Contents

[UNIDAD 1. Conceptos Introductorios. 2](#_Toc207530180)

[Nivel de un lenguaje. 2](#_Toc207530181)

[Primer Lenguaje (Fortran) 2](#_Toc207530182)

[UNIDAD 2. Codigo Intermedio 2](#_Toc207530183)

[Código de 3 direcciones 3](#_Toc207530184)

[Procedimiento de lectura 4](#_Toc207530185)

[Procedimiento de Linea 4](#_Toc207530186)

[Procedimiento $Main 4](#_Toc207530187)

[Representacion del codido C3 6](#_Toc207530188)

[Tercetos 6](#_Toc207530189)

[Cuadruplas 6](#_Toc207530190)

[Quintuplas 6](#_Toc207530191)

[Tabla de simbolos 7](#_Toc207530192)

[Para las variables 7](#_Toc207530193)

[Para el proyecto. Del C3 convertir a cuadrupla 8](#_Toc207530194)

# UNIDAD 1. Conceptos Introductorios.

## Nivel de un lenguaje.

La CPU entiende instrucciones 1 y 0

Programa(RAM)

CPU

|  |
| --- |
| 011010 |
| 1010 |
| …. |

Los programadores escribían sus programas en binario. Hallerit invento las tarjetas perforadas.

Luego se creo Asembler ASM

Programa Fuente

011010

10101…

AX=10

BX=30

ASM

## Primer Lenguaje (Fortran)

FOR-TRAN = Formula Translation o Traductor de o formulas

Las versiones de posteriores de Fortran incluían *if-then-else-while-for*.

El fortran que transformaba, funciono bien después de 18 años en ser completadas.

*Arquitectura de un computador (“Humanamente perfecto”).*

El nivel de un lenguaje es la cercania al lenguaje binario.

* Python
* Prolog, Haskell
* Pascal, C++, C#
* C
* ASM
* Lenguaje Binario

# UNIDAD 2. Codigo Intermedio

**Código Intermedio**

ASM o Binario

Un código intermedio esta “escrito” en un lenguaje intermedio (o IL).

Un lenguaje IL es un lenguaje cercano al ASM, inventado por el diseñador del compilador y corre en una computador (maquina), imaginaria virtual.

* C
* **IL**
* ASM
* Lenguaje Binario

/\* JAVA

\*/

El IL de JAVA se llama byteCodes (RAW), es decir no esta comprimido, sin perdida de calidad.

JVM=Java Virtual Machine

Programa.jar

Instrucción 1 ⇒ JVM

Instrucción 2 ⇒ JVM

.

.

JAVA.exe – jar Programa.jar

.Net Framework

## Código de 3 direcciones

Abreviado en español: C3

Abreviado en ingles: 3AC (Tree Address Code).

Se puede decir que todos los IL actuales usan un C3. “Un código C3, dice que toda instrucción a lo sumo 3 direcciones de memoria”.

¿Qué entendemos por direcciones de memoria?

* Variables (normales y temporales)
* Procedimientos
* Etiquetas

Ejemplos.

X=t1+y //correcto

Z=x+t1-z //Error, hay 4 direcciones de memoria

T1=x\*z-p //Error, hay 4 direcciones de memoria

Convertir a C3

Int x,y,z,m,q;

-

-

Y=x\*y - z/m + q\*z;

**Solucion:**

Usamos variables temporales.

t1= x\*y //t1 es variable temporal

t2=z/m //t2 es temporal

t3=t1-t2 //t3 es temporal

t4=q\*z // t4 es temporal

y=t3+t4

k1

n=5

E1:

t1=(k<=n)

if (i=0) ⇒Goto E2

write(k)

inc k //k++

Goto E1

E2:

(.) Converir a C3 el siguiente código

If x>0

Then

Begin

Z=0;

Write(“El valor de z es”, z);

End.

**Solucion:**

t1=0

t2=(x > t1)

If (t2=0) ⇒ Goto E1

z=0

write(“El valor de z es”)

write(z)

E1: //fin

(.) Hacer un programa C3 que:

* Lea N (N>0)
* Produzca un triangulo formado por N líneas

e.g N=4

\*

\*\* N=4

\*\*\*

\*\*\*\*

Solucion:

Haremos un procedimiento leer N que lea (read) N y valide que N>0

### Procedimiento de lectura

E1:

writeS(“Introduzca N”)

read(N);

t1=0

t2=(N<=t1)

if (t2=1) ⇒ Goto E1

RET //return

Haremos un procedimiento

Linea k //k es var global

Que imprima en la consola k asteriscos

K=3 print

Call línea ⇒ \*\*\*

### Procedimiento de Linea

i=1

While(i<=k){

Print(“\*”);

