matemática para el cálculo de dicho resultado podría plantearse del siguiente modo: $"result = base^{exponente}"$.

Para efectuar esta operación, al no disponer el procesador de una instrucción capaz de calcular una potencia en forma directa, la misma es lleva a cabo de manera indirecta. Este mecanismo implica la inicialización en 1 (uno) del registro destinado a almacenar el resultado, seguida de la ejecución de un bucle. La cantidad de iteraciones de éste será igual a "exponente": en cada una de ellas se multiplicará sucesivamente el contenido actual del aludido registro por el valor correspondiente a "base".

En particular, para los valores de prueba definidos en este escenario, es decir, 5 y 4 para "base" y "exponente" respectivamente, el número almacenado en "result" será igual a 625: $"result = base^{exponente} = 5^4 = 625"$.

Con respecto a la estructuración del código del programa, se puede observar la ausencia de directivas "ORG" y "END". En otras palabras, a diferencia de la arquitectura x86, no se requiere determinar ni asignar manualmente direcciones de inicio para las secciones de datos, programa principal y/o subrutinas de la memoria principal, ni tampoco establecer explícitamente el fin del programa. Sin embargo, aunque es posible intercambiar sin inconvenientes el orden de la declaración de las secciones de datos (".data") e instrucciones (".text"), dentro de esta última resulta obligatorio escribir siempre en primer lugar el código correspondiente al programa principal y luego las distintas subrutinas utilizadas. Esta restricción especial se debe a que la ejecución de cualquier programa en el procesador MIPS64 comenzará ineludiblemente a partir de la primera instrucción definida en la memoria de instrucciones de la computadora (sección ".code" del programa).

b)
La instrucción de transferencia de control “jal potencia” efectúa la invocación de la subrutina denominada “potencia”. Esta llamada involucra un salto incondicional hacia la dirección etiquetada o rotulada “potencia”, a partir de la cual se ubica “daddi $v0, $zero, 1”, siendo ésta la primera instrucción a ejecutar de la subrutina. Asimismo, se carga en el registro r31 o $ra una copia de la dirección de retorno asociada a la subrutina. Dicha dirección se encuentra contenida en el registro IP antes de ejecutarse la invocación de la mencionada subrutina y corresponde al comienzo de la instrucción “sd $v0, result($zero)”, localizada inmediatamente después de “jal potencia” en el programa principal.
La instrucción de transferencia de control “jr $ra” finaliza la ejecución de la subrutina denominada “potencia” y gestiona el mecanismo de retorno de la misma. El objetivo aquí consiste en garantizar que el programa pueda proseguir sin inconvenientes con su flujo de ejecución normal a partir de la instrucción “sd $v0, result($zero)”, ubicada en el programa principal inmediatamente a continuación de la instrucción “jal potencia”, la cual, como se recordará, llevó a cabo originalmente la invocación de la subrutina. Más precisamente, la tarea efectuada por la instrucción “jr $ra” se trata simplemente de un salto incondicional hacia la dirección contenida en el registro r31 o $ra. Este último, al no haber sido modificado en ningún momento dentro de la subrutina “potencia”, continúa conservando inalterable la dirección de retorno de la misma, cargada inicialmente por “jal potencia” y correspondiente al comienzo de “sd $v0, result($zero)”.

c)
Tal como se indica en la tabla definida en el ejercicio 4 del trabajo práctico anterior (práctica N°4), en la cual se establece la convención empleada para nombrar a los 32 registros enteros del MIPS64 y cuya consideración resulta especialmente relevante para la implementación de subrutinas, el registro r31 o $ra (Return Address) contiene la dirección de retorno en un llamado a una subrutina.
En este caso, tal como se indicó en el inciso previo, dicha dirección es cargada inicialmente por la instrucción “jal potencia” durante la invocación de la subrutina “potencia”. La dirección se encuentra contenida en el registro IP antes de ejecutarse la llamada y corresponde al comienzo de la instrucción “sd $v0, result($zero)”, localizada inmediatamente después de “jal potencia” en el programa principal.
Por su parte, los registros $a0 y $a1 son cargados con los valores almacenados en las direcciones “base” y “exponente” respectivamente. El objetivo aquí consiste en que, al invocarse la subrutina “potencia”, estas variables puedan enviarse a la misma como argumentos o parámetros de entrada por valor a través de estos dos registros. Tal como se encuentra implementada actualmente la subrutina, el valor del registro $a0 (variable “base”) puede ser cualquier número entero negativo, neutro o positivo representable con los 64 bits disponibles en dicho registro. Sin embargo, el valor del registro $a1 (variable “exponente”) debe ser mayor o igual a cero, ya que en caso contrario el bucle “lazo”, presente en la subrutina “potencia”, se ejecutará indefinidamente, desencadenando un comportamiento indeseado e incorrecto del programa. Cabe indicar que esta última restricción en realidad no debería existir para la operación matemática realizada aquí debido a que, en el campo de la aritmética, una potencia

