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

Introducción

La programación en lenguaje ensamblador representa una de las formas más directas de interacción entre el ser humano y la arquitectura interna de un sistema informático. A diferencia de los lenguajes de alto nivel, que abstraen los detalles del hardware, el ensamblador permite un control preciso sobre los recursos del procesador, la gestión de la memoria y la ejecución de instrucciones específicas. Por ello, continúa siendo un conocimiento esencial en ámbitos donde el rendimiento, la eficiencia y la optimización de recursos son aspectos críticos, como en sistemas embebidos, desarrollo de firmware, drivers o entornos de tiempo real.

Aunque en muchos desarrollos actuales predominan los lenguajes de alto nivel, el lenguaje ensamblador desempeña un papel insustituible en el diagnóstico de bajo nivel, el análisis forense de software, la ingeniería inversa y la programación de sistemas críticos. Este lenguaje, adaptado a cada familia de procesadores, permite a los profesionales comprender la lógica de ejecución máquina a máquina, manipular registros, gestionar interrupciones y definir la estructura de datos con precisión milimétrica. El dominio de sus instrucciones básicas, sus formatos de codificación y los distintos modos de direccionamiento es esencial para comprender la arquitectura del procesador y optimizar algoritmos en condiciones de máxima restricción.

En el entorno formativo de la Formación Profesional, introducir al alumnado en la programación en ensamblador no solo proporciona una visión técnica profunda del funcionamiento de los sistemas, sino que también fortalece competencias transversales como la lógica, la precisión, el razonamiento algorítmico y la capacidad de optimización. Es, además, una herramienta didáctica valiosa para reforzar el conocimiento de estructuras internas, tales como pilas, registros, flags y buses de datos, así como para fomentar una comprensión real de cómo se ejecutan las instrucciones a nivel físico.

En este contexto, la LOMLOE y la Ley Orgánica 3/2022 de Ordenación e Integración de la Formación Profesional refuerzan la importancia del desarrollo de las competencias del alumnado para mejorar su potencial de empleabilidad. La inclusión de contenidos como la programación en ensamblador potencia el perfil técnico del alumnado y su capacidad de adaptación a entornos profesionales que demandan precisión, control y especialización.

La comprensión de las instrucciones básicas, los formatos de codificación y los distintos modos de direccionamiento en lenguaje ensamblador constituye, por tanto, un conocimiento clave en el campo de los sistemas y aplicaciones informáticas, tanto por su valor técnico como por su

impacto formativo. A continuación, se profundizará en estos aspectos esenciales para su correcta aplicación y desarrollo profesional.

Programación en lenguaje ensamblador. Instrucciones básicas

Fundamentos del lenguaje ensamblador

El lenguaje ensamblador, también conocido como assembly, es una representación simbólica del lenguaje máquina que permite al programador interactuar directamente con la arquitectura del procesador. A diferencia de los lenguajes de alto nivel, el ensamblador no requiere de un compilador, sino de un ensamblador que traduce mnemónicos y operandos en instrucciones binarizadas, específicas para cada arquitectura de CPU, como x86, ARM, MIPS o RISC-V.

Cada instrucción del lenguaje ensamblador corresponde de forma unívoca con una operación de bajo nivel ejecutable por la Unidad de Control del procesador. Por ello, la programación en ensamblador exige una comprensión profunda de los registros, el sistema de memoria, el conjunto de instrucciones y la gestión del flujo de control interno. Su ventaja principal radica en el control total sobre el hardware y la capacidad de optimizar la ejecución en términos de velocidad y uso de recursos.

En el contexto de los sistemas informáticos y de la ingeniería del software, el uso de ensamblador es habitual en los niveles más bajos del sistema operativo, en BIOS/UEFI, microcontroladores, controladores de dispositivos, desarrollo de rutinas críticas de rendimiento o en ingeniería inversa.

