Décima Clase

Presentación

En la segunda unidad usted ha desarrollado con Assembler programas que le permitieron sumar, armar listas, ordenar números, etc. En esta unidad estudiaremos la forma utilizar secuencias de instrucciones ya confeccionadas que constituyen una subrutina, para facilitar su tarea como programador. Por ejemplo, cómo utilizar una subrutina para obtener una raíz cuadrada.

Antes de abordar las subrutinas, dedicaremos un espacio para comprender el segmentado de memoria. Como es posible que los programas posean direcciones relativas a su posición en la memoria y cómo la configuración de los segmentos por el SO, permiten que el programa pueda funcionar en cualquier lugar de la memoria

Luego, veremos cómo mediante una instrucción se llama, desde cualquier punto de un programa, a una subrutina para ser ejecutada y cómo se retorna al programa que la solicitó. En la clase siguiente, presentaremos la forma de llamar a subrutinas o módulos de un sistema operativo mediante interrupciones por software y por hardware.

A través del estudio de esta clase esperamos que usted adquiera la capacidad para:

- Comprender el direccionamiento relativo y absoluto de la memoria y la utilización de los segmentos
- Comprender la importancia y la secuencia de pasos presente en el llamado a una subrutina y las características básicas del uso de la pila y de su registro puntero: el stack pointer.

A continuación, le presentamos un detalle de los contenidos y actividades que integran esta unidad. Usted deberá ir avanzando en el estudio y profundización de los diferentes temas, realizando las lecturas requeridas y elaborando las actividades propuestas.

Contenidos y Actividades

1. Direcciones absolutas de Memoria y Direcciones relativas al segmento

1.1 Segmentación de memoria

Lectura requerida

- Ginzburg, M.; Direcciones efectivas y Registros de Segmento En su: **Assembler desde cero e interrupciones**. 3ª ed ampliada. Buenos Aires: 2010.

2. Llamada a una subrutina

2.1 Detalle del llamado a una subrutina

2.2 Instrucciones Push y pop en las subrutinas

Lectura requerida

- Ginzburg, M.; Llamado a subrutinas En su: **Assembler desde cero e interrupciones**. 3ª ed ampliada. Buenos Aires: 2010.

Lo/a invitamos a comenzar con el estudio de los contenidos que conforman esta primera unidad.

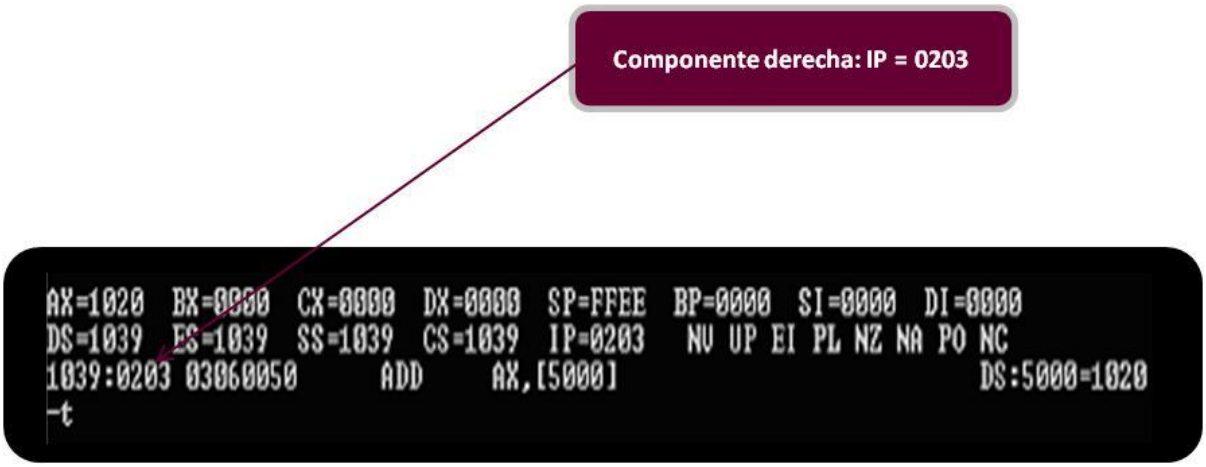
1. Direcciones absolutas de Memoria y Direcciones relativas al segmento

1.1 Segmentación de Memoria

Probablemente Ud. haya advertido que una dirección de memoria posee dos componentes numéricas separadas por dos puntos.

