INTRODUCCIÓN BLOQUE III

Este es el tercero de un total de cuatro libros que forman el temario actualizado de Oposiciones de Secundaria de **Sistemas y Aplicaciones Informáticas**. Su objetivo fundamental es el de facilitar al opositor la preparación de la prueba escrita.

El contenido de este libro está desarrollado basándose en la legislación actual que regula el contenido de estas pruebas y está completamente adaptado a la especialidad de **Sistemas y Aplicaciones Informáticas**.

Este volumen contiene los temas desde el 33 al 48, de los 65 que componen el temario de Informática de Secundaria de Sistemas y Aplicaciones Informáticas. En estos temas se desarrollan los bloques de Lenguaje C, Bases de Datos y Aplicaciones Informáticas, ofreciendo un contenido totalmente actualizado recogiendo las últimas novedades en las disciplinas que se presentan.

Cada uno de los temas consta de un índice que presenta el esquema general del tema, la introducción, el desarrollo del tema en cuestión, una conclusión y bibliografía/webgrafía.

En el apartado de la introducción aparece un subapartado que recoge la **contextualización** de cada tema. La contextualización está especificada para los módulos de la especialidad de Sistemas y Aplicaciones Informáticas para todos los niveles: **formación profesional básica, ciclo de grado medio, ciclos de grado superior y cursos de especialización**.

En la prueba escrita es recomendable que se introduzca el punto de bibliografía/webgrafía al final del tema. En este libro se presenta la bibliografía agrupada por bloques con el fin de facilitar al opositor la tarea de recordarla.

Los temas se presentan de forma acotada en un número de páginas tal que permita al opositor desarrollarlo en el tiempo estipulado, asegurando que se tratan todos los puntos de interés con la profundidad adecuada.

Los temas pertenecientes al mismo bloque tienen contenidos en común, lo que permitirá al opositor rentabilizar tiempo de estudio y poder amortizar píldoras de conocimiento aplicables a distintos temas.

Además, este volumen viene acompañado de material adicional en el que el lector puede encontrar trucos sobre cómo afrontar el examen, ejemplos para añadir a los temas y otros recursos de interés.

TEMA 33

PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR. INSTRUCCIONES BÁSICAS. FORMATOS. DIRECCIONAMIENTOS

33.1 INTRODUCCIÓN 20
33.1.1 Contextualización 21
33.2 PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR 21
33.2.1 Directivas 22
33.2.2 Macros 24
33.3 INSTRUCCIONES BÁSICAS 24
33.3.1 Instrucciones de Transferencia de Datos 24
33.3.2 Instrucciones Aritméticas 25
33.3.3 Instrucciones Lógicas 26
33.3.4 Instrucciones de Transferencia de Control 26
33.3.5 Bucles 27
33.3.6 Llamada a procedimientos 27
33.3.7 Instrucciones para Manejo de Cadenas 28
33.3.8 Instrucciones de Control de Flags 28
33.3.9 Instrucciones de entrada/salida 28
33.4 FORMATOS 29
33.5 DIRECCIONAMIENTOS 29
33.6 REGISTROS 31
33.6.1 Registros de propósito general 32
33.6.2 Registros de Segmento 32
33.7 EJEMPLO PRÁCTICO 34
33.8 CONCLUSIÓN 34
33.9 BIBLIOGRAFÍA

Tema 33

PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR. INSTRUCCIONES BÁSICAS. FORMATOS. DIRECCIONAMIENTOS

33.1 INTRODUCCIÓN

La forma en la que el usuario especifica al ordenador como tratar la información y los datos se realiza a través de lo que se conoce como lenguaje de programación. Dentro de la clasificación que se puede realizar de los lenguajes de programación, el lenguaje ensamblador está categorizado como un lenguaje de bajo nivel, de hecho, se caracteriza por su gran proximidad al lenguaje máquina, sus instrucciones son reglas mnemotécnicas que se aplicaron en su día para hacer su correspondiente traducción al lenguaje máquina.

En 1978 Intel presentó el procesador 8086, esta arquitectura implementa el lenguaje de programación ensamblador como método de programación más avanzada en comparación con el código máquina. Poco después, desarrolló una variación del 8086 para ofrecer un diseño más sencillo y compatible con los dispositivos de entrada/salida de ese momento. Este nuevo procesador, el 8088, fue seleccionado por IBM para su PC en 1981.