admite para su cálculo exponentes tanto positivos como neutros y negativos.
Finalmente, el registro $v0 es utilizado por la subrutina "potencia" para almacenar en él su respectivo valor de retorno. Este último es establecido como parámetro de salida de la misma y devuelto por valor vía registro al programa principal que había realizado originalmente la invocación. Dicho parámetro contiene el resultado de la potencia, calculado en la subrutina y requerido para poder guardarlo en la dirección de la memoria de datos etiquetada "result".

d)
En el repertorio de instrucciones soportado por el simulador WinMIPS64 se establece claramente que, en cada invocación de una subrutina realizada mediante la ejecución de una instrucción "jal", el registro r31 o $ra se cargará ineludiblemente con la dirección de retorno correspondiente a la subrutina llamada. Esta carga, obviamente, implicará la sobreescritura y consecuente pérdida de cualquier valor que se encontrase previamente almacenado en dicho registro. Por lo tanto, ante un escenario de anidamiento de subrutinas como aquél planteado en la pregunta del presente inciso, resulta mandatorio, tal como se indica en la convención definida en el ejercicio 4 del trabajo práctico anterior (práctica N°4) para el uso de registros, preservar la dirección de retorno contenida en $ra antes de proceder a la invocación de una nueva subrutina dentro de otra actualmente en ejecución. Esta preservación debe efectuarse almacenando la aludida dirección en la memoria de pila de la computadora (apilamiento). Del mismo modo, a fin de respetar la convención para los registros del MIPS64 ya presentada y conocida, es obligatorio recurrir exclusivamente al registro $ra para retornar de cualquier subrutina a través de la ejecución de la instrucción "jr", aunque este requerimiento no se encuentre explicitado en el repertorio de instrucciones correspondiente al WinMIPS64.

e)
El presente programa realiza una tarea ligeramente diferente a aquélla solicitada en la consigna ya que no solo imprime en pantalla el resultado de la potencia, sino también la operación aritmética completa realizada para calcularlo.

```
            .data
control:    .word32 0x10000
data:       .word32 0x10008
msj_ing_1: .asciiz "Ingrese el valor de la base (entero): "
msj_ing_2: .asciiz "Ingrese el valor del exponente (entero): "
msj_calc:  .asciiz "Calculo de la potencia:"
msj_op:    .asciiz "^"
msj_igu:   .asciiz "----"
msj_lf:    .asciiz "\n"

            .code
            lwu $s0, control($0)
            lwu $s1, data($0)
            daddi $t0, $0, 6
            sd $t0, 0($s0)
            daddi $t0, $0, 4
            daddi $t1, $0, msj_ing_1
            sd $t1, 0($s1)
```

```
            sd $t0, 0($s0)
            daddi $t0, $0, 8
            sd $t0, 0($s0)
            ld $s2, 0($s1)
            daddi $t0, $0, 4
            daddi $t1, $0, msj_ing_2
            sd $t1, 0($s1)
            sd $t0, 0($s0)
            daddi $t0, $0, 8
            sd $t0, 0($s0)
            ld $s3, 0($s1)
            daddi $t0, $0, 4
            daddi $t1, $0, msj_calc
            sd $t1, 0($s1)
            sd $t0, 0($s0)
            daddi $t1, $0, msj_lf
            sd $t1, 0($s1)
            sd $t0, 0($s0)
            daddi $t0, $0, 1
            sd $s2, 0($s1)
            sd $t0, 0($s0)
            daddi $t0, $0, 4
            daddi $t1, $0, msj_op
            sd $t1, 0($s1)
            sd $t0, 0($s0)
            daddi $t1, $0, msj_lf
            sd $t1, 0($s1)
            sd $t0, 0($s0)
            daddi $t0, $0, 1
            sd $s3, 0($s1)
            sd $t0, 0($s0)
            daddi $t0, $0, 4
            daddi $t1, $0, msj_igu
            sd $t1, 0($s1)
            sd $t0, 0($s0)
            daddi $t1, $0, msj_lf
            sd $t1, 0($s1)
            sd $t0, 0($s0)
            daddi $a0, $s2, 0
            daddi $a1, $s3, 0
            jal potencia
            daddi $t0, $0, 1
            sd $v0, 0($s1)
            sd $t0, 0($s0)
            halt

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

f)

