

Universidad Autónoma de Baja California

Ingeniero en computación



Organización y Arquitectura De Computadoras

Practica 9

Nombre Del Alumno: López Mercado Brayan

Matrícula: 1280838

Grupo: 551

Docente: José Isabel García Rocha

Fecha de entrega: 2 de noviembre del 2023

Objetivo

Seleccionar las instrucciones correctas en aplicaciones de sistemas basados en microprocesador mediante la distinción de su funcionamiento, de forma lógica y responsable.

Desarrollo

1- Cree un programa llamado *p9.asm* que realice lo siguiente:

Crear un programa en el cual el usuario ingrese el tamaño de un arreglo de números enteros sin signo y posteriormente los solicite en el programa, estos se guardaran en un arreglo en memoria en desorden para después ordenarlos de manera ascendente de izquierda a derecha.

El programa debe de contener las siguientes subrutinas:

1. ***CapturarDecimal: Subrutina que retorna en AX el valor capturado de 3 caracteres, ejemplo, 005 es un 5 en decimal, 123 es 123 en decimal, esta subrutina hace uso de la subrutina printDecimal.***

```
capturarDecimal:
    pushad
    mov eax,3
    mov ebx,0
    mov ecx,tmp
    mov edx,4
    int 0x80
    mov ecx,3
    call st2num
    popad
    ret

st2num:
    pushad
    xor ebx,ebx
    lea edi,tmp
    .next:
        movzx eax,byte[edi]
        inc edi
        sub al,'0'
        imul ebx,10
        add ebx,eax
        loop .next
    mov eax,ebx
    mov [tmp],eax
    popad
    ret
```

Figura 1: Subrutina para capturar decimales desde la terminal.

Descripción: La captura de caracteres de terminal se realiza por medio de la directiva 3 de NASM cuya cadena se guarda en tmp y con una longitud de 4 caracteres, el motivo de que la longitud de la cadena sea de 4 en lugar de 3 se debe a que al usar la terminal de Linux Mint para la captura es que al usar la tecla

ENTER para terminar la cadena esta agrega un carácter extra que no se muestra en terminal, lo cual causa errores al momento de la captura, para evitar ese problema se incrementó la longitud a 4, para posteriormente usar la interrupción 0x80 para leer la cadena de la terminal y luego se pasa a ECX un 3 para indicar la cantidad de dígitos que contendrá la cadena final por medio de la subrutina auxiliar st2num, esta subrutina tiene la finalidad pasar a números enteros la cadena capturada previamente, la manera de hacerlo es usando limpiando EBX y cargando a EDI la dirección efectiva de tmp (recordando que es donde se guardó la cadena), dentro de la etiqueta .next se mueve a EAX el primer byte que se contiene en EDI, luego se incrementa EDI para la siguiente iteración, debido a que se trabajan con bytes, se le resta al registro AL el carácter 0 de la tabla ASCII (lo pasa de carácter a numero), luego se multiplica por un 10 para obtener el equivalente decimal cuyo resultado se guarda en EBX y se realiza una suma con el contenido de EAX para la siguiente iteración, luego sé que hayan completado las 3 iteraciones se guardará el acumulado de EBX en EAX y luego se guardara en tmp como un número.

Nota: la instrucción imul es la única instrucción de multiplicación que es capaz de trabajar con 2 operandos en lugar de solo uno a comparación de mul, en este caso al usar (imul ebx,10) se multiplica el contenido de EBX por 10 decimal.

2. ***PrintDecimal: es una subrutina que imprime los valores en formato decimal, ejemplo, se ingresa en AX un 0x64 y se muestra en pantalla un 100 decimal como cadena.***

```
printDec:
    pushad
    mov bx,0xA
    xor edx,edx
    mov ecx,8
    mov esi,cadDec
    call limpiarCadena
    .convert:
        xor dx,dx
        div bx
        add dl,'0'
        mov [esi+ecx],dl
        loop .convert
    mov eax,4
    mov ebx,1
    mov ecx,cadDec
    mov edx,12
    int 0x80
    popad
    ret
```

Figura 2: Subrutina para imprimir decimales en la terminal.