I++;

}

For i=1 to k do

Begin

Print(“\*”);

End.

i=1

E1:

T1=(i <= k)

IF (T1=0) ⇒ Goto E2

writeS(“\*”)

inc i //i++

Goto E1

E2: RET

### Procedimiento $Main

N=4

For i=1 to N do

Begin

Linea();

NL();

End

Call leer N //leer N();

K=1

E3:

T1=(K <= N)

If (T1=0) ⇒ Goto E4

Call linea

NL

Inc K

Goto E3

E5: RET

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

If (a<b){ //ExprBoole

Print(a+”es menor que”+b) //sentencias

}

If (ExprBoole){

Sentencias;

}

Respuesta:

C3- ExprBoole(ti) //ti: Cualquier nro de etiqueta

If(ti=0) ⇒Goto Ek //Ek: Finalizando la etqueta

C3-Sentencias //C3: Convertir a C3

EK:

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(.) Convertir A C3

If a<b || p!=5 {

P=s+a;

Print(p);

}

**Solucion:**

t1= (a<b)

t2=(p!=5) C3-ExprBoole(t3)

t3=(t1 or t2)

if(t3=0) ⇒Goto E1

p=s+a C3-Sentencias

write(p)

E1:

(.) Convertir a C3 la expresión aritmética

Z=[3+(x/y)\*t] mod 5; //Expresión aritmética

**Solucion:**

C3-Expr(ti)

Z=ti

**Ei:**

(.) Escriba un esquema de traducción para

While (ExprBoole) {

ExprBoole

Sentencias;

}

**Solucion**:

**true**

**false**

Ei:

C3-ExprBoole(tk)

Sentencias

If (tk=0) ⇒Goto Ef //Ef: salir del bucle

C3-Sentencias

Goto Ei:

**EF:**

Ef:

(.) Convertir a C3

While (x<y) && z≠0 {

x=z+3

y++;

}

**Solucion:**

E1:

t1=(x<y)

t2=0

t3=(z≠t2)

t4=t1 AND t3

if (t4=0) Goto E2

t5=3

x=z+t5

inc y

Goto E1

E2:

(.) Convertir a C3

While(x<y) {

X++;

While(z<=p) {

Print(“\*”)

}

}

**Solucion:**

*E1:*

T1=(x<y)

If (t1=0) Goto⇒E2

Inc x

*E3:*

T2=(z<=p)

If(t2=0) Goto⇒*E4*

writeS(“\*”)

*Goto⇒E3*

*E4:*

*Goto ⇒E1*

*E2:*

**Ei:**

**false**

**true**

ExprBoole

Sentencias

(.) Convertir a esquema de traduccion

REPEAT

Sentencias;

UNTIL ExprBoole;

**Solucion:**

Ei:

C3-Sentencias;

C3-ExprBoole(tk)

If (tk=0) Goto ⇒ Ei

(.)

if (ExprBoole) {

ExprBoole

Sentencias;

}

**Solucion:**

**true**

C3-ExprBoole

Sentencias

If (tk=0) ⇒ Goto Ef

**false**

C3-Sentencias

**Ef:**

Ef:

(.) Realizar un esquema de traduccion

If (ExprBoole) {

Sentencias;

}

Else{

Sentencias;

}

**Solucion:**

C3-ExprBoole(tk)

If (tk=0) Goto⇒Ea

C3-Sentencias1

Goto⇒Ef

Ea: //else

C3-Sentencias2

Ef:

## Representacion del codido C3

El c3 no es un texto (archivo o Plain Text). Dependiendo de cuantos campos utilicemos, para interpretar una instrucción c3, la representación recibe un nombre:

Tercetos: La representación usa 3 campos.

* Ocupa menos memoria
* Algoritmos mas complicados

Ejemplo:

Class Terceto {

Int a;

Int b;

Float c;

}

Cuadruplas: La representación usa 4 campos.

* Ocupa un poco de memoria (desperdicia memoria)
* Algoritmos simples

Quintuplas: La representación usa 5 campos.

