COMPILADOR Y PROCESADOR PARA UN MICROLENGUAJE ESTRUCTURADO

ESQUETINI, CESAR ING.
ESCUELA POLITECNICA NACIONAL

RESUMEN

La idea fundamental es didáctica, mostrar a- los estudiantes el proceso que sufre un pro grama dentro de un computador digital.

En su litro " Algorithms + Data Structures = Programs ", Wirth estudia los elementos esen ciales de la compilación, con la ayuda de un minipascal que le denomina PLO. Para inter pretar el código generado por el compilador (código objeto), Wirth define como una máqui na ficticia (software) al procesador PLO.

Este trabajo presenta una realización mate rial (hardware) de dicho procesador.

INTRODUCCION

Editado un texto, existe un programa llamado compilador que es el encargado de traducir el texto editado, a una serie de instruccio- nes que pueden ser interpretadas por un pro cesador.

Hemos escogido como ejemplo el Compilador - PLO, por ser, como lo define su propio autor, lo suficientemente sencillo para que la com prensión sea fácil y lo suficientemente com plejo para que exista interés el analizarlo detenidamente.

La máquina, material que hemos diseñado y construído, es un procesador que realiza la misma tarea que el programa intérprete. En la realización material del procesador PLO, preferimos la utilización de circuitos sim ples; simples, en el sentido de la función- que éllos realizan, de tal manera que el es tudiante pueda observar, en lo posible, toda la información interna del procesador en cualquier sitio e instante deseados. Esta de cisión significa la utilización de un mayor número de elementos y en cada uno de ellos displays hexadecimales que nos indiquen su estado y contenido.

1.- COMPILADOR

1.1 DESCRIPCION GENERAL

La estructura del compilador PLO es bastante completa, admite operaciones lógicas y arit- méticas sobre un único tipo de datos, los en teros. Posee las principales estructuras de los lenguajes de alto nivel: BEGIN, END, IF THEN, WHILE, declaraciones y llamadas a pro cedimientos.

La utilización de los procedimientos nos- muestra la visibilidad de los objetos del lenguaje (las variables y los procedimien - tos). Un objeto es visible solamente al inte rior del procedimiento donde él ha sido de - clarado. Las variables son locales a los pro cedimientos.

1.2 ARQUITECTURA DE LA MAQUINA PLO

Como los procedimientos pueden ser activados recursivamente, no puede asignarse memoria a las variables antes que se produzca la llama da de los mismos, razón ésta para escoger u na arquitectura a pila que da facilidades pa ra una asignación dinámica de memoria.

Los elementos de esta máquina son los si- guientes (Fig. 1):

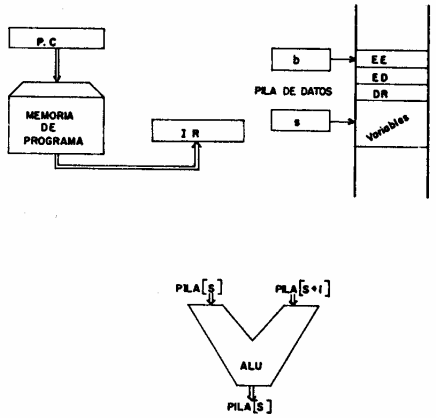

Fig. 1.- Elementos de la máquina PLO.

- Una memoria de programa que contiene el có digo objeto generado por el compilador.

- Una memoria de datos, organizada en forma de pila. Todas las operaciones del pro- cesador se efectúan con los dos elementos de la parte superior de la pila (cima, cima-1) y se reemplaza los operadores por el resultado (cima).

  Esta memoria se descompone en bloques, que son activados dinámicamente en el momento que existe una llamada a un procedimiento. Las tres primeras palabras del bloque - contienen una información interna al pro- cedimiento: el enlace estático (EE), el enlace dinámico (ED), y la dirección de - regreso (DR). Las palabras siguientes del bloque contienen los valores de las varia bles locales al procedimiento.

