

Programación en lenguaje ensamblador. Instrucciones básicas. Formatos. Direccionamientos.

TEMA 33

ABACUS NT

Índice

- 1. Introducción**
- 2. Características del lenguaje ensamblador**
- 3. Instrucciones básicas**
 - 3.1. Tipos de instrucciones**
 - 3.1.1. Instrucciones de transferencia de datos:
 - 3.1.2. Instrucciones aritméticas:
 - 3.1.3. Instrucciones de ruptura de secuencia:
 - 3.1.4. Instrucciones de control de la CPU:
 - 3.1.5. Operadores
- 4. Formatos de las instrucciones**
 - 4.1. Partes constituyentes de una instrucción**
 - 4.2. Formatos de código de operación, campos de operando e instrucción**
- 5. Modos de direccionamiento**
 - 5.1. Direccionamiento inmediato.**
 - 5.2. Direccionamiento directo absoluto.**
 - 5.3. Direccionamiento directo relativo.**
 - 5.3.1. Direccionamiento relativo al contador de programa PC.
 - 5.3.2. Direccionamiento directo relativo a registro base.
 - 5.3.3. Direccionamiento directo relativo a registro índice.
 - 5.3.4. Direccionamiento a pila.
 - 5.4. Direccionamiento indirecto.**
 - 5.5. Direccionamiento implícito.**
- 6. Conclusión**
 - 6.1. Relación del tema con el sistema educativo actual**
- 7. Bibliografía**

1. Introducción

Se denominan lenguajes de bajo nivel porque dependen de la arquitectura del procesador en el que queremos ejecutar el programa y porque no disponen de sentencias con una estructura lógica que faciliten la programación y la comprensión del código para el programador, sino que están formados por una lista de instrucciones específicas de una arquitectura.

Podemos distinguir entre dos lenguajes:

Lenguaje máquina

Lenguaje que puede interpretar y ejecutar un procesador determinado. Este lenguaje está formado por instrucciones codificadas en binario (0 y 1). Es generado por un compilador a partir de las especificaciones de otro lenguaje simbólico o de alto nivel. Es muy difícil de entender para el programador y sería muy fácil cometer errores si se tuviera que codificar.

Lenguaje ensamblador

Lenguaje simbólico que se ha definido para que se puedan escribir programas con una sintaxis próxima al lenguaje de máquina, pero sin tener que escribir el código en binario, sino utilizando una serie de mnemónicos más fáciles de entender para el programador. Para ejecutar estos programas también es necesario un proceso de traducción, generalmente denominado ensamblaje, pero más sencillo que en los lenguajes de alto nivel.

Se denomina **mnemónico** o **mnemotécnico** a una instrucción en lenguaje ensamblador que sustituye a un equivalente en lenguaje máquina.

2. Características del lenguaje ensamblador

El código escrito en lenguaje ensamblador posee una cierta dificultad de ser entendido directamente por un ser humano ya que su estructura se acerca más bien al lenguaje máquina, es decir, lenguaje de bajo nivel.

El lenguaje ensamblador es difícilmente portable, es decir, un código escrito para un Microprocesador, suele necesitar ser modificado, muchas veces en su totalidad para poder ser usado en otra máquina distinta, aun con el mismo Microprocesador, solo pueden ser reutilizados secciones especiales del código programado.

Los programas hechos en lenguaje ensamblador, al ser programado directamente sobre Hardware, son generalmente más rápidos y consumen menos recursos del sistema (memoria RAM y ROM). Al programar cuidadosamente en lenguaje ensamblador se pueden crear programas que se ejecutan más rápidamente y ocupan menos espacio que con lenguajes de alto nivel.

Con el lenguaje ensamblador se tiene un control muy preciso de las tareas realizadas por un Microprocesador por lo que se pueden crear segmentos de código difíciles de programar en un lenguaje de alto nivel.

También se puede controlar el tiempo en que tarda una Rutina en ejecutarse, e impedir que se interrumpa durante su ejecución.

El lenguaje ensamblador es un código estructurado y jerárquico desarrollado sobre un archivo de programación (.asm), en el cual pueden existir varios programas, macros o rutinas que pueden ser llamados entre sí.

