Sistemas Digitales

[redacted]

72

"Desarrollo de la Unidad Aritmética
de una minicomputadora digital
de propósitos generales".

Lidia Estela LOISONI

19/4/74

Ing. Jorge Santos

Ing Santos
Ing Arango
Ing Pascual

PROYECTO tema:

"Desarrollo de la Unidad Aritmética
de una minicomputadora digital de propósitos generales"

INDICE

BIBLIOGRAFIA:

Ref.1: Ing. Jorge Santos, "Diseño lógico de un divisor digital rápido". de Ciencia y Técnica, Revista del centro de estudiantes de ingeniería. Vol.129 - Nº 651 - pág 386 a 395

Ref.2: Bell and Newell, "Spring Joint Computer Conference" -

Ref.3: Richards R K, "Arithmetic operation in digital Computer" pagina 165 a 170

Ref.4: Domenico Ferrari, "A Division Method Using a Parallel Multiplier". IEEE Transaction on Electronic Computers. April 1967 - pág 224 a 226.

PROYECTO

tema:

"Desarrollo de la Unidad Aritmética
de una Minicomputadora digital de propósitos generales"

1.-Objetivo:

La idea de desarrollar una minicomputadora digital de propósitos generales está motivada por el deseo de iniciar desde la Universidad un aporte al desarrollo de la tecnología nacional.

Es así como contando con algunos elementos disponibles en el área, tales como circuitos integrados de varias clases (compuertas, sumadores, flip flop, etc.) se pensó en utilizarlos de una manera eficaz en un diseño relativamente elaborado.

El objetivo de este proyecto es especializarse en técnicas de diseño, desarrollo y ejecución de circuitos más ó menos complejos a partir de elementos integrados.

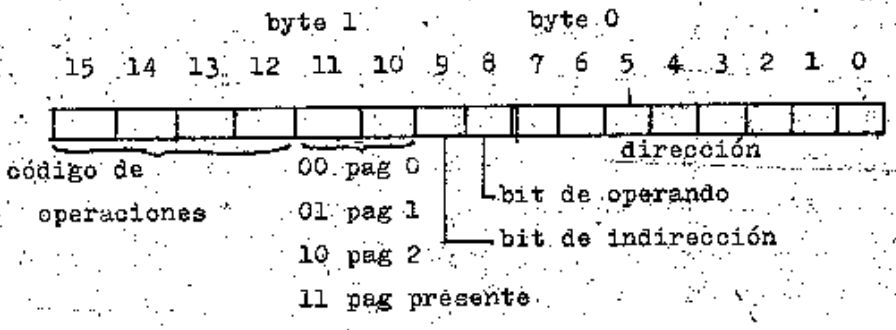
Existen intentos de desarrollo de integración en pequeña y mediana escala en nuestro país, comenzando con el encapsulamiento de circuitos MOS; y el posible prever a mediano plazo la integración vertical de una industria de computación.

Esta minicomputadora que hemos intentado construir será quizá el comienzo de una serie que tenderá a perfeccionarla y lo que hasta ahora hemos hecho no es sino la etapa más elemental. Consta de una Unidad de Control (UC), una Unidad de Entrada y Salida (UES) que controla los respectivos equipos periféricos y una Unidad Aritmética (UA) que realiza directamente las operaciones de suma y resta y mediante programa, otras operaciones como multiplicación y división. Esta última es la que me corresponde desarrollar.

2.- Introducción:

Una somera descripción dela minicomputadora es la siguiente: Se trata de un procesador central (Pc) compuesto por varios registros. El reg. acumulador (A) consta de 16 bits numerados del 0 al 15; al mismo se le concatena un reg. T de un solo bit cuya función es repetir el bit signo y detectar cuándo el operando excede el rango. Para extender el rango del A existe el reg. de extensión (E) que también está formado por 16 bits; su función es ser auxiliar de A y como en aquél sus bits están dispuestos en 2 bytes.

Para calcular la dirección de la instrucción existe un reg. de control de dirección (C) el cual tiene su propio sumador para calcular la dirección de la nueva instrucción. Con ella se accede a la memoria de primer nivel (M) -la cual todavía no está implementada- y se extrae la instrucción que va al reg. P de instrucción presente. Este también tiene 16 bits ordenados en 2 bytes que cumplen las sig. funciones



Como cuando se trata de acceso indirecto, la dirección que figura aquí es la dirección de la dirección del operando; con ella se accede a M y se extrae la dirección del operando colocándose el byte más significativo de ésta en un reg. auxiliar (D), conservándose así en el byte más significativo de P la instrucción que se debe obedecer.

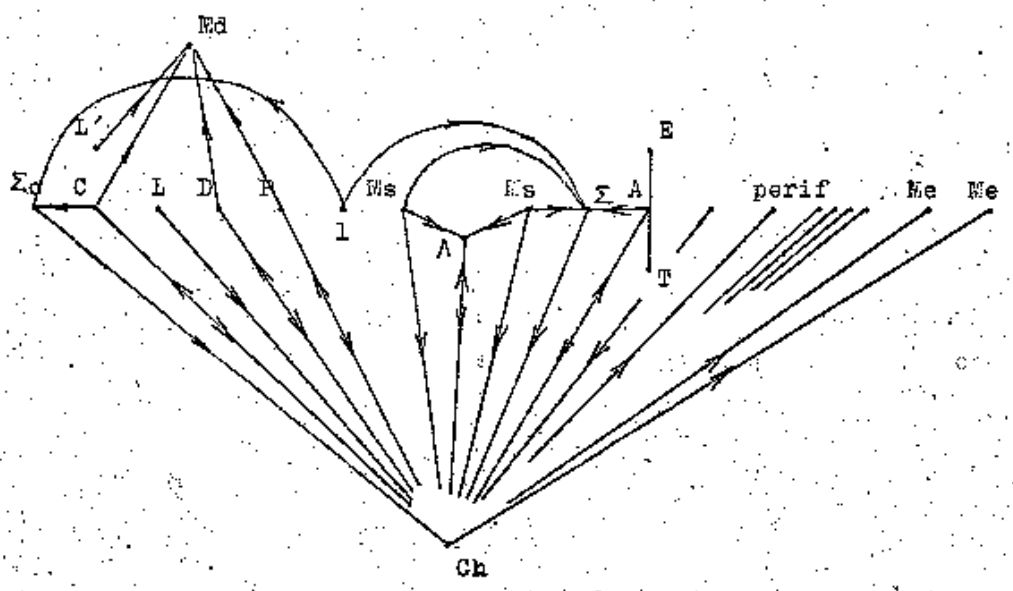
Las operaciones se realizan en el banco sumador que opera A con el reg. de salida de M, el cual cumple las funciones de segundo operando (SO); además permite sumar y restar uno a A y sumar uno a SO para obedecer las instrucciones respectivas.

Tiene capacidad para 16 instrucciones, una de las cuales permite expandirse a 16 microinstrucciones más.

Dispone además de los correspondientes equipos periféricos de entrada y salida.

El flujo de información entre cualesquiera de las partes de la máquina pasa siempre por un canal quehabilita las compuertas correspondientes a manera de árbol Lagrangiano.

El grafo del sistema es el sig.



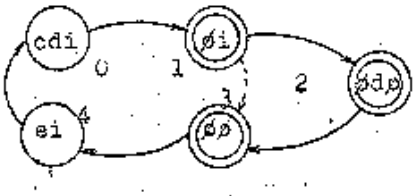
Σc := sumador del reg. C
C := control de dirección
L' := llaves de dirección
L := llaves de información
D := reg. auxiliar de P
P := reg. de instrucción presente
Md := entrada de dirección de M
1 := uno
Ms := reg. de salida de M
A := acumulador
∧ := and
E := Estensión de A
T := reg. para desborde
Me := entrada de M
Σ := sumador general

La consola contiene 16 llaves de dirección y 16 llaves de información que permitirán introducir el primer programa a la máquina. Provisoriamente, como aún no se dispone de M, las llaves de dirección funcionarán como reg. de salida de M.

Cada registro presenta su contenido al exterior mediante luces.

El funcionamiento de la maquina puede realizarse en forma continua, por compases ó por pulsos, accionando los conmutadores correspondiente. Cada vez que se produzca un rebalse en el rango de los números con que trabaja, el procesador deja de funcionar y enciende el indicador correspondiente.

Las fases de obedecimiento son las que a continuación están representadas, siguiendo la notación de Bell y Newell (Ref. 1).



cdi:= cálculo de la dirección de la instrucción
øi := obtener la instr.
ødø:= obtener dirección operan.
øø := obtener operando
ei := ejecutar instr.

donde los doble círculos indican fases de acceso a M.

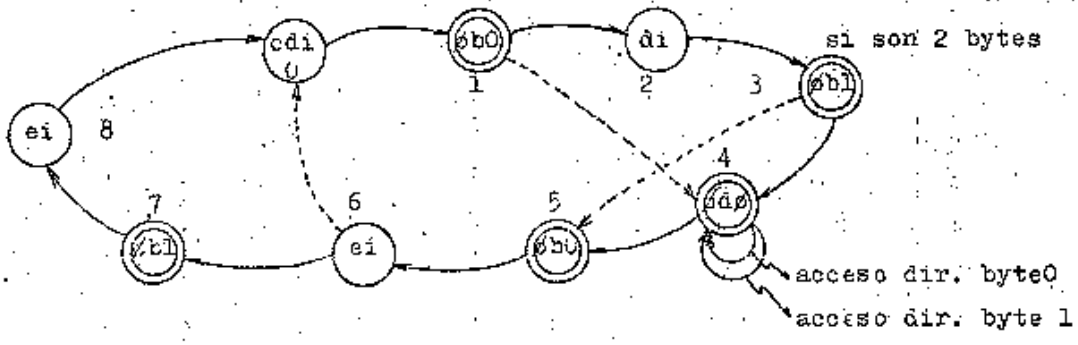
3.- Descripción de las sucesivas alternativas de arquitectura:

Se descutió mucho antes de adoptar el esquema actual y se iban descartando posibilidades a medida que se demostraba su inconveniencia ó su infuncionalidad.

La reseña cronológica en que se aceptaban y rechazaban ideas con el objeto de tender a algo mínimo-dinámico-funcional fue la sig.

