PRACTICA 1

Subrutinas y pasaje de parámetros

Objetivos: Comprender la utilidad de las subrutinas y la comunicación con el programa principal a través de una pila. Escribir programas en el lenguaje assembly del simulador MSX88. Ejecutarlos y verificar los resultados, analizando el flujo de información entre los distintos componentes del sistema.

1) *** Repaso de uso de la pila** Si el registro SP vale 8000h al comenzar el programa, indicar el valor del registro SP **luego de ejecutar** cada una de las instrucciones de la tabla, **en el orden en que aparecen**. Indicar, de la misma forma, los valores de los registros AX y BX.

	Instrucción	**Valor del registro SP**	**AX**	**BX**
1	`mov ax,5`			
2	`mov bx,3`			
3	`push ax`			
4	`push ax`			
5	`push bx`			
6	`pop bx`			
7	`pop bx`			
8	`pop ax`			

2) *** Llamadas a subrutinas y la pila** Si el registro SP vale 8000h al comenzar el programa, indicar el valor del registro SP **luego** de ejecutar cada instrucción. Considerar que el programa comienza a ejecutarse con el IP en la dirección 2000h, es decir que la primera instrucción que se ejecuta es la de la línea 5 (`push ax`).
Nota: Las sentencias `ORG` y `END` no son instrucciones sino indicaciones al compilador, por lo tanto no se ejecutan.

#	**Instrucción**	**Valor del registro SP**
1	`org 3000h`	`-----`
2	`rutina: mov bx,3`	
3	`ret`	
4	`org 2000h`	`-----`
5	`push ax`	
6	`call rutina`	
7	`pop bx`	
8	`hlt`	
9	`end`	`-----`

3) *** Llamadas a subrutinas y dirección de retorno**

1. Si el registro SP vale 8000h al comenzar el programa, **indicar el contenido de la pila luego** de ejecutar cada instrucción. Si el contenido es desconocido/basura, indicarlo con el símbolo **?**. Considerar que el programa comienza a ejecutarse con el IP en la dirección 2000h, es decir que la primera instrucción que se ejecuta es la

de la línea 5 (`call rut`). Se provee la ubicación de las instrucciones en memoria, para poder determinar la dirección de retorno de la rutina.

Nota: Las sentencias `ORG` y `END` no son instrucciones sino indicaciones al compilador, por lo tanto no se ejecutan ni tienen ubicación en memoria.

2. Explicar detalladamente:
   a) Las acciones que tienen lugar al ejecutarse la instrucción `call rut`
   b) Las acciones que tienen lugar al ejecutarse la instrucción `ret`

```
      org 3000h
rut:  mov bx,3     ; Dirección 3000h
      ret          ; Dirección 3002h

      org 2000h
      call rut     ; Dirección 2000h
      add cx,5     ; Dirección 2002h
      call rut     ; Dirección 2004h
      hlt          ; Dirección 2006h
      end
```

4) * Tipos de Pasajes de Parámetros Indicar con un tilde, para los siguientes ejemplos, si el pasaje del parámetro es por registro o pila, y por valor o referencia;

	Código	Registro	Pila	Valor	Referencia
a)	`mov ax,5` `call subrutina`				
b)	`mov dx, offset A` `call subrutina`				

c)	`mov bx, 5`
	`push bx`
	`call subrutina`
	`pop bx`				

d)	`mov cx, offset A`
	`push cx`
	`call subrutina`
	`pop cx`				

e)	`mov dl, 5` `call subrutina`				
f)	`call subrutina` `mov A, dx`				

5) Cálculo de A+B-C. Pasaje de parámetros a través de registros.

En este ejercicio, programarás tus primeras subrutinas. Las subrutinas recibirán tres parámetros A, B y C, y realizarán un cálculo muy simple, A+B-C, cuyo resultado deben retornar. Si bien en general no tendría sentido escribir una subrutina para una cuenta tan simple que puede implementarse con dos instrucciones, esta simplificación permite concentrarse en los aspectos del pasaje de parámetros.

a) Escribir un programa que dados los valores etiquetados como A, B y C y almacenados en la memoria de datos, calcule A+B-C y guarde el resultado en la memoria con etiqueta D, **sin utilizar subrutinas**.
b) Escribir un programa como en a) pero ahora el cálculo y el almacenamiento del resultado debe realizarse en una subrutina llamada calculo, sin recibir ni devolver parámetros, es decir, utilizando A, B, C y D como variables globales. Si bien esta técnica no está recomendada, en ejercicio sirve para ver sus diferencias con el uso de parámetros.
c) Volver a escribir el programa, pero ahora con una subrutina que reciba A, B y C por valor a través de los registros AX, BX y CX, calcule AX+BX-CX, y devuelva el resultado por valor en el registro DX. El programa principal debe llamar a la subrutina y luego guardar el resultado en la memoria con etiqueta D
d) Si tuviera que realizar el cálculo dos veces con números distintos, por ejemplo, unos guardados en variables A1, B1, C1 y otros guardados en variables A2, B2, C2, ¿podrían reutilizarse las subrutinas del inciso b) sin modificarse? ¿y las del inciso c?

```
;Memoria de Datos
      org 1000h
  A   DW 5h
  B   DW 6h
  C   DW 2h
  D   DW ?
;Memoria de programa
      org 2000h
      ….      ; COMPLETAR
end
```

6) * Multiplicación de números sin signo. Pasaje de parámetros a través de registros.
El simulador no posee una instrucción para multiplicar números. Escribir un programa para multiplicar los números NUM1 y NUM2, y guardar el resultado en la variable RES
a) Sin hacer llamados a subrutinas, resolviendo el problema desde el programa principal;
b) Llamando a una subrutina `MUL` para efectuar la operación, pasando los parámetros por **valor** desde el programa principal a través de **registros** y devolviendo el resultado a través de un **registro** por **valor**.
c) Llamando a una subrutina `MUL`, pasando los parámetros por **referencia** desde el programa principal a través de registros, y devolviendo el resultado a través de un **registro** por **valor**.

7) Multiplicación de números sin signo. Pasaje de parámetros a través de Pila.
El programa de abajo utiliza una subrutina para multiplicar dos números, pasando los parámetros por valor para NUM1 y NUM2, y por referencia (RES), en ambos casos a través de la **pila**. Analizar su contenido y contestar.

a) ¿Cuál es el modo de direccionamiento de la instrucción MOV AX, [BX]? ¿Qué se copia en el registro AX en este caso?
b) ¿Qué función cumple el registro temporal **ri** que aparece al ejecutarse una instrucción como la anterior?
c) ¿Qué se guarda en AX al ejecutarse MOV AX, OFFSET RES?
d) ¿Cómo se pasa la variable RES a la pila, por valor o por referencia? ¿Qué ventaja tiene esto?
e) ¿Cómo trabajan las instrucciones PUSH y POP?

Observaciones:
- Los contenidos de los registros AX, BX, CX y DX antes y después de ejecutarse la subrutina son iguales, dado que al comienzo se almacenan en la pila para poder utilizarlos sin perder la información que contenían antes del llamado. Al finalizar la subrutina, los contenidos de estos registros son restablecidos desde la pila.
- El programa sólo puede aplicarse al producto de dos números mayores que cero.

```
      ORG  3000H  ; Subrutina MUL
MUL:  PUSH BX     ; preservar registros
      PUSH CX
      PUSH AX
      PUSH DX
      MOV  BX, SP
      ADD  BX, 12 ; corrección por el IP(2),
                  ; RES(2) y los 4 registros(8)
      MOV  CX, [BX]  ; cx = num2
      ADD  BX, 2     ; bx apunta a num1
      MOV  AX, [BX]  ; ax = num1
      SUB  BX, 4     ; bx apunta a la dir de
                     ; resultado
      MOV  BX, [BX]  ; guardo
      MOV  DX, 0
SUMA: ADD  DX, AX
      DEC  CX
      JNZ  SUMA
      MOV  [BX], DX  ; guardar resultado
      POP  DX        ;restaurar registros
      POP  AX
      POP  CX
      POP  BX
      RET
```

```
      ORG 1000H ; Memoria de datos
NUM1  DW    5H
NUM2  DW    3H
RES   DW    ?

      ORG 2000H; Prog principal
          ; parámetros
          MOV   AX, NUM1
          PUSH  AX
          MOV   AX, NUM2
          PUSH  AX
          MOV   AX, OFFSET RES
          PUSH  AX
          CALL  MUL
          ; desapilar parámetros
          POP   AX
          POP   AX
          POP   AX
          HLT
          END
```

