UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE ORIENTE
CARRERAS DE INGENIERÍA
ING. EN CIENCIAS Y SISTEMAS
ING. FRANCISCO ARDON
ARQUITECTURA Y ENSAMBLADORES 1

DOCUMENTACIÓN CALCULADORA 32 BITS ASSEMBLER

MIGUEL SEBASTIAN VÁSQUEZ CABRERA
CARNÉ 201843279
MIÉRCOLES, 21 DE DICIEMBRE DE 2022

INTRODUCCIÓN

El código assembler inicio como uno de los primeros lenguajes de bajo nivel, esto para lograr adaptar nuestro lenguaje a un lenguaje maquina, teniendo en cuenta que este iba incrementando su funcionamiento con cada actualización en el mercado de los bits, recorriendo de 81 16 y 32 bits.
Siendo un aprendizaje bastante funcional el como una computadora funcionaba en el pasado, esto con el objetivo de dar un paso frente a como funcionaba la memoria cache, buffers y las pocas variables a las que el usuario tenia a disposición, debía ser bastante inteligente en lo que hacia y ser lo mas eficiente posible, ya que los recursos eran muy limitados.

OBJETIVOS

Determinar la manera eficiente una calculadora simple de cinco funciones en lenguaje ensamblador, utilizando la herramienta del emulador proporcionado por tutorialspoint.

Analizar todos los medios de un sistema ensamblador de 32 bits, para formular un resultado en el que el emulador pueda compilar sin ninguna interrupción alarme .

Documentar cada paso en el que se utilizo el lenguaje ensamblador, teniendo en cuenta cuales son algunos problemas frecuentes que se trabajo en el paso del proyecto.

DOCUMENTACIÓN

Para empezar, se debe analizar que el lenguaje a trabajar es el lenguaje ensamblador de 32 bits, teniendo una clara diferencia con los de 8 y 16 bits, siendo que en este las variables y los registros tienen una diferencia en su uso, las interrupciones a veces pueden cambiar en pro de mejorar la programacion del usuario.

Segundo paso, es el utilizar el medio de emulación que provee la pagina tutorialpoint mas específicamente (https://www.tutorialspoint.com/compile_assembly_online.php), se recomienda iniciar secion con la cuenta de Google, ya que esta nos provee guardar nuestro progreso.

Si se tiene mas consideración de como es que funciona el codigo se puede observar todos los registros en el Github personal (https://github.com/MiguelVasquez99/Arquitectura1-Diciembre2022).

Asi que se empezara a explicar cuales significan los códigos.

```
1  SYS_INIT_MODULE equ 0x80
2  SYS_EXIT equ 0x01
3  SYS_WRITE equ 0x04
4
```

Para iniciar se debe de presentar a sistema algunas interrupciones del sistema, la cual significan iniciar el modulo, salir y sobre escribir.

La parte inicial muestra como es que el sistema hace los moviemientos del mensaje que sera mostrado despues, usando los registros edx y ecx, usa los registros de ebx y eax para inicializar.

Segunda forma utiliza un buffer que estara encargado de guardar un valor para luego ser comparado con lo que el usuario ingresa en el teclado, en este caso un menu en el que se usara 1 al 5, para datar las funciones de cada uno, siempre utiliznado la interrupcion 0x80 por si el caso el usuario pusiera algun carácter fuera del rango esperado.

```
9  _start:
10
11      mov edx, len
12      mov ecx, msg
13      mov ebx, 1
14      mov eax, 4
15      int 0x80
16      mov edx, 2
17      mov ecx, buf
18      mov eax, 3
19      mov ebx, 0
20      int 0x80
21      cmp byte[buf], "1"
22      je suma
23      cmp byte[buf], "2"
24      je resta
25      cmp byte[buf], "3"
26      je multi
27      cmp byte[buf], "4"
28      je division
29      cmp byte [buf], "5"
30      mov esi, 2
31      mov ecx, 5
32      je potencia
33      int 0x80
34
```

```
33 suma:
34     mov edx, len_suma
35     mov ecx, msg_suma
36     mov eax, 4
37     mov ebx, 1
38     int 0x80
39     mov eax, 4
40     mov ebx, 7
41     add eax, ebx
42     aam
43     add eax, 3030h
44     mov ebp, esp
45     sub esp, 2
46     mov [esp], byte ah
47     mov [esp+1], byte al
48     mov ecx, esp
49     mov edx, 2
50     mov ebx, 1
51     mov eax, 4
52     int 0x80
53     mov esp, ebp
54
55     jmp _start
```

SUMA

Para entenderla suma se debe regresar a utilizar un mensaje con los registros edx y ecx, se vuelven a rescribir eax y ebx por la razon de que seran sumados con la funcion add que pide 2 registros para funcionar y devolver la sumatoria de estos, Aam tiene la funcion de separar esos numeros, add eax y su comando abajo es para volverlos a unir. En la Stack esp, se metera el numero mas grande en ah y al el segundo valor, ebx si existe algun error, eax, para escribir, y esp es para regresar a que el puntero apunte a la stack.

