PROBADOR DE MICRO-COMPUTADORAS

Rodríguez Rossini G. Valencia California N.

Depto. de Diseño - Div. de Informática
Inst. Nal. de Cardiología "Ignacio Chavez" - México D. F.

RESUMEN

Debido a que las micro-computadoras cada vez ofrecen mejores posibilidades de desarrollo de productos, se describe una técnica para probarlas a traves de un "Emulador" del micro-procesador Z-80.

DESCRIPCION GENERAL

Los avances tecnológicos y la competencia cada vez mayor en el mundo, está generando demanda de productos nuevos para; bajar costos, incrementar la productividad y mejorar la calidad de los pruductos [1].

El desarrollo de productos basados en micro-procesadores (uP) requiere de una micro-computadora (MC) configurada apropiadamente [2]. Si se cuenta con una computadora comercial (construída y probada) este requerimiento no significa ningun problema, pero si hay necesidad de hacerla y producirla ayuda significativamente contar con herramientas apropiadas para tal trabajo. Otra necesidad de los productos basados en uP. es la reparación, que también requiere de herramientas apropiadas.

En este trabajo se discute una técnica basada en el emulador Z-80 controlado desde el lenguaje "PASCAL" para ayudar en el diseño, la producción y el mantenimiento de productos basados en el uP Z-80.

Una MC está formada básicamente como muestra la fig. 1 por: un uP, la memoria de programas (ROM Read Only Memory), la memoria de lectura/escritura (RAM Random Acces Memory) y los periféricos que use el sistema (terminal, impresora, discos etc.) dependiendo

REV.MEX.ING.BIOMED. 6 (2) 1985

de la aplicación.

Desde este punto de vista se requieren tantas pruebas como bloques tenga el sistema específico de que se trate. En este caso cómo se discuten pruebas generales, a la memoria ROM no se le puede generar una prueba especial, debido a que no se conocen los códigos de máquina del programa de la aplicación específica para compararlos, por lo cual se opta por la alternativa de sustituírla por circuitos de memoria RAM y tratarla como memoria RAM, hecho que también garantiza una buena prueba. El mismo razonamiento se aplica para los periféricos ya que son completamente dependientes de la aplicación específica, para ellos se hacen pruebas que manejen puerto por puerto que aunque resulta más laborioso de aplicar que una prueba para un periférico específico puede probar cualquier periférico.

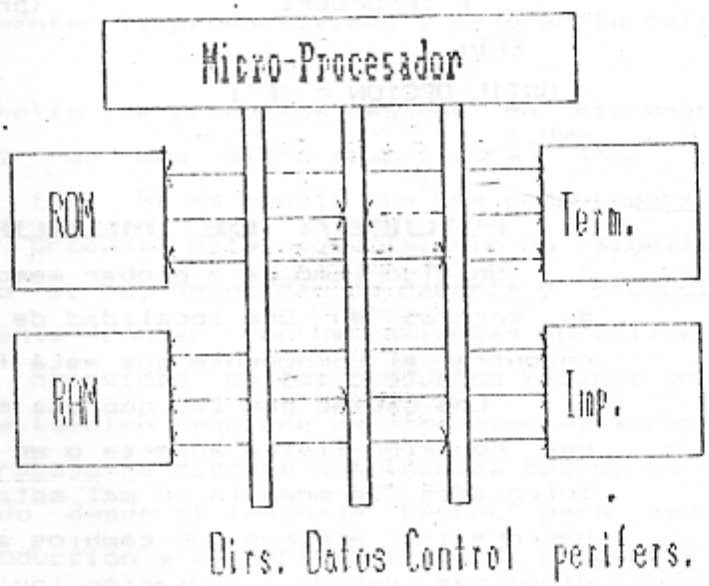

FIG. 5.1.1 DIAGRAMA A BLOQUES DE UNA MC.

Por las razones anteriores se arregla una estructura de pruebas de la siguiente manera:

```
PROBADOR DE MC-
  Prueba de Memorias
  Prueba de Puertos
  Fin
```

Que se realiza con el siguiente programa:

```
PROGRAM PMICRO;
VAR  OPCION:CHAR;
BEGIN
  REPEAT
    WRITELN;                    {pone mensaje de opción}
    WRITELN('PMICRO: Memoria Puerto Fin ');
    READ(KEYBOAD,OPCION);       {lee la opción}
    CASE OPCION OF
      'M':PRUMEM;                 {prueba memoria}
      'P':PRUPUER;                {prueba puerto}
    END;
  UNTIL OPCION = 'F';
END.
```

PRUEBA DE MEMORIAS

Un algoritmo para probar memorias debe indicar la presencia de errores en una localidad de memoria dada [2], y ayudar a encontrar el componente que está funcionando mal.

