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.

Juan Menduiña

Trabajo Práctico N° 1:
Subrutinas y Pasaje de Parámetros.

Ejercicio 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
mov ax, 5	8000h	5	---
mov bx, 3	8000h	5	3
push ax	7FFCh	5	3
push bx	7FFAh	5	3
push ax	7FFEh	5	3
pop bx	7FFCh	5	3
pop bx	7FFEh	5	3
pop ax	8000h	5	3

Ejercicio 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**
org 3000h	---
rutina: mov bx, 3	7FFCh
ret	7FFEh
org 2000h	---
push ax	7FFEh
call rutina	7FFCh
pop bx	8000h
hlt	8000h
end	8000h

<u>Ejercicio 3:</u> Llamadas a subrutinas y dirección de retorno.

(a) *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.*

```
        org 3000h
RUT:    mov bx, 3       Dirección 3000h     Pila: 2002h; 2006h
        ret             Dirección 3002h     Pila: 2002h; 2006h

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

(b) *Explicar detalladamente:*

(i) *Las acciones que tienen lugar al ejecutarse la instrucción CALL RUT.*

Al ejecutarse la instrucción CALL RUT, se guarda el valor de la posición de memoria que está en el puntero de instrucción (IP) en la pila (PUSH del IP), se asigna el valor de la posición de memoria correspondiente a la etiqueta RUT al IP y la CPU comienza a ejecutar las instrucciones de la subrutina RUT.

(ii) *Las acciones que tienen lugar al ejecutarse la instrucción RET.*

La operación que se realiza con la instrucción *ret* es retornar al programa principal a partir de la instrucción siguiente a la instrucción CALL RUT. La CPU sabe a qué dirección de memoria debe retornar desde la subrutina al programa principal porque el puntero de instrucción (IP) se carga con el valor de la posición de memoria guardada en la pila (POP del IP) y, por lo tanto, la ejecución del programa sigue a partir de la instrucción siguiente a la instrucción CALL RUT.

<u>Ejercicio 4:</u> 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
mov ax, 5 call subrutina	X		X	
mov dx, offset A call subrutina	X			X
mov bx, 5 push bx call subrutina pop bx		X	X	
mov cx, offset A push cx call subrutina pop cx		X		X
mov dl, 5 call subrutina	X		X	
call subrutina mov A, dx	X		X	

<u>Ejercicio 5:</u> 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.*

```
org 1000h
A DW 1h
B DW 2h
C DW 3h
D DW ?

org 2000h
mov ax, A
add ax, B
sub ax, C
mov D, ax
hlt
end
```

(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 este ejercicio, sirve para ver sus diferencias con el uso de parámetros.*

```
            org 1000h
            A DW 1h
            B DW 2h
            C DW 3h
            D DW ?

            org 3000h
CALCULO:    mov ax, A
            add ax, B
            sub ax, C
            mov D, ax
            ret

            org 2000h
```

```
        call CALCULO
        hlt
        end
```

(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.*

```
        org 1000h
        A DW 1h
        B DW 2h
        C DW 3h
        D DW ?

        org 3000h
CALCULO: mov dx, ax
        add dx, bx
        sub dx, cx
        ret

        org 2000h
        mov ax, A
        mov bx, B
        mov cx, C
        call CALCULO
        mov D, dx
        hlt
        end
```

(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)?*

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, no podría reutilizar la subrutina del inciso (b) sin modificarla, aunque sí la subrutina del inciso (c).

<u>Ejercicio 6:</u> 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.*

```
        org 1000h
        NUM1 DB 1
        NUM2 DB 2
        RES DW ?

        org 2000h
        mov dx, 0
        mov al, NUM1
        cmp al, 0
        jz FIN
        mov ah, 0
        mov cl, NUM2
LAZO:   cmp cl, 0
        jz FIN
        add dx, ax
        dec cl
        jnz LAZO
FIN:    mov RES, dx
        hlt
        end
```

(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.*

```
        org 1000h
        NUM1 DB 1
        NUM2 DB 2
        RES DW ?

        org 3000h
MUL:    mov dx, 0
        cmp cl, 0
        jz FIN
        mov ah, 0
LAZO:   add dx, ax
        dec cl
        jnz LAZO
```

```
FIN:    ret

        org 2000h
        mov al, NUM1
        mov cl, NUM2
        call MUL
        mov RES, dx
        hlt
        end
```

(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.*

```
        org 1000h
        NUM1 DW 1
        NUM2 DW 2
        RES DW ?

        org 3000h
MUL:    mov dx, 0
        mov bx, ax
        mov ax, [bx]
        mov bx, cx
        mov cx, [bx]
        cmp cx, 0
        jz FIN
LAZO:   add dx, ax
        dec cx
        jnz LAZO
FIN:    ret

        org 2000h
        mov ax, offset NUM1
        mov cx, offset NUM2
        call MUL
        mov RES, dx
        hlt
        end
```

Ejercicio 7.

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.

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
MUL:    PUSH BX
        PUSH CX
        PUSH AX
        PUSH DX
        MOV BX, SP
        ADD BX, 12
        MOV CX, [BX]
        ADD BX, 2
        MOV AX, [BX]
        SUB BX, 4
        MOV BX, [BX]
        MOV DX, 0
SUMA:   ADD DX, AX
        DEC CX
        JNZ SUMA
        MOV [BX], DX
        POP DX
        POP AX
        POP CX
        POP BX
        RET

        ORG 1000H
        NUM1 DW 5H
        NUM2 DW 3H
        RES DW ?

        ORG 2000H
        MOV AX, NUM1
        PUSH AX
        MOV AX, NUM2
        PUSH AX
        MOV AX, OFFSET RES
        PUSH AX
        CALL MUL
        POP AX
```

```
POP AX
POP AX
HLT
END
```

(a) *¿Cuál es el modo de direccionamiento de la instrucción MOV AX, [BX]? ¿Qué se copia en el registro AX en este caso?*

El modo de direccionamiento de la instrucción MOV AX, [BX] es indirecto por registro y el valor que se copia en el registro AX, en este caso, es 5h.

(b) *¿Qué función cumple el registro temporal ri que aparece al ejecutarse una instrucción como la anterior?*

El registro temporal denominado "ri" cumple la función de guardar, temporalmente, la dirección contenida en BX para, luego, ir a buscar el contenido de la misma.

(c) *¿Qué se guarda en AX al ejecutarse MOV AX, OFFSET RES?*

En AX, al ejecutarse MOV AX, OFFSET RES, se guarda la dirección de la variable RES.

(d) *¿Cómo se pasa la variable RES a la pila, por valor o por referencia? ¿Qué ventaja tiene esto?*

La variable RES a la pila se pasa por referencia y la ventaja que tiene esto (versus pasarla a la pila por valor) es poder, luego, en la subrutina SUMA, usar direccionamiento indirecto para guardar el resultado en la dirección de la variable RES.

(e) *¿Cómo trabajan las instrucciones PUSH y POP?*

Las instrucciones PUSH y POP trabajan para el pasaje de parámetros y para preservar el contenido de los registros.

Ejercicio 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).*

```
                org 1000h
                CADENA DB "abcde"
                DB 0
                RES DW ?

                org 3000h
LONGITUD:       mov dx, 0
LAZO:           cmp byte ptr [bx], 0
                jz FIN
                inc dx
                inc bx
                jmp LAZO
FIN:            ret

                org 2000h
                mov bx, offset CADENA
                call LONGITUD
                mov RES, dx
                hlt
                end
```

(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.*

```
                org 1000h
                CADENA DB "aBcDe"
                DB 0
                RES DW ?

                org 3000h
CONTAR_MIN:     mov dx, 0
LAZO:           cmp byte ptr [bx], 0
                jz FIN
                mov al, [bx]
                cmp al, 123
                jns NO_ES_MIN
```

```
                cmp al, 97
                js NO_ES_MIN
                inc dx
NO_ES_MIN:      inc bx
                jmp LAZO
FIN:            ret

                org 2000h
                mov bx, offset CADENA
                call CONTAR_MIN
                mov RES, dx
                hlt
                end
```

(c) *Escribir la subrutina ES_VOCAL, que determina si un caracter es vocal o no, ya sea mayúscula o minúscula. La rutina debe recibir el caracter 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.*

```
                org 1000h
                VOCALES DB 65,69,73,79,85,97,101,105,111,117
                CHAR DB "A"
                RES DB ?

                org 3000h
ES_VOCAL:       mov ah, 00h
                mov cl, offset CHAR - offset VOCALES
                mov bx, offset VOCALES
LAZO:           cmp al, [bx]
                jz VOCAL
                inc bx
                dec cl
                jz FIN
                jmp LAZO
VOCAL:          mov ah, 0FFh
FIN:            ret

                org 2000h
                mov al, CHAR
                call ES_VOCAL
                mov RES, ah
                hlt
                end
```

(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.

```
                org 1000h
                VOCALES DB 65,69,73,79,85,97,101,105,111,117
                CADENA DB "AbCdE"
                DB 0
                RES DB ?

                org 3000h
ES_VOCAL:       push bx
                mov ah, 00h
                mov cl, offset CADENA - offset VOCALES
                mov bx, offset VOCALES
LAZO1:          cmp al, [bx]
                jz VOCAL
                inc bx
                dec cl
                jz FIN1
                jmp LAZO1
VOCAL:          mov ah, 0FFh
FIN1:           pop bx
                ret

                org 4000h
CONTAR_VOC:     mov dl, 0
LAZO2:          mov al, [bx]
                cmp al, 0
                jz FIN2
                call ES_VOCAL
                cmp ah, 0FFh
                jnz NO_ES_VOCAL
                inc dl
NO_ES_VOCAL:    inc bx
                jmp LAZO2
FIN2:           ret

                org 2000h
                mov bx, OFFSET CADENA
                call CONTAR_VOC
                mov RES, dl
                hlt
                end
```

(e) *Escribir la subrutina CONTAR_CAR que cuenta la cantidad de veces que aparece un caracter dado en una cadena terminada en cero. El caracter 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.

```
                org 1000h
                CADENA DB "AbCdE"
                DB 0
                CHAR DB "A"
                RES DB ?

                org 3000h
CONTAR_CAR:     mov ah, 0
LAZO:           cmp byte ptr [bx], 0
                jz FIN
                cmp al, [bx]
                jnz NO_ES_IGUAL
                inc ah
NO_ES_IGUAL:    inc bx
                jmp LAZO
FIN:            ret

                org 2000h
                mov al, CHAR
                mov bx, offset CADENA
                call CONTAR_CAR
                mov RES, ah
                hlt
                end
```

(f) *Escribir 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 caracter ORIGINAL por el caracter REEMPLAZO.*

```
                org 1000h
                ORIGINAL DB "A"
                REEMPLAZO DB "E"
                CADENA DB "AbCdE"
                DB 0

                org 3000h
REEMPLAZAR_CAR: push ax
                push bx
                mov bx, sp
                add bx, 8
                mov ax, [bx]
                mov bx, sp
                add bx, 6
```

```
                    mov bx, [bx]
LAZO:               cmp byte ptr [bx], 0
                    jz FIN
                    cmp byte ptr [bx], al
                    jnz NO_ES_IGUAL
                    mov [bx], ah
NO_ES_IGUAL:        inc bx
                    jmp LAZO
FIN:                pop bx
                    pop ax
                    ret

                    org 2000h
                    mov al, ORIGINAL
                    mov ah, REEMPLAZO
                    mov cx, offset CADENA
                    push ax
                    push cx
                    call REEMPLAZAR_CAR
                    pop cx
                    pop ax
                    hlt
                    end
```

Ejercicio 9.