```
            .data
control:    .word32 0x10000
data:       .word32 0x10008
msj_ing:    .asciiz "Ingrese el valor del exponente (entero): "
msj_res:    .asciiz "Resultado del calculo aritmetico solicitado: "

            .code
            lwu $s0, control($0)
            lwu $s1, data($0)
            daddi $t0, $0, 6
            sd $t0, 0($s0)
            daddi $t0, $0, 4
            daddi $t1, $0, msj_ing
            sd $t1, 0($s1)
            sd $t0, 0($s0)
            daddi $t0, $0, 8
            sd $t0, 0($s0)
            ld $a1, 0($s1)
            daddi $t0, $0, 4
            daddi $t1, $0, msj_res
            sd $t1, 0($s1)
            sd $t0, 0($s0)
            daddi $a0, $0, 2
            daddi $s2, $a1, 0     ; debe preservarse el valor del
                                  ; exponente antes de invocar la
                                  ; subrutina “potencia” por primera
                                  ; vez
            jal potencia
            daddi $s3, $v0, 0
            daddi $a0, $0, 3
            daddi $a1, $s2, 0
            jal potencia
            daddi $a0, $s3, 0
            daddi $a1, $v0, 0
            jal suma
            daddi $t0, $0, 1
            sd $v0, 0($s1)
            sd $t0, 0($s0)
            halt

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

suma:       dadd $v0, $a0, $a1
            jr $ra
```

Ejercicio 2

a)

```
            .data
variable:   .word 0    # declaración añadida para probar el programa

            .code
            daddi $a0, $0, 5      # registro $t0 reemplazado por $a0
            daddi $a1, $0, 7      # registro $t1 reemplazado por $a1
            jal subrutina
            sd $v0, variable($0)  # registro $t2 reemplazado por $v0
            halt

subrutina:  daddi $t4, $0, 2      # sin cambios
            dmul $a0, $a0, $t4    # registro $t0 reemplazado por $a0
            dmul $a1, $a1, $t4    # registro $t1 reemplazado por $a1
            dadd $v0, $a1, $a0    # registro $t2 reemplazado por $v0
            jr $ra
```

b)

```
            .code
            daddi $s0, $0, tabla  # debe preservarse el valor de la
                                  # dirección de "tabla" antes de
                                  # invocar la subrutina "subrutina"
            daddi $a0, $s0, 0
            jal subrutina
            daddi $t0, $0, 10
            daddi $t1, $0, 0
loop:       bnez $t0, fin
            ld $t2, 0($s0)        # registro $a0 reemplazado por $s0
            dadd $t1, $t1, $t2
            daddi $t0, $t0, -1
            daddi $s0, $s0, 8     # registro $a0 reemplazado por $s0
                                  # "dadd" reemplazado por "daddi"
            j loop
fin:        halt
```

c)

```
            .data
variable:   .word 0    # declaración añadida para probar el programa

            .code
            daddi $s0, $0, 5      # debe preservarse el valor 5 antes
                                  # de invocar la subrutina "subrutina"
            daddi $a0, $s0, 0
            daddi $a1, $0, 7
            jal subrutina
            dmul $t2, $s0, $v0    # registro $a0 reemplazado por $s0
            sd $t2, variable($0)
            halt

subrutina:  daddi $t4, $0, 2      # sin cambios
            dmul $a0, $a0, $t4    # registro $t0 reemplazado por $a0
            dmul $a1, $a1, $t4    # registro $t1 reemplazado por $a1
```

