CROSS - ASSEMBLERS Y SIMULADORES EN EL DESARROLLO DE SISTEMAS BASADOS EN MICROPROCESADORES. ESTUDIO, IMPLEMENTACION Y EJEMPLOS DE APLICACION PARA EL Z80/Z80A.

TORRES PROAÑO, EDGAR P. ING.
INSTITUTO ECUATORIANO DE TELECOMUNICACIONES
ESCUELA POLITECNICA NACIONAL

RESUMEN

En este trabajo se abordan temas relacionados con el diseño y aplicación de traductores y simuladores. Estos están formados por un grupo de programas de computador escritos en algún lenguaje, usualmente Assembler o Fortran, y que son ejecutados en un procesador adecuado con el propósito de facilitar, organizar y optimizar la labor de desarrollo de Sistemas basados en microprocesadores. La cantidad de material involucrado hace imposible que se lo pueda exponer con algún detalle en un artículo de una extensión como la del presente. Por lo tanto el enfoque será mas bien informativo mencionando los conceptos básicos relacionados con el tema, y haciendo más énfasis en la aplicación de este tipo de herramientas.

INTRODUCCION

El tema aquí tratado nace de una inquietud del ser humano, de utilizar el computador y sus recursos adicionales de una manera más eficiente. Sin este tipo de herramientas, la labor de programación y desarrollo de Sistemas, en general, sería más tediosa y consumiría tiempo y recursos muy significativamente.

El presente trabajo se orienta fundamentalmente al diseño, implementación y aplicaciones de un cross-assembler y simulador para el popular microprocesador Z80/Z80A.

Como se verá posteriormente en los ejemplos presentados; y gracias a la forma como se orienta la solución del problema; se ha conseguido también, tener incorporado un desemsamblador que puede tomar código de máquina puro y transformarlo en un programa en assembler más comprensible para el usuario.

Es interesante mencionar que existen en el mercado cross-assemblers y simuladores suministrados por diferentes firmas y orientados a distintos microprocesadores como el 6800, el Z80, etc. Pero que para su adquisición hay que enfrentar las siguientes desventajas: no hay disponibles para todotipo de computadores, sino sólo para los más conocidos; su costo es relativamente significativo; y por lo general se venden o suministran en módulos tipo objeto, es decir que no se dispone del programa fuente que permitiría realizar modificaciones, lo que en muchos aspectos sería muy provechoso, pues toda pieza de software no es completamente perfecta y a través del tiempo requiere actualizaciones y mejoras que si tienen que ser hechas por el suministrador sufren retardos y necesitan, usualmente un pago adicional de dinero. Entonces es claro, que es extraordinariamente útil, disponer de los programas fuente de estos sistemas.

Hay varias formas de atacar el problema planteado; una de ellas es una forma intuitiva que llevaría a la consecución de traductores y simuladores no muy poderosos, ni versátiles y por otro lado ineficientes, enredados y difíciles de usar y mantener; la otra forma, que es la que se ha usado en el desarrollo del presente trabajo, se refiere a un planteamiento adecuado, organizado y científico, que haciendo uso de la teoría de compiladores y simuladores, permite obtener traductores y simuladores, eficientes , útiles, versátiles, fáciles de manejar y actualizar.

El material se ha organizado de la siguiente manera: se comienza mencionando los conceptos básicos referentes al problema de diseñar e implementar Traductores, inmediatamente se hace una particularización para el caso del traductor Cross-assembler para el microprocesador Z80/Z80A, se presenta un diagrama de bloques general que indica en forma esquemática como está configurado el Traductor y se mencionan aspectos de interés. A continuación se presentan los aspectos y conceptos fundamentales relacionados con el desarrollo de Simuladores, también aquí se incluye el respectivo, diagrama de bloques que refleja la forma como fue abordado el problema; seguidamente se muestra mediante algunos reportes obtenidos del computador, varias características y facilidades que ofrecen cada una de estas herramientas, se insertan intencionalmente algunos errores, con el fin de demostrar la forma, como estos Sistemas, mediante mensajes de error dan indicaciones precisas que permiten al usuario corregir las equivocaciones cometidas; sean en el manejo del lenguaje assembler; en la lógica de los programas; a través de interpretación de resultados; o en el manejo en sí de éstas herramientas; a continuación se presenta un diagrama que permite visualizar claramente la manera como se usan el Cross-assembler y el Simulador en el diseño y desarrollo de Sistemas , se ilustra esto con algunos ejemplos, que sin ser demasiado extensos, permiten observar como se usan estas herramientas en forma combinada y global, al mismo tiempo que, se mencionan algunos de los aspectos de más interés.

Los mensajes, leyendas , comentarios e incluso las referencias hechas en la programación misma de estos Sistemas fueron realizadas usando el idioma Inglés, entre otras razones porque, los códigos nmónicos del Set de instrucciones original de la Z80/Z80A, usados para el Cross-assembler y Simulador tienen sus raíces en el Inglés, y se ha querido

mantener la consistencia a ese nivel. Sin embargo, si se desea, se pueden realizar los cambios necesarios en los formatos, para obtener una versión equivalente en Español. Para mostrar esta posibilidad, a los sistemas desarrollados se les ha dotado de la opción de desplegar sus titulares principales tanto en Español como en Inglés.

Finalmente, cabe mencionar que el trabajo se encuentra funcionando en el computador que el Instituto Ecuatoriano de Telecomunicaciones dispone en el Departamento de Procesamiento Automático de Datos de la Región 1.

1.- EL TRADUCTOR CROSS-ASSEMBLER

1.1.- LENGUAJES DE COMPUTADORES

Existen varios niveles de lenguajes de computadores que varían de acuerdo a la distancia existente entre las reglas y posibilidades gramaticales de cada uno y la forma de trabajo interna de la máquina.

Podemos mencionar por ejemplo:

- Lenguaje de máquina que depende de la arquitectura y del set de instrucciones específicos de un computador; es interpretado por el hardware o por microprogramas propios de la máquina.