3. Instrucciones básicas

En Informática, una *instrucción* es un conjunto de símbolos que el ordenador es capaz de interpretar con objeto de realizar las acciones que dichos símbolos le indican.

Las instrucciones en lenguaje ensamblador son muy elementales; indican acciones muy concretas, nombrando incluso a los elementos que tienen los datos o que guardan los resultados, como son los registros, posiciones de memoria, periféricos, etc.

3.1. Tipos de instrucciones

Podemos clasificar las instrucciones, según la acción que ordenan, en los siguientes tipos:

3.1.1. Instrucciones de transferencia de datos:

Indican a la CPU que debe ordenar la transferencia de datos entre algún elemento interno del ordenador (registros, posiciones de memoria, etc.) y otro elemento interno o externo (periféricos).

En la arquitectura 8086:

MOV dest,src	Copia el contenido del operando fuente (src) en el destino (dest).
PUSH src	Pone el valor en el tope del stack.
POP dest	Retira el valor del tope del stack poniéndolo en el lugar indicado.
XCHG reg,{reg mem}	Intercambia ambos valores.
IN {AL AX},{DX inmed (1 byte)}	Pone en el acumulador el valor hallado en el port indicado.
OUT {DX inmed (1 byte)},{AL AX}	Pone en el port indicado el valor del acumulador.
XLAT	Realiza una operación de traducción de un código de un byte a otro código de un byte mediante una tabla.
LEA reg,mem	Almacena la dirección efectiva del operando de memoria en un registro.
LDS reg,mem32	Operación: reg <- [mem], DS <- [mem+2]
LES reg,mem32	Operación: reg <- [mem], ES <- [mem+2]
LAHF	Copia en el registro AH la imagen de los ocho bits menos significativos del registro de indicadores.
SAHF	Almacena en los ocho bits menos significativos del registro de indicadores el valor del registro AH.
PUSHF	Almacena los flags en la pila.
POPF	Pone en los flags el valor que hay en la pila.

3.1.2. Instrucciones aritméticas:

Indican a la CPU la operación que debe realizar. Pueden ser aritméticas (sumar, restar, etc.), lógicas (O, Y, NO, etc.), de comparación y de rotación y desplazamiento de los bits de un registro o posición de memoria hacia la izquierda o hacia la derecha, etc.

En la arquitectura 8086:

ADD dest,src	Operación: dest <- dest + src.
ADC dest,src	Operación: dest <- dest + src + CF.
SUB dest,src	Operación: dest <- dest - src.
SBB dest,src	Operación: dest <- dest - src - CF.
CMP dest,src	Operación: dest - src (sólo afecta flags).
INC dest	Operación: dest <- dest + 1 (no afecta CF).
DEC dest	Operación: dest <- dest - 1 (no afecta CF).
NEG dest	Operación: dest <- -dest.
DAA	Corrige el resultado de una suma de dos valores BCD empaquetados.
DAS	Igual que DAA pero para resta.
AAA	Lo mismo que DAA para números BCD desempaquetados.
AAS	Lo mismo que DAS para números BCD desempaquetados.
AAD	Convierte AH:AL en BCD desempaquetado a AL en binario.
AAM	Convierte AL en binario a AH:AL en BCD desempaquetado.
MUL {reg8 mem8}	Realiza una multiplicación con operandos no signados de 8 por 8 bits.
MUL {reg16 mem16}	Realiza una multiplicación con operandos no signados de 16 por 16 bits.
IMUL {reg8 mem8}	Realiza una multiplicación con operandos con signo de 8 por 8 bits.
IMUL {reg16 mem16}	Realiza una multiplicación con operandos con signo de 16 por 16 bits.
CBW	Extiende el signo de AL en AX. No se afectan los flags.
CWD	Extiende el signo de AX en DX:AX. No se afectan flags.

3.1.3. Instrucciones de ruptura de secuencia:

Indican a la CPU que la siguiente instrucción que debe ejecutarse no es la que corresponde según la secuencia física del programa, sino que debe realizar un salto a otro punto del mismo programa. Estos saltos pueden ser incondicionales o condicionales, según se realicen siempre o sólo cuando de cumpla alguna condición, que, normalmente, dependerá de algún bit del registro de estado de la CPU.