En principio se pensó que el acceso por byte permitía una memoria más compacta y además la posibilidad de trabajar con operandos cortos y operandos largos, esto es de uno y dos bytes respectivamente según se desee, requiriendo estos últimos doble acceso a M.

Esto debía obedecer las sig. fases:



alternativa: la dirección del operando puede ser múltiple.

di := decodificación de instrucción

øb0, øb1 := obtención byte 0, obtención byte 1

En definitiva podríamos resumirlo en el sig. cuadro:

long. de intruc.	long. de operando	dirección	fases	accesos a M
doble	simple	directo	0-1-2-3-2-6	3
		indirecto	0-1-2-3-4-4-...-5-6	5
doble	doble	directo	0-1-2-3-5-6-7-8	4
		indirecto	0-1-2-3-4-4-..-5-6-7-8	6
simple	simple doble	no hay dir, la instr. se obedece sobre A	0-1-2-6	1

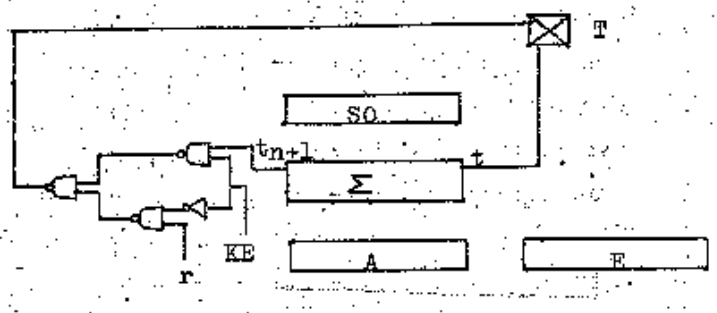
En la instrucción, los bits 12 a 15 del byte 1 indicarían el código de operaciones, el bit 11 es para acceso directo ó indirecto el bit 10 es el bit de página, que será 0 ó 1 según se haga referencia a la pág cero ó a la pág. presente. El bit 8 indicaría si se trata de operandos de uno ó dos bytes. Quedaría un bit que es el 9 con una función no bien determinada, que podría usarse de varias maneras: a) para indicar si se trata de pág. 1 ó 2; b) como quinto bit de instrucción; c) para trabajar con dos acumuladores, en cuyo caso indicaría sobre cuál de ellos se está operando; d) podría también concatenarse a la parte de dirección con lo cual se amplía el rango. Lo más aceptable hasta el momento era la posibilidad a), sobre todo pensando en una máquina que por ser el primer eslabón de una serie permita cambiar la función de este bit e incorporarlo por ej. al código de operaciones de la instrucción sin realizar muchas modificaciones. Además otra ventaja de la posibilidad a) es que de ese modo se puede ahorrar direccionado indirecto y con ello tiempos de acceso a M. La posibilidad c) se descarta por economía en una máquina pequeña. El caso d) en principio queda descartado porque introduce inconvenientes en la programación. En el caso b) son innecesarias por el momento 32 instrucciónes.

Los reg. con que se contaría serían: un reg. de salida de E, que actuaría también como SO, un reg. A y un reg. E, todos ellos de 8 bits. Estarían ligados de tal manera que el SO podría operarse directamente con A ó E, de manera que cuando se quieran operar números de 16 bits se puedan operar los 8 bits menos signifi cati-

vos entre SO y E y acumular el resultado en E; luego mediante otra instrucción se coloca a la salida de M el byte más significativo y se opera con el contenido de A, depositando el resultado en A. Paraello se necesita almacenar el transporte de la primera operación. El transporte que debe existir al iniciarse la operación debe ser un cero si se trata de suma ó un uno si se trata de resta, esto responde a la forma en que está diseñado y construído el sumador.

El programa que deberían cumplir estas operaciones se describe a continuación, acompañado de su respectiva implementación física. Supongamos que z representa la dirección de los 8 bits más significativos y z' la de los 8 menos significativos, descrito en lenguaje PCI (Procesador de conjunto de instrucciones - Ref. 5)

$E \leftarrow M[z_1']$
$E \leftarrow E + M[z_2']$
$A \leftarrow M[z_1]$
$A \leftarrow A + M[z_2]$
$M[z_1] \leftarrow A$
$A \leftarrow E$
$M[z_1'] \leftarrow A$



El diagrama lógico para guardar el transporte debería cumplir la siguiente tabla, a la cual responde el circuito.

t_n	KE	r	t_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

t_n := transporte previo
KE := control de reg. E
r := resta (:=complemento de suma)
t_{n+1} := transporte

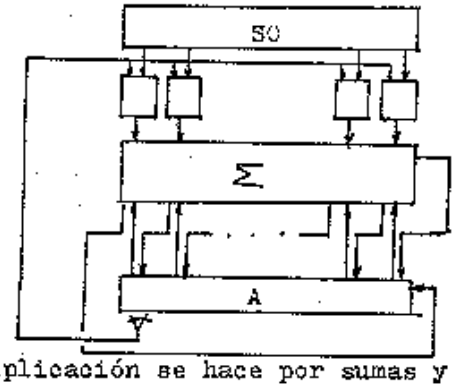
Además debe aparecer una señal que implique un aviso cada vez que el transporte del bit más significativo sea un 1 en el caso de suma ó un 0 en resta, pues esto indicaría que el resultado se ha excedido en el rango.

La tabla a la cual debe adaptarse el circuito es:

t_{n+1}	r	señal
0	0	0
0	1	1
1	0	1
1	1	0

Si se trata de operandos cortos estos deben colocarse directamente en A.

También la multiplicación entre números de 16 bits ofrece grandes inconvenientes puesto que se ha de realizar por partes teniendo reg. de 8 bits. La implementación con la que deberíamos contar para cualquier multiplicación ya sea de operandos cortos ó largos es la sig.



La multiplicación se hace por sumas y desplazamientos sucesivos según que el bit que haya que multiplicar sea un cero ó un uno.

También aquí cabe contemplar la posibilidad de realizar los desplazamientos mediante conexiones cruzadas o bien mediante conexiones directas y reg. circulantes. El primer caso introduciría gran complicación en la lógica combinacional necesaria ya que cuando se trata de sumas ó restas aisladas las conexiones deben ser directas y cuando se trata de sumas o restas como partes de una subrutina de multiplicación deben realizarse cruzadas. Entonces lo más conveniente es la conexión directa, agregando además una microinstrucción al código establecido, que permita realizar desplazamientos a izquierda y derecha.

Volviendo a la multiplicación de operandos largos sobre reg. de 8 bits, debe particionarse en cuatro submultiplicaciones y luego sumar los subproductos desplazados de la siguiente manera: llamaremos M_1 y M_0 al byte más y al menos significativo del multiplicando y m_1 y m_0 al byte más y al menos significativo del multiplicador respectivamente. Entonces se hacen los subproductos cruzados y se suman ($M_1 \times m_0 + M_0 \times m_1$), luego se hace ($M_0 \times m_0$) del cual los 8 bits ménos significativos pasan directamente a formar parte del resultado final y los 8 restantes se suman a los 8 menos signif. de la suma anterior. Los 8 bits menos signif. de esta

nueva suma pasan a formar parte del resultado final, concatenados a los anteriores. Luego realizo el subproducto $M_1 \times m_1$ y los 8 bits menos significativos de este se los sumo a los 8 más significativos de la suma anterior y si hay transporte lo sumo a los 8 bits más significativos del último subproducto, quedando de esta manera 16 bits concatenados a los anteriores, y todos forman el resultado final.

También se contempló la posibilidad de usar un reg. C de control de dirección con un sumador individual ó usar el sumador común para calcular las direcciones. Suponiendo que sea más adecuado usar la UA con un sumador común para todos los reg., aprovechando la propiedad que tienen los sumadores utilizados, de poder operar con 4 reg.: A_1, A_2, B_1, B_2 y las combinaciones: $A_1 \pm B_1$; $A_1 \pm B_2$; $A_2 \pm B_1$; $A_2 \pm B_2$ entre dichos reg., esto implicaría agregar microinstrucciones que permitan adicionar 2 a C y el resultado volver a colocarlo en C. Ello lleva mucha electrónica asociada a la salida del banco sumador. Para superar esto se adopta un sumador específico para C.

Las alternativas que se plantearon respecto de la longitud de los registros y la implementación física ó microprogramada de un multiplicador fueron:

alternativa	nro.de bits por reg.	implementación
1	8	programación
2	8	física
3	16	programación
4	16	física

Analizando cada caso, se ve que la alternativa 1 requiere un programa demasiado extenso para multiplicar operandos largos, que son los que en definitiva más se emplean; la razón es que requiere dividir la operación en 4 submultiplicaciones y luego sumar los resultados parciales desplazados.

También tiene el inconveniente de que es necesario programar operacioneselementales como suma y resta sobre operandos largos.

La alternativa 2 requiere una implementación complicada, la cual no se fustifica si se hace solo para operandos cortos, mientras que para los operandos largos igual debería usarse microprogramación.

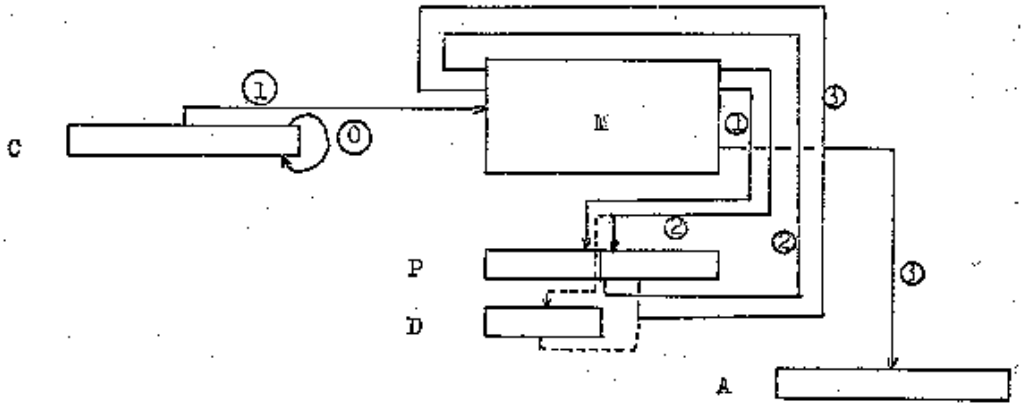
En la alternativa 3 las operaciones elementales como sumas y restas pueden realizarse directamente puesto que se cuenta con reg. de 16 bits y sus 16 sumadores correspondientes; las opera-

ciones de multiplicación y división se microprograman.

