Obligatorio Arquitectura de Computadoras

Eber Manuel Rodríguez González - 5.097.757-4

Noviembre 2022

Facultad de Ingeniería. Universidad de la República
Montevideo, Uruguay

Resumen

Implementación de calculadora con entrada en notación placa inversa en lenguaje ensamblador Intel 8086.

Índice

1. Introducción

1.1. Descripción del problema

El problema planteado para este laboratorio es la realización de un programa que modele una calculadora con operaciones básicas, y algunas funcionalidades particulares. El desarrollo de la calculadora será realizado en lenguaje ensamblador de la arquitectura 8086 con entrada en notación polaca inversa (RPN por sus siglas en ingles). La notación polaca inversa es un método de introducción de datos utilizado para evaluar expresiones matemáticas de la siguiente manera, primero se ingresan los operandos y luego el operador. La implementación de este tipo de calculadoras utiliza un stack para mantener los operandos hasta que sean requeridos por una operación, posteriormente colocan el resultado de dichos operandos en el tope del stack, para ser utilizado o no por más operaciones.

Ejemplo (operación con notación polaca inversa usando un stack).Se desea evaluar la siguiente expresión: $94 \times (42 + 5)$, por consiguiente la operación se realizara de la siguiente manera:

1. Se ingresan a la pila los operandos, $94, 42$ y 5 quedando la pila de la siguiente manera.

5 42 94	<- tope del stack

2. Luego al procesar el operador $+$ se retiran los operandos 5 y 42 y se procesa $5 + 42$ dejando en el tope de la pila el resultado de la operación quedando la pila de la siguiente manera

47 94	<- tope del stack

3. Finalmente se retiran el 47 y el 94 para procesar la operación $94/47$, dejando el resultado en el tope de la pila quedando esta de la siguiente manera:

2	<- tope del stack

1.2. Características de la implementación

1.2.1. Generales

- Se recibe la entrada de acuerdo a lo especificado en la sección 1.2.3
- El stack implementado soporta hasta 31 operandos de 16 bits representados en complemento a 2.
- Se pueden evaluar las siguientes operaciones ariméticas binarias: $+, -, *, /, \%$
- Se pueden evaluar las siguientes operaciones booleanas binarias: $, |, \ll, \gg$
- Se pueden evaluar las siguientes operaciones aritméticas unarias: Neg, Fact
- Exponer el tope de la pila en un puerto seleccionado: Top
- Realizar un volcado del stack en un puerto seleccionado: Dump
- Duplicar el top de la pila: Dup
- Intercambiar el top de la pila con el elemento debajo de él: Swap
- Sumar todos los elementos de la pila: Sum

1.2.2. Bitácora de ejecución

La calculadora mantiene un registro de su ejecución a medida que se van procesando cada comando. Se utiliza un puerto de salida definible por el usuario, la codificación de la misma es la siguiente:

- Antes de procesar un comando se manda el código 0 seguido del comando a procesar, en el caso de que el comando requiera un parámetro este también es mostrado en la bitácora.
- Luego de procesar el comando se manda uno de los siguientes codigos:

- Código 16: indica que la operación se realizó con éxito.
- Código 8: indica que la operación fallo por falta de operandos en la pila.
- Código 4: indica que la operación fallo por desbordamiento de la pila.
- Código 2: indica que no se reconoce el comando ingresado por el usuario.

1.2.3. Formato de entrada

La entrada se realizará leyendo un puerto E/S de 16 bits de solo lectura que denominaremos ENTRADA con el formato Comando [Parámetro], donde cada comando tiene predefinidos si requiere o no un parámetro, tanto los comando como los parámetros son de 16 bits. Cada comando y su operación están especificados en la siguiente tabla.

Comando	Parametro	Codigo	Descripción
NUM	Número	1	Agrega Número al stack
PORT	Puerto	2	Setea el puerto de salida
LOG	Puerto	3	Setea el puerto de la bitácora
TOP		4	Muestra el tope del stack en ele puerto de salida (no retira del stack)
DUMP		5	Realiza un volcado del stack en el puerto de salida (No destruye el stack)
DUP		6	Duplica el top de la pila
SWAP		7	Intercambia el tope de la pila con el elemento debajo
NEG		8	Calcula el opuesto
FACT		9	Calcula el factorial
SUM		10	Calcula la suma de todos los elementos de la pila (borrando la pila) y deja el resultado en el tope de la pila. Si no hay elementos deja 0 en el tope.
+,-,*,/, %,&,|,«,»		11 a 19	Realiza la operación binaria
CLEAR		254	Borra todo el contenido del stack
HALT		255	Detiene el procesamiento

2. Solución del problema

Como se mencionó en la sección anterior la solución al problema consta de una implementación de la calculadora en el lenguaje ensamblador de Intel 8086. Para esto primero se realizó un código escrito en el lenguaje C con el fin de usarlo como guía para la realización de las instrucciones a más bajo nivel, posteriormente se realizo la simulación usando el programa ArquiSim.

2.1. Características de la implementación en lenguaje ensamblador de Intel 8086

2.1.1. Declaración de constantes

Se declaran las siguientes constantes en la sección **.data** del código en assembler, utilizando la instrucción EQU para la definición de las mismas

Nombre	**Valor**	**Descipción**
ENTRADA	10	Es el puerto por defecto para la lectura de entrada
PUERTO_LOG_DEFECTO	2	Representa el puerto por defecto para la salida de la bitácora
SALIDA	1	Representa el valor del puerto (por defecto) por donde se van a retornar el resultado de ejecutar comandos que envian una salida
MAX_OPERADORES	62	Representa la cantidad de Bytes máximo para almacenar hasta 31 operadores

Cabe destacar que los valores indicados por dichas variables son los puertos que se usaran por defecto, debido a que existen las operaciones para cambiar de puertos mediante los comandos PORT y LOG, la implementación de dicho cambio sera vista en la sección 2.1.2

Adicionalmente se definen otra serie de constantes las cuales pueden resultar cómodas para ingresar los comandos, debido a su interpretación descriptiva del mismo.

Nombre	**Valor**	**Descipción**
C_NUM	1	Comando 1
C_PORT	2	Comando 2
C_LOG	3	Comando 3
C_TOP	4	Comando 4
C_DUMP	5	Comando 5
C_DUP	6	Comando 6
C_SWAP	7	Comando 7
C_NEG	8	Comando 8
C_FACT	9	Comando 9
C_SUM	10	Comando 10
C_SUMA	11	Comando 11
C_RESTA	12	Comando 12
C_MULT	13	Comando 13
C_DIV	14	Comando 14
C_MOD	15	Comando 15
C_AND	16	Comando 16
C_OR	17	Comando 17
C_SAL	18	Comando 18
C_SAR	19	Comando 19
C_CLS	254	Comando 254
C_HALT	255	Comando 255

2.1.2. Información de los puertos

Unos de los requerimientos que tiene que cumplir el programa, es la capacidad del usuario de cambiar el valor de los puerto de la bitácora y la salida de las operaciones, para ello se hace uso de ciertos espacios de memoria donde se guardan los valores de los puertos.

