FACULTAD DE
INGENIERÍA
UDELAR

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Curso Arquitectura de Computadoras. Laboratorio 2022

5.147.233-7, Ian Ignacy Arazny Casanovas

Índice:

Descripción de la tarea

Se desea construir un programa ArquiCalc que modela una sencilla calculadora RPN.

La notación polaca inversa es un método alternativo de introducción de datos en el cual para evaluar expresiones matemáticas primero se introducen los operandos y luego el operador. Por ejemplo la siguiente expresión algebráica 10 – 7 se escribe 10 7 – y un ejemplo un poco más complejo, 14 / (3 + 4) puede reescribirse en notación RPN así: 14 3 4 + /.

Las calculadoras que emplean notación polaca inversa utilizan una pila (stack) para mantener los operandos en memoria hasta que sean requeridos por la siguiente operación. Una vez que los operandos se sacan de la pila (para operaciones binarias primero se quita el lado derecho y luego el izquierdo) se realiza la operación y el resultado se coloca en el tope de la pila.

Así en el ejemplo de la expresión 14 / (3 + 4) y suponiendo que la pila inicialmente está vacía se introducen los tres operandos

4 3 14	<- Tope de la pila

Luego al procesar el operador + se retiran los operandos 4 y 3 de la pila, se computa 4+3 dando como resultado 7 y la pila queda de esta manera:

7 14	<- Tope de la pila

El siguiente operador a tomar es / por lo que se toman los operandos 7 (divisor) y 14 (dividendo) de la pila y se calcula la división dando como resultado 2 y la pila queda de esta manera:

2	<- Tope de la pila

Los usuarios de la calculadora pueden intercalar operandos y operadores de tal manera que el resultado de una expresión se utilice como operando en otra expresión. Por ejemplo si quiero que el resultado del ejemplo anterior se utilice como operando de la expresión resultado anterior + 1 basta con agregar 1 + a la entrada de datos anterior, quedando la secuencia 14 3 4 + / 1 +. Luego de evaluar hasta la división lo siguiente es agregar el valor 1 al tope de la pila:

1 2	<- Tope de la pila

Y al procesar el operador + se toman los operandos y se calcula 1 + 2 dando como resultado 3 que queda en la pila

3	<- Tope de la pila

El formato de entrada se realiza siguiendo el sistema de codificación de la siguiente tabla.

Comando	Parámetro	Codigo	Descripción
Num	Número	1	Agrega Numero a la pila
Port	Puerto	2	Setea el puerto de salida
Log	Puerto	3	Setea el puerto de la bitácora
Top		4	Muestra el tope de la pila en el puerto de salida (no retira del la pila)
Dump		5	Realiza un volcado de la pila en el puerto de salida (no retira de la pila)
DUP		6	Duplica el tope de la pila
SWAP		7	Intercambia 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 operación binaria
Clear		254	Borra todo el contenido de la pila
Halt		255	Detiene el procesamiento

Descripción de la solución incluyendo detalle de las estructuras de datos y constantes utilizadas

Mi solución general al problema consta de un arreglo de 31 celdas que acciona como pila, para poder manejar la estructura de manera correcta poseo un tope, el cual indica cuál es la última celda del arreglo utilizada, de esta manera puedo tener mejor manejo de los datos ingresados. El tope se inicializa en -1 en C y -2 en assembler, cada vez que se apila un elemento se incrementa el tope (en 1 en C y en 2 en assembler).

Las constantes y variables utilizadas son:
-PUERTO_SALIDA_DEFECTO: esta constante es utilizada para definir el puerto de salida por defecto, la defino como 0x0002 de manera arbitraria.

-PUERTO_LOG_DEFECTO: esta constante es utilizada para definir el puerto de salida por defecto para la bitácora, la defino como 0x0001 de manera arbitraria.

-puertoSalida: es una variable inicializada a partir del puerto salida defecto, esta variable es la que setea el puerto de salida, al modificar el puerto de salida se modifica esta variable.

-puertoSalidaBit: de manera similar a la variable puertoSalida, esta variable tiene la función de setear el puerto de salida de la bitácora.

Solución al problema en C

La solución proporcionada en C consta de un ciclo do while donde pido códigos de entrada mientras que este código no sea 255. Se observa que el código de assembler es una compilación manual de la versión de C, por lo que hay algunas redundancias como en el caso 255 donde se accede a un while(true){} en vez de terminar con ciclo while.
En base a este funcionamiento se actualizarán los valores de la pila ó se mostrarán por puerto según el código de entrada. La salida de datos por el puerto de bitácora en C no tiene exactamente la misma funcionalidad

Interpretación de cada caso en función de la entrada.

Aclaración: cada caso representa un comando que sigue el sistema de codificación propuesto.

Código 1: En primera instancia se verifica si el tope aún no rebasa la cantidad máxima de elementos admitida, y en caso de estar habilitado a apilar se incrementa el tope, se recibe el número a apilar por entrada y se apila en el tope, posteriormente se imprime el código . En otro caso, se muestra el código de que ocurrió un desbordamiento.

Código 2: Se lee el nuevo puerto de salida a setear y se re asigna el valor al puertoSalida.

Código 3: Se lee el nuevo puerto de salida a setear y se re asigna el valor al puertoSalidaBit.

Código 4: Se verifica con el tope si la pila tiene elementos. Si la pila tiene elementos se muestra por pantalla el elemento del tope.

Código 5: En caso de tener elementos saltamos a un ciclo que imprime todos los elementos de la pila.

Código 6: En este caso se distingue entre el error por falta de elementos y por desbordamiento de elementos. El caso de desbordamiento se distingue en dos, si estaba esperando un argumento extra (como en el caso de apilar, donde se debe ignorar este elemento) o si no, en este caso como hay posibilidad de desbordamiento a través de duplicar el elemento del tope simplemente se muestra el código de error. En otro caso se realiza la operación deseada.

Código 7: Para intercambiar los últimos dos elementos de la pila deben haber al menos dos elementos en la pila, por lo tanto, en caso de haber al menos dos elementos (tope>0) se intercambian los últimos dos elementos.

Código 8: En caso de poder, se realiza la operación correspondiente.

Código 9: En caso de que haya al menos un elemento, se asigna al tope de la pila el factorial del mismo a través de una función auxiliar con rutina recursiva.

Código 10: En caso de que haya al menos un elemento, se suman todos los elementos de la pila a través de una función auxiliar, el resultado de esta suma se pone en la primera celda del arreglo y se ajusta el tope con el fin de cumplir el comportamiento deseado.

Códigos 11-19: Si se realiza una operación binaria con un solo operando en la pila, se desapila este elemento. En otro caso se realiza la operación correspondiente y se apila el resultado.

Código 254: Borrar el contenido de la pila es el mismo comportamiento que resetear el tope en -1.

Código 255: En este caso simplemente se muestra en pantalla que se recibió el código con éxito y dado que la condición de iteración es que el código sea distinto a este, se detendrá la ejecución del ciclo.

Con esto termina la explicación de la solución proporcionada en C.

Solución al problema en Assembler

La solución general en Assembler es la interpretación de los parámetros de entrada, comparando si el código ingresado es uno de los códigos esperados especificados en la descripción del problema, en caso de serlo, se ejecuta la acción correspondiente.

Una vez se recibe un código por entrada, se almacena en el registro AX e inmediatamente después se muestra 0 en el puerto de salida de la bitácora previo al procesamiento de este comando, inmediatamente luego se imprime en el puerto el comando a procesar, aún almacenado en AX.

El algoritmo para mostrar en pantalla datos es bastante similar en todos los casos, se preserva la información que esté contenida previamente en el registro DX e igualmente con AX, se asigna en DX el valor del puerto salida o puerto salida de la bitácora dependiendo el caso y en AX lo que se quiera imprimir en pantalla, se presenta a través de un OUT el resultado en pantalla y luego a través de POP AX, POP DX, se devuelve el valor previo de los registros.

Interpretación de cada caso en función de la entrada.

Aclaración: cada caso representa un comando que sigue el sistema de codificación propuesto.

Case 1: Se intenta apilar un nuevo elemento, para esto se verifica si la cantidad de elementos en la pila es menor al límite establecido. En caso de poder apilar, se espera que el próximo parámetro de entrada sea el número a apilar, se muestra en el puerto de salida de la bitácora este mismo, se incrementa el tope y se apila, posteriormente se accede al mensaje de éxito donde se imprime 16 en en la salida del puerto bitácora y se espera el siguiente código.

Case 2 y 3: Se setea el puerto de salida, mostrando en el puerto salida de la bitácora el nuevo puerto a salida. Simplemente se asigna a la variable el valor recibido.

Case 4: En caso de no tener los suficientes elementos para realizar la operación se accede a una etiqueta encargada de mostrar el código correspondiente en salida y esperar un código siguiente, esta conducta será repetida para todos los casos que no tengan los suficientes elementos para ser realizados. En caso de poder realizar la operación se muestra en el puerto de salida el elemento del tope, siendo el tope actualizado en SI tras apilar.

Case 5: Muevo a BX el elemento del tope, y se verifica si la pila posee elementos que mostrar, en caso de tener (tope >= 0) se ejecuta cicloDump que cumple la función de imprimir en pantalla los valores de la pila, tal que, se guarda en DX el puerto de salida, se guarda en AX el valor del tope, se muestra en pantalla el elemento del tope y se decrementa el tope (almacenado en BX) mientras la pila tenga algún elemento. Al finalizar el ciclo la pila no se ve afectada ya que fue recorrida ayudándonos de un registro auxiliar y no la variable en sí.

Case 6: En caso de poder duplicar el tope, se guarda el tope en AX, se incrementa el tope y se pone en el nuevo tope.

Case 7: En caso de que haya un solo elemento se desapila el elemento que había por falta de operandos. En caso de que no hayan elementos se muestra en el puerto de salida, en otro caso, se intercambian los elementos del tope con los elementos del tope-1. Para ello se guarda en AX el elemento del tope, se disminuye en el valor de SI, dónde está almacenado el tope, se guarda la variable a intercambiar en BX, se actualiza el valor del tope-1 con AX y se actualiza el valor del tope con BX realizando exitosamente el intercambio.

Case 9: Previo a la ejecución del procedimiento factorial se guarda el valor previo de BX y se deja con 0. Se implementa el procedimiento factorial a través de una rutina recursiva de manera idéntica a la vista en el curso, adicionalmente se modifica el valor del tope con el valor del factorial almacenado en BX.

Case 10: Para este caso es pertinente aclarar que la operación siempre resulta exitosa, en caso de no tener elementos se salta a una etiqueta que se encarga de dejar 0 como único elemento de la pila y de mandar el mensaje de éxito. En caso de tener elementos se guarda lo que haya previamente en los registros BX y CX a través de PUSH (para posteriormente hacer POP y retornar el valor previo), CX será utilizado como acumulador y BX almacenará el valor del tope, de esta manera no se modificará el valor original del tope, luego, se accede a la etiqueta sum, encargada de sumar el elemento del tope de la pila en CX y disminuir la cantidad del tope en 2 mientras que el tope sea mayor que 0, es decir, mientras siga habiendo elementos sin sumar en la pila. Una vez terminado este loop, es decir, cuando BX queda con el valor -2, se re asigna el valor al tope como 0 para poner en esta posición el valor acumulado hasta el momento (contenido en CX), se devuelve el valor previo de CX, BX y se salta a la etiqueta encargada de mostrar en el puerto de salida de la bitácora el código de éxito.

En los casos 11-19 se distinguen dos tipos de errores posibles cuando se quiere realizar una operación binaria de la categoría falta de operandos, en el caso de que previamente en la pila haya un elemento y el caso donde no haya elementos en la pila, para el primero debemos desapilar el elemento sobre el cual se quiso operar. Este caso se identifica verificando si el tope de la pila es 0, en caso de serlo se accede a una etiqueta encargada de actualizar el tope y el registro SI, luego de esto

se procede a mostrar el error correspondiente por falta de elementos (con el código 8). A continuación se explica el funcionamiento natural de la operación.

Case 11: En caso de estar en condiciones de realizar la suma, previamente a realizar la operación, se usa BX con el fin de almacenar el valor del elemento del tope, se decrementa el tope, se suma el elemento y se añade el resultado a la pila.

Case 12: ídem a la suma pero reemplazando por resta.

Case 13: Se guarda la información previa de AX, BX y DX, debido al funcionamiento interno de la función MUL luego de realizar la multiplicación se verán modificados los registros AX y DX, luego, se realiza un procedimiento similar a los casos 11 y 12 pero tomando en cuenta que MUL multiplica lo que esté en AX con el argumento que se reciba, por lo tanto me aseguro previamente de cargar el elemento del tope en AX y el elemento del tope -1 en BX. Luego, se le asigna el resultado almacenado en AX al actual tope (considerando el decremento en el mismo) y se procede a la etiqueta de éxito.

Case 14: La algoritmia de este caso es similar al de la multiplicación, se guarda la información de los registros AX, BX y DX, se almacena en BX el elemento del tope y en AX el elemento del anterior al tope. En este caso debido al funcionamiento del IDIV debemos distinguir el caso de que lo almacenado en AX (lo que vendría a ser el elemento del tope al momento de acceder a la etiqueta) es negativo ó positivo, en caso de que sea negativo, dado que la representación es en complemento a dos y dado el funcionamiento de la función IDIV necesito poner en DX 0xFFFF para lograr que los resultados sean los esperados, luego de esto, en ambos casos se hace la división y el resultado esperado se encuentra en AX, se asigna el resultado a la posición correspondiente y se recupera la información previa en los registros. Se utiliza la operación IDIV para poder trabajar con la división con signo.

Case 15: Este caso es análogo al caso 14, la única diferencia es que el resto de la división entera tras el IDIV se encuentra almacenada en DX.

Case 16: Este caso es análogo al de la suma pero con la variante de la operación.

Case 17: ídem caso 16.

Case 18: Para el caso del shift izquierdo hay que distinguir el caso borde donde se al ser la operación SAL con la parte baja de CX, cuando se quiere shiftear más de 16 posiciones, el resultado generalmente da un error pues no considera el número completo, pero de todas formas sabemos que, en caso de un desplazamiento de más de 16 posiciones el resultado termina siendo cero. Ejemplo general de lo que sucede:
Se quiere desplazar (“hacer shift”) de un número genérico: XXXXXXXXXXXXXXXXb (entre 0 y 65535)
La cantidad a desplazar es: MMMMMMMMNNNNNNNb.
Como en cada desplazamiento se sustituyen por ceros por la izquierda tenemos que desplazando más de 16 posiciones el resultado será 0000000000000000b.
De otra forma la operación se realiza normalmente con un procedimiento similar a las otras operaciones.

Case 19: En este caso también hay que distinguir dos casos bordes nuevamente en base a si la cantidad desplazada es mayor a 16, bajo estos términos debemos saber si el número a desplazar es positivo o negativo, en caso de ser positivo sabemos que en la representación CA2 el bit más significativo es 0 fijo, tras el desplazamiento el número anterior se ve reemplazado por 1s en todas sus entradas menos en la del bit más significativo, dando el resultado 0. Cuando el número a desplazar es negativo tenemos que por el contrario al caso anterior tenemos 1 en el bit más significativo al igual que en el resto de las entradas, dando como resultado -1.

Case 254: En este caso se realiza un borrado lógico, re-estableciendo el tope en -2 como en la instancia inicial tal que al momento de volver a apilar elementos se realice una reasignación en los lugares correspondientes.

Case 255: Se muestra en pantalla el código de que la operación fue exitosa y se termina corta la recepción por entrada.

En caso que no se sepa reconozca el código de entrada no hacemos saltaremos previamente a ninguno de los casos anteriores y mostraremos “2” en el puerto de salida de la bitácora, para esperar otro código en la entrada.

Experimentación y problemas encontrados

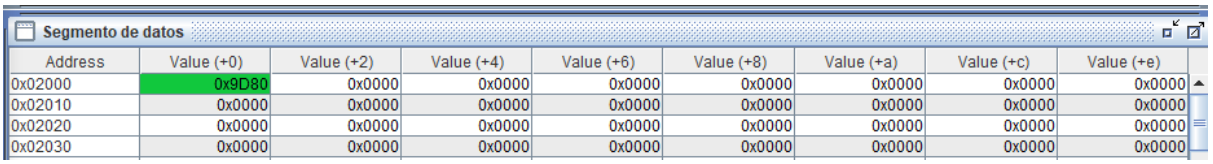
Dentro de la experimentación creo que la mayor dificultad fue aprender a manejar las estructuras manejadas en C para que tomen un significado en Assembler, el paso de estructuras de iteración así como el control de casos particulares para operaciones particulares como DIV, la cual no admite división con signo por lo que tuve que hacer una búsqueda más extensa para poder ampliar la funcionalidad de mi calculadora incorporando IDIV, operación la cual no se encuentra en la cartilla.

Conclusiones

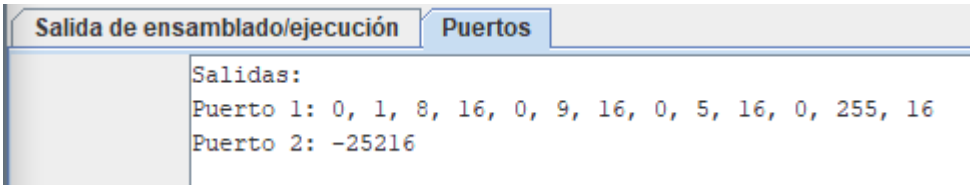
La tarea fue una buena introducción al lenguaje assembler, siento que me aportó muchos conocimientos al respecto al modo de trabajo en este lenguaje así como el manejo de los registros. Creo que la dificultad del problema estuvo muy bien regulada.

Mejoras a futuro

Una mejora a futuro sería la posibilidad de representar números mayores, se debe reconocer que la capacidad es lógicamente limitada, pero siempre sería bueno llegar más lejos. Por ejemplo, el cálculo de números mayores a 7 factorial exceden la representación y son interpretados de manera incorrecta aunque el cálculo del número sea correcto, en la siguiente imagen se muestra un ejemplo con el cálculo de 8 factorial(8!=40320 ~ 0x9D80) donde se puede ver que el cálculo almacenado en la dirección 0x02000 del número es correcto, pero al hacer el volcado de la pila la salida es -25216 por el sistema de representación interna.



Address	Value (+0)	Value (+2)	Value (+4)	Value (+6)	Value (+8)	Value (+a)	Value (+c)	Value (+e)
0x02000	0x9D80	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
0x02010	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
0x02020	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
0x02030	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000



Cuando comencé la tarea me generé la variable tope para no tener que destinar un registro para una única tarea y así tener todos los preciados registros disponibles, pero a medida que progresé en la tarea me di cuenta que simplemente podía usar el registro SI en vez de llevar ambos (SI y tope) en paralelo, creo que una mejora sería simplemente eliminar la variable tope y usar el SI siempre.

Referencias

-ArquiSim: manual de usuario.
https://eva.fing.edu.uy/mod/resource/view.php?id=61253.
-Eva y Foro del curso:
https://eva.fing.edu.uy/course/view.php?id=195&section=0