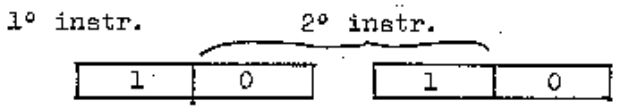
La alternativa 4 permite operar directamente mediante la implementación física pero esto implica una electrónica asociada compleja que rebasaría los límites de nuestro propósito.

En el balance entre la complejidad y el costo de la implementación y la velocidad de las operaciones, elegimos la alternativa 3 como la más conveniente.

La sucesión de accesos a M a que responde la máquina de las características dadas es la sig.

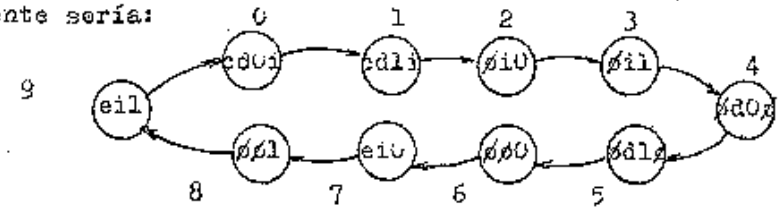


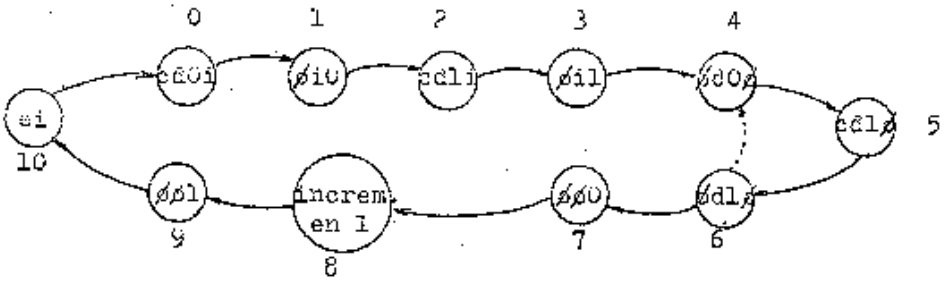
Todo esto se realizó pensando aún en reg. de salida de M de 8 bits puesto que si los reg. fueran de 16 bits y existieran instrucciones cortas y largas puede suceder que luego de una instrucción corta se quiera obtener una larga y en ese caso la instrucción larga esté comprendida entre el byte 0 de una locación y el byte 1 de la siguiente, por ej.



A menos que esto se contemple y cuando se trate de instrucciones cortas se deje uno de los bytes libres.

Si comparamos siempre con un ancho de salida de M de 1 byte, el ciclo que se cumpliría utilizando reg. de 8 y 16 bits respectivamente sería:





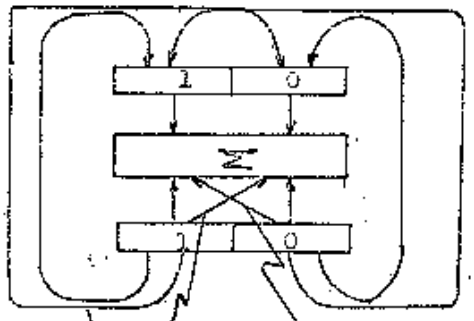
En este segundo caso se necesitaría otro reg. de 16 bits porque el que existe lo tenemos para conservar la dirección del segundo byte, entonces no se puede colocar en él la dirección del primer byte del operando cuando ocurre una indirección.

Para obviar incrementar el número de reg. del procesador se usa el acceso a M mediante 16 bits, esto es con un ancho de salida de M de 2 bytes.

Contamos por lo tanto a esta altura del estudio con un reg. de salida de M, un A, un C, y un P todos ellos de 16 bits y un auxiliar D de 8 bits.

Ocurre que dadas estas condiciones, la lógica combinacional que se requiere, suponiendo que operandos cortos puedan aparecer en bytes pares ó impares del reg. de salida de M y ellos se deban operar con operandos cortos que también pueden estar en bytes pares ó impares de A, esdemasiado complicada sobre todo para operar bytes cruzados, e introduce mucho retardo.

En ese caso en la UA el flujo de información sería:



para operandos cortos que están en el byte 1

para sumar los subproductos resultantes de multiplicar operandos cortos. Pues para mult., los operandos cortos se deben colocar en los bytes0

Esto podría simplificarse agregando un SO distinto del reg. de salida de M paramoder rbicar los operandos cortos en los bytes 0 de ambos reg. y sumar directamente, pero ello necesitaría una lógica para pasar los bytes directos o cruzados del reg. de salida de M al SO/

Para no agregar reg. lo mejor sería microprogramar el cruce de

bytes de manera que si el operando está en un byte y lo deseamos en otro, lo volvemos a introducir a alguna dirección de M en la que pueda ocupar el byte que nos interese y luego lo extraemos nuevamente. Aún más factible y práctico que esto es colocar un reg. de extensión de A (E) de 16 bits, lo cual además simplifica la multiplicación que ahora podría hacerse directamente siendo multiplicando y multiplicador ambos de 16 bits.

Igual, además de esto , la idea de microprogramar el cruce de bytes no es desechable si pensamos en que operandos cortos casi no se usan a no ser para llevar alguna cuenta en lasque en general la magnitud no excede 2^8 y por lo tanto no se reterdaría demasiado el programa. Pero dado el poco uso de operandos cortos y el agregado de instrucciones del tipo salto por memoria que permiten contar el número de veces que se realicen las operaciones, se eliminan de nuestro proyecto, provisoriamente, los operandos cortos conservándose aún el bit 8 por si se quiere posteriormente admitir la opción.

Según como están las cosas hasta ahora, al reg. C hay que sumarle un 2 para calcular la dirección de la instrucción siguiente de un programa, pero aparece un problema cuando la instrucción es de longitud simple y está ubicada en el byte impar, entonces. habría que considerar lo siguiente:

C⟨0⟩= 0 y instr. de doble long. : → C←C + 2

C⟨0⟩= 0 y instr. de simple long.: → C←C + 1

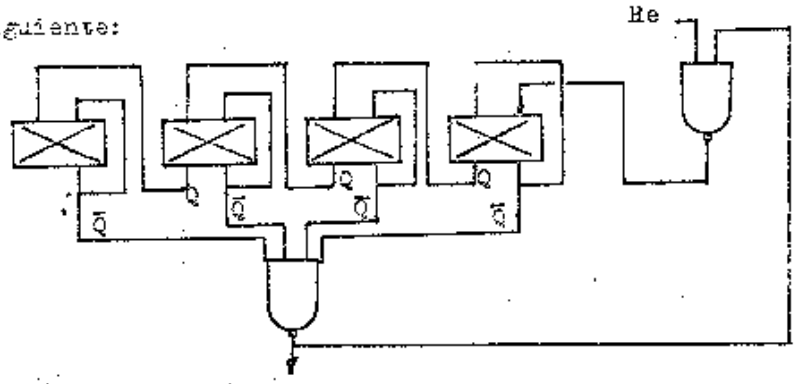
C⟨5⟩= 1 → instr. de long simple.

Por las instrucciones simples dejan la otra mitad de su locación desocupada y por lo tanto debe sumársele siempre un 2.

La máquina trabaja con números positivos y negativos y permite la posibilidad de anular la resta, lo cual reduciría el número de compuertas asociadas al sumador en 3 por bit, y realizarla por programación, pero esto implicaría agregar un reg. 80 distinto del reg. de salida de M pues para convertir un número de positivo a negativo y en lugar de restarlo, sumarle el negativo, hay que: complementarlo, sumarle 1 y colocarlo de nuevo en el reg. Por lo tanto se descartó esta posibilidad.

Otro de los elementos que se descartó en esta construcción y que se había considerado en principio necesario fue un contador que llevara la cuenta del número de veces que se hacen desplazamientos ó que se suman subproductos, por ej. en el control de

una multiplicación. Ello requiere un contador simple como el siguiente:



de todas maneras se decidió no usarlo, para reducir la implementación física de la máquina. Para ello se modificó la subrutina de multiplicación reemplazando el contador por saltos por K; previamente se fija, para cada subrutina, en algún lugar de K el número de veces que debe saltar.

La codificación de las instrucciones es la siguiente:

ca (:=op= 0000) cargar acumulador
ia (:=op= 0001) intersectar A
aa (:=op= 0010) adicionar A
ra (:=op= 0011) restar A
sa (:=op= 0100) salto por A
sk (:=op= 0101) salto por K
ta (:=op= 0110) transferir A
ss (:=op= 0111) salto a subrutina
si (:=op= 1000) salto incondicional
ma (:=op= 1001) multiplicar A
da (:=op= 1010) dividir por A
mi (:=op= 1011) microinstrucciones
   (:=op= 1100) libre
ap (:=op= 1101) arrancar periféricos
te (:=op= 1110) transferencia de entrada
ts (:=op= 1111) transferencia de salida

microinstrucciones:

if (:=i(11:8)=0000) instrucción fantasma
ka (:=i(11:8)=0001) complementar A
au (:=i(11:8)=0010) adicionar 1 a A
ru (:=i(11:8)=0011) restar 1 a A
di (:=i(11:8)=0100) desplazamiento hacia la izquierda

dd (:=i(11:8)=0101) desplazamiento hacia la derecha
ri (:=i(11:8)=0110) rotación hacia la izquierda
rd (:=i(11:8)=0111) rotación hacia la derecha
la (:=i(11:8)=1000) limpiar A
sb (:=i(11:8)=1001) salto por bit
(:=i(11:8)=1010)
1011)
1100) libres
1101)
1110)
pm (:i(11:8)= 1111) parar máquina

El rebalse de reg. se detecta con la utilización de un doble bit de signo. En condición que ambos sean iguales para que el número esté dentro del rengo admisible por la máquina. Si ello no sucede, con el pulso P_o de la instrucción siguiente a la de suma, resta o desplazamiento a izquierda, se detiene la máquina indicando el desborde.

5.- Conjunto de instrucciones que están implementadas en la UA.