- Assemblers son lenguajes de bajo nivel que dependen del Set de Instrucciones. Su programación es menos complicada que en lenguaje de máquina y permite un buen control sobre los recursos del procesador; la traducción a lenguaje de máquina que pueda ser entendido por el computador se la realiza mediante un ensamblador o es traducido en forma manual.

- Tarjetas de control y lenguajes de comandos son usados para comunicarse con el Sistema Operativo.

- Lenguajes de alto nivel son de más fácil programación y menos dependientes de la arquitectura o del Sistema Operativo del computador, pero sus programas son más lentos y requieren mayor memoria. Para que puedan ser ejecutados deben ser modificados por un procesador del lenguaje respectivo (Fortran, PL/I, Cobol).

1.2.- PROCESADORES DE LENGUAJE

En general un procesador de lenguaje está formado por un grupo de programas que permiten traducir los lenguajes de computadores a un código que pueda ser comprendido por la máquina.

Existen dos tipos de procesadores de lenguajes: intérpretes y traductores. Los primeros aceptan como entrada, un programa escrito en un lenguaje de computador (fuente), lo traduce e inmediatamente lo ejecuta. Los traductores aceptan como entrada un programa fuente y producen un programa en lenguaje objeto que usualmente es lenguaje de máquina de algún computador y por lo tanto puede ya ser ejecutado por el mismo.

1.3.- TRADUCTORES

Como se dijo anteriormente un traductor convierte un programa escrito en algún lenguaje de computador en una secuencia de instrucciones de máquina que producen las acciones deseadas por el programador y que han sido descritas en el programa de entrada. Para facilitar su comprensión, este proceso debe ser considerado como una interconección de tareas específicas más sencillamente definidas. Así se puede obtener el modelo esquemático de traductor como se muestra en la Figura 1.

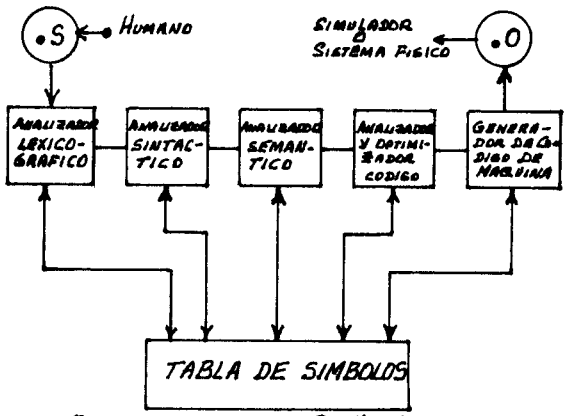

FIG.1 DIAGRAMA BASICO DE UN TRADUCTOR.

1.3.1.-CAJA DE ANALISIS LEXICOGRAFICO

En ella se produce un agrupamiento de los caracteres de entrada, para obtener entidades llamadas tokens que tienen definidas su naturaleza y valor. Por ejemplo se pueden reunir caracteres y determinar que corresponden a una variable, una constante o un operador, que tienen además un cierto valor que puede ser un puntero a una tabla de símbolos.

1.3.2.-CAJA DE ANALISIS SINTACTICO

En ella se cambia la secuencia de tokens de modo que refleje el orden de las operaciones a ser realizadas y determina si la estructura es sintácticamente correcta de acuerdo a las reglas del assembler planteado. Aquí se generan entidades básicas llamadas átomos que también tienen clase y valor.

1.3.3.-CAJA DE ANALISIS SEMANTICO

Generalmente está incluída en otras fases del traductor, aunque también puede ir entre el análisis sintáctico y la generación de código. En esta caja es de principal interés lo referente a significado de la estructura. Por ejemplo si es suma, carga, desplazamiento, etc.

1.3.4.-CAJA DE OPTIMIZACION

En ella se reordenan y cambian las operaciones en el programa que se está traduciendo a fin de hacerlo más eficiente. Para cada programa fuente hay muchos programas objeto que realizan un trabajo equivalente, unos programa objeto ocupan un menor tamaño y son más rápidos. Con el proceso de optimización se intenta producir este código objeto más eficiente.

1.3.5.-GENERACION DE CODIGO

En ella se convierten los átomos en una secuencia de instrucciones que realizan el mismo trabajo y son entendidas por el microprocesador.

1.3.6.-TABLA DE SIMBOLOS

En ellas se almacenan todos los atributos de los identificadores. Toda esta información se ingresa en los análisis sintáctico, lexicográfico y semántico; luego es utilizada en las decisiones semánticas, la optimización, la generación de código, la detección y corrección de errores.

1.3.7.-PASOS DE UN TRADUCTOR

Los procesos señalados en el modelo de traductor se pueden presentar y ejecutar en el orden mostrado o en forma paralela, entrelazada. En un traductor de un paso el control va y viene entre las cajas de acuerdo a como tokens y átomos se requieren y producen. En un traductor de dos pasos se puede tener, por ejemplo, que cada vez que un token es producido por la caja de análisis lexicográfico, el control pasa a la caja de análisis sintáctico que procesa ese token. Cuando se requiere el siguiente token el control retorna al analizador lexicografico, etc.

1.3.8.-PASADAS DE UN TRADUCTOR

La mayoría de traductores son de dos pasadas. Durante la primera pasada se crean las tablas de símbolos y se recogen todas las definiciones; en la segunda pasada se traduce el programa con la ayuda de la información recolectada en la primera pasada.

1.3.9.-FORMAS DE ENSAMBLAJE

Como una alternativa a la tediosa, cansada e ineficiente tarea de ensamblar a mano un programa aparece la posibilidad del cross-ensamblaje. En éste la generación del código objeto se realiza en un procesador y la ejecución en otro. Esto es muy útil cuando se desarrolla Sistemas con microprocesadores, pues éstos usualmente carecen de la velocidad, memoria, periféricos y software requerido para un ensamblaje conveniente. El desarrollo de programas se realiza con mayor facilidad en un computador más grande que cuenta con periféricos de entrada y salida de alta velocidad, mayor capacidad de memoria, sistemas operativos, editores y demás software útil. Una vez depurado el programa en assembler usando el cross-ensamblador se puede probar el programa objeto en un simulador y por último en el sistema físico. El desarrollo de software para microprocesadores resulta más sencillo y eficiente si se usa un computador grande.

