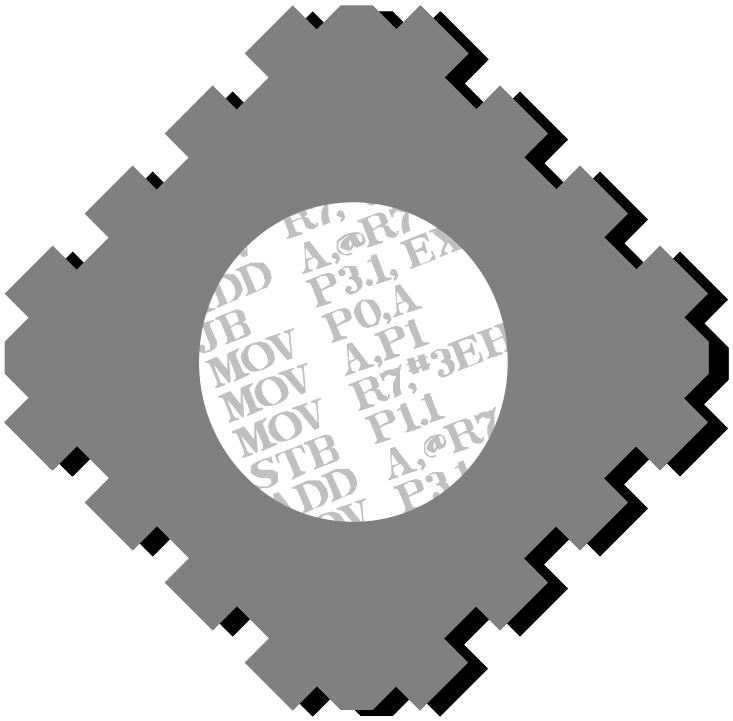
Sección 2

Implementación de Programas en Microprocesadores y Microcontroladores



INTRODUCCIÓN

En esta unidad se pesenta un panorama general del desarrollo de programas en alto y bajo nivel, así como ventajas y desventajas que conlleva trabajar en cada uno de estos niveles. Para iniciar esto es necesario tener la definición de algunos términos comúnmente utilizados en el desarrollo se programas para computadoras o microcontroladores.

Los programas que convierten un programa escrito en un lenguaje a otro lenguaje distinto se le llama *traductor*. Al lenguaje en que esta escrito el programa original se le llama *lenguaje fuente* y al que se convierte se la denomina *lenguaje objeto*. Ambos lenguajes, el fuente y el objeto definen niveles. Si se contara con un procesador que ejecutara directamente programas escritos en lenguaje fuente no seria necesario traducirlo a un lenguaje objeto.

La traducción se utiliza cuando se tiene un procesador para un lenguaje objeto y no se cuenta con uno para el lenguaje fuente. Es por ello que se realizan en lenguaje fuente y luego son traducidos para poder ejecutarlos. Los programas tanto el fuente como el objeto desde el punto de vista funcional son equivalentes, esto es producen los mismos resultados.

2.1 PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR

Los traductores se pueden dividir en dos grupos, según la relación que existe entre el lenguaje fuente y el objeto. Cuando el lenguaje fuente solamente muestra una representación simbólica de un lenguaje numérico, el traductor se denomina *ensamblador* y al lenguaje fuente se le conoce como *lenguaje ensamblador*. Sin embargo cuando el lenguaje fuente es un lenguaje mucho más complejo que el lenguaje objeto, es decir cuando una instrucción en el lenguaje fuente es traducida a varias instrucciones de lenguaje objeto de le denomina *compilador*.

Un lenguaje ensamblador puro es aquel que cada sentencia produce exactamente una instrucción máquina. Una razón para utilizar lenguaje ensamblador en lugar de programar en lenguaje numérico o *Lenguaje maquina*[5], es que es mucho más fácil programar en ensamblador. Es una gran diferencia utilizar nombres (mnemónicos) para instrucciones que utilizar únicamente números en alguna base (decimal, hexadecimal o binario). Esto es porque podemos recordar mas fácilmente abreviaturas para ciertas operaciones que números asignados a dichas operaciones, por ejemplo: es mas sencillo recordar que la instrucción ADD se utiliza para sumar, que recordar que el número 24 hace la misma operación. De esta manera el programador sólo necesita recordar los nombres simbólicos, ya que el ensamblador los traduce a las instrucciones numéricas. Esto mismo tambien es aplicado a las direcciones de memoria, es decir el programador puede dar nombres a las localidades de memoria.

[5] Lenguaje máquina se le denomina al lenguaje nato del procesaodor.