El procesador 8086 tiene una arquitectura de 16 bits, lo que implica que trabaja con registros de 16 bits separados en parte alta y parte baja. Con su estructura de 16 bits y teniendo registros de 8 y 16 bits se puede llegar a tener una capacidad de direccionamiento en memoria de 20 bits (1MB). Cada procesador tiene su propio lenguaje ensamblador y viene determinado por el fabricante del hardware.

A lo largo del tema se verán desde las estructuras más sencillas, como los registros que utiliza la arquitectura y cómo acceder a ellos utilizando el lenguaje ensamblador, hasta un ejemplo de un sencillo programa en lenguaje ensamblador.

En la actualidad el lenguaje ensamblador está casi siempre controlado por los compiladores, aun así todavía se utiliza para la manipulación directa del hardware. Por ejemplo es usado para escribir programas embebidos ya que usan muy poca memoria. Su uso más común es en la programación de microcontroladores.

33.1.1 Contextualización[1]

El tema de lenguaje ensamblador es un tema obsoleto que no forma parte del temario explícito de ningún módulo o asignatura, sin embargo, podría contextualizarse de manera transversal en cualquier módulo inicial de Sistemas Operativos o Hardware donde se explique cómo opera físicamente un equipo informático.

Así pues, este tema se contextualiza, aunque transversalmente, en los siguientes módulos:

- 3030. Operaciones auxiliares para la configuración y la explotación, perteneciente al Título Profesional Básico en Informática y Comunicaciones (R.D. 127/2014).
- 3030. Operaciones auxiliares para la configuración y la explotación. Perteneciente al Título Profesional Básico en Informática de Oficina (R.D. 356/2014).
- 0222. Sistemas Operativos Monopuesto del título de Técnico en Sistemas Microinformáticos y Redes (R.D. 1691/2007).
- 0369. Implantación de Sistemas Operativos del título de Técnico Superior en Administración de Sistemas Informáticos en Red (R.D. 1629/2009).
- 0483. Sistemas Informáticos que pertenece al título de Técnico Superior en Desarrollo de Aplicaciones Web definido en el R.D. 450/2010 y al título de Técnico Superior en Desarrollo de Aplicaciones Multiplataforma definido en el R.D. 686/2010.

33.2 PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR

El lenguaje ensamblador, como ya se ha comentado en la introducción, es el que se utiliza en la programación de programas informáticos de bajo nivel. Surgió para sustituir a la programación en lenguaje máquina ya que era muy complicado recordar posiciones de memoria y recordar los códigos de cada instrucción.

---

1 El opositor debe recordar que esta contextualización debe ser completada con la legislación perteneciente a su Comunidad Autónoma.

Un programa escrito en lenguaje ensamblador es difícil de entender puesto que está más cercano al lenguaje que entienden las máquinas que al que están acostumbrados hoy en día los programadores, que es, normalmente, el lenguaje de alto nivel.

El programa escrito en lenguaje ensamblador se llama código fuente y se almacena con la extensión `.asm`. A partir de ese fichero se obtiene el correspondiente código máquina. Los programas escritos en ensamblador tienen la característica de ser rápidos puesto que están programados directamente sobre el hardware, además ocupan mucho menos espacio de memoria que los lenguajes de alto nivel.

Un programa escrito en ensamblador está formado por tres secciones:

- **.data** para los datos inicializados.
- **.bss** para los datos no inicializados. Esta parte no es necesaria si se inicializan las variables en el apartado de .data.
- **.text** para el código. En esta sección también se pueden definir variables, por lo tanto se podría prescindir del bloque .data, pero no es recomendable.

33.2.1 Directivas

Las directivas son comandos que tienen influencia en el momento del ensamblado, por lo tanto no generan código ejecutable. A menudo se confunden con las instrucciones, y no son instrucciones, a pesar de que en algún caso en concreto se pueden añadir a un programa.

Las directivas sirven para declarar algunos elementos del programa para que sean más fácilmente identificables por el programador y para facilitar la tarea de ensamblaje.

A continuación, se muestran las directivas de ensamblador más comunes:

Operación	`TITLE`
Descripción	Indica al comienzo del programa un título.
Ejemplo	`TITLE texto`

Operación	`PROC`
Descripción	Sirve para definir un procedimiento.