Clasificación de las instrucciones básicas

El conjunto de instrucciones (ISA, Instruction Set Architecture) define el repertorio de operaciones que el procesador puede ejecutar. Estas instrucciones se dividen tradicionalmente en diferentes categorías funcionales:

Instrucciones de transferencia de datos

Permiten mover información entre registros, entre registros y memoria o entre dispositivos de entrada/salida. Las más representativas son:

- **MOV**: copia datos de un origen a un destino.
- **PUSH** y **POP**: gestionan la pila del sistema.
- **IN** y **OUT**: operaciones de entrada/salida con puertos.

Estas instrucciones son fundamentales para el flujo de datos en cualquier programa, facilitando el intercambio entre los elementos funcionales del sistema.

Instrucciones aritméticas y lógicas

Ejecutan operaciones matemáticas o booleanas sobre operandos. Incluyen:

- **ADD, SUB, INC, DEC** para operaciones aritméticas básicas.
- **MUL, DIV** para operaciones más complejas, con versiones para operandos con y sin signo.
- **AND, OR, XOR, NOT** para operaciones lógicas.
- **SHL, SHR, SAR, ROL, ROR** para desplazamientos y rotaciones de bits.

Estas instrucciones modifican, además de los operandos, los registros de estado o flags, utilizados para la toma de decisiones en instrucciones de control.

Instrucciones de control de flujo

Dirigen la ejecución del programa según condiciones específicas:

- **JMP**: salto incondicional a otra parte del programa.
- **JE, JNE, JG, JL, JGE, JLE**: saltos condicionales basados en el estado de los flags.
- **CALL y RET**: llamadas y retornos de subrutinas.
- **LOOP**: instrucción especializada para bucles con contador.

Estas instrucciones permiten estructurar el programa en rutinas reutilizables, bucles y condiciones, facilitando una lógica de control compleja a pesar del bajo nivel de abstracción.

Instrucciones de manipulación de bits y cadenas

Algunos procesadores incluyen instrucciones específicas para manipular cadenas de caracteres o modificar bits individuales:

- **STOS, LODS, SCAS, MOVS**: para manipulación de cadenas (en x86).
- **BT, BSF, BSR, BTS, BTR, BTC**: para operaciones de bit a bit.

Estas instrucciones permiten realizar operaciones altamente optimizadas, muy útiles en criptografía, gráficos, compresión o comunicaciones de bajo nivel.

Uso de registros y memoria

La programación en ensamblador requiere del uso eficiente de los registros del procesador. En la arquitectura x86, por ejemplo, los registros generales (AX, BX, CX, DX) se subdividen en registros de propósito específico y fragmentos parciales (AL, AH). En arquitecturas más modernas como ARM, los registros son uniformes y suelen identificarse con nomenclaturas como R0 a R15.

La interacción con la memoria se produce mediante modos de direccionamiento que se tratarán más adelante, pero es importante destacar que las operaciones con memoria son más lentas que las que utilizan registros, por lo que una buena práctica en ensamblador es minimizar el acceso a memoria.

Además, se hace uso del *stack pointer* y *base pointer* para gestionar la pila, muy relevantes en la creación de subrutinas y en la gestión del contexto de ejecución.

Ejemplo ilustrativo

A modo de ejemplo, el siguiente fragmento en ensamblador x86 realiza la suma de dos números y almacena el resultado en un registro:

```
MOV AX, 5      ; Carga 5 en el registro AX
MOV BX, 3      ; Carga 3 en el registro BX
ADD AX, BX     ; Suma el contenido de BX a AX (AX = AX + BX)
```

Tras la ejecución de estas instrucciones, el registro AX contendrá el valor 8. Este tipo de estructuras secuenciales permite ilustrar, en contextos educativos, cómo se traduce una operación matemática sencilla al lenguaje más cercano al hardware.

Aplicabilidad profesional y en la Formación Profesional

El dominio de las instrucciones básicas del lenguaje ensamblador permite al alumnado interpretar y optimizar programas a un nivel que no es alcanzable con lenguajes de alto nivel. Es especialmente valioso en sectores donde el control del tiempo de ejecución, el tamaño del código y el consumo energético son determinantes, como en la electrónica de consumo, las telecomunicaciones o los sistemas de defensa.