En la arquitectura 8086:

JMP label	Saltar hacia la dirección label.
-----------	----------------------------------

CALL label	Ir al procedimiento cuyo inicio es label. Para llamadas dentro del mismo segmento equivale a PUSH IP: JMP label, mientras que para llamadas entre segmentos equivale a PUSH CS: PUSH IP: JMP label.
RET	Retorno de procedimiento.
RET inmed	Retorno de procedimiento y SP <- SP + inmed.
RETN [inmed]	En el mismo segmento de código. Equivale a POP IP [:SP <- SP + inmed].
RETF [inmed]	En otro segmento de código. Equivale a POP IP: POP CS [:SP <- SP + inmed]
JL etiqueta/JNGE etiqueta	Saltar a etiqueta si es menor.
JLE etiqueta/JNG etiqueta	Saltar a etiqueta si es menor o igual.
JE etiqueta	Saltar a etiqueta si es igual.
JNE etiqueta	Saltar a etiqueta si es distinto.
JGE etiqueta/JNL etiqueta	Saltar a etiqueta si es mayor o igual.
JG etiqueta/JNLE etiqueta	Saltar a etiqueta si es mayor.
JB etiqueta/JNAE etiqueta	Saltar a etiqueta si es menor.
JBE etiqueta/JNA etiqueta	Saltar a etiqueta si es menor o igual.
JE etiqueta	Saltar a etiqueta si es igual.
JNE etiqueta	Saltar a etiqueta si es distinto.
JAE etiqueta/JNB etiqueta	Saltar a etiqueta si es mayor o igual.
JA etiqueta/JNBE etiqueta	Saltar a etiqueta si es mayor.
JC label	Saltar si hubo arrastre/préstamo (CF = 1).
JNC label	Saltar si no hubo arrastre/préstamo (CF = 0).
JZ label	Saltar si el resultado es cero (ZF = 1).
JNZ label	Saltar si el resultado no es cero (ZF = 0).
JS label	Saltar si el signo es negativo (SF = 1).
JNS label	Saltar si el signo es positivo (SF = 0).
JP/JPE label	Saltar si la paridad es par (PF = 1).
JNP/JPO label	Saltar si la paridad es impar (PF = 0).
LOOP label	Operación: CX <- CX-1. Saltar a label si CX <> 0.
LOOPZ/LOOPE label	Operación: CX <- CX-1. Saltar a label si CX <> 0 y ZF = 1.
LOOPNZ/LOOPNE label	Operación: CX <- CX-1. Saltar a label si CX <> 0 y ZF = 0.
JCXZ label	Operación: Salta a label si CX = 0.
INT número	Salva los flags en la pila, hace TF=IF=0 y ejecuta la interrupción con el número indicado.
INTO	Interrupción condicional. Si OF = 1, hace INT 4.
IRET	Retorno de interrupción. Restaura los indicadores del stack.

3.1.4. Instrucciones de control de la CPU:

Modifican el contenido del registro de estado o pueden hacer que la CPU se detenga en la interpretación de instrucciones hasta nueva orden, que se dará mediante algún dispositivo externo a ella.

En la arquitectura 8086:

CLC	CF <- 0.
STC	CF <- 1.
CMC	CF <- 1 - CF.
NOP	No hace nada.
CLD	DF <- 0 (Dirección ascendente).
STD	DF <- 1 (Dirección descendente).
CLI	IF <- 0 (Deshabilita interrupciones enmascarables).
STI	IF <- 1 (Habilita interrupciones enmascarables).
HLT	Detiene la ejecución del procesador hasta que llegue una interrupción externa.
WAIT	Detiene la ejecución del procesador hasta que se active el pin TEST del mismo.
LOCK	Prefijo de instrucción que activa el pin LOCK del procesador.

3.1.5. Operadores

Operadores aritméticos

+, -, *, /, MOD (resto de la división).

Operadores lógicos

AND, OR, XOR, NOT, SHR, SHL.

Operadores relacionales:

EQ: Igual a.

NE: Distinto de.

LT: Menor que.

GT: Mayor que.

LE: Menor o igual a.

GE: Mayor o igual a.

