Software Base
Sesión 1: Traductores

Ing. Ramón Roque Hernández, M.C.

1

Lenguajes

- Permiten la expresión de ideas y razonamientos.
- Sin un lenguaje la comunicación sería imposible.
- Problema:
  - Computadora entiende 0110011100
  - Personas entendemos lenguaje natural.

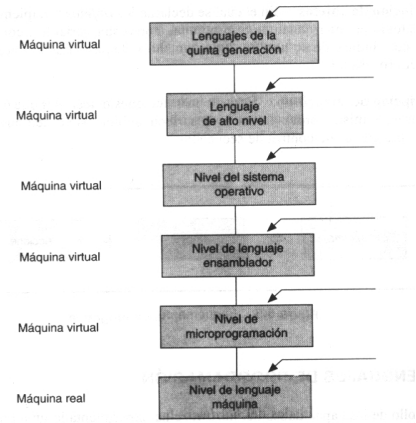
2

Lenguajes de programación

- Un lenguaje de programación es una notación para escribir programas, a través de los cuales es posible la comunicación con el hardware.
- Un lenguaje está definido por una gramática.
- Una gramática es un conjunto de reglas.

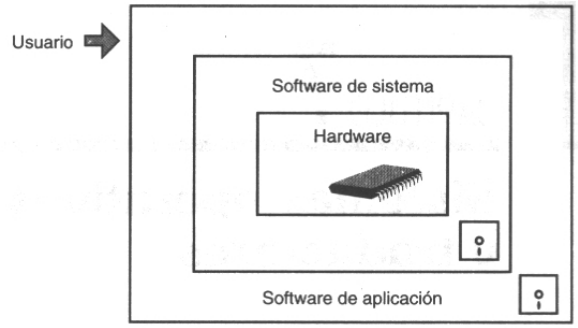
3

Niveles de acceso al hardware



4

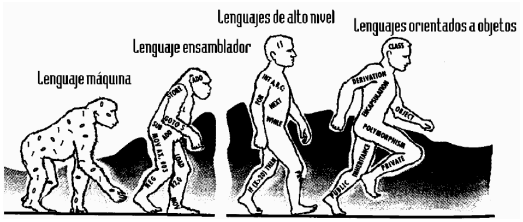
Niveles de acceso al hardware



5

Clasificación de los lenguajes de programación

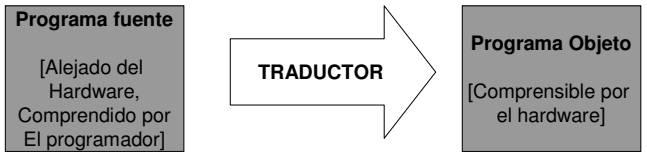
- Bajo Nivel (Lenguaje máquina)
- Nivel intermedio (Ensambladores)
- Alto Nivel (Evolucionados)



6

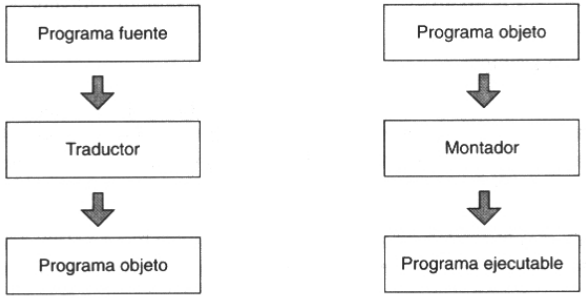
Traductores

- Son metaprogramas que toman como entrada un programa escrito en un lenguaje alejado de la máquina denominado **programa fuente** y proporcionan como salida otro programa equivalente escrito en un lenguaje comprensible por el hardware de la computadora denominado **programa objeto**.



7

Proceso de traducción



8

Traductores

- De Bajo Nivel
  - Ensambladores
  - Macroprocesadores
- De Alto Nivel
  - Intérpretes
  - Compiladores

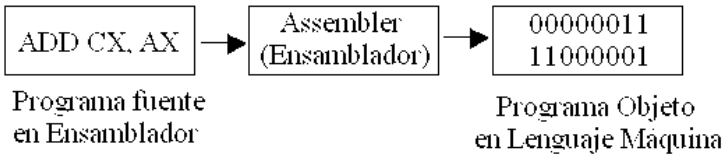
9

Ensambladores

- Primer intento de sustituir al lenguaje máquina por un lenguaje mas entendible por el programador.
- Usa etiquetas mnemotéctinas en lugar de códigos binarios.
- Programar en ensamblador sigue siendo tarea tediosa y propensa a errores.
- Se requieren muchas líneas de código.
- El ensamblador utiliza palabras para representar “códigos máquina”.

