Software

Lenguaje de máquina

En la unidad anterior hemos presentado al MCBE, un ejemplo de computador ideal muy elemental. El MCBE cuenta con una CPU, una Unidad de E/S, una memoria de unas pocas posiciones, y un reducido conjunto de instrucciones.

La CPU del MCBE es capaz de ejecutar un programa, es decir, una lista de instrucciones, que se disponen en los bytes de la memoria, conteniendo códigos de operaciones y datos en un cierto formato de bits. Esos códigos conforman el **lenguaje de máquina**[1], o código de máquina, de la MCBE.

Por supuesto, las máquinas reales modernas tienen un conjunto de instrucciones más rico, así como un hardware más completo y una conducta más compleja; pero en lo fundamental se parecen mucho al MCBE, y también tienen un lenguaje de máquina que les es propio. Como las instrucciones son muchas, de mayor longitud, y se presentan en muchos modos de funcionamiento, programar en código máquina estas computadoras de la realidad, tan complejas, rápidamente se vuelve una tarea inabordable para los humanos. Para programar estas máquinas resulta mucho más cómodo utilizar un lenguaje simbólico, el **lenguaje de ensamblado**[2], **ensamblador**, o **Assembly** (a veces también llamado lenguaje Assembler, pero que no debe confundirse con el programa Assembler que enseguida describiremos).

En el lenguaje ensamblador, cada instrucción del lenguaje máquina está representada por una abreviatura o **mnemónico** que recuerda su función, y así resulta (un poco) más fácil programar la computadora. Por ejemplo, es *un poco* más fácil escribir, comprender y corregir el programa en Assembly del MCBE que se muestra en la figura siguiente, que su equivalente en lenguaje de máquina en la misma figura.

```
LD 30   →  01011110
SUB 5   →  10100101
ST 6    →  01100110
JZ -3   →  11111101
HLT     →  00100000
1       →  00000001
0       →  00000000
```

Sin embargo, como dijimos, éste no es más que un lenguaje simbólico. Sigue siendo un hecho que lo único que comprende la computadora son números binarios, y que lo único que podemos introducir en la memoria y en los registros de la CPU son bits y bytes. La computadora, de cualquier arquitectura que se trate, no puede ejecutar más que las instrucciones del código de máquina para el cual fue diseñada. Por lo cual, en algún momento, alguien o algo debe efectuar la

1 http://es.wikipedia.org/wiki/Lenguaje_de_máquina
2 http://es.wikipedia.org/wiki/Lenguaje_ensamblador

traducción de los mnemónicos, escritos por el programador, a los bits y bytes del programa en código de máquina.

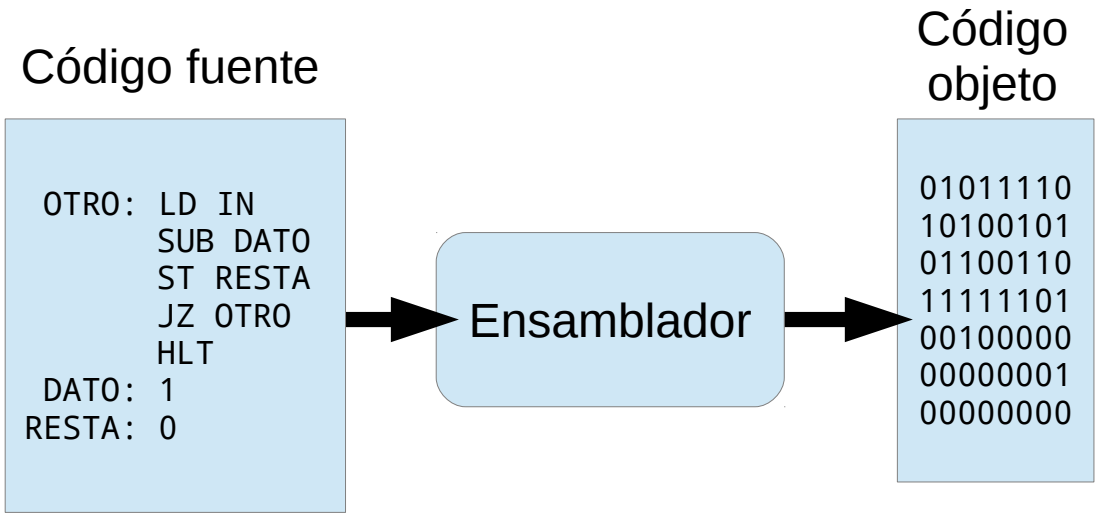
¿Quién se encargará de esta tarea de traducción? Inicialmente, la traducción era hecha a mano por el mismo programador; sin embargo, no tardó en surgir la idea de crear un programa que hiciera esta traducción. Al fin y al cabo, se trata de manipular información, cosa que las computadoras hacen bien y rápidamente. Así se creó el primer **programa ensamblador, o Assembler**, para traducir de código ensamblador a código máquina. El programa ensamblador (el Assembler) recibe el código en lenguaje ensamblador (o Assembly), y genera, línea por línea, instrucciones de código máquina que pueden cargarse en la computadora y ser ejecutadas como un programa.