```
            dadd $v0, $a1, $a0    # registro $t2 reemplazado por $v0
            jr $ra
```

d)

```
            .code
            daddi $s0, $0, 10     # dimensión
            daddi $s1, $0, 0      # contador
            daddi $s2, $0, 0      # desplazamiento
            # deben preservarse la dimensión, el contador y el
            # desplazamiento antes de invocar la subrutina "espar"
loop:       beqz $s0, fin         # registro $t0 reemplazado por $s0
                                  # "bnez" reemplazado por "beqz"
            ld $a0, tabla ($s2)   # registro $t2 reemplazado por $s2
            jal espar
            bnez $v0, seguir
            daddi $s1, $s1, 1     # registro $t1 reemplazado por $s1
                                  # "dadd1" reemplazado por "daddi"
seguir:     daddi $s0, $s0, -1    # registro $t0 reemplazado por $s0
                                  # instrucción anterior
                                  # "daddi $t2, $t2, 1" eliminada porque
                                  # se la consideró incorrecta
            daddi $s2, $s2, 8     # registro $t2 reemplazado por $s2
                                  # "dadd" reemplazado por "daddi"
            j loop
            sd $s1, resultado($0) # registro $t1 reemplazado por $s1
                                  # en la instrucción original debía
                                  # utilizarse el registro $t1 en lugar
                                  # del $t2
fin:        halt
```

Ejercicio 4

Las dos líneas de comentarios iniciales deben eliminarse ya que no poseen ninguna relación con la tarea realizada por el programa propuesto en la presente consigna.

```
            .code
            # inicializar el puntero al tope de la pila (registro $sp)
            # la pila comienza en el tope de la memoria de datos (*)
            daddi $sp, $0, 0x400
            # se apilarán 2 valores de 8 bytes cada uno: $t0 y $t1
            # (en este orden, desde la base hasta el tope de la pila)
            daddi $sp, $sp, -16
            daddi $t0, $0, 5
            daddi $t1, $0, 8
            sd $t0, 8($sp)        # push $t0
            sd $t1, 0($sp)        # push $t1
            # se desapilarán los valores originales de $t0 y $t1,
            # almacenados en la pila (apilados) al inicio del programa,
            # pero invirtiendo el orden en el cual dichos valores fueron
            # apilados de manera tal que $t0 se cargue con el valor
            # inicial de $t1, y viceversa
            ld $t0, 0($sp)        # pop $t0
            ld $t1, 8($sp)        # pop $t1
            daddi $sp, $sp, 16
            halt
```

(*) La configuración inicial de la arquitectura del WinMIPS64 establece que el procesador posee un bus de direcciones de 10 bits para la memoria de datos. Por lo tanto, la mayor dirección dentro de la memoria de datos será de $2^{10} = 1024 = 400_{16}$.

Ejercicio 5

a)

Pasaje de parámetros por referencia y registros:

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

            .code
            daddi $a0, $0, base
            daddi $a1, $0, exponente
            jal potencia
            sd $v0, result($0)
            halt

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

b)

Pasaje de parámetros por valor y pila:

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

            .code
            daddi $sp, $0, 0x400
            daddi $sp, $sp, -16
            ld $t0, base($0)
            ld $t1, exponente($0)
            sd $t0, 8($sp)
            sd $t1, 0($sp)
            jal potencia
            sd $v0, result($0)
            ld $t1, 0($sp)
            ld $t0, 8($sp)
            daddi $sp, $sp, 16
            halt

potencia:  daddi $v0, $0, 1
            ld $t0, 8($sp)
            ld $t1, 0($sp)
```

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

c)

Pasaje de parámetros por referencia y pila:

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

            .code
            daddi $sp, $0, 0x400
            daddi $sp, $sp, -16
            daddi $t0, $0, base
            daddi $t1, $0, exponente
            sd $t0, 8($sp)
            sd $t1, 0($sp)
            jal potencia
            sd $v0, result($0)
            ld $t1, 0($sp)
            ld $t0, 8($sp)
            daddi $sp, $sp, 16
            halt