En el aula, estas técnicas pueden desarrollarse mediante simuladores, prácticas con microcontroladores reales o entornos virtuales que permiten depurar paso a paso la ejecución

de las instrucciones. Esto fortalece la visión sistémica del alumno sobre cómo se ejecuta realmente un programa en el procesador y qué implicaciones tiene cada instrucción a nivel físico.

Este conocimiento es esencial para profundizar en aspectos más específicos del lenguaje ensamblador, como los formatos de instrucción y los modos de direccionamiento, que se analizarán en el siguiente apartado.

Formatos

Estructura general de una instrucción en ensamblador

Una instrucción en lenguaje ensamblador se compone, en términos generales, de cuatro elementos fundamentales: **etiqueta**, **mnemónico de operación**, **operandos** y **comentario**. Su estructura básica puede representarse de la siguiente manera:

```
[etiqueta:] instrucción [operando1, operando2] ; comentario
```

- **Etiqueta:** es un identificador simbólico que permite referirse a una dirección de memoria o una instrucción concreta, útil para la gestión de saltos y bucles.
- **Instrucción (mnemónico):** representa una operación específica del conjunto de instrucciones del procesador.
- **Operandos:** definen los datos sobre los que se va a realizar la operación. Pueden ser registros, direcciones de memoria, valores inmediatos o combinaciones.
- **Comentario:** inicia con ; y explica la función de la línea de código, sin ser interpretado por el ensamblador.

Ejemplo:

```
LOOP1: ADD AX, BX ; Suma el contenido de BX a AX
```

Este formato flexible permite al programador describir operaciones complejas de manera explícita, manteniendo control total sobre los recursos del sistema.

Tipos de formatos según el número de operandos

Los formatos de instrucción pueden clasificarse según el número de operandos que utilizan, lo cual varía en función de la arquitectura del procesador. Las principales modalidades son:

Formato de cero operandos

Este tipo de instrucciones no requiere operandos explícitos. Generalmente operan sobre registros implícitos, como en la pila o en registros acumuladores.

Ejemplos:

- **RET**: retorna de una subrutina.
- **NOP**: no realiza ninguna operación (útil para sincronización o alineación de código).

Formato de un operando

La operación afecta a un único operando, que puede ser un registro o una dirección de memoria. El segundo operando está implícito (por ejemplo, un acumulador o una constante).

Ejemplos:

- **INC AX**: incrementa el contenido del registro AX.
- **NEG AL**: invierte el signo del registro AL.

Formato de dos operandos

Es el formato más común en muchas arquitecturas como x86. El primer operando suele representar el destino, y el segundo, el origen. La operación modifica el contenido del primer operando.

Ejemplo:

- **MOV DX, AX**: copia el contenido de AX en DX.

Formato de tres operandos

Se encuentra en arquitecturas como MIPS, ARM o RISC-V, donde las instrucciones están diseñadas bajo el principio RISC. Este formato mejora la claridad y permite expresiones más compactas.

Ejemplo (MIPS):

```
ADD $t0, $t1, $t2 ; $t0 = $t1 + $t2
```

Este modelo obliga a que todas las operaciones se realicen entre registros, promoviendo una alta eficiencia en la ejecución y simplificando la lógica del procesador.

Codificación binaria y longitud de instrucción

Cada instrucción ensamblador se traduce finalmente a una representación binaria única, que el procesador interpreta directamente. Esta representación depende del diseño del conjunto de instrucciones y puede tener longitud fija o variable.

Longitud fija

Las arquitecturas RISC (como ARM o MIPS) utilizan formatos de instrucción de longitud fija (normalmente 32 bits). Esto permite simplificar el pipeline del procesador y prever la decodificación sin ambigüedades.

Ventajas:

- Decodificación más rápida.
- Uniformidad en el diseño de hardware.

Longitud variable

Las arquitecturas CISC (como x86) presentan instrucciones de longitud variable, desde 1 hasta 15 bytes. Esto permite una codificación más compacta y potente, con múltiples modos de direccionamiento y combinaciones de operandos.

Ventajas:

- Mayor expresividad.
- Código más compacto en aplicaciones complejas.

Inconvenientes:

- Mayor complejidad en la decodificación.
- Mayor dificultad en la predicción de ejecución.