(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. 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 10010100, se obtiene 00101001 y, al rotar a la izquierda 01101011, se obtiene 11010110. Para rotar a la izquierda un byte, se puede multiplicar el número por 2 o, lo que es lo mismo, sumarlo a sí mismo. 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.*

```
            org 1000h
            CADENA DB 10010100b

            org 3000h
ROTARIZQ:   add al, al
            adc al, 0
            mov CADENA, al
            ret

            org 2000h
            mov al, CADENA
            call ROTARIZQ
            hlt
            end
```

(b) *Usando la subrutina ROTARIZQ del ejercicio anterior, escribir 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 10010100, se obtiene el byte 01010010.*

```
            org 1000h
            CADENA DB 10010100b
            N DB 2

            org 3000h
ROTARIZQ:   add al, al
            adc al, 0
            ret

            org 4000h
ROTARIZQ_N: cmp ah, 0
            jz FIN
```

```
                call ROTARIZQ
                dec ah
                jmp ROTARIZQ_N
FIN:            mov CADENA, al
                ret

                org 2000h
                mov al, CADENA
                mov ah, N
                call ROTARIZQ_N
                hlt
                end
```

(c) *Usando la subrutina ROTARIZQ_N del ejercicio anterior, escribir 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 10010100, se obtiene el byte 01010010, que es equivalente a la rotación a la izquierda de 2 posiciones del ejemplo anterior.*

```
                org 1000h
                CADENA DB 10010100b
                N DB 2

                org 3000h
ROTARIZQ:       add al, al
                adc al, 0
                ret

                org 4000h
ROTARIZQ_N:     cmp ah, 0
                jz FIN
                call ROTARIZQ
                dec ah
                jmp ROTARIZQ_N
FIN:            mov CADENA, al
                ret

                org 5000h
ROTARDER_N:     mov cl, 8
                sub cl, ah
                mov ah, cl
                call ROTARIZQ_N
                ret

                org 2000h
                mov al, CADENA
                mov ah, N
```

```
                call ROTARDER_N
                hlt
                end
```

(d) *Escribir la subrutina ROTARDER del ejercicio anterior, pero sin usar la subrutina ROTARIZQ. Comparar qué ventajas tiene cada una de las soluciones.*

```
                org 1000h
                CADENA DB 10010100b
                N DB 2

                org 3000h
DIV:            cmp al, 0
                jz FIN1
                cmp al, 2
                jc FIN1
                sub al, 2
                jc FIN1
                inc cl
                jmp DIV
FIN1:           ret

                org 4000h
ROTARDER:       mov cl, 0
                call DIV
                cmp al, 1
                jnz FIN2
                add cl, 80h
FIN2:           ret

                org 5000h
ROTARDER_N:     cmp ah, 0
                jz FIN3
                call ROTARDER
                dec ah
                mov al, cl
                jmp ROTARDER_N
FIN3:           mov CADENA, cl
                ret

                org 2000h
                mov al, CADENA
                mov ah, N
                call ROTARDER_N
                hlt
                end
```

Ejercicio 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.

```
        org 1000h
        NUM1 DW 1234h
        NUM2 DW 5678h

        org 3000h
SWAP:   push ax
        push bx
        push cx
        push dx
        mov bx, sp
        add bx, 10
        mov bx, [bx]
        mov cx, [bx]
        mov bx, sp
        add bx, 12
        mov bx, [bx]
        mov dx, [bx]
        mov bx, sp
        add bx, 10
        mov bx, [bx]
        mov [bx], dx
        mov bx, sp
        add bx, 12
        mov bx, [bx]
        mov [bx], cx
        pop dx
        pop cx
        pop bx
        pop ax
        ret

        org 2000h
        mov ax, offset NUM1
        mov cx, offset NUM2
        push ax
        push cx
        call SWAP
        pop cx
        pop ax
        hlt
```

```
end
```

Ejercicio 11: Subrutinas de cálculo.

(a) *Escribir 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.*

```
        org 1000h
        NUM1 DB 10
        NUM2 DB 5
        RES DB ?

        org 3000h
DIV:    push ax
        push bx
        push cx
        mov cx, 0
        mov bx, sp
        add bx, 10
        mov ax, [bx]
LAZO:   sub al, ah
        js FIN
        inc cx
        jmp LAZO
FIN:    mov bx, sp
        add bx, 8
        mov bx, [bx]
        mov [bx], cx
        pop cx
        pop bx
        pop ax
        ret

        org 2000h
        mov al, NUM1
        mov ah, NUM2
        mov cx, offset RES
        push ax
        push cx
        call DIV
        pop cx
        pop ax
        hlt
        end
```

(b) *Escribir 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 valor.

```
        org 1000h
        NUM1 DB 10
        NUM2 DB 5

        org 3000h
RESTO:  mov cl, 0
        mov ch, 0
        cmp ah, 0
        jz FIN
        cmp al, 0
        jz FIN
DIV:    sub al, ah
        js RES
        inc cl
        jmp DIV
RES:    add al, ah
        mov ch, al
FIN:    ret

        org 2000h
        mov al, NUM1
        mov ah, NUM2
        call RESTO
        hlt
        end
```

(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.*

```
org 1000h
NUM1  DW 1,2
NUM2 DW 3,4
SUMA DW ?,?
DIR3 DW ?

org 2000h
mov ax, offset NUM1 + 2
mov cx, offset NUM2 + 2
mov DIR3, offset SUMA + 2
mov bx, ax
mov dx, [bx]
mov bx, cx
add dx, [bx]
```

```
pushf
mov bx, DIR3
mov [bx], dx
sub ax, 2
sub cx, 2
sub DIR3, 2
mov bx, ax
mov dx, [bx]
mov bx, cx
popf
adc dx, [bx]
mov bx, DIR3
mov [bx], dx
hlt
end
```

(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 referencia a través de la pila y devuelva el resultado también por referencia a través de la pila.*

```
            org 1000h
            NUM1  DW 1,2
            NUM2 DW 3,4
            SUMA DW ?,?

            org 3000h
SUM32:      push ax
            push bx
            push cx

            mov bx, sp
            add bx, 12
            mov bx, [bx]
            mov ax, [bx]

            mov bx, sp
            add bx, 10
            mov bx, [bx]
            mov cx, [bx]

            add ax, cx
            pushf

            mov bx, sp
            add bx, 10
            mov bx, [bx]
            mov [bx], ax
```

```
        mov bx, sp
        add bx, 14
        mov bx, [bx]
        sub bx, 2
        mov ax, [bx]

        mov bx, sp
        add bx, 12
        mov bx, [bx]
        sub bx, 2
        mov cx, [bx]

        popf
        adc ax, cx

        mov bx, sp
        add bx, 8
        mov bx, [bx]
        sub bx, 2
        mov [bx], ax

FIN:    pop cx
        pop bx
        pop ax
        ret

        org 2000h
        mov ax, offset NUM1 + 2
        mov cx, offset NUM2 + 2
        mov dx, offset SUMA + 2
        push ax
        push cx
        push dx
        call SUM32
        pop dx
        pop cx
        pop ax
        hlt
        end
```

Ejercicio 12.

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

```
        ORG 3000H
MUL:    CMP AX, 0
        JZ FIN
        ADD CX, AX
        DEC AX
        CALL MUL
FIN:    RET

        ORG 2000H
        MOV CX, 0
        MOV AX, 3
        CALL MUL
        HLT
        END
```

(a) *¿Qué hace la subrutina?*

La subrutina suma en CX todos los números comprendidos entre 0 y el valor del registro AX (3).

(b) *¿Cuál será el valor final de CX?*

El valor final de CX será 6.

(c) *Dibujar las posiciones de memoria de la pila, anotando qué valores va tomando.*

SP		**call MUL**	**call MUL**	**call MUL**	**call MUL**	**ret**	**ret**	**ret**	**ret**
7FF8h					0E	0E	0E	0E	0E
7FF9h					30	30	30	30	30
7FFAh				0E	0E	0E	0E	0E	0E
7FFBh				30	30	30	30	30	30
7FFCh			0E	0E	0E	0E	0E	0E	0E
7FFDh			30	30	30	30	30	30	30
7FFEh		0B	0B	0B	0B	0B	0B	0B	0B
7FFFh		20	20	20	20	20	20	20	20
8000h	?	?	?	?	?	?	?	?	?

(d) *¿Cuál será la limitación para determinar el valor más grande que se le puede pasar a la subrutina a través de AX?*

Para determinar el valor más grande que se le puede pasar a la subrutina a través de AX, se debe calcular el mínimo valor entre 255 (número que corresponde con que la suma entre 0 y ese valor sea igual a 32.767) y el tamaño de la pila (en bits) dividido 16.

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

Juan Menduiña

Trabajo Práctico N° 2:
Interrupciones.

Ejercicio 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 INFORMÁTICA - ”
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
```

Ejercicio 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.

```
        org 1000h
        CHAR DW 01h

        org 2000h
        mov al, 1
        mov bx, offset CHAR
LAZO:   int 7
        inc CHAR
        cmp CHAR, 256
        jnz LAZO
        int 0
        end
```

Ejercicio 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.

```
        org 1000h
        MAY DB 41h
        MIN DB 61h

        org 2000h
        mov bx, offset MAY
        mov al, 2
LAZO:   int 7
        inc MAY
        inc MIN
        cmp MAY, 5Bh
        jnz LAZO
        int 0
        end
```

Ejercicio 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 “INGRESAR UN NÚMERO (de un dígito): ”
FIN DB ?
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?*

En los registros BX y AL, se almacena la dirección de memoria del caracter inicial de MSJ y el tamaño del mensaje, respectivamente.

(b) *Con referencia a la interrupción INT 6, ¿qué se almacena en BX?*

En BX, se almacena la dirección de memoria de NUM.

(c) *En el programa anterior, ¿qué hace la segunda interrupción INT 7? ¿qué queda almacenado en el registro CL?*

La segunda interrupción INT 7 lo que hace es imprimir el número (de un dígito) que se ingresó por teclado. Lo que queda almacenado en registro CL es el código ASCII correspondiente al número (como caracter) ingresado por teclado.

Ejercicio 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 VÁLIDO". 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.