De ultimo se utiliza un jmp para hacer un while con el menu, y que el usuario no pueda salir si no escribe un carácter diferente al rango esperado.

RESTA

Aunque este tiene mucha estructura de la suma, siguiendo los pasos del mensaje, registros inicializados, que guarden algun dato, el aam se separa, se guarda los punteros, se resta dos para que el stack lo pueda obtener, en dos espacios de al que esta vacio se pasa, algunos registros escriben y por ultimo se regresa el puntero del mp al stack.

La unica diferencia radica en la funcion que se usa para determinar los numeros, siendo antes add en la linea 41, y en esta linea 65 se utiliza la funcion sub que es resta.

```
57 resta:
58     mov edx, len_suma
59     mov ecx, msg_suma
60     mov eax, 4
61     mov ebx, 1
62     int 0x80
63     mov eax, 8
64     mov ebx, 2
65     sub eax, ebx
66     aam
67     add eax, 3030h
68     mov ebp, esp
69     sub esp, 2
70     mov [esp], byte ah
71     mov [esp+1], byte al
72     mov ecx, esp
73     mov edx, 2
74     mov ebx, 1
75     mov eax, 4
76     int 0x80
77     mov esp, ebp
78
79     jmp _start
```

```
81  multi:
82      mov edx, len_multi
83      mov ecx, msg_multi
84      mov eax, 3
85      mov ebx, 8
86      mul ebx
87      aam
88      add eax, 3030h
89      mov ebp, esp
90      sub esp, 2
91      mov [esp], byte ah
92      mov [esp+1], byte al
93      mov ecx, esp
94      mov edx, 2
95      mov ebx, 1
96      mov eax, 4
97      int 0x80
98      mov esp, ebp
99
100     jmp _start
```

MULTIPLICACIÓN

Este al ser muy parecido a los otros, tienen la misma funcion de los registros, siempre pensando en que estos registros capten numeros, escriban, revisen y arreglen algun error si es que existe alguno.

La unica diferencia que existe entre los otros es que este tiene la particularidad de la linea 86 que seria mul.

DIVISIÓN

Al ser una estructura similar, todos los valores ya antes percibidos tienen su parte asignada con la variacion que esta funciona con numeros diferentes 40 y 2 pero con la particularidad

```
102 division:
103     mov edx, len_div
104     mov ecx, msg_div
105     mov eax, 4
106     mov ebx, 1
107     int 0x80
108     mov eax, 40
109     mov ebx, 2
110     mov edx, 0
111     div ebx
112     aam
113     add eax, 3030h
114     mov ebp, esp
115     sub esp, 2
116     mov [esp], byte ah
117     mov [esp+1], byte al
118     mov ecx, esp
119     mov edx, 2
120     mov ebx, 1
121     mov eax, 4
122     int 0x80
123
124     jmp _start
```

```
131 potencia:
132     ;mov edi, len_pot
133     ;mov ecx, msg_pot
134     add esi, esi
135     dec ecx
136     cmp ecx, 0
137     jg potencia
138     mov eax, 1
139     mul esi
140     aam
141     add eax, 3030h
142     mov ebp, esp
143     sub esp, 2
144     mov [esp], byte ah
145     mov [esp+1], byte al
146     mov ecx, esp
147    mov eax, SYS_WRITE
148     mov edx, 2
149     mov ebx, 1
150     int SYS_INIT_MODULE
151     mov esp, ebp
152     mov eax, SYS_EXIT
153     mov ebx,0
154     int SYS_INIT_MODULE
155     jmp _start
```

POTENCIA

La potencia al ser un tipo de funcion a la cual no tiene una funcion especifica, esta ser tratada como el como un complemento de varias otras funciones, hasta usar de forma retrocontinuada, siendo esta al ser multiplicada por una stack, los comandos de SYS son utilizados para captar algunas instrucciones del sistema como escritura, salir, iniciar etc. en este caso se usan en el momento para que el stack no se desborde de lo que programador quizo, para controlar lo que el dato sea trabajado en los registros ecx y esi se opto porque se trabaje con 2 y 5 teniendo el resultado de 64