Analizar el siguiente diagrama de la pila y verificarlo con el simulador:

DIRECCIÓN DE MEMORIA	CONTENIDO
7FF0H	DL
	DH
7FF2H	AL
	AH
7FF4H	CL
	CH
7FF6H	BL
	BH

7FF8H	IP RET. L
	IP RET. H
7FFAH	DIR. RES L
	DIR. RES H
7FFCH	NUM2 L
	NUM2 H
7FFEH	NUM1 L
	NUM1 H
8000H	——

8) Subrutinas para realizar operaciones con cadenas de caracteres

a) Escribir una subrutina LONGITUD que cuente el número de caracteres de una cadena de caracteres terminada en cero (00H) almacenada en la memoria. La cadena se pasa a la subrutina por referencia vía registro, y el resultado se retorna por valor también a través de un registro.
Ejemplo: la longitud de 'abcd'00h es 4 (el 00h final no cuenta)

b) Escribir una subrutina CONTAR_MIN que cuente el número de letras minúsculas de la 'a' a la 'z' de una cadena de caracteres terminada en cero almacenada en la memoria. La cadena se pasa a la subrutina por referencia vía registro, y el resultado se retorna por valor también a través de un registro.
Ejemplo: CONTAR_MIN de 'aBcDE1#!' debe retornar 2.

c) * Escriba la subrutina ES_VOCAL, que determina si un carácter es vocal o no, ya sea mayúscula o minúscula. La rutina debe recibir el carácter por valor vía registro, y debe retornar, también vía registro, el valor 0FFH si el carácter es una vocal, o 00H en caso contrario.
Ejemplos: ES_VOCAL de 'a' o 'A' debe retornar 0FFh y ES_VOCAL de 'b' o de '4' debe retornar 00h

d) * Usando la subrutina anterior escribir la subrutina CONTAR_VOC, que recibe una cadena terminada en cero por referencia a través de un registro, y devuelve, en un registro, la cantidad de vocales que tiene esa cadena.
Ejemplo: CONTAR_VOC de 'contar1#!' debe retornar 2

e) Escriba la subrutina CONTAR_CAR que cuenta la cantidad de veces que aparece un carácter dado en una cadena terminada en cero. El carácter a buscar se debe pasar por valor mientras que la cadena a analizar por referencia, ambos a través de la pila.
Ejemplo: CONTAR_CAR de 'abbcde!' y 'b' debe retornar 2, mientras que CONTAR_CAR de 'abbcde!' y 'z' debe retornar 0.

f) Escriba la subrutina REEMPLAZAR_CAR que reciba dos caracteres (ORIGINAL y REEMPLAZO) por valor a través de la pila, y una cadena terminada en cero también a través de la pila. La subrutina debe reemplazar el carácter ORIGINAL por el carácter REEMPLAZO.

9) Subrutinas para realizar rotaciones

a) Escribir una subrutina ROTARIZQ que haga **una** rotación hacia la izquierda de los bits de un byte almacenado en la memoria. Dicho byte debe pasarse por valor desde el programa principal a la subrutina a través de registros y por referencia. No hay valor de retorno, sino que se modifica directamente la memoria.

Una rotación a izquierda de un byte se obtiene moviendo cada bit a la izquierda, salvo por el último que se mueve a la primera posición. Por ejemplo al rotar a la izquierda el byte **1**0010100 se obtiene 00101001**0**, y al rotar a la izquierda **0**1101011 se obtiene 11010110**1**.

Para rotar a la izquierda un byte, se puede multiplicar el número por 2, o lo que es lo mismo sumarlo a sí mismo. Por ejemplo (verificar):

01101011
\+ **0**1101011
11010110 (CARRY=0)

Entonces, la instrucción `add ah, ah` permite hacer una rotación a izquierda. No obstante, también hay que tener en cuenta que si el bit más significativo es un 1, el carry debe llevarse al bit menos significativo, es decir, se le debe sumar 1 al resultado de la primera suma.

10010100
\+ **1**0010100
00101000 (CARRY=1)
\+ 00000001
00101001

b) Usando la subrutina ROTARIZQ del ejercicio anterior, escriba una subrutina ROTARIZQ_N que realice N rotaciones a la izquierda de un byte. La forma de pasaje de parámetros es la misma, pero se agrega el parámetro N que se recibe por valor y registro. Por ejemplo, al rotar a la izquierda 2 veces el byte **10**010100, se obtiene el byte 01010010.

c) * Usando la subrutina ROTARIZQ_N del ejercicio anterior, escriba una subrutina ROTARDER_N que sea similar pero que realice N rotaciones hacia la **derecha**.

Una rotación a derecha de N posiciones, para un byte con 8 bits, se obtiene rotando a la izquierda 8 - *N* posiciones. Por ejemplo, al rotar a la derecha 6 veces el byte 10**010100** se obtiene el byte **01010010**, que es equivalente a la rotación a la izquierda de 2 posiciones del ejemplo anterior.

d) Escriba la subrutina ROTARDER del ejercicio anterior, pero sin usar la subrutina ROTARIZ. Compare qué ventajas tiene cada una de las soluciones.

10) SWAP Escribir una subrutina SWAP que intercambie dos datos de 16 bits almacenados en memoria. Los parámetros deben ser pasados por referencia desde el programa principal a través de la pila.

Para hacer este ejercicio, tener en cuenta que los parámetros que se pasan por la pila son las *direcciones* de memoria, por lo tanto para acceder a los datos a intercambiar se requieren accesos indirectos, además de los que ya se deben realizar para acceder a los parámetros de la pila.

11) Subrutinas de cálculo

a) Escriba la subrutina DIV que calcule el resultado de la división entre 2 números positivos. Dichos números deben pasarse por valor desde el programa principal a la subrutina a través de la pila. El resultado debe devolverse también a través de la pila por valor.

b) * Escriba la subrutina RESTO que calcule el resto de la división entre 2 números positivos. Dichos números deben pasarse por valor desde el programa principal a la subrutina a través de registros. El resultado debe devolverse también a través de un registro por referencia.

c) Escribir un programa que calcule la suma de dos números de 32 bits almacenados en la memoria sin hacer llamados a subrutinas, resolviendo el problema desde el programa principal.

d) Escribir un programa que calcule la suma de dos números de 32 bits almacenados en la memoria llamando a una subrutina SUM32, que reciba los parámetros de entrada por valor a través de la pila, y devuelva el resultado también por referencia a través de la pila.

12) Analizar el funcionamiento de la siguiente subrutina y su programa principal:

```
        ORG 3000H                       ORG 2000H
MUL:    CMP AX, 0                       MOV CX, 0
        JZ  FIN                         MOV AX, 3
        ADD CX, AX                      CALL MUL
        DEC AX                          HLT
        CALL MUL                        END
FIN:    RET
```

a) ¿Qué hace la subrutina?
b) ¿Cuál será el valor final de CX?
c) Dibujar las posiciones de memoria de la pila, anotando qué valores va tomando
d) ¿Cuál será la limitación para determinar el valor más grande que se le puede pasar a la subrutina a través de AX?

Nota: Los ejercicios marcados con * tienen una solución propuesta.

PRÁCTICA 2

Interrupciones

Objetivos: Comprender la utilidad de las interrupciones por software y por hardware y el funcionamiento del Controlador de Interrupciones Programable (PIC). Escribir programas en el lenguaje assembler del simulador MSX88. Ejecutarlos y verificar los resultados, analizando el flujo de información entre los distintos componentes del microprocesador.