A la UA pertenece la implementación física que obedece a las siguientes instrucciones: ca, ia, sa, ra, ka, au, ru, di, dd, ri, rd, la. Asimismo pertenecen a ella los reg. A, E, y el banco sumador.

6.- Implementación electrónica:

Este proyecto está realizado con dispositivos de lógica TTL. La alimentación es de 5 Volts respecto de tierra y los niveles de tensión que corresponden al uno y cero lógicos son de 5 volts y tierra respectivamente.

Los elementos empleados son:

sumadores..... .... SN 7480
flip flop D dual ..... SN 7474
cuádruple compuerta NAND de 2 entradas ... SN 7400
SN 7401
SN 15946
triple compuerta NAND de 3 entradas ..... SN 7410
cuadruple compuerta AND-NOR de 2 entradas. MC 7453
extensores para dichas compuertas ..... MC 7460
séxtuple inversores ..... MC 7404

todos ellos dispuestos en la tarjeta T IV que corresponde a la UA, cuyo diseño de conexionado está en las láminas adjuntas.

7.- Lenguaje de programación:

La descripción de la máquina hecha en el lenguaje PCI (Ref.5) es la sig.

Estado de Pc

A⟨0:15⟩
T⟨0⟩
C⟨0:15⟩
E⟨0:15⟩

Estado de M

M[0:16K-1]⟨0:15⟩ (no implementada)
pag 0 $[0:255_{10}]$⟨0:15⟩
pag 1 $[256_{10}:511_{10}]$⟨0:15⟩
pag 2 $[512_{10}:766_{10}]$⟨0:15⟩

Consola de Pc

llaves de dirección⟨0:15⟩
llaves de información ⟨0:15⟩

Formato de instrucción

i⟨0:15⟩

op⟨0:3⟩	:= i⟨12:15⟩	
pC⟨0:1⟩	:= i⟨10:11⟩	pag. 0, 1, 2 y presente
bi	:= i⟨9⟩	bit de indirección.
bo	:= i⟨8⟩	bit de operando
dirección línea⟨0:7⟩	:= i⟨0:7⟩	
dirección pág⟨0:7⟩	:= i⟨8:15⟩	
selector ES⟨0:3⟩	:= i⟨8:11⟩	

Proceso de cálculo de direcciones efectivas

z⟨0:15⟩:=(¬ bi → z'' ;
bi → M[z''])
z'' 0:15 := (pC⟨10:11⟩= 00 → pag 0 □dirección de línea
pC⟨10:11⟩= 01 → pag 1 □ direc. línea
pC⟨10:11⟩= 10 → pag 2 □ direc. línea
pC⟨10:11⟩= 11 → pag presente □direcc. línea)

Conjunto de instrucciones y proceso de ejecución

Ejecución de instrucción := (
ca (:=op=0)→ T□A←b[z]
ia (:=op=1)→ A←A∧M[z]
aa (:=op=2)→T□A←T□A + M[z]
ra (:=op=3)→T□A←T□A - M[z]
sa (:=op=4)→ A≥0→C←z

```
sm (:=op=5 )→(M[z] ≠ 0→C←C + 2)
                (¬M[z] ≠ 0→M[z]←M[z] + 1)
ta (:=op=6 )→M[z]←A
ss (:=op=7 )→M[z]←C, luego C←z, luego C←C + 2
si (:=op=8 )→C←z
ma (:=op=9 )→A←A * M[z]
da (:=op=10)→A←A / M[z]
   (:=op=11) libre
mi (:=op=12) microinstrucciones
ap (:=op=13)
te (:=op=14)→A←P
ts (:=op=15)→P←A
mi (:=op=11) :=
  if  (:=i ⟨8:11⟩=0)  instrucción fantasma
  ka (:=i ⟨8:11⟩=1)→T□A←¬T□A
  au (:=i ⟨8:11⟩=2)→T□A←T□A + 1
  ru (:=i ⟨8:11⟩=3)→T□A←T□A - 1
  di (:=i ⟨8:11⟩=4)→T□A□E←T□A□E x 2
  dd (:=i ⟨8:11⟩=5)→T□A□E←T□A□E / 2
  ri (:=i ⟨8:11⟩=6)→A←A x 2
  rd (:=i ⟨8:11⟩=7)→A←A / 2
  la (:=i ⟨8:11⟩=8)→T□A←0
  sb (:=i ⟨8:11⟩=9)→(¬E⟨15⟩→C←C + 2 )
     (:=i ⟨8:11⟩=10
                =11
                =12  libres
                =13
                =14
  pm (:=i ⟨8:11⟩=15)  parar máquina
```

La subrutina de multiplicación de números enteros es:

```
      ca   A←M[z]
      dd   T□A□E←T□A□E / 2
16    sm   (M[z]≥0→C←C + 2) (¬M[z]≥0→M[z]←M[z] + 1)
veces si   C←z
      la   T□A←0
      sb   (¬E<15)→C←C + 2)
      aa   T□A←T□A + M[z]
15    dd   T□A□E←T□A□E / 2
veces sm   (M[z]≥0→C←C + 2) (¬M[z]≥0→M[z]←M[z] + 1)
      si   C←z
      sb   (¬E<15)→C←C + 2)
      ra   T□A←T□A - M[z]
      di   T□A□E←T□A□E * 2
16    sm   (M[z]≥0→C←C + 2) (¬M[z]≥0→M[z]←M[z] + 1)
veces si   C←z
      si   C←z  (retorno al programa principal)
```

Ya se han considerado las alternativas que se le fue dando a la multiplicación basándose en la implementación física. Desde el punto de vista del mecanismo usado para la multiplicación, existen también distintos métodos. El método que se empleó consiste en detectar el bit menos significativo del multiplicador, si es un cero, el primer subproducto estará formado por una sucesión de ceros; si es un uno, el primer subproducto es copia del multiplicando. Luego se observa el bit siguiente del multiplicador y se repite la operación, sumando el resultado al anterior, pero un lugar desplazado y se obtiene así el segundo subproducto acumulado. Al desplazar los subproductos parciales se repite el bit más significativo en el lugar que queda libre. Se ha logrado mayor velocidad haciendo que si el bit observado es un cero, el lugar de sumar una sucesión de ceros y luego desplazar, se desplace directamente.

Al llegar al último subproducto, si el multiplicador es un número positivo (bit signo = 0) se desplaza y si es un número negativo (bit signo = 1) se resta desplazado.

La división, luego de bastante esfuerzo, también se obtuvo a partir de las instrucciones de la máquina. Existen varios métodos de división que luego se pasará a detallar. Previamente se verá la subrutina usada. El método empleado es el de previa inversión del divisor.

(1) ca con el divisor/d
(2) sa ir a (20)
(3) ri
(4) sm
(5) ta
(6) ca con un vector = -1_{10}
(7) di
(8) ta
(9) ca con lo que había en (5)
(10) sa ir a (12)
(11) si ir a (3)
(12) rd
(13) rd
(14) ta
(15) ca con lo que había en (8)
(16) dd
(17) dd
(18) ia con lo que había en (14)
(19) si ir a (25)
(20) ri
(21) sm
(22) sa ir a (20)
(23) rd
(24) rd
(25) ta
(26) ca con el vector -16_{10}
(27) ra restar M[z]que corresponde a (21)
(28) au
(29) au
(30) au
(31) au
(32) ta
(33) ca con lo que había en (25)
(34) sa ir a (51)
(35) ka
(36) au

(37) ra (A - 2($\sqrt{3}$ - 1))
(38) dd
(39) sm
(40) si ir a (38)
(41) ss para multiplicar el contenido de A por el divisor.
(42) aa al resultado le adiciono 10.0000000000000
(esto es -2_{10} en fraccionario)
(43) ka
(44) au
(45) ss para multiplicar A por lo que había en A antes de (41)
(46) ta
(47) sm
(48) si ir a (41) (repetir 3 veces)
(49) ss para halla el dividendo por la inversa de d.
(50) si al programa principal
(51) ra (A - 2($\sqrt{3}$ - 1))
(52) ka
(53) au
(54) si ir a (38)

Esta subrutina está basada en un artículo de la referencia 1 y consiste en lo siguiente: Primeramente se debe normalizar el divisor para que quede de la forma: $d = a\ 2^p$; en la cual d es un entero mayor ó igual a 1. Se toma como primera aproximación:

$z_1 = b\ 2^q$ donde $q = 1 - p$

$b = 2(\sqrt{3} - 1) - a$ si $a \geq 0$

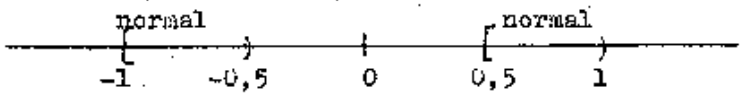
$b = 2(\sqrt{3} - 1) - a$ si $a < 0$

y las sucesivas aproximaciones son:

$$z_2 = z_1 (2 - d\ z_1)$$
$$z_3 = z_2 (2 - d\ z_2)$$
$$z = z_3 (2 - d\ z_3)$$

Teniendo en cuenta que nuestra máquina trabaja con números positivos y negativos y que el bit más significativo representa el signo, la normalización consiste en multiplicar la mantisa de un número por su base tantas veces como sea necesario para que quede de la forma 00,1..... ó 11,0..... y simultáneamente disminuir en uno su exponente cada vez que se multiplique por la base para que no varíe su valor. Con este procedimiento se logra una mantisa normalizada que por definición pertenece a alguno de los in-

tervalos marcados en la figura:



Se necesita colocar la coma decimal después del segundo bit porque en este método se debe operar con un número positivo que es: $2(\sqrt{3}-1) = 1,4641_{10} = 01,0111011011011_2$; colocar la coma binaria luego del primer bit implicaría transformar el número en negativo.

Otra alternativa que se probó fue hallar la inversa de la forma: $z_1 = \frac{b}{2}\ 2^{q+1}$ donde $\frac{b}{2} = (\sqrt{3}-1) - \frac{a}{2}$ si $a \geq 0$

$\frac{b}{2} = -(\sqrt{3}-1) - \frac{a}{2}$ si $a < 0$

y trabajar con el número $(\sqrt{3}-1) = 0,73205_{10} = 0,1011101101110011_2$ pero con ello no se ahorran instrucciones ni se simplifica el método, sino que el tiempo de operación es el mismo y da menos precisión. Por lo tanto no se aceptó.