LISTA DE MENU

```
152 section .data
153 msg: db 0Dh, "Esta es una calculadora en
        Assembler Seleccione" , 0Dh, "1. Suma " ,
        0Dh, "2. Resta" , 0Dh, "3. Multiplicacion "
        , 0Dh, "4. Division " , 0Dh, "5. Potencia",
        0Dh
154 len equ $ - msg
155 msg_suma:db "El resultado de la suma es: "
156 len_suma equ $ - msg_suma
157 msg_res: db "El resultado de la resta es: "
158 len_res equ $ - msg_res
159 msg_multi: db "El resultado de la
        multiplicacion es: "
160 len_multi equ $ - msg_multi
161 msg_div: db "El resultado de la division es: "
162 len_div equ $ - msg_div
163 msg_pot: db "El resultado de la potencia es: "
164 len_pot equ $ - msg_pot
165
```

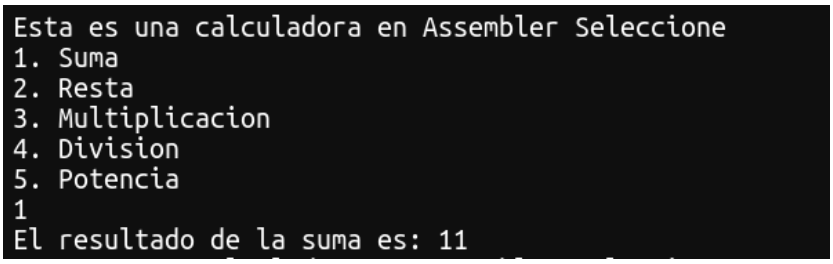
Se debe explicar que antes se habia creado un buffer para que el sistema guarde un dato, y este al ser un break, este se caracteriza por tener muchas respuestas a las que se tiene esperado enviar un mensaje a cada tipo de camino, por lo que para el sistema los mensajes se envian en la seccion de data, teniendo en cuanta que estos serán entregados por un msg que es lo que tiene el texto, y un len que es la longitud del texto.

```
175 section .bss
176     var1: resb 4
177     buf resb 1
```

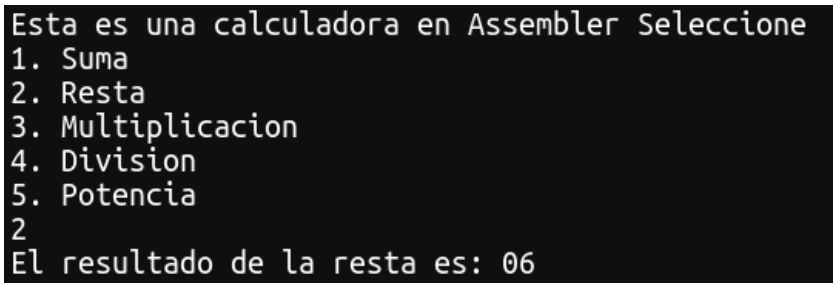
VARIABLES DINAMICAS

Para usar este tipo de variables se utiliza un tipo de cache que se reescriban muchas veces, buscando un uso total de recursos, y que el sistema no se confunda con los datos enviados, la variable de buffer se utliza para menu, siendo una fraccion de memoria para la toma de desiciones.