1) Escritura de datos en la pantalla de comandos.
Implementar un programa en el lenguaje assembler del simulador MSX88 que muestre en la pantalla de comandos un mensaje previamente almacenado en memoria de datos, aplicando la interrupción por software INT 7.

```
     ORG 1000H
MSJ  DB  "ARQUITECTURA DE COMPUTADORAS-"
     DB  "FACULTAD DE INFORMATICA-"
     DB  55H
     DB  4EH
     DB  4CH
     DB  50H
FIN  DB  ?

     ORG 2000H
     MOV BX, OFFSET MSJ
     MOV AL, OFFSET FIN-OFFSET MSJ
     INT 7
     INT 0
     END
```

2) Escribir un programa que muestre en pantalla todos los caracteres disponibles en el simulador MSX88, comenzando con el caracter cuyo código es el número 01H.

3) * Escribir un programa que muestre en pantalla las letras del abecedario, sin espacios, intercalando mayúsculas y minúsculas (AaBb…), sin incluir texto en la memoria de datos del programa. Tener en cuenta que el código de "A" es 41H, el de "a" es 61H y que el resto de los códigos son correlativos según el abecedario.

4) Lectura de datos desde el teclado.
Escribir un programa que solicite el ingreso de un número (de un dígito) por teclado e inmediatamente lo muestre en la pantalla de comandos, haciendo uso de las interrupciones por software INT 6 e INT 7.

```
      ORG 1000H
MSJ   DB      "INGRESE UN NUMERO:"
FIN   DB      ?

      ORG 1500H
NUM   DB      ?

      ORG 2000H
      MOV BX, OFFSET MSJ
      MOV AL, OFFSET FIN-OFFSET MSJ
      INT 7
      MOV BX, OFFSET NUM
      INT 6
      MOV AL, 1
      INT 7
      MOV CL, NUM
      INT 0
      END
```

Responder brevemente:

a) Con referencia a la interrupción INT 7, ¿qué se almacena en los registros BX y AL?
b) Con referencia a la interrupción INT 6, ¿qué se almacena en BX?
c) En el programa anterior, ¿qué hace la segunda interrupción INT 7? ¿qué queda almacenado en el registro CL?

5) Modificar el programa anterior agregando una subrutina llamada ES_NUM que verifique si el caracter ingresado es

realmente un número. De no serlo, el programa debe mostrar el mensaje “CARACTER NO VALIDO”. La subrutina debe recibir el código del caracter por referencia desde el programa principal y debe devolver vía registro el valor 0FFH en caso de tratarse de un número o el valor 00H en caso contrario. Tener en cuenta que el código del “0” es 30H y el del “9” es 39H.

6) * Escribir un programa que solicite el ingreso de un número (de un dígito) por teclado y muestre en pantalla dicho número expresado en letras. Luego que solicite el ingreso de otro y así sucesivamente. Se debe finalizar la ejecución al ingresarse en dos vueltas consecutivas el número cero.

7) * Escribir un programa que efectúe la suma de dos números (de un dígito cada uno) ingresados por teclado y muestre el resultado en la pantalla de comandos. Recordar que el código de cada caracter ingresado no coincide con el número que representa y que el resultado puede necesitar ser expresado con 2 dígitos.

8) Escribir un programa que efectúe la resta de dos números (de un dígito cada uno) ingresados por teclado y muestre el resultado en la pantalla de comandos. Antes de visualizarlo el programa debe verificar si el resultado es positivo o negativo y anteponer al valor el signo correspondiente.

9) Escribir un programa que aguarde el ingreso de una clave de cuatro caracteres por teclado sin visualizarla en pantalla. En caso de coincidir con una clave predefinida (y guardada en memoria) que muestre el mensaje "Acceso permitido", caso contrario el mensaje "Acceso denegado".

10) Interrupción por hardware: tecla F10.
Escribir un programa que, mientras ejecuta un lazo infinito, cuente el número de veces que se presiona la tecla F10 y acumule este valor en el registro DX.

```
PIC       EQU 20H
EOI       EQU 20H
N_F10     EQU 10

          ORG 40
IP_F10    DW  RUT_F10

          ORG 2000H
          CLI
          MOV AL, 0FEH
          OUT PIC+1, AL ; PIC: registro IMR
          MOV AL, N_F10
          OUT PIC+4, AL ; PIC: registro INT0
          MOV DX, 0
          STI
LAZO:     JMP LAZO

          ORG 3000H
RUT_F10:  PUSH AX
          INC DX
          MOV AL, EOI
          OUT EOI, AL  ; PIC: registro EOI
          POP AX
          IRET

          END
```

Explicar detalladamente:

a) La función de los registros del PIC: ISR, IRR, IMR, INT0-INT7, EOI. Indicar la dirección de cada uno.
b) Cuáles de estos registros son programables y cómo trabaja la instrucción OUT.
c) Qué hacen y para qué se usan las instrucciones CLI y STI.

11) Escribir un programa que permita seleccionar una letra del abecedario al azar. El código de la letra debe generarse en un registro que incremente su valor desde el código de A hasta el de Z continuamente. La letra debe quedar seleccionada al presionarse la tecla F10 y debe mostrarse de inmediato en la pantalla de comandos.

12) Interrupción por hardware: TIMER.
Implementar a través de un programa un reloj segundero que muestre en pantalla los segundos transcurridos (00-59 seg) desde el inicio de la ejecución.

```
TIMER     EQU 10H
PIC       EQU 20H
EOI       EQU 20H
N_CLK     EQU 10

          ORG 40
IP_CLK    DW RUT_CLK

          ORG 1000H
SEG       DB 30H
          DB 30H
FIN       DB ?

          ORG 3000H
RUT_CLK:  PUSH AX
          INC SEG+1
          CMP SEG+1, 3AH
          JNZ RESET
          MOV SEG+1, 30H
          INC SEG
          CMP SEG, 36H
          JNZ RESET
          MOV SEG, 30H
RESET:    INT 7
          MOV AL, 0
          OUT TIMER, AL
          MOV AL, EOI
          OUT PIC, AL
          POP AX
          IRET

          ORG 2000H
          CLI
          MOV AL, 0FDH
          OUT PIC+1, AL       ; PIC: registro IMR
          MOV AL, N_CLK
          OUT PIC+5, AL       ; PIC: registro INT1
          MOV AL, 1
          OUT TIMER+1, AL     ; TIMER: registro COMP
          MOV AL, 0
          OUT TIMER, AL       ; TIMER: registro CONT
          MOV BX, OFFSET SEG
          MOV AL, OFFSET FIN-OFFSET SEG
          STI
LAZO:     JMP LAZO

          END
```

Explicar detalladamente:

a) Cómo funciona el TIMER y cuándo emite una interrupción a la CPU.
b) La función que cumplen sus registros, la dirección de cada uno y cómo se programan.

13) Modificar el programa anterior para que también cuente minutos (00:00 - 59:59), pero que actualice la visualización en pantalla cada 10 segundos.

14) * Implementar un reloj similar al utilizado en los partidos de básquet, que arranque y detenga su marcha al presionar sucesivas veces la tecla F10 y que finalice el conteo al alcanzar los 30 segundos.

15) Escribir un programa que implemente un conteo regresivo a partir de un valor ingresado desde el teclado. El conteo debe comenzar al presionarse la tecla F10. El tiempo transcurrido debe mostrarse en pantalla, actualizándose el valor cada segundo.

Nota: Los ejercicios marcados poseen una resolución propuesta

PRACTICA 3

Entrada/Salida

Objetivos: Comprender la comunicación entre el microprocesador y los periféricos externos (luces, microconmutadores e impresora). Configurar la interfaz de entrada/salida (PIO), el dispositivo de handshaking (HAND-SHAKE) y el dispositivo de comunicación serie (USART) para el intercambio de información entre el microprocesador y el mundo exterior. Escribir programas en el lenguaje assembly del simulador VonSim y el MSX88. Ejecutarlos y verificar los resultados, analizando el flujo de información entre los distintos componentes del sistema.