Por ejemplo: **1039:0203.**

Por razones didácticas, en el Trabajo Practico requerido Nº 2: Codificación y ejecución de programas en Asembler para las direcciones de las instrucciones, sólo hemos operado con la componente derecha: valor del registro IP.



En la imagen se puede observar que la instrucción ADD AX [5000], cuyo código de máquina es 03060050 tiene por dirección 1039:0203. **La componente izquierda 1039, corresponde al valor de registro CS (Code Segment).**

Por lo tanto, cualquier instrucción a ejecutar consta de dos componentes separados por dos puntos. Una provista por el registro CS y la otra por IP.

Es importante considerar el valor del registro CS ya que en las interrupciones CS se guarda en la pila junto con IP.

Luego de esta indicación lo/a invitamos a continuar con la siguiente actividad de lectura.

Lectura requerida

Ginzburg, M.; Direcciones efectivas y registros de segmento. En su: **Assembler desde cero e interrupciones**. 3ª ed. ampliada. Buenos Aires: 2010.

Guía para la lectura

- En el modelo de procesador que estamos usando en los TP, que implican que nuestra CPU está operando en modo real (como sucede cuando se enciende una PC), indicar cual es el tamaño de los segmentos y el de la memoria.
- ¿Qué diferencia existe entre los programas .EXE y .COM? ¿Cuál de estas dos clases de programas utilizamos en los trabajos prácticos?
- Indique cómo mediante el registro CS se determina el comienzo del segmento de códigos
- Suponiendo CS = 3502 e IP = 2502 indicar la dirección efectiva donde está en memoria la dirección apuntada por dichos registros.

2. Llamado a una subrutina

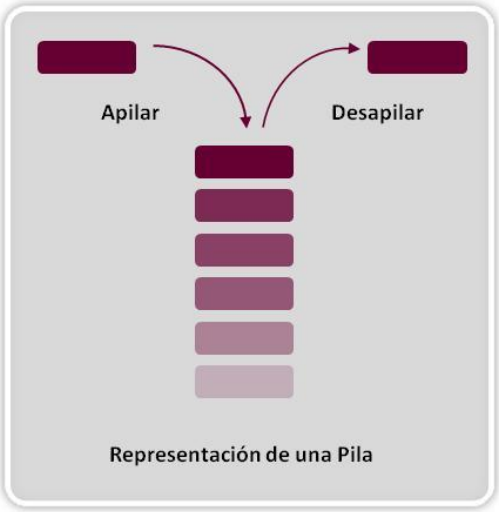
En la programación en Assembler se utiliza lo que se denomina **subrutina**. Este término alude a un procedimiento o programa construido especialmente, que se puede utilizar en distintos puntos de un programa o por diferentes programas. Por ejemplo, un procedimiento para ordenar números puede ser incorporado para ser ejecutado, en cualquier punto de un programa en ejecución. Ésta acción se denomina **"llamado"** o**"call"** a subrutina.

Se utilizan por dos razones:

1. **Para evitar incluir repetidamente el procedimiento de un programa principal. Cada vez, que se lo necesita se invoca a la función como subrutina.**
2. **Para la estructuración de programas largos en módulos más pequeños.**

Para comprender el proceso de llamado a subrutina es preciso profundizar y conocer qué son y cómo se opera con las estructuras de datos denominadas pilas.

Una **pila** o **stack** es una zona de memoria que almacena temporariamente, en forma apilada, direcciones y datos relacionados con la ejecución de subrutinas.



Representación de una Pila

Los datos almacenados tienen una restricción de acceso: sólo pueden agregarse o eliminarse por un extremo de la pila, conocido como **"cima"**. El último dato de la pila es el primero en extraerse. En ingles esto se expresa con la sigla LIFO (last in, first out). La imagen mental que lo puede ayudar a representar una pila es, justamente, una pila de platos: el ultimo en apilarse será el primero en salir.

Cuando hablamos de **Stack Pointer** (SP), también conocido como **puntero de pila**, nos referimos a un registro que proporciona la dirección del último dato que se ha almacenado en la cima de la pila. El SP es actualizado en forma automática por la UC durante la ejecución de las instrucciones que escriben o leen la pila.