Otros Operadores

Además de los señalados, pueden existir:

operadores analíticos que descomponen operandos que representan direcciones de memoria en sus componentes.

Operadores sintéticos

Componen operandos de direcciones de memoria a partir de sus componentes.

Operadores de macros

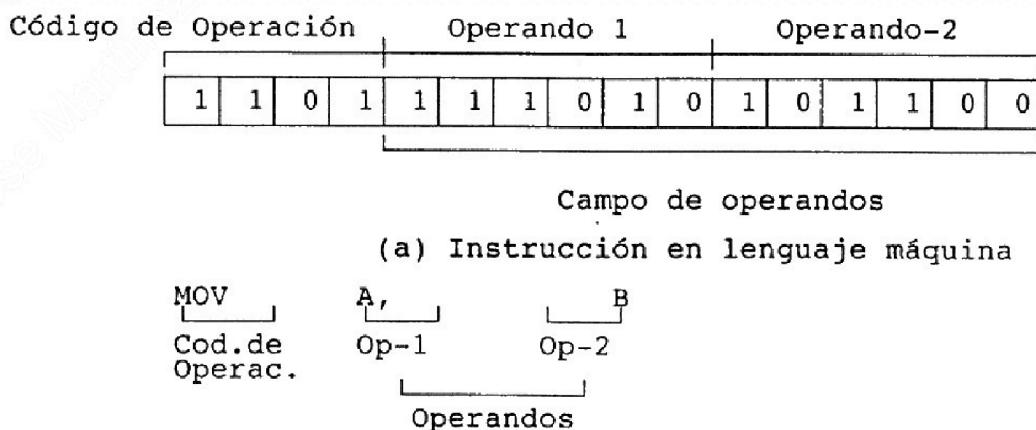
Son operadores que se utilizan en las definiciones de macros.

4. Formatos de las instrucciones

4.1. Partes constituyentes de una instrucción

En toda instrucción podemos diferenciar dos partes o campos. Una de estas partes indica a la CPU la operación que debe ejecutar, y se llama *campo de código de operación*; la otra le indica de dónde debe tomar los datos y dónde poner los resultados, y se denomina *campo de operandos*.

En la Figura se muestra una instrucción indicando el campo de código de operación y los campos de operandos. Como se puede apreciar, la parte (a) de la figura representa una instrucción en el lenguaje máquina, es decir, un conjunto de unos y ceros que la máquina es capaz de interpretar de forma sencilla, pero que al programador le resulta más complicado hacerlo. La parte (b) de la figura muestra cómo sería la misma instrucción, pero ahora en lenguaje ensamblador. En esta forma de representación el programador la entiende fácilmente, pero para que la máquina sea capaz de ejecutarla deberá traducirla primero a lenguaje máquina.



Campo código de operación:

Este campo es obligatorio en todas las instrucciones, ya que, indica a la CPU precisamente la operación que debe realizar. Cada instrucción u operación estará codificada, en lenguaje máquina, en una serie de ceros y unos de forma que no haya dos instrucciones diferentes que tengan el mismo código.

El número de bits que se emplean en el código puede ser idéntico en todas las instrucciones de una determinada máquina, en cuyo caso se dice que esa máquina tiene un código de operación fijo; o puede variar, en cuyo caso se dice que la máquina tiene un código de operación variable o expandido.

El código de operación suele estar ocupando los bits más significativos de la palabra, los situados más a la izquierda, como se ve en la Figura (a). Si una instrucción ocupa más de una palabra, el código de operación estará en la primera palabra que lea la CPU, para que ésta pueda ir interpretando la instrucción y saber si después hay más palabras pertenecientes a la misma instrucción o no.

También en lenguaje ensamblador se indica el campo de código de operación en primer término.

A los caracteres que forman el código de operación en lenguaje ensamblador se les denomina *mnemotécnicos*.

Campo de operandos:

Además del código de operación, en una instrucción podemos tener otro campo, como ya se ha indicado anteriormente, llamado campo de operandos. Este campo no está presente en todas las instrucciones, ya que hay algunas que no emplean datos y otras en la que la localización de los mismos está implícita en el propio código de operación. La longitud de este campo es normalmente variable, dependiendo del número de operandos que utilice la instrucción y de la forma que se indique a la CPU el acceso a los mismos. De existir campos de operandos, se codifica la dirección de los mismos a continuación del campo de código de operación.