1.4.- EL TRADUCTOR CROSS-ASSEMBLER PARA EL Z80/Z80A

Antes de proceder al diseño e implementación del traductor es necesario definir la gramática y los alcances del mismo. Para el presente trabajo se ha seguido básicamente la gramática considerada como stan-

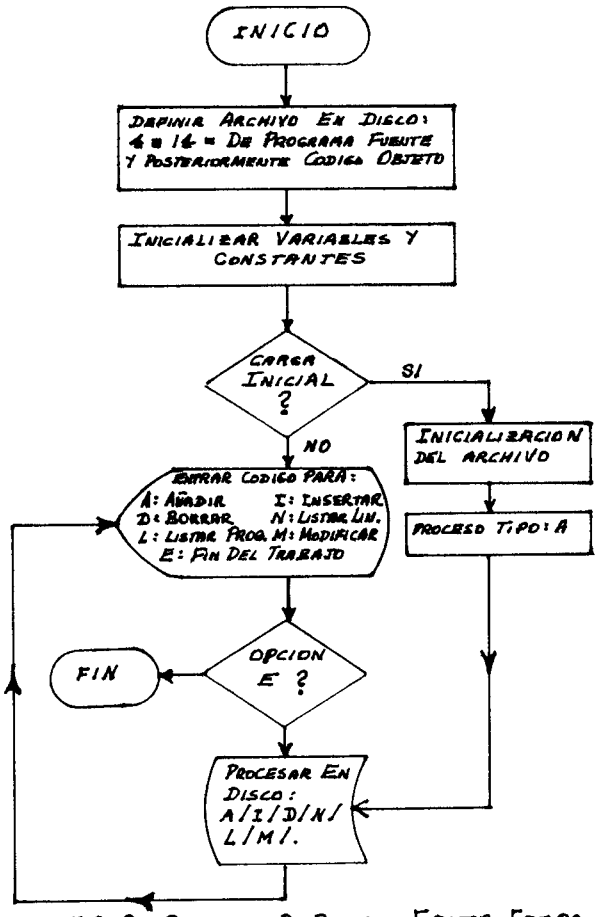

FIG. 2 DIAGRAMA DE BLOQUES EDITOR EDZ80

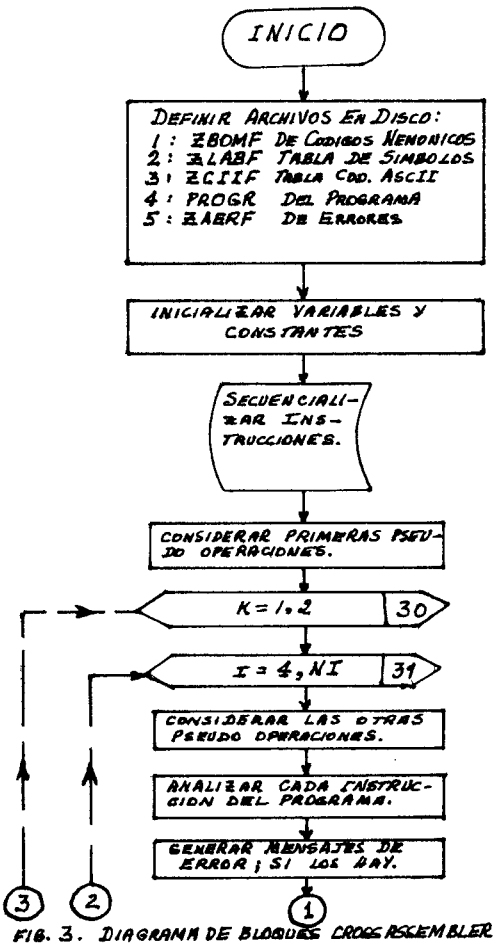

FIG. 3. DIAGRAMA DE BLOQUES CROSS ASSEMBLER AS Z80

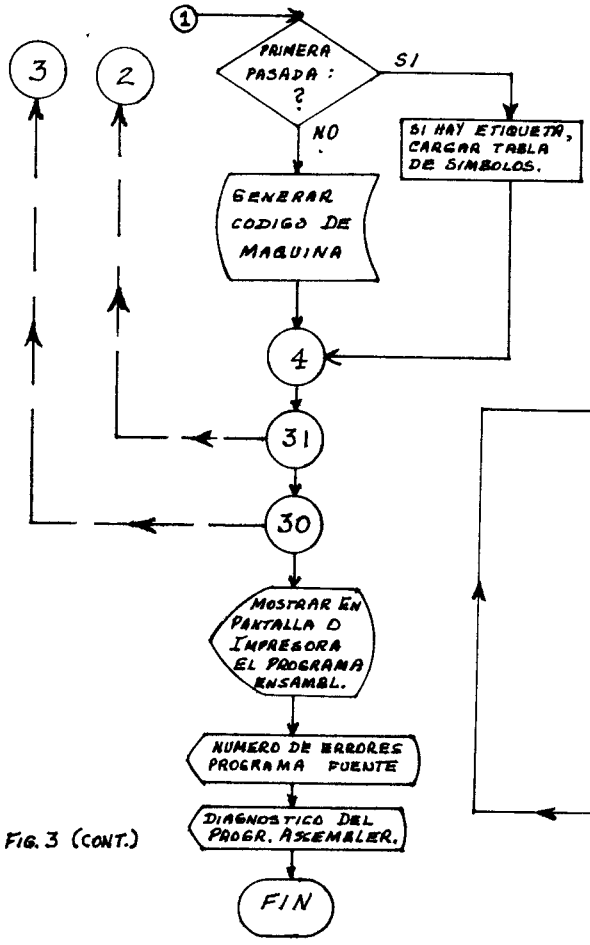

FIG. 3 (CONT.)

