

PRÁCTICA 1 – Algunas resoluciones

Subrutinas y pasaje de parámetros

Ejercicio 1

	Instrucción	Valor del registro SP	AX	BX
1	mov ax,5	8000	5	?
2	mov bx,3	8000	5	3
3	push ax	7FFEh	5	3
4	push ax	7FFCh	5	3
5	push bx	7FFAh	5	3
6	pop bx	7FFCh	5	3
7	pop bx	7FFEh	5	5
8	pop ax	8000	5	5

Ejercicio 2

#	Instrucción	Valor del registro SP
1	org 3000h	
2	rutina: mov bx,3	7FFCh
3	ret	7FFEh
4	org 2000h	
5	push ax	7FFEh
6	call rutina	7FFCh
7	pop bx	8000h
8	hlt	8000h
9	end	

Ejercicio 3

La siguiente tabla indica el contenido de las direcciones 7FFFh y 7FFEh de la pila, que son las únicas que se utilizan en todo el programa y además el valor del registro SP, luego de la ejecución de cada instrucción. Cada columna indica una instrucción ejecutada. Algunas instrucciones se repiten, ya sea porque están dos veces en el programa (como `call rut`) y otras porque se *ejecutan* dos veces, como las instrucciones que están dentro de `rut`.

	Instrucción ejecutada							
Pila	call rut	mov bx,3	ret	add cx,5	call rut	mov bx,3	ret	hlt
7FFEh	02	02	?	?	06	06	?	?
7FFFh	20	20	?	?	20	20	?	?
SP	7FFEh	7FFEh	8000h	8000h	7FFEh	7FFEh	8000h	8000h

Ejercicio 4

	Código	Registro	Pila	Valor	Referencia
a	mov ax, 5 call subrutina	SI		SI	
b	mov dx, offset A call subrutina	SI			SI
c	mov bx, 5 push bx call subrutina pop bx		SI	SI	
d	mov cx, offset A push cx call subrutina pop cx		SI		SI
e	mov dl, 5 call subrutina	SI		SI	
f	call subrutina mov A, dx	SI		SI	

Ejercicio 5a

```

; Memoria de Datos
org 1000h
A DW 5h
B DW 6h
C DW 2h
D DW ?
; Memoria de programa
org 2000h
mov ax, A
add ax, B
sub ax, C
mov D, ax
hlt
end

```

Ejercicio 5b

```

; Memoria de Datos
org 1000h
A DW 5h
B DW 6h
C DW 2h
D DW ?
org 3000h
calculo: mov ax, A
        add ax, B
        sub ax, C
        mov D, ax
        ret
; Memoria de programa
org 2000h
call calculo
hlt
end

```

Ejercicio 5c

```

; Memoria de Datos

```

```

    org 1000h
A   DW  5h
B   DW  6h
C   DW  2h
D   DW  ?
    org 3000h
; Recibe en ax, bx y cx tres valores A, B y C
; Devuelve en dx el cálculo A+B-C
calculo: mov dx,ax
        add dx,bx
        sub dx,cx
        ret
    org 2000h          ; programa principal
    mov ax, A
    mov bx, B
    mov cx, C
    call calculo
    mov D, dx
    hlt
end

```

Ejercicio 6

<p>A) ; Memoria de Datos ORG 1000H NUM1 DB 5H NUM2 DB 3H RES DW ? ; Memoria de Instrucciones ORG 2000H MOV DX, 0 MOV AL, NUM1 CMP AL, 0 JZ FIN MOV AH, 0 MOV CL, NUM2 LOOP: CMP CL, 0 JZ FIN ADD DX, AX DEC CL JMP LOOP FIN: MOV RES, DX HLT END</p>	<p>B) ; Memoria de Datos ORG 1000H NUM1 DB 5H NUM2 DB 3H RES DW ? ; Memoria de Instrucciones ORG 3000H ; Subrutina MUL MUL: MOV DX, 0 CMP CL, 0 JZ FIN MOV AH, 0 LAZO: ADD DX, AX DEC CL JNZ LAZO FIN: RET ORG 2000H ; Programa principal MOV AL, NUM1 MOV CL, NUM2 CALL MUL MOV RES, DX HLT END</p>
<p>C) ; Memoria de datos ORG 3000H ; Subrutina MUL MUL: MOV DX, 0 ; obtener operandos ; desde la memoria MOV BX, AX MOV AX, [BX] MOV BX, CX MOV CX, [BX] ; comprobar que CX > 0 CMP CX, 0 JZ FIN LAZO: ADD DX, AX DEC CX JNZ LAZO FIN: RET</p> <p>ORG 1000H NUM1 DW 5H NUM2 DW 3H RES DW ? ORG 2000H ; Programa MOV AX, OFFSET NUM1 MOV CX, OFFSET NUM2 CALL MUL MOV RES, DX HLT END</p>	

Ejercicio 8a

```

                ORG 1000H
CAD             DB  "EXCELENTE"
                DB  00H
                ORG 3000H
LONGITUD:      MOV DX, 0           ;contador
LOOP:          MOV AH, [BX]
                CMP AH, 00H
                JZ  FIN
                INC DX
                INC BX
                JMP LOOP
FIN:           RET

                ORG 2000h
                MOV BX, offset CAD
                CALL LONGITUD
                HLT
                END

```

Ejercicio 8c. Esta resolución sólo aplica para el caso en que las vocales sean mayúsculas.

```

                ORG 1000H
resultado DB ?      ; cambiar este valor y ver que queda en la variable resultado
CHAR      DB "E"

; Recibe el caracter a verificar por AH
; Devuelve el resultado en AL
                ORG 3000H
ES_VOCAL:     MOV AL, 0FFH
                CMP AH, 41H          ; A
                JZ  FIN
                CMP AH, 45H          ; E
                JZ  FIN
                CMP AH, 49H          ; I
                JZ  FIN
                CMP AH, 4FH          ; O
                JZ  FIN
                CMP AH, 55H          ; U
                JZ  FIN
                MOV AL, 00H
FIN:          RET

                ORG 2000h
                MOV AH, CHAR
                CALL ES_VOCAL
                MOV resultado, AL
                HLT
                END

```

Ejercicio 8d

```

; Recibe el caracter a verificar por AH
; Devuelve el resultado en AL
                ORG 4000H
ES_VOCAL:     MOV AL, 0FFH
                CMP AH, 41H          ;A
                JZ  FIN
                CMP AH, 45H          ;E
                JZ  FIN
                CMP AH, 49H          ;I
                JZ  FIN
                CMP AH, 4FH          ;O
                JZ  FIN
                CMP AH, 55H          ;U

```

```

        JZ FIN
        MOV AL, 00H
FIN:     RET

        ORG 1000H
CAD      DB  "EXCELENTE"
CERO     DB  0
resultado DW  ?

; Recibe en BX la dirección de la cadena
; retorna en CX la cantidad de vocales
        ORG 3000H
VOCALES: MOV CX, 0          ; cantidad de vocales
LOOP:    MOV AH, BYTE PTR [BX] ; pongo en AX el caracter correspondiente a [BX]
        CMP AH, 0          ; si llegue al valor 0 (fin de cadena)
        JZ  fin_vocales    ; retorno
        CALL ES_VOCAL
        CMP AL, 0FFH       ; si no son iguales, no es vocal
        JNZ NOES
        INC CX              ; incremento vocales
NOES:    INC BX              ; me muevo por la cadena
        JMP LOOP            ; verifico el próximo char
fin_vocales: RET

        ORG 2000h
        MOV BX, offset CAD
        CALL VOCALES
        MOV resultado, CX
        HLT
        END

```

Ejercicio 9a

```

; Recibe el caracter a rotar en AH
; Devuelve el resultado también en AH
        ORG 3000H
ROTARIZQ: ADD AH, AH
        ADC AH, 0
        RET
        ORG 1000H
b        DB  27H          ; (00100111) en binario
        ORG 2000H
        MOV AH, b          ; AH = 00100111
; Realizamos una rotación
        CALL ROTARIZQ      ; AH = 01001110
; Realizamos una segunda rotación
        CALL ROTARIZQ      ; AH = 10011100
        HLT
        END

```

Ejercicio 9b (asumimos que está disponible la subrutina ROTARIZQ definida anteriormente)

```

; Recibe el caracter a rotar en AH
; Recibe la cantidad de posiciones en BH
; Devuelve el resultado también en AH
        ORG 4000H
ROTARIZQ_N: CMP BH, 0      ; mientras BH>0
        JZ  FIN            ; si BH=0, entonces finalizar la subrut.
        CALL ROTARIZQ
        DEC BH
        JMP ROTARIZQ_N     ; aprovecho la etiqueta de la subrutina
                          ; para hacer el salto
FIN:     RET

        ORG 1000H
b        DB  27H          ; (00100111) en binario

```

```

        ORG 2000H
        MOV AH, b
; Realizamos una rotación de 2 posiciones a la izquierda
        MOV BH, 2
        CALL ROTARIZQ_N    ; AH = 10011100 (C9H)
        HLT
        END