Por lo tanto en la sección de **.code** del codigo de assembler, se almacena el valor $0x4000$ en el registro de segmento DS, con el fin de poder guardar diferentes valores en dicho segmento y conocer su ubicación en todo momento, quedando la distribución de DS de la siguiente manera:

- En los primeros 2 Byte de memoria del segmento, es decir DS:[0], se guarda el valor

del puerto de la bitácora, inicialmente dicho valor corresponde al de la constante antes definida PUERTO_LOG_DEFECTO, este puede cambiar si el usuario lo desea, mediante el comando LOG.

- En los segundos 2 Byte de memoria del segmento, es decir DS:[2], guardaremos el valor del puerto correspondiente a la salida de operaciones, inicialmente el valor corresponde al de la constante antes definida SALIDA, pero este puede cambiar si el usuario desea mediante el comando PORT.

2.1.3. Implementación del stack

Unas de las cuestiones funcionales más importantes del programa es el uso de un pila para ir guardando los operadores y los resultados de las operaciones como se menciona en la sección 1.1; para llevar a cabo la implementación de la misma se inicializa el registro de segmento de memoria ES, como $0x2000$, y se distribuye de la siguiente manera:

- Para el tope de la pila empleamos el primer byte de memoria del segmento es decir el ES:[0], ahi mantendremos el tope de la pila como un desplazamiento respecto al segmento, y cada valor lo guardamos en 2 Bytes esta es la razón por que la constante MAX_OPERANDOS se define como 62, puesto que son los 31 operandos maximos que puede tener el stack en base al planteamiento del problema. Ejemplo si se tiene el elemento 0x5 como único elemento del stack se tendría

Dirección de memoria	**Valor**
0x2000 (ES:[0])	0x2
0x2002 (ES:[2])	0x5

Se puede apreciar que el valor en el tope de la pila es el desplazamiento con respecto a ES en donde se encuentra el último valor ingresado a la misma.

2.1.4. Etiquetas y loop

Una decisión de diseño importante es como representar cuando se tiene que detener la ejecución del programa, es decir que deje de leer la entrada, en esta ocasión se opto usar una

flag para indicar cuando se tiene que detener de leer la entrada; la forma de implementar la flag es la siguiente, se utiliza el segmento DS (previamente inicializado en $0x4000$ ver sección 2.1.2) exactamente en el cuarto byte de memoria se guarda dicha flag, inicialmente con el valor 0x1; es decir DS:[4] inicialmente tiene el valor 0x1.

Ahora se mencionarán las etiquetas utilizadas y sus instrucciones; en primer lugar se tiene la etiqueta **procesar** se podría decir que esta etiqueta cumple la funcionalidad del programa principal y deriva a otras etiquetas según la entrada.

Como primera instrucción de esta etiqueta es la de verificar si la flag que se usa para controlar la lectura de entrada (guardada en DS:[4]) este con el valor 0x1, en el caso de que la flag tenga el valor 0x0, inmediatamente saltamos a la etiqueta que representa el fin del procesamiento **finProcesar**; si el valor de dicha flag es 0x1 se procede a leer la entrada del puerto ENTRADA, para posteriormente derivar a la etiqueta que ejecuta el comando correspondiente; antes de realizar la comparación para saber que tiene que hacer el programa se muestra el código 0 y el código ingresado por la entrada en el puerto de la bitácora.

Cuando ya fue leída la entrada, se realizan un conjunto de comparaciones con el fin de determinar que comando se quiere ejecutar, esto da lugar a las siguientes etiquetas, **comando1, comando2, comando3, comando4, comando5, comando6, comando7, comando8, comando9, comando10, comando11, comando12, comando13, comando14, comando15, comando16, comando17, comando18, comando19, comando254, comando255**; cada una de estas realiza las instrucciones que implementan la operación correspondiente al comando cuyo código corresponda al número final de la etiqueta, ver sección 1.2.3; si la entrada leida no corresponde a ninguno de estos códigos salta a la etiqueta **retornarCodigoBitacora2**, cuyo conjunto de instrucciones muestran el código 2 en el puerto de la bitácora, indicando que el comando no fue reconocido.

Dentro de las etiquetas **comando** se encuentran las instrucciones para realizar la operación correspondiente al código que representan, además estas saltan a las etiquetas, **retornarCodigoBitacora4, retornarCodigoBitacora8, retornarCodigoBitacora16**, estas tiene instrucciones encargadas de mostrar por el puerto de la bitácora si la operación se pudo realizar satisfactoriamente o no, retornado el código de la bitácora correspondiente al que

tiene su nombre (significado de los codigos ver sección 1.2.2), y luego son las encargadas de volver a la etiqueta **procesar** realizando así un loop, que va leyendo las entradas hasta que se desea finalizar el procesamiento.

Otro aspecto del problema importante es que si se realiza una operación que requiere dos operandos en la pila pero solo tiene uno se debe eliminar este elemento, y quedar vacía la pila, para representar este comportamiento se utiliza una etiqueta llamada **insuficientesOperadoresOperacionBinaria**, que posee dos instrucciones una elimina los valores de la pila reseteando el tope y luego salta a la etiqueta **retronarCodigoBitacora8** mencionada anteriormente.

Por último se tiene la etiqueta **finProcesar** encargada de representar el fin del procesamiento, llamándose a sí misma para que no se ejecute ninguna nueva instrucción.

Cabe destacar que dentro de las instrucciones de algunas etiquetas hay otras etiquetas que son para uso interno las cuales se mencionan en la siguiente sección.

2.1.5. Particularidades de algunas etiquetas

comando1; esta tiene las instrucciones necesarias para realizar la operación correspondiente al comando 1 es decir agregar un elemento a la pila, es uno de los comando que requieren un parámetro por lo tanto la primera instrucción de esta etiqueta es leer nuevamente la entrada, y mostrar dicho valor en la bitácora; una vez que se tiene el parámetro se chequea que la pila no este lleno, esto se realiza mediante una instrucción CMP entre el valor guardado en la dirección de memoria que especificamos para el tope de la pila y la constante MAX_OPERADORES que indica la cantidad máxima (en Bytes) de operadores admitidos en la pila; en el caso de que la pila este lleno y no se admitan más operadores saltamos a la etiqueta **retornarCodigoBitacora4** que indica que la operación fallo y vuelve a la etiqueta procesar para la lectura de un nuevo comando; si la pila no esta lleno se procede a aumentar el valor del tope de la pila (ES:[0]) en 2 (cada valor representan 2 bytes en el segmento) y luego guardar el valor ingresado en el lugar de memoria del segmento, correspondiente al desplazamiento indicado por el nuevo tope (ejemplo; si el valor del tope luego de incrementarlo queda en 0x4 y el valor ingresado como parámetro es el 0x56, luego de realizar las instruccio-

nes queda ES:[0X4] guarda el valor 0x56 y ES:[0] guarda el valor 0x4). Si la operación pudo realizarse con éxito se realiza un salto a la etiqueta **retornarCodigoBitacora16**.