  El enlace estático direcciona el bloque - de activación del procedimiento donde fue declarado. El enlace dinámico direcciona el bloque de activación del procedimiento que llamó al procedimiento que se ejecuta en ese momento (el bloque de activación - anterior).

```
            (**** PROGRAMA INTERPRETE ****)

PROGRAM EXECUTER(CODE,output);
CONST
     LONGPILA=500;
     MAXNIV=3;
     MAXID=2047;
     MAXIC=200;(* tamano maximo del codigo *)
TYPE NOMF=PACKED ARRAY[1..5] OF CHAR;
      FCN=(lit,opr,car,alm,lla,ins,sal,sac);
     LINEAC=PACKED RECORD
               fc:FCN;
               f1:NOMF;
               ni1:0..MAXNIV;
               di1:0..MAXID;
            END;
     CODEA=FILE OF LINEAC;
 VAR
  d:INTEGER;(* contador del programa *)
  b:INTEGER;(* registro de la base de la pila *)
  s:INTEGER;(* registro de la parte superior de la pila *)
  i:LINEAC;(* registro de instruccion *)
  p:ARRAY[1..LONGPILA] OF INTEGER;(* memoria de datos *)
  code:CODEA;
  i1:INTEGER;
  codigo:ARRAY[0..MAXIC] OF LINEAC;
(**********)
FUNCTION BASE(NI1:INTEGER):INTEGER;
   VAR b1:INTEGER;
BEGIN
  b1:=b;
  while ni1 ) 0 do
  begin
    b1:=p[b1];
    ni1:=ni1-1;
  end;
  BASE:=b1;
END;(* BASE *)
(**********)
BEGIN
  RESET(code);
  i1:=-1;
  while not eof(code) do
  begin
    i1:=i1+1;
    with code^ do
    begin
      codigo[i1].fc:=code^.fc;
      codigo[i1].f1:=code^.f1;
      codigo[i1].ni1:=code^.ni1;
      codigo[i1].di1:=code^.di1;
    end;
    get(code);
  end;
  s:=0;
  b:=1;
  d:=0;
  p[1]:=0;
  p[2]:=0;

  p[3]:=0;
  repeat
  i:=codigo[d];
  d:=d + 1;
  with i do
  case fc of
    lit:begin
          s:=s+1;
          p[s]:=di1;
        end;
   opr:case di1 of (* operador *)
         0:begin (* retornar *)
             s:=b-1;
             d:=p[s+3];
             b:=p[s+2];
           end;
         1:p[s]:=-p[s];
         2:begin
             s:=s - 1;
             p[s]:=p[s] + p[s+1];
           end;
         3:begin
             s:=s - 1;
             p[s]:=p[s] - p[s+1];
           end;
         4:begin
             s:=s - 1;
             p[s]:=p[s] * p[s+1];
           end;
         5:begin
             s:=s - 1;
             p[s]:=p[s] DIV  p[s+1];
           end;
         6:p[s]:=ORD(ODD(p[s]));
         8:begin
             s:=s - 1;
             p[s]:=ORD(p[s] = p[s+1]);
           end;
         9:begin
             s:=s - 1;
             p[s]:=ORD(p[s] <> p[s+1]);
           end;
        10:begin
             s:=s - 1;
             p[s]:=ORD(p[s] < p[s+1]);
           end;
        11:begin
             s:=s - 1;
             p[s]:=ORD(p[s] >= p[s+1]);
           end;
        12:begin
             s:=s - 1;
             p[s]:=ORD(p[s] > p[s+1]);
           end;
        13:begin
             s:=s - 1;
             p[s]:=ORD(p[s] <= p[s+1]);
           end;
       end;(* case interno de los operadores *)
   car:begin
         s:= s + 1;
         p[s]:=p[BASE(ni1)+di1];
       end;
   alm:begin
         p[BASE(ni1)+di1]:=p[s];
         writeln(output,p[s]);
         s:=s - 1;
       end;
   lla:begin (* genera nuevas marcas en el modulo *)
         p[s+1]:=BASE(ni1);
         p[s+2]:=b;
         p[s+3]:=d;
         b:=s+1;
         d:=di1;
       end;
   ins:s:=s + di1;
   sal:d:=di1;
   sac:begin
         if p[s] = 0 then d:=di1;
         s:=s - 1;
       end;
 end;(* del case *)

 until d=0;
 writeln(output,'*** FIN DE LA EXECUTION ***');
END.
```

El enlace dinámico y la dirección de regreso son necesarias para retomar el control del programa al final de un procedimiento mientras que el enlace estático es necesario para direccionar las variables, ya que el compilador no puede proporcionar la dirección absoluta. Cada dirección de una variable se compone de dos partes: una diferencia de nivel (n) que indica la distancia en niveles estáticos entre el bloque que se ejecuta y el bloque de activación donde fue declarada, y el desplazamiento (d) que indica la posición de la variable al interior del bloque. La Fig.2 muestra la diferencia de niveles entre los procedimientos y la organización que tendría la pila.

-Un registro S que direcciona la cima de la pila.

-Un registro B que direcciona el bloque de activación que se está ejecutando.

-Un registro código que contiene el código de la instrucción que se ejecuta.

-Un registro PC que contiene la dirección de la próxima dirección a ejecutarse.

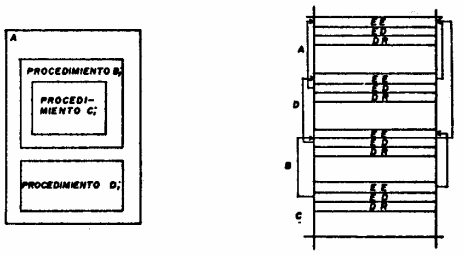

Fig.2.- Organización de la pila de datos

1.3 Repertorio de instrucciones.

1.3.1 Instrucciones que administran la pila

\- LIT 0, d

Carga la constante d en la cima de la pila:

s:=s+1

P(s):=d

\- CAR n,d

Carga a la cima de la pila la variable, cuya dirección se calcula en base de n y d:

s:=s+1

P(s):=P(base(n)+d)

\- ALM n,d

Almacena el valor que se encuentra en la cima de la pila a la variable, cuya dirección se calcula en base de n y d:

P(base(n)+d):=P(s)

s:=s-1

1.3.2 Instrucciones de control

\- LLA n,d

Llama al procedimiento que comienza en la dirección d. n indica la diferencia de niveles estáticos entre el procedimiento llamado y aquel que llamó:

P(s+1):=base(n)

P(s+2):=b

P(s+3):=PC

B:=s+1

PC:=d

\- SAL 0,d

Salto incondicional a la dirección d:

PC:=d

\- SAC 0,d

Salto condicional a la dirección d. El valor de la condición se encuentra en la cima de la pila:

Si P(s)=0 entonces PC:=d

s:=s-1

\- OPR 0,0

Regreso de un procedimiento:

s:=s-1

PC:=P(s+3)

b:=P(s+2)

1.3.3 Operaciones aritméticas

\- OPR 0,1

Negación P(s):=-P(s)

\- OPR 0,2

Suma: s:=s-1

P(s):=P(s)+P(s+1)

\- OPR 0,3

Resta: s:=s-1

P(s):=P(s)-P(s+1)

1.3.4 Operaciones lógicas

\- OPR 0,6

Prueba si la cima de la pila es impar:

P(s)=ODD(P(s))

\- OPR 0,8

Prueba si son iguales los dos valores de la cima y cima -1 de la pila:

s:=s-1

P(s):=P(s)=P(s+1)

\- OPR 0,9

Prueba si son diferentes los dos valores de la cima y cima -1 de la pila:

s:=s-1

P(s):=P(s)<>P(s+1)

\- OPR 0,10

Prueba si es más pequeño el valor de la cima -1 con respecto al valor de la cima:

s:=s-1

P(s):=P(s)<P(s+1)

\- OPR 0,11

Prueba si es más grande o igual el valor de la cima -1 con respecto al valor de la cima:

s:=s-1

P(s):=P(s)>=P(s+1)

- OPR 0,12

Prueba si es más grande el valor de la cima -1 con respecto al valor de la cima:

s:=s-1

P(s):=P(s)>P(s+1)

- OPR 0,13

Prueba si es más pequeño o igual el valor de la cima -1 con respecto al valor de la cima:

s:=s-1

P(s):=P(s)<=P(s+1)

2.- PROCESADOR

2.1 FORMATO DE LA PALABRA DE INSTRUCCION

La palabra de memoria escogida para la memoria del programa y para la pila de datos es de ocho bits. La palabra de instrucción se formará con dos palabras de memoria (16 bits), como indica la Fig. 3.

