Universidad Autónoma de Baja California
Ingeniero en computación

Organización y Arquitectura De Computadoras

Practica 10

Nombre Del Alumno: López Mercado Brayan

Matrícula: 1280838

Grupo: 551

Docente: José Isabel García Rocha

Fecha de entrega: 9 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 p10.asm que realice lo siguiente:**
Crear un programa en cual el usuario ingrese una cadena de caracteres numéricos terminados con el símbolo *, se llame a puts para mostrar la cadena en pantalla, se llame a Atoi para convertir la cadena a un número decimal, se llame a printHex para mostrar el valor de EAX y posteriormente se muestre en pantalla con printDec.

Ejemplo:
*Ingresa La Cadena: 12345**
Salida en pantalla de puts: 12345
Salida en pantalla de printHex: 00003039
Salida en pantalla de printDec: 12345

El programa debe de tener las siguientes subrutinas:

a) **Gets:** Subrutina que captura en memoria direccionada por ESI una cadena de caracteres y esta termina de almacenarlos hasta que se ingrese un símbolo '*', al final de la cadena almacenada se debe de colocar un 0.

```
gets:
    push ebx
    push ecx
    push edx
    push edi
    lea esi,tmp
    mov eax,3
    mov ebx,0
    mov ecx,esi
    mov edx,12
    int 0x80
    call len
    pop edi
    pop edx
    pop ecx
    pop ebx
    ret
```

Figura 1: Subrutina *gets* para captura de cadenas desde terminal.

Descripción: se realiza una copia de los registros EBX, ECX, EDX, EDI en pila para no alterar los registros al momento de salir de la subrutina, posteriormente se carga en ESI la dirección de tmp y se procede a realizar la captura de la cadena desde terminal por medio de la directiva 3 de NASM, la cual se guardará en la dirección contenida por ESI y esta tendrá una longitud máxima de 12 caracteres, esto se decio de esta forma tomando en cuenta de que número máximo a convertir es de 10 dígitos y los caracteres 11 y 12

representan al asterisco y al carácter que se agrega al presionar la tecla ENTER respectivamente; y se realiza la lectura de terminal al usar la interrupción 0x80 y se llama a la función len para obtener la longitud de la cadena, la cual es necesaria para que la subrutina Atoi funcione y finalmente se restauran los registros que se habían enviado a la pila y se sale de la subrutina.

```
len:
    pushad
    xor ecx,ecx
    lea edi,tmp
    lea esi,stLen
    .check:
        cmp byte[edi],'*'
        jz .stlen
        inc ecx
        inc edi
        jmp .check
    .stlen:
        mov byte[edi],0
        mov [esi],ecx
    popad
    ret
```

Figura 2: Subrutina auxiliar *len* para cálculo de longitud de la cadena capturada desde terminal.

Descripción: se realiza la operación XOR en ECX para tener únicamente ceros, ya que ECX se estará usando como acumulador, EDI y ESI contienen las direcciones de tmp y stLen respectivamente, dentro de la etiqueta check se realiza una comparación de un byte de la cadena contenida en la dirección de EDI con el carácter '*', en caso de que sean iguales se realizara un salto a la etiqueta stLen y se moverá a ese byte en específico un cero, y se moverá el valor de ECX a la dirección contenida en ESI y se saldrá de la subrutina, en caso de que la comparación no sea cero, se incrementara ECX y EDI, y se realizara un salto incondicional a la etiqueta check para realizar la comparación con el siguiente Byte, así hasta encontrar el carácter '*';.

b) **Atoi:** Subrutina que convierte una cadena de caracteres que representan un número decimal en un numero decimal, recibe en ESI la dirección de la cadena y lo retorna en EAX, el máximo valor decimal a convertir es 4,294,967,295.

```
atoi:
    push ebx
    push ecx
    push edx
    push edi
    push esi
    xor ebx,ebx
    lea esi,tmp
    .next:
        movzx eax,byte[esi]
        inc esi
        sub al,'0'
        imul ebx,10
        add ebx,eax
        loop .next
    mov eax,ebx
    lea edx,num
    mov [edx],eax
    pop esi
    pop edi
    pop edx
    pop ecx
    pop ebx
    ret
```

Figura 3: Subrutina Atoi para la conversión de cadena a números.