```

Ejercicio 9c (asumimos que está disponible la subrutina ROTARIZQ_N definida anteriormente)

```

; Utiliza los mismos registros que ROTARIZQ_N
; Recibe en BH la cantidad de posiciones
        ORG 5000H
ROTARDER_N: MOV CH, 8
            SUB CH, BH          ; cantidad de bytes que debo rotar hacia la izq.
            MOV BH, CH          ; vuelvo a copiar en BH
            ; ROTARIZQ usará el valor almacenado en BH para rotar.
            CALL ROTARIZQ_N
            RET

        ORG 1000H
        b    DB 27h            ; (00100111) en binario

        ORG 2000H
        MOV AH, b
; Realizamos una rotación de 6 posiciones a la derecha
        MOV BH, 2
        CALL ROTARDER_N    ; AH = 10011100 (C9H)
        HLT
        END

```

Ejercicio 10

```

; Recibe las direcciones de dos celdas de memoria a intercambiar M1 y M2
; a través de la pila
        ORG 3000H
SWAP:   PUSH BX                ; preservó los 3 registros
        PUSH AX
        PUSH DX
; OBTENER EL VALOR DE M2 en CX
        MOV BX, SP
        ADD BX, 8              ; apunto al segundo parámetro
; 8=6+2: 6=3*2 son de los push; los otros 2 por la dir de retorno
        MOV BX, [BX]           ; BX tiene la DIR de M2
        MOV CX, [BX]           ; CX tiene el valor de M2
; OBTENER EL VALOR DE M1 en DX
        MOV BX, SP
        ADD BX, 10             ; apunto al primer parámetro
; 10=6+2+2: 6=3*2 son de los push; 2 por la dir de retorno, y 2 por M2
        MOV BX, [BX]           ; BX tiene la DIR de M1
        MOV DX, [BX]           ; DX tiene el valor de M1
; PONER EL VALOR DE M1 (DX) en M2
        MOV BX, SP
        ADD BX, 8              ; apunto al segundo parámetro
        MOV BX, [BX]           ; BX tiene la DIR de M2
        MOV [BX], DX           ; Asigno el valor de M1 en la dir de M2
; PONER EL VALOR DE M2 (CX) en M1
        MOV BX, SP
        ADD BX, 10             ; apunto al primer parámetro
        MOV BX, [BX]           ; BX tiene la DIR de M2
        MOV [BX], CX           ; Asigno el valor de M2 en la dir de M1
; restauro los 3 registros
        POP DX
        POP AX
        POP BX

```

```

        RET

        ORG 1000H
val1    DW  1234H
val2    DW  5678H

        ORG 2000H
        MOV AX, offset val1
        PUSH AX
        MOV AX, offset val2
        PUSH AX
        CALL SWAP
; verificar que se hayan intercambiado los valores entre val1 y val2
        HLT
        END

```

Ejercicio 11b

```

        ORG 1000H
num1    DB  6H
num2    DB  4H

; subrutina resto
; Recibe dos números en los registros CH y CL
; Retorna el resto de la división entera (sin coma) de CH/CL
; Por ejemplo el resto de 6/4 es 2
        ORG 3000H
resto:  MOV AL, 0           ; inicializo el resto en 0
        MOV DH, 0           ; inicializo el cociente de la división
        CMP CH, 0           ; CH tiene NUM2
        JZ  FIN
        CMP CL, 0           ; CL tiene NUM1
        JZ  FIN
DIV:    SUB CL, CH
        JS RES              ; si resultado negativo, voy a calcular el resto
        INC DH              ; sumo al cociente
        JMP DIV
RES:    ADD CL, CH           ; sumo de vuelta CH para determinar el resto
        MOV AL, CL          ; devuelvo el resto en AX
FIN:    RET

        ORG 2000H
        MOV CL, num1
        MOV CH, num2
        CALL resto
        HLT
        END

```

PRACTICA 2 (algunos ejercicios resueltos)

Interrupciones

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 ; "A"
MIN DB 61H ; "a"
        ; La letra "Z" (mayuscula) tiene el codigo 5A

ORG 2000H
MOV AL, 2 ; Se imprime en pantalla de a 2 caracteres
MOV BX, OFFSET MAY ; a partir de la direccion de MAY
PROX: INT 7
      INC MIN ; Paso al siguiente caracter
      INC MAY ; Paso al siguiente caracter
      CMP MAY, 5BH ; comparo con el caracter siguiente al "Z", que es el ultimo valido
      JNZ PROX ; Si aun no procesamos "Z", continua con el siguiente caracter
      INT 0
END
```


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.

```

CERO    ORG 1000H
        DB      "CERO  "      ; Todos los nombres tienen 6 caracteres para
        DB      "UNO  "      ; facilitar posicionarnos al imprimir el nombre del numero
        DB      "DOS   "
        DB      "TRES  "
        DB      "CUATRO"
        DB      "CINCO "
        DB      "SEIS  "
        DB      "SIETE "
        DB      "OCHO  "
        DB      "NUEVE "
MSJ      DB      "INGRESE UN NUMERO:"
FIN      DB      ?

        ORG 1500H
NUM      DB      ?

        ORG 2000H
MOV CL, 0      ; Contador de veces que ingresa el valor 0 de forma consecutiva
OTRO:    MOV BX, OFFSET MSJ
        MOV AL, OFFSET FIN-OFFSET MSJ
        INT 7      ; Imprimo mensaje en pantalla pidiendo el ingreso de un numero
        MOV BX, OFFSET NUM
        INT 6      ; Leo un caracter y queda guardado en NUM
        CMP NUM, 30H
        JNZ NO_CERO
        INC CL      ; Si vino un valor 0, incremento el contador
        JMP SEGUIR
NO_CERO: MOV CL, 0      ; Como no vino un valor 0, reinicializo CL
SEGUIR:  MOV BX, OFFSET CERO ; La direccion BASE sera la del primer mensaje ("CERO")
        ; Luego se posicionara al inicio del mensaje adecuado
        ; Se va a imprimir 6 caracteres, todos tienen el mismo largo
        MOV AL, 6
LOOP:    CMP NUM, 30H
        JZ  IMPRIME      ; Si es el valor adecuado, imprimo en pantalla el nombre del numero
        ADD BX, 6      ; Si no es el valor adecuado, me posiciono en el siguiente nombre
        DEC NUM      ; Al llegar NUM a 0 estara posicionado en el nombre que corresponde
        JMP LOOP
IMPRIME: INT 7
        CMP CL, 2
        JNZ OTRO      ; Si no se ingreso dos veces seguidas el numero 0, sigue procesando
        INT 0      ; Se ingreso dos veces seguidas 0, por lo que el programa termina
END

```

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
MSJ DB "INGRESE UN NUMERO:"
FIN DB ?

ORG 1500H
NUM1 DB ?
NUM2 DB ?
RES_D DB "0" ; Decena del resultado.
RES_U DB ? ; Unidad del resultado.
; Por ej. si se suma "6" + "7", la decena del resultado sera "1" y la unidad "3"

ORG 2000H
MOV BX, OFFSET MSJ
MOV AL, OFFSET FIN-OFFSET MSJ
INT 7 ; Imprimo mensaje en pantalla pidiendo el ingreso de un numero
MOV BX, OFFSET NUM1
INT 6 ; Leo un caracter y queda guardado en NUM1
MOV BX, OFFSET MSJ
INT 7 ; Imprimo mensaje en pantalla pidiendo el ingreso de un numero
MOV BX, OFFSET NUM2
INT 6 ; Leo un caracter y queda guardado en NUM2
MOV AL, NUM2 ; Copio el segundo caracter leído en AL
SUB AL, 30H ; Le resto 30H, para quedarme con el valor del numero
ADD AL, NUM1 ; Le sumo el primer caracter leído
CMP AL, 3AH ; Si quedo un valor entre 30H y 39H, la suma no supero 9
; Entonces la unidad esta lista
; Y la decena tambien, ya que comienza con valor "0"

JS NUM_OK
SUB AL, 10 ; Si quedo un valor mayor a 39H
; entonces se le resta 10 para obtener la unidad
; Se suma 1 a la decena (pasa de ser el caracter "0" a "1")
NUM_OK: MOV RES_U, AL ; Copio el valor de la unidad a RES_U
MOV BX, OFFSET RES_D ; A partir de la dir. de RES_D, se imprime 2 caracteres
MOV AL, 2
INT 7
INT 0

END

```

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.

```

TIMER EQU 10H
PIC EQU 20H
EOI EQU 20H
N_CLK EQU 10
N_F10 EQU 20

IP_CLK DW ORG 40
RUT_CLK

IP_F10 DW ORG 80
RUT_F10

ORG 1000H
SEG DB 30H ; Decena
DB 30H ; Unidad
FIN DB ?

RUT_CLK: ORG 3000H
PUSH AX ; Se guarda el valor de AX, porque se va a usar el registro
INC SEG+1
CMP SEG+1, 3AH
JNZ RESET
MOV SEG+1, 30H
INC SEG
CMP SEG, 33H
JNZ RESET
MOV DL, 1 ; Pongo en TRUE el flag de finalizacion
MOV AL, 0FFH ; Deshabilito interrupciones en IMR
OUT PIC+1, AL
RESET: MOV AL, 2 ; El contador tiene 2 caracteres
INT 7 ; Se imprime el valor actual
MOV AL, 0 ; Se vuelve a cero el contador del TIMER
OUT TIMER, AL
MOV AL, EOI ; Se finaliza la atencion de la interrupcion
OUT PIC, AL
POP AX ; Se recupera el valor que contenia AX al entrar en la rutina
IRET

RUT_F10: ORG 3500H
PUSH AX ; Se guarda el valor de AX, porque se va a usar el registro
IN AL, PIC+1 ; Recupero el valor actual del IMR
XOR AL, 00000010B ; Y cambio la linea correspondiente al TIMER
OUT PIC+1, AL
MOV AL, EOI ; Se finaliza la atencion de la interrupcion
OUT PIC, AL
POP AX ; Se recupera el valor que contenia AX al entrar en la rutina
IRET

ORG 2000H
CLI
MOV AL, 0FEH
OUT PIC+1, AL ; PIC: registro IMR
MOV AL, N_F10
OUT PIC+4, AL ; PIC: registro INT0, F10
MOV AL, N_CLK
OUT PIC+5, AL ; PIC: registro INT1, TIMER
MOV AL, 1
OUT TIMER+1, AL ; TIMER: registro COMP
MOV AL, 0
OUT TIMER, AL ; TIMER: registro CONT
MOV BX, OFFSET SEG ; Direccion del contador
MOV DL, 0
STI
LAZO: CMP DL, 0
JZ LAZO
INT 0

END