comando2 y comando3; las instrucciones que poseen estas etiquetas son muy similares, en pirmer lugar leen la entrada para obtener un parámetro y notifican mediante la bitácora dicho parámetro; posteriormente una corresponde a cambiar el puerto para la salida (**comando2**) y el otro es encargado de cambiar el valor del puerto de la bitácora (**comando3**), dichos valores se encuentran en las direcciones DS:[2] Y DS:[0] respectivamente; luego de cambiado hace un salto a la etiqueta **retornarCodigoBitacora16**.

comando4, las instrucciones de esta etiqueta son las encargadas de mostrar por el puerto de salida el valor guardado en el tope de la pila, para ello, primero se busca el valor del tope de la pila que se encuentra en ES:[0] (lo guarda temporalmente en el registro SI) y se compara que este no sea 0x0 (mediante la instrucción cmp) si es así significa que la pila esta vacía y por lo tanto salta a la etiqueta **retornarCodigoBitacora8**, por el contrario si la pila no esta vacía; busca el valor guardo en ES:[SI], es decir el que esta en el tope de la pila y lo muestra en el puerto reservado para la salida que esta guardado en DS:[2] y saltando a la etiqueta **retornarCodigoBitacora16**.

comando5, las instrucciones de esta etiqueta son las encargadas de mostrar por el puerto de salida todos los elementos de la pila, en este conjunto de instrucciones se utiliza una etiqueta auxiliar con el fin de representar un while loop para la recorrida de la pila, el formato en el que se muestra la pila en el puerto salida es primero su tope y luego el resto de elementos hasta su base; como primera instrucción buscamos el tope de la pila y lo guardamos en SI, este será el iterador que se usa para ir iterando sobre los elementos de la pila; si la pila no tiene elemenos directamente se salta a la etiqueta **finWhile5** que realiza un salto a la etiqueta **retornarCodigoBitacora16**, si valor en el tope de la pila es distinto de 0, procedemos a ingresar a la etiqueta **while5**, encargada de mirar el valor de la pila en la posición donde se encuentre el iterador en ese momento, mostrarlo por el puerto de salida y decrementar el iterador, finaliza cuando el iterador llega a 0, saltando a la etiqueta antes mencionada **finWhile5**, si el valor del iterador no es 0 salta nuevamente a la etiqueta **while5** realizando así el loop.

comando6, sus instrucciones son similares a las de la etiqueta **comando1**, solo que en vez de tomar un parametro de la entrada, utiliza el elemento que esta en el tope de la pila, agregándolo si es posible nuevamente y saltando a la etiqueta **retornarCodigoBitacora16**, si no es posible agregar porque excede el tope de la pila o porque no hay elemento en la pila para tomarlo como valor salta a las etiqutas **retornarCodigoBitacora4** y **retornarCodigoBitacora8** respectivamente.

comando7, esta etiqueta contiene las instrucciones para realizar un intercambio de valores entre el tope de la pila y el elemento que esta debajo, en primer lugar se verifica que la pila tenga más de un valor si no lo tiene salta a la etiqueta **insuficientesOperadoreOperacionBinaria**; si tiene más de un elemento, se guarda temporalmente en SI el valor guardado en ES:[0], es decir el tope de la pila, posteriormente se utilizan dos registros AX y BX para guardar temporalmente los valores de ES:[SI] y ES:[SI - 2], respectivamente, para luego poner el que esta en AX en ES:[SI - 2] y el que esta en BX en ES:[SI], y queda realizado el cambio, luego se salta a la etiqueta **retornarCodigoBitacora16**.

comando8, esta etiqueta contiene las instrucciones para realizar la negación del elemento que esta en el tope de la pila y dejar su negado en el tope, para ello se guarda temporalmente en SI el valor almacenado en ES:[0] correspondiente al tope de la pila, luego se chequea que no sea 0; es decir que la pila no esta vacía, si lo esta salta a la etiqueta **retornarCodigoBitacora8**, si no lo esta se sobre escribe el valor en el tope de la pila por su negado usando la instrucción neg y luego salta a la etiqueta **retornarCodigoBitacora16**.

comando9, esta etiqueta tiene las instrucciones para realizar la operación correspondiente al factorial del número, para realizar esta operación primero se chequea que el tope de la pila (se guarda temporalemente el valor de ES:[0] en el registro SI) no sea igual a 0x0 (pila vacía) si lo es salta a la etiqueta **retornarCodigoBitacora8**, si no, las siguientes instrucciones a ejecutar son para invocar una rutina recursiva llamada **fact** cuya implementación se explica en la sección 2.1.6, para llevar a cabo dicha invocación es necesario hacer la instrucción push con el valor del tope da pila, es decir el ES:[SI], luego se procede a realizar la instrucción call fact, invocando la rutina; posteriormente de obtenido el resultado salta a la etiqueta **retornarCodigoBitacora16**.

comando10, esta etiqueta tiene las instrucciones para realizar la suma de todos los elementos de la pila borrando dichos elementos y luego dejando el resultado en el tope, la idea es iterar sobre los elementos de la pila usando la misma estrategia que en la etiqueta **comando5**; es decir utilizar al registro SI como iterador y utilizar etiquetas auxiliares **while10** y **finWhile10** para representar el loop; en cada iteración se va acumulando la suma en el registro AX (inicializado antes de comenzar la iteracion en 0x0 usando la instrucción xor AX, AX), luego cuando todos los valores ya fueron iterados se deja a AX en ES:[2] y se sobre escribe el valor guardado en ES:[0] por 2, representando que solo hay un valor en la pila. Si no hay elementos en la pila directamente se salta a la etiqueta **finWhile10** donde tambíen se realiza el guardado de AX en el tope solo que en esta situación AX tiene el valor 0x0. Luego en la etiqueta **finWhile10** esta la instrucción que salta a la etiqueta **retornarCodigoBitacora16**.

comando11 hasta **comando19**, cada una de estas etiquetas como fue mencionado anteriormente tienen las instrucciones para realizar las diferentes operaciones binarias soportadas por el programa; una funcionalidad que tienen en común son que todas como primeras instrucciones chequean que la pila tenga por lo menos dos valores, para esto comparan el valor del tope de la misma con 0x4, si es menor saltan a la etiqueta **insuficientesOperadoresOperacionBinaria**, si tienen 2 o más operadores cada etiqueta realiza la operación binaria correspondiente y guardan su resultado en el nuevo tope de la pila que es el anterior decrementado 2, también guardan en ES:[0] el nuevo tope. Cabe destacar que en su mayoría son bastante directa la implementación con respecto a al funcionamiento de la operación que representan, por lo tanto se mencionaran las más particulares o en las que se tuvo que emplear una estrategia para su resolución.

comando14 y **comando15**, son las etiquetas correspondientes a realizar la división y el módulo entre los dos elementos más arriba en la pila.