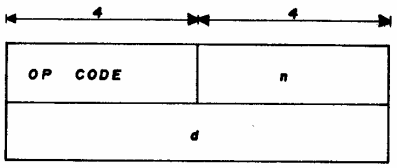

Fig.3.- Formato de la palabra de instrucción.

El formato general de las instrucciones será de cuatro bits para el código de operaciones, lo que significa que se puede definir 16 posibles operaciones diferentes. La figura 4 muestra la codificación utilizada. Los cuatro bits siguientes indicarán la diferencia de nivel (n) de los objetos, lo que significa que el valor máximo de n será 15. Los ocho restantes bits servirán para contener una constante, una dirección o el tipo de operación aritmética o lógica que se realice, el valor máximo posible será de 255.

A continuación analizaremos los diferentes componentes del computador.

2.2 ENTRADA - SALIDA

Pese a que en la descripción de la máquina original PLO no existen las instrucciones de entrada y de salida, creimos conveniente crearlas a nivel de instrucciones cable, con el objeto que el estudiante tenga flexibilidad en cambiar los datos de entrada y en visualizar los resultados.

El formato general de estas dos instrucciones es:

IN 0,d OUT 0,d

siendo d la dirección del periférico donde se va a escribir o leer. Como para especificar d contamos con ocho bits se podrá direccionar 256 periféricos de entrada y 256 periféricos de salida. En nuestro caso utilizaremos dos de entrada y dos de salida.

CODIGO				INST
0	0	0	0	Ø
0	0	0	1	Ø
0	0	1	0	Ø
0	0	1	1	Ø
0	1	0	0	Ø
0	1	0	1	Ø
0	1	1	0	LIT
0	1	1	1	CAR
1	0	0	0	ALM
1	0	0	1	LLA
1	0	1	0	INS
1	0	1	1	SAL
1	1	0	0	SAC
1	1	0	1	OPR
1	1	1	0	IN
1	1	1	1	OUT

Fig.4.- Código de operaciones.

2.3 UNIDAD DE TRATAMIENTO

La figura 6, muestra la arquitectura final de la unidad de tratamiento con el módulo entrada-salida.

Dentro de la unidad de tratamiento podemos distinguir los siguientes módulos:

2.3.1 MODULO ARITMETICO Y LOGICO

El módulo aritmético y lógico tiene como función realizar las operaciones aritméticas y lógicas.

La primera idea que nos surgió fue la de tomar un circuito comercial bastante complejo, como es el caso del circuito AM2901, que tiene características interesantes. Posee una memoria interna de 16*4, que bien podría servirnos para almacenar los resultados de los cálculos intermedios; y, un repertorio de 512 funciones diferentes. Mas, como nuestro fin es didáctico escogimos el circuito comercial LS181 (32 funciones) cuya tabla de operaciones se muestra en la tabla 1 y además incluimos dos registros auxiliares A y Al para almacenar los resultados intermedios.

Entre las señales de estado, que la unidad de tratamiento debe enviar a la unidad de comando, tenemos las banderas de comparación entre dos números. La unidad de tratamiento deberá indicar si los dos números son iguales, diferentes, si el uno es mayor que el otro, mayor o igual, menor, menor o igual. La generación de estas señales se puede realizar por un método material (hardware) o por un programa (método software), que consiste en aprovechar la -

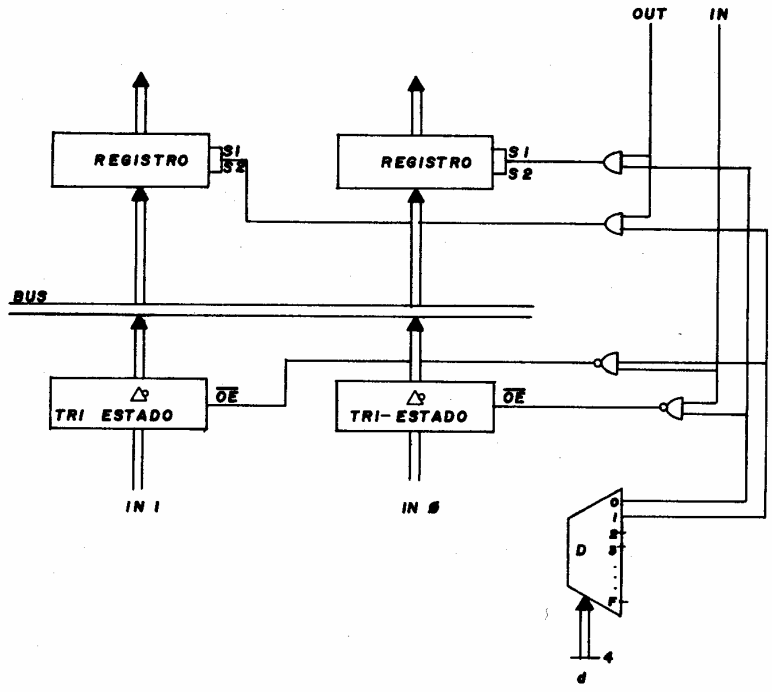

Fig. 5.- Módulo entrada-salida.