Ejemplo	`PROC`
	`//líneas de código`
	`ENDP`

Operación	`.MODEL`
Descripción	Especifica el modelo de memoria.
Ejemplo	`.MODEL compact`

Operación	`.DATA`
Descripción	Lo que venga en este apartado se guarda en el segmento de datos.
Ejemplo	`.DATA`

Operación	`.CODE`
Descripción	Lo que venga en este apartado se guarda en el segmento de código.
Ejemplo	`.CODE`

Operación	`.STACK`
Descripción	Lo que venga a continuación se guarda en el segmento de pila.
Ejemplo	`.STACK`

Operación	`END`
Descripción	Finaliza el programa.
Ejemplo	`END`

Operación	`EQU`
Descripción	Asigna un nombre al valor de una expresión.
Ejemplo	`COLUMNAS EQU 80` `FILAS EQU 25`

Directivas para definición de datos

Operación	`DB`
Descripción	Define un byte de información.
Ejemplo	`VALOR DB 111` `TEXTO DB “hola”`

DB	1 byte
DW	2 bytes
DD	4 bytes
DF	6 bytes
DQ	8 bytes
DT	10 bytes

Operación	OFFSET
Descripción	Devuelve el desplazamiento dentro del segmento indicado.
Ejemplo	MOV DX, OFFSET TEXTO

Operación	DUP (duplicate)
Descripción	Repite num. veces
Ejemplo	DB 20 DUP(100) //REPITE 20 VECES 100

33.2.2 Macros

Una macro es un símbolo que representa un bloque de texto. Las macros son definidas por el usuario para unir una o un conjunto de líneas fuente de ensamblador. Una macro una vez definida puede usarse como una directiva de ensamblador o como cualquier otro mnemotécnico del lenguaje ensamblador.

El ensamblador usa las macros de tal manera que cada vez que encuentra al símbolo que define la macro lo sustituirá en el código por el código de la macro, ejecuta código muy rápidamente, pero por el contrario, aumenta el espacio cuando genera el fichero compilado.

Su sintaxis es la siguiente:

```
<Nombre> MACRO [parámetros] ... ENDM
```

33.3 INSTRUCCIONES BÁSICAS

33.3.1 Instrucciones de Transferencia de Datos

Mueven datos entre registros y posiciones de memoria.

Operación	`MOV`
Descripción	Mueve datos entre dos registros de memoria.
Ejemplo	`MOV AX, CX`

Operación	`XCHG`
Descripción	Intercambia la información de dos registros.
Ejemplo	`XCHG BL,BH`

Operación	`PUSH`
Descripción	Guarda en la pila un dato.
Ejemplo	`PUSH AX`

Operación	`POP`
Descripción	Extrae de la pila en valor que hay en el top y lo quita de ella.
Ejemplo	`POP CX`

33.3.2 Instrucciones Aritméticas

Estas instrucciones sirven para realizar operaciones.

Operación	`ADD`
Descripción	Realiza la suma de dos registros y deposita el resultado en el destino.
Ejemplo	`ADD CX, AX`

Operación	`SUB`
Descripción	Realiza la resta de dos registros y deposita el resultado en el destino.
Ejemplo	`SUB AX, CX`

Operación	`MUL`
Descripción	Multiplica dos valores.
Ejemplo	`MUL AX  BX`

Operación	`DIV`
Descripción	Realiza la división de dos registros.
Ejemplo	`DIV AX BX`

33.3.3 Instrucciones Lógicas

Operación	`AND`
Descripción	Realiza la Y lógica entre dos operandos y activa el flag correspondiente.
Ejemplo	`AND AL AH`

Operación	`OR`
Descripción	Realiza la O lógica entre dos operandos y activa el flag correspondiente.
Ejemplo	`OR AL AH`

Operación	`NOT`
Descripción	Complementa todos los bits.
Ejemplo	`NOT CX`

33.3.4 Instrucciones de Transferencia de Control

En este apartado se verán aquellas instrucciones más relevantes para alterar el flujo de control del programa.

Instrucción		Flag
`JE/JZ`	Salto si igual	Z=1
`JNE/JNZ`	Salto si no igual	Z=0
`JA/JNBE`	Salto si superior	C=0 Y Z=0
`JAE/JNB`	Salto si superior o igual	C=0
`JB/JNAE`	Salto si inferior	C=1
`JBE/JNA`	Salto si inferior o igual	C=1 O Z=1

Operación	`JMP`
Descripción	Salto incondicional
Ejemplo	`JMP Final // salto a la etiqueta Final`

33.3.5 Bucles

Un bucle es un grupo de instrucciones que se ejecutan un número concreto de veces y se repiten hasta que haya una condición de parada.