```
            org 1000h
            MSJ1 DB "INGRESAR UN NÚMERO (de un dígito): "
            FIN1 DB ?
            MSJ2 DB "CARACTER NO VÁLIDO"
            FIN2 DB ?
            NUM DB ?

            org 3000h
ES_NUM:     mov ah, 0FFh
            cmp byte ptr [bx], 30h
            js ERROR
            cmp byte ptr [bx], 3Ah
            jns ERROR
            jmp FIN
ERROR:      mov ah, 00h
            mov bx, offset MSJ2
            mov al, offset FIN2 - offset MSJ2
            int 7
FIN:        ret

            org 2000h
            mov bx, offset MSJ1
            mov al, offset FIN1 - offset MSJ1
            int 7
            mov bx, offset NUM
            int 6
            call ES_NUM
            mov al, 1
            int 7
            int 0
            end
```

Ejercicio 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.

```
            org 1000h
            CERO DB “CERO  ”
            DB “UNO   ”
            DB “DOS   ”
            DB “TRES  ”
            DB “CUATRO”
            DB “CINCO ”
            DB “SEIS  ”
            DB “SIETE ”
            DB “OCHO  ”
            DB “NUEVE ”
            MSJ DB “INGRESAR UN NÚMERO (de un dígito): ”
            FIN DB ?
            NUM DB ?

            org 2000h
            mov cl, 0
OTRO:       mov bx, offset MSJ
            mov al, offset FIN - offset MSJ
            int 7
            mov bx, offset NUM
            int 6
            cmp NUM, 30h
            jnz NO_CERO
            inc cl
            jmp SEGUIR
NO_CERO:    mov cl, 0
SEGUIR:     mov bx, offset CERO
            mov al, 6
LAZO:       cmp NUM, 30h
            jz IMPRIME
            add bx, 6
            dec NUM
            jmp LAZO
IMPRIMIR:   int 7
            cmp cl, 2
            jnz OTRO
            int 0
            end
```

Ejercicio 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.

```
            org 1000h
            MSJ1 DB “INGRESAR UN NÚMERO (de un dígito): ”
            FIN1 DB ?
            MSJ2 DB 10, “INGRESAR OTRO NÚMERO (de un dígito): ”
            FIN2 DB ?
            MSJ3 DB 10, “RESULTADO DE LA SUMA DE AMBOS NÚMEROS
      INGRESADOS: ”
            RES_D DB “0”
            RES_U DB ?
            FIN3 DB ?
            NUM1 DB ?
            NUM2 DB ?

            org 2000h
            mov bx, offset MSJ1
            mov al, offset FIN1 - offset MSJ1
            int 7
            mov bx, offset NUM1
            int 6
            mov al, 1
            int 7
            mov bx, offset MSJ2
            mov al, offset FIN2 - offset MSJ2
            int 7
            mov bx, offset NUM2
            int 6
            mov al, 1
            int 7
            mov al, NUM1
            sub al, 30h
            add al, NUM2
            cmp al, 3Ah
            js UNIDAD
            sub al, 10
            inc RES_D
UNIDAD:     mov RES_U, al
            mov bx, offset MSJ3
            mov al, offset FIN3 - offset MSJ3
            int 7
            int 0
            end
```

Ejercicio 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.

```
            org 1000h
            MSJ1 DB “NUM1: “
            FIN1 DB ?
            MSJ2 DB 10, “NUM2: ”
            FIN2 DB ?
            MSJ3 DB 10, “RESTA: ”
            SIGNO DB “+”
            RES DB ?
            FIN3 DB ?
            NUM1 DB ?
            NUM2 DB ?

            org 2000h
            mov bx, offset MSJ1
            mov al, offset FIN1 - offset MSJ1
            int 7
            mov bx, offset NUM1
            int 6
            mov al, 1
            int 7
            mov bx, offset MSJ2
            mov al, offset FIN2 - offset MSJ2
            int 7
            mov bx, offset NUM2
            int 6
            mov al, 1
            int 7
            mov al, NUM1
            mov ah, NUM2
            cmp al, ah
            js NEGATIVO
            sub al, ah
            mov RES, al
            jmp FIN
NEGATIVO:   sub ah, al
            mov RES, ah
FIN:        add RES, 30h
            mov bx, offset MSJ3
            mov al, offset FIN3 - offset MSJ3
            int 7
            int 0
            end
```

Ejercicio 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".

```
                org 1000h
                CLAVE_PRE DB "1234"
                MSJ1 DB "INGRESAR UNA CLAVE (de cuatro caracteres): "
                FIN1 DB ?
                MSJ2 DB 10, "ACCESO PERMITIDO"
                FIN2 DB ?
                MSJ3 DB 10, "ACCESO DENEGADO"
                FIN3 DB ?
                CLAVE DB ?,?,?,?

                org 2000h
                mov bx, offset MSJ1
                mov al, offset FIN1 - offset MSJ1
                int 7
                mov bx, offset CLAVE
                mov ah, 4
LAZO1:          int 6
                mov al, 1
                int 7
                inc bx
                dec ah
                cmp ah, 0
                jnz LAZO1
                mov al, CLAVE
                mov cx, 0
LAZO2:          mov bx, offset CLAVE_PRE
                add bx, cx
                mov dl, [bx]
                mov bx, offset CLAVE
                add bx, cx
                mov dh, [bx]
                cmp dl, dh
                jnz DENEGADO
                inc cx
                cmp cx, 4
                jnz LAZO2
                mov bx, offset MSJ2
                mov al, offset FIN2 - offset MSJ2
                int 7
                jmp FIN
DENEGADO:       mov bx, offset MSJ3
                mov al, offset FIN3 - offset MSJ3
```

```
        int 7
FIN:    int 0
        end
```

Ejercicio 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 3000H
RUT_F10:    PUSH AX
            INC DX
            MOV AL, EOI
            OUT EOI, AL
            POP AX
            IRET

            ORG 2000H
            CLI
            MOV AL, 0FEH
            OUT PIC+1, AL
            MOV AL, N_F10
            OUT PIC+4, AL
            MOV DX, 0
            STI
LAZO:       JMP LAZO
            END
```

Explicar detalladamente:

(a) *La función de los registros del PIC: ISR, IRR, IMR, INT0-INT7, EOI. Indicar la dirección de cada uno.*

La función de los registros del PIC es:

- ISR (23h): Indicar la interrupciones en ejecución.
- IRR (22h): Indicar la interrupciones pedidas.
- IMR (21h): Indicar la interrupciones habilitadas.
- INT0-INT7 (24h-31h): Indicar el ID de interrupción de cada dispositivo.
- EOI (20h): Indicar la finalización de la interrupción.

(b) *Cuáles de estos registros son programables y cómo trabaja la instrucción OUT.*

De estos registros, son programables el IMR, INT0-INT7 y el EOI. La instrucción OUT trabaja moviendo contenido del registro AL al PIC.

(c) *Qué hacen y para qué se usan las instrucciones CLI y STI.*

Las instrucciones CLI y STI se usan para deshabilitar y habilitar interrupciones, respectivamente.

Ejercicio 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.

```
            EOI EQU 20h
            IMR EQU 21h
            INT0 EQU 24h
            N_F10 EQU 10

            org 40
            IP_F10 DW RUT_F10

            org 1000h
            LETRA DB ?

            org 3000h
RUT_F10:    push ax
            push bx
            inc cl
            mov LETRA, ah
            mov bx, offset LETRA
            mov al, 1
            int 7
            mov al, EOI
            out EOI, al
            pop bx
            pop ax
            iret

            org 4000h
ABCDE:      mov ah, 65
LAZO:       cmp cl, 1
            jz FIN
            inc ah
            cmp ah, 90
            jnz LAZO
            jmp ABCDE
FIN:        ret

            org 2000h
            cli
            mov al, 0FEh
            out IMR, al
            mov al, N_F10
            out INT0, al
            mov cl, 0
```

```
sti
call ABCDE
int 0
end
```

Ejercicio 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
            MOV AL, N_CLK
            OUT PIC+5, AL
            MOV AL, 10
            OUT TIMER+1, AL
            MOV AL, 0
            OUT TIMER, AL
            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.*

El Timer es otro dispositivo de ES como el F10. Se utiliza como un reloj despertador para la CPU. Se configura para contar una cantidad determinada de segundos y, cuando finaliza la cuenta, emite una interrupción. El Timer tiene dos registros, CONT (registro contador) y COMP (registro de comparación), con direcciones de la memoria de ES 10h y 11h, respectivamente.

(b) *La función que cumplen sus registros, la dirección de cada uno y cómo se programan.*

La función de los registros del Timer es:

- CONT (10h): Se incrementa, automáticamente, una vez por segundo, para contar tiempo transcurrido.
- COMP (11h): Contiene el tiempo límite del Timer. Cuando CONT vale igual que COMP, se dispara la interrupción.

Ejercicio 13.

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

```
            CONT EQU 10h
            COMP EQU 11h
            EOI EQU 20h
            IMR EQU 21h
            INT1 EQU 25h
            N_CLK EQU 10

            org 40
            IP_CLK DW RUT_CLK

            org 1000h
            MIN_D DB 30h
            MIN_U DB 30h, 58
            SEG_D DB 30h
            SEG_U DB 30h, 32
            FIN DB ?

            org 3000h
RUT_CLK:    push ax
            inc SEG_D
            cmp SEG_D, 36h
            jnz IMPRIMIR
            mov SEG_D, 30h
            inc MIN_U
            cmp MIN_U, 3Ah
            jnz IMPRIMIR
            mov MIN_U, 30h
            inc MIN_D
            cmp MIN_D, 36h
            jnz IMPRIMIR
            mov MIN_D, 30h
IMPRIMIR:   int 7
            mov al, 0
            out CONT, al
            mov al, EOI
            out EOI, al
            pop ax
            iret

            org 2000h
            cli
            mov al, 0FDh
            out IMR, al
            mov al, N_CLK
```

```
        out INT1, al
        mov al, 0
        out CONT, al
        mov al, 10
        out COMP, al
        mov bx, offset MIN_D
        mov al, offset FIN - offset MIN_D
        sti
LAZO:   jmp LAZO
        int 0
        end
```

Ejercicio 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.

```
            CONT EQU 10h
            COMP EQU 11h
            EOI EQU 20h
            IMR EQU 21h
            INT0 EQU 24h
            INT1 EQU 25h
            N_CLK EQU 10
            N_F10 EQU 20

            org 40
            IP_CLK DW RUT_CLK

            org 80
            IP_F10 DW RUT_F10

            org 1000h
            SEG_D DB 30h
            SEG_U DB 30h, 32
            FIN DB ?

            org 3000h
RUT_CLK:    push ax
            inc SEG_U
            cmp SEG_U, 3Ah
            jnz IMPRIMIR
            mov SEG_U, 30h
            inc SEG_D
            cmp SEG_D, 33h
            jnz IMPRIMIR
            mov cl, 1
            mov al, 0FFh
            out IMR, al
IMPRIMIR:   int 7
            mov al, 0
            out CONT, al
            mov al, EOI
            out EOI, al
            pop ax
            iret

            org 4000h
RUT_F10:    push ax
            in al, IMR
```

```
        xor al, 00000010b
        out IMR, al
        mov al, EOI
        out EOI, al
        pop ax
        iret

        org 2000h
        cli
        mov al, 0FCh
        out IMR, al
        mov al, N_F10
        out INT0, al
        mov al, N_CLK
        out INT1, al
        mov al, 0
        out CONT, al
        mov al, 10
        out COMP, al
        mov bx, offset SEG_D
        mov al, offset FIN - offset SEG_D
        mov cl, 0
        sti
LAZO:   cmp cl, 0
        jz LAZO
        int 0
        end
```