* Ocupa mucha memoria (desperdicia demasiada memoria)
* Algoritmos muy sencillos

**Escogemos la cuadrupla**

Una cuadrupla tiene estos campos

Class cuadrupla { Graficamente

Int opcode; v [opcode | Dir1 | Dir2 | Dir3]

Int Dir1, Dir2, Dir3;

}

Opcode = operation code

= es un numero que identifica en forma única a la operación a efectivas

/\* Todas las representaciones usan este paso \*/

Por ejemplo:

Suma = 0 Resta=1 Por=2

X=y+z [ **0** | x | y | z ] t1=z-t2 [ **1** | t1 | z | t2 ]

**Suma Resta**

Opcode Dir1 Dir2 Dir3 Opcode Dir1 Dir2 Dir3

Para trabajar el opcode, el programador usa constantes.

En delphi: En Java

CONST public static final int suma = 0;

SUMA = 0; public static final int resta = 1;

RESTA = 1; public static final int por = 2;

POR = 2

**Para las asignaciones simples**

* Var=valor

Ejemplo:

Z=50 [ Asignacion constante | z | 50 | \_ ] //50 es constante y \_ es un valor que no importa(I don’t care)

* Var1=var2

Elemplo:

x=y [ Asignacion ID | x | y | \_ ]

* Va1 = -var

Elemplo:

x=-y [ MINUS | x | y | \_ ] y=-y [ MINUS | y | y | \_ ]

#### Para las aritméticas y logicas

Var1=var2+var3

.

.

* Var1=var2 AND var3

Ejemplo: x=t1 AND z [ AND | x | t1 | z ]

* Var1=var2 OR var3

Ejemplo: x=t1 OR z [ OR | x | t1 | z ]

* Var1=var2 NOT var3

Ejemplo: x= NOT t1 [ NOT | x | t1 | \_ ]

/\*Las constantes de opcode para estos operadores usan 3 letras\*/

MAY // > MEN // < DIS // ≠

MAI // ≥ MEI // ≤ IGU // =

Para cada uno de estos saltos, creamos un código de operacion

If (x=0) ⇒ Goto E4 If (y=1) ⇒ Goto E9

[ if 0 | x | 4 | \_ ] [ if 1 | y | 9 | \_ ]

Entonces un programa c3 es es una colección (vector, lista) de cuadruplas.

Programa C3

$Main C3 [ ]

0 X = 0 0 [ ASIGN CTTE | x | 0 | \_ ]

1 writeS(“Bye”) 1 [ writeS | “Bye” | \_ | \_ ]

2 inc x 2 [ inc | x | \_ | \_ ]

3 Ret 3 [ Ret | \_ | \_ | \_ ]

# Tabla de simbolos

Este IL c3 usa 2 tablas

1. TSS = Tabla de string constantes

Images

[ ] img01.jpg

[ ] img02.jpg

[ ] img03.jpg

Dada esas 3 imágenes, validamos su tamaño, su hash, para ver si no están repetidas y no volver a descargar, además cuando quiera de descargar de nuevo, solo se descargara su acceso directo, ya que solamente una vez se descarga para optimizar memoria.

El IL c3 que manejamos no utiliza símbolos, solo números. Es estos IL’s, se tiene la siguiente regla aplicada solo a las cuadruplas.

“Toda referencia a la tabla de símbolos es negativa (ósea mayor que 0)”

**Por ejemplo:**

TSS

0 [ “Bye” ] writeS(“Hola”) [ WRITES | -1 | \_ | \_ ]

1 | “Hola”| writeS(“N”) [ WRITES | -2 | \_ | \_ ]

2 [ “N” ] writeS(“Bye”) [ WRITES | -0 | \_ | \_ ]

## Para las variables

Algunas personas utilizan 2 tablas: Una tabla para las variables y otra para los procedimientos. Pero es posible utilizar una sola tabla.

/\*

ID = Identifier

= Nombre de var, class, función que el programador se inventa

Class Pila { //Pila es el ID

Int a, b; //a, b son los ID

Void sumar () { //sumar es el ID

}

}

\*/

Esta única tabla la llamamos TSID

0 | Pila | | | En esta tabla almacenamos las variables y los procedimientos

1 | a | | |

2 | b | | |

3 | suma | | |

Para los temporales usamos estas reglas

// Los temporales se diferencian en forma positiva

t4 = t5+t1 [ suma | 4 | 5 | 1 ]

Considere la TSID

Nombre .

0 | Base | | | //var

1 | x | | | //var

2 | leer | | | //proc

3 | $ main | | | //proc

Call leer // [ CALL | -2 | | ]

X=t1+x // [ SUMA | -1 | 1 | -1 ]

Base=x-t4 // [ RESTA | 0 | -1| 4 ]

# Para el proyecto. Del C3 convertir a cuadrupla