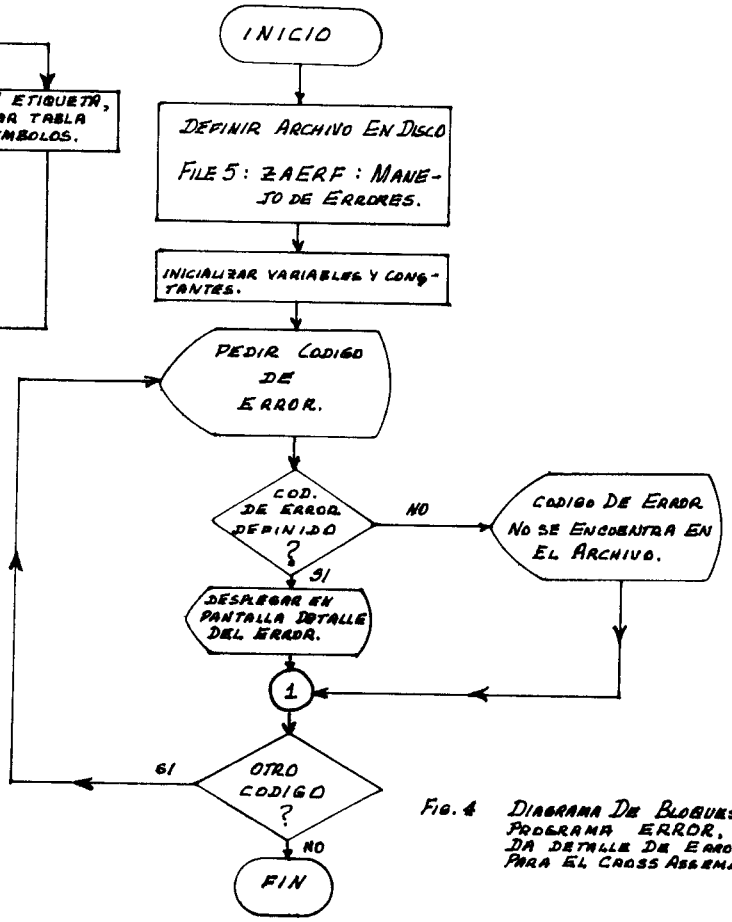

FIG. 4 DIAGRAMA DE BLOQUES DEL PROGRAMA ERROR, QUE DA DETALLE DE ERRORES PARA EL CROSS ASSEMBLER.

- dard para el Z80. Esto se ha hecho con la finalidad de que programas implementados por usuarios de otros sistemas sean compatibles a nivel de assembler con los que sean desarrollados usando el traductor producido en el presente trabajo, y viceversa.

Las funciones de las diferentes cajas que componen el traductor a menudo se combinan para de esta manera simplificar un tanto un problema particular. Un aspecto clave en el traductor desarrollado representa la tabla de códigos nmónicos, que ha sido extendida y particularizada más allá de lo que se tiene disponible en la literatura de la Z80, es así como con ayuda del computador se ha generado una tabla de seis cientos noventa y nueve entradas a partir de una de ciento sesenta nueve.

Se maneja además una tabla de símbolos para las etiquetas, una tabla de códigos ASCII, una tabla de códigos de errores comunes con su respectiva explicación. Es interesante que tanto el traductor como el simulador aprovechan en forma eficiente la capacidad de funcionamiento en modo conversacional de que dispone el computador utilizado. Básicamente en la tarea de ensamblaje se usan tres grupos de programas. El primero es un editor (EDZ80), cuyo diagrama se muestra en la Figura 2, este se encarga de dar las facilidades para la generación del programa fuente escrito en assembler. El segundo es el traductor en sí (ASZ80), cuyo diagrama de bloques se indica en la Figura 3, éste es el que toma el programa fuente, lo ensambla y el código objeto generado, con sus direcciones de memoria respectivas, se coloca a la izquierda del programa fuente dando como resultado un reporte muy claro, organizado y fácil de utilizar. Por último se dispone de un programa (ERROR), cuyo diagrama de bloques se presenta en la Figura 4, que se encarga de dar la interpretación adecuada al código de error que en el caso de que exista se coloca a la extrema-izquierda del programa fuente, en la respectiva línea.

2.- EL SIMULADOR

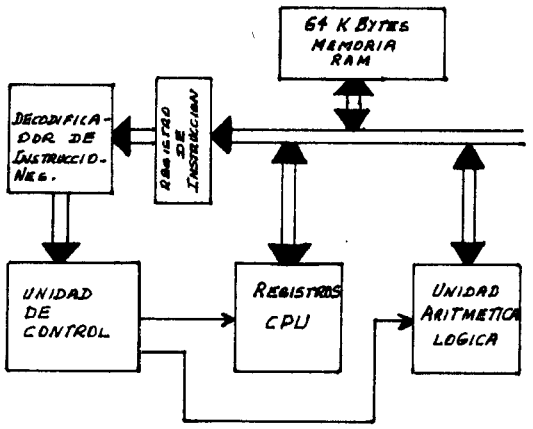

FIG. 5 PARTES DEL SISTEMA MICROPROCESADOR ASIM SIMULADAS.

2.1.- CONCEPTOS DE UN SISTEMA

A un sistema se lo tiende a considerar como un agregado de objetos que tienen alguna interdependencia o relación regular entre ellos.

En un sistema podemos encontrar básicamente los siguientes elementos :

a. Entidad.- Es un objeto de interés del sistema a considerarse.

b. Atributo.- Es una propiedad que posee la entidad.

c. Actividad.- Es un proceso que provoca cambios en el sistema.

d. Estado del Sistema.- Describe las entidades, atributos y actividades de acuerdo a su existencia en algún punto del tiempo.

e. Medio Ambiente del Sistema.- Es un conjunto de factores externos del sistema. Un paso importante en el modelado de un sistema consiste en establecer el límite entre el sistema y su medio ambiente.

2.2.- EL MODELADO DEL SISTEMA

Se trata aquí de plantear o definir un modelo que se aproxime al sistema real en los aspectos que nos interesa estudiar. Aquí es interesante anotar que no hay un modelo único para un sistema y por lo tanto definir implica tomar en consideración aquellos aspectos de interés. Esto definirá la estructura del modelo y el tipo de datos sobre los cuales vamos a trabajar.