Las causas por las que una memoria puede funcionar mal son una conexión física abierta o en corto circuito, los circuitos integrados de memoria en mal estado (errores de circuitos (hard errors)) y errores por cambios aleatorios de estado en memorias dinámicas, debido a radiación ionizante del empaque de cerámica o plástico (soft errors).

Para probar las conexiones físicas de las líneas se generan señales diferentes en cada línea, de tal manera que se pueda apreciar tanto la continuidad como los cortos entre ellas, por

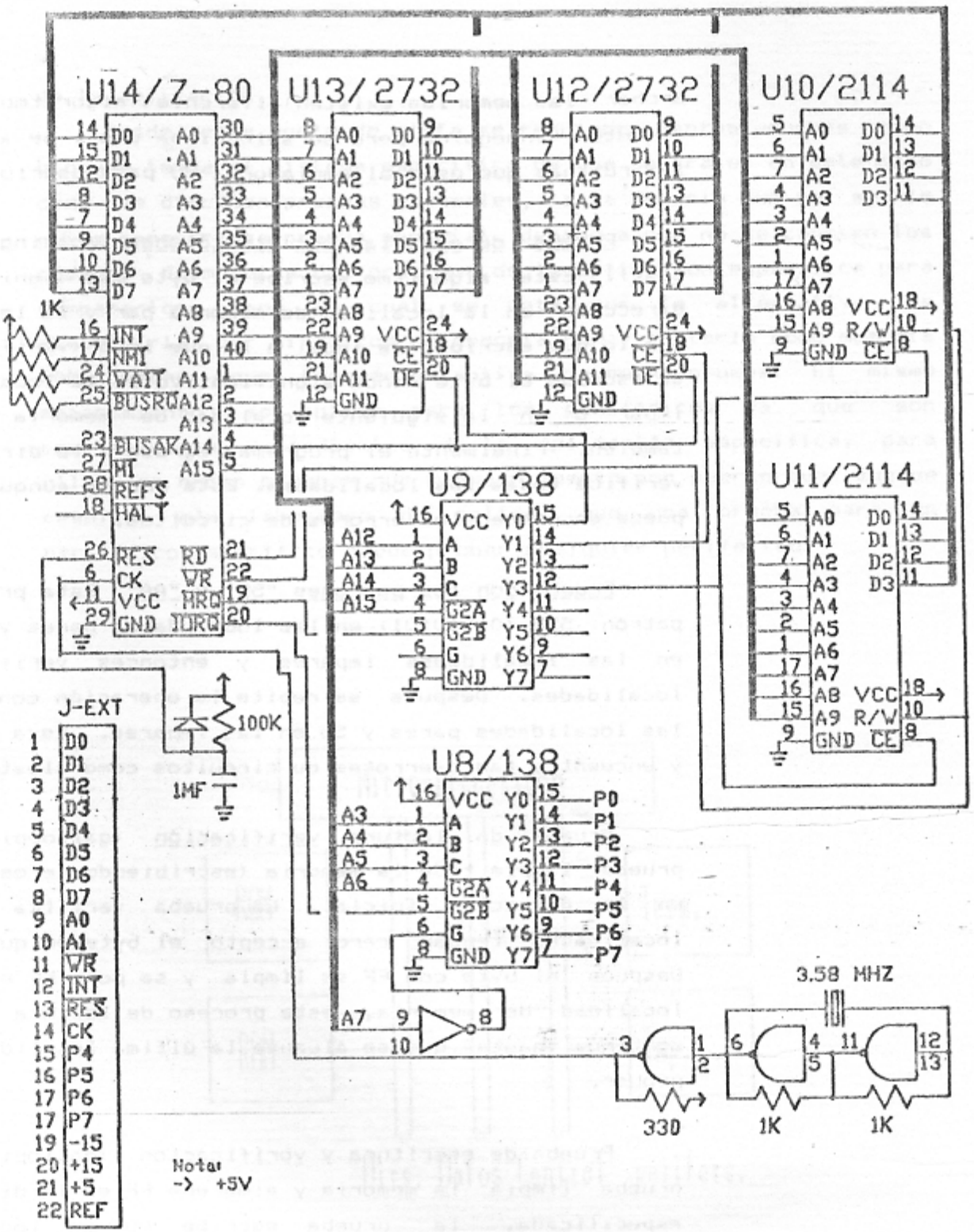

FIG. 6 DIAGRAMA 1/2 DE LA MC Z80M

probar las memorias existen diferentes algoritmos [2], que se discuten enseguida pero se selecciona uno y se arreglo junto con las rutinas que usan al emulador Z-80 para usarlo.

Prueba de memorias por direcciones (Wolking address memory test). Este algoritmo escribe el byte más significativo de la dirección en la localidad de memoria par y lo lee para verificar que lo que escribió es igual a lo que leyó (verificar), enseguida se escribe el byte menos significativo en la localidad de memoria impar o en la siguiente localidad de memoria y se verifica también. Finalmente el programa regresa a la dirección inicial y verifica todas las localidades. Esta prueba aunque es rápida sólo puede encontrar los errores de circuitos.

Prueba con los patrones "55" y "AA". Esta prueba almacena el patrón 55H (01010101) en las localidades pares y AAH (10101010) en las localidades impares y entonces verifica todas las localidades. Después se repite la operación con el patrón AA en las localidades pares y 55 en las impares. Esta prueba es rápida y encuentra tanto errores de circuitos como aleatorios.

Prueba de lectura verificación (galloping-read). Esta prueba limpia toda la memoria (escribiendo ceros) y almacena FFH en la dirección inicial. La prueba verifica que todas las localidades tengan cero excepto el byte en que se puso FF. Después el byte con FF se limpia y se pone FF en la siguiente localidad de memoria, este proceso de lectura y verificación continúa hasta que se alcanza la última localidad de memoria a probar.

Prueba de escritura y verificación (galloping-write). Esta prueba limpia la memoria y almacena FF en la dirección inicial especificada, la prueba escribe 00 en todas las otras localidades, verificando la escritura, después se limpia la localidad con FF y se mueve a la siguiente localidad de memoria. La escritura y verificación continúa hasta que se recorren todas las localidades de memoria. Estas pruebas son excelentes para