En el mismo artículo de la Ref. 1 se describe además el método no restitutivo. En este el cociente se obtiene directamente bit a bit, dependiendo del signo del resto anterior y de la operación realizada y ésta a su vez depende de si los bits más significativos del resto anterior y del divisor son iguales ó distintos, esto es la función equivalencia de signo. El procedimiento se resume en la sig. tabla.

FES (R_i, d)	operación	FES (R_i, R_{i+1})	nuevo dígito del cociente
1	resta	1	1
		0	0
0	suma	1	0
		0	1

Para acelerar este procedimiento de división se puede trabajar con números normalizados en base octal, es decir se operan por ternas de bits y el cociente también se obtiene por ternas siguiendo un método.

Además, existe otro método no restitutivo (Ref. 3) parecido al anterior que consiste en restar al dividendo el divisor y una vez realizada la resta se desplaza un lugar el divisor y se repite la operación. El cociente se obtiene observando el signo de cada resto. Si el nuevo resto es del mismo signo que el anterior el bit correspondiente del cociente es un 1; si son de signo con-

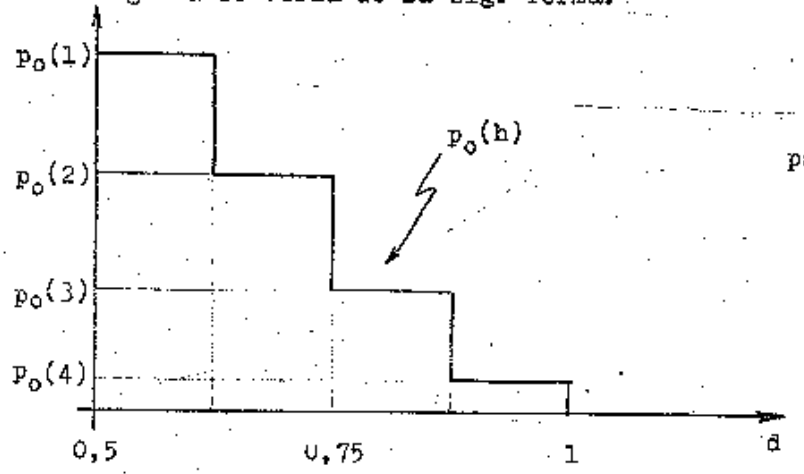
trario se le vuelve a sumar el divisor colocando un cero en el lugar del cociente correspondiente y se resta nuevamente el divisor pero desplazado un lugar más. Este método presenta el inconveniente de requerir muchas más operaciones que el primero descripto.

Otra alternativa para realizar la división hallando previamente la inversa del divisor es el método de división usando un multiplicador paralelo. (Ref. 4). En ella previamente hay que normalizar el dividendo y el divisor y la primera aproximación de la inversa del divisor es p_o, que se obtiene de la sig. manera:

$$p_o(h) = \frac{2^{j+1}}{2^j + h - \frac{1}{2}}$$

donde j son los j primeros dígitos que siguen al punto binario en el divisor. Así se obtienen p_o de valor constante en cada uno de los intervalos 2^j entre $\frac{1}{2}$ y 1. Se puede demostrar que el valor óptimo de p_o para el h-ésimo intervalo corresponde a su punto medio; donde $h = 1,2,\ldots\ldots,2^j$.

En un diagrama se vería de la sig. forma:



El algoritmo para mejorar p_o es el sig.:

constitución inicial	$p_o = f(d)$	$a_o = p_o d$
paso 1	$p_1 = p_o(2-a_o)$	$a_1 = a_o(2-a_o)$
paso i	$p_i = p_{i-1}(2-a_{i-1})$	$a_i = a_{i-1}(2-a_{i-1})$

Cuando i tiende a infinito, p_i converge a 1/d, a_i converge a 1 y la convergencia es cuadrática.

Existe un algoritmo para optimizar esté método y tomar la función 1/d con su mayor aproximación, en lugar de considerar la función escalonada. Este algoritmo también se basa en una fun-

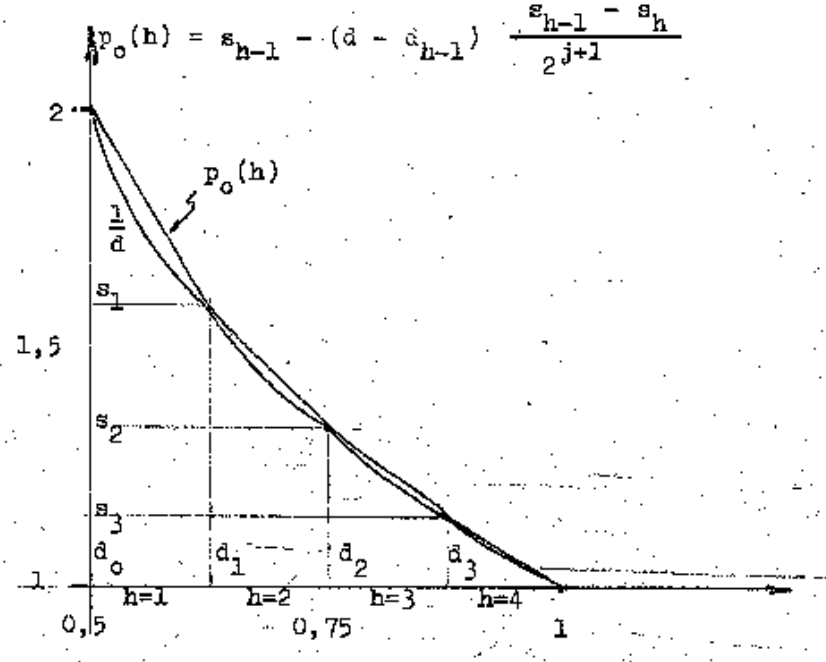
ción obtenida en base a los j primeros dígitos después de la coma. Consiste en obtener 2 números que son las inversas de d_{n-1} y d_n del intervalo h en el cual se halla el valor del divisor cuya inversa queremos hallar.

Es decir se obtiene:

$$s_{h-1} = \frac{1}{d_{h-1}} \qquad y \qquad s_h = \frac{1}{d_h}$$

y la primera aproximación es:

$$p_o(h) = s_{h-1} - (d - d_{h-1}) \frac{s_{h-1} - s_h}{2^{j+1}}$$



Se llama r_i al error relativo en p_{i-1}.

El error relativo r_1 en p_o es siempre negativo y alcanza su valor máximo cuando d está geométricamente entre los límites del primer intervalo (h = 1)

$$r_{1máx.} = \frac{4\ V2^j(2^j+1) - 2\ (2^{j+1}+1)}{2^{j+1}}$$

Estos errores relativos se pueden tabular para cada j determinado.

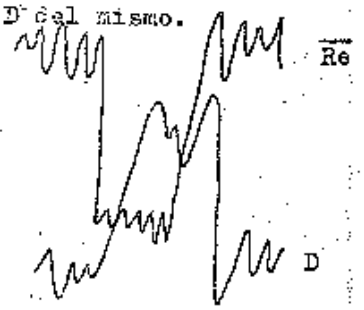
El método que se ha elegido de inversión del divisor es rápido para su ejecución y simple de programar. Otra ventaja que tiene es que cuando se trabaja con matrices generalmente se necesitan muchas divisiones por el determinante y el obtener su inversa ($\frac{1}{\Delta}$) simplifica las operaciones y ahorra tiempo.

8.- Ensayos:

Los ensayos de prueba que había sido realizados parcialmente en cada tercio de tarjeta individualmente dieron en general resultados satisfactorios. El inconveniente más grave con el que se tropezó fue el siguiente: En la microinstrucciones complementar A, rotar A a la derecha ó izquierda y desplazar A a derecha ó izquierda, cuando en el reg. A el mayor número de flip flop debería estar en estado 1, estos no cambiaban de estado. Esto es por ej.: en el caso de complementar A ocurria que si estaban todos en estado 1, al complementar pasaban correctamente al estado 0. Si estaban la mitad en 1 y la otra mitad en 0 (cualquiera fuera su disposición) se complementaban de manera correcta, pero si de todos ceros debía pasar a todos unos, esto no ocurría, acomodándose en cambio el reg. con una disposición al azar y a partir de allí sí complementaba bien.

Este problema tenía además el agravante que al marchar la máquina en forma repetitiva con la instrucción complementar, el reg. se acomodaba en losprimeros pulsos y a partir de ahí funcionaba bien, lo que impedía detectar la falla en el momento que se producía.

Dicho problema se atacó a varios niveles, de manera un poco azarosa ya que no se podía precisar de dónde provenía. En principio se pensó que el pulso que entra por la entrada Re a los flip flop y que carga a los 16 flip flop no sería suficiente para producir el cambió de todos ceros a todos unos, puesto que en el momento del cambio se consume mucha corriente. Entonces se piramidó la entrada de Re de manera que cada compuerta alimentaba solamente 4 flip flop. Esto introducía además un retardo (max de 60 nseg.) debido a las compuertas de piramidación, lo cual permitía además que la pendiente ascendente de Re con la cual cambia el estado del flip flop se produzca totalmente durante la entrada D del mismo.



Asimismo la entrada por D a los flip flop, proveniente de compuertas del árbol, era un pulso de configuración mala.