10

Función del ensamblador



11

Anatomía de una instrucción en Lenguaje Ensamblador

ETIQUETA	MNEMONICO	OPERANDOS	COMENTARIOS
Inicio:	**MOV**	**CX, AX**	;Pone el valor
			;de AX en CX

12

Instrucciones de un Lenguaje Ensamblador

- **Aritméticas**
  - ADD, SUB, MUL, IMUL, DIV, IDIV
- **Lógicas**
  - AND, OR, NOT, XOR
- **De transferencia de datos**
  - MOV, XCHG, PUSH, POP
- **De bifurcación o salto en el programa**
  - JMP
- **De Llamada y vuelta de subrutina**
  - CALL, RET

13

Macroprocesadores

- Son programas que permiten escribir, identificar y ejecutar macros.
- Las macroinstrucciones son consideradas como una extensión del lenguaje ensamblador básico.
- El macroprocesador se considera una extensión del algoritmo de ensamblador básico.
- Las macroinstrucciones (Macros) son abreviaturas de una sola línea para grupos de instrucciones.

14

Anatomía de la definición de una macro

Inicio de Defínición	MACRO
Nombre del Macro	[ ]
Cuerpo del Macro	……
	……
	…….
Fin de la Definición	MEND

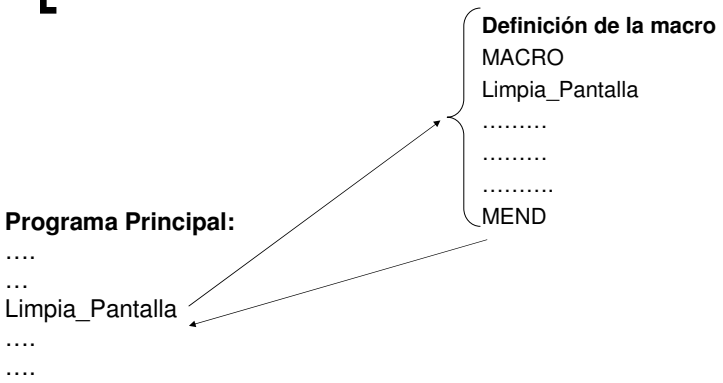
15

Cómo funciona una macro

- Una vez que el macro se ha definido, puede usarse el nombre del macro como una instrucción mnemónica del lenguaje ensamblador.
- Una macro puede contener argumentos o parámetros en las llamadas, de la misma manera que ocurre en los “Procedimientos” y “Funciones” de varios lenguajes de programación de alto nivel.
- Una macro también puede contener llamadas a otras macros.
- El macroprocesador sustituye la definición de todas las ocurrencias de la abreviación (Macrollamada) en el programa.

16

Cómo funciona una Macro



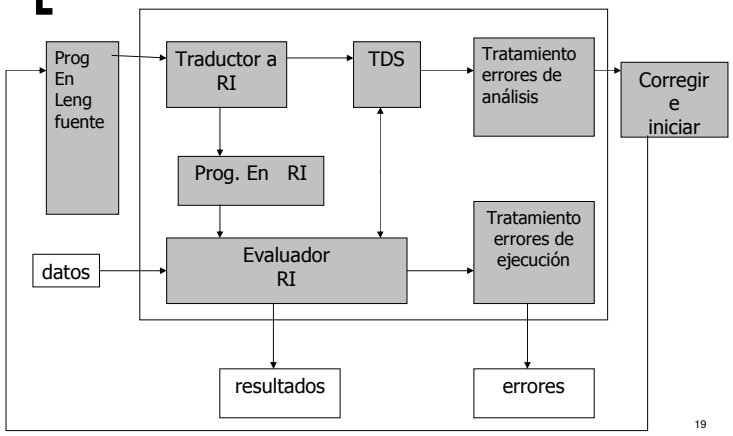
17

Intérpretes

- Un intérprete es un programa que analiza y ejecuta simultáneamente un programa escrito en un lenguaje fuente; no producen código objeto, siendo su ejecución simultánea a la del código fuente. [Traducción simultánea]