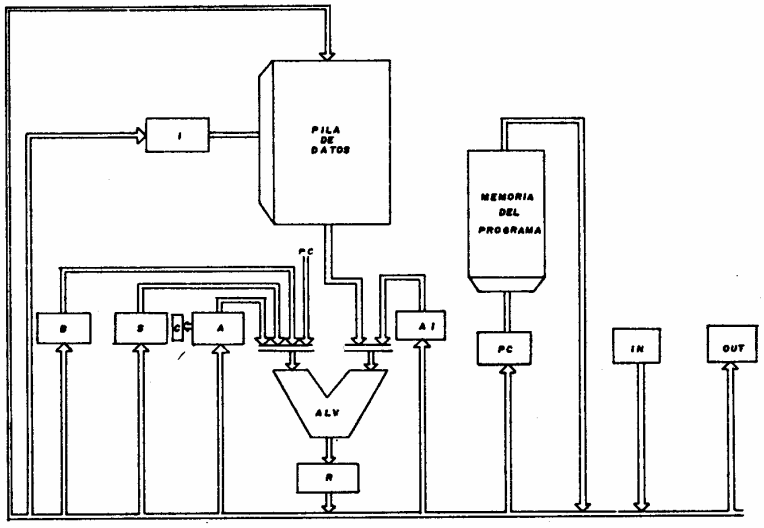

Fig. 6.- Arquitectura de la unidad de tratamiento.

función de substracción de nuestra ALU. Nosobros nos hemos inclinado por la primera solución utilizando el circuito comercial 74LS85 (identificado con la letra C en la figura 6), que es un comparador de dos números enteros positivos de cuatro bits y que proporciona tres salidas: A<B, A=B y A>B. A partir de estas señales y un circuito combinatorio debemos encontrar las seis señales buscadas (A<B, A<=B, A=B, A<>B, A>B, A>=B). La convención adoptada para la representación de los números negativos ha sido la de complemento a dos.

El registro I, que es propiamente el puntero de la pila (stack-pointer); debido a que el registro S se le utiliza para realizar cálculos. Los registros A y Al son utilizados como acumuladores. El registro R, que impide que los resultados de la ALU aparezcan instantáneamente en el bus, permitiendo además la sincronización de la escritura en la memoria de datos.

2.4 MICROPROGRAMACION

El objetivo básico de la microprogramación

opération	description	M	S_3	S_2	S_1	S_0
OP_0	$(C_4F) = A + C_0$	0	0	0	0	0
OP_1	$(C_4F) = (A \vee B) + C_0$	0	0	0	0	1
OP_2	$(C_4F) = (A \vee \overline{B}) + C_0$	0	0	0	1	0
OP_3	$(C_4F) = 15. + C_0$	0	0	0	1	1
OP_4	$(C_4F) = A + (A \wedge \overline{B}) + C_0$	0	0	1	0	0
OP_5	$(C_4F) = (A \vee B) + (A \wedge \overline{B}) + C_0$	0	0	1	0	1
OP_6	$(C_4F) = A + \overline{B} + C_0$	0	0	1	1	0
OP_7	$(C_4F) = (A \wedge \overline{B}) + 15. + C_0$	0	0	1	1	1
OP_8	$(C_4F) = A + (A \wedge B) + C_0$	0	1	0	0	0
OP_9	$(C_4F) = A + B + C_0$	0	1	0	0	1
OP_{10}	$(C_4F) = (A \vee \overline{B}) + (A \wedge B) + C_0$	0	1	0	1	0
OP_{11}	$(C_4F) = (A \wedge B) + 15. + C_0$	0	1	0	1	1
OP_{12}	$(C_4F) = A + A + C_0$	0	1	1	0	0
OP_{13}	$(C_4F) = (A \vee B) + A + C_0$	0	1	1	0	1
OP_{14}	$(C_4F) = (A \vee \overline{B}) + A + C_0$	0	1	1	1	0
OP_{15}	$(C_4F) = A + 15. + C_0$	0	1	1	1	1
OP_{16}	$F = \overline{A}$	1	0	0	0	0
OP_{17}	$F = \overline{A \vee B}$	1	0	0	0	1
OP_{18}	$F = \overline{A} \wedge B$	1	0	0	1	0
OP_{19}	$F = \emptyset$	1	0	0	1	1
OP_{20}	$F = \overline{A \wedge B}$	1	0	1	0	0
OP_{21}	$F = \overline{B}$	1	0	1	0	1
OP_{22}	$F = A \oplus B$	1	0	1	1	0
OP_{23}	$F = A \wedge \overline{B}$	1	0	1	1	1
OP_{24}	$F = \overline{A} \vee B$	1	1	0	0	0
OP_{25}	$F = \overline{A \oplus B}$	1	1	0	0	1
OP_{26}	$F = B$	1	1	0	1	0
OP_{27}	$F = A \wedge B$	1	1	0	1	1
OP_{28}	$F = 15.$	1	1	1	0	0
OP_{29}	$F = A \vee \overline{B}$	1	1	1	0	1
OP_{30}	$F = A \vee B$	1	1	1	1	0
OP_{31}	$F = A$	1	1	1	1	1

Tabla 1.- Operaciones aritméticas y lógicas de la ALU

2.3.2 MODULO PILA

El módulo pila está constituído por una memoria RAM 256*8 y que es la que va a contener los datos del programa de aplicación.

2.3.3 MODULO MEMORIA DE PROGRAMA

El módulo memoria de programa está constituído por una memoria de 256*8, en la cual se almacenará el programa objeto que se desee ejecutar.

2.3.4 MODULO REGISTROS

Se utilizan siete registros: Los registros B,S,PC, cuyas funciones se indicaron al mencionar la arquitectura de la máquina PL0. El registro código, que no se utiliza, por cuanto la salida de la memoria de programa, se conecta directamente al bus de datos a través de un pasador tri-estado.

es hacer una correspondencia entre una instrucción objeto del lenguaje PL0, con una o varias microinstrucciones (instrucciones ejecutadas por el procesaro en un sólo período de reloj), que reconoce y ejecuta nuestra arquitectura. Figura 6.