No mejorándose nada con esto se intentó piramidar la entrada proveniente del pulso que indica complementar A (obtenido de la intersección de la instrucción [ka] con el pulso P5) pero esto tam-

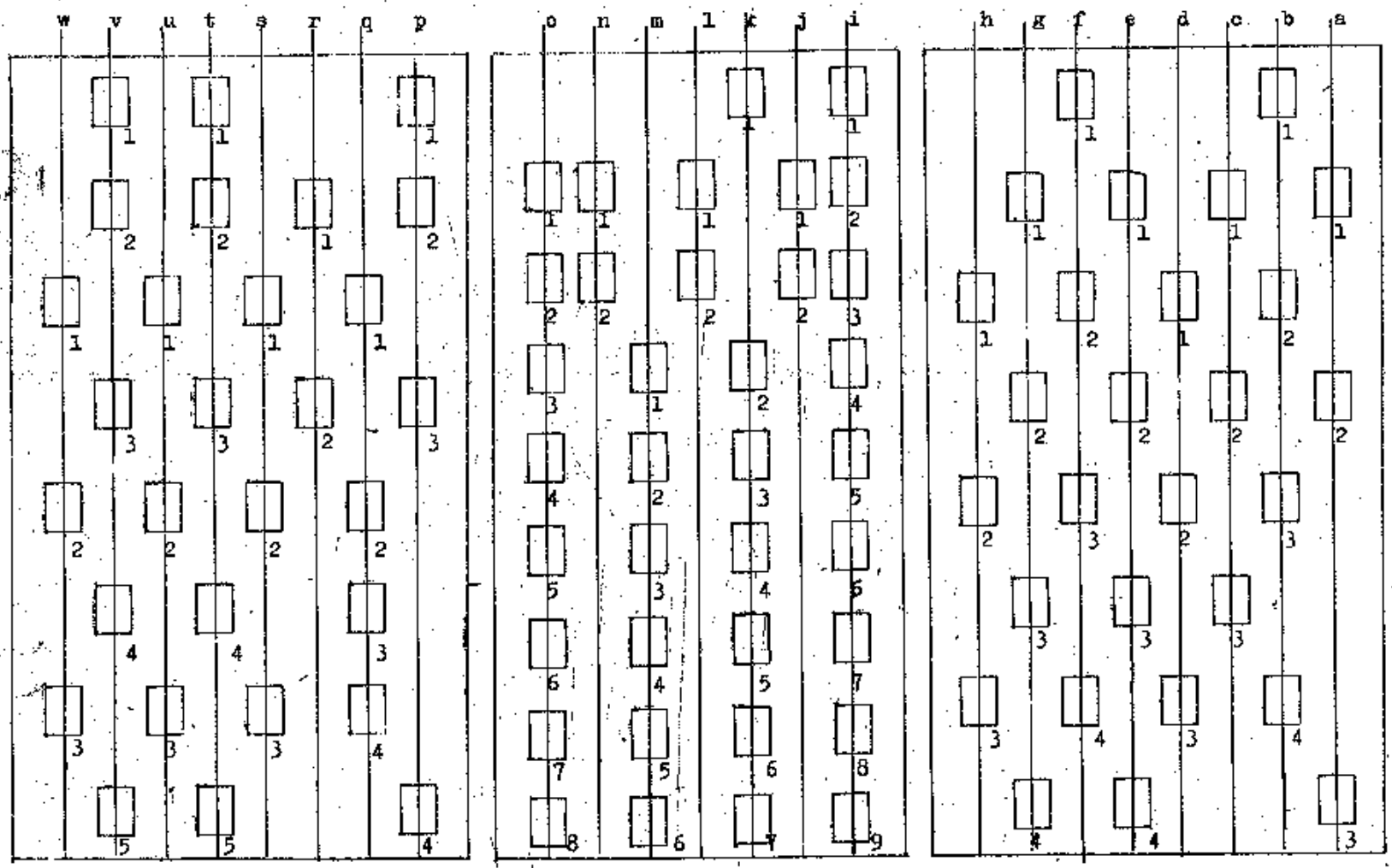
poco mejoró las cosas.

Posteriormente se comprobó que tanto la señal de tierra como la señal de +5 volts tenían superpuesta una onda de alta frecuencia cuya amplitud era suficientemente grande como para sacar las señales de alimentación de las especificaciones impuestas para cada circuito (+4,75 +5,25 volts). Esto es debido a capacidades parásitas inducidas en los cables de alimentación. Fue necesario poner condensadores en las tarjetas (un condensador cada 4 chips). Ellos se cargan y en los momentos que la fuente baja el nivel de voltaje los condensadores aportan energía para compensarlo, puesto que la fuente de alimentación regulada responde muy lentamente a la variación de tensión, comparado con los 2 Megaciclos de frecuencia a la que estamos trabajando. Con esto se logra disminuir los picos que se superponen a la señal de tierra y alimentación filtrando la onda de alta frecuencia.

Esto además hizo necesario colocar una barra de tierra para uniformar la tierra de referencia de todas las tarjetas.

9.- Agradecimientos:

Se agradece la colaboración prestada por el Ing. Manuel Pascual, Coordinador del área III de Electrónica; así como al Ing. Enrique Arroyo, Asistente del Área VI de Sistemas Digitales.



disposición física de los circuitos que forman la Unidad Aritmética

Símbolos y nomenclaturas

IV c 4 - 9 := Circuito de tarjeta IV, columna c, fila 4, pata de salida 9.

IV c 4 / 9 := Pata 9 del circuito de tarjeta IV, columna c, fila 4.

IV / 2 / 30 := Pata 30 del conector 2 de la tarjeta IV.

IV / 2 := Conector 2 de la tarjeta IV.

Las instrucciones originan las sig. funciones respectivas;

[∧] función lógica and. [∧] = [and] . P5 = [IA]

[S1M] función sumar 1 a memoria. [S1M] = [sm] . [M ≠ 0] . P5-8

[A+1] función sumar 1 a A. [A+1] = [au] . P5-8

[A-1] función restar 1 a A. [A-1] = [ru] . P5-8

[(A+1)V(A-1)] función [A+1] unida a función [A-1]

[DD] función desplazamiento a la derecha. [DD] = [dd] . P5

[DI] función desplazamiento a la izquierda. [DI] = [di] . P5

[RD] función rotación a la derecha. [RD] = [rd] . P5

[RI] función rotación a la izquierda. [RI] = [ri] . P5

[DDVRD] función despl. derecha unida a función rotación derecha.

[DIVRI] función despl. izquierda unida a función rotación izquierda.

[SM] función sumar A más memoria. [SM] = [aa] . P5-8

[RM] función restar A menos memoria. [RM] = [ra] . P5-8

[CRΣ] unión de las funciones de entrada al sumador (sumar-restar)

[SAVRA] función sumar A unida a función restar A.

[SAVRA] = ([aa] V [ra]) . P5-8

[¬A] función complementar A. [A] = [ka] . P5 = [KA]

[¬T] función complementar T. [T] = [ka] . P5 = [KT]

[Re A] función entrada de reloj de A. Es unión de todas las funciones de entrada al A.

[Re B] Función entrada de reloj a B. [Re B] = ([di]V[dd]) . P5

[NC] función no conectado. Indica que la pata está al aire.

Desde el punto de vista lógico equivale a conectar un 1.

A⟨i⟩ bit i-ésimo de A

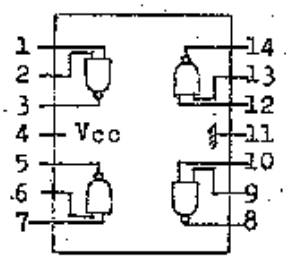
M⟨i⟩ bit i-ésimo de M

Σ⟨i⟩ sumador correspondiente al bit i-ésimo

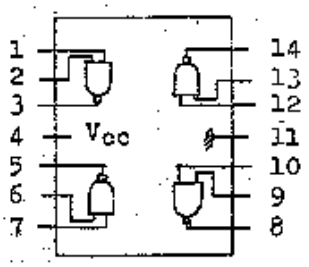
} (con i=0,1,...,15)

[LA] función limpiar A. [LA] = [la] . P5

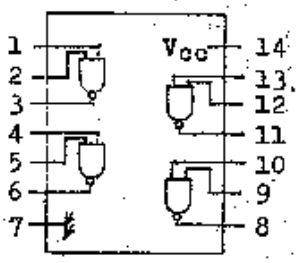
SN 7400



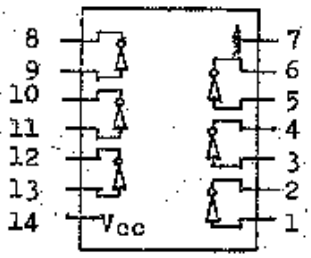
SN 7401



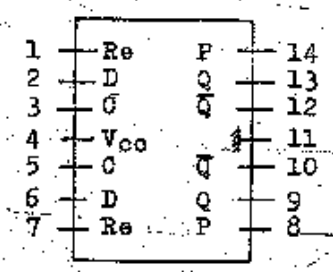
SN 15946



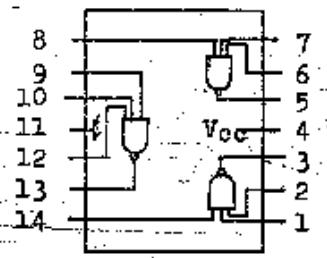
MC 7404



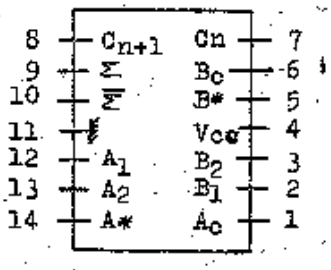
SN 7474



SN 7410



SN 7480



Circuitos integrados utilizados.