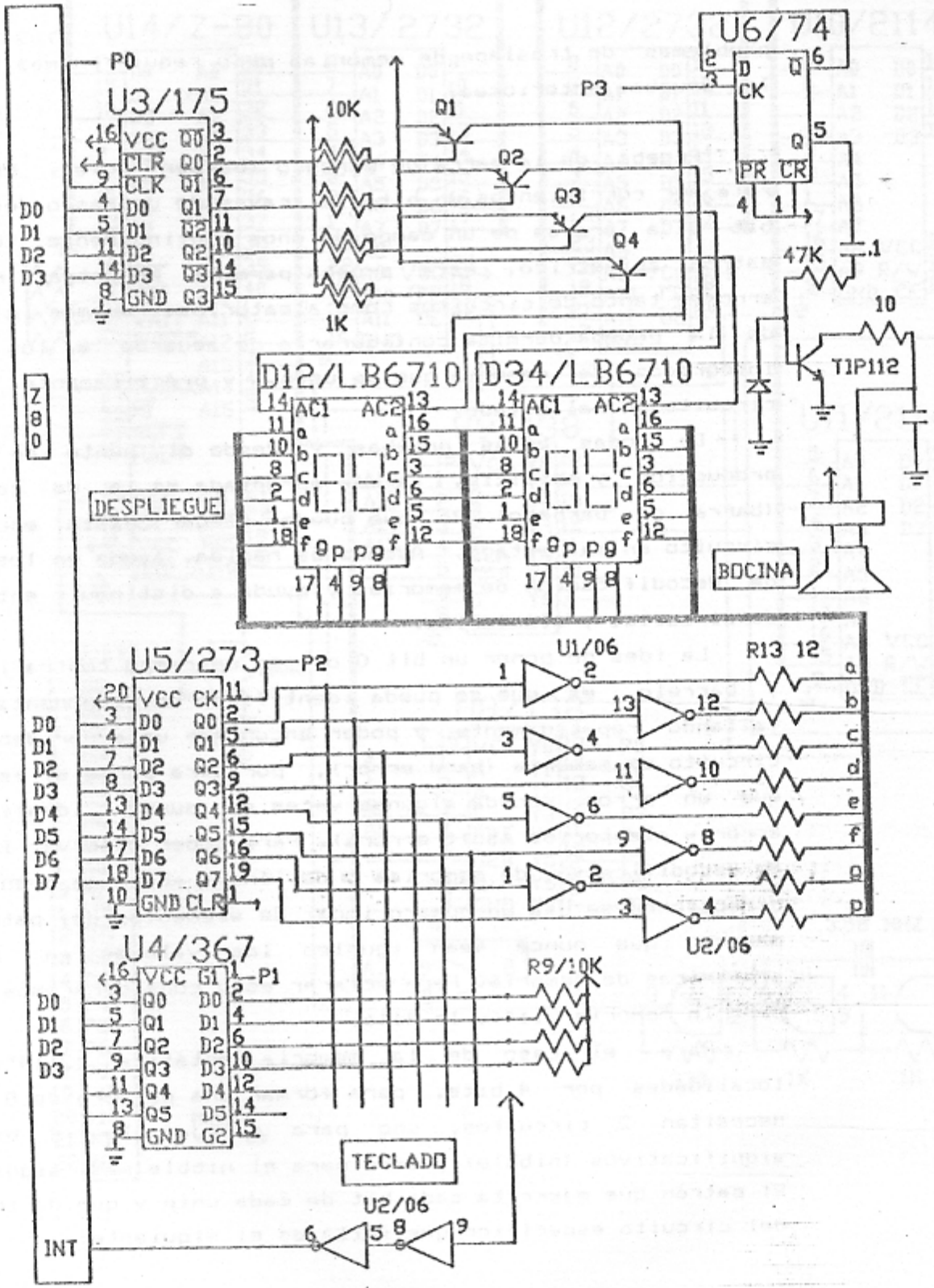

FIG 7 DIAGRAMA 2/2 DE LA MC Z80M

problemas de traslape de memorias pero requieren más tiempo que las pruebas anteriores.

Prueba de la barra de barbero (barber pole). Esta prueba rota por corrimientos un bit 1 a través de un campo de ceros y un bit 0 a través de un campo de unos continuamente formando un patrón asimétrico, esta prueba permite encontrar rápidamente errores tanto de circuitos como aleatorios, además la estructura de la prueba permite configurarla de acuerdo a los circuitos integrados de memoria que se usen, y prácticamente llegar al circuito en mal estado.

De todas estas pruebas y desde el punto de vista de producción y servicio, la más apropiada es la de corrimientos (barra de barbero) ya que puede llegar hasta encontrar el circuito en mal estado. Además es rápida, ayuda en los problemas de decodificación de memorias y ayuda a distinguir entre errores de circuitos y aleatorios.

La idea de poner un bit 0 o 1 en un campo contrario de bits y barrelo, es que se pueda identificar un componente que esté falIando repetidamente y poder encontrar un error debido a un circuito de memoria (hard error), por otra parte si se encuentra que un error sucede algunas veces se pueden identificar los errores aleatorios (soft errors). Para poder observar los errores de decodificación de memorias o de líneas abiertas o en corto de direcciones se usa un número impar de elementos del patrón de tal manera que nunca sean iguales los valores en posiciones simétricas de memoria. Para aclarar esto consideraremos los casos para la memoria 2114 y la 4116.

Para el caso de la memoria estática "2114" de 1024 localidades por 4 bits, para formar una palabra de 8 bits se necesitan 2 circuitos, uno para manejar los 4 bits menos significativos (nibble) y otro para el nibble más significativo. El patrón que ejercita cada bit de cada chip y que da información del circuito específico que falla es el siguiente:

Binario	Hexadecimal
0000 0000	00
0001 0001	11
0010 0010	22
0100 0100	44
1000 1000	88
1110 1110	EE
1101 1101	DD
1011 1011	BB
0111 0111	77

TABLA 5.1 Patrón de corrimientos para probar memorias de 8 bits - organizadas por nibbles, como la "2114".

Como muestra la tabla 5.1.1 se tienen 9 elementos (tamaño = 9) en el patrón, que se almacenan continuamente hasta alcanzar la última localidad (1024).