18

Anatomía de un intérprete



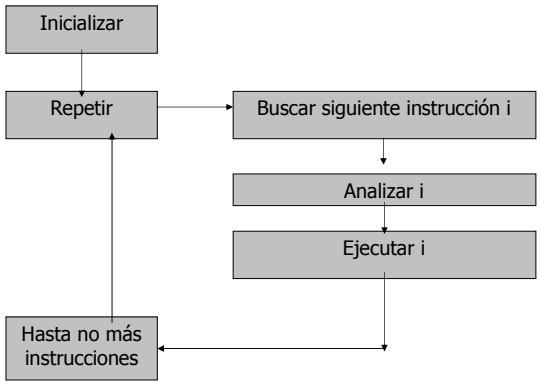
19

Anatomía de un intérprete

- ***Traductor a representación interna (RI)*** **:** Toma el programa fuente, lo analiza y lo transforma a la representación interna.
- ***Tabla de símbolos (TDS):*** Tabla donde se almacena información relativa a los símbolos que aparecen: etiquetas para las instrucciones, lo relativo a identificadores, etc.
- ***Evaluador de representación interna*****:** Partiendo de la representación interna y de los datos de entrada, se llevan a cabo las acciones indicadas para obtener los resultados.
- ***Tratamiento de errores*****:** Durante el proceso de evaluación pueden aparecer diversos errores como desbordamiento de la pila, divisiones por cero, etc.

20

Interpretación iterativa



21

Interpretación recursiva

- En un intérprete recursivo, las sentencias pueden estar compuestas de otras sentencias y la ejecución de una sentencia puede lanzar la ejecución de otras en forma recursiva.
- Éstos no son adecuados para aplicaciones prácticas a causa de su ineficiencia y se usan tan sólo como prototipo ejecutable del lenguaje.

22

Ventajas del uso de intérpretes

- Los intérpretes son más sencillos de implantar.
- Proporcionan mayor flexibilidad que permiten modificar y ampliar las características del lenguaje fuente.
- No es necesario contener en memoria todo el código fuente.
- Aumenta la portabilidad del lenguaje. Sólo es necesario que el intérprete funcione en otra máquina.
- Facilitan las tareas de depuración.

23

Aplicaciones de los sistemas basados en intérpretes

- ***Intérpretes de comandos*:** Usados por Sistemas Operativos (MS-DOS).
- ***Lenguajes de "Script":*** diseñados para que sirvan de enlaces entre diferentes sistemas o aplicaciones (Perl, JavaScript,etc.).
- ***Entornos de programación*:** Lenguajes que impiden su compilación o cuya compilación no es efectiva. (Lisp, Visual Basic,etc.).

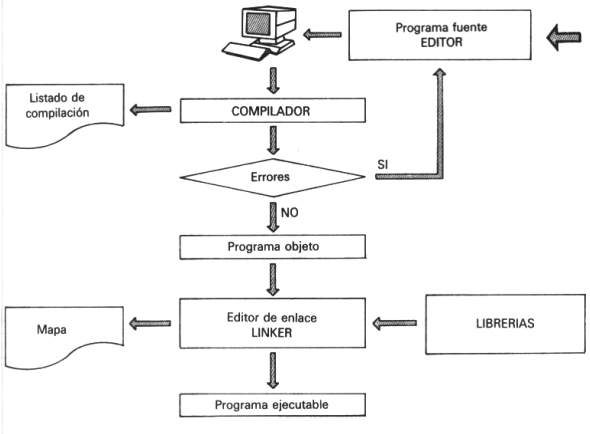
24

Aplicaciones de los sistemas basados en intérpretes

- ***Lenguajes de propósito específico*** **:** Consultas a bases de datos (SQL).
- ***Sistemas en tiempo real*** **:** Entornos que permiten modificar el código de una aplicación en tiempo de ejecución de manera interactiva.
- ***Intérprete de código intermedio*** : Máquina virtual de Java, .NET framework, etc.