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

Ejercicio 6

a)

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

# $v0: devuelve 1 si $a0 es par y 0 dlc (de lo contrario)
# $a0: número entero cualquiera
# debe preservarse en la pila la dirección de retorno de la subrutina
# “espar” (registro $ra) antes de que ésta invoque la subrutina anidada
# “esimpar”
# no corresponde emplear el registro $s0 en este caso ya que la
# llamada a la subrutina “esimpar” precede el comienzo de la
# utilización del mismo, así que puede recurrirse en su lugar sin
# inconvenientes al registro $t0; además de que, en el código original,
```

```
# debería preservarse en la pila el valor original del registro $s0
# al comienzo de la subrutina "espar"
espar:      daddi $sp, $sp, -8
            sd $ra, 0($sp)
            jal esimpar
            # truco: espar = 1 - esimpar
            daddi $t0, $0, 1
            dsub $v0, $t0, $v0
            ld $ra, 0($sp)
            daddi $sp, $sp, 8
            jr $ra
```

b)

```
# $v0: devuelve la cantidad de bits 0 que tiene un número de 64 bits
# $a0: número entero cualquiera
# deben preservarse los valores del número a analizar (registro $a0),
# el contador (registro $t0) y la longitud en bits del número (registro
# $t1) antes de invocar la subrutina anidada "espar", para lo cual
# habrá de recurrirse al conjunto de registros $s y la pila
# también debe preservarse en la pila la dirección de retorno de la
# subrutina "cant0" (registro $ra) antes de que ésta invoque la
# subrutina anidada "espar"
cant0:      daddi $sp, $sp, -32
            sd $ra, 24($sp)
            sd $s0, 16($sp)
            sd $s1, 8($sp)
            sd $s2, 0($sp)
            daddi $s0, $0, 0
            daddi $s1, $0, 64
            daddi $s2, $a0, 0
loop:       daddi $a0, $s2, 0
            jal espar
            dadd $s0, $s0, $v0
            # desplazo a la derecha para quitar el último bit
            dsrl $s2, $s2, 1
            daddi $s1, $s1, -1
            bnez $s1, loop
            ld $s2, 0($sp)
            ld $s1, 8($sp)
            ld $s0, 16($sp)
            ld $ra, 24($sp)
            daddi $sp, $sp, 32
            jr $ra
```

c)

```
# $v0: volumen de un cubo
# $a0: longitud del lado del cubo
# no existe invocación alguna a ninguna subrutina anidada dentro de
# "vol", por lo que $ra no será sobrescrito en ningún momento y no es
# necesario el uso de $s0 (puede emplearse directamente $v0 en su
# lugar); en consecuencia, no correspondería preservarlos en la pila
# al comienzo de la subrutina
# se eliminaron las instrucciones "dadd $s0, $0, $a0" y "daddi $v0,
$s0, 0" porque se las consideraron incorrecta e innecesaria
```

```
# respectivamente para el cálculo del volumen del cubo
vol:        dmul $v0, $a0, $a0
            dmul $v0, $v0, $a0
            jr $ra

# $v0: diferencia de volumen de los cubos
# $a0: longitud del lado del cubo más grande
# $a1: longitud del lado del cubo más chico
# debe preservarse el valor de la longitud del lado del cubo más chico
# (registro $a1) y el volumen del cubo más grande (registro
# $t0) antes de la primera y segunda invocación de la subrutina
# anidada "vol" respectivamente, para lo cual habrá de recurrirse al
# registro $s0 y la pila
# también debe preservarse en la pila la dirección de retorno de la
# subrutina "diffvol" (registro $ra) antes de que ésta invoque la
# subrutina anidada "vol"
diffvol:    daddi $sp, $sp, -16
            sd $ra, 8($sp)
            sd $s0, 0($sp)
            daddi $s0, $a1, 0
            jal vol
            # se invirtió el orden de estas dos instrucciones con
            # respecto al programa original para posibilitar el uso
            # de un único registro $s en lugar de dos; no obstante,
            # el funcionamiento general permanece inalterable
            daddi $a0, $s0, 0
            daddi $s0, $v0, 0
            jal vol
            dsub $v0, $s0, $v0
            ld $s0, 0($sp)
            ld $ra, 8($sp)
            daddi $sp, $sp, 16
            jr $ra
```

Ejercicio 7

a)