1) **Uso de las luces y las llaves a través del PIO**. Ejecutar los programas con el simulador VonSim utilizando los dispositivos "Llaves y Luces" que conectan las llaves al puerto PA del PIO y a las luces al puerto PB.

   a) * Escribir un programa que encienda las luces con el patrón 11000011, o sea, solo las primeras y las últimas dos luces deben prenderse, y el resto deben apagarse.
   b) * Escribir un programa que verifique si la llave de más a la izquierda está prendida. Si es así, mostrar en pantalla el mensaje "Llave prendida", y de lo contrario mostrar "Llave apagada". Solo importa el valor de la llave de más a la izquierda (bit más significativo). Recordar que las llaves se manejan con las teclas 0-7.
   c) * Escribir un programa que permite encender y apagar las luces mediante las llaves. El programa no deberá terminar nunca, y continuamente revisar el estado de las llaves, y actualizar de forma consecuente el estado de las luces. La actualización se realiza simplemente prendiendo la luz **i** si la llave **i** correspondiente está encendida (valor 1), y apagándola en caso contrario. Por ejemplo, si solo la primera llave está encendida, entonces solo la primera luz se debe quedar encendida.
   d) * Escribir un programa que implemente un encendido y apagado sincronizado de las luces. Un contador, que inicializa en cero, se incrementa en uno una vez por segundo. Por cada incremento, se muestra a través de las luces, prendiendo solo aquellas luces donde el valor de las llaves es 1. Entonces, primero se enciende solo la luz de más a la derecha, correspondiente al patrón 00000001. Luego se continúa con los patrones 00000010, 00000011, y así sucesivamente. El programa termina al llegar al patrón 11111111.
   e) Escribir un programa que encienda una luz a la vez, de las ocho conectadas al puerto paralelo del microprocesador a través de la PIO, en el siguiente orden de bits: 0-1-2-3-4-5-6-7-6-5-4-3-2-1-0-1-2-3-4-5-6-7-6-5-4-3-2-1-0-1-..., es decir, 00000001, 00000010, 00000100, etc. Cada luz debe estar encendida durante un segundo. El programa nunca termina.

2) Uso de la impresora a través de la PIO. Ejecutar los programas configurando el simulador VonSim con los dispositivos "Impresora (PIO)". En esta configuración, el puerto de datos de la impresora se conecta al puerto PB del PIO, y los bits de busy y strobe de la misma se conectan a los bits 0 y 1 respectivamente del puerto PA. Presionar F5 para mostrar la salida en papel. El papel se puede blanquear ingresando el comando BI.

   a) * Escribir un programa para imprimir la letra "A" utilizando la impresora a través de la PIO.
   b) * Escribir un programa para imprimir el mensaje "ORGANIZACION Y ARQUITECTURA DE COMPUTADORAS" utilizando la impresora a través de la PIO.
   c) * Escribir un programa que solicita el ingreso de cinco caracteres por teclado y los envía de a uno por vez a la impresora a través de la PIO a medida que se van ingresando. No es necesario mostrar los caracteres en la pantalla.
   d) * Escribir un programa que solicite ingresar caracteres por teclado y que recién al presionar la tecla F10 los envíe a la impresora a través de la PIO. No es necesario mostrar los caracteres en la pantalla.

3) **Uso de la impresora a través del HAND-SHAKE**. Ejecutar los programas configurando el simulador VonSim con los dispositivos "Impresora (Handshake)"

   a) * Escribir un programa que imprime "INGENIERIA E INFORMATICA" en la impresora a través del HAND-SHAKE. La comunicación se establece por **consulta de estado** (polling). ¿Qué diferencias encuentra con el ejercicio 2b?

b) ¿Cuál es la ventaja en utilizar el HAND-SHAKE con respecto al PIO para comunicarse con la impresora? Sacando eso de lado, ¿Qué ventajas tiene el PIO, en general, con respecto al HAND-SHAKE?
c) * Escribir un programa que imprime "UNIVERSIDAD NACIONAL DE LA PLATA" en la impresora a través del HAND-SHAKE. La comunicación se establece por **interrupciones** emitidas desde el HAND-SHAKE cada vez que la impresora se desocupa.
d) Escribir un programa que solicite el ingreso de cinco caracteres por teclado y los almacene en memoria. Una vez ingresados, que los envíe a la impresora a través del HAND-SHAKE, en primer lugar tal cual fueron ingresados y a continuación en sentido inverso. Utilizar el HAND-SHAKE en modo **consulta de estado**. ¿Qué diferencias encuentra con el ejercicio 2c?
e) Idem d), pero ahora utilizar el HAND-SHAKE en modo **interrupciones**.

4) Uso de la impresora a través del dispositivo USART por consulta de estado. Ejecutar utilizando el simulador MSX88 (versión antigua del VonSim) en configuración P1 C4 y utilizar el comando PI que corresponda en cada caso (ver uso de Comando PI en el simulador).

a) * Escribir un programa que imprima el carácter `"A"` en la impresora a través de la USART usando el protocolo **DTR** . La comunicación es por **consulta de estado**.
b) * Escribir un programa que imprima la cadena `"USART DTR POLLING"` en la impresora a través de la USART usando el protocolo **DTR** . La comunicación es por **consulta de estado**.
c) * Escribir un programa que imprima la cadena `"USART XON/XOFF POLLING"` en la impresora a través de la USART usando el protocolo **XON/XOFF** realizando la comunicación entre CPU y USART por **consulta de estado**.

Nota: Los ejercicios marcados con * tienen una solución propuesta.

Anexo DMA

Objetivos: Comprender el funcionamiento del Controlador de Acceso Directo a Memoria (CDMA) incluido en el simulador MSX88. Configurarlo para la transferencia de datos memoria-memoria y memoria-periférico en modo bloque y bajo demanda. Escribir programas en el lenguaje assembly del simulador MSX88. Ejecutarlos y verificar los resultados, analizando el flujo de información entre los distintos componentes del sistema

1- **DMA**. Transferencia de datos memoria-memoria.

Programa que copia una cadena de caracteres almacenada a partir de la dirección 1000H en otra parte de la memoria, utilizando el CDMA en modo de transferencia por bloque. La cadena original se debe mostrar en la pantalla de comandos antes de la transferencia. Una vez finalizada, se debe visualizar en la pantalla la cadena copiada para verificar el resultado de la operación. Ejecutar el programa en la configuración P1 C3.

```
PIC       EQU 20H
DMA       EQU 50H
N_DMA     EQU 20

          ORG 80

IP_DMA    DW  RUT_DMA

          ORG 1000H
MSJ       DB  "FACULTAD DE"
          DB  " INFORMATICA"
FIN       DB  ?
NCHAR     DB  ?

          ORG 1500H
COPIA     DB  ?

; rutina aten interrupción del CDMA
          ORG 3000H
RUT_DMA:  MOV AL, 0FFH    ;inhabilita..
          OUT PIC+1, AL   ;interrupc de PIC
          MOV BX, OFFSET COPIA
          MOV AL, NCHAR
          INT 7
          MOV AL, 20H
          OUT PIC, AL     ; EOI
          IRET
```

```
ORG 2000H
CLI
MOV AL, N_DMA
OUT PIC+7, AL      ; reg INT3 de PIC
MOV AX, OFFSET
MSJ
OUT DMA, AL        ; dir comienzo ..
MOV AL, AH         ; del bloque ..
OUT DMA+1, AL      ; a transferir
MOV AX, OFFSET FIN-OFFSET MSJ
OUT DMA+2, AL      ; cantidad ..
MOV AL, AH         ; a ..
OUT DMA+3, AL      ; transferir
MOV AX, OFFSET COPIA
OUT DMA+4, AL      ; dir destino ..
MOV AL, AH         ; del ..
OUT DMA+5, AL      ; bloque
MOV AL, 0AH        ; CDMA en transfer..
OUT DMA+6, AL      ; mem-mem por bloque
MOV AL, 0F7H
OUT PIC+1, AL      ; habilita INT3
STI
MOV BX, OFFSET MSJ
MOV AL, OFFSET FIN-OFFSET MSJ
MOV NCHAR, AL
INT 7              ; mensaje original
MOV AL, 7H
OUT DMA+7, AL      ; arranque Transfer
INT 0
END
```

Cuestionario:

a) Analizar minuciosamente cada línea del programa anterior.
b) Explicar qué función cumple cada registro del CDMA e indicar su dirección.
c) Describir el significado de los bits del registro CTRL.
d) ¿Qué diferencia hay entre transferencia de datos por bloque y bajo demanda?
e) ¿Cómo se le indica al CDMA desde el programa que debe arrancar la transferencia de datos?
f) ¿Qué le indica el CDMA a la CPU a través de la línea hrq? ¿Qué significa la respuesta que le envía la CPU a través de la línea hlda?
g) Explicar detalladamente cada paso de la operación de transferencia de un byte desde una celda a otra de la memoria. Verificar que en esta operación intervienen el bus de direcciones, el bus de datos y las líneas mrd y mwr.
h) ¿Qué sucede con los registros RF, CONT y RD del CDMA después de transferido un byte?
i) ¿Qué evento hace que el CDMA emita una interrupción y a través de qué línea de control lo hace?
j) ¿Cómo se configura el PIC para atender la interrupción del CDMA?
k) ¿Qué hace la rutina de interrupción del CDMA del programa anterior?

