90

ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA

DE TELECOMUNICACION

TITULO:

ESTUDIO DE UNA CPU DE 16 BITS CON MICROPROCESADORES MICROPROGAMABLES EN BIT-SLICE

AUTOR:

ANGEL GALLO LUÑO

TUTOR:

SEBASTIAN SUAREZ GIL

LAS PALMAS DE GRAN CANARIA FEBRERO 1982

INDICE

I) MEMORIA.

./.

MEMORIA

1.- OBJETIVO.-

Se redacta el presente proyecto, en calidad de tabajo de fin de carrera, como complemento de las enseñanzas recibidas a lo largo de la carrera de INGENIERIA TECNICA DE TELECOMUNICACION.

Dicho trabajo es requisito previo e indispensable para la obtención del título correspondiente, según el vigente Plan de Estudios, en el que está incluído como asignatura del último curso de la mencionada carrera.

La redacción del proyecto se lleva a cabo a petición de la Escuela Universitaria de Ingeniería Técnica de Telecomunicación, con domicilio social en la Calle de Tomás de Morales, s/n, y en cuya representación actúa como tutor el Profesor de la misma, Sr. D. Sebastián Suárez Gil.

2.- LA TECNOLOGIA DE LOS CIRCUITOS INTEGRADOS LSI Y LA MICROPROGRAMACION.

Es sabida la importancia de la reducción de tamaño a la construcción de módulos funcionales complejos.

Basados en esta importancia, los fabricantes de circuitos integrados han suministrado a los usuarios una amplia gama de microprocesadores, intentando cada uno de ellos aventajar al competidor ofreciendo al usuario un conjunto de instrucciones más potentes.

Sin embargo, en el campo de la microprogrmación la norma fundamental es la flexibilidad del conjunto de instrucciones. Por tal causa el fabricante de circuitos integrados ha accedido al mercado de una manera diferente, y apoyándose en los siguientes criterios:

a) Construir módulos que flexibilizan la construcción de elementos de proceso adecuados a distintos usuarios.
b) Construir módulos que simplifiquen la construcción de los elementos de control microprogramado.
c) Suministrar herramientas que simplifiquen el tiempo de desarrollo y puesta a punto.

Así por ejemplo, los módulos de proceso se suministran para un número de bits (bit slice) que el usuario puede incorporar en su diseño en la cantidad adecuada a sus necesidades de proceso. Se ha de resaltar que estos módulos de proceso no suelen ser meros operadores o unidades aritmético-lógicas (ALU), sino que suelen llevar incorporados registros de trabajo con el

mismo número de bits que maneja la unidad de proceso. Es por eso que también suele llamarse a estos módulos, unidades aritmético-lógicas con registros (ALU).

Por otro lado los módulos que tienden a la simplificación de la unidad de control microprogramada son las denominadas unidades de control microprogramada que engloban con un criterio particular del fabricante lo que nosotros hemos llamado selector de direcciones de la ROM de control de microprograma.

Por último las herramientas suministradas por el fabricante para simplificar el desarrollo y puesta a punto son análogas a las suminstradas con los microprocesadores convencionales, tanto desde el punto de vista *software como hardware*, como una solución alternativa a la dada anteriormente que arranca de un mercado promocionado por el fabricante de circuitos integrados, está dada por los fabricantes de ordenadores que arranca de un mercado creado por ellos mismos al aplicar sus productos hardware/software en una época anterior a la de la tecnologia actual.

La solución del fabricante de ordenadores es la de incorporar la nueva tecnología a los productos existentes con tecnología convencional; para ello encuentra nuevamente en las técnicas de microprogramación el camino ideal para materializar máquinas nuevas compatibles con las convencionales pero internamente diferentes.

Sin embargo, la solución del fabricante de ordenadores, que ya tiene asegurado un mercado, es diferente, pues normalmente encarga la integración de su estructura interna a un fabricante de circuitos integrados (CUSTOM DESIGN) a la cual aña-

de la ROM de control (CROM) con el microprograma correspondiente al conjunto de instrucciones y al código máquina de sus predecesores tecnológicos.

3.- ARQUITECTURA DEL ORDENADOR.

En este capítulo de introducción intentaré:

a) Desarrollar una terminología común para capítulos posteriores.

b) Introducir diversos tópicos sobre el diseño de computadoras de programa almacenado.

c) Plantear algunos problemas de la arquitectura de los ordenadores (que serán resueltos a lo largo de este estudio).

Para realizar estas premisas, empezaré con el ordenador básico. Sin embargo, las ideas generales descritas serán apropiadas para ganar alguna familiaridad con los dispositivos microprogramables bit-slice.

3.1.- COMPUTADORA DE PROGRAMA ALMACENADO.

Se define como una máquina capaz de manipular datos acorde a reglas predefinidas (instrucciones), donde el programa (colección de instrucciones) y los datos se almacenan en memoria. Sin algún medio de comunicación con el exterior, el programa y los datos no pueden cargarse en memoria, ni los resultados pueden salir al exterior. Por lo tanto, se requiere un dispositivo de entrada/salida como muestra la figura 1.

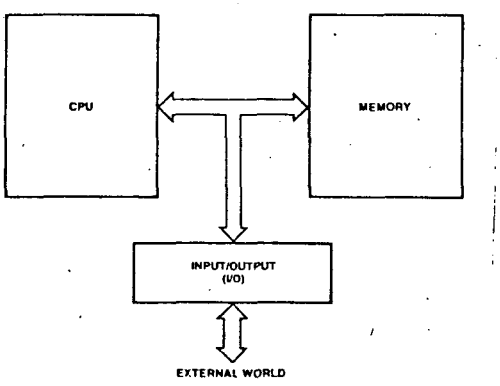


Fig. 1

La memoria está organizada generalmente en palabras, cada una contiene N bits de información. Se le asigna una única dirección para cada palabra, que define su posición relativa respecto a las otras palabras. La Unidad Central de Proceso (CPU) lee o escribe una palabra en un tiempo por direccionamiento de la memoria y luego cuando la memoria está preparada, lee el contenido de la palabra o escribe un nuevo contenido en esa palabra. Para realizar esta operación, se usan dos registros: el Registro de Dirección de Memoria (MAR), el cual contiene las direcciones y el Registro de Datos de Memoria (MDR) el cual contiene los datos. (Fig. 2)

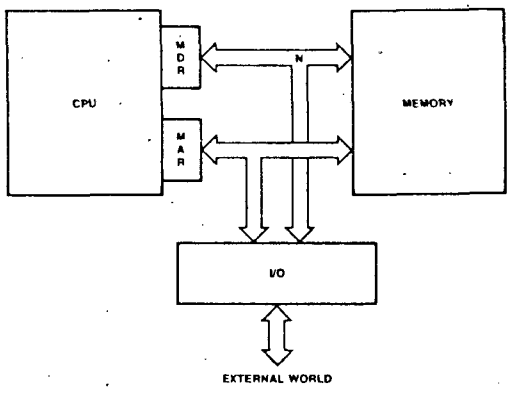


Fig. 2

El acceso a memoria (tanto para leer como para escribir) es generalmente un proceso lento, por esto es ventajoso tener una pequeña localización de memoria dentro de la CPU, en la cual se pueda leer o escribir muy rápido. Estos registros temporales de memoria se llaman Acumuladores o Registros de trabajo. Tienen estos registros gran rapidez de acceso dentro de la CPU, que posibilitan la ejecución de muchas operaciones sin referirse a la memoria (a través del MAR y del MDR) y por lo tanto estas

operaciones se ejecutan más rápido.