25

Compiladores



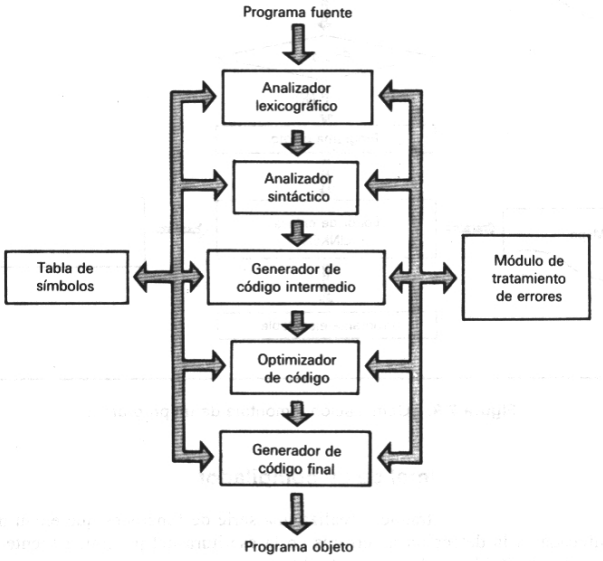
26

Estructura general de un compilador

- Analizador Lexicográfico (Scanner)
- Analizador Sintáctico (Parser)
- Analizador Semántico*
- Generador de Código intermedio
- Optimizador de Código
- Generador de Código Final
- Tabla de Símbolos
- Módulo de tratamiento de errores

27

COMPILADOR



Estructura general de un compilador

Analizador Lexicográfico (Scanner)

- Examina en el programa fuente las unidades básicas de información (Tokens).
- Elimina información superflua (Comentarios y espacios en blanco no significativos).
- Produce una "Tira de tokens" que indica cada elemento ya reconocido por el compilador; esta "tira de tokens" pasa al analizador sintáctico.

29

Ejemplo Analisis Léxico

DO WHILE (A <= 50)

El analizador léxico posee la descripción de cada elemento correcto posible que se puede encontrar en un programa:

PR1 → DO
PR2 → WHILE
CE1 → (
ID → A
OR → <=
CN → 50
CE2 →)
… ETC …

De esta manera, el analizador léxico produce la siguiente tira de tokens, identificando la instrucción como:

PR1 PR2 CE1 ID OR CN CE2

30

Analizador Sintáctico (Parser)

- Recibe la tira de tokens del analizador léxico, e investiga si son correctas:
  - El formato de las instrucciones
  - La duplicidad de identificadores
- Todos los errores se informan al programador indicando su localización y el tipo de error.
- Puede haber "warnings" (Advertencias).

31

Ejemplo Análisis Sintáctico

PR1 PR2 CE1 ID OR CN CE2

El analizador sintáctico posee las definiciones que indican los formatos correctos de la siguiente manera:

CORRECTO → ID OR CN
CORRECTO → CE1 CORRECTO CE2
CORRECTO → PR1 PR2 CORRECTO

Por lo tanto, el resultado del análisis sintáctico es:

CORRECTO.

32

Analizador Semántico

- Verifica que los tipos de datos de las variables involucradas sea el correcto.
- Es decir, que no se realicen operaciones no permitidas entre tipos de datos incompatibles.

33

Generador de Código Intermedio

- Traduce el resultado del análisis anterior (En caso de No-Errores) a un código intermedio propio del compilador para permitir la portabilidad del lenguaje (Posibilidad de utilización en distintas máquinas).

34

Ejemplo de Generación de Código Intermedio

PR1 PR2 CE1 ID OR CN CE2

El generador de código intermedio utiliza variables temporales para almacenar en "fragmentos" la instrucción de la siguiente manera:

T1 = ID OR CN
T2 = CE1 T1 CE2
T3 = PR1 PR2
T4 = T3 T2

35

Optimización de código

- Toma el código intermedio y lo optimiza, adaptándolo a las características del procesador al que va dirigido.
- Adapta el código a un procesador en particular.

36

Ejemplo de optimizador de código

T1 = ID OR CN
T2 = CE1 T1 CE2
T3 = PR1 PR2
T4 = T3 T2

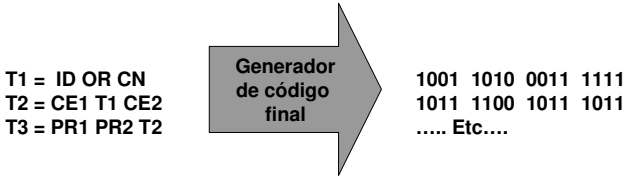
El optimizador de código lo convierte en:

T1 = ID OR CN
T2 = CE1 T1 CE2
T3 = PR1 PR2 T2

37

Generador de Código Final

- Traduce el código intermedio optimizado en el lenguaje máquina del procesador al que va dirigido.