A continuación presentamos, por falta de espacio, únicamente la microprogramación para la etapa de inicialización, para las instrucciones LIT 0,d,ALM n,d y para la subrutina dirección. Hemos tomado como convención de notación, que el elemento que recibe la información (destinación) se escriba a la izquierda y el elemento que proporcion a la información (fuente) a la derecha. El símbolo de asignación que hemos escogido es los dos puntos seguidos del signo igual.

En el momento inicial suponemos la existencia de un interruptor CLEAR que coloca ce

ros en los registros.

```
Inicialización

                                (*** P(0):=0 ***)
bus:=s              0           9003
ri:=bus             1           8A03
p(s):=pc            2           B007
                                (*** P(1):=0 ***)
bus:=s+1            3           900E
s:=bus              4           8403
bus:=s              5           9003
ri:=bus             6           8403
p(ri):=pc           7           B007
                                (*** P(2):=0 ***)
bus:=s+1            8           900B
s:=bus              9           8403
bus:=s              A           9003
ri:=bus             B           8A03
p(ri):=pc           C           B007
                                (*** s:=FF ***)
bus:=pc-1           D           B007
s:=bus              E           8403
                                (*** pc:=FF ***)
bus:=pc-1           F           B073
pc:=bus            10           8803
                                (*** pc:=pc+1 ***)
1$ bus:=pc+1       11           B00B
pc:=bus            12           8803
2$ case cableado   13           4000

Subrutina dirección

bus:=pc+1          25           B00B
pc:=bus            26           8803
al:=d              27           8C01
bus:=a+al          28           A053
al:=bus            29           8C03
return             2A           E000

LIT 0,d

bus:=pc+1          2B           B00B
pc:=bus            2C           8803
bus:=s+1           2D           900B
s:=bus             2E           8403
bus:=s             2F           9003
ri:=bus            30           8A03
p(ri):=d           31           8005
goto 1$            32           0011

ALM n,d

call base          33           C014
call dirección     34           C025
bus:=s             35           9003
ri:=bus            36           8A03
bus:=p(ri)         37           81C3
a:=bus             38           8603
bus:=al            39           80C3
ri:=bus            3A           8A03
p(ri):=a           3B           A007
bus:=s-1           3C           9073
s:=bus             3D           8403
goto 1$            3E           0011
```

La microinstrucción CASE, es un CASE cableado que permite apuntar una localidad de memoria de acuerdo al código de operación.

La subrutina base y dirección son procedimientos que tienen como función calcular la dirección de la base del bloque, donde fue definida la variable y la dirección final, sumando a la base el desplazamiento.

Además, de todas las microinstrucciones indicadas, es necesario la microinstrucción NOP no-operación que permite a la unidad de control hacer diferentes operaciones sin cambiar en nada el estado de la unidad de tratamiento; es decir, la operación que hace "nada".

En la tabla 2 se puede observar una parte de las microinstrucciones necesarias, para ser ejecutadas en la unidad de tratamiento, a fin de simular las instrucciones objeto de PLO, y los valores que deben tener cada una de las variables de los elementos para cumplir su fin.

Como ejemplo, para una mejor comprensión, tomemos la primera microinstrucción bus:=s. En la fila correspondiente a esta microinstrucción, se puede observar los diferentes valores que toman las variables (únicamente B1 y B2 tienen el valor de uno lógico). El efecto que tiene esta microinstrucción, es que la información que se encuentra en el registro S pasa a ser un operando de la ALU, el cual debe aparecer a la entrada del registro R y en el momento que exista una transición positiva del reloj, en el bus.

2.5 UNIDAD DE CONTROL

Es la encargada del control de todos los elementos del procesador. Ordena la unidad de tratamiento efectuar una determinada microoperación y, de acuerdo al estado de las operaciones, controla la secuencia de la ejecución.

Leído el código de operación de una palabra de instrucción, la unidad de control identifica el código y ordena efectuar una serie de microinstrucciones a la unidad de tratamiento; en otras palabras, significa poner a las variables de los diferentes elementos de la unidad de tratamiento, los valores correspondientes a la microinstrucción deseada. Ver tabla 2.

Es posible dos realizaciones para la implementación de la unidad de control: La realización cableada que obedece las reglas clásicas de diseño de una máquina secuencial, utilizando sistemas lógicos discretos, lo que significa que la información se encuentre dispersa y que cualquier modificación de las especificaciones del problema, implica una modificación cableada del sistema, o eventualmente un nuevo diseño; y, la **segunda**, la realización microprogramada que es la que escogimos y consiste en servirnos de una máquina de decisión binaria, como la que aparece en la figura 7, con una micromemoria donde se encuentra almacenada la información de control. Esta segunda solución no tiene los problemas de la primera, razón por la cual es el método utilizado actualmente en el diseño de procesadores.

2.5.1 REPERTORIO DE INSTRUCCIONES

La máquina que nos servirá como unidad de control, debe efectuar el repertorio de instrucciones que indica la figura 8.

La instrucción de salida DO, cuya función es poner el valor de las variables en la unidad de tratamiento, es la que fija el ancho de la palabra de la micromemoria. En nuestro caso, como las variables de control

son catorce más dos necesarias para el código de operación de la microinstrucción, hemos fijado la longitud de la palabra de memoria en 16 bits.

La microinstrucción de test binario: IF THEN GOTO.

La microinstrucción de llamada a un subprograma: CALL.

La microinstrucción de test múltiple: CASE.

En la figura 9 se muestra el formato de estas instrucciones.

La unidad de control se compone de diferentes módulos:

2.5.2 MODULO DE LA MICROMEMORIA

El módulo de la micromemoria está compuesta de una memoria tipo EPROM, que es la que contiene el microprograma y está direccionada por la salida del secuenciador.