2- **DMA**. Transferencia de datos memoria-periférico.

Programa que transfiere datos desde la memoria hacia la impresora sin intervención de la CPU, utilizando el CDMA en modo de transferencia bajo demanda.

```
PIC       EQU 20H
HAND      EQU 40H
DMA       EQU 50H
N_DMA     EQU 20

          ORG 80
IP_DMA    DW  RUT_DMA

          ORG 1000H
MSJ       DB  " INFORMATICA"
FIN       DB  ?
FLAG      DB  0

; rutina atención interrupción del CDMA
          ORG 3000H
RUT_DMA:  MOV AL, 0        ;inhabilita..
          OUT HAND+1, AL  ;interrup de HAND
          MOV FLAG, 1
          MOV AL, 0FFH     ;inhabilita..
          OUT PIC+1, AL   ;interrup de PIC
          MOV AL, 20H
          OUT PIC, AL      ; EOI
          IRET
```

```
          ORG 2000H
          CLI
          MOV AL, N_DMA
          OUT PIC+7, AL    ; reg INT3 de PIC
          MOV AX, OFFSET MSJ
          OUT DMA, AL      ; dir comienzo ..
          MOV AL, AH       ; del bloque ..
          OUT DMA+1, AL    ; a transferir
          MOV AX, OFFSET FIN-OFFSET MSJ
          OUT DMA+2, AL    ; cantidad ..
          MOV AL, AH       ; a ..
          OUT DMA+3, AL    ; transferir
          MOV AL, 4        ; inicialización ..
          OUT DMA+6, AL    ; de control DMA
          MOV AL, 0F7H
          OUT PIC+1, AL    ; habilita INT3
          OUT DMA+7, AL    ; arranque Transfer
          MOV AL, 80H
          OUT HAND+1, AL   ; interrup de HAND
          STI

LAZO:     CMP FLAG, 1
          JNZ LAZO
          INT 0
          END
```

Cuestionario:

a) Analizar minuciosamente cada línea del programa anterior.
b) ¿Qué debe suceder para que el HAND-SHAKE emita una interrupción al CDMA?
c) ¿Cómo demanda el periférico, en este caso el HAND-SHAKE, la transferencia de datos desde memoria? ¿A través de qué líneas se comunican con el CDMA ante cada pedido?
d) Explicar detalladamente cada paso de la operación de transferencia de un byte desde una celda de memoria hacia el HAND-SHAKE y la impresora.
e) ¿Qué evento hace que el CDMA emita una interrupción al PIC?
f) ¿Cuándo finaliza la ejecución del LAZO?

3. * Configuración del CDMA. Indique cómo configurar el registro **Control** del CDMA para las siguientes transferencias:

a) Transferencia Memoria → Memoria, por robo de ciclo
b) Transferencia Periférico → Memoria, por ráfagas
c) Transferencia Memoria → Periférico, por robo de ciclo

PRACTICA 4

Segmentación de cauce en procesador RISC

Objetivos: Comprender el funcionamiento de la segmentación de cauce del procesador MIPS de 64 bits. Analizar las ventajas e inconvenientes de este tipo de arquitectura. Familiarizarse con el desarrollo de programas para procesadores con sets reducidos de instrucciones (RISC). Resolver problemas y verificarlos a través de simulaciones (winmips64)

Los ejercicios con * tienen solución propuesta total o parcial

1) Muchas instrucciones comunes en procesadores con arquitectura CISC no forman parte del repertorio de instrucciones del MIPS64, pero pueden implementarse haciendo uso de una única instrucción. Evaluar las siguientes instrucciones, indicar qué tarea realizan y cuál sería su equivalente en lenguaje assembly del x86.
   a) `dadd r1, r2, r0`
   b) `daddi r3, r0, 5`
   c) `dsub r4, r4, r4`
   d) `daddi r5, r5, -1`
   e) `xori r6, r6, 0xffffffffffffffff`

2) * El siguiente programa intercambia el contenido de dos palabras de la memoria de datos, etiquetadas A y B.

```
    .data
A:  .word  1
B:  .word  2
    .code
    ld     r1, A(r0)
    ld     r2, B(r0)
    sd     r2, A(r0)
    sd     r1, B(r0)
    halt
```

   a) Ejecutarlo en el simulador con la opción Configure/Enable Forwarding deshabilitada. Analizar paso a paso su funcionamiento, examinar las distintas ventanas que se muestran en el simulador y responder:
      - ¿Qué instrucción está generando atascos (stalls) en el cauce (ó pipeline) y por qué?
      - ¿Qué tipo de 'stall' es el que aparece?
      - ¿Cuál es el promedio de Ciclos Por Instrucción (CPI) en la ejecución de este programa bajo esta configuración?
   b) Una forma de solucionar los atascos por dependencia de datos es utilizando el Adelantamiento de Operandos o Forwarding. Ejecutar nuevamente el programa anterior con la opción Enable Forwarding habilitada y responder:
      - ¿Por qué no se presenta ningún atasco en este caso? Explicar la mejora.
      - ¿Qué indica el color de los registros en la ventana Register durante la ejecución?
      - ¿Cuál es el promedio de Ciclos Por Instrucción (CPI) en este caso? Comparar con el anterior.

3) * Analizar el siguiente programa con el simulador MIPS64:

```
        .data
A:      .word 1
B:      .word 3
        .code
        ld     r1, A(r0)
        ld     r2, B(r0)
loop:   dsll   r1, r1, 1
        daddi  r2, r2, -1
        bnez   r2, loop
        halt
```

   a) Ejecutar el programa con Forwarding habilitado y responder:
      - ¿Por qué se presentan atascos tipo RAW?
      - Branch Taken es otro tipo de atasco que aparece. ¿Qué significa? ¿Por qué se produce?
      - ¿Cuántos CPI tiene la ejecución de este programa? Tomar nota del número de ciclos, cantidad de instrucciones y CPI.
   b) Ejecutar ahora el programa deshabilitando el Forwarding y responder:
      - ¿Qué instrucciones generan los atascos tipo RAW y por qué? ¿En qué etapa del cauce se produce el atasco en cada caso y durante cuántos ciclos?
      - Los Branch Taken Stalls se siguen generando. ¿Qué cantidad de ciclos dura este atasco en cada vuelta del lazo 'loop'? Comparar con la ejecución con Forwarding y explicar la diferencia.
      - ¿Cuántos CPI tiene la ejecución del programa en este caso? Comparar número de ciclos, cantidad de instrucciones y CPI con el caso con Forwarding.
   c) Reordenar las instrucciones para que la cantidad de RAW sea '0' en la ejecución del programa (Forwarding habilitado)
   d) Modificar el programa para que almacene en un arreglo en memoria de datos los contenidos parciales del registro r1 ¿Qué significado tienen los elementos de la tabla que se genera?