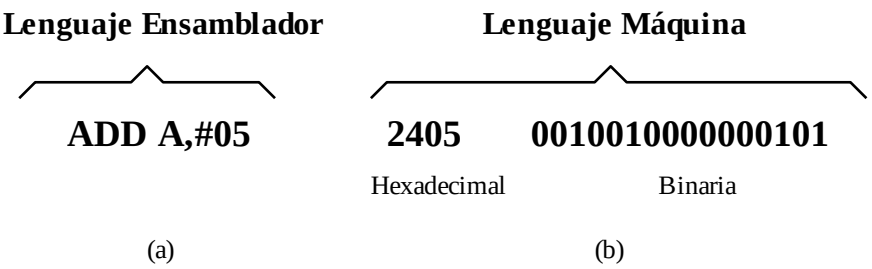


Figura 2.1 Instrucción para suma. a) leguaje ensamblador b) lenguaje máquina.

La figura 2.1 muestra los distintos formatos de la instrucción para sumar el acumulador y el numero 5. Como puede observarse, el lenguaje ensamblador tiene una gran ventaja sobre el lenguaje máquina, esto debido al manejo de mnemónicos (símbolos) que son equivalentes en cuanto a funcionalidad con los conjuntos de números ya sea en forma binaria o hexadecimal.

2.2 LENGUAJES DE ALTO NIVEL

El concepto de lenguaje de alto nivel se define como un lenguaje orientado a la solución de problemas, ademas están diseñados para permitir al programador concentrarse en la lógica del problema a resolver, liberándolo de la necesidad de un conocimiento profundo de lenguaje máquina de la computadora. El termino alto nivel significa que están orientados hacia la gente, en contraste con el lenguaje ensamblador de bajo nivel el cual esta orientado hacia la máquina.

Las características que identifican un lenguaje de alto nivel son:

- **Utiliza una notación especial orientada al problema a resolver.**

  Las expresiones de las instrucciones son congruentes con el tipo de problema. Con ensamblador para resolver una formula matemática puede tener la siguiente forma:

```
MOV A,X
MOV B,Y
MUL AB
SUB A,Z
MOV R,Z
```

  En un lenguaje de alto nivel , la instrucción para resolver el mismo problema puede expresarse como:

```
R=Z-X*Y
```

- **No se requiere conocimiento del código máquina.**

  El programador no necesita tener conocimientos del lenguaje máquina o de los códigos que la computadora maneja internamente. Esta facilidad ha permitido que una persona aprenda a programar y usar un lenguaje de programación de alto nivel prácticamente con cualquier computadora, ya que éste se independiza de las características de la máquina. Sin embargo siempre es conveniente conocer la arquitectura de la computadora y la forma en que internamente maneja la información, para así poder obtener el máximo provecho de su capacidad.

- **Facilidad para procesar el programa en otra computadoras (portabilidad).**

  El concepto de independencia del lenguaje con respecto a la computadora significa que un programa escrito para procesarse en una máquina puede ser ejecutado en otro sin tener problemas. Esto tiene una validez relativa, puesto que es necesario compilar el programa para esa nueva máquina, utilizando un compilador para dicha máquina. Generalmente es necesario hacer algunas modificaciones en el código fuente.

- **Expansión de las instrucciones.**

  La facilidad de programar con lenguaje de alto nivel se debe, en buena parte a que no es necesario escribir una docena de instrucciones cuando se puede ordenar lo mismo con una sola expresión. Sin embargo la máquina no puede procesar este tipo de instrucciones con tal grado de complejidad y por tal motivo tiene que expandir cada instrucción a una serie de instrucciones en lenguaje máquina.

Ventajas

- Facilidad de aprendizaje.
- Facilidad de utilizarlo.
- Facilidad para documentación y el mantenimiento.
- Facilidad de depuración.
- Facilidad para transportarlo a otra computadora o microcontrolador.
- Reducción del tiempo total para el desarrollo de programas.

Desventajas

- Tiempo necesario para la compilación.
- No siempre se tiene el programa objeto en forma optima.

Existen tres requerimientos suficientes para calificar a un lenguaje de alto nivel para su utilización en microcontroladores.

1.- La posibilidad de definir nuevas estructuras de alto nivel para E/S.
2.- El Acceso directo a memoria y al hardware para lectura y escritura.
3.- La posibilidad de llamar rutinas creadas en lenguaje ensamblador.

2.3 FUNCIONES Y MACROS

Antes de iniciar una comparación entre programación en lenguaje ensamblador y lenguajes de alto nivel (HLL[6]), existen dos puntos que se aplican a ambos tipos de lenguaje, uno de ellos son las *macro*[7] instrucciones y las funciones.

Los programadores de lenguaje ensamblador necesitan repetir frecuentemente grupos de instrucciones dentro de un programa. La forma mas sencilla de resolver esto es escribirlas cada vez que sea necesario, pero esto es un procedimiento tedioso. Los macros proporcionan una solución sencilla y eficiente al problema a esos bloques o grupos de instrucciones que se utilizan repetidamente en el programa.