Operación	`LOOP`
Descripción	El registro CX hace de contador y en cada pasada se decrementa.

Ejemplo	`MOX CX, AX. //AX tiene almacenado el número de veces que se ejecuta el loop.`
	
	`Etiqueta:`
	`  instrucciones`
	`LOOP Etiqueta`

33.3.6 Llamada a procedimientos

Los procedimientos son zonas de código a las que se les puede llamar, se utilizan para reutilizar código.

Operación	`CALL`
Descripción	Llama a un procedimiento
Ejemplo	`CALL procedimiento`

Operación	RET
Descripción	Vuelve de un procedimiento
Ejemplo	RET

33.3.7 Instrucciones para Manejo de Cadenas

Una cadena es una secuencia de bytes contiguos. Las operaciones que se pueden realizar sobre las cadenas son las siguientes:

Operación	LODSB/LODSW
Descripción	Carga el registro acumulador con el valor referenciado en DS:DI
Ejemplo	LODSB

33.3.8 Instrucciones de Control de Flags

Se puede poner a 1 o a 0, con las instrucciones `Clear` y `Set`.

CLD	DF=0
STD	DF=1
CLI	IF=0
STI	IF=1

Estas instrucciones permiten manipular los bits del registro de estado.

33.3.9 Instrucciones de entrada/salida

Los puertos de entrada y salida (E/S) permiten al procesador comunicarse con los periféricos (pantalla, impresora, etc.). Para acceder a los puertos de entrada y salida se utilizan las instrucciones `IN` y `OUT`.

Operación	IN
Descripción	Desde el registro AX se transfiere a un puerto.
Ejemplo	IN AL, 0Bh

Operación	`OUT`
Descripción	Transfiere la información desde el registro al puerto.
Ejemplo	`OUT AL, 0BH`

33.4 FORMATOS

El lenguaje ensamblador está compuesto por una sucesión de líneas de texto. Cada línea puede tener hasta cuatro campos o columnas separadas entre sí por espacios o tabulaciones.

El formato general de una instrucción en lenguaje ensamblador es como se muestra a continuación:

```
[etiqueta:] instrucción [destino [, fuente] ] [;comentario]
```

Aclaraciones con respecto al formato:

- Las etiquetas hacen referencia a un elemento del programa. Sirven para que el programador pueda referenciar a los distintos elementos del programa. Sirven para definir variables, constantes y posiciones de código. Deben comenzar por un carácter alfabético o con ciertos símbolos determinados.

  Ejemplo de etiqueta:

  ```
  Etiqueta4: mov rax, 0
  ```

- Hay que tener en cuenta el uso de mayúsculas y minúsculas puesto que en lenguaje ensamblador sí se distingue entre ellas.
- `destino` y `fuente` representan los operandos de la instrucción.
- Los elementos entre corchetes [] son opcionales.
- El único elemento imprescindible es el nombre de la instrucción.

33.5 DIRECCIONAMIENTOS

El modo de direccionamiento indica la manera en la que se va a acceder a la información. Cuando un operando está en memoria hay que considerar qué tipo de direccionamiento se va a utilizar para acceder a un dato concreto.

A continuación se muestran los distintos tipos de direccionamientos del lenguaje ensamblador, cabe destacar que el más habitual es el direccionamiento a registro.

Direccionamiento directo a registro

El operando hace referencia a un dato almacenado en un registro, se puede especificar cualquier registro de propósito general (registros de datos, registros índices y registros apuntadores).

Ejemplo:

```
mov rax, rbx
```

Direccionamiento inmediato

El operando se especifica en la instrucción, es decir, hace referencia a un dato que se encuentra en la misma instrucción, no hay que acceder a memoria. Solo se puede utilizar un direccionamiento inmediato como operando fuente.

Ejemplo:

```
mov rbx,var    ;carga la dirección de var en el registro
               ;rbx
```

Direccionamiento directo a memoria

El operando hace referencia a un dato almacenado en una posición de memoria.

Ejemplo:

```
mov rax, [var]    ;el segundo operando utiliza
                  ;direccionamiento directo a memoria,
                  ;rax=[var]
```

Direccionamiento Indexado

El operando hace referencia a un dato que está almacenado en una posición de memoria. La dirección de memoria es una dirección base que se puede expresar como una suma de registros y una variable, por ejemplo.