Ejercicio 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.

```
            CONT EQU 10h
            COMP EQU 11h
            EOI EQU 20h
            IMR EQU 21h
            INT0 EQU 24h
            INT1 EQU 25h
            N_CLK EQU 10
            N_F10 EQU 20

            org 40
            IP_CLK DW RUT_CLK

            org 80
            IP_F10 DW RUT_F10

            org 1000h
            NUM DB ?, 32
            FIN_NUM DB ?

            org 3000h
RUT_CLK:    push ax
            int 7
            dec cl
            cmp cl, 30h
            jns SEGUIR
            mov al, 0FFh
            out IMR, al
            jmp FIN
SEGUIR:     mov NUM, cl
            mov al, 0
            out CONT, al
FIN:        mov al, EOI
            out EOI, al
            pop ax
            iret

            org 4000h
RUT_F10:    push ax
            in al, IMR
            xor al, 00000010b
            out IMR, al
            mov al, EOI
            out EOI, al
```

```
        pop ax
        iret

        org 2000h
        cli
        mov al, 0FEh
        out IMR, al
        mov al, N_F10
        out INT0, al
        mov al, N_CLK
        out INT1, al
        mov al, 0
        out CONT, al
        mov al, 10
        out COMP, al
        mov bx, offset NUM
        int 6
        mov bx, offset NUM
        mov al, offset FIN_NUM - offset NUM
        mov cl, NUM
        sti
LAZO:   cmp cl, 30h
        jns LAZO
        int 0
        end
```

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

Juan Menduiña

Trabajo Práctico N° 3:
Entrada/Salida.

Ejercicio 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 sólo las primeras y las últimas dos luces deben prenderse y el resto deben apagarse.*

```
PB EQU 31h
CB EQU 33h

org 1000h
PATRON DB 11000011b

org 2000h
mov al, 0
out CB, al
mov al, PATRON
out PB, al
int 0
end
```

(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". Sólo 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.*

```
PA EQU 30h
CA EQU 32h

org 1000h
MSJ1 DB "Llave prendida"
FIN1 DB ?
MSJ2 DB "Llave apagada"
FIN2 DB ?

org 2000h
mov al, 0FFh
out CA, al
in al, PA
and al, 80h
cmp al, 0
jz APAGADA
```

```
            mov bx, offset MSJ1
            mov al, offset FIN1 - offset MSJ1
            jmp FIN
APAGADA:    mov bx, offset MSJ2
            mov al, offset FIN2 - offset MSJ2
FIN:        int 7
            int 0
            end
```

(c) *Escribir un programa que permita 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 sólo la primera llave está encendida, entonces, sólo la primera luz se debe quedar encendida.*

```
            PA EQU 30h
            PB EQU 31h
            CA EQU 32h
            CB EQU 33h

            org 2000h
            mov al, 0FFh
            out CA, al
            mov al, 0
            out CB, al
POLL:       in al, PA
            out PB, al
            jmp POLL
            int 0
            end
```

(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 sólo aquellas luces donde el valor de las llaves es 1. Entonces, primero, se enciende sólo 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.*

```
            CONT EQU 10h
            COMP EQU 11h
            EOI EQU 20h
            IMR EQU 21h
            INT1 EQU 25h
            PB EQU 31h
```

```
            CB EQU 33h
            N_CLK EQU 10

            org 40
            IP_CLK DW RUT_CLK

            org 3000h
RUT_CLK:    push ax
            mov ax, cx
            out PB, ax
            inc cx
            cmp cx, 256
            jnz SEGUIR
            mov al, 0FFh
            out IMR, al
            jmp FIN
SEGUIR:     mov al, 0
            out CONT, al
FIN:        mov al, EOI
            out EOI, al
            pop ax
            iret

            org 2000h
            cli
            mov al, 0FDH
            out IMR, al
            mov al, N_CLK
            out INT1, al
            mov al, 0
            out CONT, al
            mov al, 10
            out COMP, al
            mov al, 0
            out CB, al
            mov cx, 0
            sti
LAZO:       cmp cx, 256
            jnz LAZO
            int 0
            end
```

(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.*

Opción 1:

```
            PB EQU 31h
            CB EQU 33h

            org 1000h
            PATRON DB 0,1,2,4,8,16,32,64,128

            org 2000h
            mov bx, offset PATRON
            mov al, 0
            out CB, al
CRECER:     inc bx
            mov al, [bx]
            out PB, al
            cmp byte ptr [bx], 128
            jnz CRECER
DECRECER:   dec bx
            mov al, [bx]
            out PB, al
            cmp byte ptr [bx], 1
            jnz DECRECER
            jmp CRECER
            int 0
            end
```

Opción 2:

```
            CONT EQU 10h
            COMP EQU 11h
            EOI EQU 20h
            IMR EQU 21h
            INT1 EQU 25h
            PB EQU 31h
            CB EQU 33h
            N_CLK EQU 10

            org 40
            IP_CLK DW RUT_CLK

            org 1000h
            PATRON DB 0,1,2,4,8,16,32,64,128

            org 3000h
RUT_CLK:    push ax
            mov al, 0
            out CONT, al
            mov al, EOI
            out EOI, al
            pop ax
```

```
                iret

                org 2000h
                cli
                mov al, 0FDh
                out IMR, al
                mov al, N_CLK
                out INT1, al
                mov al, 0
                out CONT, al
                mov al, 10
                out COMP, al
                mov al, 0
                out CB, al
                mov bx, offset PATRON
                sti
CRECER:         cli
                inc bx
                mov al, [bx]
                out PB, al
                sti
                cmp byte ptr [bx], 128
                jnz CRECER
DECRECER:       cli
                dec bx
                mov al, [bx]
                out PB, al
                sti
                cmp byte ptr [bx], 1
                jnz DECRECER
                jmp CRECER
                int 0
                end
```

<u>Ejercicio 2:</u> Uso de la impresora a través del 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.*

```
            PA EQU 30h
            PB EQU 31h
            CA EQU 32h
            CB EQU 33h

            org 1000h
            CHAR DB "A"

            org 3000h
PIO:        push ax
            mov al, 1
            out CA, al
            mov al, 0
            out CB, al
            pop ax
            ret

            org 4000h
STROBE0:    push ax
            in al, PA
            and al, 11111101b
            out PA, al
            pop ax
            ret

            org 5000h
STROBE1:    push ax
            in al, PA
            or al, 00000010b
            out PA, al
            pop ax
            ret

            org 6000h
POLL:       push ax
            in al, PA
            and al, 1
            jnz POLL
```

```
        pop ax
        ret

        org 2000h
        call PIO
        call STROBE0
        call POLL
        mov al, CHAR
        out PB, al
        call STROBE1
        nop
        nop
        nop
        nop
        nop
        int 0
        end
```

(b) *Escribir un programa para imprimir el mensaje "ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS" utilizando la impresora a través de la PIO.*

```
        PA EQU 30h
        PB EQU 31h
        CA EQU 32h
        CB EQU 33h

        org 1000h
        MSJ DB "ORGANIZACIÓN Y ARQUITECTURA DE
    COMPUTADORAS"
        FIN DB ?

        org 3000h
PIO:    push ax
        mov al, 1
        out CA, al
        mov al, 0
        out CB, al
        pop ax
        ret

        org 4000h
STROBE0: push ax
        in al, PA
        and al, 11111101b
        out PA, al
        pop ax
        ret
```

```
            org 5000h
STROBE1:    push ax
            in al, PA
            or al, 00000010b
            out PA, al
            pop ax
            ret

            org 6000h
POLL:       push ax
            in al, PA
            and al, 1
            jnz POLL
            pop ax
            ret

            org 2000h
            call PIO
            call STROBE0
            mov bx, offset MSJ
            mov cl, offset FIN - offset MSJ
LAZO:       call POLL
            mov al, [bx]
            out PB, al
            call STROBE1
            call STROBE0
            inc bx
            dec cl
            jnz LAZO
            int 0
            end
```

(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.*

```
            PA EQU 30h
            PB EQU 31h
            CA EQU 32h
            CB EQU 33h

            org 1000h
            CHAR DB ?
            CHARS DB 5

            org 3000h
PIO:        push ax
            mov al, 1
```

```
            out CA, al
            mov al, 0
            out CB, al
            pop ax
            ret

            org 4000h
STROBE0:    push ax
            in al, PA
            and al, 11111101b
            out PA, al
            pop ax
            ret

            org 5000h
STROBE1:    push ax
            in al, PA
            or al, 00000010b
            out PA, al
            pop ax
            ret

            org 6000h
POLL:       push ax
            in al, PA
            and al, 1
            jnz POLL
            pop ax
            ret

            org 2000h
            call PIO
            call STROBE0
            mov bx, offset CHAR
            mov cl, CHARS
LAZO:       int 6
            call POLL
            mov al, [bx]
            out PB, al
            call STROBE1
            call STROBE0
            dec cl
            jnz LAZO
            int 0
            end
```

(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.*

```
            EOI EQU 20h
            IMR EQUI 21h
            INT0 EQU 24h
            PA EQU 30h
            PB EQU 31h
            CA EQU 32h
            CB EQU 33h
            N_F10 EQU 10

            org 40
            ID_F10 DW RUT_F10

            org 1000h
            CADENA DB ?

            org 3000h
RUT_F10:    push ax
            mov ch, 1
            mov al, 0FFh
            out IMR, al
            mov al, EOI
            out EOI, al
            pop ax
            iret

            org 4000h
PIO:        push ax
            mov al, 1
            out CA, al
            mov al, 0
            out CB, al
            pop ax
            ret

            org 4500h
STROBE0:    push ax
            in al, PA
            and al, 11111101b
            out PA, al
            pop ax
            ret

            org 5000h
STROBE1:    push ax
            in al, PA
            or al, 00000010b
            out PA, al
            pop ax
            ret
```

```
        org 5500h
POLL:   push ax
        in al, PA
        and al, 1
        jnz POLL
        pop ax
        ret

        org 6000h
PIC:    push ax
        mov al, 0FEh
        out IMR, al
        mov al, N_F10
        out INT0, al
        pop ax
        ret

        org 2000h
        cli
        call PIO
        call STROBE0
        call PIC
        sti
        mov bx, offset CADENA
        mov cl, 0
        mov ch, 0
LAZO1:  int 6
        inc bx
        inc cl
        cmp ch, 1
        jnz LAZO1
        mov bx, offset CADENA
LAZO2:  call POLL
        mov al, [bx]
        out PB, al
        call STROBE1
        call STROBE0
        inc bx
        dec cl
        jnz LAZO2
        int 0
        end
```

Ejercicio 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 imprima "INGENIERÍA E INFORMÁTICA" en la impresora a través del HAND-SHAKE. La comunicación se establece por consulta de estado (polling). ¿Qué diferencias se encuentran con el Ejercicio 2b?*

```
        DATO EQU 40h
        ESTADO EQU 41h

        org 1000h
        MSJ DB "INGENIERÍA E INFORMÁTICA"
        FIN DB ?

        org 2000h
        mov bx, offset MSJ
        mov cl, offset FIN - offset MSJ
POLL:   in al, ESTADO
        and al, 1
        jnz POLL
        mov al, [bx]
        out DATO, al
        inc bx
        dec cl
        jnz POLL
        int 0
        end
```

Las diferencias que se encuentran con el Ejercicio 2b son que no es necesario configurar el PIO ni tampoco es necesario configurar las señales de strobe.