Descripción: La manera de realizar la conversión a decimal es por medio de divisiones entre 10, cuyo residuo es un carácter de numero en decimal, la forma

en la que maneja en el código de la figura 2 es moviendo a EBX el número 10 (0xA en hexadecimal) y se realiza la operación XOR de EDX con EDX para eliminar cualquier bit que pueda afectar en la conversión, ECX contiene la cantidad de divisiones a realizar y ESI contiene la dirección de inicio de cadDec (localidad de memoria donde se guarda la conversión) y se realiza un llamado a la subrutina LimpiarCadena para eliminar los posibles caracteres basura que existan en la cadena, en la etiqueta .convert se realiza una limpieza de DX, esto se debe principalmente a que al realizar una división con un divisor de 16 bits el residuo se almacena en DL, luego se realiza una suma de '0' a DL, lo que al final lo terminara convirtiéndolo en un carácter que es posible de mostrar en terminal y este se guardará en la dirección contenida de ESI; una vez completadas las 8 iteraciones se imprimirán en terminal por medio de la directiva 4 de NASM, la cadena a mostrar en terminal se almacena en cadDec, el motivo de que la longitud de la cadena sea 12 se obtuvo por medio de una heurística, en pocas palabras, no se mostraba toda la cadena en terminal y se muestra en terminal por medio de la interrupción 0x80.

3. LimpiarCadena: Subrutina utilizada para limpiar la cadena utilizada por printDecimal y así no mostrar en la terminal caracteres basura.

```
limpiarCadena:
    push esi
    mov esi,[cadDec]
    xor esi,esi
    mov [cadDec],esi
    pop esi
    ret
```

Figura 3: Subrutina para limpiar la cadena utilizada por printDec.

Descripción: Se mueve a la pila una copia de ESI, de esta manera no se alteran los contenidos de ESI al salir de la subrutina; en ESI se mueve el contenido de cadDec para realizar la operación XOR ESI, ESI lo que dará como resultado 0 y ese resultado se guardará en cadDec y luego se saca el contenido original de ESI de la pila y se sale de la subrutina.

4. CapturarArreglo: Subrutina que utiliza la subrutina CapturarDecimal para guardar un arreglo en una posición de memoria indicada por el buffer.

```
capturarArreglo:
    pushad
    mov edx,sizeTag
    call puts
    call capturarDecimal
    mov eax,[tmp]
    mov [size],eax
    mov ecx,[size]
    mov edi,0x0
    lea ebx,array
    .capture:
        mov edx,captureTag
        call puts
        call capturarDecimal
        mov eax,[tmp]
        mov [ebx+4*edi],eax
        inc edi
        loop .capture
    popad
    ret
```

Figura 4: Subrutina para capturar arreglos en memoria.

Descripción: primero se muestra en terminal un mensaje que indica al usuario que debe de capturar la cantidad de elementos que contendrá el arreglo y este mensaje se muestra en pantalla por medio de la subrutina puts y se llama a CapturarDecimal para realizar la captura, una vez capturada la longitud, se mueve a EAX el contenido de tmp (la longitud de la cadena) y esta se guarda en size y luego se mueve el contenido de la variable size a ECX, el cual es el contador del ciclo de captura de elementos, en EDI, se contiene el índice para moverse entre los elementos del arreglo, en este caso se manejan números de 32 bits, y por eso se maneja una escala de 4, EBX contiene la dirección efectiva del arreglo (en este caso se le llamo array), dentro de la etiqueta .capture se muestra al usuario el mensaje de que capture un numero de 3 dígitos, para ello se mueve el contenido de tmp a EAX y se realiza una copia en la dirección contenida por EBX con un offset indicado por el valor de EDI multiplicado por 4, luego se incrementa EDI para la siguiente iteración; una vez que han completado todas la iteraciones se sale de la subrutina.

5. **ImprimirArreglo:** Subrutina que muestra en terminal de manera decimal el arreglo contenido en memoria.

```
imprimirArreglo:
    pushad
    mov ebx,array
    mov ecx,[size]
    mov edi,0x0
    .print:
        mov eax,[ebx+edi*4]
        call printDec
        call new_line
        inc edi
        loop .print
    popad
    ret
```

Figura 5: Subrutina para imprimir arreglos en la terminal.

Descripción: la manera en la que funciona esta subrutina es similar a capturar arreglo, EBX contiene la dirección de inicio del arreglo y ECX contiene la cantidad de elementos a imprimir, EDI es un contador inicializado en cero el cual tiene la finalidad de ser el offset dentro del ciclo, dentro de la etiqueta .print se mueve a EAX el valor contenido por EBX+4*EDI (en este caso los números son de 32 bits, por eso se utiliza una escala de 4) y se muestran en pantalla por medio de la subrutina printDec y se realiza un salto de línea por simple estética, luego se incrementa EDI para la siguiente iteración; una vez completada las iteraciones se sale de la subrutina por medio de la instrucción ret.