Descripción: Esta subrutina funciona de una manera similar a la subrutina st2num de la practica anterior con la diferencia principal de que esta permite convertir cadenas de tamaño variable donde ECX es la longitud de la cadena. La forma en la que funciona la subrutina es realizando una copia de los registros que no se desean alterar en este caso son EBX, ECX, EDX, EDI y ESI; se realiza una limpieza del registro EBX para eliminar cualquier bit que pueda afectar a la conversión, ESI contiene la dirección de tmp, que es donde se guardó la cadena capturada, dentro de la etiqueta next se copia a EAX un byte de la cadena contenida por ESI por medio de la instrucción MOVZX y se incrementa ESI para la siguiente iteración, en este caso al tratarse de un byte este contiene en AL y se le realiza una resta del carácter ‘0’, lo que esto hace es pasarlo de carácter a número, luego se multiplicara el contenido de EBX por 10 y este mismo se guardara en EBX, y ese resultado se le sumara el contenido EAX, y este se guardara en EBX, una vez que hayan completado todas la iteraciones se moverá el contenido de EBX a EAX este contenido será la cadena convertida en número decimal, debido a que se solicitó utilizar registros en lugar de memoria directa, se carga en EDX la dirección de num, y posteriormente se copia el contenido de EAX a la dirección contenida en EDX y se restauran todos los registros que habían sido enviados a la pila y se sale de la función.

c) **PrintDec:** Esta subrutina debe ser modificada de manera que sea capaz de mostrar en terminal un número máximo de 4,294,967,295.

```
printDec:
    pushad
    mov ebx,0xA
    xor edx,edx
    mov ecx,10
    mov esi,cadDec
    call limpiarCadena
    .convert:
        xor edx,edx
        div ebx
        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 4: Subrutina printDec modificada para convertir números de hasta 32 bits.

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 numero 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 EDX, esto se debe principalmente a que al realizar una división con un divisor de 32 bits el residuo se almacena en EDX aunque al ser un residuo relativamente pequeño este se contiene 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 10 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.

Nota: Las únicas modificaciones que se realizaron respecto a la versión anterior fue incrementar ECX a 10, realizar la operación XOR EDX, EDX en lugar de XOR DX, DX; y el divisor ahora es de 32 bits en lugar de 16 bits.

Pruebas

Prueba 1:

```
brayan@Cake-Roll:~/OacAgain/p10$ ./run_all.sh
Ingresa Una Cadena: 445*
Salida En Terminal De Puts: 445
Salida En Terminal De PrintHex: 000001BD
Salida En Terminal De PrintDec: 0000000445
```

Figura 5: Prueba del programa con la cadena 445*.

Prueba 2:

```
brayan@Cake-Roll:~/OacAgain/p10$ ./run_all.sh
Ingresa Una Cadena: 4294967295*
Salida En Terminal De Puts: 4294967295
Salida En Terminal De PrintHex: FFFFFFFF
Salida En Terminal De PrintDec: 4294967295
```

Figura 6: Prueba del programa con la cadena máxima que es posible capturar.

Prueba 3:

```
brayan@Cake-Roll:~/OacAgain/p10$ ./run_all.sh
Ingresa Una Cadena: 78964*
Salida En Terminal De Puts: 78964
Salida En Terminal De PrintHex: 00013474
Salida En Terminal De PrintDec: 0000078964
```

Figura 7: Prueba del programa con la cadena 78964*.

Prueba 4:

```
brayan@Cake-Roll:~/OacAgain/p10$ ./run_all.sh
Ingresa Una Cadena: 77766695*
Salida En Terminal De Puts: 77766695
Salida En Terminal De PrintHex: 04A2A027
Salida En Terminal De PrintDec: 0077766695
```

Figura 8: Prueba del programa con la cadena 77766695*.

Script De Bash Utilizado