4) * Dado el siguiente programa:

```
        .data
tabla: .word 20, 1, 14, 3, 2, 58, 18, 7, 12, 11
num:   .word 7
```

```
long:  .word 10
       .code
       ld     r1, long(r0)
       ld     r2, num(r0)
       dadd   r3, r0, r0
       dadd   r10, r0, r0
loop:  ld     r4, tabla(r3)
       beq    r4, r2, listo
       daddi  r1, r1, -1
       daddi  r3, r3, 8
       bnez   r1, loop
       j      fin
listo: daddi  r10, r0, 1
fin:   halt
```

a) Ejecutar en simulador con Forwarding habilitado. ¿Qué tarea realiza? ¿Cuál es el resultado y dónde queda indicado?
b) Re-Ejecutar el programa con la opción Configure/Enable Branch Target Buffer habilitada. Explicar la ventaja de usar este método y cómo trabaja.
c) Confeccionar una tabla que compare número de ciclos, CPI, RAWs y Branch Taken Stalls para los dos casos anteriores.

5) El siguiente programa multiplica por 2 los elementos de un arreglo llamado datos y genera un nuevo arreglo llamado res. Ejecutar el programa en el simulador winmips64 con la opción Delay Slot habilitada.

```
       .data
cant:  .word 8
datos: .word 1, 2, 3, 4, 5, 6, 7, 8
res:   .word 0
       .code
       dadd   r1, r0, r0
       ld     r2, cant(r0)
loop:  ld     r3, datos(r1)
       daddi  r2, r2, -1
       dsll   r3, r3, 1
       sd     r3, res(r1)
       daddi  r1, r1, 8
       bnez   r2, loop
       nop
       halt
```

a) ¿Qué efecto tiene habilitar la opción Delay Slot (salto retardado)?.
b) ¿Con qué fin se incluye la instrucción NOP? ¿Qué sucedería si no estuviera?.
c) Tomar nota de la cantidad de ciclos, la cantidad de instrucciones y los CPI luego de ejecutar el programa.
d) Modificar el programa para aprovechar el 'Delay Slot' ejecutando una instrucción útil. Simular y comparar número de ciclos, instrucciones y CPI obtenidos con los de la versión anterior.

6) Escribir un programa que lea tres números enteros A, B y C de la memoria de datos y determine cuántos de ellos son iguales entre sí (0, 2 o 3). El resultado debe quedar almacenado en la dirección de memoria D.

7) * Escribir un programa que recorra una TABLA de diez números enteros y determine cuántos elementos son mayores que X. El resultado debe almacenarse en una dirección etiquetada CANT. El programa debe generar además otro arreglo llamado RES cuyos elementos sean ceros y unos. Un '1' indicará que el entero correspondiente en el arreglo TABLA es mayor que X, mientras que un '0' indicará que es menor o igual.

8) * Escribir un programa que multiplique dos números enteros utilizando sumas repetidas (similar a Ejercicio 6 o 7 de la Práctica 1). El programa debe estar optimizado para su ejecución con la opción Delay Slot habilitada.

9) Escribir un programa que implemente el siguiente fragmento escrito en un lenguaje de alto nivel:

```
while a > 0 do
begin
       x := x + y;
       a := a - 1;
end;
```

Ejecutar con la opción Delay Slot habilitada.

10) Escribir un programa que cuente la cantidad de veces que un determinado caracter aparece en una cadena de texto. Observar cómo se almacenan en memoria los códigos ASCII de los caracteres (código de la letra "a" es 61H). Utilizar la instrucción lbu (load byte unsigned) para cargar códigos en registros. La inicialización de los datos es la siguiente:

```
       .data
cadena: .asciiz "adbdcdedfdgdhdid"   ; cadena a analizar
car:    .asciiz "d"                  ; caracter buscado
cant:   .word 0                      ; cantidad de veces que se repite el caracter car en cadena.
```

PRACTICA 5

Procesador RISC: instrucciones de Punto Flotante y pasaje de parámetros

Objetivos: Familiarizarse con las instrucciones de MIPS64 para manejar datos en formato de Punto Flotante. 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.

1) Simular el siguiente programa de suma de números en punto flotante y analizar minuciosamente la ejecución paso a paso. Inhabilitar Delay Slot y mantener habilitado Forwarding.

```
          .data
n1:       .double 9.13
n2:       .double 6.58
res1:     .double 0.0
res2:     .double 0.0

          .code
(1)       l.d         f1, n1(r0)
(2)       l.d         f2, n2(r0)
(3)       add.d       f3, f2, f1
(4)       mul.d       f4, f2, f1
(5)       s.d         f3, res1(r0)
(6)       s.d         f4, res2(r0)
(7)       halt
```

   a) Tomar nota de la cantidad de ciclos, instrucciones y CPI luego de la ejecución del programa.
   b) ¿Cuántos atascos por dependencia de datos se generan? Observar en cada caso cuál es el dato en conflicto y las instrucciones involucradas.
   c) ¿Por qué se producen los atascos estructurales? Observar cuales son las instrucciones que los generan y en qué etapas del pipeline aparecen.
   d) Modificar el programa agregando la instrucción **`mul.d`** `f1, f2, f1` entre las instrucciones **`add.d`** y **`mul.d`**. Repetir la ejecución y observar los resultados. ¿Por qué aparece un atasco tipo WAR?
   e) Explicar por qué colocando un NOP antes de la suma, se soluciona el RAW de la instrucción ADD y como consecuencia se elimina el WAR.

2) *Es posible convertir valores enteros almacenados en alguno de los registros `r1`-`r31` a su representación equivalente en punto flotante y viceversa. Describa la funcionalidad de las instrucciones **`mtc1`**, **`cvt.l.d`**, **`cvt.d.l`** y **`mfc1`**.

3) *Escribir un programa que calcule la superficie de un triángulo rectángulo de base 5,85 cm y altura 13,47 cm. Pista: la superficie de un triángulo se calcula como:

$$\text{Superficie} = (\text{base} \times \text{altura}) / 2$$

4) El índice de masa corporal (IMC) es una medida de asociación entre el peso y la talla de un individuo. Se calcula a partir del peso (expresado en kilogramos, por ejemplo: 75,7 kg) y la estatura (expresada en metros, por ejemplo 1,73 m), usando la fórmula:

$$\text{IMC} = \text{peso} / (\text{estatura})^2$$

   De acuerdo al valor calculado con este índice, puede clasificarse el estado nutricional de una persona en: **Infrapeso** (IMC < 18,5), **Normal** (18,5 ≤ IMC < 25), **Sobrepeso** (25 ≤ IMC < 30) y **Obeso** (IMC ≥ 30).
   Escriba un programa que dado el peso y la estatura de una persona calcule su IMC y lo guarde en la dirección etiquetada `IMC`. También deberá guardar en la dirección etiquetada `estado` un valor según la siguiente tabla:

IMC	Clasificación	Valor guardado
< 18,5	Infrapeso	1
< 25	Normal	2
< 30	Sobrepeso	3
≥ 30	Obeso	4

5) El procesador MIPS64 posee 32 registros, de 64 bits cada uno, llamados r0 a r31 (también conocidos como $0 a $31). Sin embargo, resulta más conveniente para los programadores darles nombres más significativos a esos registros. La siguiente tabla muestra la convención empleada para nombrar a los 32 registros mencionados:

Registros	Nombres	¿Para que se los utiliza?	¿Preservado?
r0	$zero		
r1	$at		
r2-r3	$v0-$v1		
r4-r7	$a0-$a3		
r8-r15	$t0-$t7		
r16-r23	$s0-$s7		
r24-r25	$t8-$t9		
r26-r27	$k0-$k1		
R28	$gp		
R29	$sp		
R30	$fp		
R31	$ra		

Complete la tabla anterior explicando el uso que normalmente se le da cada uno de los registros nombrados. Marque en la columna "¿Preservado?" si el valor de cada grupo de registros debe ser preservado luego de realizada una llamada a una subrutina. Puede encontrar información útil en el apunte "Programando sobre MIPS64".