Para el caso de la memoria dinámica 4116 se usa el mismo razonamiento, excepto que se rotan los bits en un campo de 8 bits, como se muestra en la siguiente tabla.

Binario	Hexadecimel
0000 0001	00
0000 0010	01
0000 0010	02
0000 0100	04
0000 1000	08
0001 0000	10
0010 0000	20
0100 0000	40
1000 0000	80
1111 1110	FE
1111 1101	FD
1111 1011	FB
1111 0111	F7
1110 1111	EF
1101 1111	DF
1011 1111	BF
0111 1111	7F

TABLA 5.1.2 Patrón de corrimientos para probar memorias de 8 bits por palabra, organizadas por un bit, - como la "4116".

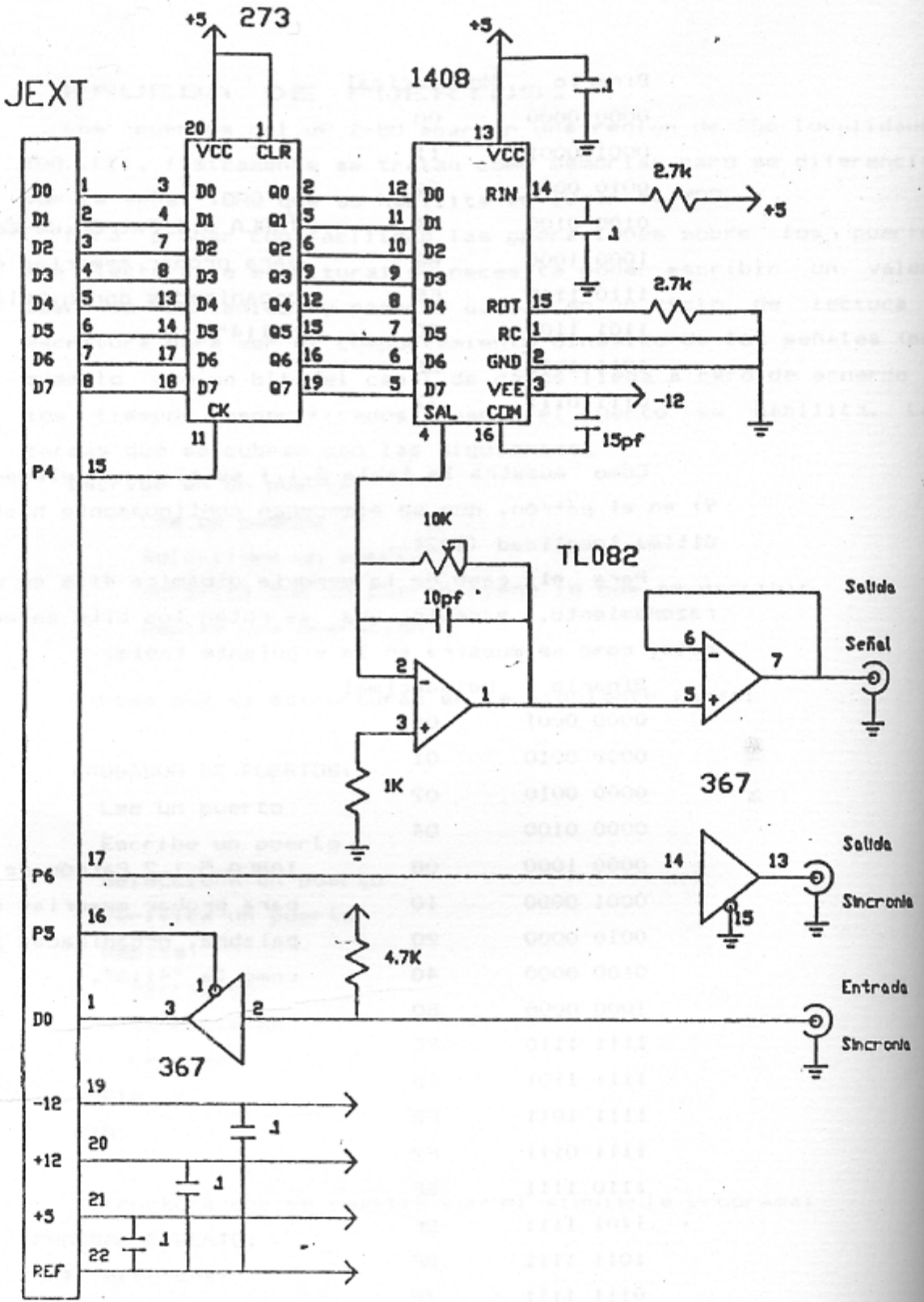

FIG 6 DIAGRAMA DEL CONVERTIDOR DIGITAL ANALOGICO

SEP MEX.ING.BIOMED. 6 (2) 1985

El tamaño del patrón de la tabla es de 17 elementos, que se almacenan continuamente hasta alcanzar las 16 K palabras del arreglo de la memoria "4116".