(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?*

La ventaja en utilizar el HAND-SHAKE con respecto al PIO para comunicarse con la impresora es que manda señal de strobe automáticamente. Sacando eso de lado, las ventajas que tiene el PIO, en general, con respecto al HAND-SHAKE, es que sirve para comunicarse con otros dispositivos.

(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.*

```
            EOI EQU 20h
            IMR EQU 21h
            INT2 EQU 26h
            DATO EQU 40h
            ESTADO EQU 41h
            N_HSK EQU 10

            org 40
            IP_HSK DW RUT_HSK

            org 1000h
            MSJ DB “UNIVERSIDAD NACIONAL DE LA PLATA”
            FIN_MSJ DB ?

            org 3000h
RUT_HSK:    push ax
            mov al, [bx]
            out DATO, al
            inc bx
            dec cl
            cmp cl, 0
            jnz FIN
            mov al, 0FFh
            out IMR, al
            in al, ESTADO
            and al, 0111111b
            out ESTADO, al
FIN:        mov al, EOI
            out EOI, al
            pop ax
            iret

            org 2000h
            cli
            mov al, 0FBh
            out IMR, al
            mov al, N_HSK
            out INT2, al
            in al, ESTADO
            or al, 10000000b
            out ESTADO, al
            mov bx, offset MSJ
            mov cl, offset FIN_MSJ - offset MSJ
            sti
LAZO:       cmp cl, 0
            jnz LAZO
            int 0
            end
```

(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 se encuentran con el Ejercicio 2c?*

```
        DATO EQU 40h
        ESTADO EQU 41h

        org 1000h
        CHARS DB 5
        CADENA DB ?,?,?,?,?
        FIN DB ?

        org 2000h
        mov bx, offset CADENA
        mov cl, CHARS
LAZO:   int 6
        inc bx
        dec cl
        cmp cl, 0
        jnz LAZO
        mov bx, offset CADENA
        mov cl, offset FIN - offset CADENA
POLL1:  in al, ESTADO
        and al, 1
        jnz POLL1
        mov al, [bx]
        out DATO, al
        inc bx
        dec cl
        jnz POLL1
        mov bx, offset CADENA+4
        mov cl, offset FIN - offset CADENA
POLL2:  in al, ESTADO
        and al, 1
        jnz POLL2
        mov al, [bx]
        out DATO, al
        dec bx
        dec cl
        jnz POLL2
        int 0
        end
```

Las diferencias que se encuentran con el Ejercicio 2c son que los cinco caracteres son enviados a la impresora todos a la vez y no de a uno por vez.

(e) *Idem (d), pero, ahora, utilizar el HAND-SHAKE en modo interrupciones.*

```
            EOI EQU 20h
            IMR EQU 21h
            INT2 EQU 26h
            DATO EQU 40h
            ESTADO EQU 41h
            N_HSK EQU 10

            org 40
            IP_HSK DW RUT_HSK

            org 1000h
            CADENA DB ?,?,?,?,?
            FIN_CADENA DB ?

            org 3000h
RUT_HSK:    push ax
            mov al, [bx]
            out DATO, al
            dec ch
            cmp ch, 6
            js DESC
            inc bx
            jmp SEGUIR
DESC:       cmp ch, 5
            jz SEGUIR
            dec bx
SEGUIR:     dec cl
            cmp cl, 0
            jnz FIN
            mov al, 0FFh
            out IMR, al
            in al, ESTADO
            and al, 0111111b
            out ESTADO, al
FIN:        mov al, EOI
            out EOI, al
            pop ax
            iret

            org 2000h
            cli
            mov al, 0FBh
            out IMR, al
            mov al, N_HSK
            out INT2, al
            in al, ESTADO
```

```
        or al, 10000000b
        out ESTADO, al
        mov bx, offset CADENA
        mov cl, offset FIN_CADENA - offset CADENA
LAZO1:  int 6
        inc bx
        dec cl
        cmp cl, 0
        jnz LAZO1
        mov bx, offset CADENA
        mov cl, 10
        mov ch, 10
        sti
LAZO2:  cmp cl, 0
        jnz LAZO2
        int 0
        end
```

<u>Ejercicio 4:</u> 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.*

<u>Ejercicio 5:</u> 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.

(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?*

<u>Ejercicio 6:</u> 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.

(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?*

Ejercicio 7: Configuración del CDMA.

Indicar 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.
```

Juan Menduiña

Trabajo Práctico N° 4:
Segmentación de Cauce en Procesador RISC.

Ejercicio 1.

Muchas instrucciones comunes en procesadores con arquitectura RISC 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.*

La tarea que realiza esta instrucción es sumar r2 y r0 (0) y guardar el resultado en r1. Su equivalente en lenguaje Assembly del x86 es *mov r1, r2*.

(b) *daddi r3, r0, 5.*

La tarea que realiza esta instrucción es sumar r0 (0) y 5 y guardar el resultado en r3. Su equivalente en lenguaje Assembly del x86 es *mov r3, 5*.

(c) *dsub r4, r4, r4.*

La tarea que realiza esta instrucción es restar r4 menos r4 y guardar el resultado en r4. Su equivalente en lenguaje Assembly del x86 es *mov r4, 0*.

(d) *daddi r5, r5, -1.*

La tarea que realiza esta instrucción es sumar r5 más -1 y guardar el resultado en r5. Su equivalente en lenguaje Assembly del x86 es *dec r5*.

(e) *xori r6, r6, 0xffffffffffffffff.*

La tarea que realiza esta instrucción es hacer un XOR entre r6 y ffffffffffffffff y guardar el resultado en r6. Su equivalente en lenguaje Assembly del x86 es *XOR r6, 0FFFFh*.

Ejercicio 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) *Ejecutar 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 (o 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?*

La instrucción que está generando atascos (*stalls*) en el cauce (o *pipeline*) es *sd r2, A(r0)* (en su etapa ID) porque necesita que la instrucción *ld r2, B(r0)* finalice su etapa WB.

El tipo de '*stall*' que aparece es RAW (*Read After Write*).

El promedio de Ciclos Por Instrucción (CPI) en la ejecución de este programa bajo esta configuración es 2,2.

(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.*

En este caso, no se presenta ningún atasco porque el dato contenido en el registro R2 podrá ser leído por la instrucción *sd r2, A(r0)* (en su etapa MEM) cuando la instrucción *ld r2, B(r0)* se encuentra finalizando su etapa MEM, es decir, la instrucción *sd r2, A(r0)* (en su etapa ID) no tiene que esperar a que la instrucción *ld r2, B(r0)* finalice su etapa WB, por lo que no aparecen atascos del tipo RAW.

El color de los registros en la ventana Register durante la ejecución indica que el dato (registro R1) está disponible en etapa MEM para adelantamiento. Además, los registros pueden tener color rojo, indicando que el resultado está disponible en EX y puede ser adelantado. Si el color es gris, el valor no está disponible en este ciclo para adelantamiento.

El promedio de Ciclos Por Instrucción (CPI) en la ejecución de este programa, bajo esta configuración, es 1,8, menor que el anterior.

Ejercicio 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.*

Se presentan atascos tipo RAW porque la instrucción *bnez r2, loop* necesita, en su etapa ID, del contenido del registro R2, que está siendo utilizado por la instrucción *daddi r2, r2, -1* (en su etapa EX).

Branch Taken significa que se produjo una incorrecta ejecución de la instrucción siguiente a una instrucción condicional y se produce porque la condición a evaluar tarda algunos ciclos en ser ejecutada, mientras que, durante esos ciclos, siguen entrando nuevas instrucciones al *pipeline*. Luego de evaluada la condición, si la instrucción posterior a ésta que se ejecutó no es la que debía ser ejecutada, su ejecución se trunca y se ejecuta la que está en el lugar de memoria indicada por la etiqueta en la instrucción condicional.

La ejecución de este programa tiene 1,75 CPI (21 ciclos y 12 instrucciones).

(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.*

Las instrucciones que generan los atascos tipo RAW son *dsll r1, r1, 1* (en su etapa ID) y *bnez r2, loop* (en su etapa ID). La primera porque necesita que la instrucción *ld r1, A(r0)* finalice su etapa WB y la segunda porque necesita que la instrucción *daddi r2, r2, -1* finalice su etapa WB. En el primer caso, el atasco se produce durante 1 ciclo y, en el segundo caso, durante 2 ciclos.

La cantidad de ciclos que dura el atasco *Branch Taken Stalls* en cada vuelta del lazo *"loop"* es 2. La diferencia con la ejecución con *Forwarding* se debe a que, en ese caso, el dato contenido en el registro R2 podrá ser leído por la instrucción *bnez r2, loop* (en su etapa ID) cuando la instrucción *daddi r2, r2, -1* se encuentra finalizando su etapa MEM y no su etapa WB (como sucede sin *Forwarding*).

En este caso, la ejecución del programa tiene 2,083 CPI (25 ciclos y 12 instrucciones).

(c) *Reordenar las instrucciones para que la cantidad de RAW sea "0" en la ejecución del programa (Forwarding habilitado).*

```
        .data
A:      .word 1
B:      .word 3

        .code
        ld r2, B(r0)
        ld r1, A(r0)
LOOP:   daddi r2, r2, -1
        dsll r1, r1, 1
        bnez r2, LOOP
        halt
```

(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?*

```
        .data
A:      .word 1
B:      .word 3
C:      .word 0, 0, 0

        .code
        ld r2, B(r0)
        ld r1, A(r0)
        daddi r3, r0, 0
LOOP:   sd r1, C(r3)
        dsll r1, r1, 1
        daddi r2, r2, -1
        daddi r3, r3, 8
```

```
        bnez r2, LOOP
        halt
```

Los elementos de la tabla que se genera hacen referencia al número decimal que representa R1 previo a cada uno de los tres corrimientos hacia la izquierda.

Ejercicio 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?*

La tarea que realiza este programa es buscar el número 7 en la tabla con 10 números y, al encontrarlo, poner el valor 1 en el registro R10.

(b) *Re-ejecutar el programa con la opción Configure/Enable Branch Target Buffer habilitada. Explicar la ventaja de usar este método y cómo trabaja.*

La ventaja de utilizar la opción *Branch Target Buffer* es reducir a 4 los atascos tipo BTS (*Branch Taken Stall*). Esta opción carga la dirección del último salto. Es un algoritmo de predicción para cargar la próxima instrucción. Si nunca se ejecutó, carga la siguiente instrucción, sino carga instrucción de la tabla, la cual se actualiza cuando sucede un atasco tipo BTS. Tener en cuenta que esta opción es útil cuando aumenta la cantidad de iteraciones de un lazo.

(c) *Confeccionar una tabla que compare número de ciclos, CPI, RAWs y Branch Taken Stalls para los dos casos anteriores.*

	Inciso (a)	**Inciso (b)**
Número de ciclos	71	67
CPI	1,651	1,558
RAWs	16	16
Branch Taken Stalls	8	4

Ejercicio 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)?*

El efecto que tiene habilitar la opción *Delay Slot* (salto retardado) es saltar un ciclo después, por lo que ejecuta siempre la instrucción siguiente al salto y hay 0 atascos tipo BTS siempre.

(b) *¿Con qué fin se incluye la instrucción NOP? ¿Qué sucedería si no estuviera?*

La instrucción NOP se incluye con el fin de no modificar el funcionamiento del programa, como solución simple al *Delay Slot*. Si no estuviera, el programa finalizaría a causa del HLT.

(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.*

```
            .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)
        bnez r2, LOOP
        daddi r1, r1, 8
        halt
```

	Inciso (a)	**Inciso (d)**
Número de ciclos	63	55
Instrucciones	59	51
CPI	1,068	1,078

Ejercicio 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.