2.3.- TIPOS DE MODELO

Los modelos de sistemas se pueden clasificar de diversas formas. En algunas ocasiones se toma en cuenta su naturaleza y se los clasifica en continuos o discretos; determinísticos o estocásticos. También se pueden clasificar como físicos y matemáticos ; dentro de los matemáticos podemos tener estáticos y dinámicos, dentro de los dinámicos tendremos analíticos y numéricos, y por último como una aplicación directa a un modelo de tipo numérico tenemos la Simulación de Sistemas.

2.4.- SIMULADORES

Son una herramienta para depuración y desa

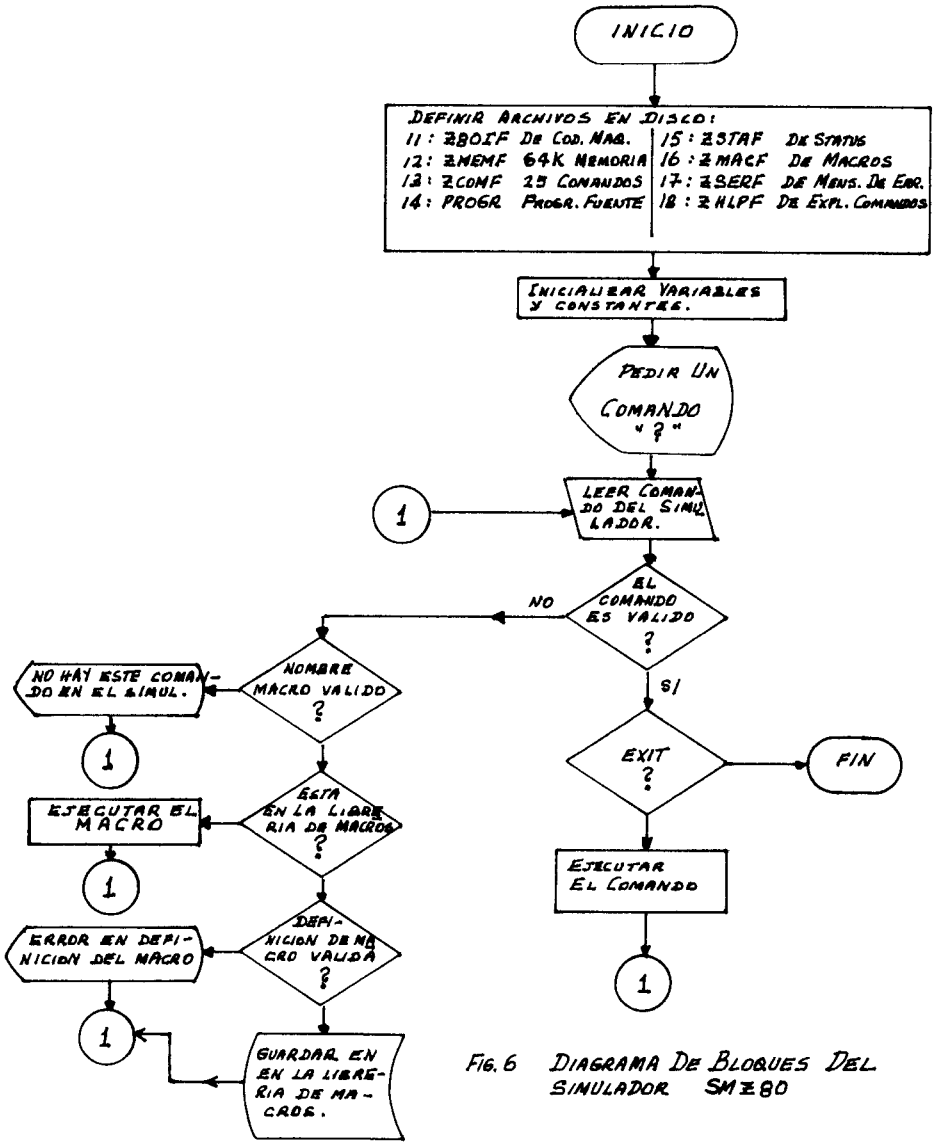

Fig. 6 Diagrama De Bloques Del Simulador SMZ80

rrollo de software. Representan el funcionamiento de un sistema mediante programas de computadora que se ejecutan en otro procesador. El simulador acepta los mismos datos, ejecuta los programas y logra los mismos resultados que el sistema imitado, hace exactamente lo que el programador haría con papel y lápiz para rastrear los efectos de las instrucciones con la ventaja de que es más rápido, completo y no comete errores. Generalmente los simuladores corren en computadores y simulan el trabajo de computadores más pequeños y microprocesadores. Usando las facilidades que brinda un computador más grande los simuladores permiten más flexibilidad en el proceso de desarrollo y depuración de programas y simplifican tareas como chequeo en la lógica de un programa, cambio de datos, prueba de registros, etc. consiguiéndose cualquier grado de complejidad en la interpretación de instrucciones de máquina, mientras que en el desarrollo con un sistema real existen limitaciones de hardware en cuanto a memoria, el número de registros break-point para rastreo de instrucciones y la configuración de los pines de la C.P.U.

Además de la flexibilidad que brinda un simulador, debe considerarse que muchas veces su utilización es esencial, por ejemplo si el sistema real es necesitado para otros propósitos, si no está trabajando o si no ha sido implementado aún.

Entre las desventajas que pueden existir en el uso de simuladores, están : que no pueden simular en forma muy cercana las funciones de entrada/salida; que la simulación comparada con el funcionamiento del sistema real, es más lenta; que no puede simular interfases, entre otras.