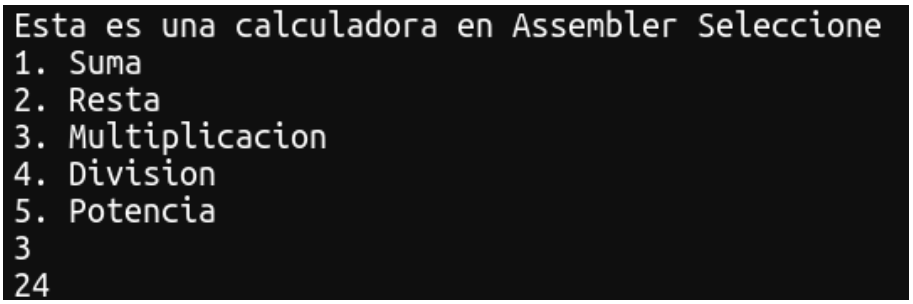
FUNCIONAMIENTO
SUMA 7+4



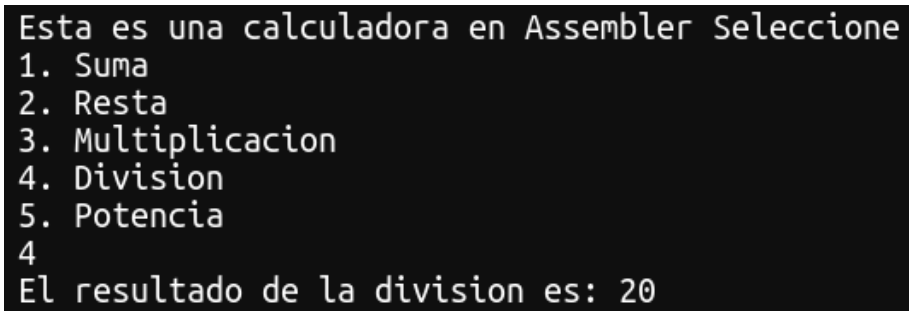
RESTA 8-2



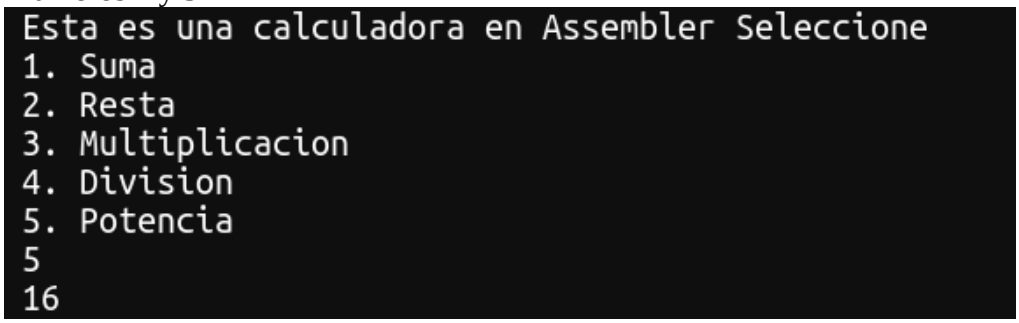
MULTIPLICACION 8 * 3



DIVISION 40 / 2



POTENCIA Numeros 2 y 5



GLOSARIO

FUNCION	DESCRIPCION
MOV	UTILIZA EL REGISTRO Y SE ESCRIBE ALGUN TIPO DE VALOR
INT	INTEGER QUE FUNCIONA COMO INTERMEDIADOR DE REGISTRO E INTERRUPCION CON ALGUNA FUNCION ESPECIFICA
ADD	FUNCION DE SUMATORIA DE DOS REGISTROS
SUB	FUNCION DE RESTA DE DOS REGISTROS
MUL	FUNCION DE MULTIPLICACION DE DOS REGISTROS
DIV	FUNCION DE DIVISION DE DOS REGISTROS
JE	LLAMADO DE UNA FUNCION ESPECIFICA
CALL	LLAMADO DE UNA FUNCION PERO ENVIANDO EL VALOR DE UN REGISTRO Y DEVUELTO POR LA FUNCION
JMP	SALTO DE LINEA PARA QUE RETORNE AL COMANDO DADO
SYS_INIT_MODULE	SIMBOLO DE SISTEMA PARA INICIAR UNA OPERACION MACRO
SYS_FINISH	SIMBOLO DE SISTEMA PARA CREAR UN BREAK TOTAL
SYS_WRITE	SIMBOLO DE SISTEMA EL QUE REESCRIBE ALGUN TIPO DE REGISTRO

CONCLUSIONES

Se determino que la forma de calcular las funciones es crear una función por cada una, llamándolas directamente desde los mensajes preliminares, ya que tutorialspoint al ser un emulador de compilador de 32 bits este desarrolla el código de forma mas facil sin hacer una función call por separado.

Se analizo para crear la calculadora fue usar un bufer, para escuchar el teclado y usar los registros dados por el sistema de 32 bits, los cuales son int 0x80, los metodos SYS, mov, add, sub, mul, div, 0Dh, jmp, je etc.

Se documento cada paso en el que el lenguaje ensamblador cuenta algunos problemas frecuentes, teniendo la mayor parte el de segmentar las secciones de texto>bss, alternando los registros, ya que algunas veces estos al ser usados como stack no pueden ser vueltos a usar o reescribirse.