```
            .data
A:          .word 1
B:          .word 2
C:          .word 3
D:          .word 0

            .code
            ld r1, A(r0)
            ld r2, B(r0)
            ld r3, C(r0)
            dadd r4, r0, r0
            bne r1, r2, NOIGUAL1
            daddi r4, r4, 1
NOIGUAL1:   bne r1, r3, NOIGUAL2
            daddi r4, r4, 1
            j fin2
NOIGUAL2:   bnez r4, FIN2
            bne r2, r3, FIN3
            daddi r4, r4, 2
            j FIN3
FIN1:       beqz r4, FIN2
FIN2:       daddi r4, r4, 1
FIN3:       sd r4, D(r0)
            halt
```

Ejercicio 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.

```
            .data
TABLA:      .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
RES:        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
X:          .word 5
TAM:        .word 10
CANT:       .word 0

            .code
            ld r1, TAM(r0)
            ld r2, X(r0)
            dadd r3, r0, r0
            dadd r4, r0, r0
            daddi r5, r0, 1
            daddi r2, r2, 1
LAZO:       ld r6, TABLA(r3)
            slt r7, r6, r2
            bnez r7, MENOR
            sd r5, RES(r3)
            daddi r4, r4, 1
MENOR:      daddi r1, r1, -1
            daddi r3, r3, 8
            bnez r1, LAZO
            sd r4, CANT(r0)
            halt
```

Ejercicio 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.

```
        .data
NUM1:   .word 1
NUM2:   .word 2
RES:    .word 0

        .code
        ld r1, NUM1(r0)
        ld r2, NUM2(r0)
        dadd r3, r0, r0
LAZO:   daddi r2, r2, -1
        dadd r3, r3, r1
        bnez r2, LAZO
        sd r3, RES(r0)
        halt
```

Ejercicio 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.

```
            .data
A:          .word 5
X:          .word 0
Y:          .word 5

            .code
            ld r1, A(r0)
            ld r2, X(r0)
            ld r3, Y(r0)
LAZO:       beqz r1, FIN
            daddi r1, r1, -1
            dadd r2, r2, r3
            j LAZO
FIN:        sd r2, X(r0)
            halt
```

Ejercicio 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
```

```
            .data
CADENA:     .asciiz  "adbdcdedfdgdhdid"
CAR:        .asciiz  "d"
CANT:       .word 0

            .code
            dadd r1, r0, r0
            dadd r2, r0, r0
            lbu r3, CAR(r0)
LAZO:       lbu r4, CADENA(r1)
            beqz r4, FIN
            bne r3, r4, NOIGUAL
            daddi r2, r2, 1
NOIGUAL:    daddi r1, r1, 1
            j LAZO
FIN:        sd r2, CANT(r0)
            halt
```

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.

Juan Menduiña

Trabajo Práctico N° 5:
Procesador RISC (Instrucciones de Punto Flotante y Pasaje de Parámetros).

Ejercicio 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
            l.d f1, N1(r0)
            l.d f2, N2(r0)
            add.d f3, f2, f1
            mul.d f4, f2, f1
            s.d f3, RES1(r0)
            s.d f4, RES2(r0)
            halt
```

(a) *Tomar nota de la cantidad de ciclos, instrucciones y CPI luego de la ejecución del programa.*

Ciclos: 16.
Instrucciones: 7.
CPI: 2,286.

(b) *¿Cuántos atascos por dependencia de datos se generan? Observar, en cada caso, cuál es el dato en conflicto y las instrucciones involucradas.*

Se generan 4 atascos por dependencia de datos RAW. El dato en conflicto y las instrucciones involucradas son:

- 1 RAW: *add.d f3, f2, f1* (en su etapa A0, debe esperar que la instrucción *l.d f2, N2(r0)*) finalice su etapa MEM.
- 2 RAW: *s.d f3, RES1(r0)* (en su etapa EX, debe esperar que la instrucción *add.d f3, f2, f1* finalice su etapa A3)
- 1 RAW: *s.d f4, RES2(r0)* (en su etapa EX, debe esperar que la instrucción *mul.d f4, f2, f1* finalice su etapa M6).

(c) *¿Por qué se producen los atascos estructurales? Observar cuáles son las instrucciones que los generan y en qué etapas del pipeline aparecen.*

Los atascos estructurales se producen por conflictos por los recursos. Las instrucciones que las generan y las etapas del *pipeline* que aparecen son:

- 1 Atasco Estructural: *s.d f3, RES1(r0)* (la etapa MEM de esta instrucción se encuentra, al mismo tiempo, con la etapa MEM de la instrucción *add.d f3, f2, f1*).
- 1 Atasco Estructural: *s.d f4, RES2(r0)* (la etapa MEM de esta instrucción se encuentra, al mismo tiempo, con la etapa MEM de la instrucción *mul.d f4, f2, f1*).

(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?*

```
            .data
N1:         .double 9.13
N2:         .double 6.58
RES1:       .double 0.0
RES2:       .double 0.0

            .code
            l.d f1, N1(r0)
            l.d f2, N2(r0)
            add.d f3, f2, f1
            mul.d f1, f2, f1
            mul.d f4, f2, f1
            s.d f3, RES1(r0)
            s.d f4, RES2(r0)
            halt
```

Aparece un atasco tipo WAR porque la instrucción *mul.d f1, f2, f1* (en su etapa ID) necesita escribir el registro F1 que aún la instrucción *add.d f3, f2, f1* (en su etapa A0) no leyó.

(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.*

```
            .data
N1:         .double 9.13
N2:         .double 6.58
RES1:       .double 0.0
RES2:       .double 0.0

            .code
```

```
l.d f1, N1(r0)
l.d f2, N2(r0)
nop
add.d f3, f2, f1
mul.d f1, f2, f1
mul.d f4, f2, f1
s.d f3, RES1(r0)
s.d f4, RES2(r0)
halt
```

Colocando un NOP antes de la suma se soluciona el RAW de la instrucción ADD y, como consecuencia, se elimina el WAR porque, ahora, cuando la instrucción *mul.d f1, f2, f1* se encuentran en su etapa ID, la instrucción *add.d f3, f2, f1* finaliza su etapa A0.

Ejercicio 2.

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

La funcionalidad de las siguientes instrucciones es:

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

Ejercicio 3.

Escribir un programa que calcule la superficie de un triángulo rectángulo de base 5,85 cm y altura 13,47 cm. La superficie de un triángulo se calcula como: Superficie= $\frac{base*altura}{2}$.

```
            .data
BASE:       .double 5.85
ALTURA:     .double 13.47
MEDIO:      .double 0.5
RES:        .double 0.0

            .code
            l.d f1, BASE(r0)
            l.d f2, ALTURA(r0)
            l.d f4, MEDIO(r0)
            mul.d f3, f1, f2
            mul.d f5, f3, f4
            s.d f5, RES(r0)
            halt
```

Ejercicio 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: IMC= $\frac{peso}{altura^2}$*. De acuerdo al valor calculado con este índice, puede clasificarse el estado nutricional de una persona en: Infrapeso (IMC* $< 18{,}5$*), Normal (*$18{,}5 \leq IMC < 25$*), Sobrepeso (*$25 \leq IMC < 30$*) y Obeso (IMC* ≥ 30*). Escribir 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*	*5*

```
                .data
ESTATURA:       .double 1.65
PESO:           .double 83.0
INFRAPESO:      .double 18.5
NORMAL:         .double 25.0
SOBREPESO:      .double 30.0
IMC:            .double 0.0
ESTADO:         .word 0

                .code
                l.d f1, ESTATURA(r0)
                mul.d f6, f1, f1
                l.d f2, PESO(r0)
                l.d f3, INFRAPESO(r0)
                l.d f4, NORMAL(r0)
                l.d f5, SOBREPESO(r0)
                div.d f7, f2, f6
                c.lt.d f7, f3
                bc1t INFRA
                c.lt.d f7, f4
                bc1t NORM
                c.lt.d f7, f5
                bc1t SOBRE
                daddi r1, r0, 4
                j FIN
INFRA:          daddi r1, r0, 1
                j FIN
NORM:           daddi r1, r0, 2
                j FIN
SOBRE:          daddi r1, r0, 3
```

```
FIN:    s.d f7, IMC(r0)
        sd r1, ESTADO(r0)
        halt
```

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

Registro	Nombre	¿Para qué se lo utiliza?	¿Preservado?
r0	$zero	Siempre tiene valor 0 y no se puede cambiar	
r1	$at	*Assembler Temporary* - Reservado para ser usado por el ensamblador	
r2-r3	$v0-$v1	Valores de retorno de la subrutina llamada	
r4-r7	$a0-$a3	Argumentos pasados a la subrutina llamada	
r8-r15	$t0-$t7	Registros temporarios. No son conservados en el llamado a subrutinas	
r16-r23	$s0-$s7	Registros salvados durante el llamado a subrutinas	x
r24-r25	$t8-$t9	Registros temporarios. No son conservados en el llamado a subrutinas	
r26-r27	$k0-$k1	Para uso del kernel del sistema operativo	
r28	$gp	*Global Pointer* - Puntero a la zona de la memoria estática del programa	x

r29	$sp	*Stack Pointer* - Puntero al tope de la pila	x
r30	$fp	*Frame Pointer* - Puntero al marco actual de la pila	x
r31	$ra	*Return Address* - Dirección de retorno en un llamado a una subrutina	x

Ejercicio 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?*

El programa calcula $16^4 = 65.536$ y almacena el resultado en la variable RESULT. En la variable VALOR1, guarda la base de la potencia y, en el VALOR2, guarda el exponente. Luego, carga estos valores en los registros y salta a una subrutina que se encarga de calcular la potencia y guarda el resultado en el registro $v0.

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

La instrucción *jal* salta a la dirección de memoria de la subrutina A_LA_POTENCIA y copia en *$ra* la dirección de retorno. Y la instrucción *jr* salta a la dirección contenida en *$ra*.

(c) *¿Qué valor se almacena en el registro $ra? ¿Qué función cumplen los registros $a0 y $a1? ¿Y el registro $v0?*

El valor que se almacena en el registro *$ra* es la dirección de memoria de la instrucción siguiente al llamado de la subrutina A_LA_POTENCIA. La función que cumplen los

registros *$a0* y *$a1* son de argumentos/parámetros pasados a la subrutina llamada (en este caso, la base y el exponente). Y el registro *$v0* contiene el valor de retorno de la subrutina llamada (en este caso, el resultado de la potencia).

(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?*

Si la subrutina A_LA_POTENCIA necesitara invocar a otra subrutina para la realizar la multiplicación, esta subrutina volvería a la dirección de retorno incorrecta (la de la subrutina interna). Para que cada una de las subrutinas sepa a qué dirección de memoria deben retornar se debe guardar el *$ra* de la primera subrutina usando la pila (*push $ra*) y, una vez que se retorna de la segunda subrutina, se debe recuperar el *$ra* de la primera subrutina usando la pila (*pop $ra*).

Ejercicio 7.

Escribir 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.