2.5.3 MODULO MEMORIA DECODIFICADORA

Se le llama así, por el tipo de trabajo que realiza. Dado un código de operación, este servirá como dirección. El contenido de es

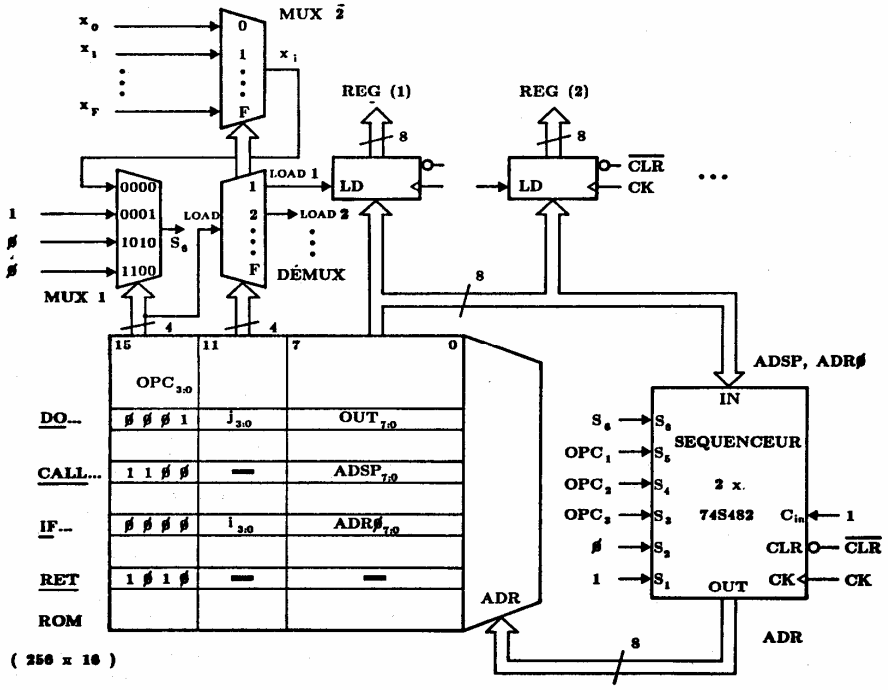

Fig. 7.- Máquina de decisión binaria de cuatro instrucciones.

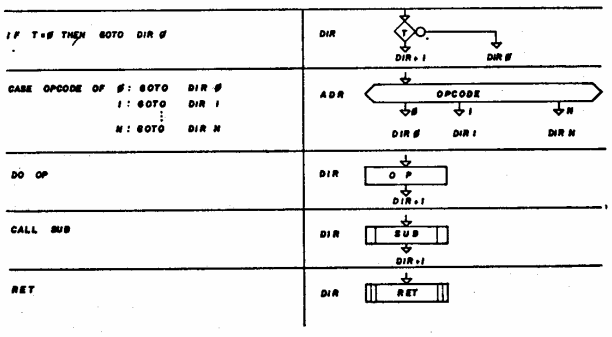

Fig. 8.- Microinstrucciones de la unidad de control

INSTRUCCION	V2	V1	I2	I1	I0	A2	M	S3	S2	S0	C0	W	B2	R1
bus:=s	0	1	0	0	0	0	0	0	0	0	0	0	1	1
I:=bus	0	0	1	0	1	0	0	0	0	0	0	0	1	1
P(I):=PC	1	1	0	0	0	0	0	0	0	0	0	1	1	1
bus:=s+1	0	1	0	0	0	0	0	0	0	0	1	0	1	1
s:=bus	0	0	0	1	0	0	0	0	0	0	0	0	1	1
bus:=PC-1	1	1	0	0	0	0	0	1	1	1	0	0	1	1
P(I):-d	0	0	0	0	0	0	0	0	0	0	0	1	0	1
bus:=PC+1	1	1	0	0	0	0	0	0	0	0	1	0	1	1
Al:=d	0	0	1	1	0	0	0	0	0	0	0	0	0	1
bus:=B	0	0	0	0	0	0	0	0	0	0	0	0	1	1
bus:=P(I)	0	0	0	0	0	1	1	1	0	0	0	0	1	1
Al:=bus	0	0	1	1	0	0	0	0	0	0	0	0	1	1
bus:=Al	0	0	0	0	0	0	1	1	0	0	0	0	1	1
bus:=A-1	1	0	0	0	0	0	0	1	1	1	0	0	1	1
A:=bus	0	0	0	1	1	0	0	0	0	0	0	0	1	1
bus:=A+Al	1	0	0	0	0	0	0	0	0	0	0	1	1	1
NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 2.- Valores que deben tomar las diferentes variables de la unidad de tratamiento, de acuerdo a la microinstrucción.

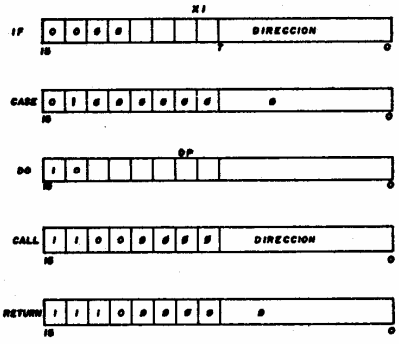

Fig. 9.- Formato delas microinstrucciones de la unidad de control

ta dirección apuntará la dirección del microprograma donde empieza la operación en cuestión. La memoria decodificadora es el elemento que realiza el CASE.

A continuación se indica la dirección y el contenido de las celdas usadas de la memoria decodificadora.

Dirección	Contenido
6	2B (LIT)
7	3F (CAR)
8	33 (ALM)
9	71 (LLA)
A	4B (INS)
B	51 (SAL)
C	55 (SAC)
D	89 (OPR)
E	60 (IN)
F	68 (OUT)
100	8C (RETURN)
101	9F (P(S)=-P(S))
102	B0 (P(S)=P(S)+P(S+1))
103	B3 (P(S)=P(S)-P(S+1))
107	C4 (P(S)=ODD(P(S)))
108	D1 (P(S)=ORD(P(S)-P(S+1)))
109	D7 (P(S)=ORD(P(S)<>P(S+1)))
10A	DD (P(S)=ORD(P(S)<P(S+1)))
10B	E3 (P(S)=ORD(P(S)>=P(S+1)))
10C	E9 (P(S)=ORD(P(S)>P(S+1)))
10D	EF (P(S)=ORD(P(S)<=P(S+1)))

2.5.4 MODULO DEL SECUENCIADOR

El módulo del secuenciador, es el elemento encargado de proporcionar la dirección futura del microprograma. Se compone de un adicionador, una pila a tres niveles, un multiplexor, un registro y una salida tri-estado. La figura 10 muestra el equema general del circuito comercial 74S482. Se utilizaron dos de estos circuitos en cascada.