6. **OrdenarArreglo:** Subrutina que ordena de manera ascendente de izquierda a derecha el arreglo contenido en memoria.

```
ordenarArreglo:
    pushad
    mov dword[i],1
    .outLoop:
        lea esi,array
        mov edx,[size]
        .check:
            mov eax,[esi]
            add esi,0x4
            mov ebx,[esi]
            cmp eax,ebx
            jb .next
        .exchange:
            mov [tmp],eax
            mov eax,ebx
            mov ebx,[tmp]
            mov [esi],ebx
            sub esi,0x4
            mov [esi],eax
            add esi,0x4
        .next:
            dec edx
            cmp edx,[i]
            jne .check
    inc dword[i]
    mov ecx,[i]
    cmp ecx,[size]
    jne .outLoop
    mov edx,orderedTag
    call puts
    call imprimirArreglo
    popad
    ret
```

Figura 5: Subrutina para ordenar arreglo por medio de ordenamiento de burbuja.

Descripción: La subrutina OrdenarArreglo hace uso del algoritmo “Bubble Sort” para ordenar el arreglo, primero se mueve a la variable i el número 1, la variable i funciona como el contador del ciclo FOR externo que realiza el ordenamiento, dentro de la etiqueta outLoop se carga en ESI la dirección efectiva del arreglo a ordenar y EDX contiene el tamaño del arreglo de igual manera EDX es el contador del ciclo FOR interno, en la etiqueta check se mueve a EAX el elemento contenido por ESI, luego se incrementa ESI en 4 para guardar el elemento contenido en esa localidad de memoria en EBX (en pocas palabras, EBX contiene el siguiente elemento del arreglo) se realiza una comparación entre EAX y EBX, en caso de que el resultado de esa resta menor que 0 (EBX es mayor que EAX), se realiza un salto a la etiqueta next para realizar un decremento a EDX y luego compáralo con el valor de i en caso de que estos no sean iguales se realizara un salto a la etiqueta check, se saldrá de del ciclo interno hasta que EDX e i sean iguales, en caso de que la comparación de EAX y EBX sea mayor que cero, se entrara en la etiqueta Exchange, la manera de realizar el intercambio es por medio de una

variable auxiliar, este caso se está usando tmp para ello; primero se mueve a tmp el contenido de EAX, el contenido de EBX se mueve a EAX y lo que se tiene en tmp se mueve a EBX, después de estos movimientos se guardara en la dirección contenida por ESI el valor que era mayor (EBX contiene ese valor) y el valor de EAX se guarda en la posición anterior, luego se le suma a ESI un 0x4 para dejarlo en la posición siguiente (la dirección donde se guardó EBX) y se procederá realizar las instrucciones de la instrucción next, una vez que se haya completado el ciclo interno se incrementara i en 1 y ese resultado se moverá a ECX, se realizara una comparación entre ECX y el contenido de la variable size, en caso de que estos no sean iguales se volverá a entrar al ciclo externo, en caso de que ECX y size sean iguales se mostrara en terminal el mensaje indicando como se ve el arreglo ordenado y muestra en terminal por medio de la subrutina puts, y luego se imprimen los valores del arreglo por medio de la subrutina ImprimirArreglo.

El programa se debe probar con el siguiente arreglo:

57,53,21,37,17,36,22,3,44,97,89,26,31,4,8

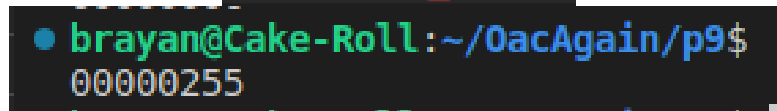
La salida en terminal debe dar lo siguiente:

3,8,17,21,22,26,31,36,37,44,47,53,57,89,97

Pruebas

PrintDecimal:

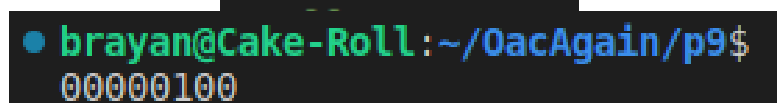
```
mov eax,0xFF
call printDec
call new_line
```



● brayan@Cake-Roll:~/0acAgain/p9\$.
00000255

Figura 6: Código utilizado y salida en terminal.

```
mov eax,0x64
call printDec
call new_line
```



● brayan@Cake-Roll:~/0acAgain/p9\$.
00000100

Figura 7: Código utilizado y salida en terminal.


```

mov eax,1010b
call printDec
call new_line
● brayan@Cake-Roll:~/OacAgain/p9$ ./
00000010

```

Figura 8: Código utilizado y salida en terminal.

```

mov eax,12
call printDec
call new_line
● brayan@Cake-Roll:~/OacAgain/p9$ ./
00000012

```

Figura 9: Código utilizado y salida en terminal.