Cada programa al llamar una subrutina, origina una pila en memoria principal o en la UCP. Ésta desaparece al finalizar con la última instrucción de esa subrutina.

2.1 Detalle del llamado a una subrutina

En el Assembler de Intel/AMD una subrutina es llamada desde un programa, cada vez que se la necesita, mediante la instrucción **CALL**. Cuando se ejecuta ésta instrucción, el hardware de la UC realiza los siguientes pasos: calcula la dirección de retorno, prepara el apilamiento, apila y salta a la subrutina.

A continuación detallaremos cada uno de los pasos mencionados y un ejemplo en la columna derecha para ayudar a su comprensión:

1. **Cálculo de la dirección de retorno:** la UC le suma 3 al valor que tenía el registro IP cuando se pidió la instrucción CALL porque su código de máquina ocupa 3 celdas de memoria. De este modo, el IP sirve para determinar la dirección de la instrucción que sigue a CALL en memoria.

Por ejemplo, si CALL comienza en la dirección 2433, la instrucción que le sigue debe comenzar en 2436, o sea que a la dirección 2433 que apuntaba el IP para localizar CALL se le suma 3. Así se calcula la dirección 2436, denominada ***"dirección de retorno"****, dirección de la instrucción que sigue en memoria a CALL, la cual se pasará a ejecutar cuando se retorne al programa que llamó a la subrutina, una vez que se terminen de ejecutar las instrucciones de esta subrutina llamada* por CALL.

2. **Preparación del apilamiento**: le resta 2 al valor que tiene el registro SP, para que la nueva cima de la pila se encuentre 2 celdas más arriba en la zona de memoria donde se formará la pila.

*Si **SP** tenía el valor FFEE, pasará a valer FFEC..*

3. **Apilamiento:** a partir de la dirección de la nueva cima de la pila, la UC escribirá la dirección de retorno que calculó.

Esto significa que, la pila se comenzará a formar con la dirección 2436 del ejemplo.

4. **Salto a la subrutina**: el IP tomará el valor de la dirección de la primera instrucción donde en memoria está la subrutina.

Es decir, si la instrucción es CALL 3000, la subrutina empieza en la dirección 3000 y el valor que tomará el IP (3000), apuntará a la primera instrucción de la subrutina.

Asimismo, toda subrutina llamada por **CALL** debe terminar con la instrucción **RET.** Ésta **ordena retornar** para ejecutar la instrucción que en el programa sigue a CALL.

La ejecución de RET supone las siguientes acciones por parte de la UC:

a) **Que el IP tome el valor de la dirección de retorno:** la dirección de retorno estaba guardada en la pila luego de la ejecución de CALL, su copia (siguiendo el ejemplo mencionado es de 2436) deberá pasar, de la pila (que será leída en memoria) al registro IP (que será escrito en la UCP). Entonces, la ejecución de las instrucciones por parte de la UC continuará con la instrucción del programa que sigue a CALL.

b) **Desapilamiento:** al SP se le suma 2, de manera que la pila desaparece. En nuestro caso SP vuelve al valor originario (FFEE) que tenía antes de que se llame a la subrutina, el cual es asignado por el sistema operativo como administrador de la memoria.

2.2 Instrucciones Push y Pop en las subrutinas

El programador que escribe una subrutina no tiene por qué ser la misma persona que escribe el programa que la va a llamar. Por consiguiente, los registros de la UCP que usa la subrutina pueden coincidir o no, total o parcialmente, con los registros que está usando el programa que la llama. Como esto no puede conocerse, **toda subrutina debe empezar guardando en la pila el contenido de cada uno de los registros que utiliza**, en el momento que es llamada. Esto debe ser así porque es muy factible que, alguno o todos ellos hayan sido usados por el programa en el momento en que éste llamó a la subrutina.

Infiera que cuando el programa llamó a la subrutina dejo el registro AX con el valor 0042. Si una instrucción de la subrutina como ADD AX,BX cambia el valor de AX, cuando se termine de

ejecutar la subrutina y se retorne al programa llamador, éste al querer volver a usar AX lo encontrará con otro valor que no será 0042. De la misma manera ADD AX, BX puede cambiar el valor de los flags guardados en RE.