2.5.5 MODULO GENERADOR DE VARIABLES NULAS

El módulo generador de variables nulas, tiene como misión enviar a la unidad de tratamiento las señales correspondientes a la instrucción no-operación. Esto se debe producir siempre y cuando no se ejecute en la unidad de comando una microinstrucción DO. En nuestro caso las señales de comando para la instrucción no-operación son ceros, que hemos implementado con un conjunto de compuertas AND.

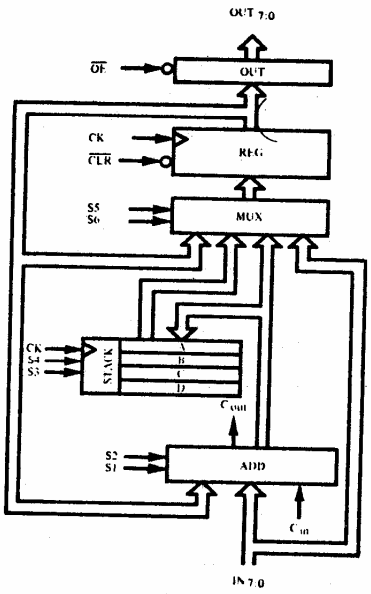

Fig. 10.- Esquema general del secuenciador.

0	0	0	#
0	0	1	=
0	1	0	<>
0	1	1	<
1	0	0	≥
1	0	1	>
1	1	0	≤
1	1	1	A

Fig. 11.- Codificación de las variables de test.

Luego de la definición de todos los elementos de la unidad de control, podemos codificar las microinstrucciones y asignarles una dirección. Ver las dos columnas de la derecha del fragmento del microprograma anteriormente descrito.

Con el objeto de dejar claros algunos conceptos, presentamos un ejemplo completo.

3. EJEMPLO COMPLETO

El programa escogido fue el cálculo del factorial, utilizando un procedimiento para efectuar las multiplicaciones.

3.1 PROGRAMA CON ERRORES SINTACTICOS

Se puede apreciar la calidad del analizador sintáctico, que desde nuestro punto de vista - es eficaz.

```
CONST m=5;
VAR x,y,p,f,f1;
PROCEDURE MULTI;
  VAR z;

  BEGIN
    z:=0;
    WHILE y ) 0
    BEGIN
          '18 Se espera un do
      z=z + x;
        '13 Se espera el operador de asignacion
        '24 Una expresion no puede empezar con este simbolo
      y:=y - 1;
    END;
    p:=z;
  END
 /
BEGIN
     ' 5 Falta una coma o un punto y coma
  x:=m;
   '11 Identificador no declarado
  f:=1;
  while f1 ) 1 do
  begin
    x:=f;
    y:=f1;
    CALL DIVI;
             '11 Identificador no declarado
    f:=p;
    f1:=f1-1;
  end;
END.
```

3.2 PROGRAMA SIN ERRORES SINTACTICOS

```
CONST m=5;
VAR x,y,p,f,f1;
PROCEDURE MULTI;
  VAR z;

  BEGIN
    z:=0;
    WHILE y > 0 DO
    BEGIN
      z:=z + x;
      y:=y - 1;
    END;
    p:=z;
  END;

BEGIN
  f1:=m;
  f:=1;
  while f1 > 1 do
  begin
    x:=f;
    y:=f1;
    CALL MULTI;
    f:=p;
    f1:=f1-1;
  end;
END.
```

3.3 CODIGO MNEMOTECNICO Y TABLA DE IDENTIFICADORES

*** CODIGO DEL PROGRAMA ***

0	SAL	0	21
1	SAL	0	2
**********			
2	INS	0	4
3	LIT	0	0
4	ALM	0	3
5	CAR	1	4
6	LIT	0	0
7	OPR	0	12
8	SAC	0	18
9	CAR	0	3
10	CAR	1	3
11	OPR	0	2
12	ALM	0	3
13	CAR	1	4
14	LIT	0	1
15	OPR	0	3
16	ALM	1	4
17	SAL	0	5
18	CAR	0	3
19	ALM	1	5
20	OPR	0	0
**********			
21	INS	0	8
22	LIT	0	5
23	ALM	0	7
24	LIT	0	1
25	ALM	0	6
26	CAR	0	7
27	LIT	0	1
28	OPR	0	12
29	SAC	0	42
30	CAR	0	6
31	ALM	0	3
32	CAR	0	7
33	ALM	0	4
34	LLA	0	2
35	CAR	0	5
36	ALM	0	6
37	CAR	0	7
38	LIT	0	1
39	OPR	0	3
40	ALM	0	7
41	SAL	0	26
42	OPR	0	0

TABLA DE IDENTIFICADORES

0	CONSTANTE	f1	0	
1	CONSTANTE	m	5	
2	VARIABLE	x	0	3
3	VARIABLE	y	0	4
4	VARIABLE	p	0	5
5	VARIABLE	f	0	6
6	VARIABLE	f1	0	7
7	PROCEDIMIENTO	MULTI	0	2
8	VARIABLE	z	1	3