En lenguaje ensamblador se emplean símbolos alfanuméricos para representar los operandos. Pueden ser nombres de variables que representen posiciones de memoria, registros, etc., utilizando una representación decimal, octal, hexadecimal, etc.

4.2. Formatos de código de operación, campos de operando e instrucción

Como ya se ha visto en los apartados anteriores, una instrucción puede tener, además del código de operación, un número variable de operandos; pueden ser varios, sólo uno o ninguno. Esto hace que una misma máquina pueda tener instrucciones de diferentes formas y longitudes.

Las variantes posibles en cuanto a las longitudes de código de operación, campos de operando e instrucción son:

Longitud de instrucción fija, código de operación extendido.

Longitud de instrucción variable, código de operación fijo.

Longitud de instrucción variable, código de operación extendido.

A continuación, se estudiarán los diferentes formatos que podemos encontrar en diferentes máquinas, dependiendo del número de operandos o direcciones que utilicen en la instrucción.

Formato de cuatro direcciones.

En la actualidad no es frecuente encontrar máquinas con este formato de instrucción. Dos direcciones indican los operandos con los que operará la CPU, otra dirección le indicará dónde debe guardar el resultado, y la otra le dirá dónde se encuentra la siguiente instrucción a ejecutar.

Estas máquinas no utilizan registro contador de programa, ni son necesarias las instrucciones de salto específicas. El problema de este formato es la gran cantidad de bits que necesitan las instrucciones para poder codificar tantas direcciones.

Formato de tres direcciones.

Ante la necesidad de simplificar las máquinas para abaratar costes y aumentar el rendimiento, se fue reduciendo la longitud de las instrucciones. Una manera de conseguirlo es eliminando información en las mismas.

Un avance muy importante en este sentido es la aparición del contador de programa, registro que guarda la dirección de la siguiente instrucción a ejecutar, por lo que ya no es necesario incluirla en la instrucción. De esta manera se pierde algo de flexibilidad en el secuenciamiento de las instrucciones, ya que la dirección se va calculando a medida que se ejecuta el programa de forma independiente a las instrucciones. Se hace necesario el uso de instrucciones de ruptura de secuencia, pero se ahorran bits en cada instrucción.

Formato de dos direcciones.

El formato anterior aún ocupa muchas posiciones. En consecuencia, se optó por eliminar la dirección que indica dónde se guarda el resultado, empleándose para este fin uno de los dos operandos especificados en la instrucción. El operando que recoge el resultado se llama destino; el otro se llama operando fuente. El operando destino puede utilizarse también para aportar el dato que se necesita para realizar la operación indicada en el código de operación junto con el operando fuente. Indudablemente, después de ejecutada la instrucción el contenido del operando destino se habrá modificado, perdiéndose el contenido anterior.

Este formato tiene variantes.

Una consistiría en emplear un registro para especificar uno de los operandos; de esta forma se reduce la longitud total de la instrucción, ya que para indicar un registro de la CPU son necesarios muchos menos bits que para una dirección de memoria.

Por ejemplo, tomando la misma capacidad de memoria que en los apartados anteriores, vimos que para indicar una dirección de memoria eran necesarios 16 bits. Si suponemos que la CPU posee 8 registros de trabajo, serían suficientes tres bits para hacer referencia a cada uno de ellos. Esta variante del formato de dos direcciones también se denomina formato de una dirección y media.

Otra variante consistiría en especificar como operandos dos registros, con lo que se reduciría aún más la longitud de la instrucción.

Formato de una dirección.

En este formato se indica el registro de forma implícita en el código de operación, por lo que en el campo de operandos queda sólo la dirección de un operando. El otro operando, que recibe el resultado, se supone que es un determinado registro de trabajo de la CPU, denominado generalmente acumulador.

Formato de cero direcciones.