La definición de un macro es un método que permite asignar un nombre a una porción de texto (varias instrucciones). Después de haber definido un macro, el programador puede escribir el nombre de dicho macro en vez de escribir toda la porción del texto. Por ejemplo, en un microcontrolador de la familia MCS-51 de Intel, la secuencia para sumar el puerto 1 con el puerto 2 y dejar el resultado en el puerto1 es de la siguiente manera:

```
MOV    A,P1     ; Mover el dato del puerto 1 al acumulador
ADD    A,P2     ; Sumar el contenido del puerto 2 al acumulador
MOV    P1,A     ; Sacar la suma por el puerto 1
```

Si es necesario realizar esta secuencia a lo largo del programa, una posibilidad de hacerlo seria repetir la secuencia cada vez que se necesite, otra es definir el siguiente macro para esta secuencia.

```
AddPorts    MACRO
            MOV A,P1      ; Mover el dato del puerto 1 al acumulador
            ADD A,P2      ; Sumar el contenido del puerto 2 al acumulador
            MOV    P1,A   ; Sacar la suma por el puerto 1
            ENDM
```

Una vez definido podemos utilizar la macro instrucción AddPorts para realizar la secuencia. Esto hace mas inteligible el programa. Sin embargo la utilización de macro instrucciones tiene una desventaja, esto es, cada vez que utilizamos o hacemos referencia a un macro, se repite cada una de las instrucciones que componen el macro, por tanto; el total de veces que se llamada al macro es el total de veces que se almacena la secuencia en memoria, esto hace crecer el requerimiento de memoria.
Una forma de re-utilizar el código, es almacenar en memoria la secuencia una unica vez y que se tenga la posibilidad de hacer referencia a dicha secuencia cuando se desee, dicho lo anterior es posible utilizando los *procedimientos* o *funciones*.

[6] HLL del ingle *High Level Lenguage*.
[7] Prefijo (del gr. *makros*), que significa muy grande.

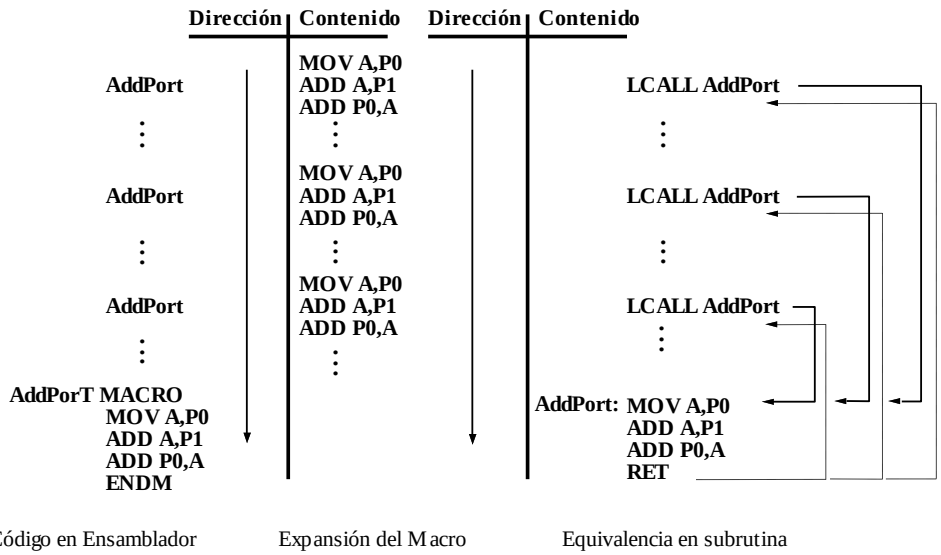


Figura 2.2 Comparación de la expansión de macros y la utilización de subrutinas.

Los terminos *función* y *procedimiento* son algunas veces utilizados como términos equivalentes, pero existen distinción entre ellos. Las funciones siempre retornan un valor y los procedimientos no.

Procedimientos

El procedimiento o *subrutina* es una parte importante de la arquitectura del sistema de cualquier computadora. Un procedimiento es un grupo de instrucciones que usualmente desarrollan una tarea. Un procedimiento es una porción que tiene la posibilidad de ser nuevamente usada dentro del programa y que se almacena en memoria una vez, pero se usa tan seguido como sea necesario. Esto permite ahorrar espacio de memoria y hace más fácil el desarrollo de programas. La única desventaja de un procedimiento es que hacen que la computadora tome una pequeña cantidad de tiempo para encadenarse al procedimiento y regresar de él. La instrucción LCALL se encadena a las subrutinas, y la instrucción RET hace que se regrese de ella.

La pila almacena la dirección de retorno siempre que una subrutina es llamada durante la ejecución del programa. La instrucción LCALL empuja en la pila la dirección de la siguiente instrucción. La instrucción RET remueve una dirección de la pila así que el programa regresa a la siguiente instrucción del LCALL. La figura 2.2 muestra la comparación de el uso de macros y subrutinas.