```
                .data
TABLA:          .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
M:              .word 5
TAM:            .word 10
RES:            .word 0

                .code
                daddi $sp, $0, 0x400
                ld $a0, M($0)
                ld $a1, TAM($0)
                daddi $a2, $0, TABLA
                daddi $a0, $a0, 1
                jal EMPEZAR
                sd $v0, RES($0)
                halt

EMPEZAR:        daddi $sp, $sp, -16
                sd $ra, 0($sp)
                sd $s2, 8($sp)
                dadd $t0, $a1, $0
                dadd $s2, $a2, $0
                dadd $v0, $0, $0
LAZO:           ld $t1, 0($s2)
                slt $t2, $t1, $a0
                bnez $t2, MENOR
                daddi $v0, $v0, 1
MENOR:          daddi $t0, $t0, -1
                daddi $s2, $s2, 8
                bnez $t0, LAZO
                ld $ra, 0($sp)
                ld $s2, 8($sp)
                daddi $sp, $sp, 16
                jr $ra
```

Ejercicio 8.

Escribir 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.

```
            .data
CAD1:       .asciiz  "abcde"
CAD2:       .asciiz  "abcd"
RES:        .word 0

            .code
            daddi $sp, $0, 0x400
            daddi $a0, $0, CAD1
            daddi $a1, $0, CAD2
            jal COMP
            sd $v0, RES($0)
            halt

COMP:       daddi $sp, $sp, -24
            sd $ra, 0($sp)
            sd $s0, 8($sp)
            sd $s1, 16($sp)
            dadd $s0, $a0, $0
            dadd $s1, $a1, $0
            dadd $v0, $0, $0
LAZO:       lbu $t0, 0($s0)
            lbu $t1, 0($s1)
            beqz $t0, FIN1
            beqz $t1, FIN2
            bne $t0, $t1, FIN2
            daddi $v0, $v0, 1
            daddi $s0, $s0, 1
            daddi $s1, $s1, 1
            j LAZO
FIN1:       bnez $t1, FIN2
            daddi $v0, $v0, -1
FIN2:       ld $ra, 0($sp)
            ld $s0, 8($sp)
            ld $s1, 16($sp)
            daddi $sp, $sp, 24
            jr $ra
```

Ejercicio 9.

Escribir 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 o 0 en caso contrario.

```
                .data
VOCALES:        .asciiz "AEIOUaeiou"
CHAR:           .ascii "a"
RES:            .word 0

                .code
                daddi $sp, $0, 0x400
                lbu $a0, CHAR($0)
                daddi $a1, $0, VOCALES
                jal ES_VOCAL
                sd $v0, RES($0)
                halt

ES_VOCAL:       daddi $sp, $sp, -16
                sd $ra, 0($sp)
                sd $s1, 8($sp)
                dadd $v0, $0, $0
                dadd $s1, $a1, $0
LAZO:           lbu $t1, 0($s1)
                beqz $t1, FIN
                beq $a0, $t1, VOCAL
                daddi $s1, $s1, 1
                j LAZO
VOCAL:          daddi $v0, $v0, 1
FIN:            ld $ra, 0($sp)
                ld $s1, 8($sp)
                jr $ra
```

Ejercicio 10.

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

```
                .data
VOCALES:        .asciiz "AEIOUaeiou"
CADENA:         .ascii "AbCdE"
RES:            .word 0

                .code
                daddi $sp, $0, 0x400
                daddi $a0, $0, CADENA
                jal CONTAR_VOC
                sd $v1, RES($0)
                halt

CONTAR_VOC:     daddi $sp, $sp, -16
                sd $ra, 0($sp)
                sd $s0, 8($sp)
                daddi $a1, $0, VOCALES
                dadd $v1, $0, $0
                dadd $s0, $a0, $0
LAZO1:          lbu $a0, 0($s0)
                beqz $a0, FIN1
                jal ES_VOCAL
                dadd $v1, $v1, $v0
                daddi $s0, $s0, 1
                j LAZO1
FIN1:           ld $ra, 0($sp)
                ld $s0, 8($sp)
                jr $ra

ES_VOCAL:       daddi $sp, $sp, -8
                sd $s1, 0($sp)
                dadd $v0, $0, $0
                dadd $s1, $a1, $0
LAZO2:          lbu $t1, 0($s1)
                beqz $t1, FIN2
                beq $a0, $t1, VOCAL
                daddi $s1, $s1, 1
                j LAZO2
VOCAL:          daddi $v0, $v0, 1
FIN2:           ld $s1, 0($sp)
                daddi $sp, $sp, 8
                jr $ra
```

Ejercicio 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. Esta 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.

```
                .data
TABLA:          .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0
RES:            .word 0

                .code
                daddi $sp, $0, 0x400
                daddi $a0, $0, TABLA
                jal CANT_IMP
                sd $v1, RES($0)
                halt

CANT_IMP:       daddi $sp, $sp, -16
                sd $ra, 0($sp)
                sd $s0, 8($sp)
                dadd $v1, $0, $0
                dadd $s0, $a0, $0
LAZO:           ld $a0, 0($s0)
                beqz $a0, FIN1
                jal ES_IMPAR
                dadd $v1, $v1, $v0
                daddi $s0, $s0, 8
                j LAZO
FIN1:           ld $ra, 0($sp)
                ld $s0, 8($sp)
                daddi $sp, $sp, 16
                jr $ra

ES_IMPAR:       dadd $v0, $0, $0
                andi $t0, $a0, 1
                beqz $t0, FIN2
                daddi $v0, $v0, 1
FIN2:           jr $ra
```

Ejercicio 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
                ld $a0, VALOR($zero)
                jal FACTORIAL
                sd $v0, RESULT($zero)
                halt
FACTORIAL:      ...
                ...
                ...
```

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

```
                .data
VALOR:          .word 10
RESULT:         .word 0

                .code
                daddi $sp, $0, 0x400
                ld $a0, VALOR($0)
                jal FACTORIAL
                sd $v0, RESULT($0)
                halt

FACTORIAL:      daddi $sp, $sp, -16
                sd $ra, 0($sp)
                sd $s0, 8($sp)
                beqz $a0, FIN1
                dadd $s0, $a0, $0
                daddi $a0, $a0, -1
                jal FACTORIAL
                dmul $v0, $v0, $s0
                j FIN2
FIN1:           daddi $v0, $0, 1
FIN2:           ld $ra, 0($sp)
                ld $s0, 8($sp)
                daddi $sp, $sp, 16
                jr $ra
```

(b) *¿Es posible escribir la subrutina factorial sin utilizar una pila? Justificar.*

No, no es posible escribir la subrutina FACTORIAL sin utilizar una pila, ya que es lo que permite guardar los distintos *$ra* y *$s0* de la recursión.

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.

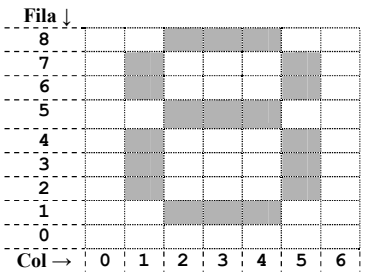
5) Escriba un programa que calcule el resultado de elevar un valor en punto flotante a la potencia indicada por un exponente que es un número entero positivo. Para ello, en el programa principal se solicitará el ingreso de la base (un número en punto flotante) y del exponente (un número entero sin signo) 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.

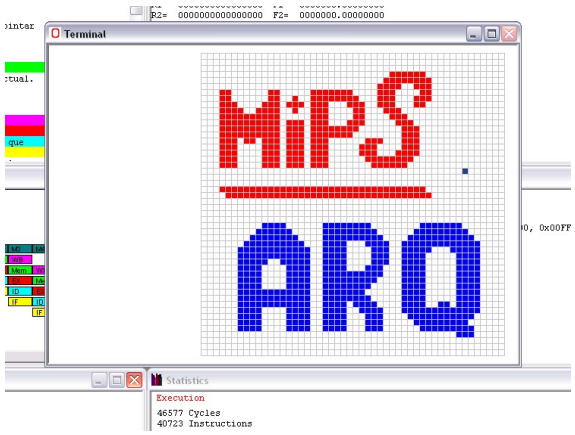
El carácter '8' es representado como:



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.

Juan Menduiña

Trabajo Práctico N° 6:
Procesador RISC (Utilizando la ES/S).

Ejercicio 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), modificar 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

            .code
            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
      alfanumérica
            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
```

```
            .data
CONTROL:    .word 0x10000
DATA:       .word 0x10008
MSJ:        .asciiz "INTRODUCIR Hola, Mundo!:\n"
STR:        .asciiz ""

            .code
            ld $s0, CONTROL($0)
            ld $s1, DATA($0)
            daddi $t1, $0, MSJ
            sd $t1, 0($s1)
            daddi $t0, $0, 4
            sd $t0, 0($s0)
            daddi $t0, $0, 9
            dadd $t2, $0, $0
            daddi $t3, $0, 12
LAZO:       sd $t0, 0($s0)
```

```
lbu $t1, 0($s1)
sb $t1, STR($t2)
daddi $t3, $t3, -1
daddi $t2, $t2, 1
bnez $t3, LAZO
daddi $t1, $0, STR
sd $t1, 0($s1)
daddi $t0, $0, 4
sd $t0, 0($s0)
halt
```

Ejercicio 2.

Escribir 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"). Establecer el pasaje de parámetros entre subrutinas respetando las convenciones para el uso de los registros y minimizar las detenciones del cauce (ejercicio similar al Ejercicio 6 de la Práctica 2).

