El Software

En esta unidad, **El Software**, nos interesa conocer el proceso de desarrollo de software, desde el punto de vista de la organización de computadoras. Explicaremos cómo se llega desde un programa, en un lenguaje de alto o bajo nivel, a obtener una sucesión de instrucciones de máquina para un procesador.

Traductores

Uno puede pensar en un programa cualquiera como si se tratara de una máquina, cuyo funcionamiento es, en principio, desconocido. Todo lo que vemos es que, si introducimos ciertos datos, de alguna forma esta "máquina" devolverá un resultado.

Traductores

Si pensamos en una clase especial de estos programas, donde los datos que ingresan son a su vez un programa, y donde la salida devuelta por la máquina es, a su vez, un programa, entonces esa clase especial de programas son los **traductores**.

Como hemos dicho anteriormente, una CPU como el MCBE sólo sabe ejecutar instrucciones de código máquina expresadas con unos y ceros. Cuando vimos el lenguaje ensamblador del MCBE lo propusimos simplemente como una forma de abreviar las instrucciones de máquina, o como una forma de facilitar la escritura, porque los mnemónicos eran más fáciles de memorizar que las instrucciones con unos y ceros. Sin embargo, un programa escrito en ensamblador del MCBE podría ser traducido automáticamente, por un traductor, a código de máquina MCBE, ahorrándonos mucho trabajo y errores.

Esta clase de traductores, que reciben un programa en lenguaje ensamblador y devuelven un programa en código de máquina, son los llamados ensambladores.

Ensamblador x86

Cada CPU tiene su propio lenguaje ensamblador, y existen programas traductores (ensambladores) para cada una de ellas. Por ejemplo, la familia de procesadores de Intel para computadoras personales comparte el mismo ISA, o arquitectura y conjunto de instrucciones. Cualquiera de estos procesadores puede ser programado usando un ensamblador para la familia **x86**.

Según la tradición, el primer programa que uno debe intentar escribir cuando comienza a aprender un lenguaje de programación nuevo es "Hola mundo". Es un programa que simplemente escribe esas palabras por pantalla. Aquí mostramos el clásico ejemplo de "Hola mundo" en el lenguaje ensamblador de la familia x86.

Procesador x86

Los procesadores de la familia x86 se encuentran en casi todas las computadoras personales y notebooks.

Ensamblador ARM

Por supuesto, los procesadores de familias diferentes tienen conjuntos de instrucciones diferentes. Así, un lenguaje y un programa ensamblador están ligado a un procesador determinado. El código máquina producido por un ensamblador no puede ser trasladado sin cambios a otro procesador que no sea aquel para el cual fue ensamblado. Las instrucciones de máquina tendrán sentidos completamente diferentes para uno y otro.

Por eso, el código máquina producido por un ensamblador para x86 no puede ser trasladado directamente a una computadora basada en un procesador como, por ejemplo, ARM; sino que el programa original, en ensamblador, debería ser **portado** o traducido al ensamblador propio de ARM, por un programador, y luego ensamblado con un ensamblador para ARM.

Procesador ARM

El ARM es un procesador que suele encontrarse en plataformas móviles como *tablets* o teléfonos celulares, porque ha sido diseñado para minimizar el consumo de energía, una característica que lo hace ideal para construir esos productos portátiles. Su arquitectura, y por lo tanto, su conjunto de instrucciones, están basados en esos principios de diseño.

Ensamblador PowerPC

Lo mismo ocurre con otras familias de procesadores como el Power PC, un procesador que fue utilizado para algunas generaciones de consolas de juegos, como la PlayStation 3.

Procesador PowerPC

Lenguajes de programación

Como vemos, tanto el lenguaje de máquina como el ensamblador o **Assembler** son lenguajes **orientados a la máquina**. Ofrecen control total sobre lo que puede hacerse con un procesador o con el sistema construido alrededor de ese procesador; pero resultan poco **portables**, y escribir un programa para un problema complejo suele ser muy costoso en tiempo y esfuerzo.

Otros lenguajes son más **orientados al problema**, lo que quiere decir que nos aíslan de cómo se escriben las instrucciones de máquina, y nos permiten especificar las operaciones que necesitamos para resolver nuestro problema en forma más parecida al lenguaje natural, matemático, o humano.

Lenguajes y niveles

Se han diseñado muchísimos lenguajes de programación, que podemos organizar en una jerarquía de lenguajes, de acuerdo a su orientación.

Lenguajes y niveles

Entre estos lenguajes encontramos los de *bajo nivel* u orientados a la máquina, los de *alto nivel*, u orientados al problema, y algunos en una zona intermedia. Cada uno de ellos es más apto para alguna clase de tareas de programación y cada uno tiene sus aplicaciones.

Terminología

Cuando utilizamos un traductor para obtener un programa **ejecutable**, el programa que nosotros escribimos, en algún lenguaje, se llama **programa fuente**, y estará generalmente contenido en algún **archivo fuente**.

El resultado de la traducción será un archivo llamado **objeto** conteniendo las instrucciones de código máquina equivalentes. Sin embargo, este archivo objeto puede no estar completo, ya que el programador puede hacer uso de rutinas o funciones que vienen provistas con el sistema, y no necesita especificar cómo se realizan esas funciones. Al no aparecer en el programa fuente, esas funciones no aparecerán en el archivo objeto.