Existen máquinas que utilizan para guardar datos una estructura denominada pila. La pila no es más que un conjunto de posiciones consecutivas de memoria, y para acceder a las mismas se utiliza un registro llamado "apuntador de pila", que guarda la dirección de la cabecera de la pila, o elemento al que vamos a acceder. Cuando almacenemos un nuevo dato, este registro se incrementará y, cuando saquemos un dato, se decrementará para indicar el siguiente elemento.

Con esta estructura sólo hace falta especificar la operación, ya que la CPU supone que los operandos están guardados en la pila, donde los busca con la ayuda del apuntador de pila, y una vez realizada la operación supone que el resultado debe guardarlo en la misma pila.

5. Modos de direccionamiento

Las diferentes formas de indicar en el campo de operandos dónde se encuentran los datos que la CPU tendrá que procesar se denominan "modos de direccionamiento".

5.1. Direccionamiento inmediato.

El direccionamiento se llama inmediato cuando el objeto, en este caso un operando, se encuentra contenido en la propia instrucción.: Hay máquinas que permiten distintos tamaños de operandos inmediatos. Por ejemplo, el VAX permite inmediatos de 6, 8, 16, 32 y 64 bits. Con ello se pretende reducir la memoria necesaria, adaptando la instrucción al tamaño de dato deseado.

5.2. Direccionamiento directo absoluto.

Un direccionamiento se llama directo, en contraposición con el indirecto, cuando expresa la dirección real del objeto. Por otro lado, el direccionamiento absoluto indica que la instrucción contiene una dirección efectiva sin compactar. Por tanto, el direccionamiento directo absoluto indica que la instrucción contiene la dirección real, sin compactar, del objeto.

5.3. Direccionamiento directo relativo.

En el direccionamiento directo relativo la instrucción no contiene la dirección del objeto, sino un desplazamiento D sobre una dirección marcada por un puntero. La dirección se calcula sumando el desplazamiento D al puntero de referencia, que suele estar almacenado en un registro.

Este tipo de direccionamiento suele necesitar menos bits que el directo absoluto, puesto que el desplazamiento D puede tener bastantes menos bits que los que exige el mapa de direcciones. Notemos que el registro que sirve de puntero puede ser del tamaño deseado. Por ello, este direccionamiento es más compacto, pero requiere, en contrapartida, realizar la operación de suma del puntero más el desplazamiento.

La mayoría de los computadores permiten desplazamientos positivos y negativos. De esta forma, se puede alcanzar una zona de memoria principal alrededor de la posición marcada por el puntero.

El interés de este tipo de direccionamiento se fundamenta en que es la base del código reentrantе y reubicable, puesto que permite cambiar las direcciones de datos y de bifurcaciones sin más que cambiar el contenido de un registro. Además, algunas técnicas de protección de memoria también residen en este direccionamiento. Finalmente, también permite recorrer de forma eficaz las estructuras de datos.

Existen distintas posibilidades en cuanto al registro que se emplea como puntero y en cuanto al tratamiento que sufre este último.

5.3.1. Direccionamiento relativo al contador de programa PC.

En el direccionamiento relativo a contador de programa, como su nombre indica, el puntero empleado es el contador de programa PC, esto es, el registro que almacena la dirección de la instrucción que se va a ejecutar.

Este tipo de direccionamiento está especialmente indicado para alcanzar instrucciones próximas a la que se está ejecutando, por ejemplo, para hacer bifurcaciones que permitan construir bucles. Dado que la mayoría de los bucles son muy cortos, pues contienen unas pocas instrucciones, un desplazamiento de 1 octeto o menor es perfectamente adecuado, permitiendo un código muy compacto (esto es, unas instrucciones muy cortas).

5.3.2. Direccionamiento directo relativo a registro base.

El direccionamiento relativo a registro base emplea como puntero un registro base RB.

Generalmente, los computadores disponen de varios registros que pueden actuar como base, ya sean éstos los registros generales o bien registros específicos para ese fin.

La instrucción deberá contener la identificación del registro que se emplea como base, así como el desplazamiento. Para obtener la dirección se ha de seleccionar el registro base RB y sumarle el desplazamiento D.

Este direccionamiento se diferencia del relativo a índice en que el registro de base no suele modificarse y el de índice sí. El direccionamiento es muy conveniente, por ejemplo, cuando se dispone de una zona de datos. Cargando en el registro base la primera posición de esta zona, se pueden alcanzar los distintos datos sin más que conocer su posición relativa dentro de la zona de datos. Un pequeño desplazamiento será suficiente para recoger esta posición relativa.