Para mostrar como se realizó el programa (procedimiento en Pascal) de prueba de memorias, se muestra primero la secuencia de acciones y después los aspectos generales de programación. Para generalizar la prueba con diferentes memorias se usa la tabla 5.1.2 como patrón de prueba.

El algoritmo de prueba se resume en la siguiente secuencia:

PRUEBA DE MEMORIAS POR CORRIMIENTOS

1. Solicitar la dirección inicial a probar. (DIRINI)
2. Solicitar la direccion final a probar (DIRFIN)
3. Repite
   - 3.1 Escribe el patrón de DIRINI a DIRFIN
   - 3.2 Verifica que el patrón este almacenado
   - 3.3 Rota el patrón un lugar

   Hasta que el no. de rotaciones = tamaño del pat.
4. Fin

Que se traduce en el siguiente programa:

```
PROCEDURE PRUMEM;
CONST     TAMANO=17;
VAR       PATRON:PACKED ARRAY[0..ESPATRON] OF BYTE;
          OPMEM:CHAR;
          DIRINI,DIRFIN,VECES:INTEGER;

BEGIN
  WRITE('Dirección inicial ? ');
  READLN(DIRINI);
  WRITE('Dirección final   ? ');
  READLN(DIRFIN);
  FOR VECES:=1 TO TAMANO DO BEGIN
    ESCPATRON;                  {escribe el patrón}
    VERPATRON;                  {verifica el patrón}
    ROTPATRON;                  {rota el patrón}
  END;
END;
```

REV. MEX. ING. BIOMED. 6 (2) 1985

PRUEBA DE PUERTOS.

Los puertos del uP Z-80 abarcan una región de 256 localidades (00..FF), físicamente se tratan como memorias pero se diferencian por la señal IORQ que se habilita en lugar de MRQ.

Para probar con facilidad las operaciones sobre los puertos (de lectura o escritura) se necesita poder escribir un valor, leer su contenido y poderse quedar en un ciclo de lectura o escritura para ver el comportamiento dinámico de las señales (por ejemplo que un bit del canal de datos llega a cero de acuerdo a los tiempos especificados) cuando el puerto se habilita. Las tareas que se cubren son las siguientes:

Escribe en un puerto
Lee un puerto
Selecciona un puerto
Verifica que un puerto tiene lo que se escribió
Repite una operación

Tareas que se estructuran en la siguiente forma:

```
PROBADOR DE PUERTOS:
  Lee un puerto
  Escribe un puerto
  Selecciona un puerto
  Verifica un puerto
  Repite
    Barriendo
    Escribiendo
    Leyendo
  Fin
FIN
```