```

PRACTICA 3- SOLUCIONES

Entrada/Salida

Ejercicio 1a

```
ORG 1000H    ; Memoria de datos
patron db 0C3h    ;1100 0011b
```

```
CB EQU 33h
PB EQU 31h
```

```
ORG 2000H    ;Prog principal
mov al, 0
out CB, al
mov al, patron
out PB, al
HLT
END
```

Ejercicio 1b

```
ORG 1000H    ; Memoria de datos
prendida db "Llave prendida"
apagada db "Llave apagada"
fin_apagada db ?
```

```
CA EQU 32h
PA EQU 30h
```

```
ORG 2000H    ; Prog principal
mov al, 0ffh
out CA, al

in al, PA
; poner en 0 todos los bits menos el más sig
and al, 80h ; 1000 0000
; si es 0
cmp al, 0
jz esta_apagada
; esta prendida
mov bx, offset prendida
mov al, OFFSET apagada - OFFSET prendida
jmp fin
esta_apagada: mov bx, offset apagada
mov al, OFFSET fin_apagada - OFFSET apagada

fin: int 7 ; imprimir
HLT
END
```

Ejercicio 1c

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

```
ORG 2000H
MOV AL, 0FFH    ; PA entradas (Micro-conmutadores)
OUT CA, AL
MOV AL, 0       ; PB salidas (Luces)
```

```

        OUT CB, AL
POLL:   IN  AL, PA
        OUT PB, AL
        JMP POLL
END

```

Ejercicio 1d

PIC	EQU 20H		
TIMER	EQU 10H		
PIO	EQU 30H		
N_CLK	EQU 10		
	ORG 40		
IP_CLK	DW RUT_CLK		
	ORG 1000H		
PATRON	DB 0		
FINAL	DB 0		
	ORG 2000H		ORG 3000H
	CLI	RUT_CLK:	INC PATRON
	MOV AL, 0FDH		CMP PATRON, 0FFH
	OUT PIC+1, AL		JNZ LUCES
			MOV FINAL, 1
	MOV AL, N_CLK		MOV AL, 0FFh
			OUT PIC+1, AL
			JMP FIN
	OUT PIC+5, AL	LUCES:	MOV AL, PATRON
	MOV AL, 1		OUT PIO+1, AL
	OUT TIMER+1, AL		MOV AL, 0
	MOV AL, 0		OUT TIMER, AL
	OUT PIO+3, AL	FIN:	MOV AL, 20H
	OUT PIO+1, AL		OUT PIC, AL
	OUT TIMER, AL		IRET
	STI		END
LAZO:	CMP FINAL, 1		
	JNZ LAZO		
	HLT		

Ejercicio 2a

```

        ORG 1000H; Memoria de datos
char    db "A"

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

        ORG 2000H    ; Prog principal
        mov al, 01h ; strobe salida (0), busy entrada (1)
        out CA, al
        mov al, 0    ; puerto de datos todo salida
        out CB, al

; inicializo strobe en 0
        in  al, PA
        and al, 11111101b
        out PA, al
; espero que busy=0
poll:   in  al, PB
        and al, 01h ; 1000 0000
        jnz poll
; se que busy es 0, mandar caracer

```

```

    mov al, char
    out PB, al
; mandar flanco ascendente de strobe
    in  al, PA
    or  al, 00000010b
    out PA, al

    nop    ; esperamos un poco que imprima
    nop    ; esperamos un poco que imprima
    nop    ; esperamos un poco que imprima
    nop    ; esperamos un poco que imprima
    nop    ; esperamos un poco que imprima
    nop    ; esperamos un poco que imprima

    HLT
    END

```

Ejercicio 2b

```

PIO  EQU 30H

    ORG 1000H

MSJ  DB "ORGANIZACIÓN Y      "
     DB "ARQUITECTURA DE    "
     DB "COMPUTADORAS"
FIN  DB ?

    ORG 2000H
; INICIALIZACION PIO PARA IMPRESORA
; CA
    MOV AL, 0FDH
    OUT PIO+2, AL
; CB
    MOV AL, 0
    OUT PIO+3, AL
; Strobe
    IN  AL, PIO
    AND AL, 0FDH
    OUT PIO, AL
; FIN INICIALIZACION

    MOV BX, OFFSET MSJ
    MOV CL, OFFSET FIN - OFFSET MSJ
POLL: IN  AL, PIO
     AND AL, 1
     JNZ POLL
; Enviar carácter
    MOV AL, [BX]
    OUT PIO+1, AL
; Pulso STROBE
    IN  AL, PIO
    OR  AL, 02H
    OUT PIO, AL
; Reiniciar STROBE
    IN  AL, PIO
    AND AL, 0FDH
    OUT PIO, AL
    INC BX      ; Mover el puntero de la cadena
    DEC CL
    JNZ POLL    ; Verificar fin de la cadena
    INT 0
    END

```

Ejercicio 2c

```

PIO          EQU 30H

                ORG 1000H
NUM_CAR      DB 5
CAR          DB ?

; SUBROUTINA DE INICIALIZACION
; PIO PARA IMPRESORA
                ORG 3000H
INI_IMP:     MOV AL, 0FDH
                OUT PIO+2, AL
                MOV AL, 0
                OUT PIO+3, AL
                IN AL, PIO
                AND AL, 0FDH
                OUT PIO, AL
                RET

; PROGRAMA PRINCIPAL
                ORG 2000H
                PUSH AX
                CALL INI_IMP
                POP AX
                MOV BX, OFFSET CAR
                MOV CL, NUM_CAR
LAZO:         INT 6
POLL:         IN AL, PIO
                AND AL, 1
                JNZ POLL
                MOV AL, [BX]
                OUT PIO+1, AL
                PUSH AX
                CALL PULSO
                POP AX
                DEC CL
                JNZ LAZO
                INT 0
                END

```

```

; SUBROUTINA DE GENERACIÓN
; DE PULSO 'STROBE'
                ORG 4000H
PULSO:        IN AL, PIO
                OR AL, 02H
                OUT PIO, AL
                IN AL, PIO
                AND AL, 0FDH
                OUT PIO, AL
                RET

```

Ejercicio 2d

```

EOI    EQU 20h
IMR    EQU 21h
INT0   EQU 24h

```

```
IDINT0 EQU 10
```

```

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

```

```

                ORG 1000H
flag    db 0
longitud db 0
cadena db ?

```

```

                org 40
dir_rut dw rut_f10

```

```

                org 3000h
; cancelar interrupciones futuras
rut_f10: mov al, 0FFH
                out IMR, al
; indicamos al programa que no lea más
                mov flag, 1

```

```

    mov al, 20h
    out EOI, al
    iret

    ORG 2000H
    cli
; INICIALIZACION PIO PARA IMPRESORA
    MOV AL, 0FDH
    OUT CA, AL
    MOV AL, 0
    OUT CB, AL
    IN  AL, PA
    AND AL, 0FDH
    OUT PA, AL
; Inicialización del PIC
    mov al, 0FEh; FE = 1111 1110
    out IMR, al
    mov al, IDINT0
    out INT0, al
    sti

; Lectura de cadena
    MOV BX, OFFSET cadena
loop: int 6          ; leer char
    inc bx
    inc longitud
    cmp flag, 0 ; verifico si presionaron f10
    jz loop

; Impresión de los caracteres leídos
    MOV BX, OFFSET cadena ; reiniciar puntero al comienzo
POLL: nop
    IN  AL, PA
    AND AL, 1
    JNZ POLL
; Enviar carácter
    MOV AL, [BX]
    OUT PB, AL
; Pulso STROBE
    IN  AL, PA
    OR  AL, 02H
    OUT PA, AL
; Reiniciar STROBE
    IN  AL, PA
    AND AL, 0FDH
    OUT PA, AL
; pasar al siguiente char
    INC BX
    DEC longitud
    JNZ POLL
    INT 0
    END

```

Ejercicio 3a

```

HAND EQU 40H
ORG 1000H
MSJ   DB "INGENIERIA E      "
      DB "INFORMATICA"
FIN   DB ?

      ORG 2000H
      IN  AL, HAND+1
      AND AL, 7FH

```



```

        OUT HAND+1, AL
        MOV BX, OFFSET MSJ
        MOV CL, OFFSET FIN-OFFSET MSJ
POLL:   IN  AL, HAND+1
        AND AL, 1
        JNZ POLL
        MOV AL, [BX]
        OUT HAND, AL
        INC BX
        DEC CL
        JNZ POLL
        INT 0
        END

```

Ejercicio 3d

```

PIC      EQU 20H
HAND     EQU 40H
N_HND    EQU 10

```

```

IP_HND   ORG 40
        DW  RUT_HND

```

```

RUT_HND: ORG 3000H
        PUSH AX
        MOV AL, [BX]
        OUT HAND, AL
        INC BX
        DEC CL
        JNZ FINAL
        MOV AL, 0FFH
        OUT PIC+1, AL
FINAL:   MOV AL, 20H
        OUT PIC, AL
        POP AX
        IRET

```

```

MSJ      ORG 1000H
        DB  "UNIVERSIDAD      "
        DB  "NACIONAL DE LA PLATA"
FIN      DB  ?

```

```

        ORG 2000H
        MOV BX, OFFSET MSJ
        MOV CL, OFFSET FIN-OFFSET MSJ
        CLI
        MOV AL, 0FBH
        OUT PIC+1, AL
        MOV AL, N_HND
        OUT PIC+6, AL
        MOV AL, 80H
        OUT HAND+1, AL
        STI
LAZO:    CMP CL, 0
        JNZ LAZO
        IN  AL, HAND+1
        AND AL, 7FH
        OUT HAND+1, AL
        INT 0
        END

```

Ejercicio 4a

```

DIN      EQU 60h
DOUT     EQU 61h
CTRL     EQU 62H

```

```

        ORG 1000H
char     DB  "A"

```

```

; programa principal
        ORG 2000H
; programo la USART
; Bits de CTRL:
; Sync | ER | RTS | DTR | RxEN | TxEN | Vb | Sy/As
; Para comunicación asíncrona (Sy/As = 1)
; Velocidad 6 baudios (VB=0)
; Comunicación por DTR (DTR=1)
; Reiniciando flags de errores (ER =1)
; El resto no importa (x)
        MOV AL, 51H      ; binario=01010001 o x1x1xx01
        OUT CTRL, AL

```

```
POLL: IN  AL, CTRL
      AND AL, 81H
; verifico que el bit 0 y el 7
; estén ambos en 1
      CMP AL, 81H
      JNZ POLL
      MOV AL, char
      OUT DOUT, AL
      INT 0
      END
```

Ejercicio 4b

```
DIN  EQU 60h
DOUT EQU 61h
CTRL EQU 62H

      ORG 1000H
cadena DB "USART DTR POLLING"
fin    DB ?

; programa principal
      ORG 2000H
      MOV BX, OFFSET cadena
      MOV CX, OFFSET fin - OFFSET tabla
; programo la USART
      MOV AL, 51H      ; binario=01010001
      OUT CTRL, AL

POLL: IN  AL, CTRL
      AND AL, 81H
; verifico que el bit 0 y el 7
; estén ambos en 1
      CMP AL, 81H
      JNZ POLL
; Envío el caracter
      MOV AL, [BX]
      OUT DOUT, AL
      INC BX
      DEC CX
      JNZ POLL
      INT 0
      END
```

Ejercicio 4c

```
USART EQU 60H
XON    EQU 11H
XOFF   EQU 13H

; definición de datos
      ORG 1000H
caracteres DW 0
TABLA      DB "XON/XOFF Polling"
FIN        DB ?

; PROGRAMA PRINCIPAL
      ORG 2000H
INICIO: MOV BX, OFFSET TABLA      ; puntero a Tabla
; programo la USART
      MOV AL, 51H      ;binario= 01010001
      OUT USART+2, AL
TEST:  IN  AL, USART+2      ; espero a que se
      AND AL, 01H          ; envíe el carácter
      CMP AL, 01H          ; a la impresora.
      JNZ TEST
```

```

MOV AL, [BX]
OUT USART+1, AL
INC BX
INC caracteres
CMP caracteres, (OFFSET FIN) - (OFFSET TABLA)
JZ FINAL
IN AL, USART+2          ; Consulto si RxRDY
AND AL, 02H             ; se activó. De ser
CMP AL, 02H             ; así, la impresora
JZ RXON                 ; transmite un XON ó
JMP TEST               ; un XOFF al CPU.
; espera recibir XON
RECIBIR: IN AL, USART+2
AND AL, 02H
CMP AL, 02H
JNZ RECIBIR
RXON: IN AL, USART
MOV AH, AL
CMP AL, XON             ; si es XON sigo
JZ TEST                ; la impresión.
CMP AH, XOFF            ; si es XOFF espero
JZ RECIBIR              ; que libere el buffer
FINAL: INT 0
END

```

Anexo DMA

El formato del registro control es el siguiente

TC				MT	ST	TT	STOP
----	--	--	--	----	----	----	------

Donde:

TC: Terminal Count

MT: Modo de transferencia

ST: Sentido de transferencia

TT: Tipo de transferencia

STOP: habilitar o detener transferencia

Ejercicio 2

b) Para que el al HAND-SHAKE emita una interrupción, la línea busy del procesador debe estar en 0

c) El al HAND-SHAKE utiliza la línea DREC del CMDA para indicarle que debe iniciar la transferencia. Se comunican a través de la línea DREC y la línea DACK

d) EL DMAC lee desde memoria un byte, en la dirección especificada en el registro RF (compuesto por RFL y RFH). Luego envía ese byte al HAND-SHAKE cuando este le indica mediante DREQ que puede recibir datos. Finalmente, el HAND-SHAKE envía el carácter a la impresora.

e) El DMAC genera una interrupción cuando finaliza de enviar los caracteres a la impresora

f) Cuando todos los caracteres han sido enviados a la impresora, detectado mediante la variable FLAG cuyo valor se cambia desde la subrutina que maneja las interrupciones del CMDA (RUT_DMA)

Ejercicio 3a

Al ser memoria memoria, el bit TT=1. Al ser por robo de ciclo MT=0. Como queremos que se realice, STOP=0. Entonces el byte de configuración debe ser **XXXX0X10**

El carácter X indica que el valor no importa. El bit ST no importa porque es transferencia memoria memoria.

Ejercicio 3b

Al ser entre un Periférico y Memoria, el bit TT=0. Al ser Periférico → Memoria, el bit ST=0 Al ser por ráfagas, MT=1. Como queremos que se realice, STOP=0. Entonces el byte de configuración debe ser **XXXX1000**

El carácter X indica que el valor no importa.

Ejercicio 3c

Al ser entre un Periférico y Memoria, el bit TT=0. Al ser Memoria → Periférico, el bit ST=1 Al ser por robo de ciclo, MT=0. Como queremos que se realice, STOP=0. Entonces el byte de configuración debe ser **XXXX0100**

El carácter X indica que el valor no importa.

PRACTICA 4 - Algunas soluciones

Segmentación de cauce en procesador RISC

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
```

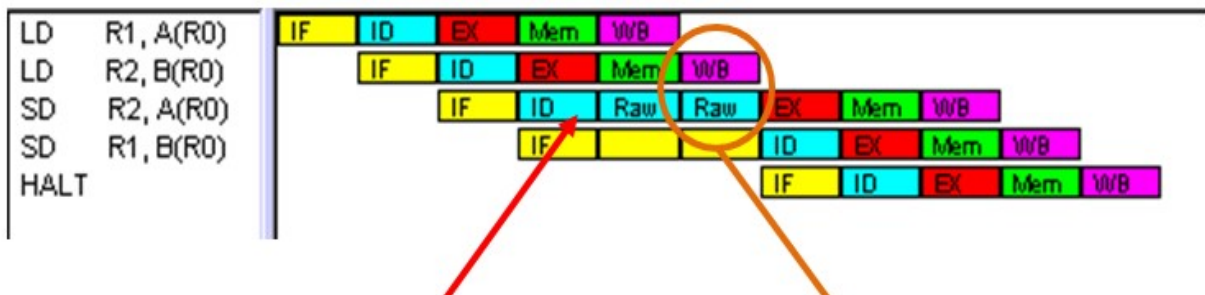
- a) Ejecutarlo en el simulador con la opción Configure/Enable Forwarding deshabilitada. Analizar paso a paso su funcionamiento, examinar las distintas ventanas que se muestran en el simulador y responder:

- ¿Qué instrucción está generando atascos (stalls) en el cauce (ó pipeline) y por qué?

La instrucción SD R2,A(r0) almacena en la dirección de memoria A el valor contenido en el registro R2, mientras que la instrucción anterior LD R2,B(r0) carga desde la dirección de memoria B el contenido de esa posición en el registro R2.

Cuando forwarding no está habilitado la instrucción SD R2,A(r0) que en la etapa ID trata de leer el contenido de R2, pero dicho contenido no estará disponible hasta que la instrucción anterior LD R2,B(r0) llegue a la etapa WB. Y Debido a esto se genera un atasco en la etapa ID donde se procesa la instrucción SD R2,A(r0) retrasando la salida de esta etapa (con RAWs) a la espera del contenido del registro. Esto además genera que la instrucción posterior SD R1, B(r0) deba permanecer en la etapa IF sin poder avanzar a la etapa ID (el pipeline se detuvo) como se puede ver en la imagen.

Tener en cuenta que en la primera mitad de la etapa WB se escribe el contenido en R2 y en el segundo RAW se lee el contenido de dicho registro.



- ¿Qué tipo de 'stall' es el que aparece?

Aparecen atascos de tipo RAW (Read After Write) causado por una dependencia de datos, en este caso se intenta leer un dato antes que esté guardado en el registro. Serán dos atascos, equivalentes a dos ciclos hasta que LD R2, B(r0) salga de la etapa WB.

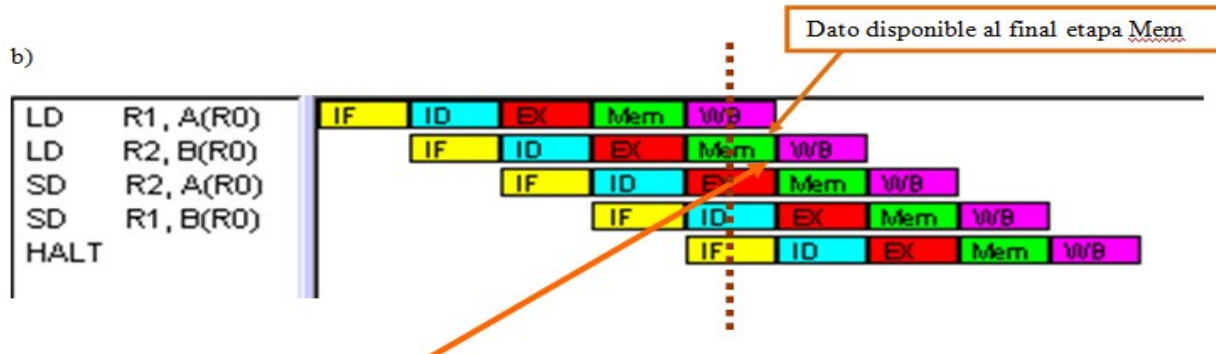
- ¿Cuál es el promedio de Ciclos Por Instrucción (CPI) en la ejecución de este programa bajo esta configuración?

El CPI de la ejecución de este programa es de 2.2



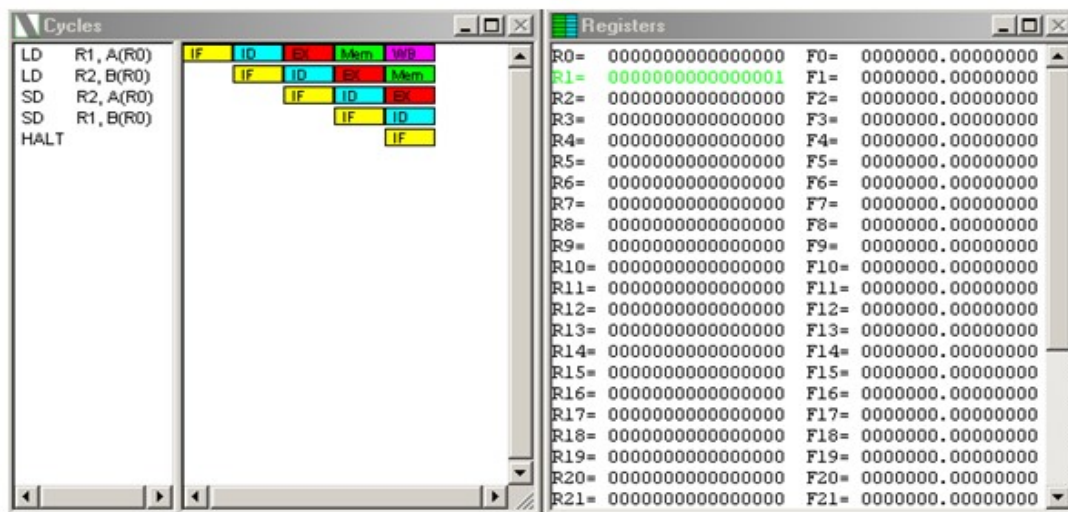
- b) Una forma de solucionar los atascos por dependencia de datos es utilizando el Adelantamiento de Operandos o Forwarding. Ejecutar nuevamente el programa anterior con la opción Enable Forwarding habilitada y responder:
- ¿Por qué no se presenta ningún atasco en este caso? Explicar la mejora.

Con la opción forwarding habilitada el dato contenido en el registro R2 podrá ser leído por la instrucción SD R2, A(R0) cuando la instrucción LD R2, B(R0) se encuentra finalizando la etapa MEM. La instrucción SD R2, A(R0) no tiene que esperar a que la instrucción LD R2, B(R0) salga de la etapa WB. De esta manera no aparecen atascos del tipo RAW.



- ¿Qué indica el color de los registros en la ventana Register durante la ejecución?

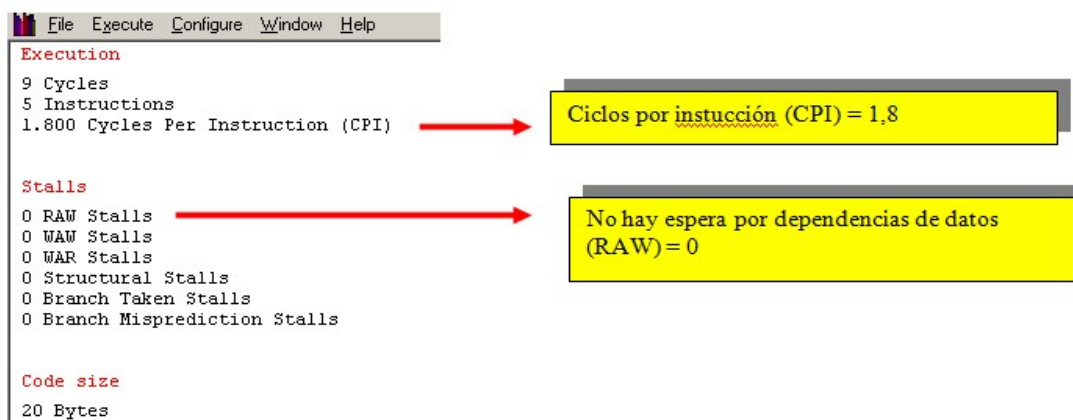
El color verde en el registro R1 significa que el dato está disponible en etapa MEM para adelantamiento.



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.

- ¿Cuál es el promedio de Ciclos Por Instrucción (CPI) en este caso? Comparar con el anterior.

En este caso el CPI es de 1,8 y por estar más cercana a 1 (valor ideal) la ejecución del programa está más cerca de un desempeño óptimo.



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
```

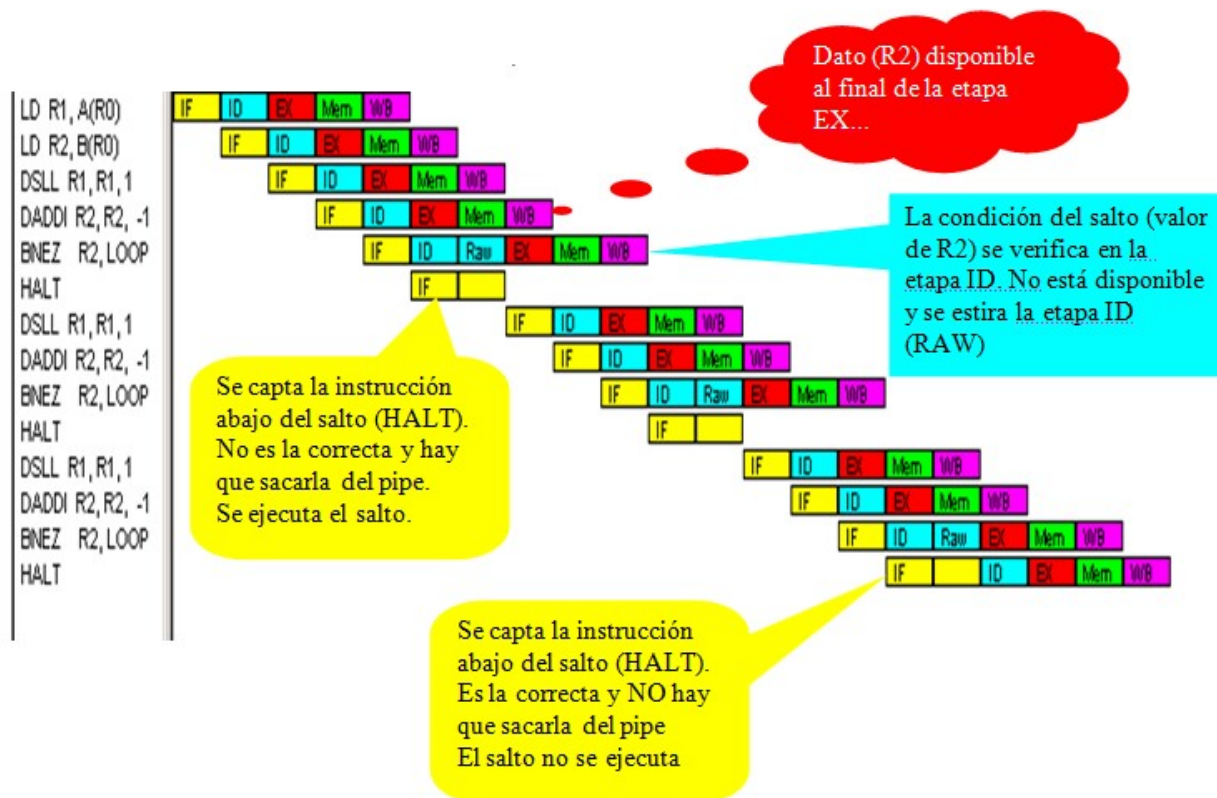
a) Ejecutar el programa con Forwarding habilitado y responder:

- ¿Por qué se presentan atascos tipo RAW?

Se presentan atasco por dependencia de datos de tipo RAW causado por la instrucción BNEZ R2, loop al procesarse en la etapa ID. Esta instrucción necesita del contenido del registro R2 que está siendo utilizado por la instrucción DADDI R2,R2,-1 en la etapa EX sin salir aún de esta.

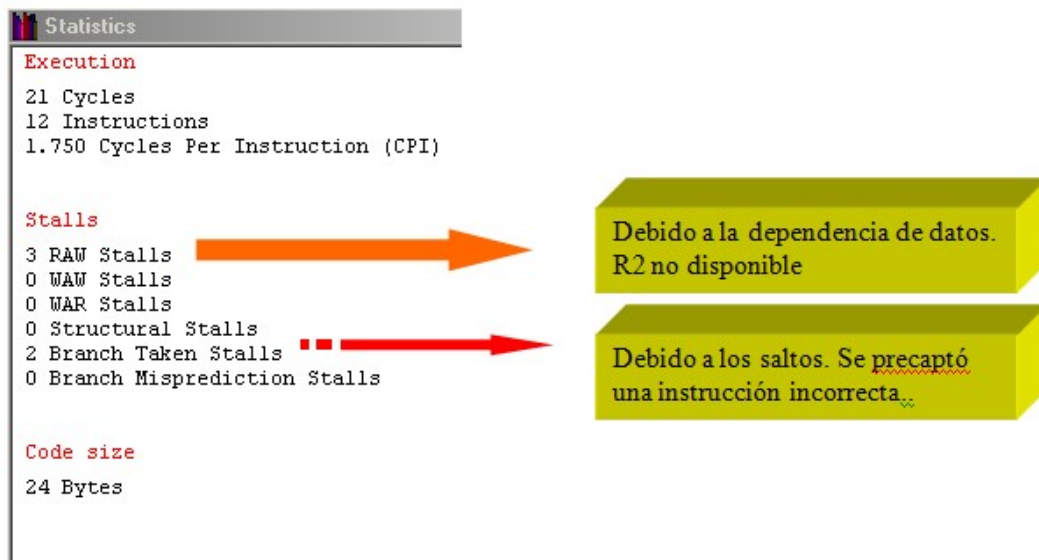
- Branch Taken es otro tipo de atasco que aparece. ¿Qué significa? ¿Por qué se produce?

El atasco de tipo Branch Taken Stalls (BTS), ocurre como consecuencia de la ejecución incorrecta de la instrucción siguiente a una instrucción condicional. Esto se debe a que 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.



- ¿Cuántos CPI tiene la ejecución de este programa? Tomar nota del número de ciclos, cantidad de instrucciones y CPI.

Se ejecutan 12 instrucciones en 21 ciclos dando un CPI de 1.750



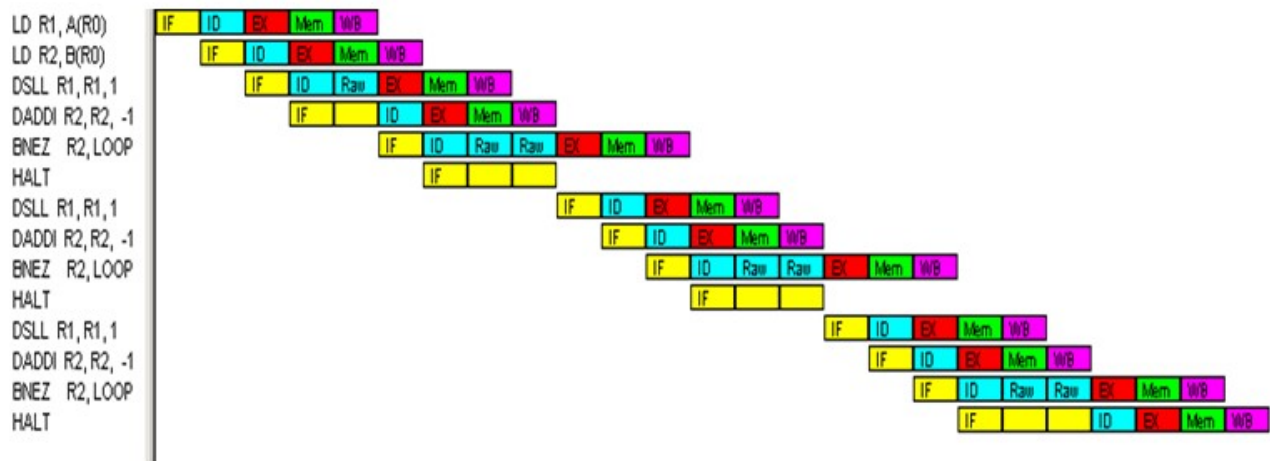
b) Ejecutar ahora el programa deshabilitando el Forwarding y responder:

- ¿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?

Las instrucciones que generan los atascos RAW son:

la instrucción DSLR R1,R1,1 (ver registro R1 coloreado en gris), que trata de leer el contenido del registro R1, mientras que la instrucción LD R1,A(r0) todavía no copio el contenido de la dirección de memoria A en R1 y permanece aún en la etapa WB (RAW durante 1 ciclo).

Y la instrucción BNEZ R2,loop (ver registro R2 coloreado en gris), que trata de leer el contenido del registro R2, mientras que DADDI R2,R2,-1 está buscando copiar el resultado de la operación en dicho registro, permaneciendo en la etapa MEM y posteriormente en la etapa WB (RAW durante 2 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.

Con forwarding deshabilitada, Los atascos por Branch Taken Stalls duran 2 ciclos en cada vuelta del lazo loop, mientras que con dicha opción habilitada se reducen a 1 ciclo por vuelta de lazo.

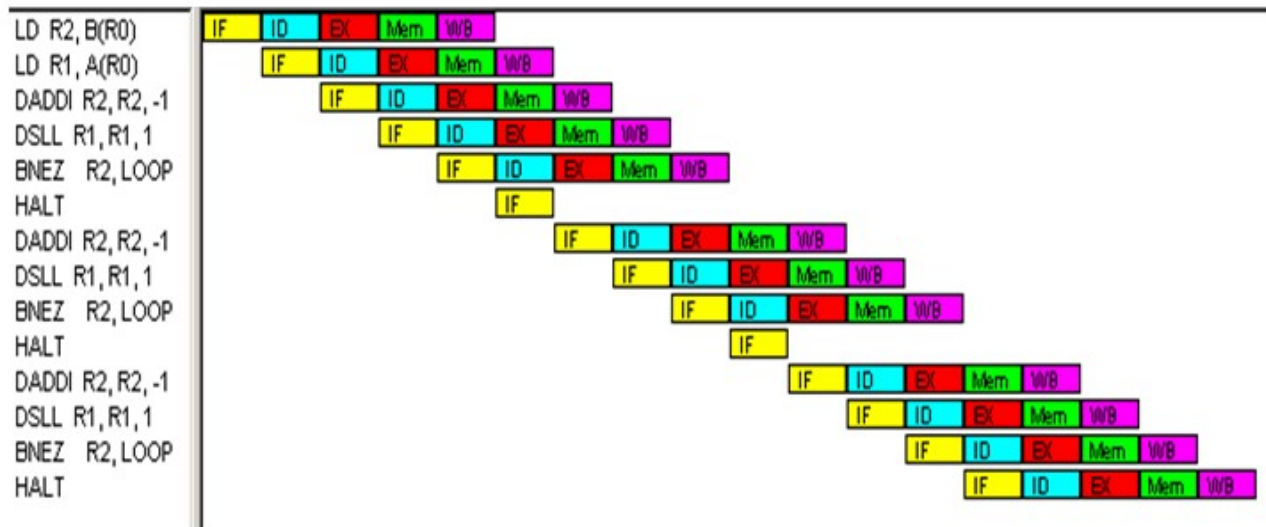
Esta diferencia tiene su causa en la instrucción condicional que es la que está generando los atascos RAW; entonces al disminuir la cantidad de RAWs producidos por esta, también disminuyen los ciclos de espera de la instrucción siguiente, que además se dejara de ejecutar si la condicional así se lo indica al procesador.

- ¿Cuántos CPI tiene la ejecución del programa en este caso? Comparar número de ciclos, cantidad de instrucciones y CPI con el caso con Forwarding.

A REALIZAR POR EL ALUMNO

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

Con la opción forwarding habilitada aún siguen apareciendo atascos RAW. Existe la posibilidad de tratar de eliminarlos reordenando las instrucciones en el programa y Dicho reordenamiento no debe afectar la lógica de este. El resultado de este método puede observarse en la imagen siguiente.



Statistics	
Execution	
18 Cycles	
12 Instructions	
1.500 Cycles Per Instruction (CPI)	
Stalls	
0 RAW Stalls	
0 WAW Stalls	
0 WAR Stalls	
0 Structural Stalls	
2 Branch Taken Stalls	
0 Branch Misprediction Stalls	
Code size	
24 Bytes	

- 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?

A REALIZAR POR EL ALUMNO

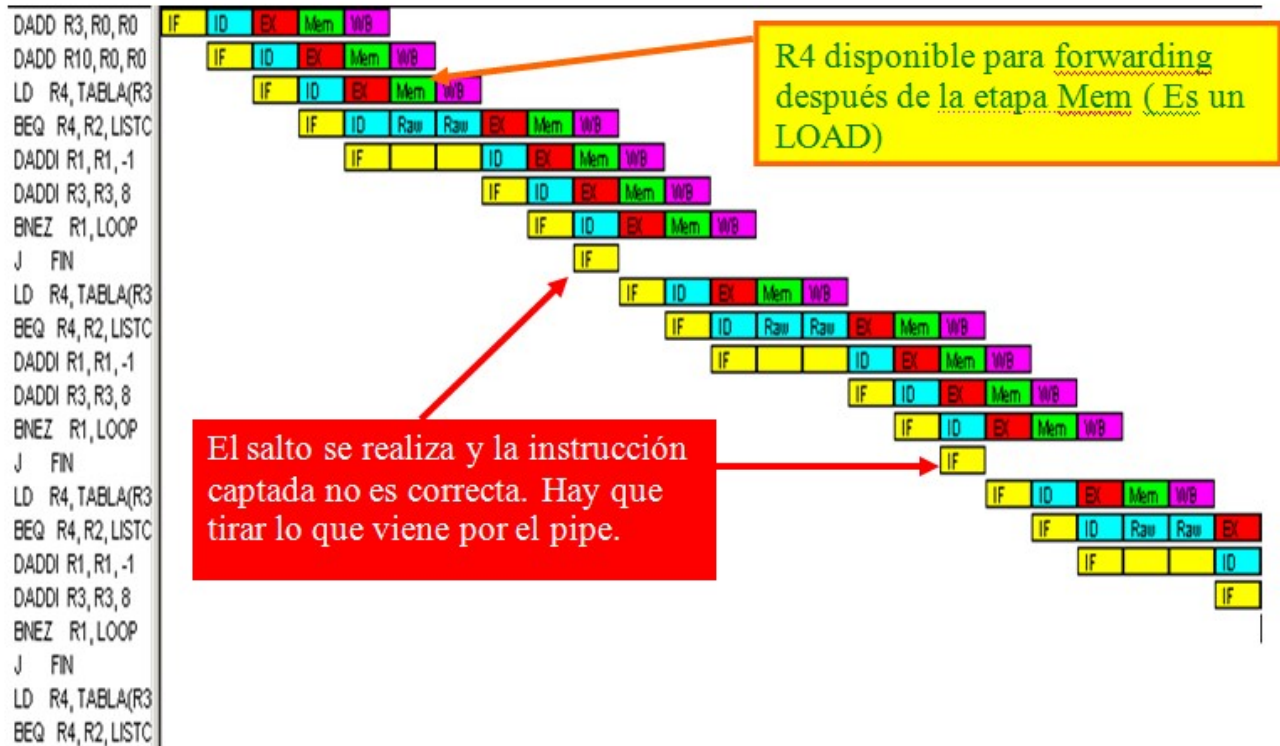
Ejercicio 4

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
```

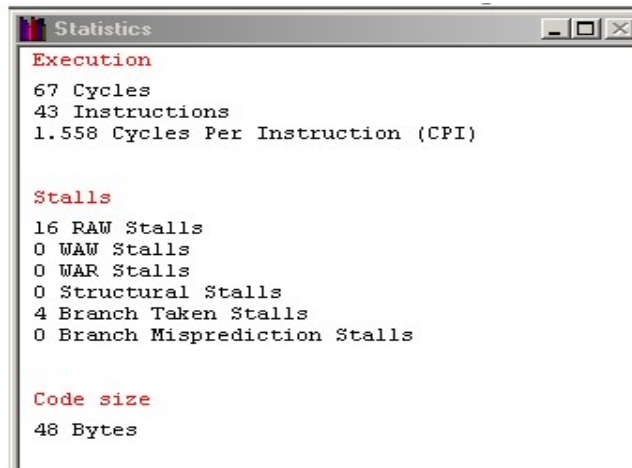
- Ejecutar en simulador con Forwarding habilitado. ¿Qué tarea realiza? ¿Cuál es el resultado y dónde queda indicado?
- Re-Ejecutar el programa con la opción Configure/Enable Branch Target Buffer habilitada. Explicar la ventaja de usar este método y cómo trabaja.

c) Confeccionar una tabla que compare número de ciclos, CPI, RAWs y Branch Taken Stalls para los dos casos anteriores. El programa busca en TABLA un elemento igual al contenido en la dirección de memoria NUM. En este caso dicha coincidencia se produce cuando el contenido del registro R4 es igual al contenido del registro R2 ($R4=R2$), razón por la cual luego de evaluada esta condición y de resultar verdadera se salta a la posición de memoria indicada por la etiqueta "listo". Cuando hay coincidencia la línea de programa en listo suma al registro R10 un 1, caso contrario el contenido del registro R10 queda en 0. Este es el resultado y queda almacenado en el registro R10. El registro R3 se utiliza como índice para recorrer la TABLA. El contenido del registro R3 se incrementa de 8 porque cada elemento de tabla es del tamaño word, es decir de 64 bits (8 bytes).



Statistics	
Execution	
71 Cycles	
43 Instructions	
1.651 Cycles Per Instruction (CPI)	
Stalls	
16 RAW Stalls	
0 WAW Stalls	
0 WAR Stalls	
0 Structural Stalls	
8 Branch Taken Stalls	
0 Branch Misprediction Stalls	
Code size	
48 Bytes	

Habilitando la opción Branch Target Buffer (BTB) logramos reducir los atascos Branch Taken stalls a la mitad. Tener en cuenta que esta opción es útil cuando aumenta la cantidad de iteraciones de un lazo. Como vemos también esta opción no actúa sobre los atascos por dependencia de datos (RAW en este caso) que no se modifican.



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.

Definir en la zona de memoria de datos una tabla (TABLA) con 10 números enteros, a continuación un arreglo (RES) donde se cargaran 1s o 0s según los números definidos en tabla sean "mayor" o "menor o igual" que un número (X) que en memoria a continuación del arreglo RES. También definiremos en memoria una variable LONG que valdrá 10 y otra CANT inicializada en 0. Todas estas variables serán de tamaño word. Faltaría definir una máscara para detectar si el número verificado es mayor o menor, para este fin utilizaremos la 0x8000 como MASK que también definiremos en memoria (con una operación AND logramos este cometido).

Recordar que se debe contabilizar la cantidad de números que son mayores que el número "x" y guardar ese resultado en la variable en memoria CANT.

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.

El programa podría ser así:

```

.data
NUM1: .word 3
NUM2: .word 8
RES:   .word 0
.code
    ld    R1, NUM1(r0)
    ld    R2, NUM2(r0)
    dadd  R3, R0, R0
loop:  dadd R3, R3, R1
    daddi R2, R2, -1
    bnez  R2, loop
    sd    R3, RES(r0)
    halt

```

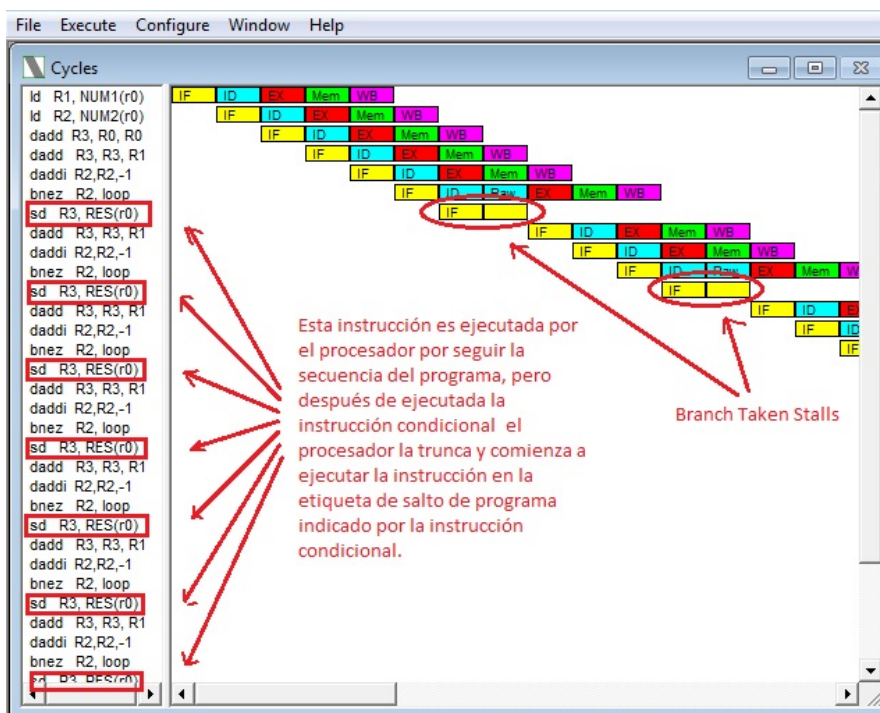
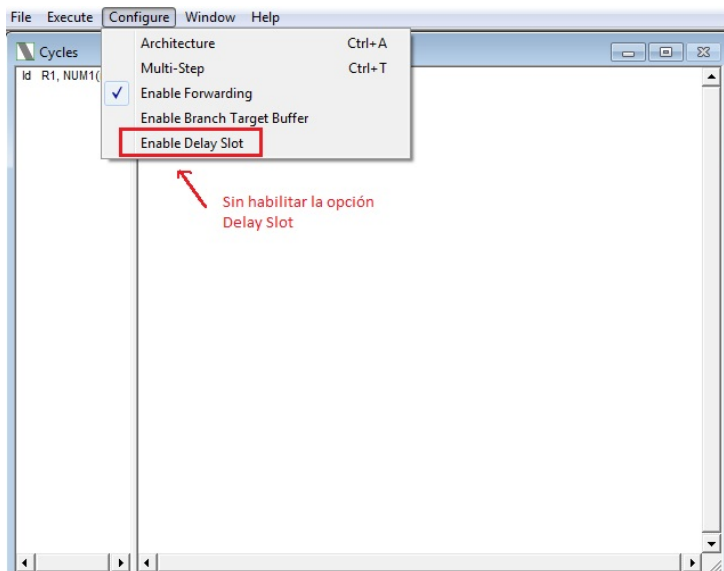
Otra versión para que puedan analizar es:

```

.data
num1: .word 3
num2: .word 5
res:   .word 0
.code
    LD    r1, num1(r0)
    LD    r2, num2(r0)
    DADD  r10, r0, r0
LOOP:   DADDI r2, r2, -1
    BNEZ  r2, LOOP
    DADD  r10, r10, r1
    SD    r10, res(r0)
    HALT

```

Su ejecución en el simulador con la opción Delay Slot deshabilitada mostraría lo siguiente:



Statistics	
Execution	
48 Cycles	
29 Instructions	
1.655 Cycles Per Instruction (CPI)	
Stalls	
8 RAW Stalls	
0 WAW Stalls	
0 WAR Stalls	
0 Structural Stalls	
7 Branch Taken Stalls	
0 Branch Misprediction Stalls	
Code size	
32 Bytes	

Con opción
Delay Slot
deshabilitada

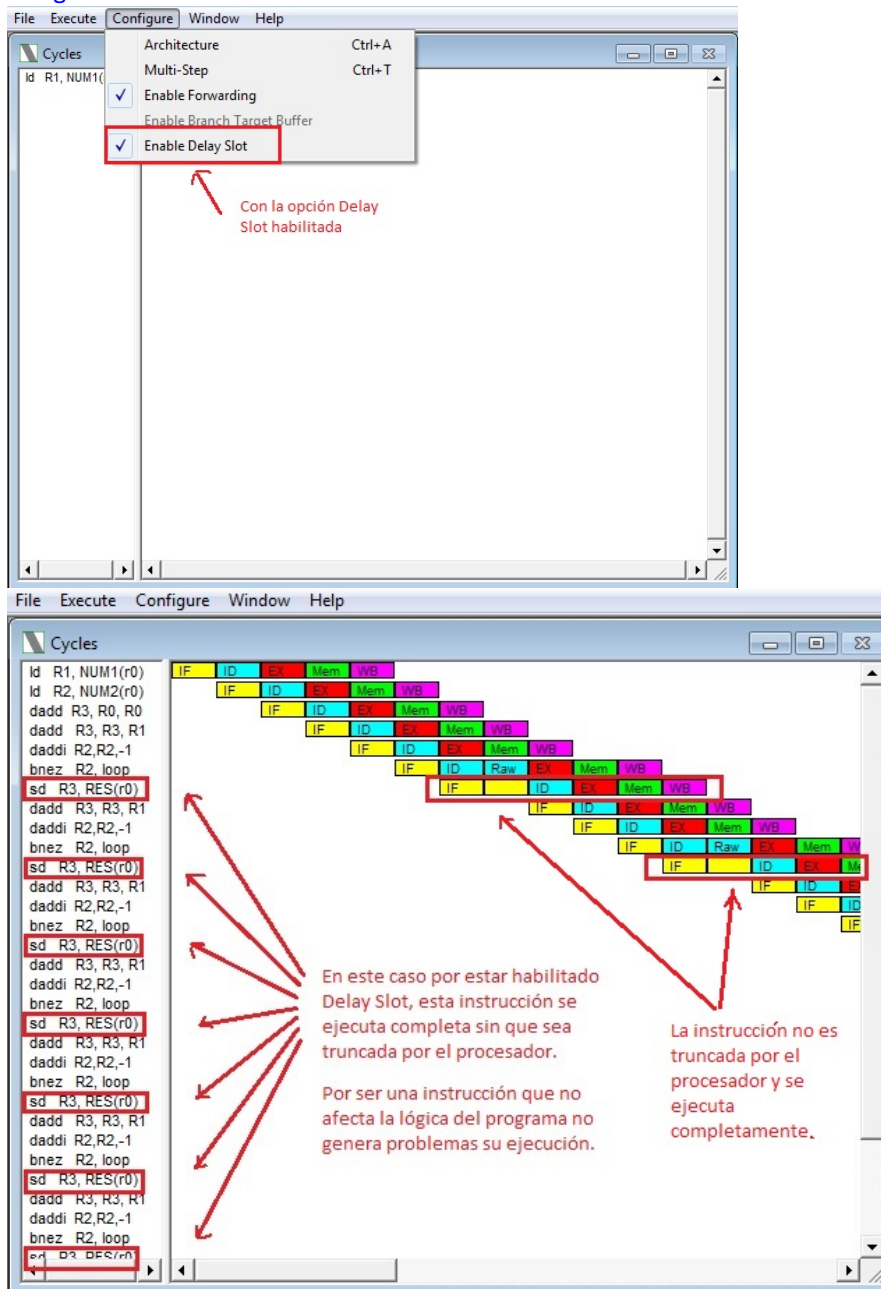
Los atascos por dependencia de
datos RAW, son independientes
de estos atascos Branch Taken
Stalls.

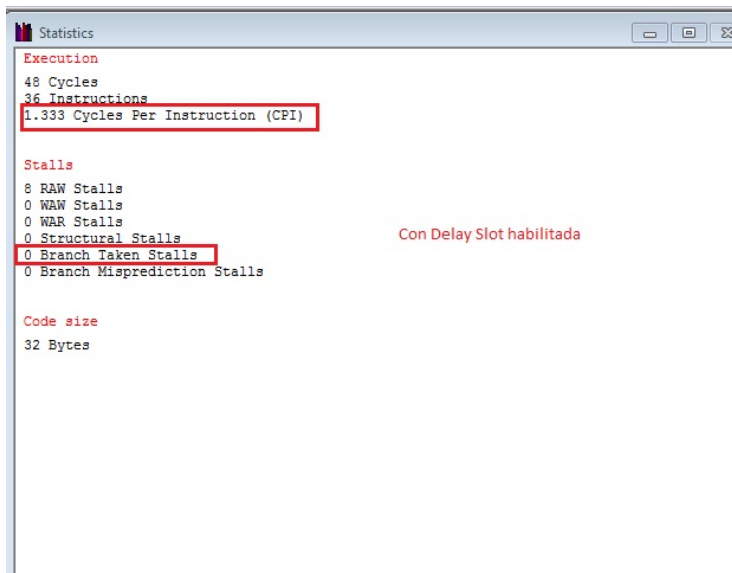
Con la opción Delay Slot habilitada se podrá observar que eliminamos los atascos Branch Taken Stalls, logrando optimizar el CPI de 1.655 a 1.333.

Tener presente que habilitar esta opción equivale a ejecutar completamente la instrucción siguiente a una

instrucción condicional, razón por la cual nunca debe haber una instrucción HALT inmediatamente después de una instrucción condicional.

Por otro lado se debe aprovechar esta bondad de la opción, para colocar luego de las instrucciones condicionales aquellas que deban ejecutarse en algún momento del programa y que al permutarlas a esa posición no cambien la lógica de este.





PRACTICA 5

EJERCICIOS RESUELTOS

Procesador RISC: instrucciones de Punto Flotante y pasaje de parámetros

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`. Del set de instrucciones:

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

Para copiar el valor que tengo en un registro entero (`r0` a `r31`) a uno de punto flotante (`f0` a `f31`):

- Copiar los 64 bits del registro entero `rf` al registro `fd` de punto flotante
 - `mtc1 rf, fd`
- Convertir a punto flotante el valor entero copiado al registro `ff`, dejándolo en `fd`
 - `cvt.d.l fd, ff`

Importante: los números muy grandes serán redondeados en su mejor representación de punto flotante.

Para copiar el valor que tengo en un registro de punto flotante (`f0` a `f31`) a un registro entero (`r1` a `r31`):

- Convertir a entero el valor en punto flotante contenido en `ff`, dejándolo en `fd`
 - `cvt.l.d fd, ff`
- Copiar los 64 bits del registro `ff` de punto flotante al registro `rd` entero
 - `mfc1 rd, ff`

Importante: El número se trunca, no se redondea.

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:

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

```

.data
base: .double 5.85
altura: .double 13.47
sup: .double 0.0

.code
l.d    f1, base(R0)
l.d    f2, altura(R0)
mul.d  f3, f2, f1 ; f3 = base * altura

daddi  r2, r0, 2
mtc1   r2, f4
cvt.d.l f5, f4 ; f5 = 2.0

div.d  f6, f3, f5 ; f6 = base * altura / 2
s.d    f6, superficie(r0)
Halt

```

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.

```

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

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

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

```

9) 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 y debe retornar el valor 1 si el carácter es una vocal, o 0 en caso contrario

```

        .data
letra:   .ascii 'O'
vocales: .asciiz 'AEIOUaeiou'
result:  .word 0

        .code
        lbu $a0, letra($0)
        jal es_vocal
        sd $v0, result($zero)
        halt

es_vocal: dadd $v0, $0, $0
          daddi $t0, $0, 0
          loop: lbu $t1, vocales($t0)
                beqz $t1, fin_vocal
                beq $a0, $t1, si_es_voc
                daddi $t0, $t0, 1
                j loop
si_es_voc: daddi $v0, $0, 1
fin_vocal: jr $ra

```

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
(1)      daddi $sp, $zero, 0x400      ; Inicializa puntero al tope de la pila

        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:

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

```

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

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

```

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

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