2.5 PROGRAMACIÓN EN LENGUAJE "C"

Este lenguaje es el producto final de una serie de tres pasos que se dieron para diseñar un lenguaje de programación. El primero fue BCPL, escrito por Martín Richards, el segundo fue B, producido por Ken Thompson, y el tercero fue C, creado por Dennis Ritchie en 1972 en los Laboratorios Bell establecido en Murray, Nueva Jersey. Originalmente fue diseñado para procesarse con un sistema operativo UNIX de la minicomputadora PDP-11, de Digital Equipment Coporation, pero después se extendió a otras máquinas y sistemas operativos. Actualmente tiene un gran aceptación en computadoras personales.

El lenguaje C aún considerado como lenguaje de alto nivel, también permite el acceso a la programación a bajo nivel (haciendo referencia al nivel de bits) esto lo hace muy poderoso en la programación de sistemas. Lo que ha contribuido a su popularidad son los siguientes puntos:

1) Maneja todo tipo de organización de datos.
2) Tiene un completo conjunto de operadores y un moderno control de estructuras.
3) Maneja bibliotecas que facilitan el manejo de la entrada y salida de datos.
4) Es eficaz y compacto.
5) Tiene un alto grado de portabilidad.

El siguientes listado muestra un pequeño programa escrito en lenguaje C que imprime los números impares del 1 al 100;

```
/* programa escrito en lenguaje C */

#include <stdio.h>

#define MAX   100

main()
{
      int num=1;
      while(num<MAX)
      {
            if(num%2)
            {
              printf('%d  ',num);
            }
            num++;
      }
}
```

El lenguaje C tiene una jerarquía descendiente de funciones con formato para entrada o salida, posee las funciones printf() y scanf() descendientes de funciones de E/S para un sólo carácter como putchar(), putc(), getchar() y getc(); las cuales a su vez están definidas de secuencias de lenguaje ensamblador.

Algunas veces es necesario escribir funciones en ensamblador, C permite que se pueda mezclar rutinas de ensamblador en el programas, ya sea de una manera externa o interna (directamente en el programa). Por ejemplo, la secuencia del macro **AddPort** puede ser codificada en una línea de C de la siguiente manera:

```
asm(" AddPorts: \MOV A,P1 \  ADD A,P2  \ MOV P1,A");
```

De esta manera puede verse ese poder del lenguaje C, puesto que tiene las estructuras de control de un lenguaje de alto nivel, así como la posibilidad de manejar código ensamblador directamente, para así tener acceso directo a registros de microcontrolador o microprocesador.

2.6 COMPARACIÓN LENGUAJE "C" Y LENGUAJE ENSAMBLADOR

La comparación del lenguaje C y el lenguaje ensamblador puede ser realizada sobre algunos niveles. El primer nivel es la productividad, como una medida en el tiempo de producir un código ejecutable sin errores, C tiene la ventaja en este nivel. Esto es debido a que C trabaja con sentencias de mucho mas alto nivel que ensamblador, y ademas el programador no tiene la preocupación del contenido de cada registro y bits de estado, así puede concentrarse más en el flujo lógico del problema.

Un segundo nivel es la accesibilidad al programador de los recursos del hardware de la computadora, microcontrolador o sistema embebido. Obviamente todos esos recursos están disponibles para el programador en el lenguaje ensamblador, sin embargo muchos de estos recursos (si no es que todos) pueden ser accesados por el lenguaje C. En este punto el lenguaje C no se queda atrás. Por ejemplo, si se desea escribir o leer de una localidad o un puerto del la sección de E/S mapeada sobre la sección de memoria, un apuntador puede declararse para el tipo de acceso deseado. Por ejemplo, si el hardware tiene un puerto de 8 bits para controlar un convertidor A/D, en este lenguaje se puede utilizar una directiva para direccionar el puerto y un apuntador a un carácter[8] sin signo, como es mostrado a continuación.

```
#define CTL_AD 0x10ff       /* definición del macro CTL_AD igual a 0x10ff      */
unsigned char *DireccAD     /* declaración del apuntador a carácter DireccAD   */
```

Una vez hecho esto, es necesario hacer la asignación de la dirección del A/D al apuntador con la instrucción:

```
DireccAD=(unsigned char *) CTL_AD;       /*asigna dirección del A/D */
```

Ahora, el control del A/D puede hacerse mediante un acceso por referencia al apuntador como:

```
*DireccAD=Palabra_de_Control;      /* inicializa el convertidor para lectura */
```

[8] Se refiere a un tipo de variable de 8 bits en lengauje C llamado *char*.