3.4 CODIGO OBJETO EN DECIMAL Y HEXADECIMAL

DD	CD	DH	CH
0	176	00000000	000000B0
1	21	00000001	00000015
2	176	00000002	000000B0
3	2	00000003	00000004
4	160	00000004	000000A0
5	4	00000005	00000004
6	96	00000006	00000060
7	0	00000007	00000000
8	128	00000008	00000080
9	3	00000009	00000003
10	113	0000000A	00000071
11	4	0000000B	00000004
12	96	0000000C	00000060
13	0	0000000D	00000000
14	208	0000000E	000000D0
15	12	0000000F	0000000C
16	192	00000010	000000C0
17	18	00000011	00000024
18	112	00000012	00000070
19	3	00000013	00000003
20	113	00000014	00000071
21	3	00000015	00000003
22	208	00000016	000000D0
23	2	00000017	00000002
24	128	00000018	00000080
25	3	00000019	00000003
26	113	0000001A	00000071
27	4	0000001B	00000004
28	96	0000001C	00000060
29	1	0000001D	00000001
30	208	0000001E	000000D0
31	3	0000001F	00000003
32	129	00000020	00000081
33	4	00000021	00000004
34	176	00000022	000000B0
35	5	00000023	0000000A
36	112	00000024	00000070
37	3	00000025	00000003
38	129	00000026	00000081
39	5	00000027	00000005
40	208	00000028	000000D0
41	0	00000029	00000000
42	160	0000002A	000000A0
43	8	0000002B	00000008
44	96	0000002C	00000060
45	5	0000002D	00000005
46	128	0000002E	00000080
47	7	0000002F	00000007
48	96	00000030	00000060
49	1	00000031	00000001

50	128	00000032	00000080
51	6	00000033	00000006
52	112	00000034	00000070
53	7	00000035	00000007
54	96	00000036	00000060
55	1	00000037	00000001
56	208	00000038	000000D0
57	12	00000039	0000000C
58	192	0000003A	000000C0
59	42	0000003B	00000054
60	112	0000003C	00000070
61	6	0000003D	00000006
62	128	0000003E	00000080
63	3	0000003F	00000003
64	112	00000040	00000070
65	7	00000041	00000007
66	128	00000042	00000080
67	4	00000043	00000004
68	144	00000044	00000090
69	2	00000045	00000002
70	112	00000046	00000070
71	5	00000047	00000005
72	128	00000048	00000080
73	6	00000049	00000006
74	112	0000004A	00000070
75	7	0000004B	00000007
76	96	0000004C	00000060
77	1	0000004D	00000001
78	208	0000004E	000000D0
79	3	0000004F	00000003
80	128	00000050	00000080
81	7	00000051	00000007
82	176	00000052	000000B0
83	26	00000053	00000034
84	208	00000054	000000D0
85	0	00000055	00000000

DD= Direccion en decimal.
CD= Contenido en decimal.
DH= Direccion en hexadecima
CH= Contenido en hexadecimal.

3.5 VALORES QUE TOMA LA CIMA DE LA PILA PROPORCIONADOS POR EL PROGRAMA INTERPRETE.

```
 5          0
 1         20
 1         20
 5          3
 0         20
 1          3
 4          0
 2         20
 3          2
 3         40
 2          1
 4         60
 1          0
 5         60
 0         60
 5          2
 5         60
 4          2
 5          0
 4         60
 0          1
 5        120
 3          0
10        120
 2        120
15          1
 1
20        *** FIN DE LA EXECUTION *
```

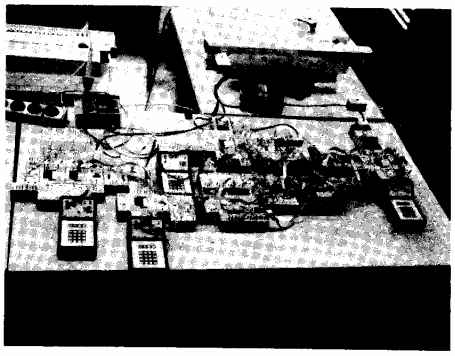

Foto 1.- Montaje final.

COMENTARIOS Y CONCLUSIONES

Hemos escogido tanto el compilador como el procesador con fines didácticos, muchas veces a expensas de otros factores, tales como la rapidez de ejecución y el número de circuitos utilizados. El montaje ha permitido visualizar los diferentes bloques funcionales y, de esta manera, seguir el camino de la información a través de los diferentes bloques.

En el desarrollo del montaje hemos tenido mucha suerte de trabajar con "logidules", dispositivos desarrollados en la Escuela Politécnica Federal de Lausanne, que permiten trabajar de una manera más cómoda, rápida y segura. Dichas cajas negras indican las variables de entrada y salida, y el estudiante no se preocupa ni de la polarización ni de la distribución de los pines. Lamentablemente el costo de los "logidules" es demasiado elevado, se estima que el costo total del material utilizado en este proyecto alcanza a la suma de tres mil dólares.

Para evitarnos el trabajo manual de llenar la memoria con el código del programa a ejecutarse, una buena solución sería la creación de una interface entre el computador que hace la compilación y nuestra máquina PLO.

Como sugerencia, se debería enriquecer el lenguaje PLO en su parte de estructura de datos.

No podría terminar este trabajo, sin antes dejar constancia de mi agradecimiento a ese magnífico equipo, que conforma el Laboratorio de Sistemas Lógicos de la Escuela Politécnica Federal de Lausanne-Suiza, dirigido por el Profesor Daniel Mange, por la decidida colaboración.

BIBLIOGRAFIA

BERNARD, MANGE, STAUFFER, Catalogue Logidules, L.S.L., Edition 1983, EP.F.L.

TEXAS INSTRUMENTS, Th TTL Data Book for Design Engineers.

D. MANGE, E. SANCHEZ, A. STAUFFER, Système

Longique Programmés.

N. WIRTH, Algorithms + data Structures = Programs, Prentice-Hall.

ESQUETINI, CESAR. Nació en Quito, el 3 de Enero de 1957. Recibió el título de Ingeniero en Electrónica y Telecomunicaciones en 1981 de la Escuela Politécnica Nacional. Becado por el Gobierno Suizo en el período 1982-1984, para realizar estudios de post-grado en Informática y Computación, equivalente al Master americano. Asistente de la Cátedra de Sistemas Lógicos Programados en la Escuela Politécnica Federal de Lausanne-Suiza. Actualmente es Profesor del Instituto de Informática y Computación de la E.P.N.