Formatos según la función de la instrucción

Dependiendo de la naturaleza de la operación, también se pueden clasificar los formatos en función de su propósito:

- **Formatos aritméticos:** incluyen mnemónicos como **ADD**, **SUB**, **MUL**, con operandos que pueden ser registros o memoria.
- **Formatos lógicos:** **AND**, **OR**, **XOR**, útiles para máscaras de bits y operaciones booleanas.
- **Formatos de control:** **JMP**, **CALL**, **RET**, que utilizan etiquetas o direcciones como operandos.
- **Formatos de manipulación de cadenas:** como **MOVS**, **STOS**, que trabajan sobre segmentos de memoria consecutivos.
- **Formatos de entrada/salida:** **IN**, **OUT**, usados para comunicación con dispositivos periféricos a través de puertos.

Representación simbólica y sistemas numéricos

El lenguaje ensamblador permite especificar datos en diferentes bases numéricas, lo que facilita la comprensión y optimización del código a nivel binario:

- **Hexadecimal:** común en direcciones y valores de bajo nivel (**0x3F**, **3Fh**).
- **Binario:** útil en operaciones de bit a bit (**10110101b**).
- **Decimal:** para valores humanos o constantes claras (**42**).
- **Octal:** menos usado actualmente, pero presente en ciertos contextos históricos (**052o**).

Estas representaciones simbólicas permiten una codificación más precisa y visualmente comprensible del comportamiento de cada instrucción.

Aplicación práctica y educativa de los formatos

El conocimiento de los formatos de instrucción es crucial para interpretar correctamente el comportamiento del procesador, identificar errores lógicos y optimizar recursos en sistemas de

baja capacidad. En entornos profesionales, esta habilidad permite analizar desensamblados, diseñar rutinas eficientes o depurar firmware en condiciones reales.

En el aula, la comprensión de estos formatos permite al alumnado trabajar con simuladores, descompiladores y entornos de desarrollo para microcontroladores, desarrollando una visión detallada del ciclo de ejecución del software. Además, facilita la integración con módulos relacionados con la arquitectura de computadores, el análisis del rendimiento y la ciberseguridad.

El dominio de los formatos de instrucción constituye una base sólida para abordar uno de los aspectos más determinantes del lenguaje ensamblador: los modos de direccionamiento, que serán analizados a continuación desde una perspectiva técnica y funcional.

Direccionamientos

Concepto y relevancia del modo de direccionamiento

El modo de direccionamiento en lenguaje ensamblador define cómo se accede a los datos que una instrucción necesita para su ejecución. Es un componente esencial del formato de instrucción, ya que permite indicar al procesador si debe operar con un valor inmediato, con el contenido de un registro, con una posición de memoria o con una dirección calculada dinámicamente. La versatilidad de los modos de direccionamiento otorga al lenguaje ensamblador un enorme potencial expresivo, sin renunciar a la eficiencia y control característicos del bajo nivel.

Comprender los modos de direccionamiento es clave para optimizar el uso de los recursos del sistema, ya que afecta directamente al número de ciclos de reloj necesarios para ejecutar una instrucción, al uso del bus de datos y a la eficiencia del uso de registros y memoria. Además, determina la capacidad del lenguaje para trabajar con estructuras complejas como arrays, pilas, punteros o registros de estructuras.

Clasificación de los modos de direccionamiento

Aunque cada arquitectura de procesador define su propio conjunto de modos de direccionamiento, existen categorías comunes aplicables a la mayoría de los entornos ensamblador:

Direccionamiento inmediato

El operando es un valor constante codificado directamente en la instrucción. Es el modo más rápido, ya que no requiere accesos a memoria.

Ejemplo:

```
MOV AX, 5 ; Carga el valor 5 directamente en el registro AX
```

Este modo es útil cuando se necesita cargar valores fijos, configurar contadores o establecer máscaras binarias.

Direccionamiento directo o absoluto

La dirección de memoria donde reside el operando se especifica de forma explícita en la instrucción.

Ejemplo:

```
MOV AL, [1234h] ; Carga en AL el valor almacenado en la dirección 1234h
```

Aunque simple, este modo es menos flexible, ya que la dirección está codificada de forma fija en el código, dificultando la reutilización dinámica.