Para evitar que la subrutina al finalizar cambie el valor de los registros que usó el programa cuando la llamo, las subrutinas comienzan con instrucciones tipo "**PUSH**". La palabra PUSH está asociada con la acción de empujar (apilar) como una moneda en un monedero con resortes, siendo que POP implica lo contrario.

La ejecución de PUSH AX guardará en la pila una copia de 0042, apilándolo sobre la dirección de retorno (2436 del ejemplo anterior), por lo que antes la UC, como en el caso de CALL, deberá restarle 2 a SP, para que la nueva cima sea FFEA (FFEC – 2 = FFEA, siguiendo el ejemplo). Igualmente, también, la subrutina deberá tener al comienzo la instrucción PUSH F (push Flags) para que se guarde en la pila una copia del registro de estado (RE) con el valor de cada flag tal cual como lo dejó el programa. Asimismo, SP luego de que se ejecute PUSHF quedará en FFE8.

Por lo tanto, **al comenzar una subrutina deberá haber una instrucción PUSH para cada registro que utilizará la subrutina.** de esta forma se guardará en la pila el valor con que lo venía usando el programa llamador, aunque algunos de esos registros el programa no los use.

Durante su ejecución, la subrutina podrá cambiar el valor de esos registros, y sus últimas instrucciones (antes de RET citada) serán del tipo "**POP**", que ordenan restaurar desde la pila hacia la UCP el valor de cada registro guardado mediante el PUSH correspondiente. Entonces, siguiendo con el ejemplo, si la subrutina empezaba con PUSH AX y PUSHF, que apilaban una copia de los contenidos de AX y RE, deberá terminar con POPF y POP AX (y al final RET) para que esos contenidos vuelvan a AX y RE, para que luego de RET se retorne al programa llamador (en la instrucción que sigue a CALL), *y encuentre a todos los registros como los había dejado antes de que se ejecute* CALL.

De manera semejante a RET, cada POP desapila el registro involucrado, cuyo contenido vuelve a la UCP, y luego le suma 2 al valor del SP, para que la cima de la pila descienda.

Es importante profundizar sobre la implementación y concreción de las subrutinas. Es por ello que lo/a invitamos a realizar la siguiente actividad de lectura.

Lectura requerida

Ginzburg, M.; Llamado a subrutinas En su: **Assembler desde cero e interrupciones**. 3ª ed. Buenos Aires, 2010 p. 54 a 71.

Guía para la lectura

- ¿Cuáles son los pasos para el proceso de llamado a una subrutina?
- ¿Cuál es la función de una pila? ¿Qué se indica con las instrucciones "PUSH" y "POP"?
- ¿Qué indica el Stack Pointer?
- ¿Qué hacen las instrucciones CALL, PUSH, POP y RET?
- Algunos conceptos que debemos tener en cuenta:

- **Anidamiento de subrutinas**: una subrutina puede llamar a otra, ésta a una tercera y, así, sucesivamente. Este proceso se denomina "anidamiento de subrutinas". Cuando una subrutina llamada termina de ejecutarse, se retorna a la subrutina que la llamó, ésta a la que la llamó, etc. Los retornos se dan de modo inverso al de las llamadas, es decir el último retorno será hacia el programa principal, que efectuó el primer llamado. Puede ser factible que la subrutina llamada, a su vez, tenga una instrucción como CALL 6500 que llama a otra subrutina. En ese caso el SP (supuesto en FFE8 luego de PUSHF) pasará a valer FFE6, y en la pila se guardará la dirección de la instrucción que sigue a CALL 6500. Cuando termine la ejecución de la nueva subrutina llamada (que tendrá sus PUSH y POP), su instrucción RET hará que la primera subrutina citada prosiga en la instrucción que sigue a CALL 6500 y el SP volverá a valer FFE8**.**
- **Pasaje de Parámetros:** Se denomina pasaje de parámetros al pase de datos del programa a la subrutina y de resultados de la subrutina al programa. Refiere a la necesidad de establecer dónde el programa debe dejar los datos que procesará la subrutina que llamó y dónde la subrutina dejará los resultados esperados. Puede ser en la pila o en registros de la UCP. Dichos datos serán usados por las instrucciones que siguen a la instrucción CALL, que llamó a la subrutina.