Por ejemplo, los programas para diferentes procesadores que vimos hace instantes, en lenguaje ensamblador, imprimen todos en pantalla un mensaje; pero la acción de imprimir algo en pantalla no es trivial ni sencillo, y la explicación de cómo se hace no está contenida en esos programas. En su lugar, existe una llamada a una función de impresión cuya definición reside en algún otro lugar.

Ese otro lugar donde están definidas funciones disponibles para el programador son las **bibliotecas**. Las bibliotecas son archivos conteniendo grupos o familias de funciones.

El proceso de **vinculación**, que es posterior a la traducción, debe buscar en esas bibliotecas la definición de las funciones faltantes en el archivo objeto.

Traductores

Los traductores pueden funcionar de dos maneras: o bien producen una versión en código máquina del programa fuente (**compiladores**) o bien analizan instrucción por instrucción del programa fuente y además de generar una traducción a código máquina de cada línea, la ejecutan (**intérpretes**).

Luego de la compilación, el programa en código máquina obtenido puede ser ejecutado muchas veces. En cambio, el programa interpretado debe ser traducido cada vez que se ejecute.

Traductores

Una ventaja comparativa de la compilación respecto de la interpretación es la mayor velocidad de ejecución. Al separar las fases de traducción y ejecución, un compilador alcanza la máxima velocidad de ejecución posible en un procesador dado. Por el contrario, un intérprete alterna las fases de traducción y ejecución, por lo cual la ejecución completa del mismo programa tardará algo más de tiempo.

Inversamente, el código interpretado presenta la ventaja de ser directamente portable. Dos plataformas diferentes podrán ejecutar el mismo programa interpretable, siempre que cuenten con intérpretes para el mismo lenguaje. Por el contrario, un programa compilado está en código máquina para alguna arquitectura específica, así que no será compatible con otras.

Ciclo de compilación

El desarrollador que necesita producir un archivo ejecutable utilizará varios programas de sistema como editores, traductores, vinculadores, etc.

Ciclo de compilación

En algún momento anterior, alguien habrá creado una biblioteca de funciones para uso futuro. Esa biblioteca consiste en versiones objeto de varias funciones, compiladas, y reunidas con un programa bibliotecario, en un archivo.

Ciclo de compilación

Esa biblioteca es consultada por el vinculador para completar las referencias pendientes del archivo objeto.

Ciclo de compilación

Ciclo de compilación

En resumen, la primera fase del ciclo de compilación es necesariamente la edición del programa fuente.

Ciclo de compilación

Luego, la traducción para generar un objeto con referencias pendientes.

Ciclo de compilación

Luego, la vinculación con bibliotecas para resolver esas referencias pendientes.

Ciclo de compilación

El resultado final del ciclo de compilación es un ejecutable.

Muchos desarrolladores utilizan algún **ambiente integrado de desarrollo (IDE)**, que es un programa que actúa como intermediario entre el usuario y los componentes del ciclo de compilación (compilador, vinculador, bibliotecas). Sin embargo, aunque el ambiente integrado lo oculte, el sistema de desarrollo sigue trabajando como se ha descrito, con pasos separados para edición, traducción, vinculación y ejecución.

Paradigmas de lenguajes de programación

La programación en lenguajes de alto nivel puede adoptar varias formas. Existen diferentes modos de diseñar un lenguaje, y varios modos de trabajar para obtener los resultados que necesita el programador. Esos modos de pensar o trabajar se llaman **paradigmas de lenguajes de programación**.

Hay al menos cuatro paradigmas reconocidos, que son, aproximadamente en orden histórico de aparición, **imperativo**, **lógico o declarativo**, **funcional** y **orientado a objetos**. Los paradigmas lógico y funcional son los más asociados a la disciplina de la Inteligencia Artificial.

1. Paradigma imperativo

Bajo el paradigma imperativo, los programas consisten en una sucesión de instrucciones o comandos, como si el programador diera órdenes a alguien que

las cumpliera. El ejemplo en lenguaje **C** explica cuáles son las órdenes que deben ejecutarse, una por una.

2. Paradigma lógico

La definición de **factorial** en lenguaje Prolog que mostramos se compone de un hecho y dos reglas. El hecho consiste en que el **factorial** de 0 vale 1. La primera regla expresa que el factorial de un número **N** se calcula como el factorial de **N-1** multiplicado por N. Es una definición **recursiva** porque la definición de la regla se utiliza a sí misma.

El usuario de este programa puede usarlo de dos maneras. Podría preguntar el valor del factorial de un número N, o consultar si es cierto que el factorial de N es otro número dado Y.

3. Paradigma funcional

En el lenguaje Lisp, perteneciente al paradigma funcional, una función es un enunciado entre paréntesis que puede contener a otras funciones. En particular la definición de **factorial** presentada aquí contiene a su vez una invocación de la misma función, volviéndola una función **recursiva**.

El lenguaje Lisp utiliza notación prefija para los operadores.

4. Paradigma orientado a objetos

En un lenguaje **orientado a objetos**, definimos una **clase** que funciona como un molde para crear múltiples instancias de objetos que se parecen entre sí, ya que tienen los mismos datos que los componen y la misma funcionalidad.

En el ejemplo de programación orientada a objetos en Python, definimos una clase **Combinatoria** que producirá objetos con la conducta **factorial**. El programa crea un objeto, instancia de la clase Combinatoria, llamado **c**, al cual se le envía el mensaje **factorial**, que dispara la conducta correspondiente especificada en el método del mismo nombre. Finalmente se imprime su valor.