2.5.- EL SIMULADOR PARA EL Z80/Z80A

En la figura 5, podemos observar como se ha planteado la simulación de un sistema microprocesador en términos de sus elementos de interés para el presente trabajo. Así tenemos: la memoria RAM con capacidad para 64K Bytes; los registros de la C.P.U. que permiten una simulación de doscientos ocho bits, configurados en dieciocho registros de ocho bits y cuatro registros de dieciseis bits, alternativamente se pueden agrupar en dos sets de seis registros de propósito general que pueden ser usados individualmente como registros de ocho bits o como registros pares de dieciseis bits cada uno, además se dispone de dos juegos de acumuladores y registros de banderas; también se considera el registro de instrucciones; el decodificador de instrucciones ; la unidad de control y la unidad aritmética y lógica. Sumado a todas estas consideraciones se añaden otras opciones o facilidades del simulador tales como un juego de veinte y cinco comandos que desempeñan las siguientes funciones: establecer un sistema de numeración de referencia (BS), seleccionar los registros que aparecerán en pantalla (SD), encabezonamiento (HR), listar comandos antes de ser ejecutados (LC), no listar comandos (NL), poner valores en la memoria (SM), mostrar el valor de los registros seleccionados (DR), mostrar la última instrucción ejecutada (DL), rastrear n instrucciones (TI), ejecutar n instrucciones (RN), guardar el status (SS), recuperar el status (RS), salir del simulador (EX), borrar un macro (MD), borrar la librería de macros (MC), listar la librería de macros (ML), describir errores del simulador (ER), ayuda para obtener la descripción de un comando (HL), rastrear n saltos (TB), selección del juego normal de registros (NS), seleccionar el grupo de registros alternos (AS), desplegar/no desplegar las opciones seleccionadas (OP), cargar el programa desde el programa ensamblado al simulador (PL).

El funcionamiento del simulador se logra optimizar con una tabla de códigos de máquina equivalente al caso del cross-ensamblador pero ordenado de acuerdo al código de máquina, ademas de ésta, se maneja una tabla que simula los 64K Bytes de memoria, como ya se mencionó anteriormente; una tabla que simula los registros internos de la C.P.U.; una tabla para la librería de macros; una tabla que guarda los códigos de error detectables con su correspondiente explicación ; y por último una tabla con información pertinente al comando HL (HELP). Es importante indicar aquí que si bien es cierto usualmente a este conjunto de tablas se sumará un archivo conteniendo el programa fuente con su correspondiente ensamblaje, y sin errores; no es requisito indispensable, para cargar un programa y simularlo, que éste haya sido escrito en assembler y ensamblado usando los sistemas EDZ80 y ASZ80; de hecho se puede ensamblar a mano un programa y/o escribirlo directamente en código de máquina, luego cargarlo directamente en la memoria y simularlo. En este último caso se puede conseguir haciendo uso de la característica de desensamblador que tiene; las instrucciones equivalentes en lenguaje assembler de la Z80. Todo el proceso de simulación se maneja con un sistema, grupo de programas, llamado SMZ80 y cuyo diagrama de bloques bastante simplificado se presenta en la figura 6.

3.- EJEMPLOS DE USO DEL CROSS-ASSEMBLER

En el reporte de computador Figura 8, se puede seguir la secuencia de una corta sesión de trabajo usando el editor, cross assembler y librería de errores detectables. Como se puede ver, a la final se puede conseguir un progra,a sin errores de sintaxis o lenguaje, pero no se garantiza que esté libre de errores de lógica; será pues función del simulador el verificar si el programa, sin errores, hace lo que se supone debe hacer.

4.- EJEMPLOS DE USO DEL SIMULADOR

En el reporte de computador, Figura 9, se puede ver como se usa el simulador sin haber ensamblado el programa que se desea simular. La secuencia es la siguiente: se pone un valor en el acumulador A, luego se ingresa el código para desplazar en forma aritmética, el contenido del registro A hacia la izquierda, se ejecuta el código de instrucción y se verifica que el contenido del acumulador A se ha duplicado, como era de esperarse.

5.- USO COMBINADO DEL CROSS-ASSEMBLER Y SIMULADOR

El uso del cross-assembler en combinación con el simulador, representa una poderosa

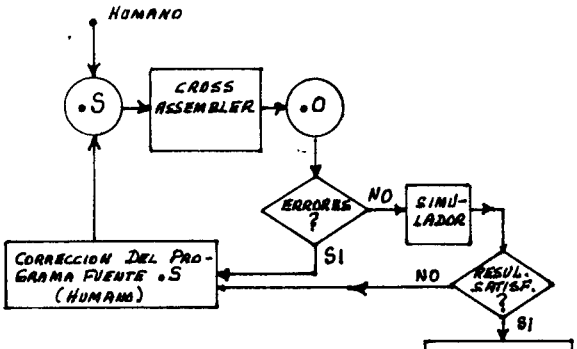

FIG. 7. SECUENCIA DE UTILIZACION DE UN CROSS ASSEMBLER Y SIMULADOR PARA EL DESARROLLO DE SISTEMAS BASADOS EN MICROPROCESADORES.

```
LABEL   MNEM OPERAND
        NEM   MASKO
       ORG    100
        LD    A,(40H) ; GET DATA
       AND    00001111B ; MASK 4 LSB'S
        LD    (41H),A ; STORE RESULT
        HALT
       END

/*
```

```
ALLOCATE MASKO SECTORS 10
MASKO.S(4010)
CART ID 4010   DB ADDR  4D50   DB CNT

==>
EDZ80

WHAT FILE  4?>
MASKO

     6 RECORDS ON FILE

     ( INITIAL LOAD OF PROGRAM? (Y/N) )
Y

     ENTER THE PROGRAM
     END LOAD WITH '/*' ON FIRST TWO COL

     7 RECORDS ON FILE

     END OF LOAD

A : ADDING LINES
I : INSERTING LINES
D : DELETING LINES
N : LISTING A NUMBER OF LINES
L : LISTING THE PROGRAM
M : MODIFYING A LINE
E : END OF JOB

SELECT APPROPRIATE ENTRY :
X
```

```
==========================
!  CROSS-ASSEMBLER ASZ80  !
==========================

  PARA ZILOG (Z80/Z80A) CPU

AUTOR : ING. EDGAR P. TORRES P. -  ENERO  1984
     ASZ80 ES DE PROPIEDAD DE EL  AUTOR

     CROSS ASSEMBLER ASZ80, VERSION 1.1
          QUITO - ECUADOR

ERR  LINE   ADDR OBJ.CODE      ASSEMBLER PROGRAM
===  ====   ==== ========      =================

*E05*00001                       NEM   MASKO
*E06*00002                      ORG    100
     00003 0000 3A4000           LD    A,(40H) ; GET DATA
     00004 0003 E60F             AND   00001111B ; MASK 4 LSB'S
     00005 0005 324100           LD    (41H),A ; STORE RESULT
     00006 0008 76               HALT
     00007                       END

     ***    2 ERRORS DETECTED IN THE ASSEMBLY

     END OF ASSEMBLER PRINTOUT

==>
ERROR

     ENTER ERROR CODE (TWO DIGITS)

ER
05
05   NAM MISSING : FIRST LINE OF PROGRAM MUST BE A NAM

     END OF JOB? (Y/N)
```