5.3.3. Direccionamiento directo relativo a registro índice.

El direccionamiento relativo al registro índice es una variación del anterior. En este caso, el registro puntero (que llamamos registro índice RI) es modificado para ir recorriendo los elementos de una tabla o vector. En efecto, si cada elemento de la tabla ocupa 1 palabra de la memoria y, cada vez que se emplea, el registro índice se incrementa en 1, se van obteniendo las direcciones de los elementos sucesivos de la tabla.

5.3.4. Direccionamiento a pila.

El direccionamiento a pila es un caso particular de direccionamiento relativo, muy empleado en los microprocesadores y minicomputadoras.

La máquina deberá disponer de uno o varios registros SP, que realicen la función de puntero de la pila. Cada uno de ellos contiene la dirección de la posición de memoria principal que actúa de cabecera de pila.

- Para insertar un nuevo elemento, se realiza un direccionamiento relativo al registro SP con preautoincremento.
- Para extraer un elemento, se debe hacer postautodecremento.

El direccionamiento a pila permite instrucciones muy compactas, puesto que, si sólo se dispone de un único registro SP, la instrucción no requiere ninguna información de dirección.

5.4. Direccionamiento indirecto.

El direccionamiento indirecto comienza con un direccionamiento directo (ya sea absoluto o relativo), pero se diferencia de aquél en que la dirección obtenida no apunta al objeto deseado sino su dirección. Por tanto, para obtener el objeto deseado se requiere un acceso adicional a memoria principal.

5.5. Direccionamiento implícito.

En el direccionamiento implícito, la instrucción no contiene información sobre la ubicación del objeto, porque éste está en un lugar predeterminado (registro o posición de memoria).

La ventaja del direccionamiento implícito es que no ocupa espacio en la instrucción.

6. Conclusión

A modo de síntesis, se podría destacar el éxito en el sector productivo de los SGBD, cuya utilidad hoy en día, ya nadie cuestiona. Estos ofrecen, en su mayoría, lenguajes de manipulación de datos declarativos, con los que los programadores de aplicaciones pueden abstraerse de cómo se realizan las consultas, e incluso, de cómo se optimizan. En este sentido cabe destacar SQL, como el lenguaje más utilizado, aunque también merece fijarse en el surgimiento de nuevas propuestas, a nivel comercial, de nuevos lenguajes y la existencia de herramientas de mapeo, como los ORM, para mapear SQL con lenguajes orientados a objetos.

6.1. Relación del tema con el sistema educativo actual

Este tema puede ser desarrollado en los siguientes módulos formativos (para atribución docente de PES)

- GS – DAW - DAM –Programación de Servicios y Procesos (PES)

7. Bibliografía

- Introduction to Java Programming and Data Structures, Comprehensive Version. Y. Daniel Liang. Pearson, 11^a edición. 2017.
- Java 9. Francisco Javier Moldes Teo. Anaya. 2017.
- Introducción a la computación. J. Glenn Brookshear, Pearson, 11^º edición. 2012
- Fundamentos de programación. Algoritmos, estructuras de datos y objetos. Luis Joyanes Aguilar, McGraw-Hill, 4^º edición. 2008.
- Langsam, Augenstein y Tanembaum: “Estructuras de Datos con C y C++”, Prentice-Hall 1997
- Prieto A., Lloris A. y Torres J.C.: Introducción a la Informática, 4^a ed (2006) McGraw-Hill
- Lenguajes de programación, principios y práctica. 2^a Ed (2004). Kenneth C.Louden

<https://sites.google.com/site/portafoliocarlosmacallums/unidad-i/lenguajeensamblador> (2020)

[https://www.exabyteinformatica.com/uoc/Informatica/Estructura_de_computadores/Estructura_de_computadores_\(Modulo_6\).pdf](https://www.exabyteinformatica.com/uoc/Informatica/Estructura_de_computadores/Estructura_de_computadores_(Modulo_6).pdf) (2020)

<https://www.alpertron.com.ar/INST8088.HTM> (2020)