El programa en Assembly es esencialmente un texto, compuesto por una secuencia de caracteres legibles, aquellos que conforman la secuencia de mnemónicos que haya utilizado el programador. Por eso, el programa en Assembly es completamente diferente de las instrucciones de código máquina a las cuales equivale.

```
L  4C  01001100
D  44  01000100  → 01000100
   20  00100000
4  34  00110100
```

La instrucción **LD 4** en ensamblador para el MCBE equivale a la instrucción en código máquina **01000100**. La instrucción en ensamblador está conformada por los caracteres L, D, espacio, y 4. Por lo tanto, mide 4 bytes. Si viéramos este texto escrito en ASCII, se trataría de la secuencia de códigos hexadecimales **4C, 44, 20, 34**. Si los vemos en binario, se trata de los bytes **01001100, 01000100, 00100000, y 00110100**. Obviamente, estos cuatro bytes dispuestos en memoria no son lo mismo que la instrucción en código máquina que necesitamos hacer aparecer en la memoria del MCBE, que no es otra que **01000100**.

La tarea del ensamblador es precisamente convertir esos caracteres de las instrucciones en **código fuente** (el que escribe el programador) en los correspondientes bytes de **código objeto** (el que es ejecutable por la computadora).



Contar con un programa ensamblador nos permite disfrutar de ciertas comodidades a la hora de programar en Assembly. Por ejemplo, podemos agregar al programa ensamblador la capacidad de manejar **constantes simbólicas**, **rótulos** o etiquetas en el programa fuente, así como **macros** (secuencias de instrucciones que se repiten frecuentemente). El programa para el MCBE de más arriba se puede hacer aún más fácil de entender y de modificar, si reemplazamos las direcciones y

desplazamientos usados por nombres simbólicos:

```
OTRO: LD IN       →   01011110
      SUB DATO    →   10100101
      ST RESTA    →   01100110
      JZ OTRO     →   11111101
      HLT         →   00100000
DATO: 1           →   00000001
RESTA: 0          →   00000000
```

El ensamblador (que está hecho a medida para el MCBE) conoce por construcción a qué dirección corresponde el rótulo **IN**, y puede calcular por sí solo las direcciones representadas por los rótulos **DATO** y **RESTA**, posiciones de memoria ubicadas a continuación de la última instrucción ejecutable; o computar el desplazamiento para el PC necesario a fin de desviar el control a la instrucción rotulada **OTRO.** Así, el programador no necesita efectuar estos cálculos incómodos y propensos a errores. Escribe el texto en Assembly creando un archivo de texto; luego, llama al programa Assembler que transforma ese archivo de texto y crea otro archivo conteniendo el código objeto. Para hacer su trabajo, el ensamblador procesa todo el texto del programa fuente, identificando elementos léxicos, detectando posibles errores de programación, y creando tablas de símbolos y estructuras de datos auxiliares. Todo este trabajo del ensamblador es invisible; lo que ve el programador es simplemente el código objeto generado, listo para ejecutar en la máquina.

Para investigar
¿Qué aspecto tienen los programas en lenguaje ensamblador para otras arquitecturas? ¿Cómo es un programa en ensamblador para la PC (arquitectura Intel x86)? ¿Para otras computadoras, con arquitectura MIPS? ¿Para el microprocesador Z80? ¿Qué procesadores llevan las consolas de juegos XBox, PlayStation, Wii, y qué aspecto tiene un programa para estos microprocesadores?

Lenguajes de alto nivel

Los lenguajes de máquina son **lenguajes de bajo nivel**[3], en el sentido de que sus instrucciones y su forma de programación están condicionados por la arquitectura de la máquina. Pese a que las instrucciones de estos lenguajes son sencillas, la resolución de problemas usando programación en lenguajes de bajo nivel no resulta fácil, ya que el programador necesita especificar con mucho detalle las acciones a realizar.

En otras palabras, programar con lenguajes extremadamente simples como los lenguajes de bajo nivel no es difícil, pero resolver problemas con ellos sí, porque requieren bastante esfuerzo de programación. Inversamente, un lenguaje más complejo puede ser más difícil de aprender, pero más expresivo, y reducir el esfuerzo de programación.