En algunos casos cuando se lee un convertidor A/D es necesario esperar un bit (EOC[9]) que proporciona el convertidor para indicar que la conversión a finalizado y el dato es válido. En C esta espera de ese bit se puede hacer mediante:

```
while(!(*DireccAD & EOC));              /* espera  conversión completa */
```

donde EOC esta definido con 0x80, como una mascara para verificar únicamente el bit mas significativo del valor leído de la dirección de control del convertidor. En la instrucción se puede ver un símbolo '&' este es el operador AND para bits en C, ademas de este operador, C posee una gran diversidad símbolos para operadores lógicos para bits (bitwise), la tabla 2.1 muestra todos los operadores lógicos que maneja C.

Operador	Acción
&	AND
|	OR
^	OR exclusiva (XOR)
~	NOT
>>	Corrimiento a la derecha
<<	Corrimiento a la izquierda.

Tabla 2.1 Operadores para bits del lenguaje C.

Ejemplos:

```
char A=15;      /* A=00001111      */

B=A|0x80;       /* B=10001111      */

B=~A;           /* B=11110000      */

B=A<<3;         /* B=01111000      */

B=A>>2;         /* B=00000011      */
```

2.7 DESARROLLO DE SOFTWARE

Un enfoque muy utilizado en el desarrollo de programas es el enfoque sistemático llamado *top-down*. En este enfoque un gran problema es partido en pequeños *módulos*. El nivel más alto muestra relaciones y funciones entre esos módulos. Este nivel muestra un vista general del programa completo. Cada uno de los módulos es dividido en módulos aun mas pequeños. La división es continuamente hasta que los pasos en cada modulo sean claramente comprendidos. Ahora a cada programador se le asigna un módulo o un conjunto de módulos para que escriba el programa de dicho módulo.

[9] EOC del ingles *End Of Convertion*.

Un enfoque opuesto a esto es el diseño *botton-up*. En este enfoque cada programador inicia escribiendo módulos en bajo nivel y esperan que todas las piezas sean unidas posteriormente. Muchos equipos modernos de programación usan una combinación de estas técnicas. El desarrollo de herramientas para programación fue ayudado por el descubrimiento de que cualquier programa puede ser representado por tres tipos de operaciones básicas. Primer tipo de operación básica es la *secuencia* la cual significa simplemente hacer una serie de acciones. El segundo tipo de operación es la *decisión* o *selección* esto significa escoger entre dos o más acciones. La tercera operación básica es la *repetición* o *iteración*, la cual significa repetir una serie de acciones hasta que una condición esté o no presente.

Todos los programadores utilizan tres a siete estructuras estándares para representar todas las operaciones en sus programas. Actualmente sólo tres estructuras (SECUENCIA, IF-THEN-ELSE y WHILE-DO) se requieren para representar cualquier acción de un programa, las otras cuatro estructuras se derivan de las ya mencionadas. La representación gráfica de estas estructuras se muestra en la figura 2.3 y 2.4.

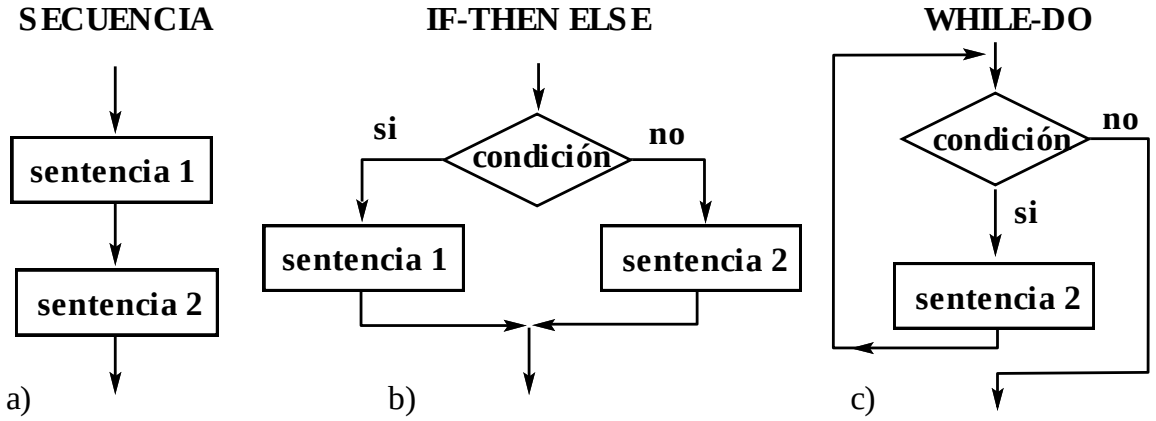


Figura 2.3 Estructuras básicas para representación de acciones en programas.
a) SENTENCIA b) IF-THEN-ELSE y c) WHILE-DO.