6) Como ya se observó anteriormente, 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.

```
                .data
valor1:         .word 16
valor2:         .word 4
result:         .word 0

                .text
                ld    $a0, valor1($zero)
                ld    $a1, valor2($zero)
                jal   a_la_potencia
                sd    $v0, result($zero)
                halt

a_la_potencia:  daddi $v0, $zero, 1
         lazo:  slt   $t1, $a1, $zero
                bnez  $t1, 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?
b) ¿Qué acciones produce la instrucción **`jal`**? ¿Y la instrucción **`jr`**?
c) ¿Qué valor se almacena en el registro `$ra`? ¿Qué función cumplen los registros `$a0` y `$a1`? ¿Y el registro `$v0`?
d) ¿Qué sucedería si la subrutina `a_la_potencia` necesitara invocar a otra subrutina para realizar la multiplicación, por ejemplo, en lugar de usar la instrucción **`dmul`**? ¿Cómo sabría cada una de las subrutinas a que dirección de memoria deben retornar?

7) Escriba una subrutina que reciba como parámetros un número positivo M de 64 bits, la dirección del comienzo de una tabla que contenga valores numéricos de 64 bits sin signo y la cantidad de valores almacenados en dicha tabla. La subrutina debe retornar la cantidad de valores mayores que M contenidos en la tabla.

8) *Escriba una subrutina que reciba como parámetros las direcciones del comienzo de dos cadenas terminadas en cero y retorne la posición en la que las dos cadenas difieren. En caso de que las dos cadenas sean idénticas, debe retornar -1.

9) *Escriba la subrutina ES_VOCAL que determina si un caracter es vocal o no, ya sea mayúscula o minúscula. La rutina debe recibir el caracter y debe retornar el valor 1 si es una vocal ó 0 en caso contrario.

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

11) Escribir una subrutina que reciba como argumento una tabla de números terminada en 0. La subrutina debe contar la cantidad de números que son impares en la tabla. Ésta condición se debe verificar usando la subrutina ES_IMPAR. La subrutina ES_IMPAR debe devolver 1 si el número es impar y 0 si no lo es.

12) El siguiente programa espera usar una subrutina que calcule en forma recursiva el factorial de un número entero:

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

                .text
                daddi $sp, $zero, 0x400  ; Inicializa el puntero al tope de la pila (1)
                ld    $a0, valor($zero)
                jal   factorial
                sd    $v0, result($zero)
                halt

factorial:      ...
                ...
                ...
```

(1) 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 $2^{10} = 1024 = 400_{16}$.

a) *Implemente la subrutina `factorial` definida en forma recursiva. Tenga presente que el factorial de un número entero n se calcula como el producto de los números enteros entre 1 y n inclusive:

$$\text{factorial}(n) = n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

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

PRACTICA 6

Procesador RISC: utilizando la E/S

Objetivos: Familiarizarse con el desarrollo de programas para procesadores con sets reducidos de instrucciones (RISC). Resolver problemas y verificarlos a través de simulaciones. Dominar el uso del puerto de E/S provisto en el WinMIPS64.

Puerto de E/S mapeado en memoria de datos – Resumen

CONTROL y DATA son direcciones de memoria fijas para acceder al puerto de E/S. Aplicando distintos comandos a través de CONTROL, es posible producir salidas o ingresar datos a través de la dirección DATA. Las direcciones de memoria de CONTROL y DATA son 0x10000 y 0x10008 respectivamente. Como el set de instrucciones del procesador MIPS64 no cuenta con instrucciones que permitan cargar un valor inmediato de más de 16 bits (como es el caso de las direcciones mencionadas), resulta conveniente definirlas en la memoria de datos, así:

```
CONTROL: .word32 0x10000
   DATA: .word32 0x10008
```

De esa manera, resulta sencillo cargar esas dos direcciones en registros mediante:

```
    lwu  $s0, DATA($zero)     ; Carga el valor 0x10000 en $s0
    lwu  $s1, CONTROL($zero)  ; Carga el valor 0x10008 en $s1
```

- Si se escribe en DATA un número entero y se escribe un 1 en CONTROL, se interpretará el valor escrito en DATA como un entero sin signo y se lo imprimirá en la pantalla alfanumérica de la terminal.
- Si se escribe en DATA un número entero y se escribe un 2 en CONTROL, se interpretará el valor escrito en DATA como un entero con signo y se lo imprimirá en la pantalla alfanumérica de la terminal.
- Si se escribe en DATA un número en punto flotante y se escribe un 3 en CONTROL, se imprimirá en la pantalla alfanumérica de la terminal el número en punto flotante.
- Si se escribe en DATA la dirección de memoria del comienzo de una cadena terminada en 0 y se escribe un 4 en CONTROL, se imprimirá la cadena en la pantalla alfanumérica de la terminal.
- Si se escribe en DATA un color expresado en RGB (usando 4 bytes para representarlo: un byte para cada componente de color e ignorando el valor del cuarto byte), en DATA+4 la coordenada Y, en DATA+5 la coordenada X y se escribe un 5 en CONTROL, se pintará con el color indicado un punto de la pantalla gráfica de la terminal, cuyas coordenadas están indicadas por X e Y. La pantalla gráfica cuenta con una resolución de 50x50 puntos, siendo (0, 0) las coordenadas del punto en la esquina inferior izquierda y (49, 49) las del punto en la esquina superior derecha.
- Si se escribe un 6 en CONTROL, se limpia la pantalla alfanumérica de la terminal.
- Si se escribe un 7 en CONTROL, se limpia la pantalla gráfica de la terminal.
- Si se escribe un 8 en CONTROL, se permitirá ingresar en la terminal un número (entero o en punto flotante) y el valor del número ingresado estará disponible para ser leído en DATA.
- Si se escribe un 9 en CONTROL, se esperará a que se presione una tecla en la terminal (la cuál no se mostrará al ser presionada) y el código ASCII de dicha tecla estará disponible para ser leído en DATA.

1) El siguiente programa produce la salida de un mensaje predefinido en la ventana Terminal del simulador WinMIPS64. Teniendo en cuenta las condiciones de control del puerto de E/S (en el resumen anterior), modifique el programa de modo que el mensaje a mostrar sea ingresado por teclado en lugar de ser un mensaje fijo.

```
        .data
texto:  .asciiz  "Hola, Mundo!"     ; El mensaje a mostrar
CONTROL: .word32  0x10000
DATA:   .word32  0x10008

        .text
        lwu    $s0, DATA($zero)     ; $s0 = dirección de DATA
        daddi  $t0, $zero, texto    ; $t0 = dirección del mensaje a mostrar
        sd     $t0, 0($s0)          ; DATA recibe el puntero al comienzo del mensaje

        lwu    $s1, CONTROL($zero)  ; $s1 = dirección de CONTROL
        daddi  $t0, $zero, 6        ; $t0 = 6 -> función 6: limpiar pantalla alfanumérica
        sd     $t0, 0($s1)          ; CONTROL recibe 6 y limpia la pantalla

        daddi  $t0, $zero, 4        ; $t0 = 4 -> función 4: salida de una cadena ASCII
        sd     $t0, 0($s1)          ; CONTROL recibe 4 y produce la salida del mensaje
        halt
```

2) Escriba un programa que utilice sucesivamente dos subrutinas: La primera, denominada `ingreso`, debe solicitar el ingreso por teclado de un número entero (de un dígito), verificando que el valor ingresado realmente sea un dígito. La segunda, denominada `muestra`, deberá mostrar en la salida estándar del simulador (ventana Terminal) el valor del número ingresado expresado en letras (es decir, si se ingresa un '4', deberá mostrar 'CUATRO'). Establezca el pasaje de parámetros entre subrutinas respetando las convenciones para el uso de los registros y minimice las detenciones del cauce (ejercicio similar al ejercicio 6 de Práctica 2).

3) Escriba un programa que realice la suma de dos números enteros (de un dígito cada uno) utilizando dos subrutinas: La denominada `ingreso` del ejercicio anterior (ingreso por teclado de un dígito numérico) y otra denominada `resultado`, que muestre en la salida estándar del simulador (ventana Terminal) el resultado numérico de la suma de los dos números ingresados (ejercicio similar al ejercicio 7 de Práctica 2).

4) Escriba un programa que solicite el ingreso por teclado de una clave (sucesión de cuatro caracteres) utilizando la subrutina `char` de ingreso de un carácter. Luego, debe comparar la secuencia ingresada con una cadena almacenada en la variable `clave`. Si las dos cadenas son iguales entre si, la subrutina llamada `respuesta` mostrará el texto "Bienvenido" en la salida estándar del simulador (ventana Terminal). En cambio, si las cadenas no son iguales, la subrutina deberá mostrar "ERROR" y solicitar nuevamente el ingreso de la clave.

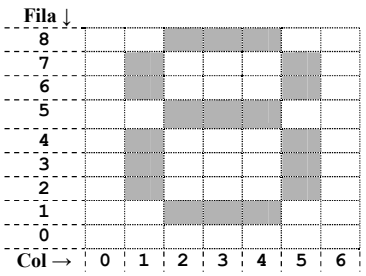
5) Escriba un programa que calcule el resultado de elevar un valor en punto flotante a la potencia indicada por un exponente que es un número entero positivo. Para ello, en el programa principal se solicitará el ingreso de la base (un número en punto flotante) y del exponente (un número entero sin signo) y se deberá utilizar la subrutina `a_la_potencia` para calcular el resultado pedido (que será un valor en punto flotante). Tenga en cuenta que cualquier base elevada a la 0 da como resultado 1. Muestre el resultado numérico de la operación en la salida estándar del simulador (ventana Terminal).

6) El siguiente programa produce una salida estableciendo el color de un punto de la pantalla gráfica (en la ventana Terminal del simulador WinMIPS64). Modifique el programa de modo que las coordenadas y color del punto sean ingresados por teclado.

```
        .data
coorX:  .byte   24                  ; coordenada X de un punto
coorY:  .byte   24                  ; coordenada Y de un punto
color:  .byte   255, 0, 255, 0      ; color: máximo rojo + máximo azul => magenta
CONTROL: .word32 0x10000
DATA:   .word32 0x10008

        .text
        lwu    $s6, CONTROL($zero)  ; $s6 = dirección de CONTROL
        lwu    $s7, DATA($zero)     ; $s7 = dirección de DATA

        daddi  $t0, $zero, 7        ; $t0 = 7 -> función 7: limpiar pantalla gráfica
        sd     $t0, 0($s6)          ; CONTROL recibe 7 y limpia la pantalla gráfica

        lbu    $s0, coorX($zero)    ; $s0 = valor de coordenada X
        sb     $s0, 5($s7)          ; DATA+5 recibe el valor de coordenada X
        lbu    $s1, coorY($zero)    ; $s1 = valor de coordenada Y
        sb     $s1, 4($s7)          ; DATA+4 recibe el valor de coordenada Y
        lwu    $s2, color($zero)    ; $s2 = valor de color a pintar
        sw     $s2, 0($s7)          ; DATA recibe el valor del color a pintar

        daddi  $t0, $zero, 5        ; $t0 = 5 -> función 5: salida gráfica
        sd     $t0, 0($s6)          ; CONTROL recibe 5 y produce el dibujo del punto
        halt
```

7) Se desea realizar la demostración de la transformación de un carácter codificado en ASCII a su visualización en una matriz de puntos con 7 columnas y 9 filas. Escriba un programa que realice tal demostración, solicitando el ingreso por teclado de un carácter para luego mostrarlo en la pantalla gráfica de la terminal.

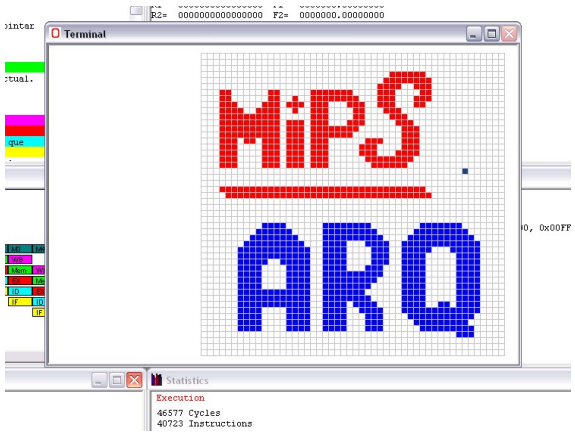
El carácter '8' es representado como:



8) El siguiente programa implementa una animación de una pelotita rebotando por la pantalla. Modifíquelo para que en lugar de una pelotita, se muestren simultáneamente varias pelotitas (cinco, por ejemplo), cada una con su posición, dirección y color particular.

```
        .data
CONTROL:        .word32 0x10000
DATA:           .word32 0x10008
color_pelota:   .word32 0x00FF0000  ; Azul
color_fondo:    .word32 0x00FFFFFF  ; Blanco

        .text
        lwu     $s6, CONTROL($zero)
        lwu     $s7, DATA($zero)
        lwu     $v0, color_pelota($zero)
        lwu     $v1, color_fondo($zero)
        daddi   $s0, $zero, 23      ; Coordenada X de la pelota
        daddi   $s1, $zero, 1       ; Coordenada Y de la pelota
        daddi   $s2, $zero, 1       ; Dirección X de la pelota
        daddi   $s3, $zero, 1       ; Dirección Y de la pelota
        daddi   $s4, $zero, 5       ; Comando para dibujar un punto
loop:   sw      $v1, 0($s7)     ; Borra la pelota
        sb      $s0, 4($s7)
        sb      $s1, 5($s7)
        sd      $s4, 0($s6)
        dadd    $s0, $s0, $s2   ; Mueve la pelota en la dirección actual
        dadd    $s1, $s1, $s3
        daddi   $t1, $zero, 48      ; Comprueba que la pelota no esté en la columna de más
        slt     $t0, $t1, $s0   ; a la derecha. Si es así, cambia la dirección en X.
        dsll    $t0, $t0, 1
        dsub    $s2, $s2, $t0
        slt     $t0, $t1, $s1   ; Comprueba que la pelota no esté en la fila de más arriba.
        dsll    $t0, $t0, 1     ; Si es así, cambia la dirección en Y.
        dsub    $s3, $s3, $t0
        slti    $t0, $s0, 1     ; Comprueba que la pelota no esté en la columna de más
        dsll    $t0, $t0, 1     ; a la izquierda. Si es así, cambia la dirección en X.
        dadd    $s2, $s2, $t0
        slti    $t0, $s1, 1     ; Comprueba que la pelota no esté en la fila de más abajo.
        dsll    $t0, $t0, 1     ; Si es así, cambia la dirección en Y.
        dadd    $s3, $s3, $t0
        sw      $v0, 0($s7)     ; Dibuja la pelota.
        sb      $s0, 4($s7)
        sb      $s1, 5($s7)
        sd      $s4, 0($s6)
        daddi   $t0, $zero, 500   ; Hace una demora para que el rebote no sea tan rápido.
demora: daddi   $t0, $t0, -1    ; Esto genera una infinidad de RAW y BTS pero...
        bnez    $t0, demora     ; ¡hay que hacer tiempo igualmente!
        j       loop
```

9) Escriba un programa que le permita dibujar en la pantalla gráfica de la terminal. Deberá mostrar un cursor (representado por un punto de un color particular) que pueda desplazarse por la pantalla usando las teclas ‘a’, ‘s’, ‘d’ y ‘w’ para ir a la izquierda, abajo, a la derecha y arriba respectivamente. Usando la barra espaciadora se alternará entre modo desplazamiento (el cursor pasa por arriba de lo dibujado sin alterarlo) y modo dibujo (cada punto por el que el cursor pasa quedará pintado del color seleccionado). Las teclas del ‘1’ al ‘8’ se usarán para elegir uno entre los ocho colores disponibles para pintar.



Observaciones: Para poder implementar este programa, se necesitará almacenar en la memoria la imagen completa de la pantalla gráfica.

Si cada punto está representado por un byte, se necesitarán 50x50x1 = 2500 bytes. El simulador WinMIPS64 viene configurado para usar un bus de datos de 10 bits, por lo que la memoria disponible estará acotada a 2^{10}=1024 bytes.

Para poder almacenar la imagen, será necesario configurar el simulador para usar un bus de datos de 12 bits, ya que 2^{12}=4096 bytes, los que si resultarán suficientes. La configuración se logra yendo al menú “Configure → Architecture” y poniendo “Data Address Bus” en 12 bits en lugar de los 10 bits que trae por defecto.