Direccionamiento indirecto

La dirección del operando se encuentra en un registro, que actúa como puntero.

Ejemplo:

```
MOV AX, [BX] ; Carga en AX el valor de la memoria apuntada por BX
```

Este modo es fundamental en operaciones con arrays, punteros y acceso dinámico a estructuras de datos. Su uso intensivo requiere una planificación cuidadosa del uso de registros.

Direccionamiento por registro

El operando reside en un registro interno del procesador.

Ejemplo:

```
ADD AX, BX ; Suma el contenido de BX al registro AX
```

Este es el modo más eficiente, ya que no implica accesos a memoria y permite una ejecución muy rápida de operaciones aritméticas o lógicas.

Direccionamiento indexado

Se calcula una dirección sumando un valor base (registro) y un desplazamiento (constante o variable).

Ejemplo:

```
MOV AL, [SI+2] ; Accede a la posición de memoria dos bytes más allá del contenido de SI
```

Se utiliza en el recorrido de vectores o tablas, donde el índice varía en tiempo de ejecución. En arquitecturas como x86 se puede extender combinando múltiples registros y escalas, lo que permite acceder eficientemente a matrices o estructuras complejas.

Direccionamiento basado o con desplazamiento

Similar al indexado, pero el desplazamiento se suma a un registro base que generalmente apunta a una estructura o bloque de memoria.

Ejemplo:

```
MOV AX, [BP+4] ; Recupera un valor de la pila con base en BP y desplazamiento de 4 bytes
```

Este modo es imprescindible en la implementación de llamadas a subrutinas, gestión de parámetros y variables locales, especialmente en arquitecturas que utilizan marcos de pila.

Direccionamiento relativo

Se utiliza principalmente en instrucciones de salto, donde el destino se calcula sumando un desplazamiento relativo a la dirección actual del contador de programa.

Ejemplo:

```
JE LABEL ; Salta a LABEL si se cumple la condición, relativo a la posición actual
```

Este modo permite construir código más compacto y relocizable, especialmente útil en sistemas operativos y firmware.

Interacción entre modos de direccionamiento y eficiencia del código

La elección del modo de direccionamiento tiene un impacto directo en la eficiencia del programa ensamblador. El direccionamiento por registro es el más rápido, pero también el más limitado por el número de registros disponibles. En cambio, los modos basados o indexados, aunque más flexibles, requieren más ciclos de reloj y generan instrucciones más largas.

Una práctica común en programación optimizada consiste en utilizar registros para operaciones intensivas y recurrir al direccionamiento indirecto o indexado cuando se trabaja con grandes volúmenes de datos o estructuras dinámicas. La elección correcta permite reducir el uso de memoria, minimizar accesos a RAM y evitar cuellos de botella en el bus de datos.

Además, el diseño de subrutinas eficientes, el paso de parámetros a través de la pila y el uso de estructuras complejas dependen estrechamente del manejo adecuado de los modos de direccionamiento. Por ello, este conocimiento es esencial no solo para escribir código funcional, sino también para depurar, desensamblar o analizar el comportamiento de sistemas ya compilados.

Aplicaciones formativas y profesionales

El dominio de los modos de direccionamiento permite al alumnado identificar cómo se organiza la memoria, cómo se estructuran las llamadas a funciones y cómo se construyen algoritmos a nivel máquina. En la práctica profesional, estas habilidades son claves en sectores como el diseño de microcontroladores, el desarrollo de sistemas embebidos, la ingeniería inversa o la ciberseguridad.

En el aula, este conocimiento puede abordarse mediante simuladores de CPU, proyectos de codificación directa en ensamblador o análisis de trazas de ejecución. Estas actividades facilitan la interiorización de conceptos abstractos y permiten la experimentación directa con recursos de bajo nivel.

Una vez comprendidos los distintos modos de direccionamiento, se puede completar la reflexión global sobre el lenguaje ensamblador, considerando sus implicaciones en el rendimiento del software, la eficiencia del hardware y la empleabilidad del alumnado.