```
            .data
N:          .word 5    # debe ser un número mayor o igual a cero
result:     .word 0

            .code
            ld $a0, N($0)
            jal factorial
            sd $v0, result($0)
            halt

factorial: daddi $v0, $0, 1
bucle_f:    slti $t0, $a0, 2
            bnez $t0, fin_f
            dmul $v0, $v0, $a0
            daddi $a0, $a0, -1
            j bucle_f
fin_f:      jr $ra
```

b)

El cálculo del número combinatorio o coeficiente binomial no debe efectuarse como se indica en la consigna, sino del siguiente modo:

comb(m, n) = m! / [n! * (m-n)!] → m debe ser un número mayor o igual a n

```
            .data
m:          .word 13   # debe ser un número mayor o igual a "n"
n:          .word 3    # debe ser un número mayor o igual a cero
result:     .word 0

            .code
            daddi $sp, $0, 0x400
            ld $a0, m($0)
            ld $a1, n($0)
            jal comb
            sd $v0, result($0)
            halt

comb:       daddi $sp, $sp, -32
            sd $ra, 24($sp)
            sd $s0, 16($sp)
            sd $s1, 8($sp)
            sd $s2, 0($sp)
            daddi $s0, $a0, 0
            daddi $s1, $a1, 0
            jal factorial
            daddi $s2, $v0, 0
            daddi $a0, $s1, 0
            jal factorial
            ddiv $s2, $s2, $v0
            dsub $a0, $s0, $s1
            jal factorial
            ddiv $v0, $s2, $v0
            ld $s2, 0($sp)
            ld $s1, 8($sp)
            ld $s0, 16($sp)
            ld $ra, 24($sp)
            daddi $sp, $sp, 32
            jr $ra

factorial: daddi $v0, $0, 1
bucle_f:    slti $t0, $a0, 2
            bnez $t0, fin_f
            dmul $v0, $v0, $a0
            daddi $a0, $a0, -1
            j bucle_f
fin_f:      jr $ra
```

Ejercicio 8

a), b), c) y d)

Se realizaron ciertos cambios con respecto a los requisitos planteados en las consignas para la implementación de algunas de las subrutinas. Más precisamente, aunque esto no era necesario, se recurrió a la subrutina “longitud”, correspondiente al inciso a), para calcular la longitud de la cadena “cadena” a fin de lograr recorrerla desde el primer carácter hasta alcanzar el último de ellos.

```
            .data
cadena:     .asciiz "Prueba Ejercicio 8ABCD."
cad_voc:    .asciiz "AEIOUaeiou"
vocales:    .word 0

            .code
            daddi $sp, $0, 0x400
            daddi $a0, $0, cadena
            jal cant_voc
            sd $v0, vocales($0)
            halt

cant_voc:   daddi $sp, $sp, -32         # inciso d)
            sd $ra, 24($sp)
            sd $s0, 16($sp)
            sd $s1, 8($sp)
            sd $s2, 0($sp)
            daddi $s0, $a0, 0
            jal longitud
            daddi $s1, $v0, 0
            daddi $s2, $0, 0
bucle_v:    beqz $s1, fin_v
            lbu $a0, 0($s0)
            jal es_vocal
            dadd $s2, $s2, $v0
            daddi $s0, $s0, 1
            daddi $s1, $s1, -1
            j bucle_v
fin_v:      daddi $v0, $s2, 0
            ld $s2, 0($sp)
            ld $s1, 8($sp)
            ld $s0, 16($sp)
            ld $ra, 24($sp)
            daddi $sp, $sp, 32
            jr $ra

longitud:   daddi $v0, $0, 0            # inciso a)
bucle_l:    lbu $t0, 0($a0)
            beqz $t0, fin_l
            daddi $v0, $v0, 1
            daddi $a0, $a0, 1
            j bucle_l
fin_l:      jr $ra
```