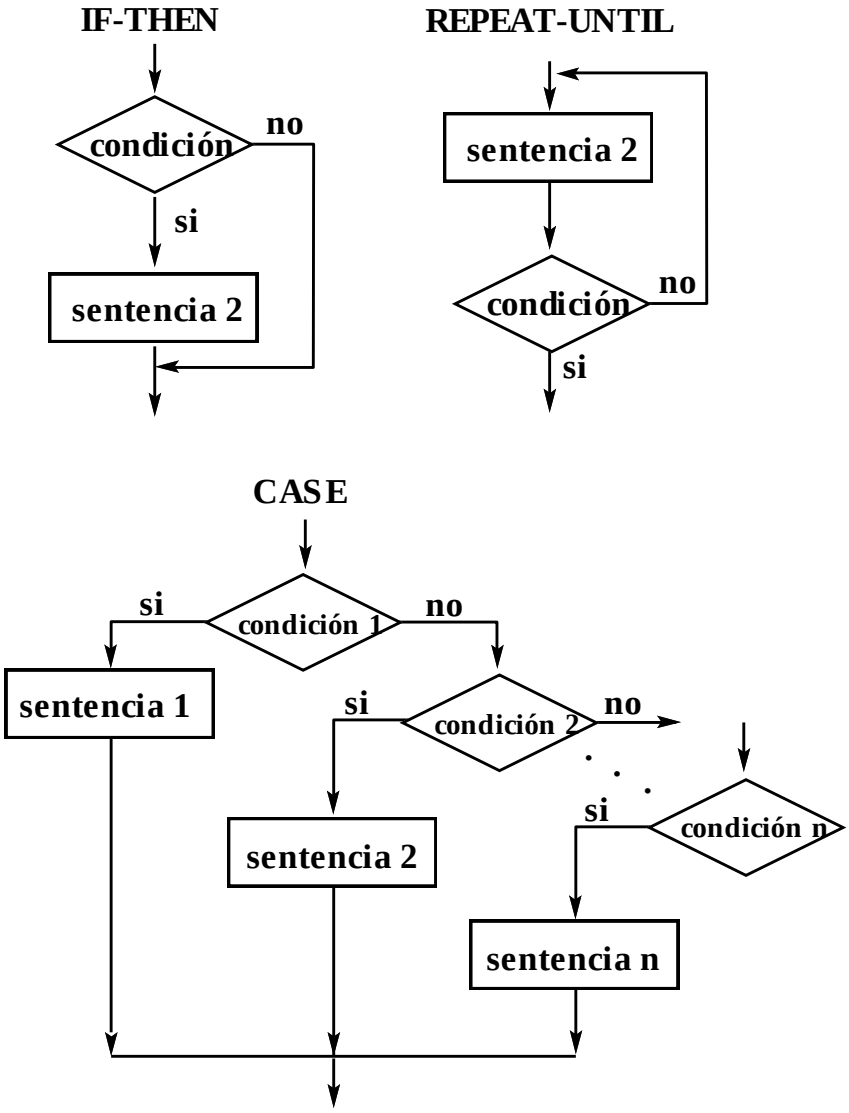


Figura 2.4 Estructuras derivadas para representación de acciones en programas.
a) IF-THEN b) REPEA-UNTIL y c) CASE.

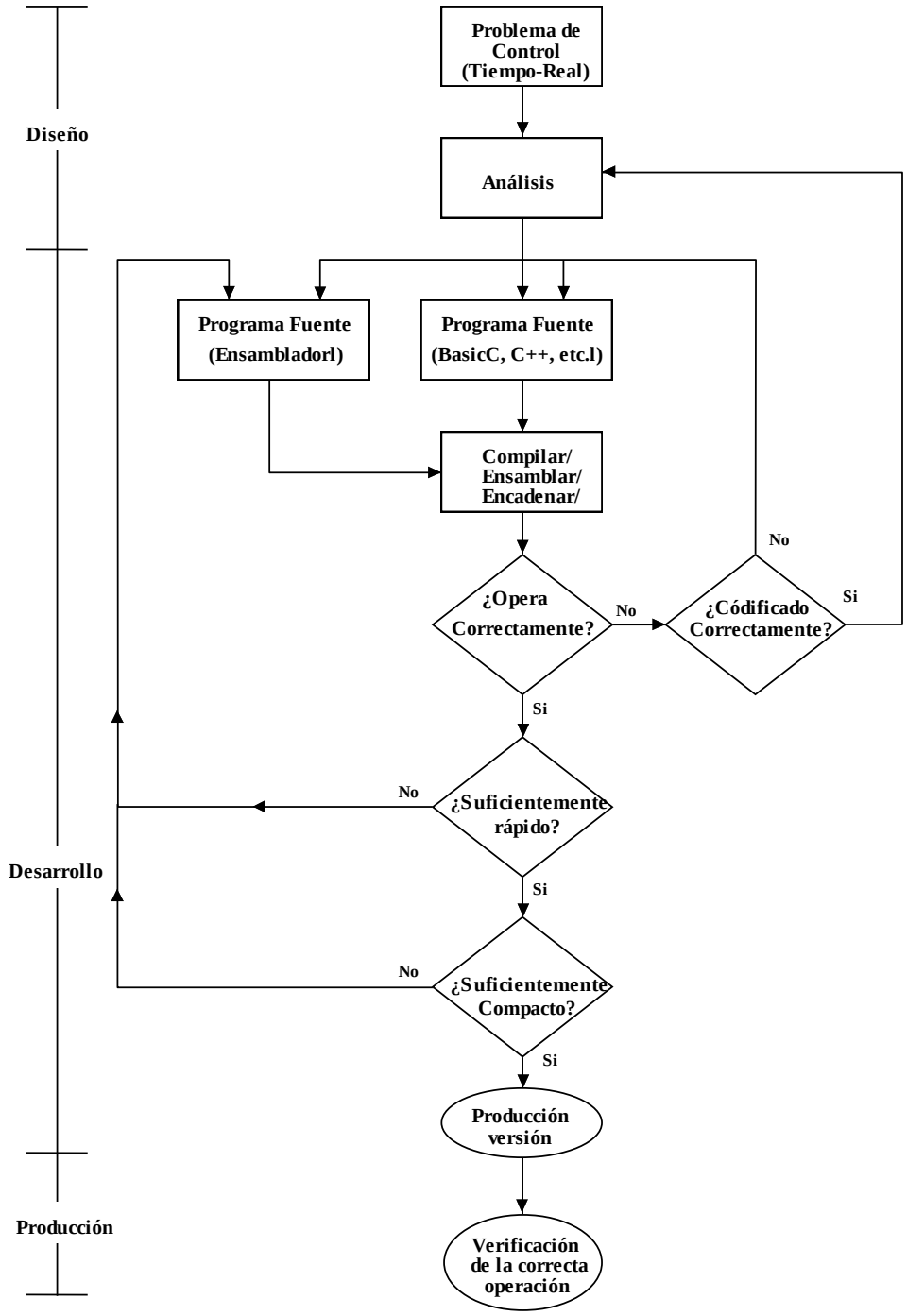


Figura 2.5 Proceso de Arriba hacia abajo (Top-down) para el desarrollo de programas para microcontroladores.

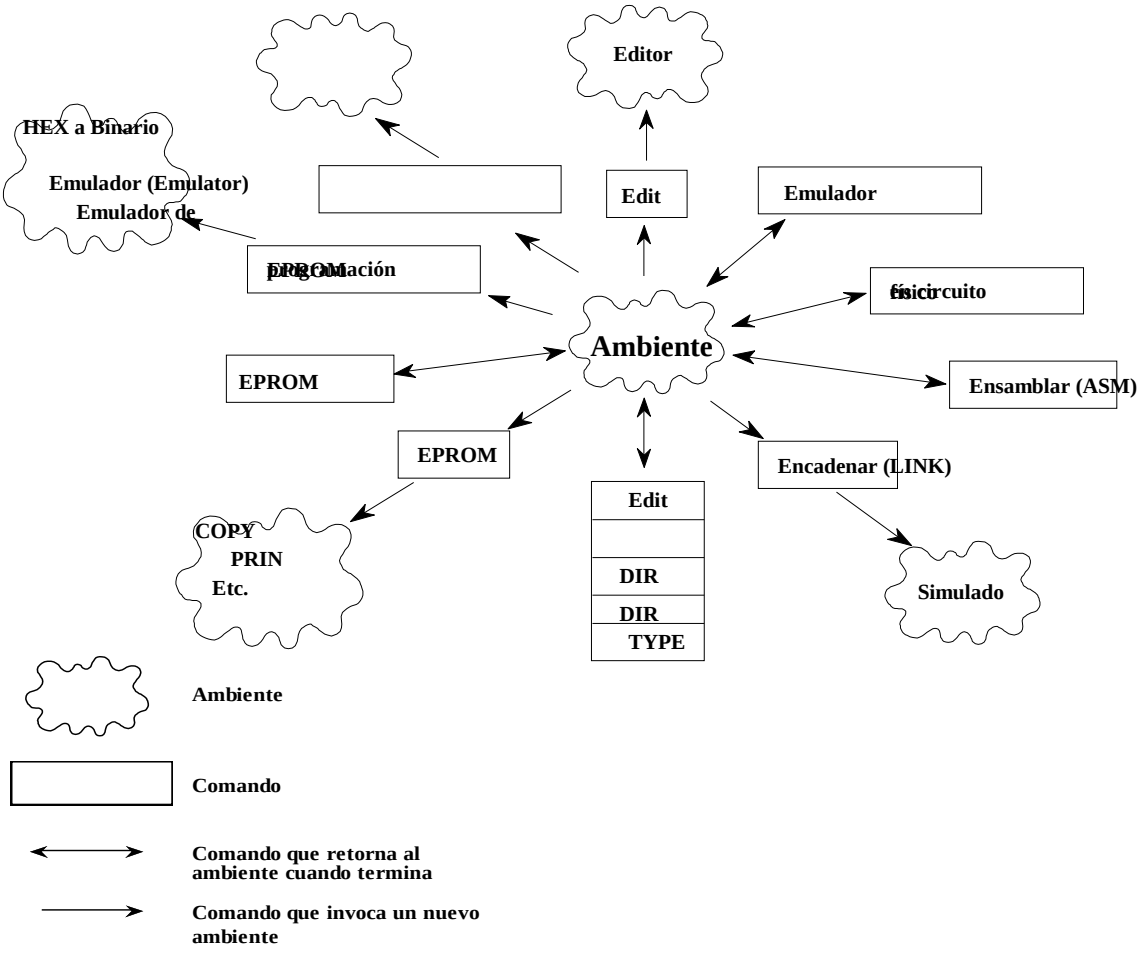


Figura 2.6 Ambiente del Desarrollo de programas para microcontroladores.

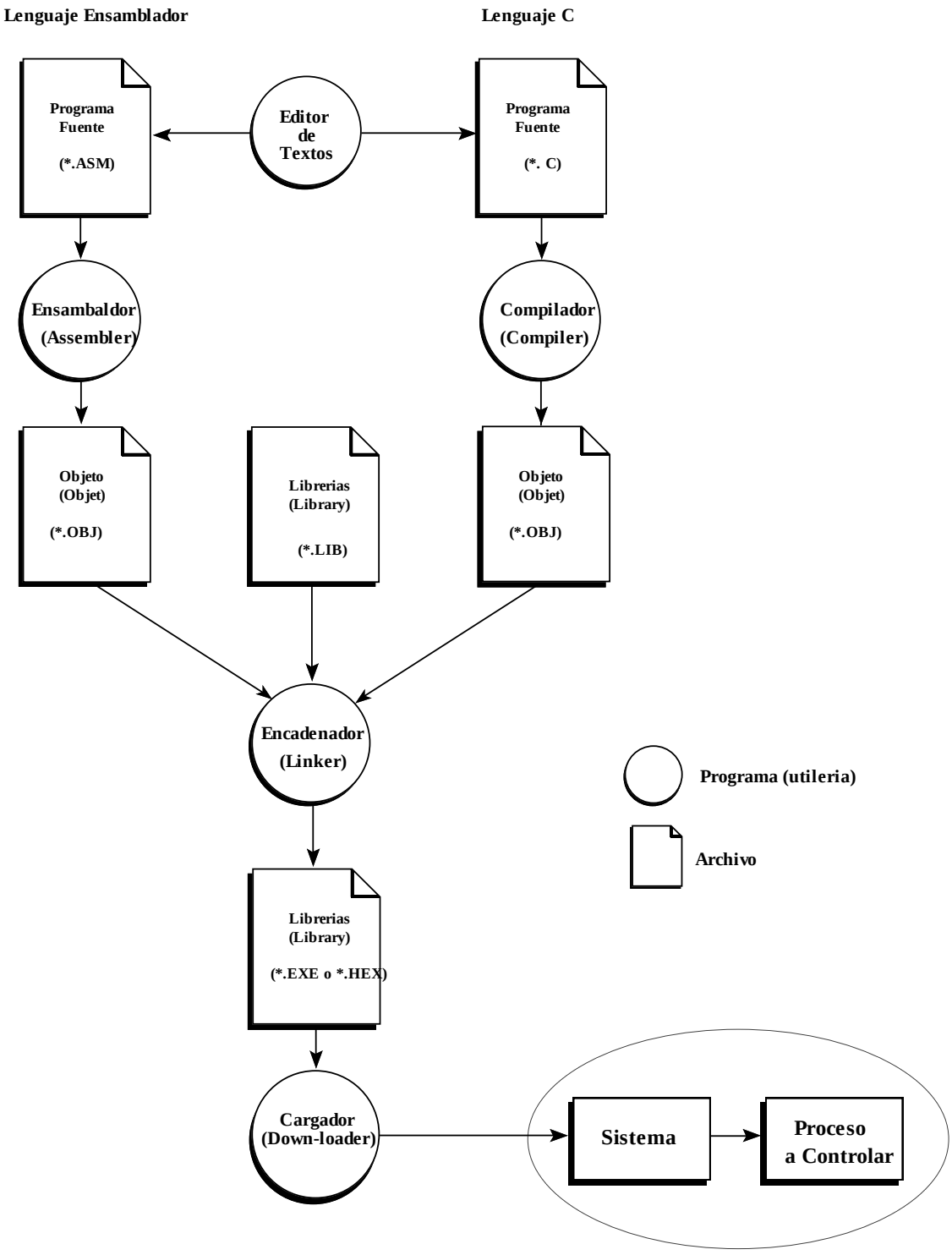


Figura 2.7 Diagramas de bloques de la conversión de programas de un código fuente a un programa ejecutable por el microcontrolador.