Conclusión

La programación en lenguaje ensamblador representa una disciplina esencial en el ámbito de los sistemas y aplicaciones informáticas, no solo por su proximidad al hardware, sino por el nivel de precisión, control y eficiencia que ofrece frente a los lenguajes de alto nivel. A través del estudio de sus instrucciones básicas, formatos y modos de direccionamiento, se obtiene una visión integral del funcionamiento interno del procesador, de la gestión de la memoria y del ciclo de ejecución de las instrucciones, lo que permite desarrollar soluciones optimizadas para entornos donde cada recurso cuenta.

Las instrucciones básicas en ensamblador permiten operar directamente sobre registros, gestionar flujos de control, realizar cálculos aritméticos y manipular bits de manera eficiente. La comprensión de los distintos formatos de instrucción, ya sean de uno, dos o tres operandos, así como la diferencia entre arquitecturas de longitud fija o variable, es crucial para escribir código legible, eficiente y fácilmente depurable. Por su parte, los modos de direccionamiento amplían enormemente la capacidad expresiva del lenguaje, al permitir acceder dinámicamente a datos en memoria, gestionar estructuras complejas y controlar el comportamiento de la pila en la ejecución de subrutinas.

Desde el punto de vista profesional, el dominio del ensamblador es especialmente valioso en contextos donde se requiere un alto grado de especialización técnica: desarrollo de firmware, sistemas embebidos, programación de controladores, análisis forense, ingeniería inversa o ciberseguridad. En todos estos ámbitos, el conocimiento profundo de cómo interactúan las instrucciones con el hardware puede marcar la diferencia entre una solución funcional y una solución excelente.

En la Formación Profesional, trabajar con lenguaje ensamblador no solo aporta un conocimiento técnico avanzado, sino que fortalece competencias clave como el razonamiento lógico, la planificación, la depuración sistemática y la comprensión estructural del software. Estas competencias, alineadas con los principios de la LOMLOE y la Ley Orgánica 3/2022 de Ordenación e Integración de la Formación Profesional, permiten formar perfiles técnicos polivalentes, con alta empleabilidad en sectores estratégicos de la industria tecnológica.

El tratamiento de estos contenidos puede enriquecerse en el aula mediante metodologías activas como el aprendizaje basado en proyectos, la resolución de retos técnicos o el uso de simuladores y microcontroladores reales. Estas prácticas favorecen la transferencia de conocimientos al entorno laboral y promueven el pensamiento crítico, la autonomía y la innovación técnica.

En definitiva, integrar el estudio del lenguaje ensamblador en el currículo de sistemas y aplicaciones informáticas contribuye a una formación sólida, versátil y orientada a los desafíos reales del sector. Potenciar estas competencias técnicas es clave para impulsar la empleabilidad del alumnado, su capacidad de adaptación y su preparación para un futuro profesional exigente y en continua evolución.

Bibliografía

Ley Orgánica 3/2020, de 29 de diciembre, por la que se modifica la Ley Orgánica 2/2006, de 3 de mayo, de Educación. Boletín Oficial del Estado, núm. 340, 30 de diciembre de 2020.

Ley Orgánica 3/2022, de 31 de marzo, de ordenación e integración de la Formación Profesional

Stallings, W. (2016). *Computer organization and architecture: Designing for performance* (10th ed.). Pearson.

- Tanenbaum, A. S., & Bos, H. (2023). *Modern operating systems* (5th ed.). Pearson.
- Kurose, J. F., & Ross, K. W. (2020). *Computer networking: A top-down approach* (8th ed.). Pearson.
- Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database system concepts* (7th ed.). McGraw-Hill Education.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms* (4th ed.). MIT Press.
- Kernighan, B. W., & Ritchie, D. M. (1988). *The C programming language* (2nd ed.). Prentice Hall.
- Felke-Morris, T. A. (2021). *Web development and design foundations with HTML5* (10th ed.). Pearson.
- Pressman, R. S., & Maxim, B. R. (2019). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.
- Project Management Institute. (2017). *A guide to the project management body of knowledge (PMBOK Guide)* (6th ed.). Project Management Institute.