La unidad que ejecuta la manipulación de los datos se llama la Unidad Aritmética Lógica (ALU). Esta tiene dos entradas para los operandos y una salida para el resultado. Opera generalmente en paralelo con todos los bits de una palabra. La ALU puede realizar todas o parte de las siguientes operaciones:

ARTIMETICAS	LOGICAS
Suma	OR
Complementación	AND
Sustracción	XOR
Incrementación	NAND
Decrementación	NOR
	XNOR
	Complementación

En algunas arquitecturas, uno de los operandos debe estar siempre en un registro especial (acumulador) y el resultado de la operación en la ALU se transfiere siempre a este registro.

En una CPU más general, cualquiera de los dos registros internos pueden contener los operandos y el resultado de la operación en la ALU puede trasnferirse a cualquiera de ellos.

Otra característica muy útil de la CPU es la capacidad de desplazar el contenido de un registro a la salida de la ALU

uno o más bits en cualquier dirección como se muestra en la Fig. 3.

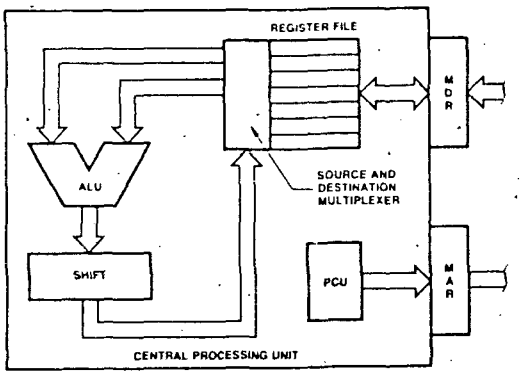


Fig. 3

Ya tenemos los elementos para realizar cualquier manipulación requerida con los datos, pero todavía necesitamos una unidad que pueda colocar adecuadamente el MAR para encontrar la próxima instrucción del programa en la memoria y encontrar sus datos asociados. Esta unidad se llama la Unidad de Control de Programa (PCU) y su misión es cargar el MAR con la dirección correcta para encontrar la próxima instrucción o los datos de partida o localizar un lugar de memoria donde pueda escribirse un dato.

A menudo, los pasos de programas (instrucciones, datos) se almacenan en memoria de una manera secuencial, empezando en la dirección cero o en cualquier otra dirección pre-establecida. En relación con esto la PCU puede incrementarse después de cada acceso a memoria a la dirección de memoria de la próxima instrucción. Este tipo de contador de PCU tiene muy poca flexibilidad. Algunas veces queremos cambiar el flujo normal

de las instrucciones, particularmente si quisiéramos hacer que nuestra computadora "tomara decisiones" acorde a condiciones que predominan en el punto de ejecución. Por ejemplo, queremos que pueda ejecutar una o dos secuencias diferentes de instrucciones dependiendo del resultado de la última operación realizada. Esto se llevará a cabo cargando el MAR con un nuevo valor (la dirección de la próxima instrucción a ser ejecutada) al contrario que incrementarlo. Esta operación se llama RAMA o SALTO y puede ser incondicional (que permite la ejecución de una cadena no contigua de instrucciones) o condicional (dependiendo, por ejemplo, si la última operación fue cero o no, fue negativa o positiva, verdadera o falsa, etc.)

Se puede conseguir una mayor flexibilidad usando un stack (un grupo de lugares de memoria temporales internas o externas) para almacenar datos vitales. Un Stack Pointer se utiliza para direccionar el luagr de memoria del comienzo del stack. Ya discutiremos más tarde los direccionamientos indirectos y relativos, así como otros modos de direccionamiento más sofisticados.

La ejecución de una instrucción en nuestra computadora requiere ahora los siguientes pasos:

a) El PCU carga la dirección de la próxima instrucción en el MAR y señala a la memoria que una lectura es requerida. La memoria carga el MAR con el contenido del lugar direccionado.

b) La CPU decodifica la instrucción: extraer los operandos que están en registros internos, seleccionar los registros adecuados para alimentar la ALU, seleccionar la función a ejecutar por la ALU, activar el shifter para desplazar

el resultado y seleccionar el registro enel cual puede alcanzarse el resultado.

c) La ALU ejecuta la función deseada.

d) El resultado se carga en el registro destinado.

e) El resultado se examina para determinar si hay que realizar algún salto.

f) La PCU calcula la dirección de la siguiente instrucción (generalmente llamada "FETCH").

Este procedimiento será más complicado si los operandos no están almacenados en los registros internos o si el resultado no se almacena en uno de ellos.

Como podemos suponer, se necesitarían muchos más pasos para ejecutar una simple adición usando el direccionamiento relativo.

Obviamente nuestra CPU necesita alguna clase de "coordinador" que pueda:

1) Decodificar una instrucción traída de la memoria.

2) Iniciar el ciclo de pasos para ejecutarse.

3) Activar los controles necesarios para cada paso.

4) Ejecutar los pasos en un orden secuencial.

5) Tomar decisiones teniendo en cuenta el estado de varias señales (condiciones).

A este coordinador lo llamaremos la Unidad de Control del

Computador y está representado en la Fig. 4.

Ahora nuestra CPU está completa (más o menos) y entraremos en detalle más tarde.

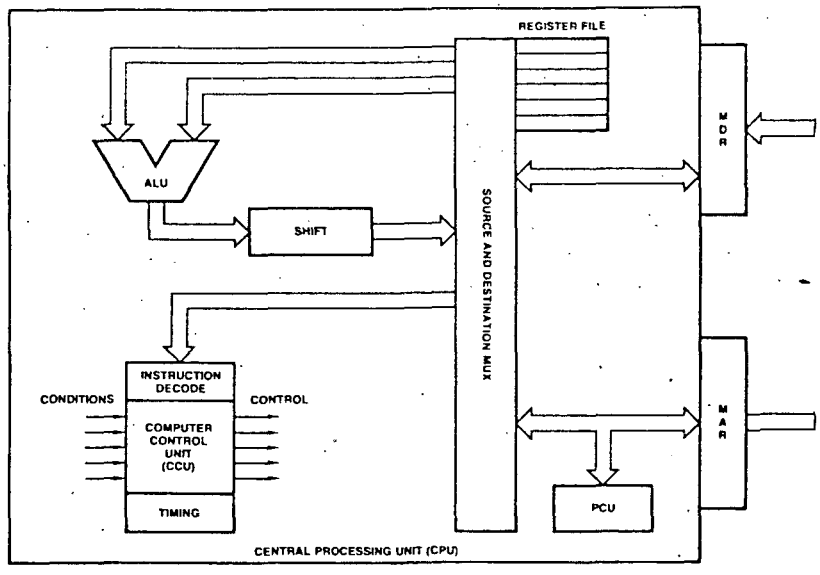


Fig. 4

3.2.- LA MEMORIA.

La información almacenada en la memoria está organizada en palabras, donde cada palabra contiene N bits. Este número de bits puede ser tan pequeño como 8 bits para procesadores muy simples, o tan extenso como 64. El ancho más común para minicomputadores es de 16 bits. El número N se llama al ancho de memoria y el número de bits, por lo tanto en el MAR es también N, igual al ancho de memoria.