Captura y Ordenamiento Del Arreglo

```

● brayan@Cake-Roll:~/OacAgain/p9$ ./run_all.sh
Ingresa Tamaño Del Arreglo: 005
Ingresa Valor Decimal De 3 Dígitos: 099
Ingresa Valor Decimal De 3 Dígitos: 054
Ingresa Valor Decimal De 3 Dígitos: 001
Ingresa Valor Decimal De 3 Dígitos: 006
Ingresa Valor Decimal De 3 Dígitos: 000
El Arreglo Ordenado Es:
00000000
00000001
00000006
00000054
00000099

```

Figura 10: Captura y ordenamiento de un arreglo de 5 valores.

```

● brayan@Cake-Roll:~/OacAgain/p9$ ./run_all.sh
Ingresa Tamaño Del Arreglo: 015
Ingresa Valor Decimal De 3 Dígitos: 057
Ingresa Valor Decimal De 3 Dígitos: 053
Ingresa Valor Decimal De 3 Dígitos: 021
Ingresa Valor Decimal De 3 Dígitos: 037
Ingresa Valor Decimal De 3 Dígitos: 017
Ingresa Valor Decimal De 3 Dígitos: 036
Ingresa Valor Decimal De 3 Dígitos: 022
Ingresa Valor Decimal De 3 Dígitos: 003
Ingresa Valor Decimal De 3 Dígitos: 044
Ingresa Valor Decimal De 3 Dígitos: 097
Ingresa Valor Decimal De 3 Dígitos: 089
Ingresa Valor Decimal De 3 Dígitos: 026
Ingresa Valor Decimal De 3 Dígitos: 031
Ingresa Valor Decimal De 3 Dígitos: 047
Ingresa Valor Decimal De 3 Dígitos: 008
El Arreglo Ordenado Es:
00000003
00000008
00000017
00000021
00000022
00000026
00000031
00000036
00000037
00000044
00000047
00000053
00000057
00000089
00000097

```

Figura 11: Captura y ordenamiento del arreglo propuesto en la práctica.

Script De Bash Utilizado

```

$ run_all.sh
1  nasm -f elf p9.asm
2  ld -m elf_i386 -s -o p9 p9.o libpc_io.a
3  ./p9

```

Figura 12: Script para la ejecución del código.

Link De GitHub Con Código Completo

https://github.com/BrayanLMercado/OAC_Practica9_v2.git

Código Completo

```
%include "pc_io.inc"
section .data
    NL: db 13,10
    NL_L: equ $-NL
    sizeTag: db "Ingresa Tamaño Del Arreglo: ",0
    captureTag: db "Ingresa Valor Decimal De 3 Dígitos: ",0
    orderedTag: db "El Arreglo Ordenado Es: ",10,0

section .bss
    size resb 4
    num resb 3
    array resw 256
    i resb 4
    j resb 4
    tmp resb 3
    cad resb 256
    cadDec resb 256

section .text
global _start:
_start: mov esi, cad
        call capturarArreglo
        call ordenarArreglo
        ;Exit Call
        mov eax, 1
        mov ebx, 0
        int 0x80

capturarDecimal:
    pushad
    mov eax, 3
    mov ebx, 0
    mov ecx, tmp
    mov edx, 4
    int 0x80
    mov ecx, 3
    call st2num
    popad
    ret

st2num:
    pushad
    xor ebx, ebx
    lea edi, tmp
.next:
```

```

    movzx eax,byte[edi]
    inc edi
    sub al,'0'
    imul ebx,10
    add ebx,eax
    loop .next
mov eax,ebx
mov [tmp],eax
popad
ret

```

limpiarCadena:

```

push esi
mov esi,[cadDec]
xor esi,esi
mov [cadDec],esi
pop esi
ret

```

capturarArreglo:

```

pushad
mov edx,sizeTag
call puts
call capturarDecimal
mov eax,[tmp]
mov [size],eax
mov ecx,[size]
mov edi,0x0
lea ebx,array
.capture:
    mov edx,captureTag
    call puts
    call capturarDecimal
    mov eax,[tmp]
    mov [ebx+4*edi],eax
    inc edi
    loop .capture
popad
ret

```

imprimirArreglo:

```

pushad
mov ebx,array
mov ecx,[size]
mov edi,0x0
.print:

```

```
    mov eax,[ebx+edi*4]
    call printDec
    call new_line
    inc edi
    loop .print
popad
ret
```

```
printDec:
    pushad
    mov bx,0xA
    xor edx,edx
    mov ecx,8
    mov esi,cadDec
    call limpiarCadena
.convert:
    xor dx,dx
    div bx
    add dl,'0'
    mov [esi+ecx],dl
    loop .convert
    mov eax,4
    mov ebx,1
    mov ecx,cadDec
    mov edx,12
    int 0x80
popad
ret
```

```
ordenarArreglo:
    pushad
    mov dword[i],1
.outLoop:
    lea esi,array
    mov edx,[size]
.check:
    mov eax,[esi]
    add esi,0x4
    mov ebx,[esi]
    cmp eax,ebx
    jb .next
.exchange:
    mov [tmp],eax
    mov eax,ebx
    mov ebx,[tmp]
    mov [esi],ebx
```

```

        sub esi,0x4
        mov [esi],eax
        add esi,0x4
.next:
        dec edx
        cmp edx,[i]
        jne .check
inc dword[i]
mov ecx,[i]
cmp ecx,[size]
jne .outLoop
mov edx,orderedTag
call puts
call imprimirArreglo
popad
ret

```

;Subrutinas Auxiliares

printBin:

```

pushad
mov edi,eax
mov ecx,32
.cycle:
xor al,al
shl edi,1
adc al,'0'
call putchar
loop .cycle
popad
ret

```

new_line:

```

pushad
mov eax, 4
mov ebx, 1
mov ecx, NL
mov edx, NL_L
int 0x80
popad
ret

```

printHex:

```

pushad
mov edx, eax
mov ebx, 0fh
mov cl, 28

```

```

.nxt: shr eax,cl
.msk: and eax,ebx
cmp al, 9
jbe .menor
add al,7
.menor:add al,'0'
mov byte [esi],al
inc esi
mov eax, edx
cmp cl, 0
je .print
sub cl, 4
cmp cl, 0
ja .nxt
je .msk
.print: mov eax, 4
mov ebx, 1
sub esi, 8
mov ecx, esi
mov edx, 8
int 80h
popad
ret

```

clearReg:

```

xor eax,eax ; Limpieza De Registros
xor ebx,ebx
xor ecx,ecx
xor edx,edx
ret

```

Conclusiones y Comentarios

- Se aprendió el cómo se realiza una captura de numero en lenguaje ensamblador, a pesar de que este no cuenta con una manera directa de realizar la conversión de cadena a número entero o viceversa.
- El intentar implementar el ordenamiento de burbuja en ensamblador es un tanto abstracto por la forma en la que se deben de colocar las etiquetas y los saltos para evitar segmentaciones y bucles infinitos.
- En caso de que desee visualizar el código con colores, le recomiendo que use el link de GitHub que se anexo a la práctica, así se puede evitar daños en los ojos a causa del fondo blanco del documento.

Dificultades En El Desarrollo

- El convertir la cadena a números trajo consigo un buen dolor de cabeza a causa de forma en la que se guarda la cadena en memoria, la forma de resolver fue pasar al menos unas 2 horas buscando en internet una manera de realizar la conversión sin que esta terminara con caracteres basura y en el orden correcto.
- Para el ordenamiento de burbuja se tuvo que buscar un pseudocódigo lo suficientemente genérico para poder implementarlo en ensamblador, lo cual tomo al menos unas 3 horas de búsqueda en internet.
- Para imprimir en decimal se tuvo que ver que versión de la instrucción div se tenía que usar para poner manejar el residuo de manera sencilla, el limpiar el registro que contenía el residuo permitió evitar la excepción de coma flotante.

Referencias

Guide to x86 Assembly. (s/f). Recuperado el 2 de noviembre de 2023, de <https://www.cs.virginia.edu/~evans/cs216/guides/x86.html>