```
$ run_all.sh
1  nasm -f elf p10.asm
2  ld -m elf_i386 -s -o p10 p10.o libpc_io.a
3  ./p10
```

Figura 9: Script de Bash para la ejecución del programa.

Link De GitHub Con Código Completo

https://github.com/BrayanLMercado/OAC_Practica10_v2.git

Código Completo

```
%include "pc_io.inc"
section .data
   NL: db  13,10
   NL_L:    equ $-NL
   captureTag: db "Ingresa Una Cadena: ",0
```

```
    outputPuts: db "Salida En Terminal De Puts: ",0
    outputHex: db "Salida En Terminal De PrintHex: ",0
    outputDec: db "Salida En Terminal De PrintDec: ",0

section .bss
    tmp resb 12
    stLen resb 4
    num resb 4
    cad resb 256
    cadDec resb 256

section .text
global _start:
    _start:
    ;Captura De La Cadena
    mov edx,captureTag
    call puts
    call gets

    ; Salida Con Puts
    mov edx,outputPuts
    call puts
    mov edx,esi
    call puts
    call new_line

    ;Llamada a Atoi (st2Num con otro nombre)
    mov ecx,[stLen]
    call atoi
    ;Salida Con PrintHex
    mov edx,outputHex
    call puts
    call printHex
    call new_line

    ;Salida Con PrintDec
    mov edx,outputDec
    call puts
    call printDec
    call new_line

    ;Exit Call
    mov eax,1
    mov ebx,0
    int 0x80
```

```
gets:
    push ebx
    push ecx
    push edx
    push edi
    lea esi,tmp
    mov eax,3
    mov ebx,0
    mov ecx,esi
    mov edx,12
    int 0x80
    call len
    pop edi
    pop edx
    pop ecx
    pop ebx
    ret

len:
    pushad
    xor ecx,ecx
    lea edi,tmp
    lea esi,stLen
    .check:
        cmp byte[edi],'*'
        jz .stlen
        inc ecx
        inc edi
        jmp .check
    .stlen:
        mov byte[edi],0
        mov [esi],ecx
    popad
    ret

atoi:
    push ebx
    push ecx
    push edx
    push edi
    push esi
    xor ebx,ebx
    lea esi,tmp
    .next:
        movzx eax,byte[esi]
        inc esi
```

```
        sub al,'0'
        imul ebx,10
        add ebx,eax
        loop .next
    mov eax,ebx
    lea edx,num
    mov [edx],eax
    pop esi
    pop edi
    pop edx
    pop ecx
    pop ebx
    ret

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

printDec:
    pushad
    mov ebx,0xA
    xor edx,edx
    mov ecx,10
    mov esi,cadDec
    call limpiarCadena
    .convert:
        xor edx,edx
        div ebx
        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
```

```
;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
    lea esi,cad
    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

- Durante el transcurso de la práctica se aprendió mas acerca sobre el cómo se puede crear código que sea capaz de convertir cadenas de texto a números con una longitud variable, aunque el detalle del asterisco es un tanto innecesario en Linux, ya que se podría utilizar en carácter que se introduce al presionar el ENTER.

- Aunque la práctica aparentaba ser larga, dos de las subrutinas requeridas se tuvieron que realizar de manera indirecta en la practica 9, por ejemplo, Atoi es la misma subrutina que st2num, solo que se tuvo que cambiar el nombre para que cumpliera con el requisito del nombre.

- 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.

- En el caso de la subrutina printDec estaba tan bien diseña que solo se le tuvieron que modificar pocas líneas para que fuese capaz de trabajar con 32 bits, que da la casualidad que el número 0xFFFFFFFF es el 4,294,967,295 decimal.

Dificultades En El Desarrollo

- La mayor dificultad que existió durante la práctica fue el hecho de saber la longitud de la cadena que se capturaba, para ello se creó la subrutina len para realizar ese cálculo, una vez que se pensó el cómo realizar ese cálculo se realizaron varias pruebas hasta que se obtuvo el cálculo de manera correcta, una vez resuelto ese problema la subrutina Atoi ya funcionaba de manera correcta, ya que esta es dependiente del tamaño de la cadena a convertir.

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