La capacidad de una memoria es el número de palabras que contiene. Si el MAR tiene K bits, se pueden direccionar 2^K lugares consecutivos de memoria. Las direcciones van entonces

desde cero hasta $2^k - 1$.

El tiempo de acceso de lectura de una memoria directamente accesible por la CPU es el tiempo que se necesita desde que se establece la dirección de memoria hasta que el dato se almacena en el MAR. Este tiempo de acceso depende del tipo de memoria que se utiliza y puede ser tan corto como unas decenas de nanosegundos y tan largo como algunos microsegundos.

No todos los programas y los datos asociados necesitan residir en la memoria al mismo tiempo. Podemos tener el programa principal (o sólo parte de él) en la memoria mientras los otros programas o filas de datos pueden residir en cualquier otra parte y pueden ser traídos a la memoria cuando sean requeridos. Este "cualquier otra parte" puede ser una cinta magnética, un cassette, disco, diskette, etc., y se llamarán memorias de masa. Estas se caracterizan por:

1) Gran capacidad.
2) No volátilies (retienen la información cuando no se usan).
3) Gran tiempo de acceso.
4) No son aleactoriamente accesibles.
5) Baratas (por bit).

El dispositivo que se encarga de comunicar la memoria principal con las memorias auxiliares se llama control de acceso directo a la memoria (DMA controller).

3.3.- UNIDADES PERIFERICAS.

En cualquier máquina útil se necesita algún medio de comunicación con el exterior. Este puede ser un pupitre, un CRT,

una tarjeta perforada, o en control de procesos, lectura de sensores o colocar un actuador. El denominador común de casi todos los dispositivos de entrada/salida es que son mucho más lentos que la CPU y por lo tanto aparecen problemas de sincronismo. La CPU debe saber cuándo el dispositivo de I/O está preparado para la transferencia de datos. Generalmente, el dispositivo envía una señal a la CPU para demandar su atención. La CPU puede hacer una o dos cosas:

1) Testear esta señal periódicamente y cuando ésta esté presente, salta a un programa que se encarga de la transferencia de los datos. Este tipo de operación se llama "Polling". Esta técnica tiene dos inconvenientes: primero, se pierde un tiempo apreciable en ejecutar estos tests periódicos, porque la mayor parte del tiempo la señal de "preparado" no está presente. Segundo, el reconocimiento por la CPU de la aparición de una señal se retarda hasta que la CPU llega a este dispositivo en su secuencia de Polling (escrutinio).
   Imaginen qué ocurriría si hubiese un gran número de dispositivos de I/O. Ocurrirían muchos retardos, si muchos dispositivos de I/O estuvieran conectados simultáneamente.

2) Incluir algo de hardware en la CPU de tal manera que pueda sentir la presencia de una señal de "preparado" (Ready) e interrumpir el flujo normal de las instrucciones y forzar al computador a "saltar" al programa de servicio de I/O cuando sea requerido. Este puede mandar a la CPU los diferentes programas acorde con el dispositivo de I/O cuyo flag de "ready" sea detectado y establecer una prioridad entre los distintos dispositivos si más de uno de

ellos quisiera obtener la atención de la CPU al mismo tiempo. Además, bajo el programa de control este circuito puede ignorar alguna o todas las señales si la CPU no debe interrumpirse en todo el tiempo.

Ya vemos que pagando en precio de un poco de hardware, ganamos enormenente en la ejecución de la computadora. Llamaremos a este hardware el Controlador de Interrupción.

Nuestra computadora incluye ya la ALU, los registros internos y el circuito de desplazamiento en un solo bloque, que "llamaremos "Unidad Procesadora Aritmética", como muestra la Fig. 5.

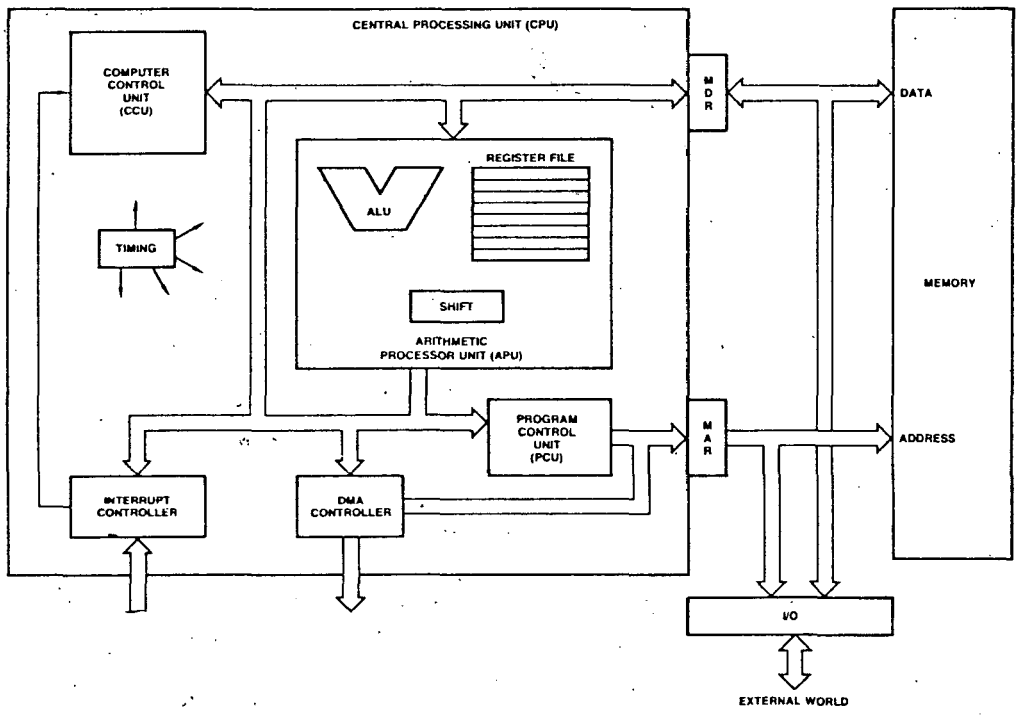


Fig. 5

4.- LA MICROPROGRAMACION.

Hemos visto que la ejecución de uns instrucción obedecía a un encadenamiento de micro-órdenes encargadas de preparar la ruta de datos y de gobernar las unidades funcionales y las memorias.

La microprogramación consiste en reemplazar el secuenciador cableado por un secuenciador programado. A cada instrucción del calculador corresponde generalmente en una memoria especializada, llamada Memoria de Control, un microprograma, cuyo desarrollo genera las micro-órdenes que gobiernan la ejecución de las instrucciones. Se llama microinstrucciones a las instrucciones del microprograma y la ejecución de una microinstrucción genera una o varias micro-órdenes.

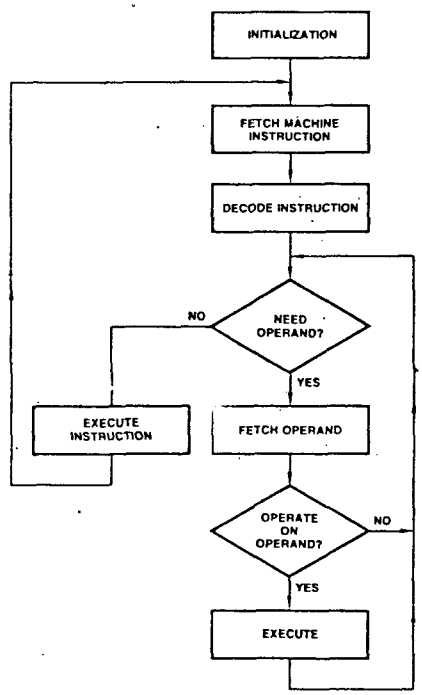
Una microinstrucción consta de dos partes:

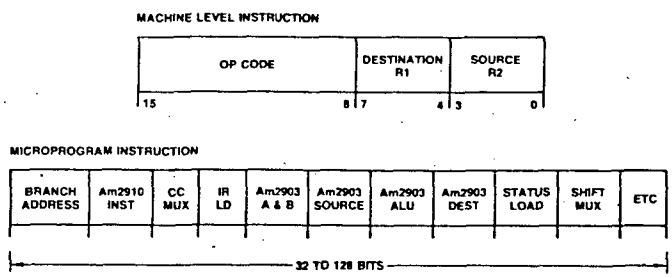
a) La generación y el control de todas las micro-órdenes a ser llevadas a cabo.
b) La definición de la dirección de la próxima microinstrucción que se va a ejecutar.

El OP-CODE (tipo de instrucción a ejecutarse por el computador) se carga en el Registro de Instrucción y el Decodificador de Instrucción lo decodifica. Este genera la microdirección donde residirá el primer paso de la secuencia de ejecución de la instrucción en la memoria de control (Memoria del Microprograma). El secuenciador AM 2910 genera entonces la microdirección de la próxima instrucción. Los operandos del microprograma suministran las señales de control que se necesitan para controlar todas las partes del ordenador, incluyendo el secuenciador.

Cuando todos los pasos de una instrucción de máquina se ejecutan, el microprograma "fetch", la lectura de la próxima instrucción de máquina desde la memoria principal.

La CCU se usa para ir a buscar (fetch) las instrucciones y decodificarlas usando una PROM para organizar (mapping) el OP CODE para la dirección inicial de la secuencia de microinstrucciones que se usan para ejecutar esta instrucción particular. También buscaría (fetch) todos los operandos necesitados por la instrucción de máquina y los entregaría a la ALU para su procesamiento. Un ejemplo del flujo de la CCU típica, sería el de la Fig. 6.





Figs. 6 y 7

Supongamos que:

El OP CODE de la instrucción de máquina tiene 8 bits de ancho. Esto permite ejecutar un mínimo de 256 instrucciones diferentes. Imaginemos también que se necesitan un promedio de 6 pasos para ejecutar estas instrucciones. Aunque se usaran lugares separados para la memoria del microprograma, la capacidad de memoria de esta memoria sería solamente $1-\frac{1}{2}k$ ($k = 1024$). Pero en este caso el secuenciador puede ser reemplazado por un simple contador. Generalmente nos gustaría dividir algunas micro-rutinas entre diferentes instrucciones. Por otro lado es importante entender la diferencia entre instrucciones de máquina e instrucciones del microprograma. La Fig. 7 nos muestra una típica instrucción de máquina de una minicomputadora de 16 bits que tiene un OP CODE de 8 bits para identificar una de las 256 instrucciones, 4 bits de especificación de registros fuente para identificar uno de los 16 registros fuente y 4 bits de especificación de registros de destino para identificar uno de los 16 registros de destino. La instrucción de microprograma de la Fig. 7 puede contener desde 32 a 128 bits en un diseño normal o aún más. Esta microinstrucción contendrá campos para el suministro de operandos para la ALU, función de la ALU, destino del resultado de la ALU, control de desplazamiento del multiplexor, ciclo de control de bus, etc. Estos campos se utilizan para controlar los distintos dispositivos de la máquina. A este tipo de codificación se denomina Codificación por Campos. Se dividen las distintos micro-órdenes en grupos independientes, de tal suerte que sea imposible que, en cada grupo, se den micro-órdenes simultáneas. Tiene la ventaja de una decodificación más directa y el inconveniente de una microinstrucción más larga. Facilita a los programadores expertos en la estructura de la máquina una mejor optimi

zación de los microprogramas.

5.- DESCRIPCION DE LOS ELEMENTOS UTILIZADOS.

Los elementos elegidos para el desarrollo son:

AM 2903 Control Processing Element
AM 2902 A Look-ahead Carry Adder
AM 2910 Microprogram Control Unit

5.1.- AM 2902 A Look-ahead Carry Adder.

Antes de meterse de lleno en describir este elemento, me gustaría explicar primeramente lo que es un "basic full adder", para así entender mejor la utilización y el uso del AM 2902 A Look-ahead Carry Adder.

5.1.1.- Basic Full Adder.

Los resultados de una operación aritmética unitaria no sólo dependen de los dos bits que ocupan esa posición, sino también de todos los bits menos significativos de las dos variables de entrada. El resultado final para cualquier bit, por lo tanto, no se podrá obtener hasta que los acarreos de todos los bits anteriores no se hayan activado (Ripple) a través del dispositivo comenzando por el bit menos significativo y propagándose hasta el bit de mayor peso. Un Full Adder es un dispositivo que admite dos bits individuales con el mismo peso y también acepta un bit de entrada de acarreo procedente del Full Adder del peso menos significativo inmediatamente anterior. El Full Adder ejecuta la suma de los dos bits del mismo peso y produce también un bit de acarreo como entrada en el siguiente Full Adder de peso inmediatamente superior.

La tabla de verdad de un Full Adder es la siguiente:

A y B: Datos de Entrada
C: Acarreo de Entrada
} Entradas

S: Resultado
Co: Acarreo generado
} salidas

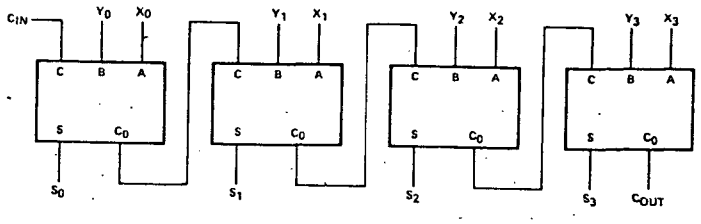
ENTRADAS			SALIDAS	
A	B	C	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Cuyas ecuaciones son:

$$S = A \oplus B \oplus C$$

$$Co = AB + BC + AC$$

El resultado de la suma, S, representa la suma de los operandos de entrada A y B, y del acarreo de entrada. El acarreo de salida, Co, representa el acarreo de salida de esta célula y podrá usarse como acarreo de entrada en la siguiente célula de mayor peso del sumador. Las células del Full-adder se pueden conectar en cascada como se ve en la figura y formar así un Four



bit Ripple Carry Parallel Adder.

Nótese que una ves que tenemos estos dispositivos en cascada, como en la figura, podemos intentar darle un sentido matemático a lo que sucede en el bit i, y si pensamos un poco llegamos a la conclusión de que:

$$Si = Ai \oplus Bi \oplus Ci$$

$$C_{i+1} = AiBi + BiCi + AiCi$$

Donde Ai y Bi son los operandos de entrada en el bit i, y Ci es el acarreo de entrada para ese mismo bit. Tenemos que darnos cuenta que las ecuaciones para este sumador son iterativas por naturaleza y cada una de ellas depende del resultado de los anteriores bits menos significativos de la cadena de sumadores.

El problema que se nos plantea más inmediatamente con este tipo de conexión es el tiempo de propagación que se necesita para analizar una suma de 16 bits, ya que cada célula debe esperar el resultado del acarreo generado en la célula inmediatamente anterior para poder llevar a cabo su suma. Para subsanar este inconveniente tenemos el Carry Lookahead Adder o Sumador de Acarreo Adelantado.

5.1.2.- AM 2902 A: Carry Lookahead Adder.

Mirando hacia atrás y viendo las ecuaciones desarrolladas para el bit i de un sumador, volvamos a escribir la ecuación del acarreo de una forma un poco diferente:

$$C_{i+1} = AiBi + BiCi + AiCi = AiBi + Ci\ (Ai + Bi)$$

Ahora definamos dos ecuaciones adicionales:

$$Gi = AiBi$$

$$Pi = Ai + Bi$$

Quedando entonces la ecuación del acarreo:

$$Ci+1 = Gi + PiCi$$

A estos nuevos términos, Gi y Pi, vamos a llamarles Carry Generate (Acarreo Generado) y Carry Propagate (Acarreo Propagado) respectivamente. Se puede generar un acarreo adelantado en cualquier estado del sumador, implementando la ecuación de arriba, tanto si es debido a las funciones Gi o Pi. También es interesante que la ecuación de la suma se pueda poner en función de estos términos:

$$Si = Ai \oplus Bi \oplus Ci = (Ai + Bi)(AiBi) \oplus Ci = PiGi \oplus Ci$$

Ahora vamos a detenernos un poco, para explicar por qué a Gi y a Pi les hemos dado esos nombres determinados y ver las consecuencias que se pueden generar debido a estos términos.

La función auxiliar Gi se llama Acarreo Generado (Carry Generate), porque si es verdad, entonces un acarreo se produce inmediatamente para el próximo estado del sumador. La función Pi se llama Acarreo Propagado (Carry Propagate) porque implica que habrá un acarreo dentro del próximo estado del sumador. Esto es, Gi, genera una señal de acarreo en el estado i del sumador, para ser generado y presentado al próximo estado del sumador, mientras que Pi causa un acarreo que existe en la entra-

da del estado i del sumador para propagarlo al próximo estado del sumador.

Escribamos ahora todas las ecuaciones de suma y acarreo requeridas para un Full Four-bit Lookahead Carry Adder:

$$S_o = A_o \oplus B_o \oplus C_o$$

$$S_1 = A_1 \oplus B_1 \oplus (G_o + P_oC_o)$$

$$S_2 = A_2 \oplus B_2 \oplus (G_1 + P_1G_o + P_1P_oC_o)$$

$$S_3 = A_3 \oplus B_3 \oplus (G_2 + P_2G_1 + P_2P_1G_o + P_2P_1P_oC_o)$$

$$C_{i+4} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_o + P_3P_2P_1P_oG_o$$

Podríamos poner todas estas ecuaciones en función de Ai, Bi y Co, con lo cual podríamos hacer un diagrama a base de puertas e inversores que cumpla estas ecuaciones.

El dispositivo quedaría:

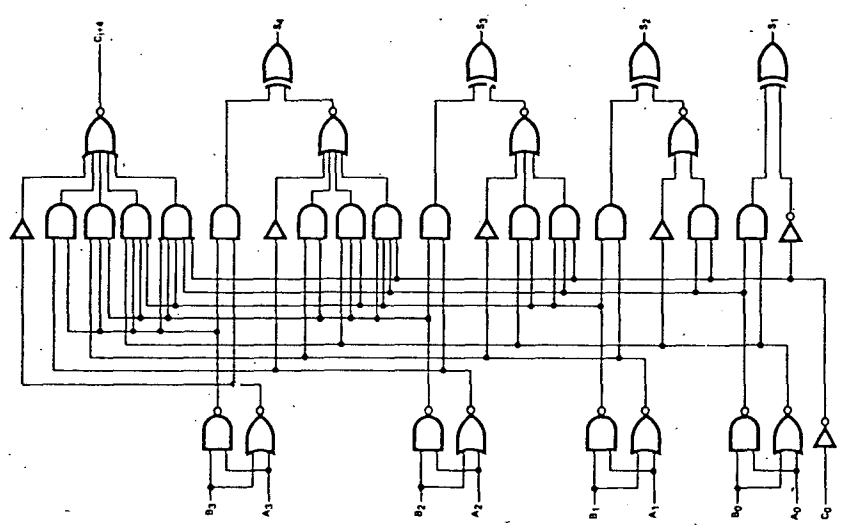


Fig. 8

FULL FOUR BIT CARRY-LOOKAHEAD ADDER

Diferenciamos dos partes bien definidas en este esquema: por un lado, las puertas que generan las funciones auxiliares Pi y Gi, a partir de Ai, Bi y por otra parte, las puertas que generan la suma de cada posición.

Un serio problema que se le plantea a este dispositivo es que la longitud de la palabra se incremente. La función carry, sería muy complicada, llegándose a hacer impracticable un dispositivo como el estudiado anteriormente, debido al gran número de conexiones que se necesitarían y a que las funciones auxiliares Gi, Pi serían enormes. pero el concepto de función auxiliar se puede extender, sin embargo, si dividimos la longitud de palabras en razonables proporciones y definimos así bloques de funciones auxiliares G y P.

Es posible, para un bloque dado, definir una función G como el acarreo de salida generado por el bloque; y P se puede definir como el acarreo propagado sobre el bloque. Si el tamaño del bloque es de cuatro bits, entonces las funciones G y P para este bloque se definen como siguen:

$$G = G3 + P3G2 + P3P2G1 + P3P2P1Go$$

$$P = P3P2P1Po$$

Es importante darse cuenta que ninguno de estos términos depende de Co (Carry-in).

Las funciones G y P se pueden "puertear" para producir un acarreo de entrada para cada bloque de cuatro bits, como una función de los bloques menos significativos. El acarreo de entra

sa para un bloque es:

$$C_n = G_{n-1} + P_{n-1}\,G_{n-2} + P_{n-1}\,P_{n-2}\,G_{n-3} + \ldots + P_{n-1}\,P_{n-2}\ldots P_2 P_1 P_0 G_0$$

Finalmente, el acarreo de entrada para cada uno de los bits en un bloque de cuatro bits debe incluir un término para el actual acarreo de una entrada menos significativo. Nótese por lo tanto, que las ecuaciones para el sumador completo presentadas anteriormente incluyen un término de acarreo de entrada para cada posición.

Los siguientes esquemas muestran la típica configuración bit-slice de cuatro ALU AM 2903. El primer esquema es un diseño de 16 bits de alta velocidad, utilizando el generador de acarreo adelantado AM 2902 A.

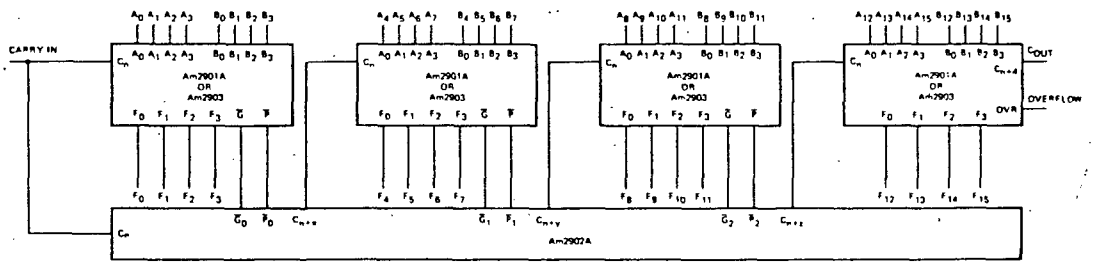


Fig. 9

La segunda conexión es de tipo Ripple-carry y entraña una mayor lentitud en sus operaciones. Se suele utilizar en dispositivos en los que la velocidad no es el objetivo principal.

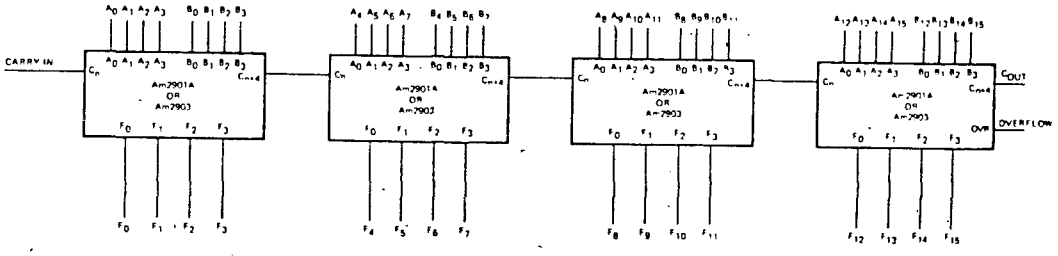


Fig. 10

5.2.- AM 2910 MICROPROGRAM CONTROLLER.

5.2.1.- ARQUITECTURA.

El AM 2910 es un controlador de microprograma que se usa en microprocesadores de alta velocidad. Permite direccionamientos superiores a 4K palabras de microprograma. El diagrama de bloques es:

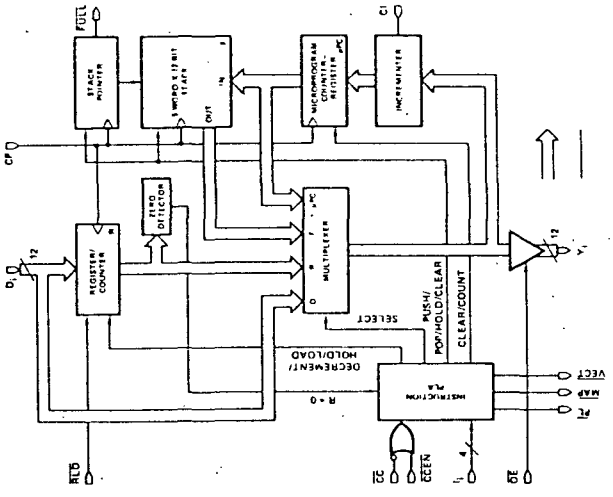


Fig. 11

El controlador contiene un multiplexor de 4 entradas que se utiliza para seleccionar el Register/Counter, o una entrada directa, o el contador del microprograma o el stack o la fuente de la dirección de la próxima instrucción.

El Register/Counter consta de 12 entradas tipo D (Direct Input Bit i) asociadas a otros tantos flip-flops disparados por flanco, con un reloj. Cuando su carga de control, $\overline{RLD}$ es LOW, un nuevo dato se carga en una transición positiva de reloj. Cuando su carga de control, $\overline{RLO}$ es baja (LOW), se carga un nuevo dato en una transición positiva de reloj.

La salida del Register/Counter está dispuesta como una fuente de la dirección de la próxima instrucción y está conectada con una de las entradas del multiplexor. La entrada directa suministra una fuente de datos para cargar el Register/Counter.

El AM 2910 contiene además un contador de microprogramas (μPC) que está conpuesto por un incrementador de 12 bits, seguido por un registro de 12 bits. El μPC se puede usar de cualquiera de las dos maneras. Cuando el acarreo de entrada para el incrementador es HIGH, el registro del microprograma se carga en el prócimo ciclo de reloj con la palabra de salida Y más uno ($Y + 1 \rightarrow \mu PC$). Así se ejecuta la secuencia de microinstrucciones.

Cuando el acarreo de entrada es LOW, el incrementador pasa la palabra de salida Y sin modificar, cargándose de nuevo el μPC con la misma palabra en el próximo ciclo de reloj ($Y \rightarrow \mu PC$). De esta manera se ejecuta la misma microinstrucción cualquier número de veces.

La tercera fuente del multiplexor es la entrada directa (D). Esta fuente se utiliza en bifurcaciones. (BRANCH)

La cuarta fuente disponible en la entrada del multiplexor es un stack de 5 palabras por 12 bits. El stack se usa para proveer las direcciones de retorno cuando se ejecutan subrutinas o lazos. El stack contiene un Stack Pointer (SP) el cual siempre señala la última fila de palabras escrita. El Stack Pointer opera como un control UP/DOWN. Durante las microinstrucciones 2, 4 y 5, se ejecuta la operación PUSH. Esto motiva que el Stack Pointer se incremente y que la fila se escriba con el retorno requerido. En el ciclo siguiente a la operación PUSH, los datos de retorno están en una nueva localización señalada por el Stack Pointer.

Durante otras seis microinstrucciones, ocurre una operación POP. Esta coloca la información en lo alto del stack sobre la salida Y el Stack Pointer se decrece en el siguiente ciclo de reloj (flanco de subida) siguiendo la operación POP, removiendo eficazmente toda la información antigua en lo alto del stack.

La encadenación del Stack Pointer es tal que se puede ejecutar cualquier secuencia de operaciones PUSH, POP o REFERENCE STACK. En RESET (instrucción 0), la capacidad de almacenamiento se hace cero. Por cada PUSH, la capacidad de almacenamiento se incrementa en uno; por cada POP, la capacidad se decrece en uno. La capacidad puede crecer hasta cinco. Cuando se alcanza esta capacidad, entonces FULL se pone a LOW. Cualquier PUSH ulterior sobre un stack lleno, sobreescribe la información en lo alto del stack, pero no altera el Stack Pointer. Esta operación destruirá posiblemente una información útil y por supuesto

no es deseado. Un POP de un stack vacío coloca unos datos no significativos en la salida Y, pero por otro lado, es seguro. El Stack Pointer permanece en cero siempre que se intente un POP desde un stack ya vacío. El Register/Counter opera durante tres microinstrucciones (8, 9, 15) como un contador descendentemente de 12 bits, con un resultado igual a cero disponible como una microinstrucción de bifurcación para cualquier test.

Esto facilita la repetición de microinstrucciones. El Register/Counter se ordena de tal manera que si se precarga con un número N y luego se utiliza como un contador de terminación de lazo, la secuencia se ejecutará exactamente N + 1 veces. Durante la instrucción 15 se queda disponible el código de condición. El dispositivo provee las salidas Y en Three-State. Esto es bastante útil en diseños que requieren comprobaciones automáticas del procesador. Las salidas del controlador del microprograma se pueden forzar a un estado de alta impedancia, y se pueden ejecutar secuencias preprogramadas de microinstrucciones desde el exterior a las líneas de dirección.

Abbreviation	Name	Function
D_i	Direct Input Bit i	Direct input to register/counter and multiplexer. D_0 is LSB
I_i	Instruction Bit i	Selects one-of-sixteen instructions for the Am2910
$\overline{CC}$	Condition Code	Used as test criterion. Pass test is a LOW on $\overline{CC}$.
$\overline{CCEN}$	Condition Code Enable	Whenever the signal is HIGH, $\overline{CC}$ is ignored and the part operates as though $\overline{CC}$ were true (LOW).
CI	Carry-In	Low order carry input to incrementer for microprogram counter
$\overline{RLD}$	Register Load	When LOW forces loading of register/counter regardless of instruction or condition
$\overline{OE}$	Output Enable	Three-state control of Y_i outputs
CP	Clock Pulse	Triggers all internal state changes at LOW-to-HIGH edge
V_{CC}	+5 Volts	
GND	Ground	
Y_i	Microprogram Address Bit i	Address to microprogram memory. Y_0 is LSB, Y_{11} is MSB
$\overline{FULL}$	Full	Indicates that five items are on the stack
$\overline{PL}$	Pipeline Address Enable	Can select #1 source (usually Pipeline Register) as direct input source
$\overline{MAP}$	Map Address Enable	Can select #2 source (usually Mapping PROM or PLA) as direct input source
$\overline{VECT}$	Vector Address Enable	Can select #3 source (for example, Interrupt Starting Address) as direct input source

TABLA 1

5.2.2.- OPERACION

La Tabla 2 muestra el resultado de cada instrucción de control del multiplexor que determina las salidas Y, y del control de las tres señales habilitadas $\overline{PL}$, $\overline{MAP}$ y $\overline{VECT}$. Se muestra también el efecto en el μPC, en el Register/Counter, y en el stack después del siguiente ciclo positivo de reloj.

Es el multiplexor el que determina qué fuente interna es la que va a gobernar las salidas Y. El valor que se le ha cargado en el μPC es en todo caso idéntico al de la salida Y, o aumentado en uno, según esté determinado por CI.

HEX I_3-I_0	MNEMONIC	NAME	REG/ CNTR CONTENTS	FAIL $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH		PASS $\overline{CCEN}$ = HIGH or $\overline{CC}$ = LOW		REG/ CNTR	ENABLE
				Y	STACK	Y	STACK		
0	JZ	JUMP ZERO	X	0	CLEAR	0	CLEAR	HOLD	PL
1	CJS	COND JSB PL	X	PC	HOLD	D	PUSH	HOLD	PL
2	JMAP	JUMP MAP	X	D	HOLD	D	HOLD	HOLD	MAP
3	CJP	COND JUMP PL	X	PC	HOLD	D	HOLD	HOLD	PL
4	PUSH	PUSH/COND LD CNTR	X	PC	PUSH	PC	PUSH	Note 1	PL
5	JSRP	COND JSB R/PL	X	R	PUSH	D	PUSH	HOLD	PL
6	CJV	COND JUMP VECTOR	X	PC	HOLD	D	HOLD	HOLD	VECT
7	JRP	COND JUMP R/PL	X	R	HOLD	D	HOLD	HOLD	PL
8	RFCT	REPEAT LOOP, CNTR ≠ 0	≠ 0	F	HOLD	F	HOLD	DEC	PL
			= 0	PC	POP	PC	POP	HOLD	PL
9	RPCT	REPEAT PL, CNTR ≠ 0	≠ 0	D	HOLD	D	HOLD	DEC	PL
			= 0	PC	HOLD	PC	HOLD	HOLD	PL
A	CRTN	COND RTN	X	PC	HOLD	F	POP	HOLD	PL
B	CJPP	COND JUMP PL & POP	X	PC	HOLD	D	POP	HOLD	PL
C	LDCT	LD CNTR & CONTINUE	X	PC	HOLD	PC	HOLD	LOAD	PL
D	LOOP	TEST END LOOP	X	F	HOLD	PC	POP	HOLD	PL
E	CONT	CONTINUE	X	PC	HOLD	PC	HOLD	HOLD	PL
F	TWB	THREE-WAY BRANCH	≠ 0	F	HOLD	PC	POP	DEC	PL
			= 0	D	POP	PC	POP	HOLD	PL

Note: If $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH, hold; else load. X = Don't Care.

TABLA 2 AM 2910 INSTRUC. SET.

Por cada instrucción, una y sólo una de las tres salidas, $\overline{PL}$, $\overline{MAP}$ o $\overline{VECT}$, es LOW. Si estas salidas de control (Three-State) permiten que la fuente primaria del microprograma salte (Generalmente, parte de un registro Pipeline), una PROM que organiza la instrucción para la localización del comienzo de la microinstrucción, y una tercera fuente opcional (a menudo, un vec

tor de un DMA o una fuente de interrupción) respectivamente, las fuentes en Three-State pueden gobernar las entradas D sin ninguna lógica adicional.

Algunas entradas, tal como muestra la Tabla 2, pueden modificar la ejecución de una instrucción. La combinación $\overline{CC}$ HIGH y $\overline{CCEN}$ LOW se usa como un test en 10 de 16 instrucciones. Cuando $\overline{RLD}$ es LOW, da origen a que las entradas se carguen en el Register/Counter, en especial cualquier operación HOLD o DEC que se especifiquen en la instrucción.

$\overline{OE}$, que generalmente está en LOW, se puede forzar a HIGH para sacar las salidas Y del AM 2910 de un bus en Three/State.

El stack tiene un señalador que direccina el valor que está en lo alto del stack. El control explícito del Stack Pointer se origina durante la instrucción o (Reset), la cual hace que el stack se vacíe al poner SP a cero. Después de un RESET, y siempre que el stack permanezca vacío, el contenido en lo alto del stack no está definido hasta que se origina un PUSH. Cualquier operación POP ejecutada mientras el stack está vacío, coloca una serie de datos indefinidos sobre las salidas F y deja el Stack Pointer a cero. Si en algún momento el stack se llena (se habrán sucedido cinco operaciones PUSH más que POP desde que el stack estaba vacío), actuará la señal de aviso $\overline{FULL}$. Cuando esto suceda, no se podrá intentar más operaciones PUSH, debido a que como comentábamos en el apartado anterior, la información que ocupa el luagr de encabezamiento en el stack, se pierde.

5.2.3.- JUEGO DE INSTRUCCIONES DEL AM 2910.

El AM 2910 está provisto de 16 instrucciones que selecionan

la siguiente microinstrucción que se va a ejecutar. Cuatro de estas instrucciones son incondicionales, esto es, su efecto depende únicamente de la instrucción. Otras diez instrucciones tienen un efecto parcialmente controlado por una condición externa dependiente de los datos. Tres de estas instrucciones, a su vez, tienen un efecto parcialmente controlado por los contenidos internos del Register/Counter. El juego de instrucciones se muestra en la Tabla 2 . Para la explicación que a continuación voy a detallar, voy a suponer que CI está siempre a HIGH.

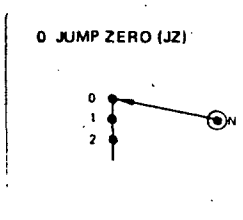
En las diez instrucciones condicionales, el resultado del test de dependencia de datos se aplica a $\overline{CC}$. Si la entrada de $\overline{CC}$ es LOW, se considera que el test se ha superado y entonces se lleva a cabo la acción especificada por el nombre. Si por el contrario, el test falla, se produce una alternativa (a menudo simplemente la ejecución de la siguiente secuencia de microinstrucciones). Si $\overline{CCEN}$ se encuentra en HIGH, no podremos testear a $\overline{CC}$, puesto que si $\overline{CCEN}$ está a HIGH, fuerza incondicionalmente la acción especificada en el nombre, esto es, se fuerza un paso. Otra manera de usar $\overline{CCEN}$ es, o ponerla permanentemente a HIGH, lo cual es muy útil si ninguna microinstrucción depende de los datos, o ponemos $\overline{CCEN}$ a LOW si las instrucciones dependientes de los datos no se fuerzan nunca incondicionalmente; o si lo conectamos al bit lo del AM 2910, lo cual deja a las instrucciones 4, 6 y 10 como dependientes de los datos, pero hace otras como incondicionalmente. Todos estos trucos ahorran un bit del ancho del microcódigo.

El efecto de las tres instrucciones depende del contenido del Register/Counter. A menos que el contador esté a cero debido a un decremento. Si se mantiene a cero, se selecciona la direc-

ción del siguiente microprograma diferente. Estas instrucciones son útiles para ejecutar un lazo de una microinstrucción ur número conocido de veces. La instrucción 15 está doblemente afectada por el código de condición externo y por el contenido interno del Register/Counter.

Pasemos ahora a explicar cada instrucción y revisar sus operaciones. Acompañaré también a esta explicación con algunos ejemplos gráficos para que se entienda mejor la operación de cada ura de las instrucciones.

- Instrucción 0, J Z (JUMP y ZERO o RESET): Incondicionalmente se especifica que la dirección de la próxima micro instrucción es cero.



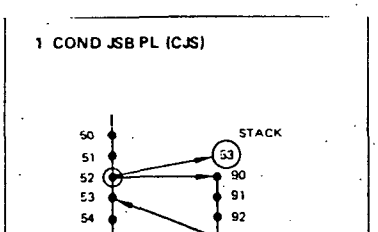


Fig. 12 Fig. 13

- Instrucción 1 es un CONDITIONAL JUMP-TO-SUBROUTINE (Salto condicional a subrutina) a la dirección especificada en el registro PIPELINE. Como se muestra en la Figura 13, la máquina debe haber ejecutado las palabras en las direcciones 50, 51 y 52. Cuando los contenidos de la dirección 52, están en el registro PIPELINE, la siguiente dirección de la función de control es el CONDITIONAL JUMP TO SUBROUTINE. Aquí, si se ha pasado el

test, la próxima instrucción a ejecutar debe ser el contenido del lugar 90 de la memoria del microprograma. Si por el contrario, el test ha fallado, el JUMP TO SUBROUTINE no se ejecutará; el contenido del lugar 53 de la memoria del microprograma se ejecuta al instante. De este modo, la instrucción CJS en el lugar 52, originará que se produzca la instrucción del lugar 90 o la del lugar 53. Si la entrada del test es tal que se seleccione el lugar 90, el valor 53 se colocará en el stack interno. Este está provisto de un retorno lincado para la máquina cuando se complete la subrutina que comenzaba en el lugar 90. En este ejemplo, la subrutina se completó en el lugar 93 y un RETURN FROM SUBROUTINE se encontraría en dicho lugar.

- Instrucción 2, es el JUMP MAP. Esta es una instrucción incondicional que hace que la salida MAP se habilite de tal manera que el lugar de la próxima microinstrucción esté determinada por la dirección que se aplique mediante lo estipulado en los PROMS. Generalmente la instrucción JM se utiliza al final de una secuencia de instrucción de búsqueda por la máquina. En el ejemplo de la Figura 14, las microinstrucciones localizadas en 50,51,52 y 53, habrán sido la secuencia de búsqueda, la función JM debe estar en el registro PIPELINE. En este ejemplo se muestra que lo estipulado por la PROM es el lugar 90, por lo tanto se ejecuta un salto incondicional a la dirección de la memoria del microprograma cuyo lugar es 90.

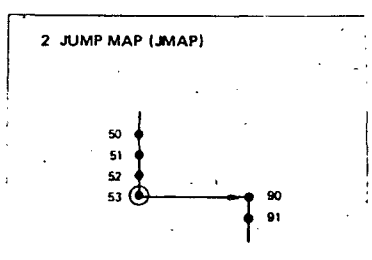


Fig. 14

- Instrucción 3, CONDITIONAL JUMP PIPELINE, deriva su dirección de bifurcación del valor de la dirección de bifurcación contenida en el registro PIPELINE, (BRo - BR11, Fig. 15). Esta instrucción está provista de una técnica que permite bifurcar varias secuencias del microprograma dependiendo de las entradas del test de condición. Los estados de máquina se diseñan, bastante a menudo, de tal manera que sólo ejecutan tests sobre varias entradas, esperando que la condición que viene es verdadera. Cuando la condición de VERDAD se alcanza, entonces la máquina bifurca y ejecuta un juego de microinstrucciones para realizar alguna función. Esto tiene generalmente el efecto de resetear la ENTRADA, que se testea hasta algún punto en el futuro. La fig. 15 muestra el salto condicional por medio de la dirección contenida en el registro PIPELINE en el lugar 52. Cuando los contenidos de la palabra 52 de lamemoria del microprograma están en el registro pipeline, la siguiente dirección será o el lugar 53 ó el 30, según este ejemplo. Si se pasa el test se selecciona el valor presente en el registro pipeline. Si el test falla, se seleccionará la dirección contenida en el contador del microprograma que en este caso es el lugar 53.

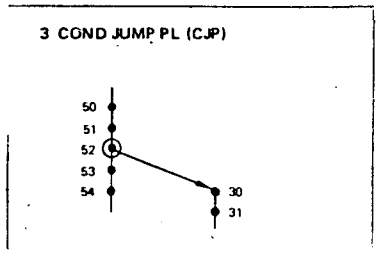


Fig. 15

- Instrucción 4, es la instrucción PUSH/CONDITIONAL LOAD COUNTER (PUSH). Se utiliza primordialmetne para poner en marcha lazos en el FIRWARE (1) del microprograma. En la figura 16, cuando la instrucción 52 está en el registro PIPELINE, se creará un PUSH sobre el STACK y el contador se cargará en base a la condición. Cuando sucede un PUSH, el valor "empujado" es siempre la dirección de la siguiente instrucción secuencial. En este caso, es la dirección 53. Si el test falla, el contador no se cargará; si se pasa, el contador se carga con el valor del contenido de la dirección del campo de bifurcación del registro PIPELINE.

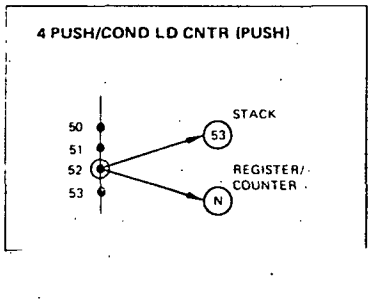


Fig. 16

De este modo, una simple microinstrucción se puede usar para poner en marcha un lazo, parq que se ejecute un número determinado de veces. La instrucción 8 describirá cómo se utiliza el valor "empujado" y el REGISTER/COUNTER

---

(1) FIRMWARE: Por analogía con el HARDWARE y el SOFTWARE, el FIRMWARE designa la microprogramación que, en un sentido estricto, no pertenece completamente al terreno del HARDWARE ni al terreno del SOFTWARE.

para realizar los lazos.

- Instrucción 5, es la instrucción CONDITIONAL JUMP TO SUBROUTINE a través del REGISTER/COUNTER o según el contenido del registro PIPELINE. