



**Arellano Granados Angel Mariano**

**218123444**

**Seminario de Traductores de Lenguajes I**

**I7026 D02**

**Reporte de Actividades 5**

**Actividades 5 – Parte 1 – 3**

# Actividad 5 – Parte 1

## Descripción

Ejecutar el siguiente programa y responde las siguientes preguntas

1. ¿Cómo se ejecutan las llamadas a procedimientos?
2. ¿Cuál es la diferencia entre FAR y NEAR?
3. ¿Qué produce la línea de código <SHL CX,1> en el registro CX?
4. ¿Qué registros participan en la ejecución de la instrucción RET?

## Desarrollo y Resultados

### 1. ¿Cómo se ejecutan las llamadas a procedimientos?

Con la instrucción CALL <nombre del procedimiento>.

Aparece la línea CALL B10 que llama al proceso llamado “B10”

The screenshot shows the 8086 emulator interface. The registers window on the left shows CS:0714 and IP:0006. The instruction list window in the center shows the instruction CALL B10 at address 07146. The source code window on the right shows the assembly code with CALL B10 highlighted.

Registers	H	L
AX	00	01
BX	00	01
CX	00	5D
DX	00	00
CS	0714	
IP	0006	
SS	0710	
SP	0040	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
07140: B8 184 0001  CALL 0000Eh
07141: 01 001 04C00h  MOV AX, 04C00h
07142: 00 000 NI      INT 021h
07143: BB 187 00001h  MOV CX, 00001h
07144: 01 001 00017h  CALL 00017h
07145: 00 000 NI      SHL CX, 1
07146: E8 232 0000h  RET
07147: 05 005 00001h  ADD AX, 00001h
07148: 00 000 NI      ADD BX, AX
07149: B8 184 0000h  RET
0714A: 00 000 NI      NOP
0714B: 4C 076 L        NOP
0714C: CD 205 =       NOP
0714D: 21 033 !       NOP
0714E: B9 185 !       NOP
0714F: 01 001 0000h  NOP
07150: 00 000 NI      NOP
07151: FA 222 b       NOP
```

Entonces salta al inicio del procedimiento “B10”

The screenshot shows the 8086 emulator interface. The registers window on the left shows CS:0714 and IP:000E. The instruction list window in the center shows the instruction MOV CX, 00001h at address 0714E. The source code window on the right shows the assembly code with the B10 procedure highlighted.

Registers	H	L
AX	00	01
BX	00	01
CX	00	5D
DX	00	00
CS	0714	
IP	000E	
SS	0710	
SP	003E	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

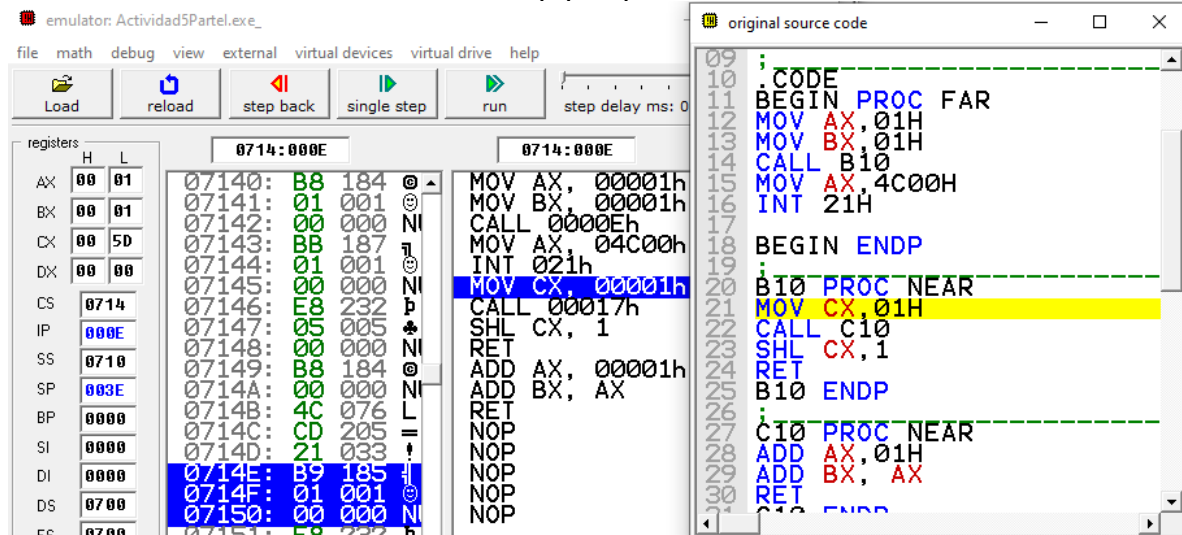
```
07140: B8 184 0001  CALL 0000Eh
07141: 01 001 04C00h  MOV AX, 04C00h
07142: 00 000 NI      INT 021h
07143: BB 187 00001h  MOV CX, 00001h
07144: 01 001 00017h  CALL 00017h
07145: 00 000 NI      SHL CX, 1
07146: E8 232 0000h  RET
07147: 05 005 00001h  ADD AX, 00001h
07148: 00 000 NI      ADD BX, AX
07149: B8 184 0000h  RET
0714A: 00 000 NI      NOP
0714B: 4C 076 L        NOP
0714C: CD 205 =       NOP
0714D: 21 033 !       NOP
0714E: B9 185 !       MOV CX, 00001h
0714F: 01 001 0000h  NOP
07150: 00 000 NI      NOP
07151: FA 222 b       NOP
```

## 2. ¿Cuál es la diferencia entre FAR y NEAR?

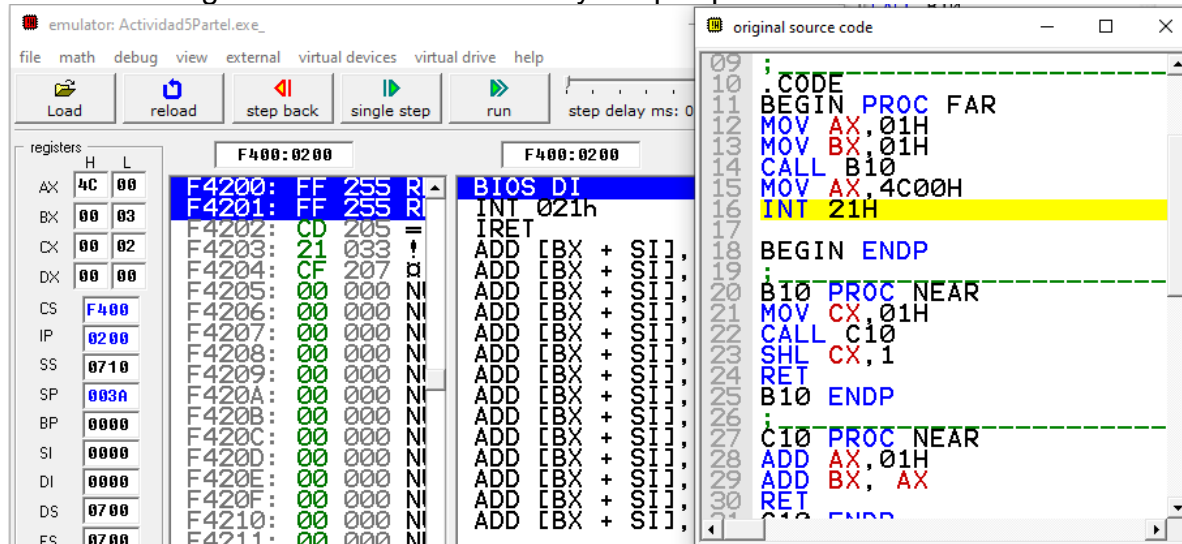
Near contains a 16-bit offset. For calls it will save the IP only.

Far contains a segment and a 16-bit offset. For calls it will save IP and CS.

Cuando se llama a B10 solo cambia Ip porque es NEAR:



Cuando se regresa a BEGIN cambia IP y CS porque es FAR:



## 3. ¿Qué produce la línea de código <SHL CX,1> en el registro CX?

hace un desplazamiento lógico a la izquierda SHL destino, origen.

CX antes de ejecutar la instrucción:

emulador: Actividad5Partel.exe

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	00	02
BX	00	03
CX	00	01
DX	00	00
CS	0714	
IP	0014	
SS	0710	
SP	003E	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0714:0016

07154:	D1	209	D
07155:	E1	225	B
07156:	C3	195	F
07157:	05	005	♣
07158:	01	001	⊙
07159:	00	000	NI
0715A:	03	003	♥
0715B:	D8	216	i
0715C:	C3	195	F
0715D:	90	144	E
0715E:	90	144	E
0715F:	90	144	E
07160:	90	144	E
07161:	90	144	E
07162:	90	144	E
07163:	90	144	E
07164:	90	144	E
07165:	90	144	E

RET  
ADD AX, 00001h  
ADD BX, AX

extended value viewer

watch: CX

word byte

	H	L
hex:	00	01
bin:	00000000	00000001
oct:	000	001

decimal 8 bit

unsigned:	0	1
signed:	0	1

original source code

```

10 ; CODE
11 BEGIN PROC FAR
12 MOV AX, 01H
13 MOV BX, 01H
14 CALL B10
15 MOV AX, 4C00H
16 INT 21H
17
18 BEGIN ENDP
19
20 B10 PROC NEAR
21 MOV CX, 01H
22 CALL C10
23 SHL CX, 1
24 RET
25 B10 ENDP
26
27 C10 PROC NEAR
28 ADD AX, 01H
29 ADD BX, AX
30 RET
31 C10 ENDP

```

CX después de hacer el desplazamiento:

emulador: Actividad5Partel.exe

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	00	02
BX	00	03
CX	00	02
DX	00	00
CS	0714	
IP	0016	
SS	0710	
SP	003E	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0714:0016

07154:	D1	209	D
07155:	E1	225	B
07156:	C3	195	F
07157:	05	005	♣
07158:	01	001	⊙
07159:	00	000	NI
0715A:	03	003	♥
0715B:	D8	216	i
0715C:	C3	195	F
0715D:	90	144	E
0715E:	90	144	E
0715F:	90	144	E
07160:	90	144	E
07161:	90	144	E
07162:	90	144	E
07163:	90	144	E
07164:	90	144	E
07165:	90	144	E

RET  
ADD AX, 00001h  
ADD BX, AX

extended value viewer

watch: CX

word byte

	H	L
hex:	00	02
bin:	00000000	00000010
oct:	000	002

decimal 8 bit

unsigned:	0	2
signed:	0	2

original source code

```

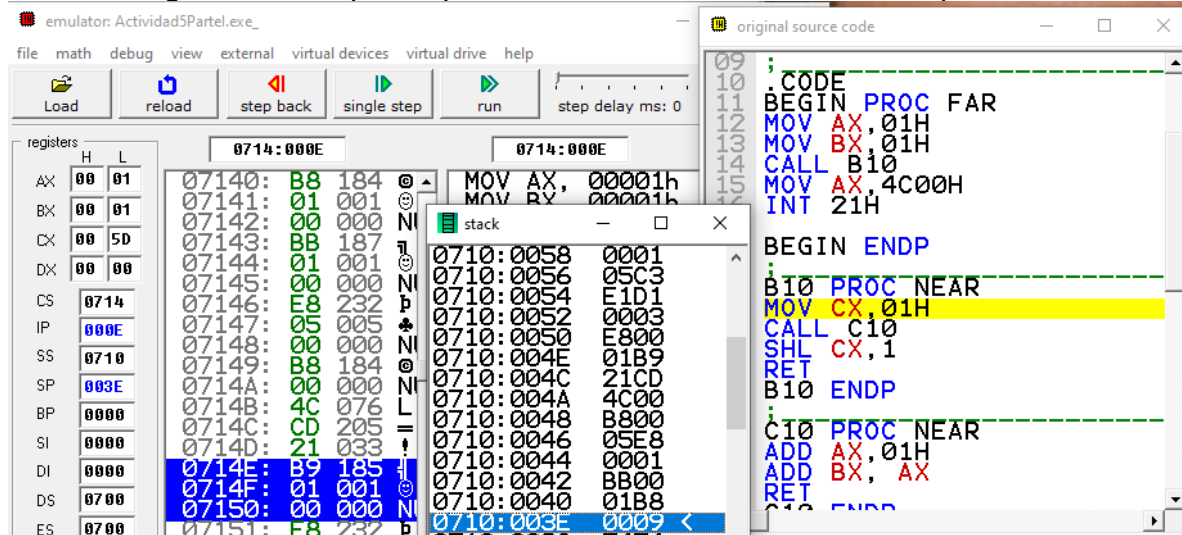
10 ; CODE
11 BEGIN PROC FAR
12 MOV AX, 01H
13 MOV BX, 01H
14 CALL B10
15 MOV AX, 4C00H
16 INT 21H
17
18 BEGIN ENDP
19
20 B10 PROC NEAR
21 MOV CX, 01H
22 CALL C10
23 SHL CX, 1
24 RET
25 B10 ENDP
26
27 C10 PROC NEAR
28 ADD AX, 01H
29 ADD BX, AX
30 RET
31 C10 ENDP

```

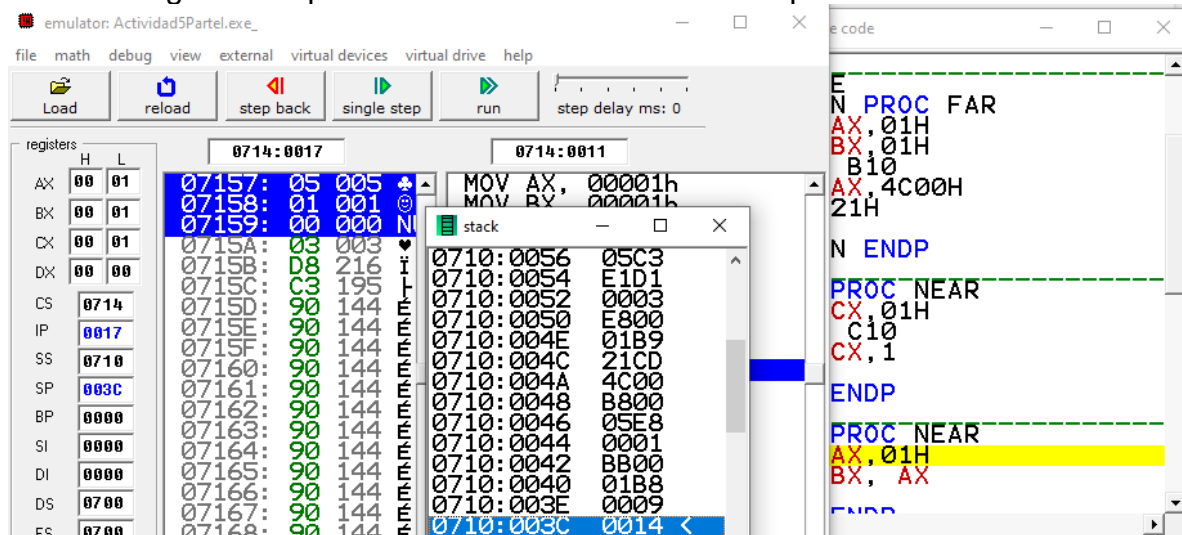
#### 4. ¿Qué registros participan en la ejecución de la instrucción RET?

Se usan los registros IP y SP porque CALL guarda en la pila el puntero de la instrucción a la que tendrá que regresar tras terminar el proceso, RET toma el elemento más reciente de la pila y lo coloca en IP así se mantiene el flujo del programa.

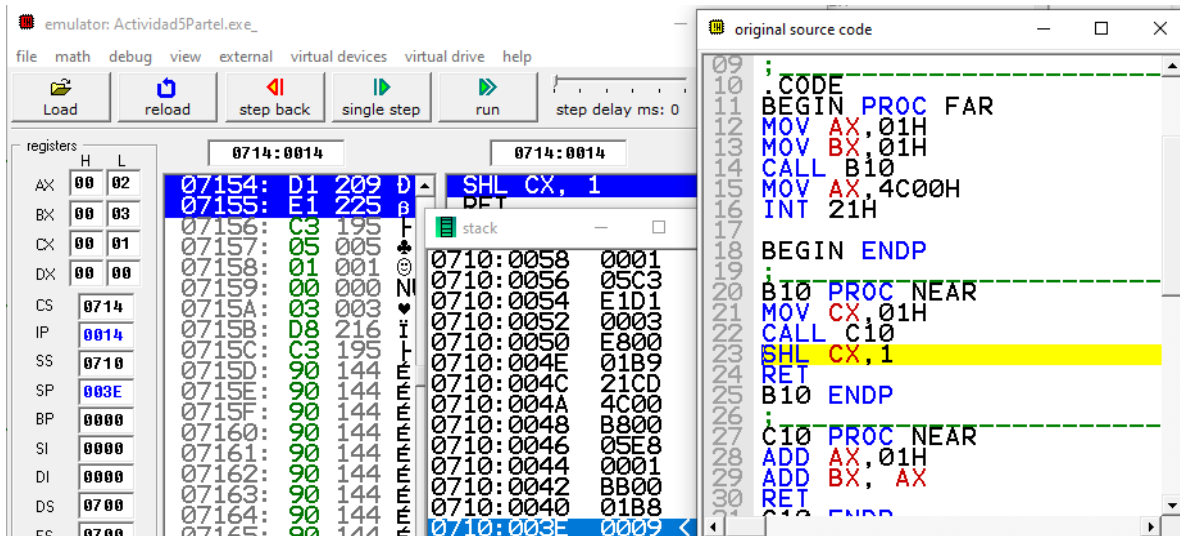
CALL B10 guarda en la pila el puntero a la instrucción 0009 en la pila:



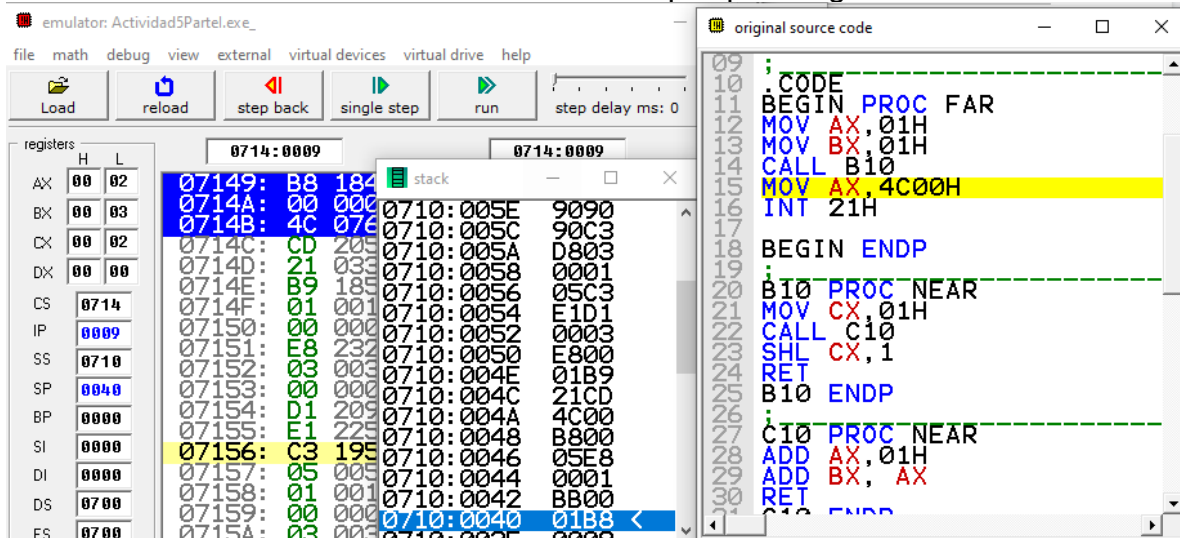
CALL C10 guarda el puntero a la instrucción 0014 en la pila:



La instrucción RET en C10 toma el 0014 de la pila para regresar a B10:



La instrucción RET en B10 toma el 0009 en la pila para regresar a BEGIN:



## Reflexión

Los procedimientos son la instrucción equivalente a las funciones en C junto a las etiquetas y las macros que nos ayudan a modular nuestros programas y no repetir código innecesario, también vimos instrucciones mas complejas en sus aplicaciones como los desplazamientos de bits que en este punto del curso aun no les encuentro una utilidad.

## Actividad 5 – Parte 2

### Descripción

Investigue el funcionamiento de la instrucción LEA, ejecute el siguiente código fuente y explique paso a paso lo que sucede.

### Desarrollo y Resultados

#### Instrucción LEA

Carga en un registro especificado la dirección efectiva especificada como en el operando origen:

***LEA reg, mem***

Ejemplos:

LEA AX, [BX] ; carga en AX la dirección apuntada por BX.

LEA AX, 3[BP+SI] ; carga en AX la dirección resultado de multiplicar por 3 la suma de los contenidos de BP y SI.

Para este tipo de acceso (el del segundo ejemplo) la instrucción LEA es más eficiente, que con la instrucción MOV e instrucciones de multiplicación.

[https://www.cs.buap.mx/~mgonzalez/asm\\_mododir2.pdf](https://www.cs.buap.mx/~mgonzalez/asm_mododir2.pdf)

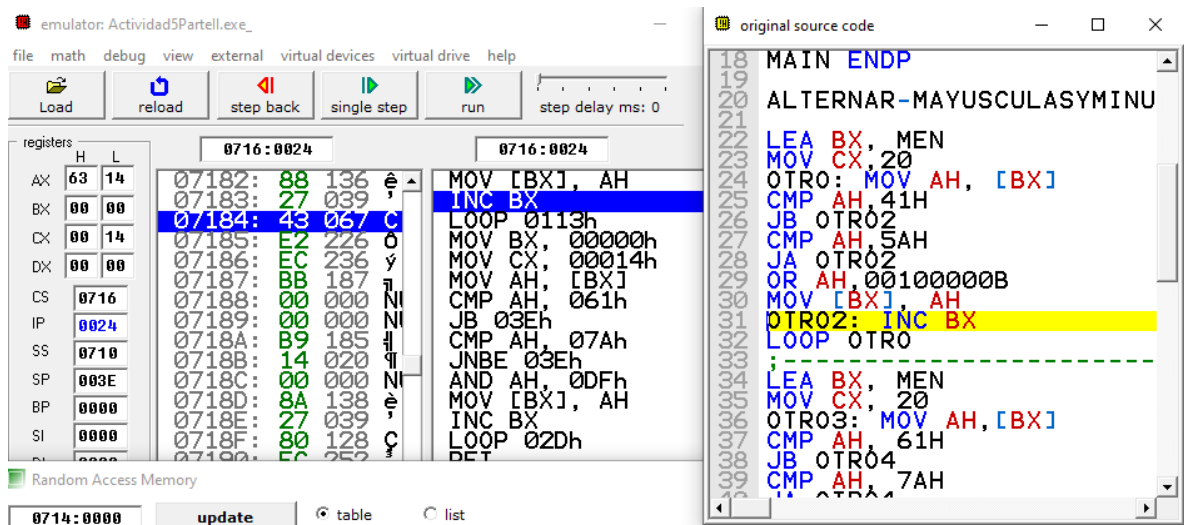
#### Reporte

Tras declarar todas las generalidades del programa y se declara una variable llamada "MEN" con la cadena 'CAMBIAR A MINUSCULAS'.

Ya iniciando el segmento de código se declara un procedimiento llamado "MAIN" que carga el segmento de datos y llama a otro procedimiento "ALTERNAR-MAYUSCULASYMINUSCULAS"

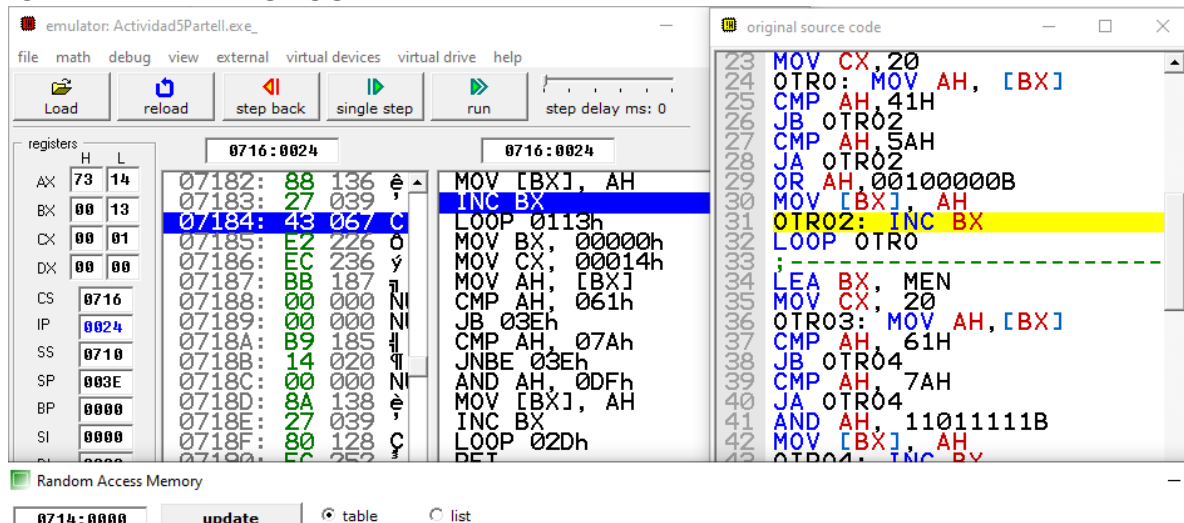
Donde este con la instrucción LEA coloca en la dirección de memoria de "MEN" en BX, coloca un 20d en CX (El número de caracteres en "MEN") y comienza una etiqueta llamada "OTRO" que mueve a AH el valor de cada carácter en la cadena "MEN" y compara si AH es 41h si es menor salta directamente a la etiqueta llamada "OTRO2", si no pasara de todas formas pero antes compara si AH es 5Ah si es mayor salta directamente a "OTRO2", si no se hace una función OR en AX con 00100000b esto equivale a sumar 20h al número lo cual convierte al carácter a su misma letra en minúscula y terminar moviendo el número resultante de AH y moverlo a la dirección de memoria de "MEN".





0714:0000	63	41	40	42	49	41	52	20-41	20	40	49	4E	55	53	43	cAMBIAR A MINUSC
0714:0010	B5	4C	41	53	00	00	00	00-00	00	00	00	00	00	00	00	ULAS.....
0714:0020	B8	14	07	8E	D8	E8	05	00-B8	00	4C	CD	21	BB	00	00	@.Aib+.@.L=!.:
0714:0030	B9	14	00	8A	27	80	FC	41-72	0A	80	FC	5A	77	05	80	!q.e.C.Ar.C.zw+C
0714:0040	CC	20	88	27	43	E2	EC	BB-00	00	B9	14	00	8A	27	80	f e.Coyh..!q.e.C
0714:0050	FC	61	72	0A	80	FC	7A	77-05	80	E4	DF	88	27	43	E2	ar.C.zw+C@e.C@
0714:0060	EC	C3	90	90	90	90	90	90-90	90	90	90	90	90	90	90	yEEEEEEEEEEEEEEE
0714:0070	90	90	90	90	90	90	F4	00-00	90	00	00	00	00	00	00	EEEEEE.....

Repitiendo esto un total de 20 veces con un LOOP cambia la cadena de "MEN" de 'CAMBIAR A MINUSCULAS' a 'cambiar a minúsculas'



0714:0000	63	61	60	62	69	61	72	20-61	20	60	69	6E	75	73	63	cambiar a minusc
0714:0010	75	6C	61	73	00	00	00	00-00	00	00	00	00	00	00	00	ulas.....
0714:0020	B8	14	07	8E	D8	E8	05	00-B8	00	4C	CD	21	BB	00	00	@.Aib+.@.L=!.:

Se reinicia el registro BX y se vuelve a poner 20d en CX para entrar a la etiqueta "OTRO3" que hace básicamente lo mismo que "OTRO" pero este cambia los números de las comparaciones y del OR para esta vez cambiar las letras de la cadena "MEN" de minúsculas a mayúsculas.



emulador: Actividad5Partell.exe\_

file

math

debug

view

external

virtual devices

virtual drive

help

Load

reload

step back

single step

run

step delay ms: 0

registers

	H	L
AX	43	14
BX	00	00
CX	00	14
DX	00	00
CS	0716	
IP	003E	
SS	0710	
SP	003E	
BP	0000	
SI	0000	

0716:003E

0716:003E

07194:	80	128	C	LOOP 0113h
07195:	FC	252		MOV BX, 00000h
07196:	7A	122	z	MOV CX, 00014h
07197:	77	119	w	MOV AH, [BX]
07198:	05	005		CMP AH, 061h
07199:	80	128	C	JB 03Eh
0719A:	E4	228		CMP AH, 07Ah
0719B:	DF	223		JNBE 03Eh
0719C:	88	136	e	AND AH, 0DFh
0719D:	27	039		MOV [BX], AH
0719E:	43	067	C	INC BX
0719F:	E2	226	o	LOOP 02Dh
071A0:	EC	236	y	RET
071A1:	C3	195	f	NOP
071A2:	00	144		NOP

Random Access Memory

0714:0000

update

table

list

original source code

```

30 MOV [BX], AH
31 OTR02: INC BX
32 LOOP OTR0
33 ;
34 LEA BX, MEN
35 MOV CX, 20
36 OTR03: MOV AH, [BX]
37 CMP AH, 61h
38 JB OTR04
39 CMP AH, 7Ah
40 JA OTR04
41 AND AH, 11011111B
42 MOV [BX], AH
43 OTR04: INC BX
44 LOOP OTR03
45 RET
46
47 ALTERNAR-MAYUSCULASYMINU
48
49
50 END MAIN
51

```

0714:0000 43 61 6D 62 69 61 72 20-61 20 6D 69 6E 75 73 63 Cambiar a minusc  
0714:0010 75 6C 61 73 00 00 00 00-00 00 00 00 00 00 00 00 ulas.....  
0714:0020 B8 14 07 8E D8 E8 05 00-B8 00 4C CD 21 BB 00 00 00 00 01.A!p+.0.L=!1..

También entrará en un LOOP que se repetirá 20 veces hasta que la cadena regrese a su estado original con todas sus letras en mayúsculas.

emulador: Actividad5Partell.exe\_

file

math

debug

view

external

virtual devices

virtual drive

help

Load

reload

step back

single step

run

step delay ms: 0

registers

	H	L
AX	53	14
BX	00	14
CX	00	01
DX	00	00
CS	0716	
IP	003F	
SS	0710	
SP	003E	
BP	0000	
SI	0000	

0716:003F

0716:003E

0719F:	E2	226	o	LOOP 0113h
071A0:	EC	236	y	MOV BX, 00000h
071A1:	C3	195	f	MOV CX, 00014h
071A2:	90	144		MOV AH, [BX]
071A3:	90	144		CMP AH, 061h
071A4:	90	144		JB 03Eh
071A5:	90	144		CMP AH, 07Ah
071A6:	90	144		JNBE 03Eh
071A7:	90	144		AND AH, 0DFh
071A8:	90	144		MOV [BX], AH
071A9:	90	144		INC BX
071AA:	90	144		LOOP 02Dh
071AB:	90	144		RET
071AC:	90	144		NOP
071AD:	00	144		NOP

Random Access Memory

0714:0000

update

table

list

original source code

```

30 MOV [BX], AH
31 OTR02: INC BX
32 LOOP OTR0
33 ;
34 LEA BX, MEN
35 MOV CX, 20
36 OTR03: MOV AH, [BX]
37 CMP AH, 61h
38 JB OTR04
39 CMP AH, 7Ah
40 JA OTR04
41 AND AH, 11011111B
42 MOV [BX], AH
43 OTR04: INC BX
44 LOOP OTR03
45 RET
46
47 ALTERNAR-MAYUSCULASYMINU
48
49
50 END MAIN
51

```

0714:0000 43 41 4D 42 49 41 52 20-41 20 4D 49 4E 55 53 43 CAMBIAR A MINUSC  
0714:0010 55 4C 41 53 00 00 00 00-00 00 00 00 00 00 00 00 ULAS.....  
0714:0020 B8 14 07 8E D8 E8 05 00-B8 00 4C CD 21 BB 00 00 00 00 01.A!p+.0.L=!1..

Para terminar retorna a "MAIN" para con la interrupción 21h termina el programa con un mensaje de aviso.

## Código

```
01 PAGE 60,132
02 TITLE PROG6.EXE
03 .MODEL SMALL
04 ;
05 .STACK 64
06 .DATA
07 MEN DB 'CAMBIAR A MINUSCULAS'
08 ;
09 .CODE
10 MAIN PROC NEAR
11
12     MOV AX, @DATA
13     MOV DS, AX
14     CALL ALTERNAR-MAYUSCULASYMINUSCULAS
15     MOV AX, 4C00H
16     INT 21H
17
18 MAIN ENDP
19
20 ALTERNAR-MAYUSCULASYMINUSCULAS PROC NEAR
21
22     LEA BX, MEN
23     MOV CX, 20
24     OTR0: MOV AH, [BX]
25     CMP AH, 41H
26     JB OTR02
27     CMP AH, 5AH
28     JA OTR02
29     OR AH, 00100000B
30     MOV [BX], AH
31     OTR02: INC BX
32     LOOP OTR0
33 ;
34     LEA BX, MEN
35     MOV CX, 20
36     OTR03: MOV AH, [BX]
37     CMP AH, 61H
38     JB OTR04
39     CMP AH, 7AH
40     JA OTR04
41     AND AH, 11011111B
42     MOV [BX], AH
43     OTR04: INC BX
44     LOOP OTR03
45     RET
46
47 ALTERNAR-MAYUSCULASYMINUSCULAS ENDP
48
49
50 END MAIN
```

## Reflexión

En esta actividad vimos 2 instrucciones nuevas totalmente útiles e interesantes dado a que los ciclos repetitivos son una de las estructuras mas usadas en la programación para infinidad de implementaciones y LEA que nos ofrece una alternativa mas simple para encontrar las direcciones de memoria de nuestras vaciables sin usar MOV y offset.

## Actividad 5 – Parte 3

### Descripción

Explique el funcionamiento de la interrupción 10H en el siguiente programa.

### Desarrollo y Resultados

#### Reporte:

El programa inicia en un procedimiento llamado BEGIN que inicia el segmento de datos y llama otros 3 procedimientos para despues terminar el programa, primero llama a PANT0 donde se usa la interrupción 10H con 06h en AH para recorrer hacia arriba la pantalla donde con AL= 0 no recorre ninguna línea hacia arriba, con BH = 7 escribe 7 líneas vacías en el fondo de la ventana, con CX = 0 no cambia la esquina superior izquierda y con DX = 184Fh modifica la esquina inferior derecha.

```
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0

registers
H L
AX 06 00
BX 07 00
CX 00 00
DX 18 4F
CS F400
IP 0190
SS 0710
SP 0038
BP 0000
SI 0000
DI 0000
DS 0714
ES 0700

F400:0190 F4190: FF 255 RI BIOS DI
F4191: FF 255 RI INT 010h
F4192: CD 205 = IRET
F4193: 10 016 > ADD [BX + SI], AL
F4194: CF 207 < ADD [BX + SI], AL
F4195: 00 000 NI ADD [BX + SI], AL
F4196: 00 000 NI ADD [BX + SI], AL
F4197: 00 000 NI ADD [BX + SI], AL
F4198: 00 000 NI ADD BH, BH
F4199: 00 000 NI DEC BP
F419A: 00 000 NI ADC CL, BH
F419B: 00 000 NI ADD [BX + SI], AL
F419C: 00 000 NI ADD [BX + SI], AL
F419D: 00 000 NI ADD [BX + SI], AL
F419E: 00 000 NI ADD [BX + SI], AL
F419F: 00 000 NI ADD [BX + SI], AL
F41A0: FF 255 RI ADD BH, BH
F41A1: FF 255 RI ...

original source code
13 MOV AX, @DATA
14 MOV DS, AX
15 CALL PANT0
16 CALL CURSOR
17 CALL DESPL0
18 MOV AX, 4C00H
19 INT 21H
20 BEGIN ENDP
21
22 PANT0 PROC NEAR
23 MOV AX, 0600H
24 MOV BH, 7H
25 MOV CX, 0000H
26 MOV DX, 184FH
27 INT 10H
28 RET
29 PANT0 ENDP
30
31 CURSOR PROC NEAR
32 MOV AH, 02H
33 MOV BH, 00
34 MOV DX, 0502H
35 INT 10H
36 RET
37 CURSOR ENDP
38
39 DESPL0 PROC NEAR
40 MOV AH, 09H
41 LEA DX, ETIQ
42 INT 21H
43 RET
44 DESPL0 ENDP
```

Después llama al proceso CUSOR que con interrupción 10H con 02h en AH establece la posición del cursor con BH = 0 establece el número de la página y con DX = 0502h establece que se comenzara a escribir en la fila 5 y columna 2.

```
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0

registers
H L
AX 02 00
BX 00 00
CX 00 00
DX 05 02
CS F400
IP 0190
SS 0710
SP 0038
BP 0000
SI 0000
DI 0000
DS 0714
ES 0700

F400:0190 F4190: FF 255 RI BIOS DI
F4191: FF 255 RI INT 010h
F4192: CD 205 = IRET
F4193: 10 016 > ADD [BX + SI], AL
F4194: CF 207 < ADD [BX + SI], AL
F4195: 00 000 NI ADD [BX + SI], AL
F4196: 00 000 NI ADD [BX + SI], AL
F4197: 00 000 NI ADD [BX + SI], AL
F4198: 00 000 NI ADD BH, BH
F4199: 00 000 NI DEC BP
F419A: 00 000 NI ADC CL, BH
F419B: 00 000 NI ADD [BX + SI], AL
F419C: 00 000 NI ADD [BX + SI], AL
F419D: 00 000 NI ADD [BX + SI], AL
F419E: 00 000 NI ADD [BX + SI], AL
F419F: 00 000 NI ADD [BX + SI], AL
F41A0: FF 255 RI ADD BH, BH
F41A1: FF 255 RI ...

original source code
22 PANT0 PROC NEAR
23 MOV AX, 0600H
24 MOV BH, 7H
25 MOV CX, 0000H
26 MOV DX, 184FH
27 INT 10H
28 RET
29 PANT0 ENDP
30
31 CURSOR PROC NEAR
32 MOV AH, 02H
33 MOV BH, 00
34 MOV DX, 0502H
35 INT 10H
36 RET
37 CURSOR ENDP
38
39 DESPL0 PROC NEAR
40 MOV AH, 09H
41 LEA DX, ETIQ
42 INT 21H
43 RET
44 DESPL0 ENDP
```

Por último, llama al procedimiento llamado DESPLO que con la interrupción 21H y 09h en AH escribe una cadena de caracteres en la consola con LEA obtiene la dirección de memoria de la cadena previamente declarada en el segmento de datos "DESPLIEGUE DE VIDEO DIRECTO ", "\$" y la coloca en DX.

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	09	24
BX	00	00
CX	00	00
DX	00	00
CS	F400	
IP	0204	
SS	0710	
SP	0038	
BP	0000	

F400:0204

F4200: FF 255 RI- BIOS DI  
 F4201: FF 255 RI- INT 021h  
 F4202: CD 205 = IREI  
 F4203: 21 033 !  
 F4204: CF 207 !  
 F4205: 00 000 NI  
 F4206: 00 000 NI  
 F4207: 00 000 NI  
 F4208: 00 000 NI  
 F4209: 00 000 NI  
 F420A: 00 000 NI  
 F420B: 00 000 NI

original source code

```

35 INT 10H
36 RET
37 CURSOR ENDP
38 ;
39 DESPLO PROC NEAR
40 MOV AH, 09H
41 LEA DX, ETIQ
42 INT 21H
43 RET
44 DESPLO ENDP
45 ;
46 END BEGIN
47
48
  
```

emulator screen (80x25 chars)

DESPLIEGUE DE VIDEO DIRECTO

Así terminamos con la consola donde se puede leer la cadena en un punto centrado gracias a las modificaciones que se hicieron con la interrupción 10H.

emulator screen (80x25 chars)

DESPLIEGUE DE VIDEO DIRECTO

### Código:

```
01 PAGE 60,132
02 TITLE PROG7.EXE
03 .MODEL SMALL
04 .STACK 64
05 ;
06 -----
07 .DATA
08
09 ETIQ DB "DESPLIEGUE DE VIDEO DIRECTO ", "$"
10 ;
11 .CODE
12 BEGIN PROC FAR
13 MOV AX, @DATA
14 MOV DS, AX
15 CALL PANT0
16 CALL CURSOR
17 CALL DESPL0
18 MOV AX, 4C00H
19 INT 21H
20 BEGIN ENDP
21 ;
22 PANT0 PROC NEAR
23 MOV AX, 0600H
24 MOV BH, 7H
25 MOV CX, 0000H
26 MOV DX, 184FH
27 INT 10H
28 RET
29 PANT0 ENDP
30 ;
31 CURSOR PROC NEAR
32 MOV AH, 02H
33 MOV BH, 00
34 MOV DX, 0502H
35 INT 10H
36 RET
37 CURSOR ENDP
38 ;
39 DESPL0 PROC NEAR
40 MOV AH, 09H
41 LEA DX, ETIQ
42 INT 21H
43 RET
44 DESPL0 ENDP
45 ;
46 END BEGIN
```

### Reflexión

En esta actividad pudimos implementar la interrupción 10h usando solo 2 de sus funciones pero por actividades anteriores sabemos que esta se basa en su mayor parte en modificar la salida visual del programa aquí solo imprimimos una simple cadena, pero marca un inicio para hacer programas mas complejos en el futuro.