```
                .data
CONTROL:        .word 0x10000
DATA:           .word 0x10008
MSJ:            .asciiz "INTRODUCIR UN NUMERO ENTERO DE UN
DIGITO:\n"
ERROR:          .asciiz "ERROR"
MENOSNUEVE:     .asciiz "MNUEVE"
MENOSOCHO:      .asciiz "MOCHO"
MENOSSIETE:     .asciiz "MSIETE"
MENOSSEIS:      .asciiz "MSEIS"
MENOSCINCO:     .asciiz "MCINCO"
MENOSCUATRO:    .asciiz "MCUATRO"
MENOSTRES:      .asciiz "MTRES"
MENOSDOS:       .asciiz "MDOS"
MENOSUNO:       .asciiz "MUNO"
CERO:           .asciiz "CERO"
UNO:            .asciiz "UNO"
DOS:            .asciiz "DOS"
TRES:           .asciiz "TRES"
CUATRO:         .asciiz "CUATRO"
CINCO:          .asciiz "CINCO"
SEIS:           .asciiz "SEIS"
SIETE:          .asciiz "SIETE"
OCHO:           .asciiz "OCHO"
NUEVE:          .asciiz "NUEVE"

                .code
                ld $s0, CONTROL($0)
                ld $s1, DATA($0)
                daddi $t1, $0, MSJ
                sd $t1, 0($s1)
                daddi $t0, $0, 4
                sd $t0, 0($s0)
                jal INGRESO
                dadd $a0, $v0, $0
                jal MUESTRA
                halt
```

```
INGRESO:    daddi $v0, $0, -10
            daddi $t0, $0, 8
            sd $t0, 0($s0)
            ld $t1, 0($s1)
            slti $t2, $t1, 10
            bez $t2, FIN
            dadd $v0, $t1, $0
FIN:        jr $ra

MUESTRA:    daddi $t3, $0, 8
            dmul $t4, $a0, $t3
            daddi $t5, $t4, CERO
            sd $t5, 0($s1)
            daddi $t6, $0, 4
            sd $t6, 0($s0)
            jr $ra
```

Ejercicio 3.

Escribir 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 la Práctica 2).

```
                .data
CONTROL:        .word 0x10000
DATA:           .word 0x10008
MSJ1:           .asciiz "INTRODUCIR DOS NUMEROS ENTEROS:\n"
MSJ2:           .asciiz "SU SUMA ES IGUAL A:\n"

                .code
                ld $s0, CONTROL($0)
                ld $s1, DATA($0)
                daddi $t1, $0, MSJ1
                sd $t1, 0($s1)
                daddi $t0, $0, 4
                sd $t0, 0($s0)
                daddi $t3, $0, -99
                jal INGRESO
                beq $v0, $t3, FIN1
                dadd $a0, $v0, $0
                jal INGRESO
                beq $v0, $t3, FIN1
                dadd $a1, $v0, $0
                jal RESULTADO
FIN1:           halt

INGRESO:        daddi $v0, $0, -99
                daddi $t0, $0, 8
                sd $t0, 0($s0)
                ld $t1, 0($s1)
                slti $t2, $t1, 10
                beqz $t2, FIN2
                dadd $v0, $t1, $0
FIN2:           jr $ra

RESULTADO:      daddi $t1, $0, MSJ2
                sd $t1, 0($s1)
                daddi $t0, $0, 4
                sd $t0, 0($s0)
                dadd $t0, $a0, $a1
                sd $t0, 0($s1)
                daddi $t1, $0, 2
                sd $t1, 0($s0)
                jr $ra
```

Ejercicio 4.

Escribir un programa que solicite el ingreso por teclado de una clave (sucesión de cuatro caracteres) utilizando la subrutina CHAR de ingreso de un caracter. Luego, se debe comparar la secuencia ingresada con una cadena almacenada en la variable CLAVE. Si las dos cadenas son iguales entre sí, 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.

```
                    .data
CONTROL:            .word 0x10000
DATA:               .word 0x10008
CLAVE:              .asciiz "JUAN"
MSJ_BIENV:          .asciiz "BIENVENIDO"
MSJ_ERROR:          .asciiz "ERROR. INTRODUCIR, NUEVAMENTE, LA
CLAVE:\n"
MSJ:                .asciiz "INTRODUCIR UNA CLAVE DE CUATRO
CARACTERES:\n"

                    .code
                    daddi $sp, $0, 0x400
                    ld $s0, CONTROL($0)
                    ld $s1, DATA($0)
                    daddi $t1, $0, MSJ
                    sd $t1, 0($s1)
                    daddi $t0, $0, 4
                    sd $t0, 0($s0)
                    daddi $a2, $0, CLAVE
                    jal CHAR
                    halt

CHAR:               daddi $sp, $sp, -16
                    sd $ra, 0($sp)
                    sd $s2, 8($sp)
                    daddi $t0, $0, 9
                    dadd $s2, $a2, $0
                    daddi $t3, $0, 4
                    dadd $t4, $0, $0
LAZO:               sd $t0, 0($s0)
                    lbu $t1, 0($s1)
                    lbu $t5, 0($s2)
                    beq $t1, $t5, NO_SON_IGUALES
                    daddi $t4, $t4, 1
NO_SON_IGUALES:     daddi $t3, $t3, -1
                    daddi $s2, $s2, 1
                    beqz $t3, FIN1
                    j LAZO
FIN1:               beq $t4, $t3, FIN2
```

```
                daddi $t6, $0, MSJ_ERROR
                sd $t6, 0($s1)
                jal RESPUESTA
                daddi $t3, $0, 4
                dadd $t4, $0, $0
                dadd $s2, $a2, $0
                j LAZO
FIN2:           daddi $t6, $0, MSJ_BIENV
                sd $t6, 0($s1)
                jal RESPUESTA
                ld $ra, 0($sp)
                ld $s2, 8($sp)
                daddi $sp, $sp, 16
                jr $ra

RESPUESTA:      daddi $t7, $0, 4
                sd $t7, 0($s0)
                jr $ra
```

Ejercicio 5.

Escribir 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). Tener en cuenta que cualquier base elevada a la 0 da como resultado 1. Mostrar el resultado numérico de la operación en la salida estándar del simulador (ventana Terminal).

```
                .data
CONTROL:        .word 0x10000
DATA:           .word 0x10008
BASE:           .word 0.0
EXP:            .word 0
RES:            .double 0
UNO:            .double 1.0
MSJ1:           .asciiz "BASE en Flotante: \n"
MSJ2:           .asciiz "EXPONENTE en BSS: \n"
MSJ3:           .asciiz "RESULTADO:\n"

                .code
                ld $s0, CONTROL($0)
                ld $s1, DATA($0)
                daddi $t1, $0, MSJ1
                sd $t1, 0($s1)
                daddi $t0, $0, 4
                sd $t0, 0($s0)
                daddi $t0, $0, 8
                sd $t0, 0($s0)
                l.d f1, 0($s1)
                s.d f1, BASE($0)
                daddi $t1, $0, MSJ2
                sd $t1, 0($s1)
                daddi $t0, $0, 4
                sd $t0, 0($s0)
                daddi $t0, $0, 8
                sd $t0, 0($s0)
                ld $a0, 0($s1)
                sd $a0, EXP($0)
                jal A_LA_POTENCIA
                s.d f2, RES($0)
                daddi $t1, $0, MSJ3
                sd $t1, 0($s1)
                daddi $t0, $0, 4
                sd $t0, 0($s0)
                s.d f2, 0($s1)
                daddi $t0, $0, 3
```

```
                        sd $t0, 0($s0)
                        halt

A_LA_POTENCIA:          l.d f2, UNO($0)
                        dadd $t0, $a0, $0
LAZO:                   beqz $t0, FIN
                        mul.d f2, f2, f1
                        daddi $t0, $t0, -1
                        j LAZO
FIN:                    jr $ra
```

Ejercicio 6.

El siguiente programa produce una salida estableciendo el color de un punto de la pantalla gráfica (en la ventana Terminal del simulador WinMIPS64). Modificar 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

            .code
            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
```

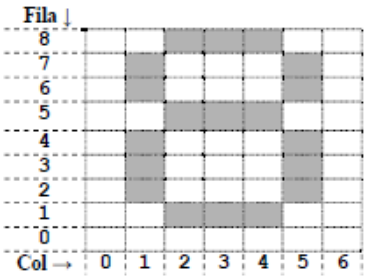
```
            .data
CONTROL:    .word 0x10000
DATA:       .word 0x10008
COORX:      .byte 24
COORY:      .byte 24
COLOR:      .byte 255, 0, 255, 0

            .code
            ld $s0, CONTROL($0)
            ld $s1, DATA($0)
            daddi $t0, $0, 7
            sd $t0, 0($s0)
            lbu $t0, COORX($0)
            sb $t0, 5($s1)
            lbu $t1, COORY($0)
            sb $t1, 4($s1)
            lwu $t2, COLOR($0)
```

```
                sw $t2, 0($s1)
                daddi $t0, $0, 5
                sd $t0, 0($s0)
                halt

                .data
CONTROL:        .word 0x10000
DATA:           .word 0x10008
COORX:          .byte 0
COORY:          .byte 0
COLOR:          .byte 0, 0, 0, 0
MSJ1:           .asciiz "INTRODUCIR COORDENADA X:\n"
MSJ2:           .asciiz "INTRODUCIR COORDENADA Y:\n"
MSJ3:           .asciiz "INTRODUCIR RGBA:\n"

                .code
                ld $s0, CONTROL($0)
                ld $s1, DATA($0)
                daddi $t1, $0, MSJ1
                sd $t1, 0($s1)
                daddi $t0, $0, 4
                sd $t0, 0($s0)
                daddi $t0, $0, 8
                sb $t0, 0($s0)
                lbu $t2, 0($s1)
                sb $t2, COORX($0)
                daddi $t1, $0, MSJ2
                sd $t1, 0($s1)
                daddi $t0, $0, 4
                sd $t0, 0($s0)
                daddi $t0, $0, 8
                sb $t0, 0($s0)
                lbu $t2, 0($s1)
                sb $t2, COORY($0)
                daddi $t1, $0, MSJ3
                sd $t1, 0($s1)
                daddi $t0, $0, 4
                sd $t0, 0($s0)
                daddi $a0, $0, 0
                daddi $a1, $0, 4
LAZO:           daddi $t0, $0, 8
                sb $t0, 0($s0)
                lbu $t2, 0($s1)
                sb $t2, COLOR($a0)
                daddi $a1, $a1, -1
                daddi $a0, $a0, 1
                bnez a1, LAZO
                daddi $t0, $0, 7
                sd $t0, 0($s0)
                lbu $t0, COORX($0)
```

```
sb $t0, 5($s1)
lbu $t1, COORY($0)
sb $t1, 4($s1)
lwu $t2, COLOR($0)
sw $t2, 0($s1)
daddi $t0, $0, 5
sd $t0, 0($s0)
halt
```

Ejercicio 7.

Se desea realizar la demostración de la transformación de un caracter codificado en ASCII a su visualización en una matriz de puntos con 7 columnas y 9 filas. Escribir 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:



Ejercicio 8.

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

Ejercicio 9.

Escribir 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.*