El paso siguiente en la evolución de los lenguajes de computación fue liberarse de la correspondencia “1 a 1” en términos de una instrucción de Assembly por una instrucción de código máquina. Teniendo la posibilidad de escribir programas traductores, era posible definir nuevos

3 http://es.wikipedia.org/wiki/Lenguaje_de_bajo_nivel

lenguajes de programación que se orientaran mejor a la forma como pensamos la resolución de los problemas, y no nos mantuvieran atados a la forma como las computadoras procesan los datos.

Así se crearon los **lenguajes de alto nivel**[4], de los cuales hay varias familias o **paradigmas**[5]. Hay lenguajes multipropósito y los hay especializados en la resolución de determinados problemas. Algunos tienen el objetivo de ser más veloces en su ejecución, y otros, el de ser más fáciles de programar o de mantener. Algunos lenguajes se utilizan sobre todo en Inteligencia Artificial, otros en programación científica, otros en aplicaciones de negocios, etc. En general, los lenguajes de alto nivel se parecen lo más posible a un idioma humano, pero sus características varían muchísimo, en especial su **sintaxis** (la forma como se escriben las instrucciones) puede tomar muchas formas. Puede encontrarse una colección de programas "Hola, mundo" muy interesante[6], mostrando la gran variedad de formas sintácticas de los lenguajes más conocidos, en Wikipedia[7].

Los lenguajes de bajo nivel tienen las ventajas de ser muy precisos, ofrecer control total del hardware, y tener traductores fáciles de implementar, pero las dificultades ya vistas para colaborar en la resolución de problemas. Además, están diseñados para un hardware determinado, y cada procesador tiene el suyo, es decir, resultan dependientes de la arquitectura de la máquina. Por estos motivos generalmente quedan relegados a la escritura de partes muy específicas del software, que estén fuertemente relacionadas con el hardware (como programas controladores de dispositivos o *drivers*). Los lenguajes de alto nivel se sitúan en el otro extremo: se programan siempre de la misma manera, sin importar cuál sea el procesador que ejecute los programas. Un lenguaje de alto nivel intenta ser independiente de la plataforma y ocultar las particularidades del hardware al programador, permitiéndole concentrarse en el problema.

Un lenguaje de alto nivel es diferente del Assembly y se aproxima mucho más a un lenguaje humano; pero, para ejecutar programas escritos en un lenguaje de alto nivel, sigue siendo necesaria la traducción al lenguaje de máquina. Los programas traductores para estos lenguajes se dividen en dos grandes categorías: **intérpretes**[8] y **compiladores**[9]. La diferencia entre ambos radica en el modo como realizan la traducción. Si el traductor realiza la traducción completa del programa fuente a código máquina, y sólo entonces se tiene disponible el código máquina para su ejecución, el traductor se denomina un compilador. Si, en cambio, el traductor realiza el ciclo traducir – ejecutar, línea por línea del programa (o en base a unidades de traducción más o menos pequeñas), se llama un intérprete. La situación es parecida a la diferencia que existe entre traductores e intérpretes de idiomas humanos: se llama intérprete (humano) a la persona que traduce oralmente, frase a frase, durante una conversación. En cambio, se llama traductor (humano) al que elabora la traducción completa de un documento y entrega esa traducción sólo una vez que está completa. Lenguajes muy conocidos como **Pascal, C, C++, Java, Fortran**, son lenguajes compilados. **Python, Perl, Ruby, PHP**, son lenguajes interpretados.

4 http://es.wikipedia.org/wiki/Lenguaje_de_alto_nivel
5 https://es.wikipedia.org/wiki/Paradigma_de_programación
6 Es tradición en la enseñanza de lenguajes de programación hacer los primeros pasos en el lenguaje demostrando la forma de crear un programa muy sencillo, que simplemente imprima por pantalla un mensaje. Por motivos históricos, este mensaje suele ser casi siempre el saludo "¡Hola, mundo!".
7 https://es.wikipedia.org/wiki/Anexo:Ejemplos_de_implementación_del_«Hola_mundo»
8 http://es.wikipedia.org/wiki/Intérprete_(informática)
9 http://es.wikipedia.org/wiki/Compilador