EDZ80 ERROR ASZ80

FIG. 8 EJEMPLO USANDO

```
SELECT APPROPRIATE ENTRY :
X
M

     ENTER RECORD POINTER
9999
0001

     ENTER NEW RECORD BELOW LAST ONE

                          LABEL  MNEM OPERAND
*E05*00001                       NEM  MASKO
                                 NAM  MASKO

     ENTER NEW RECORD BELOW LAST ONE

                          LABEL  MNEM OPERAND
*E06*00002                       ORG  100
                                 ORG  100

ERR  LINE  ADDR OBJ.CODE      ASSEMBLER PROGRAM
===  ====  ==== ========      =================

     00001                       NAM  MASKO
     00002 0064                  ORG  100
     00003 0064 3A4000           LD   A,(40H) ; GET DATA
     00004 0067 E60F             AND  00001111B ; MASK 4 LSB'S
     00005 0069 324100           LD   (41H),A ; STORE RESULT
     00006 006C 76               HALT
     00007                       END

     NO ERRORS DETECTED IN THE ASSEMBLY OF THIS PROGRAM

     END OF SUCCESSFUL ASSEMBLY

SMZ80

WHAT FILE 14??
MASKO

1
```

```
NAM  MASKO
ORG  100
LD   A,(40H) ; GET DATA
AND  00001111B ; MASK 4 LSB'S
LD   (41H),A ; STORE RESULT
HALT
END
```

FIG. 8 (CONT.)

```
                    ======================
                    |  SMZ80 SIMULATOR   |
                    ======================

                  FOR ZILOG (Z80/Z80A) CPU)

          SMZ80 IS THE PROPERTY OF THE AUTHOR
     AUTHOR: ING. EDGAR P. TORRES P. - FEBRUARY 1984

          SMZ80 SIMULATOR, RELEASE 1.1
                QUITO - ECUADOR

?
SD FBSPMNT
?
SR F00,A12.DR
F = 0 0
B = 0 0
S = 0 0 0 0
P = 0 0 0 0
M = 0 0 0 0
N = 0 0 0 0
T = 0 0 0 0 0 0 0 0
?
SB F00,B12
*E01* UNDEFINED SIMULATOR COMMAND
?
ER 01

01   UNDEFINED SIMULATOR COMMAND :
     ----------------------------
     AN UNDEFINED SIMULATOR COMMAND WAS ENTERED. SIMULATOR COMMAND MUST
     CONSIST OF TWO ALPHABETIC CHARACTERS FOLLOWED BY ONE SPACE.
```

FIG. 9 USO DEL SIMULADOR SMZ80

```
/
SR B12
?.
DR
F = 0 0
B = 1 2
SM 40,CB,20.RN 40,1
-------------------------------------------------------------------------
 MM    (O : ASSM.CODE)  UU  ZZ  YYYY      NN      T     SZ-A-PNC
 SP     PC     IX     IY      FF  AA  BB  CC  DD  EE  HH  LL     II  RR
-------------------------------------------------------------------------

0040  SLA  B                          |  0042  0000008  00-0-100
0000  0042  0000  0000  |  04  12  24  00  00  00  00  00  |  00  00
.
DR
F = 0 4
B = 2 4
S = 0 0 0 0
P = 0 0 4 2
M = 0 0 4 0
N = 0 0 4 2
T = 0 0 0 0 0 0 0 8
 ?
 MC
 ?
 ML

 MACRO LIBRARY LISTING
 =====================

 ?
 MAC1 (SD FDEHLSPT.DR.SM 40,C1,23,F0,38,00,00.DM 40,6.PL)

 ?
 MAC2 (TI.HR 08.RN 500,10.DR.DM 40,6)
 ?
 ML

 MACRO LIBRARY LISTING
 =====================

   1: MAC1  (SD FDEHLSPT.DR.SM 40,C1,23,F0,38,00,00.DM 40,6.PL)
   2: MAC2  (TI.HR 08.RN 500,10.DR.DM 40,6)
 ?
 MAC1
 F = 0 0
 D = 0 0
 E = 0 0
 H = 0 0
 L = 0 0
 S = 0 0 0 0
 P = 0 0 0 0
 T = 0 0 0 0 0 0 0 0
 (0040) = C1
 (0041) = 23
 (0042) = F0
 (0043) = 38
 (0044) = 00
 (0045) = 00
 LOADING TO MEMORY AN (.S/.O) PROGRAM OF    9 RECORDS
 STARTING ON MEMORY LOCATION 0500 HEXADECIMAL
 ?
 MAC2
 -------------------------------------------------------------------------
  MM    (O : ASSM.CODE)  UU  ZZ  YYYY      NN      T     SZ-A-PNC
  SP     PC     IX     IY      FF  AA  BB  CC  DD  EE  HH  LL     II  RR
 -------------------------------------------------------------------------

 0500  LD   HL,(Y)                  0040  |  0040  0000016  00-0-000
 0000  0503  0000  0000  |  00  00  00  00  00  00  23  C1  |  00  00

 0503  LD   DE,(Y)                  0042  |  0042  0000036  00-0-000
 0000  0507  0000  0000  |  00  00  00  00  38  F0  23  C1  |  00  00

 0507  ADD  HL,DE                         |  0508  0000047  00-0-000
 0000  0508  0000  0000  |  00  00  00  00  38  F0  50  B1  |  00  00

 0508  LD   (Y),HL                  0044  |  0044  0000067  00-0-000
 0000  050B  0000  0000  |  00  00  00  00  38  F0  50  B1  |  00  00

 050B  HALT                               |  050D  0000071  00-0-000
 0000  050D  0000  0000  |  00  00  00  00  38  F0  50  B1  |  00  00
```

FIG. 9 (CONT.)