tarjeta IV - Unidad Aritmética

conector 1:	exterior	interior
1.- salida de $\overline{A\langle 15\rangle}$	II/2/5	IVv2/13
2.-		
3.- sal. de $(\overline{A \wedge M\langle 15\rangle})$	I/3/5	IVv5/13
4.- M⟨15⟩		IVv5/10
5.-		
6.- Vcc (conector 1)	barra	IVt5/4
7.- tierra (conector 1)	barra	IVp4/11
8.-		
9.- sal de $(\overline{A \wedge M\langle 14\rangle})$	I/2/17	IVv5/5
10.- M⟨14⟩		IVv5/7
11.- M⟨13⟩		IVt5/9
12.- sal de $(\overline{A \wedge M\langle 13\rangle})$	I/3/13	IVt5/13
13.- tierra (conector 2)	barra	IVo8/4
14.- Vcc (conector 2)	barra	IVo8/11
15.- sal de $(\overline{A \wedge M\langle 12\rangle})$	I/2/25	IVt5/3
16.- M⟨12⟩		IVt5/1
17.- M⟨11⟩		IVt5/7
18.- sal de $(\overline{A \wedge M\langle 11\rangle})$	I/2/9	IVt5/5
19.- viene de T	IV/5/23	IVv1/13
20.- control M[z]	V/2/17	IVo4/6
21.- [CE Σ]	IV/3/24	IVt1/6
22.- [KA]	III/4/12	IVi9/9
23.- [RD]	III/4/9	IVi6/2
24.- [DD]	III/4/17	IVi6/1
25.- M⟨10⟩		IVp4/9
26.- sal de $(\overline{A \wedge M\langle 10\rangle})$	I/1/25	IVp4/13
27.- [RIVDI]	III/4/8	IVe1/9
28.- sal de $(\overline{A \wedge M\langle 9\rangle})$	I/2/1	IVp4/3
29.- M⟨9⟩		IVp4/14
30.- M⟨8⟩		IVp4/7
31.- sal de $(\overline{A \wedge M\langle 8\rangle})$	I/1/17	IVp4/5
32.- viene de A⟨4⟩	IV/4/16	IVm5/10

conector 2:

	externa	interna
1.-[ke E]	IV/3/8	IVq1/1
2.- A(4)	I/5/11	IVo7/8
3.- A(3)	I/5/27	IVo8/8
4.- A(11)	I/2/10	IVo6/8
5.- A(12)	I/2/26	IVo5/8
6.- viene de A(5)	IV/4/17	IVk6/10
7.- viene de A(11)	IV/6/10	IVo5/10
8.- A(13)	I/3/14	IVm3/8
9.- A(10)	I/1/26	IVm4/8
10.- A(5)	I/5'/11	IVm5/8
11.- A(2)	I/5/19	IVm7/8
12.- viene de A(3)	IV/4/10	IVo7/10
13.- viene de A(12)	IV/6/16	IVm3/10
14.- viene de A(10)	IV/6/11	IVo6/10
15.- A(14)	I/2/18	IVk4/8
16.- A(9)	I/2/2	IVk5/8
17.- A(6)	I/5/3	IVk6/8
18.- A(1)	I/5'/19	IVk7/8
19.- viene de A(2)	IV/4/11	IVo8/10
20.- viene de A(7)	IV/4/27	IVi7/10
21.- viene de A(13)	IV/6/17	IVk4/10
22.- viene de A(14)	IV/6/26	IVi6/10
23.- viene de A(9)	IV/6/2	IVm4/10
24.- A(15)	I/3/6	IVi6/8
25.- A(8)	I/1/18	IVi7/8
26.- A(7)	I/5'/3	IVi8/8
27.- A(0)	I/5'/27	IVi9/8
28.- viene de A(1)	IV/4/2	IVm7/10
29.- viene de A(8)	IV/6/1	IVk5/10
30.- viene de A(6)	IV/4/26	IVk6/10
31.- viene de A(0)	IV/4/1	IVk7/10
32.- viene de A(15)	IV/6/27	IVm2/3

conector 3:

	exterior	interior
1.- Vcc (conector 3)	barra	IVg4/4
2.- tierra (conector 3)	barra	IVg4/11
3.- [DDVRD]	III/4/10	IVi9/1
4.- [A - 1]	III/4/20	IVe3/13
5.- M<6>		IVg4/8
6.- M<7>		IVg4/14
7.- [DI]	III/4/7	IVi9/2
8.- [Re E]	III/4/18	IVh1/7
9.- sal de $(\overline{A \wedge M\langle 7\rangle})$	I/5'/2	IVg4/3
10.- sal de $(\overline{A \wedge M\langle 6\rangle})$	I/5/2	IVg4/5
11.- M<5>		IVe4/10
12.- [SAVRA]	V/2/15	IVh2/12
13.- [RI]	III/4/19	IVi9/5
14.- sal de $(\overline{A \wedge M\langle 5\rangle})$	I/5'/10	IVe4/13
15.- M<4>		IVe4/7
16.- M<3>		IVe4/3
17.- [A + 1]	V/2/8	IVe3/6
18.-		
19.- [LA]	V/5/14	IVg1/5
20.- [(A+1) V (A-1)]	III/3/24	IVc3/12
21.- sal de $(\overline{A \wedge M\langle 3\rangle})$	I/5/26	IVe4/3
22.- sal de $(\overline{A \wedge M\langle 4\rangle})$	I/5/10	IVe4/5
23.- [SM]	III/4/15	IVb4/7
24.- [CE]	V/2/10	IVb3/1
25.- M<0>		IVa3/10
26.- sal de $(\overline{A \wedge M\langle 0\rangle})$	I/5'/26	IVa3/13
27.- [IA]	III/4/6	IVa3/12
28.- M<2>		IVa3/14
29.- sal de $(\overline{A \wedge M\langle 1\rangle})$	I/5'/18	IVa3/5
30.- M<1>		IVa3/7
31.- [RM]	III/4/11	IVb4/12
32.- sal de $(\overline{A \wedge M\langle 2\rangle})$	I/5/18	IVa3/3

conector 4:

	exterior	interior
1.- sal de A(0)	I/6/31	IVa1/9
2.- sal de A(1)	I/6/27	IVa1/13
3.- sal de Σ(3)	I/6/20	IVd2/10
4.- sal de Σ(0)	I/6/32	IVa2/9
5.- sal de Σ(1)	I/6/28	IVb3/10
6.- [SIM]	III/4/13	IVb1/2
7.- sal de E(0)	IV/5/3	IVb2/9
8.- sal de E(1)	IV/5/5	IVb2/13
9.- sal de Σ(2)	I/6/24	IVc2/9
10.- sal de A(3)	I/6/19	IVc1/13
11.- sal de A(2)	I/6/23	IVc1/9
12.- entr a A(0)	I/5'/25	IVa1/6
13.- entr a A(1)	I/5'/17	IVa1/2
14.- entr a A(3)	I/5/25	IVc1/2
15.- entr a A(2)	I/5/17	IVc1/6
16.- sal de A(4)	I/4/16	IVe1/9
17.- sal de A(5)	I/4/12	IVe1/13
18.- sal de E(2)	IV/5/10	IVd1/9
19.- sal de E(3)	IV/5/11	IVd1/13
20.-sal de E(4)	IV/5/17	IVf2/9
21.- sal de E(5)	IV/5/18	IVf2/13
22.- sal de Σ(4)	I/4/13	IVe2/9
23.- sal de Σ(5)	I/4/9	IVf2/10
24.- sal de Σ(6)	I/4/5	IVg2/9
25.- sal de Σ(7)	I/4/1	IVh2/10
26.- sal de A(6)	I/4/8	IVg1/9
27.- sal de A(7)	I/4/4	IVg1/13
28.- entr a A(5)	I/5'/9	IVe1/2
29.- entr a A(4)	I/5/9	IVe1/6
30.- entr a A(7)	I/5'/1	IVg1/2
31.- entr a A(6)	I/5/1	IVg1/6
32.- sal de E(6)	IV/5/19	IVh1/9

conector 5:

	exterior	interior
1.- sal de E⟨7⟩	IV/5/6	IVh1/13
2.- [LA]	IV/3/19	IVg1/3
3.- de E⟨0⟩	IV/4/7	IVj2/10
4.- [Re A]	IV/5/30	IVg1/1
5.- de E⟨1⟩	IV/4/8	IVj1/12
6.- de E⟨7⟩	IV/5/1	IVo2/10
7.-		
8.- [SM]	IV/3/23	IVi4/2
9.- [RM]	IV/3/31	IVi4/9
10.- de E⟨2⟩	IV/4/18	IV12/10
11.- de E⟨3⟩	IV/4/19	IVn2/10
12.-		
13.-		
14.- de E⟨15⟩	IV/6/32	IVi9/3
15.- de E⟨14⟩	IV/6/21	IVi2/12
16.- de E⟨13⟩	IV/6/20	IVi1/12
17.- de E⟨4⟩	IV/4/20	IVo2/12
18.- de E⟨5⟩	IV/4/21	IVi2/10
19.- de E⟨6⟩	IV/4/32	IVj1/10
20.-		
21.-		
22.- sal de E⟨0⟩	V/4/31	IVb2/10
23.- sal de T	III/3/31	IVv1/13
24.-		
25.- de E⟨12⟩	IV/6/19	IVk1/7
26.- de E⟨11⟩	IV/6/18	IVn1/10
27.- de E⟨10⟩	IV/6/6	IV11/10
28.- de E⟨8⟩	IV/6/7	IVo1/10
29.- de E⟨9⟩	IV/6/8	IV11/13
30.- [Re A]	V/5/2	IVp2/1
31.- [DD]	IV/1/24	IVo1/13
32.- [DI]	IV/3/7	IVo1/9

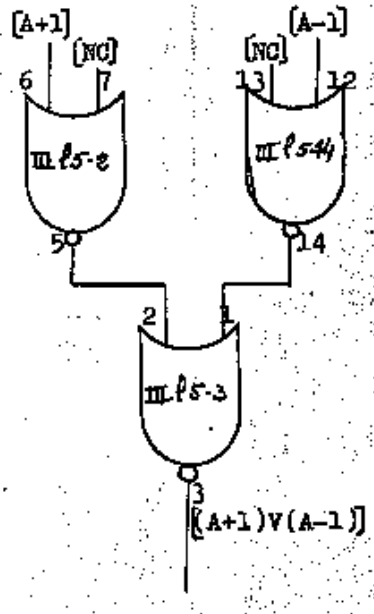
conector 6:

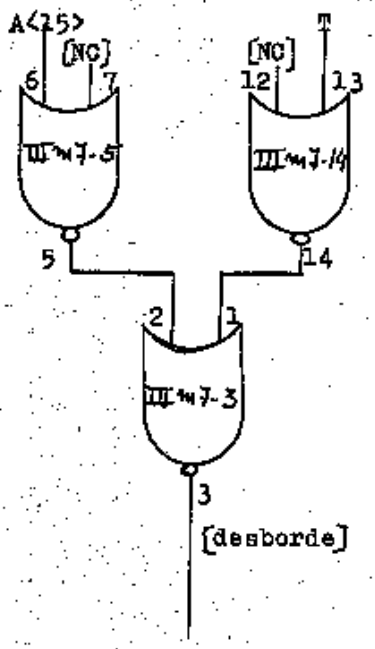
	exterior	interior
1.- sal de A⟨8⟩	I/1/2	IVp2/9
2.- sal de A⟨9⟩	I/1/6	IVp2/13
3.- sal de Σ⟨8⟩	I/1/1	IVp3/9
4.- sal de Σ⟨9⟩	I/1/5	IVq2/10
5.- sal de Σ⟨10⟩	I/1/9	IVr2/9
6.- sal de E⟨10⟩	IV/5/27	IVs1/9
7.- sal de E⟨8⟩	IV/5/28	IVq1/9
8.- sal de E⟨9⟩	IV/5/29	IVp1/13
9.- sal de Σ⟨11⟩	I/1/13	IVs2/10
10.- sal de A⟨11⟩	I/1/14	IVr1/13
11.- sal de A⟨10⟩	I/1/10	IVr1/9
12.- entr a A⟨9⟩	I/2/8	IVp2/2
13.- entr a A⟨8⟩	I/1/24	IVp2/6
14.- entr a A⟨11⟩	I/2/16	IVr1/2
15.- entr a A⟨10⟩	I/1/32	IVr1/6
16.- sal de A⟨12⟩	I/3/19	IVt2/9
17.- sal de A⟨13⟩	I/3/23	IVt2/13
18.- sal de E⟨11⟩	IV/5/26	IVs1/13
19.- sal de E⟨12⟩	IV/5/25	IVu1/9
20.- sal de E⟨13⟩	IV/5/16	IVu1/13
21.- sal de E⟨14⟩	IV/5/14	IVw1/9
22.- sal de Σ⟨12⟩	I/3/20	IVt3/9
23.- sal de Σ⟨13⟩	I/3/24	IVu2/10
24.- sal de Σ⟨14⟩	I/3/28	IVv3/9
25.- sal de Σ⟨15⟩	I/3/32	IVw2/10
26.- sal de A⟨14⟩	I/3/27	IVv2/9
27.- sal de A⟨15⟩	I/3/31	IVv2/13
28.- entr a A⟨13⟩	I/3/16	IVt2/2
29.- entr a A⟨12⟩	I/2/32	IVt2/6
30.- entr a A⟨15⟩	I/3/8	IVt2/2
31.- entr a A⟨14⟩	I/2/24	IVt2/6
32.- sal de E⟨15⟩	IV/5/10	IVw1/13

tarjeta III

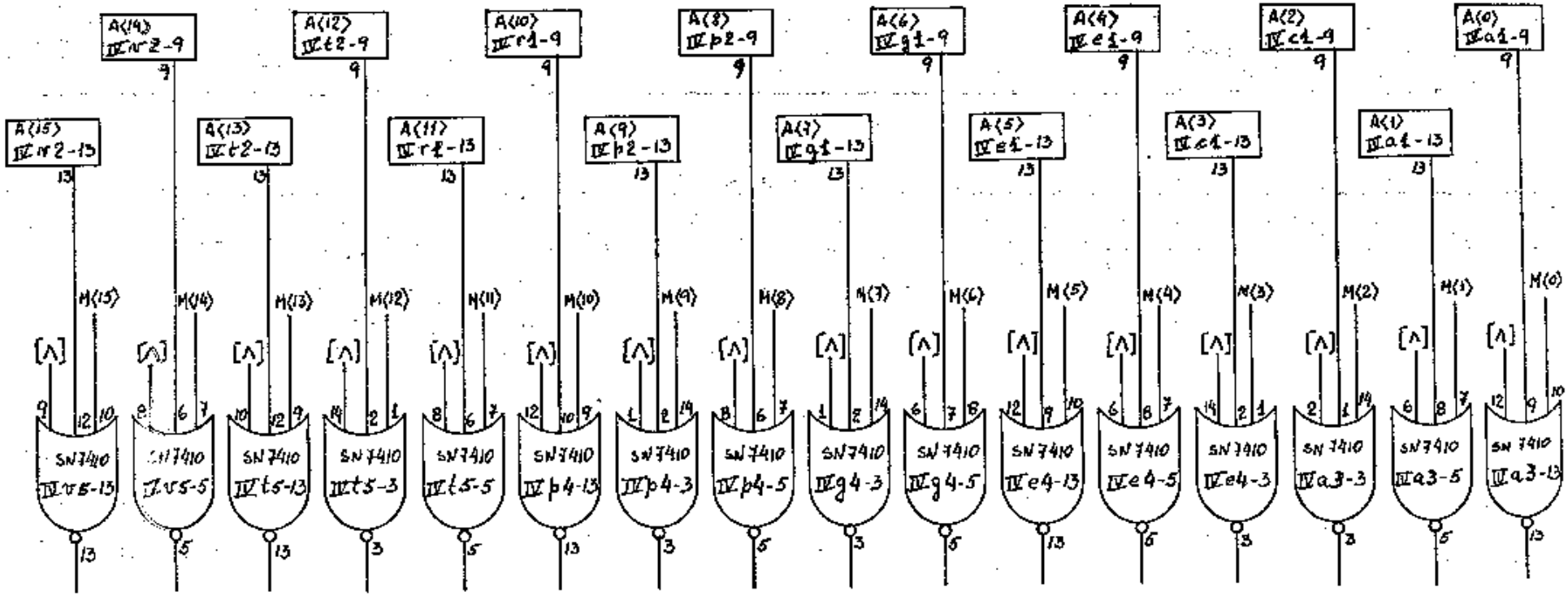
conector 3:

22.- tierra	barra	III15/4
23.- Vcc	barra	III15/11
24.- [(A+1) V (A-1)]	IV/3/20	III15/3
25.- [A + 1]	IV/3/17	III15/6
26.- [A - 1]	IV/3/4	III15/12
27.-		
28.- A<15>	IV/6/27	IIIm7/6
29.-		
30.-		
31.- T	IV/5/23	IIIm7/13
32.- sal de [desborde]	V/3/3	IIIm7/3

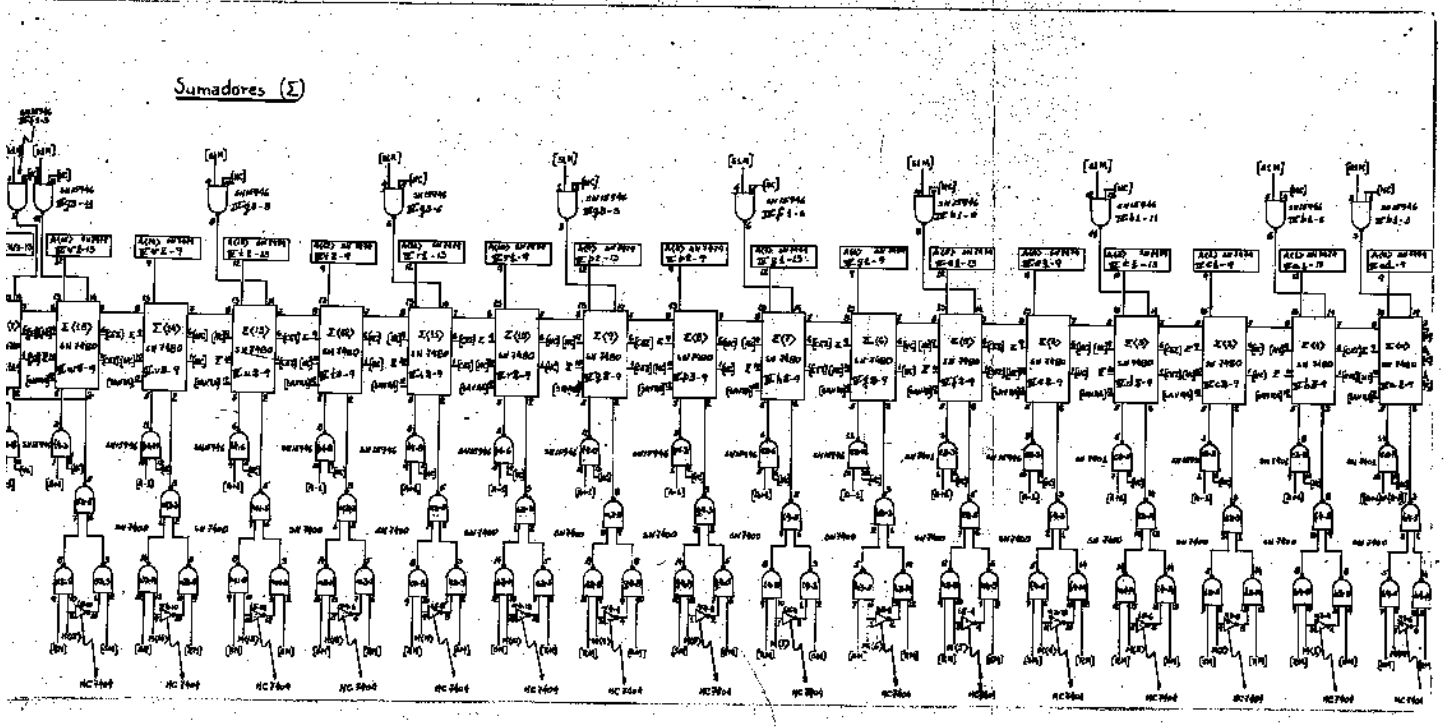




and (∧)


A(15)
IV n2-13
13
M(15)
[∧]
SN7410
IV n5-13
A(14)
IV n2-9
9
M(14)
[∧]
SN7410
IV n5-5
A(13)
IV t2-13
13
M(13)
[∧]
SN7410
IV t5-13
A(12)
IV t2-9
9
M(12)
[∧]
SN7410
IV t5-3
A(11)
IV r2-13
13
M(11)
[∧]
SN7410
IV t5-5
A(10)
IV r1-9
9
M(10)
[∧]
SN7410
IV p4-13
A(9)
IV p2-13
13
M(9)
[∧]
SN7410
IV p4-3
A(8)
IV p2-9
9
M(8)
[∧]
SN7410
IV p4-5
A(7)
IV g1-13
13
M(7)
[∧]
SN7410
IV g4-3
A(6)
IV g1-9
9
M(6)
[∧]
SN7410
IV g4-5
A(5)
IV e1-13
13
M(5)
[∧]
SN7410
IV e4-13
A(4)
IV e1-9
9
M(4)
[∧]
SN7410
IV e4-5
A(3)
IV c1-13
13
M(3)
[∧]
SN7410
IV e4-3
A(2)
IV c1-9
9
M(2)
[∧]
SN7410
IV a3-3
A(1)
IV a1-13
13
M(1)
[∧]
SN7410
IV a3-5
A(0)
IV a1-9
9
M(0)
[∧]
SN7410
IV a3-13

Sumadores (Σ)



Desplazamientos y Rotaciones