Se analizara primero las instrucciones de **comando14**, la particularidad viene cuando el elemento en la posición ES:[SI - 2] (SI fue inicializado en ES:[0] es decir el tope de la pila) cuyo rol en la operación es ser el divisor, es negativo. Como la operación de la división en 16 bits hace uso de 2 registros (DX y AX) para almacenar al divisor, cuando este es negativo se tiene que poner en DX (parte alta del número) el valor 0xFFFF, que representa el número

negativo en complemento a 2; para ello se utiliza la instrucción cmp entre el registro AX (inicialmente cargado con el valor ES:[SI-2]) y 0x8000; como los número son representados en complemento a 2 de 16 bits si en su representación es mayor que 0x8000 significa que es negativo y por lo tanto se salta a la etiqueta auxiliar **setearDivNegativo** donde se carga en DX el valor 0xFFFF, luego se salta a la etiqueta **hacerDiv** donde estan las instrucciones para realizar la división (utilizando la instrucción idiv) y guardarla en el tope de la pila para posteriormente saltar a la etiqueta **retornarCodigoBitacora16**; por otro lado si el número no es negativo es decir que el valor cargado en AX es más chico que 0x8000, se carga 0x0 en CX y se procede a realizar las instrucciones de la etiqueta **hacerDiv**.

Consideración importante utilizar la instrucción idiv guarda el resultado de la división en el registro AX y el resultado del módulo en el registro DX, por lo tanto para la división el conjunto de instrucciones posterior a idiv son para cargar en el tope de la pila el valor guardado en AX, y en el caso del módulo la única diferencia es que guarda en el tope el valor guardado en DX, la diferencia en el conjunto de instrucciones es que se usaron nombres diferentes en las etiquetas auxiliares en el caso del modulo (etiqueta **comando15**), las etiquetas auxiliares se llaman **hacerMod** y **setearModNegativo**, las instrucciones en dichas etiquetas y las comparaciones que saltan hasta ellas son análogas a las mencionadas para la división.

comando18 y **comando19**, son las encargadas de realizar las instrucciones para realiar las operaciones shift izquierda y shift derecha respectivamente.

La particularidad de dichas etiquetes es cuando el valor guardado den el tope de la pila supera 0x10.

comando18, en primer lugar se carga el valor del tope de la pila (ES:[0]) en SI, luego se guarda en AX el valor guardado en ES:[SI], se compara dicho valor con 0x10 y si es mayor salta a la etiqueta auxiliar **shiftIzqMas16**, donde directamente setea en el nuevo tope de la pila (SI luego de ser decrementado en 2) el valor 0x0 y luego salta a la etiqueta **retornarCodigoBitacora16**; si la comparación falla y el valor en AX es menor a 0x10, se procede a realizar la instrucción shl cargando en CL el valor de AL y guardar su resultado en el nuevo tope de la pila.

comando19 al igual que el anterior se realiza la misma comparación entre AX (donde se

cargo el valor del tope de la pila) y 0x10, en este caso saltando a la etiqueta **shiftDerMas16**, las instrucciones en dicha etiqueta es realizar la instrucción sar previamente cargando CL con 0x11, esto asegura que si se realiza un corrimiento de más de 16 el valor guardado en el nuevo tope es 0x0 (si el valor guardado en AX es positivo) o 0xFFFF (si el valor guardado en AX es negativo); por otro lado si el salto a la etiqueta **shiftDerMas16** no se realiza, simplemente se carga el valor de AL en CL y se hace uso de la instrucción sar guardando el resutlado en el nuevo tope de la pila; en ambos casos se finaliza saltando a la etiqueta **retornarBitacoraCodigo16**

comando254, esta tiene las instrucciones para eliminar los elementos de la pila, para ello guarda el valor 0x0 en ES:[0], es decir deja el valor referencia del tope de la pila en 0x0.

comando255, tiene las instrucciones encargadas de finalizar el procesamiento y no leer más por entrada para esto setea la flag que se utiliza para tomar la lectura de entrada (guardada en DS:[4]) en 0x0.

2.1.6. Procedimiento recursivo para FACT

Se define un procedimiento llamado **fact** mediante la directiva **proc**, dicho procedimiento es implementado de forma recursiva por lo tanto hace uso del stack para poder ejecutarse.

Parámetros del procedimiento; el parámetro para el procedimiento debe estar en el tope del stack antes de realizar el call al procedimiento.

Valores que se interesan preservar; en primer lugar se tiene los registros que se desean preservar y que son utilizados a lo largo de las instrucciones; BP, AX, BX y DX, para ello la estrategia que se utiliza al inicio del procedimiento es pushar dichos registros para tener en el stack los valores que tienen al momento de ingresar al mismo.

Luego de preservar los valores de los registros que nos interesan, se procede a copiar el valor del registro SP en BP, y en el registro AX el valor en SS:[BP + 0XA], en AX queda el valor que entro como parámetro del procedimiento (el 0xA representa la cantidad de bytes que se tiene que desplazar en SS para obtener el valor, puesto que se pusharon 4 registros y el IP); una vez con el valor del parámetro en AX se tiene que chequear las condiciones si es un caso base que son si dicho valor es 0x1 o 0x0, en cuyo caso se realiza un salto a la

etiqueta **pasoBase**; la cual pone en la dirección SS:[BP + 0xA] el valor 0x1, luego realiza un salto a la etiquta **finRec**; en el caso de que las comparaciones fallen y no sea un caso base se ejecutan las instrucciones encargadas del paso recursivo que son en primer lugar, cargar en BX el valor de AX, decrementar BX en 1 y pushar BX, luego realizar el call **fact**, luego sacar el resutaldo del tope del stack multiplicarlo con el valor guardado en AX (valor anterior) y guardar el resultado en SS:[BP + 0xA] luego saltar a la etiqueta **finRec**.

Etiqueta **finRec**; tiene las instrucciones para preservar el contexto es decir restaurar los registros utilizados a sus valores antes de la llamada al procedimiento, para esto realiza una secuencia de instrucciones pop en el orden inverso a las instrucciones push al inicio del procedimiento, luego realiza la instrucción ret para finalizar el procedimiento.

3. Pruebas y problemas encontrados

En esta sección se mostraran algunas instancias de entradas y sus salidas respectivas luego de realizar el procesamiento de dicha entrada en el programa, cabe destacar que se mostraran las entradas y las respuestas en los respectivos puertos, el resultado en el caso de todos los test concuerdan con su comportamiento esperado.

3.1. Pruebas proporcionada por facultad

ENTRADA	**RESPUESTA**
1,1, 1,2, 1,3, 1,4, 1,5, 1,1, 1,9, 1,8, 1,-1400, 1,10, 1,11, 1,12, 1,13, 11, 4, 12, 4, 13, 4, 14, 4, 15, 4, 16, 4, 17, 4, 18, 4, 7, 4, 19, 4, 10, 4, 8, 4, 6, 5, 254, 255	Puerto 2: 0, 1, 1, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 1, 4, 16, 0, 1, 5, 16, 0, 1, 1, 16, 0, 1, 9, 16, 0, 1, 8, 16, 0, 1, -1400, 16, 0, 1, 10, 16, 0, 1, 11, 16, 0, 1, 12, 16, 0, 1, 13, 16, 0, 11, 16, 0, 4, 16, 0, 12, 16, 0, 4, 16, 0, 13, 16, 0, 4, 16, 0, 14, 16, 0, 4, 16, 0, 15, 16, 0, 4, 16, 0, 16, 16, 0, 4, 16, 0, 17, 16, 0, 4, 16, 0, 18, 16, 0, 4, 16, 0, 7, 16, 0, 4, 16, 0, 19, 16, 0, 4, 16, 0, 10, 16, 0, 4, 16, 0, 8, 16, 0, 4, 16, 0, 6, 16, 0, 5, 16, 0, 254, 16, 0, 255, 16 Puerto 1: 25, -14, -140, 10, 8, 8, 9, 2560, 4, 160, 166, -166, -166, -166
11, 6, 1,1234, 7, 1,4321, 5, 12, 5, 9, 5, 1,-5, 8, 16, 1,1, 1,2, 1,3, 1,4, 1,5, 1,6, 1,7, 1,8, 1,9, 1,10, 1,11, 1,12, 1,13, 1,14, 1,15, 1,16, 1,17, 1,18, 1,19, 1,20, 1,21, 1,22, 1,23, 1,24, 1,24, 1,26, 1,27, 1,28, 1,29, 1,30, 1,31, 1,32, 1,33, 5, 255	Puerto 2: 0, 11, 8, 0, 6, 8, 0, 1, 1234, 16, 0, 7, 8, 0, 1, 4321, 16, 0, 5, 16, 0, 12, 8, 0, 5, 16, 0, 9, 8, 0, 5, 16, 0, 1, -5, 16, 0, 8, 16, 0, 16, 8, 0, 1, 1, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 1, 4, 16, 0, 1, 5, 16, 0, 1, 6, 16, 0, 1, 7, 16, 0, 1, 8, 16, 0, 1, 9, 16, 0, 1, 10, 16, 0, 1, 11, 16, 0, 1, 12, 16, 0, 1, 13, 16, 0, 1, 14, 16, 0, 1, 15, 16, 0, 1, 16, 16, 0, 1, 17, 16, 0, 1, 18, 16, 0, 1, 19, 16, 0, 1, 20, 16, 0, 1, 21, 16, 0, 1, 22, 16, 0, 1, 23, 16, 0, 1, 24, 16, 0, 1, 24, 16, 0, 1, 26, 16, 0, 1, 27, 16, 0, 1, 28, 16, 0, 1, 29, 16, 0, 1, 30, 16, 0, 1, 31, 16, 0, 1, 32, 4, 0, 1, 33, 4, 0, 5, 16, 0, 255, 16 Puerto 1: 4321, 31, 30, 29, 28, 27, 26, 24, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
1,1, 1,2, 1,3, 9, 5, 2,3, 9, 5, 3,4, 7, 9, 5, 999, -999, 2,5, 5, 254, 1,-6, 1,2, 14, 4, 1,-2, 13, 4, 255	Puerto 2: 0, 1, 1, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 9, 16, 0, 5, 16, 0, 2, 3, 16, 0, 9, 16, 0, 5, 16, 0, 3, 4 Puerto 1: 6, 2, 1 Puerto 3: 720, 2, 1, 2, 720, 1 Puerto 4: 16, 0, 7, 16, 0, 9, 16, 0, 5, 16, 0, 999, 2, 0, -999, 2, 0, 2, 5, 16, 0, 5, 16, 0, 254, 16, 0, 1, -6, 16, 0, 1, 2, 16, 0, 14, 16, 0, 4, 16, 0, 1, -2, 16, 0, 13, 16, 0, 4, 16, 0, 255, 16 Puerto 5: 2, 720, 1, -3, 6

3.2. Pruebas comando 1

- Test 1, se testea la inserción de un elemento chequeamos imprimiendo la pila si se ingreso.
- Test 2, se testea insertar 31 elementos y luego insertar 1 más, se chequea que la bitácora retorne código 4 y se imprime la pila para ver si están todos y no esta el último que se quería ingresar.

TEST	**ENTRADA**	**RESPUESTA**
1	1, 1, 5, 255	Puerto 2: 0, 1, 1, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 1
2	1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1, 19, 1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1, 27, 1, 28, 1, 29, 1, 30, 1, 31, 1, 2, 1, 5, 5, 255	Puerto 2: 0, 1, 1, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 1, 4, 16, 0, 1, 5, 16, 0, 1, 6, 16, 0, 1, 7, 16, 0, 1, 8, 16, 0, 1, 9, 16, 0, 1, 10, 16, 0, 1, 11, 16, 0, 1, 12, 16, 0, 1, 13, 16, 0, 1, 14, 16, 0, 1, 15, 16, 0, 1, 16, 16, 0, 1, 17, 16, 0, 1, 18, 16, 0, 1, 19, 16, 0, 1, 20, 16, 0, 1, 21, 16, 0, 1, 22, 16, 0, 1, 23, 16, 0, 1, 24, 16, 0, 1, 25, 16, 0, 1, 26, 16, 0, 1, 27, 16, 0, 1, 28, 16, 0, 1, 29, 16, 0, 1, 30, 16, 0, 1, 31, 16, 0, 1, 2, 4, 0, 1, 5, 4, 0, 5, 16, 0, 255, 16 Puerto 1: 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

3.3. Pruebas comando 6

- Test 1, se testea la duplicación del tope de la pila sin elementos.
- Test 2, se testea la duplicación del tope de la pila con algún elemento pero no el máximo de elementos.
- Test 3, se testea la duplicación del tope de la pila con el máximo de elementos, no debería aparecer en la pila cuando se imprime, y hay un código 4 en la bitácora luego de ingresado el comando 6.

TEST	ENTRADA	RESPUESTA
1	6, 5, 255	Puerto 2: 0, 6, 8, 0, 5, 16, 0, 255, 16
2	1, 2, 6, 5, 255	Puerto 2: 0, 1, 2, 16, 0, 6, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 2, 2
3	1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1, 19, 1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1, 27, 1, 28, 1, 29, 1, 30, 1, 31, 6, 5, 255	Puerto 2: 0, 1, 1, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 1, 4, 16, 0, 1, 5, 16, 0, 1, 6, 16, 0, 1, 7, 16, 0, 1, 8, 16, 0, 1, 9, 16, 0, 1, 10, 16, 0, 1, 11, 16, 0, 1, 12, 16, 0, 1, 13, 16, 0, 1, 14, 16, 0, 1, 15, 16, 0, 1, 16, 16, 0, 1, 17, 16, 0, 1, 18, 16, 0, 1, 19, 16, 0, 1, 20, 16, 0, 1, 21, 16, 0, 1, 22, 16, 0, 1, 23, 16, 0, 1, 24, 16, 0, 1, 25, 16, 0, 1, 26, 16, 0, 1, 27, 16, 0, 1, 28, 16, 0, 1, 29, 16, 0, 1, 30, 16, 0, 1, 31, 16, 0, 6, 4, 0, 5, 16, 0, 255, 16 Puerto 1: 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

3.4. Pruebas comando 7

- Test 1, se testea realizar un swap sin elementos en la pila.
- Test 2, se testea realizar un swap con solo un elemento en la pila, se muestra un código 8 en la bitácora y no se debería imprimir nada en la pila, ya que el elemento se elimina.
- Test 3, se testea ingresar dos elementos imprimir la pila luego realizar el swap y volver a imprimir la pila, se realiza un cambio de puerto salida para que se vea mejor el resultado.

TEST	ENTRADA	RESPUESTA
1	7, 255	Puerto 2: 0, 7, 8, 0, 255, 16
2	1, 2, 7, 5 255	Puerto 2: 0, 1, 2, 16, 0, 7, 8, 0, 5, 16, 0, 255, 16
3	1, 2, 1, 3, 5, 7, 2, 3, 5, 255	Puerto 2: 0, 1, 2, 16, 0, 1, 3, 16, 0, 5, 16, 0, 7, 16, 0, 2, 3, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 3, 2 Puerto 3: 2, 3

3.5. Pruebas comando 8

- Test 1, se testea intentar procesar un comando neg sin elementos en la pila, el resultado es un código 8 en la bitácora.

- Test 2, se testea realizar un neg con el elemento positivo 42.
- Test 3, se testea realizar un neg con el elemento negativo -42.

TEST	**ENTRADA**	**RESPUESTA**
1	8, 255	Puerto 2: 0, 8, 8, 0, 255, 16
2	1, 1, 42, 8, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 8, 16, 0, 5, 16, 0, 255, 16 Puerto 1: -42
3	1, 1, -42, 8, 5, 255	Puerto 2: 0, 1, -42, 16, 0, 8, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 42

3.6. Pruebas comando 9

- Test 1, se testea intentar procesar un comando fact sin elementos en la pila.
- Test 2, se testea intentar procesar un comando fact con el elemento 0x0 en la pila.
- Test 3, se testea intentar procesar un comando fact con el elemento 5 en la pila.

TEST	**ENTRADA**	**RESPUESTA**
1	9, 255	Puerto 2: 0, 9, 8, 0, 255, 16
2	1, 0, 9, 5, 255	Puerto 2: 0, 1, 0, 16, 0, 9, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 1
3	1, 5, 9, 5, 255	Puerto 2: 0, 1, 5, 16, 0, 9, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 120

3.7. Pruebas comando 10

- Test 1, se testea intentar procesar un comando Sum sin elementos en la pila, el resultado esperado es que la pila tenga únicamente el elemento 0 en el tope.
- Test 2, se testea intentar procesar un comando sum con los elementos 1, 2, 3 y 4 en la pila, el resutlado esperado es 10 en el tope de la pila y único elemento.

TEST	ENTRADA	RESPUESTA
1	10, 5, 255	Puerto 2: 0, 10, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 0
2	1, 1, 1, 2, 1, 3, 1, 4, 10, 5, 255	Puerto 2: 0, 1, 1, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 1, 4, 16, 0, 10, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 10

3.8. Pruebas comando 11

- Test 1, se testea intentar procesar un comando + sin elementos en la pila, el resultado esperado es un código 8 en la bitácora.
- Test 2, se testea intentar procesar un comando + con un un solo elemento en la pila (el 42), el resultado esperado es un código 8 en la bitácora y que elimine dicho elemento, para esto imprimimos la pila antes de ejecutar el comando 11 o otra vez luego de procesado, se puede notar que el 42 ya no está en la pila.
- Test 3, ingresamos 3 elementos a la pila el 42, 2 y 3, luego mostramos la pila y se ve que están los elementos, 5 y 42, siendo 5 el resultado de 2 + 3 en el tope.

TEST	ENTRADA	RESPUESTA
1	11, 255	Puerto 2: 0, 11, 8, 0, 255, 16
2	1, 42, 5, 11, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 5, 16, 0, 11, 8, 0, 5, 16, 0, 255, 16 Puerto 1: 42
3	1, 42, 1, 2, 1, 3, 11, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 11, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 5, 42

3.9. Pruebas comando 12

- Test 1, se testea intentar procesar un comando − sin elementos en la pila, el resultado esperado es un código 8 en la bitácora.
- Test 2, se testea intentar procesar un comando − con un un solo elemento en la pila (el 42), el resultado esperado es un código 8 en la bitácora y que elimine dicho elemento,

para esto imprimimos la pila antes de ejecutar el comando 12 o otra vez luego de procesado, se puede notar que el 42 ya no está en la pila.

- Test 3, ingresamos 3 elementos a la pila el 42, 2 y 3, luego mostramos la pila y se ve que están los elementos, -1 y 42, siendo -1 el resultado de 2 - 3 en el tope.
- Test 4, ingresamos 3 elementos a la pila el 42, -2 y 3, luego mostramos la pila y vemos que estan los elementos, -5 y 42, siendo -5 el resultado de 2 - 3 en el tope.

TEST	**ENTRADA**	**RESPUESTA**
1	12, 255	Puerto 2: 0, 12, 8, 0, 255, 16
2	1, 42, 5, 12, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 5, 16, 0, 12, 8, 0, 5, 16, 0, 255, 16 Puerto 1: 42
3	1, 42, 1, 2, 1, 3, 12, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 12, 16, 0, 5, 16, 0, 255, 16 Puerto 1: -1, 42
4	1, 42, 1, -2, 1, 3, 12, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, -2, 16, 0, 1, 3, 16, 0, 12, 16, 0, 5, 16, 0, 255, 16 Puerto 1: -5, 42

3.10. Pruebas comando 13

- Test 1, se testea intentar procesar un comando $*$ sin elementos en la pila, el resultado esperado es un código 8 en la bitácora.
- Test 2, se testea intentar procesar un comando $*$ con un un solo elemento en la pila (el 42), el resultado esperado es un código 8 en la bitácora y que elimine dicho elemento, para esto imprimimos la pila antes de ejecutar el comando 13 o otra vez luego de procesado, se puede notar que el 42 ya no esta en la pila.
- Test 3, ingresamos 3 elementos a la pila el 42, 2 y 3, luego mostramos la pila y se ve que están los elementos, 6 y 42, siendo 6 el resultado de 2 * 3 en el tope.
- Test 4, ingresamos 3 elementos a la pila el 42, -2 y 3, luego se muestra la pila y se ve que están los elementos, -6 y 42, siendo -6 el resultado de -2 * 3 en el tope.

- Test 4, ingresamos 3 elementos a la pila el 42, -2 y 3, luego se muestra la pila y se ve que están los elementos, -6 y 42, siendo -6 el resultado de -2 * 3 en el tope.
- Test 5, ingresamos 3 elementos a la pila el 42, -2 y -3, luego se muestra la pila y se ve que están los elementos, 6 y 42, siendo 6 el resultado de -2 * -3 en el tope.

TEST	**ENTRADA**	**RESPUESTA**
1	13, 255	Puerto 2: 0, 13, 8, 0, 255, 16
2	1, 42, 5, 13, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 5, 16, 0, 13, 8, 0, 5, 16, 0, 255, 16 Puerto 1: 42
3	1, 42, 1, 2, 1, 3, 12, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 13, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 6, 42
4	1, 42, 1, -2, 1, 3, 13, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, -2, 16, 0, 1, 3, 16, 0, 13, 16, 0, 5, 16, 0, 255, 16 Puerto 1: -6, 42
5	1, 42, 1, -2, 1, -3, 13, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, -2, 16, 0, 1, -3, 16, 0, 13, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 6, 42

3.11. Pruebas comando 14

- Test 1, se testea intentar procesar un comando / sin elementos en la pila, el resultado esperado es un código 8 en la bitácora.
- Test 2, se testea intentar procesar un comando / con un un solo elemento en la pila (el 42), el resultado esperado es un código 8 en la bitácora y que elimine dicho elemento, para esto imprimimos la pila antes de ejecutar el comando 14 o otra vez luego de procesado, se puede notar que el 42 ya no está en la pila.
- Test 3, ingresamos 3 elementos a la pila el 42, 6 y 3, luego mostramos la pila y se ve que están los elementos, 2 y 42, siendo 2 el resultado de 6 / 3 en el tope.
- Test 4, ingresamos 3 elementos a la pila el 42, -6 y 3, luego se muestra la pila y se ve que están los elementos, -2 y 42, siendo -2 el resultado de -6 / 3 en el tope.

- Test 5, ingresamos 3 elementos a la pila el 42, -6 y -3, luego se muestra la pila y se ve que están los elementos, 2 y 42, siendo 2 el resultado de -6 / -3 en el tope.

TEST	**ENTRADA**	**RESPUESTA**
1	14, 255	Puerto 2: 0, 14, 8, 0, 255, 16
2	1, 42, 5, 14, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 5, 16, 0, 14, 8, 0, 5, 16, 0, 255, 16 Puerto 1: 42
3	1, 42, 1, 6, 1, 3, 14, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 6, 16, 0, 1, 3, 16, 0, 14, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 2, 42
4	1, 42, 1, -6, 1, 3, 14, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, -6, 16, 0, 1, 3, 16, 0, 14, 16, 0, 5, 16, 0, 255, 16 Puerto 1: -2, 42
5	1, 42, 1, -6, 1, -3, 14, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, -6, 16, 0, 1, -3, 16, 0, 14, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 2, 42

3.12. Pruebas comando 15

- Test 1, se testea intentar procesar un comando % sin elementos en la pila, el resultado esperado es un código 8 en la bitácora.
- Test 2, se testea intentar procesar un comando % con un un solo elemento en la pila (el 42), el resultado esperado es un código 8 en la bitácora y que elimine dicho elemento, para esto imprimimos la pila antes de ejecutar el comando 15 o otra vez luego de procesado, se puede notar que el 42 ya no está en la pila.
- Test 3, ingresamos 3 elementos a la pila el 42, 3 y 6, luego mostramos la pila y se ve que están los elementos, 3 y 42, siendo 3 el resultado de 3 % 6 en el tope.
- Test 4, ingresamos 3 elementos a la pila el 42, -3 y 6, luego se muestra la pila y se ve que están los elementos, -3 y 42, siendo -3 el resultado de -3 % 6 en el tope.
- Test 5, ingresamos 3 elementos a la pila el 42, 896 y 94, luego se muestra la pila y se ve que están los elementos, 50 y 42, siendo 50 el resultado de 896 % 94 en el tope.

TEST	ENTRADA	RESPUESTA
1	15, 255	Puerto 2: 0, 15, 8, 0, 255, 16
2	1, 42, 5, 15, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 5, 16, 0, 15, 8, 0, 5, 16, 0, 255, 16 Puerto 1: 42
3	1, 42, 1, 3, 1, 6, 15, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 3, 16, 0, 1, 6, 16, 0, 15, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 3, 42
4	1, 42, 1, -3, 1, 6, 15, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, -3, 16, 0, 1, 6, 16, 0, 15, 16, 0, 5, 16, 0, 255, 16 Puerto 1: -3, 42
5	1, 42, 1, 896, 1, 94, 15, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 896, 16, 0, 1, 94, 16, 0, 15, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 50, 42

3.13. Pruebas comando 16

- Test 1, se testea intentar procesar un comando & sin elementos en la pila, el resultado esperado es un código 8 en la bitácora.
- Test 2, se testea intentar procesar un comando & con un un solo elemento en la pila (el 42), el resultado esperado es un código 8 en la bitácora y que elimine dicho elemento, para esto imprimimos la pila antes de ejecutar el comando 16 o otra vez luego de procesado, se puede notar que el 42 ya no está en la pila.
- Test 3, ingresamos 3 elementos a la pila el 42, 57 y 95, luego mostramos la pila y se ve que están los elementos, 25 y 42, siendo 25 el resultado de 57 & 95 en el tope.

TEST	ENTRADA	RESPUESTA
1	16, 255	Puerto 2: 0, 16, 8, 0, 255, 16
2	1, 42, 5, 16, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 5, 16, 0, 16, 8, 0, 5, 16, 0, 255, 16 Puerto 1: 42
3	1, 42, 1, 57, 1, 95, 16, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 57, 16, 0, 1, 95, 16, 0, 16, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 25, 42

3.14. Pruebas comando 17

- Test 1, se testea intentar procesar un comando | sin elementos en la pila, el resultado esperado es un código 8 en la bitácora.
- Test 2, se testea intentar procesar un comando | con un un solo elemento en la pila (el 42), el resultado esperado es un código 8 en la bitácora y que elimine dicho elemento, para esto imprimimos la pila antes de ejecutar el comando 17 o otra vez luego de procesado, se puede notar que el 42 ya no está en la pila.
- Test 3, ingresamos 3 elementos a la pila el 42, 57 y 95, luego mostramos la pila y se ve que están los elementos, 127 y 42, siendo 127 el resultado de 57 | 95 en el tope.

TEST	**ENTRADA**	**RESPUESTA**
1	17, 255	Puerto 2: 0, 17, 8, 0, 255, 16
2	1, 42, 5, 17, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 5, 16, 0, 17, 8, 0, 5, 16, 0, 255, 16 Puerto 1: 42
3	1, 42, 1, 57, 1, 95, 17, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 57, 16, 0, 1, 95, 16, 0, 17, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 127, 42

3.15. Pruebas comando 18

- Test 1, se testea intentar procesar un comando $\ll$ sin elementos en la pila, el resultado esperado es un código 8 en la bitácora.
- Test 2, se testea intentar procesar un comando $\ll$ con un un solo elemento en la pila (el 42), el resultado esperado es un código 8 en la bitácora y que elimine dicho elemento, para esto imprimimos la pila antes de ejecutar el comando 18 o otra vez luego de procesado, se puede notar que el 42 ya no está en la pila.
- Test 3, ingresamos 3 elementos a la pila el 42, 57 y 3, luego mostramos la pila y se ve que están los elementos, 456 y 42, siendo 456 el resultado de 57 $\ll$ 3 en el tope.

- Test 4, ingresamos 3 elementos a la pila el 42, 57 y 17, luego mostramos la pila y se ve que están los elementos, 0 y 42, siendo 0 el resultado de 57 $\ll$ 17 en el tope.

TEST	**ENTRADA**	**RESPUESTA**
1	18, 255	Puerto 2: 0, 18, 8, 0, 255, 16
2	1, 42, 5, 18, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 5, 16, 0, 18, 8, 0, 5, 16, 0, 255, 16 Puerto 1: 42
3	1, 42, 1, 57, 1, 3, 18, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 57, 16, 0, 1, 3, 16, 0, 18, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 456, 42
4	1, 42, 1, 57, 1, 17, 18, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 57, 16, 0, 1, 17, 16, 0, 18, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 0, 42

3.16. Pruebas comando 19

- Test 1, se testea intentar procesar un comando $\gg$ sin elementos en la pila, el resultado esperado es un código 8 en la bitácora.
- Test 2, se testea intentar procesar un comando $\gg$ con un un solo elemento en la pila (el 42), el resultado esperado es un código 8 en la bitácora y que elimine dicho elemento, para esto imprimimos la pila antes de ejecutar el comando 19 o otra vez luego de procesado, se puede notar que el 42 ya no está en la pila.
- Test 3, ingresamos 3 elementos a la pila el 42, 57 y 3, luego mostramos la pila y se ve que están los elementos, 456 y 42, siendo 456 el resultado de 57 $\gg$ 3 en el tope.
- Test 4, ingresamos 3 elementos a la pila el 42, -57 y 17, luego mostramos la pila y se ve que están los elementos, -1 y 42, siendo -1 el resultado de -57 $\gg$ 17 en el tope.
- Test 5, ingresamos 3 elementos a la pila el 42, 57 y 17, luego mostramos la pila y se ve que están los elementos, 0 y 42, siendo 0 el resultado de 57 $\gg$ 17 en el tope.

TEST	ENTRADA	RESPUESTA
1	19, 255	Puerto 2: 0, 19, 8, 0, 255, 16
2	1, 42, 5, 19, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 5, 16, 0, 19, 8, 0, 5, 16, 0, 255, 16 Puerto 1: 42
3	1, 42, 1, 57, 1, 3, 19, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 57, 16, 0, 1, 3, 16, 0, 19, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 7, 42
4	1, 42, 1, -57, 1, 17, 19, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, -57, 16, 0, 1, 17, 16, 0, 19, 16, 0, 5, 16, 0, 255, 16 Puerto 1: -1, 42
5	1, 42, 1, 57, 1, 17, 19, 5, 255	Puerto 2: 0, 1, 42, 16, 0, 1, 57, 16, 0, 1, 17, 16, 0, 19, 16, 0, 5, 16, 0, 255, 16 Puerto 1: 0, 42

3.17. Pruebas comando 254

- Test 1, se testea intentar procesar un comando Clear sin elementos en la pila, el resultado esperado es un código 16 en la bitácora.
- Test 2, se ingresan 31 elementos a la pila luego se ejecuta el comando Clear, y se chequea los elementos de la pila con el comando 5 como no se muestra salida es que la pila esta vacía.

TEST	ENTRADA	RESPUESTA
1	254, 255	Puerto 2: 0, 254, 16, 0, 255, 16
2	1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1, 19, 1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1, 27, 1, 28, 1, 29, 1, 30, 1, 31, 254, 5, 255	Puerto 2: 0, 1, 1, 16, 0, 1, 2, 16, 0, 1, 3, 16, 0, 1, 4, 16, 0, 1, 5, 16, 0, 1, 6, 16, 0, 1, 7, 16, 0, 1, 8, 16, 0, 1, 9, 16, 0, 1, 10, 16, 0, 1, 11, 16, 0, 1, 12, 16, 0, 1, 13, 16, 0, 1, 14, 16, 0, 1, 15, 16, 0, 1, 16, 16, 0, 1, 17, 16, 0, 1, 18, 16, 0, 1, 19, 16, 0, 1, 20, 16, 0, 1, 21, 16, 0, 1, 22, 16, 0, 1, 23, 16, 0, 1, 24, 16, 0, 1, 25, 16, 0, 1, 26, 16, 0, 1, 27, 16, 0, 1, 28, 16, 0, 1, 29, 16, 0, 1, 30, 16, 0, 1, 31, 16, 0, 254, 16, 0, 5, 16, 0, 255, 16

3.18. Problemas encontrados a lo largo del desarrollo

Los problemas encontrados a lo largo del desarrollo que cabe mencionar debido a que replantearon estrategias de implementación, fueron, para la implementación del comando 14

y 15, en principio no se estaba considerando los casos donde el divisor fuera negativo, por esto se tuvo que agregar una forma de representar esto, este error fue detectado gracias a los test realizados en la secciones 3.11 y 3.12.

4. Mejoras a futuro

La principal mejora a futuro que se encuentra en esta implementación es la redundancia de instrucciones, es decir en muchas partes del código se esta realizando repetidamente las mismas instrucciones, por lo tanto identificar esa redundancia y plantear una estrategia para factorizar instrucciones en común ya sea mediante directivas de etiquetas o procedimientos, con el fin de mejorar el rendimiento global del programa.

5. Conclusiones

Es un programa útil para resolver operaciones básicas.

Lo importante es que el laboratorio, es capaz de englobar conceptos del lenguaje ensamblador de Intel 8086, los cuales proporcionan una idea de como realizar instrucciones y programas a bajo nivel.

Referencias

[1] ArquiSim: manual de usuario. https://eva.fing.edu.uy/mod/resource/view.php?id=61253. Accedido: 12-10- 2021.