```
contiene:  daddi $v0, $0, 0          # inciso b)
bucle_c:   lbu $t0, 0($a0)
           beqz $t0, fin_c
           bne $t0, $a1, sig_c
           daddi $v0, $0, 1
           j fin_c
sig_c:     daddi $a0, $a0, 1
           j bucle_c
fin_c:     jr $ra

es_vocal:  daddi $sp, $sp, -8        # inciso c)
           sd $ra, 0($sp)
           daddi $a1, $a0, 0
           daddi $a0, $0, cad_voc    # uso de variable global
           jal contiene
           ld $ra, 0($sp)
           daddi $sp, $sp, 8
           jr $ra
```

Ejercicio 9

a)

```
           .data
tabla:     .word 10, -8, 14, 0, 9, -5, -20, 2, 6
total:     .word 0

           .code
           daddi $a0, $0, tabla
           daddi $a1, $0, 9
           jal suma
           sd $v0, total($0)
           halt

suma:      daddi $v0, $0, 0
bucle_s:   ld $t0, 0($a0)
           dadd $v0, $v0, $t0
           daddi $a0, $a0, 8
           daddi $a1, $a1, -1
           bnez $a1, bucle_s
           jr $ra
```

b)

```
           .data
tabla:     .word 10, -8, 14, 0, 9, -5, -20, 2, 6
cant_pos:  .word 0

           .code
           daddi $sp, $0, 0x400
           daddi $a0, $0, tabla
           daddi $a1, $0, 9
           jal positivos
           sd $v0, cant_pos($0)
           halt
```

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

positivos: daddi $sp, $sp, -32
           sd $ra, 24($sp)
           sd $s0, 16($sp)
           sd $s1, 8($sp)
           sd $s2, 0($sp)
           daddi $s0, $a0, 0
           daddi $s1, $a1, 0
           daddi $s2, $0, 0
bucle_p:   ld $a0, 0($s0)
           jal es_pos
           dadd $s2, $s2, $v0
           daddi $s0, $s0, 8
           daddi $s1, $s1, -1
           bnez $s1, bucle_p
           daddi $v0, $s2, 0
           ld $s2, 0($sp)
           ld $s1, 8($sp)
           ld $s0, 16($sp)
           ld $ra, 24($sp)
           daddi $sp, $sp, 32
           jr $ra
```

Ejercicio 10

a)

```
           .data
valor:     .word 5
result:    .word 0

           .code
           daddi $sp, $0, 0x400
           ; inicializar el puntero al tope de la pila (registro SP)
           ; la pila comienza en el tope de la memoria de datos (*)
           ld $a0, valor($0)
           jal factorial
           sd $v0, result($0)
           halt

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

(*) La configuración inicial de la arquitectura del WinMIPS64 establece que el procesador posee un bus de direcciones de 10 bits para la memoria de datos. Por lo tanto, la mayor dirección dentro de la memoria de datos será de $2^{10} = 1024 = 400_{16}$.

La subrutina “factorial”, implementada específicamente para la resolución del presente ejercicio, recibe el número entero “n”, contenido en la dirección “valor” y cuyo factorial se desea calcular, como parámetro de entrada por valor a través del registro $a0. Por otro lado, el factorial obtenido, el cual se almacenará en la dirección “result”, es retornado por la subrutina como parámetro de salida por valor por medio del registro $v0.
Este programa puede encontrarse en su versión original, idéntica a aquélla aquí presentada, incluyendo tanto la sección de datos como el programa principal y la subrutina “factorial”, en la explicación ofrecida para este trabajo práctico en el sitio web oficial de la cátedra y disponible en el mismo para su respectiva descarga.

b)
Si se desea continuar empleando la técnica de programación recursiva para escribir la subrutina “factorial”, resultará absolutamente imposible implementarla sin utilizar una pila. Básicamente, esta última se trata de la única estructura de almacenamiento de datos que presenta el dinamismo, la flexibilidad y la capacidad requeridas para preservar los valores de los registros $ra y $s0.
Como se observará, “factorial” modifica $ra dado que llama a otra subrutina (en este caso, a sí misma) antes de finalizar, mientras que altera $s0 al copiar en él el valor del factor para el cálculo parcial del factorial correspondiente a la ejecución actual de “factorial”. Ergo, tanto para respetar la convención como para garantizar el correcto funcionamiento del propio programa, la preservación de estos registros constituye una tarea obligatoria. Para cumplirla, los mismos se apilarán en repetidas ocasiones al comienzo de cada una de las sucesivas ejecuciones anidadas y recursivas de “factorial” y, del mismo modo, procederán a desapilarse en reiteradas veces al final de cada ejecución de esta subrutina.
No obstante, si la programación de la subrutina fuera iterativa en lugar de recursiva, las sucesivas llamadas anidadas que “factorial” se realiza a sí misma se remplazarían en su totalidad por la ejecución reiterada de un único lazo o bucle, sin que se encuentre involucrada en la misma ningún tipo de anidamiento de subrutinas. De esta manera, si se lograra escribir una nueva versión de “factorial” tal que durante su ejecución no se invoque a sí misma ni a ninguna otra subrutina, no se modificarán en ella el registro $ra ni ninguno de los registros $s. En consecuencia, no será necesario preservar ninguno de ellos en la memoria de pila de la computadora, resultando absolutamente innecesario el uso de esta última estructura de datos.
Afortunadamente, ya se dispone de una versión alternativa de la subrutina “factorial”, programada en forma iterativa en lugar de recursiva a fin de evitar por completo el uso de la memoria de pila de la computadora. La misma fue desarrollada en el inciso a) del ejercicio 7) de la presente práctica. No obstante, a continuación se la expondrá nuevamente para facilitar su comparación con la implementación de la subrutina recursiva:

```
            .data
valor:      .word 5    # debe ser un número mayor o igual a cero
result:     .word 0

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

factorial: daddi $v0, $0, 1
bucle_f:   slti $t0, $a0, 2
           bnez $t0, fin_f
           dmul $v0, $v0, $a0
           daddi $a0, $a0, -1
           j bucle_f
fin_f:     jr $ra
```

Parte 3: Ejercicios tipo parcial

Ejercicio 2

```
            .data
control:    .word32 0x10000
data:       .word32 0x10008
colores:    .word32 0xFF, 0xFF0000     # colores rojo y azul en RGB

            .code
            daddi $sp, $0, 0x400
            lwu $t0, control($0)
            daddi $t1, $0, 7
            sd $t1, 0($t0)
            daddi $t0, $0, colores
            lwu $a0, 0($t0)
            lwu $a1, 4($t0)
            jal imprimir
            halt

imprimir:  daddi $sp, $sp, -32
           sd $ra, 24($sp)
           sd $s0, 16($sp)
           sd $s1, 8($sp)
           sd $s2, 0($sp)
           daddi $a2, $a1, 0
           daddi $a1, $a0, 0
           daddi $s0, $0, 0
           daddi $s1, $a1, 0
           daddi $s2, $a2, 0
bucle_i:   daddi $a0, $s0, 0
           jal fila_alt
           daddi $s0, $s0, 1
           andi $t0, $s0, 1
           bnez $t0, f_impar
```

```
f_par:      daddi $a1, $s1, 0
            daddi $a2, $s2, 0
            j sig_fila
f_impar:    daddi $a1, $s2, 0
            daddi $a2, $s1, 0
sig_fila:   slti $t0, $s0, 50
            bnez $t0, bucle_i
            ld $s2, 0($sp)
            ld $s1, 8($sp)
            ld $s0, 16($sp)
            ld $ra, 24($sp)
            daddi $sp, $sp, 32
            jr $ra

fila_alt:   lwu $t0, control($0)        # uso de variables globales
            lwu $t1, data($0)
            daddi $t2, $0, 0
            daddi $t3, $a1, 0
            sb $a0, 4($t1)
bucle_f:    sb $t2, 5($t1)
            sw $t3, 0($t1)
            daddi $t4, $0, 5
            sd $t4, 0($t0)
            daddi $t2, $t2, 1
            andi $t4, $t2, 1
            bnez $t4, c_impar
c_par:      daddi $t3, $a1, 0
            j sig_punto
c_impar:    daddi $t3, $a2, 0
sig_punto:  slti $t4, $t2, 50
            bnez $t4, bucle_f
            jr $ra
```