38

Módulo de tratamiento de errores

- Facilita la detección y tal vez la recuperación de errores en las distintas fases de la compilación.
  - Errores Léxicos
  - Erroes sintácticos
  - Errores semánticos
  - Errores lógicos (Algoritmos mal implementados)
  - Errores de ejecución (Overflow,división por cero)

39

Tabla de símbolos

- Almacena todos los datos correspondientes a las variables y estructuras de datos del programa que se está compilando. Estos datos suelen ser: Tipo de cada variable, dimensiones, características, etc.

40

Nociones básicas de gramáticas

- Una Gramática:
  - Describe de forma natural la estructura jerárquica de muchas construcciones de los lenguajes de programación.
  - Es la definición formal de la sintaxis de un lenguaje de programación.

41

Una gramática se puede definir formalmente con:

- **Terminales.-** Simbolos básicos con que se forman cadenas. No se pueden dividir nuevamente. (Por ejemplo Palabras reservadas).
- **No terminales.-** Simbolos que pueden ser divididos nuevamente en Terminales y No-Terminales. (Generalmente son símbolos auxiliares).
- **Símbolo inicial.-** Es un no-terminal a partir del cual pueden obtenerse todas las sentencias del lenguaje aplicando las reglas de la gramática.
- **Reglas de producción.-** Son transformaciones de cadenas. Especifican como se pueden combinar terminales y no-terminales para formar cadenas.

42

Expresiones Regulares (ER)

- Son notaciones que permiten definir de manera precisa los patrones para obtener los componentes léxicos.
- Una ER se construye de ER mas pequeñas utilizando un conjunto de reglas que definen el lenguaje.
- Una ER utiliza los siguientes operadores:
  - **( )** .- Agrupar símbolos.
  - ***** .- Cero o mas casos
  - **+** .- Uno o mas casos
  - **?** .- Cero o un caso
  - | .- Alternativa “O”

43

Ejemplos de expresiones regulares

- Ejemplo 1: Realizar la expresión regular para un identificador (Nombre de variable) que inicie con letra y le sigan letras o digitos o nada.

  Letra (Letra | Digito) *

- Ejemplo 2: Expresión regular para una Constante entera sin signo

  Digito^{+}

44

Ejemplos de expresiones regulares

- Ejemplo 3: Expresión regular para una constante entera con signo requerido

$$(- \mid +)\ \text{Digito}^{+}$$

- Ejemplo 4: Expresión regular para operadores relacionales (<> ,<=, <, >, >=, =)

$$< (= \mid >)^{?} \mid > \ (=)^{?} \mid =$$

45

Ejemplos de expresiones regulares

- Ejemplo 3: Expresión regular para una constante entera con signo requerido

$$(- \mid +)\ \text{Digito}^{+}$$

- Ejemplo 4: Expresión regular para operadores relacionales (<> ,<=, <, >, >=, =)

$$< (= \mid >)^{?} \mid > \ (=)^{?} \mid =$$

46

Ejemplos de expresiones regulares

- Ejemplo 5: Expresión regular para la palabra reservada “PRINT”

$$(P)(R)(I)(N)(T)$$

47

Definición regular

- Una definición regular es una expresión regular a la cual se le asignó un nombre

Ejemplo 6: Definición regular para un identificador que inicie con letra y le sigan letras o digitos o nada.

$$L \rightarrow A \mid B \mid C \mid \ldots \mid Z$$

$$\#\rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9$$

$$ID \rightarrow L\ (\ L \mid \#\ )^{*}$$

48

Ejemplos…

Ejemplo 7: Definición regular para operadores aritméticos

$$OA \rightarrow = \mid + \mid - \mid * \mid / \mid \hat{}$$

49

Ejemplos…

Ejemplo 8: Escribir una definición regular para constantes numéricas con las siguientes reglas:

- Puede Tener signo o no.
- Puede tener decimales o no.
- Puede comenzar con punto.
- Si tiene decimales debe tener por lo menos 2 digitos después del punto.
- Puede tener exponente E. El digito del exponenete puede tener signo y debe ser entero.

$$\# \rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9$$
$$S \rightarrow + \mid -$$
$$D \rightarrow .\# \#^{+}$$
$$EX \rightarrow ES^{?}\#^{+}$$
$$CN \rightarrow S^{?} (D \mid \#^{+} D^{?}) EX^{?}$$

50