Ejemplo:

```
mov rax, [vector+rsi]    ;vector contiene la dir. base
                         ;rsi es el registro índice
```

Direccionamiento Indirecto al registro

El operando hace referencia a un dato almacenado en una posición de memoria. Se debe especificar el registro entre corchetes [].

Ejemplo:

```
mov rax,[rbx]    ;el segundo operando utiliza la dirección tenemos para acceder
                 ;a memoria
```

Direccionamiento Relativo

El operando hace referencia a un dato almacenado en una posición de memoria. Se da direccionamiento relativo cuando se especifica un registro sumado a un número entre corchetes.

Ejemplo:

```
mov rax,[rbx+4]    ;el segundo operando utiliza direccionamiento relativo,
                   ;4 es el desplazamiento respecto a la dirección
```

Direccionamiento relativo a PC

Se permite direccionamiento relativo a PC en cualquier instrucción del modo de 64 bits del PC. Este tipo de direccionamiento se reserva para las instrucciones de salto condicional.

Ejemplo:

```
je etiqueta5
```

Direccionamiento a pila

Es un tipo de direccionamiento implícito, se trabaja con la cima de la pila mediante el registro apuntador a pila. Solo existen dos instrucciones específicas en este direccionamiento.

Ejemplo:

```
push fuente    ;coloca un dato en la pila
pop destino    ;extrae un dato de la pila
```

33.6 REGISTROS

Los registros del procesador son espacios de memoria etiquetados para poder identificarlos. El 8086 dispone de 14 registros de 16. Los registros pueden

ser identificados por medio de un nombre. Como se ha visto en los modos de direccionamiento, con el lenguaje ensamblador se hace referencia a los registros. Por ello en este punto se van a mostrar los distintos registros que se pueden encontrar.

Los diferentes registros del 8086 se clasifican en:

- Registros de propósito general o de datos.
- Registros de segmento.
- Registro apuntador de instrucciones (`IP`).
- Registros base (`SP` y `BP`).
- Registros índices (`SI` y `DI`).
- Registro de estado (`FL`).

AX		SP		CS		IP
BX		BP		DS		FLAGS
CX		SI		SS		
DX		DI		ES		
Registros de propósito general		Registros punteros e índice		Registros de segmentos		Puntero a instrucciones y flags

33.6.1 Registros de propósito general

Los registros de propósito general se usan para cálculo y almacenamiento de propósito general. Hay cuatro registros de propósito general:

- `AX`: registro acumulador, participa de las operaciones aritméticas y lógicas como un operando más.
- `BX`: registro base, participa en cálculos de direcciones de acceso a memoria y operaciones aritméticas y lógicas.
- `CX`: registro Contador, se suele utilizar como contador en bucles.
- `DX`: registro de datos.

Todos estos registros, son registros de 16 bits, lo que implica que se pueden dividir en parte alta y baja, como si fueran registros de 8 bits. Por ejemplo el registro `AX`, se puede dividir en `AH` (high) y `AL` (low).

33.6.2 Registros de Segmento

Los registros de segmento son registros de 16 bits que dividen el espacio de memoria en segmentos, cada uno tendrá un propósito.

Registro CS	Registro de segmento de código, establece la zona de memoria donde se alojarán las instrucciones del programa.
Registro DS	Registro de segmento de datos, especifica la zona de memoria donde se leen y escriben datos.
Registro SS	Registro de segmento de pila, zona de memoria donde se almacena temporalmente información o direcciones de memoria.
Registro ES	Registro de segmento extra, se suele utilizar para operaciones con strings.

- Registro Apuntador de Instrucciones (`IP`): registro de 16 bits que contiene la dirección de la siguiente instrucción a ejecutar.
- Registros de pila (`SP` y `BP`). Permiten acceder a la pila de memoria, a la base como al top (Stack pointer y Base pointer).
- Registros Índice (`SI` y `DI`). Los registros índices se utilizan fundamentalmente en operaciones con cadenas y para direccionamiento indexado
- Registro de `FLAGS`, o registro de estado (`FL`). Es un registro de 16 bits, pero sólo se utilizan nueve de ellos. Sirven para indicar el estado actual de las operaciones que se estén realizando en ese momento.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**O**	**D**	**I**	**T**	**S**	**Z**	**-**	**A**	**-**	**P**	**-**	**C**

`OF`	Bit de overflow, indica desbordamiento.
`DF`	Bit de dirección, indica la dirección en operaciones de cadena.
`IF`	Bit de interrupción, detecta si hay alguna interrupción a procesar.
`TF`	Bit de trap, ignora la interrupción en curso.
`SF`	Bit de signo, indica el signo de después de una operación.
`ZF`	Bit de Zero, indica si la operación ha devuelto Zero.
`AF`	Bit de Carry auxiliar, indica si en la operación hay un acarreo, indicado para operaciones BCD.
`PF`	Bit de paridad, indica si el resultado es par.
`CF`	Bit de Carry, indica si en la operación hay un acarreo.