Estructura que se realiza con el siguiente programa:

```
PROGRAM PPUERTO;
(*$I B:BASES*)
CONST   HABAD=$E0AC;     {L habilita el acceso directo}
        DESAD=$E0AD;     {L deshabilita el acceso directo}
```

```
          RFSH =$EOAD;     {L genera la senal de refresco}
          DIRME=$EOAC;     {E pone la dirección menos significativa}
          DIRMA=$EOAD;     {E pone la dirección mas significativa}
          REGDAT=$EOAE;    {L/E el canal de datos}
          REGEST=$EOAF;    {L/E el registro de estado}
VAR       OP:CHAR;
          FIN:BOOLEAN;
          PUERTO:BYTE;
          BAS:BYTE;

PROCEDURE ESCPUERTO(VALOR:BYTE);  {escribe un puerto}
BEGIN
  MEM[DIRME]:=PUERTO;
  MEM[REGEST]:=$41;
  MEM[REGDAT]:=VALOR;
  MEM[REGEST]:=$C1;
END;

PROCEDURE LEEPUERTO(VAR VALOR:BYTE);{lee un puerto}
BEGIN
  MEM[DIRME]:=PUERTO;
  MEM[REGEST]:=$41;
  VALOR:=MEM[REGDAT];
  MEM[REGEST]:=$C1;
END;

PROCEDURE TRAEPUERTO;             {trae el puerto a probar}
VAR       E:INTEGER;
BEGIN
  WRITELN;
  WRITE('  Puerto ? ');
  LEHEX(E);
  PUERTO:=LO(E);
END;

PROCEDURE ESCRIBE;                {escribe en un puerto}
VAR       VAL:INTEGER; V:BYTE; VH:THEX;
BEGIN
```

```
    WRITE('  Valor ? ');
    LEHEX(VAL);
    WHILE NOT ESC DO BEGIN
      V:=LO(VAL);
      ESCPUERTO(V);
      LEEPUERTO(V);
      BYTEAHEX(V,VH);
      WRITELN('  ',VH);
      WRITE('  Valor ? ');
      LEHEX(VAL);
    END;
  END;

  PROCEDURE LEE;                          {lee un puerto}
  VAR     VH:THEX; V:BYTE;
  BEGIN
    REPEAT
      LEEPUERTO(V);
      BYTEAHEX(V,VH);
      WRITELN(VH);
    UNTIL KEYPRESSED;
  END;

  PROCEDURE REPITE;                       {repite una opción}
  VAR     O:CHAR; VAL:INTEGER; V:BYTE; VH:THEX;
  BEGIN
    REPEAT
      WRITE('  Barriendo Escribiendo Leyendo ');
      O:=LECAR(['B','b','E','e','L','l']);
      IF NOT ESC THEN
      CASE O OF
        'B','b':BEGIN
                  WRITE('  ...');
                  REPEAT
                    ESCPUERTO(V);
                    V:=V+1;
                  UNTIL KEYPRESSED;
```

```
                WRITELN;
              END;
        'E','e':BEGIN
                WRITELN;
                WRITE('   Valor ? ');
                LEHEX(VAL);
                WRITE(' ...');
                V:=LO(VAL);
                REPEAT ESCPUERTO(V); UNTIL KEYPRESSED;
                WRITELN;
              END;
        'L','l':BEGIN
                WRITE('...');
                REPEAT LEEPUERTO(V); UNTIL KEYPRESSED;
                BYTEAHEX(V,VH);
                WRITELN('  Val: ',VH);
              END;
    END;
  UNTIL ESC;
END;
PROCEDURE VERIFICA;                    {verifica un puerto}
VAR     ESP,ENC:BYTE; ESPH,ENCH:THEX;
BEGIN
  WRITE(' ...');
  ESP:=0;
  REPEAT
    ESCPUERTO(ESP);
    LEEPUERTO(ENC);
    IF ESP <> ENC THEN BEGIN
      BYTEAHEX(ESP,ESPH);
      BYTEAHEX(ENC,ENCH);
      WRITELN('Esp. ',ESPH:4,'Enc. ',ENCH:4);
    END;
    ESP:=ESP+1;
  UNTIL KEYPRESSED;
END;
```

```
                WRITELN;
              END;
        'E','e':BEGIN
                WRITELN;
                WRITE('   Valor ? ');
                LEHEX(VAL);
                WRITE('  ...');
                V:=LO(VAL);
                REPEAT ESCPUERTO(V); UNTIL KEYPRESSED;
                WRITELN;
              END;
        'L','l':BEGIN
                WRITE('...');
                REPEAT LEEPUERTO(V); UNTIL KEYPRESSED;
                BYTEAHEX(V,VH);
                WRITELN('  Val: ',VH);
              END;
    END;
  UNTIL ESC;
END;
PROCEDURE VERIFICA;                          {verifica un puerto}
VAR     ESP,ENC:BYTE; ESPH,ENCH:THEX;
BEGIN
  WRITE(' ...');
  ESP:=0;
  REPEAT
    ESCPUERTO(ESP);
    LEEPUERTO(ENC);
    IF ESP <> ENC THEN BEGIN
      BYTEAHEX(ESP,ESPH);
      BYTEAHEX(ENC,ENCH);
      WRITELN('Esp. ',ESPH:4,'Enc. ',ENCH:4);
    END;
    ESP:=ESP+1;
  UNTIL KEYPRESSED;
END;
```