FIG. 10 SUMA DE DOS NUMEROS DE 16 BITS

```
F = 0 0
D = 3 8
E = F 0
H = 5 C
L = B 1
S = 0 0 0 0
P = 0 5 0 D
T = 0 0 0 0 0 7 1
(0040) = C1
(0041) = 23
(0042) = F0
(0043) = 38
(0044) = B1
(0045) = 5C
?
EX
```

FIG. 10 (CONT.)

```
L
    1:      00001                           NAM   DAT11
    2:      00002 0200                      ORG   200H
    3:      00003 0200 3E05                 LD    A,5
    4:      00004 0202 3D         AGAIN:    DEC   A
    5:      00005 0203 2803                 JR    Z,OUT
    6:      00006 0205 C30202               JP    AGAIN
    7:      00007 0208 06FE       OUT:      LD    B,0FEH
    8:      00008 020A 76                   HALT
    9:      00009                           END
?
PL
LOADING TO MEMORY AN (.S/.O) PROGRAM OF   10 RECORDS
STARTING ON MEMORY LOCATION 0200 HEXADECIMAL
?
RN 200,100
```

MM	(O : ASSM.CODE)	UU	ZZ	YYYY	NN	T	SZ-A-PNC
SP	PC IX IY		FF	AA BB	CC DD	EE HH	LL II RR
020A	HALT				020B	0000118	01-0-010
0000	020B 0000 0000		42	00 FE	00 00	00 00	00 00
0205	JP Y			0202	0202	0000264	00-0-010
0000	0202 0000 0000		02	04 FE	00 00	00 00	00 00
0205	JP Y			0202	0202	0000285	00-0-010
0000	0202 0000 0000		02	03 FE	00 00	00 00	00 00
0205	JP Y			0202	0202	0000306	00-0-010
0000	0202 0000 0000		02	02 FE	00 00	00 00	00 00
0205	JP Y			0202	0202	0000327	00-0-010
0000	0202 0000 0000		02	01 FE	00 00	00 00	00 00
0203	JR Z,U	03			0208	0000343	01-0-010
0000	0208 0000 0000		42	00 FE	00 00	00 00	00 00
020A	HALT				020B	0000354	01-0-010
0000	020B 0000 0000		42	00 FE	00 00	00 00	00 00

```
?
EX
==>
```

FIG. 11 CARGA Y DECAEMENTO DE UN ACUMULADOR HASTA CERO. USO COMANDO TB.

herramienta para el diseño e implementación de sistemas basados en microprocesadores. En los ejemplos que se presentan a continuación ya se asume que los programas han pasado por la fase de desarrollo, ensamblaje y depuración de errores hasta obtener un código objeto sin errores de sintaxis o gramática.

a. Ejemplo 1.- En el reporte de computador de la Figura 10, se presenta un programa que suma dos números de dieciseis bits, en las localidades 40,41 y 42,43; el resultado se coloca en la localidad 44,45, la forma en la que varía el contenido de los diferentes registros se muestra en el reporte.

b. Ejemplo 2.- En el reporte de la Figura 11, se presenta un programa que carga el acumulador con un valor y luego lo decrementa hasta que llega a cero. Aquí, se hace uso del comando TB, que permite visualizar el contenido de los registros solo cuando hay un salto en el programa.

CONCLUSIONES

En el presente trabajo se ha visualizado y verificado la gran utilidad que el Traductor Cross-assembler y el Simulador tienen como herramientas en el campo de desarrollo de software para microprocesadores. El trabajo, motivo de este artículo, representa la primera versión (Versión 1.1) por lo tanto es probable que con el uso en labores de diseño de sistemas se detecten pequeñas inconsistencias, al mismo tiempo que queden en evidencia posibles mejoras a la actual organización. Será pues objeto de un trabajo futuro el hacer las respectivas modificaciones y actualizaciones para que el trabajo se torne más completo y depurado.

Vale mencionar el hecho de que este tipo de sistemas se podría usar con mucho éxito en cursos de microprocesadores por las facilidades didácticas que presta. Finalmente es conveniente tomar nota del hecho de que el microprocesador 8080 es ciento por ciento compatible a nivel de código de máquina con el Z80, por lo que aplicaciones potenciales podrían surgir para el simulador de la Z80 relacionadas al software de la 8080.

BIBLIOGRAFIA

(1) BARRON D.W., Assemblers and Loaders, American Elsevier, New York, 1972.

(2) CAMPBELL KELLY M., An Introduction to Macros, American Elsevier Inc., New York, 1973

(3) Z80/Z80A Technical Manual, Zilog, 1977.

(4) LEWIS P. M., ROSENKRANTZ D.J., STEARNS R., Compiler Design Theory, Addison-Wesley, 1978.

(5) KOBAYASHI H., Modeling and Analysis, Addison Wesley, 1978.

(6) SIPPL CHARLES J., Computer Dictionary, Sams, 1974.

(7) CANNON DON L., Understanding Microprocessors, Radio Shack, 1979.

(8) HERNANDEZ A. MIRIAM B., Traductor Cross Assembler para el Micropr. M6800, EPN, 1981.

TORRES PROAÑO, EDGAR P. Nació en Quito, Ecuador, el 29 de mayo de 1955. Obtuvo el título de Ingeniero en Electrónica y Telecomunicaciones en al Escuela Politécnica Nacional de Quito, en agosto de 1978 y el título de Master of Science in Electrical Engineering en Ohio University, Athens, E.E.U.U., en septiembre de 1980. Actualmente trabaja en el I.E.TEL. en el Departamento de Procesamiento Automático de Datos. Desde octubre de 1981 es profesor de la Escuela Politécnica del Ejército, en donde actualmente dicta la materia de Microprocesadores y Sistemas Digitales. Desde marzo de 1982, se halla vinculado a la Escuela Politécnica Nacional en donde ha dictado la materia de Sistemas Discretos de Control y ha dirigido Proyectos y Tesis de Grado.