33.7 EJEMPLO PRÁCTICO

A continuación se realizará el típico hola mundo pero en código ensamblador.

```
.model small   //se define un modelo corto
.data         //variables para el segmento de datos

mensaje db 'Hola mundo'

.code         //segmento de código

inicio       //se define el comienzo del programa

mov ax,@data //se mueve al registro de datos los datos a usar

mv al,09h    //en él se cargan las funciones
mov dx, offset mensaje

int 21h      //se lanza la interrupción para que se //muestre por pantalla
end inicio   //se finaliza el programa
```

33.8 CONCLUSIÓN

Cuando surgió la programación, casi todos los programas estaban escritos en lenguaje ensamblador, si no completamente, al menos una parte de ellos. Cuando los ordenadores fueron evolucionando y los procesadores adquirieron más velocidad, los programas eran bastante más complejos y ya no podían desarrollarse en lenguaje ensamblador. Es entonces cuando surgieron los lenguajes de programación de alto nivel, como por ejemplo C o Java.

En principio puede parecer atractivo crear un programa en ensamblador, pero el principal inconveniente que se tiene actualmente es que los conceptos que se han desarrollado como por ejemplo: objetos, herencia, clases, son excesivamente largos y tediosos de implementar en ensamblador. Es mucho más fácil para el programador desarrollar un programa usando un lenguaje de alto nivel.

Aun así, académicamente todavía se utiliza la programación en ensamblador. Este lenguaje se caracteriza por su eficiencia y bajo coste.

Por lo tanto, aprender ensamblador podría ser un buen paso para cualquier programador para aprender a gestionar los enormes recursos de los que se dispone actualmente. Es una buena forma de entender la programación de bajo nivel para poder entender después mejor y valorar la programación con lenguajes de alto nivel.

En futuro no es fácil saber cómo evolucionará el lenguaje ensamblador pero es posible que se siga utilizando en aplicaciones especializadas que requieran un alto rendimiento y control preciso del hardware.

TEMA 34

LENGUAJE C: CARACTERÍSTICAS GENERALES. ELEMENTOS DEL LENGUAJE. ESTRUCTURA DE UN PROGRAMA. FUNCIONES DE LIBRERÍA Y USUARIO. ENTORNO DE COMPILACIÓN. HERRAMIENTAS PARA LA ELABORACIÓN Y DEPURACIÓN DE PROGRAMAS EN LENGUAJE C

34.1 INTRODUCCIÓN 36
34.1.1 Contextualización 37
34.2 CARACTERÍSTICAS GENERALES 38
34.3 ELEMENTOS DEL LENGUAJE 39
34.3.1 Palabras reservadas 39
34.3.2 Tipos de datos 39
34.3.3 Variables 43
34.3.4 Constantes 44
34.3.5 Operadores 44
34.4 INSTRUCCIONES DE CONTROL 47
34.4.1 Instrucciones condicionales simples 47
34.4.2 Instrucciones condicionales compuestas 47
34.4.3 Bucles 48
34.5 ESTRUCTURA DE UN PROGRAMA 49
34.6 FUNCIONES 50
34.6.1 Definición de funciones 50
34.6.2 Paso de parámetros a una función 50
34.7 FUNCIONES DE LIBRERÍA Y DE USUARIO 51
34.7.1 Funciones de librería 51
34.7.2 Funciones de usuario 52
34.8 ENTORNO DE COMPILACIÓN 52
34.9 HERRAMIENTAS PARA LA ELABORACIÓN Y DEPURACIÓN DE PROGRAMAS EN LENGUAJE C 54
34.9.1 Herramientas para la elaboración de programas 54
34.9.2 Depuración de programas 54
34.10 CONCLUSIÓN 55
34.11 BIBLIOGRAFÍA