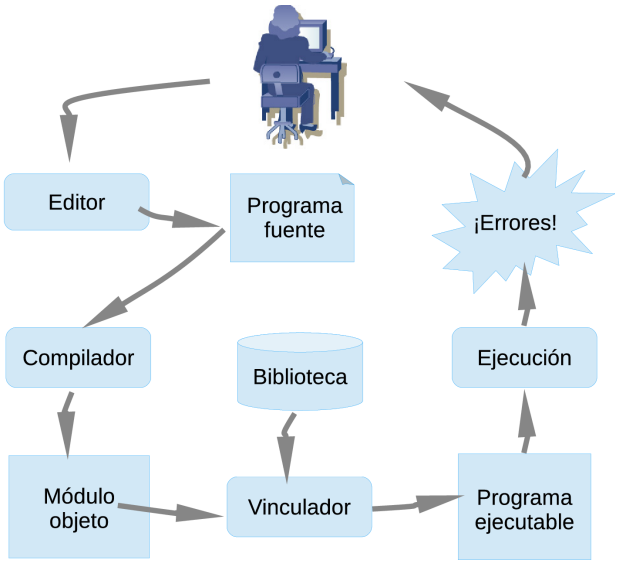
Ciclo de compilación

Al trabajar con un lenguaje de programación, el programador utiliza determinadas **herramientas** de software (programas que utiliza para desarrollar otros programas). Estas herramientas pueden ser programas separados o pueden estar contenidos dentro de un **ambiente de desarrollo integrado (IDE)**.

En primer lugar, el programador utiliza un **editor de textos** para crear un **archivo fuente** que contiene el **programa en código fuente**. Si el lenguaje que utiliza el programador es compilable, entonces **compila** el archivo fuente con un **compilador** para ese lenguaje, que genera un archivo conteniendo **código objeto**. Ese código objeto puede estar incompleto (porque el programador no ha especificado toda la conducta del programa) y requerir otros trozos de código almacenados en **bibliotecas.** Las bibliotecas son archivos que contienen trozos de código objeto usado frecuentemente. El código objeto del programa será **enlazado** con las bibliotecas utilizando un **vinculador** (o **linkeditor**, o **linker**) para generar, finalmente, un **archivo ejecutable**.

Si, por otro lado, el programador trabaja con un lenguaje interpretable, todo el proceso de la traducción de fuente a objeto y la vinculación se harán dinámicamente, al momento de ejecución.

Una vez que el programa está listo, el programador **prueba** el resultado ejecutándolo. La gran mayoría de las veces habrá que corregir **errores** (“**bugs**”) o defectos de funcionamiento del programa. Para esto puede ser necesario recurrir a un **debugger** o depurador de programas, que ayuda en la detección y corrección de errores, mostrando el funcionamiento del programa por dentro, o ejecutando paso a paso las instrucciones para encontrar el punto donde el programa falla. En este momento y con la información obtenida de la depuración, se vuelve a la fase de edición y el ciclo recomienza, repitiéndose hasta que la calidad del software elaborado sea aceptable.



Este ciclo de compilación determina entonces varios momentos:

Lenguaje compilable	Lenguaje interpretable
1. Edición del programa 2. Compilación del fuente 3. Vinculación del módulo objeto con bibliotecas 4. Ejecución de prueba 5. Depuración 6. Vuelta a 1, tantas veces como sea necesario 7. Puesta en producción	1. Edición del programa 2. Traducción del fuente, vinculación con bibliotecas y ejecución de prueba 3. Depuración 4. Vuelta a 1, tantas veces como sea necesario 5. Puesta en producción

A este ciclo de producción hay que agregar las etapas posteriores de **mantenimiento** y salida de servicio o **baja** del software.

Sistema de computación

Una vez que tenemos un compilador o intérprete, podemos crear **aplicaciones.** Estas aplicaciones se ejecutarán en una computadora y los **usuarios** del sistema las utilizarán. El sistema de computación donde correrán estas aplicaciones debe dar soporte a la ejecución de estas aplicaciones.

Usuarios
Programas de aplicación
SO
HW

Evolución
Máquina “pelada” (década del 50)
Sistemas batch simples
Sistemas batch multiprogramados
Sistemas de tiempo compartido
Sistemas desktop
Sistemas multiprocesadores
Sistemas de tiempo real
Sistemas Embebidos

Sistema operativo
Núcleo o kernel
Shell o intérprete de comandos

Funciones
Planificación de Procesos
Administración de Memoria principal
Sistema de Archivos
Administración de E/S
Administración de almacenamiento secundario
Protección
Intérprete de Comandos

Servicios
Ejecución de programas
Operaciones de E/S
Manejo de sistema de archivos
Detección de errores de hw y sw
Comunicaciones entre procesos y entre sistemas

Sistemas paralelos
De tiempo real
Empotrados - móviles
Sistemas distribuidos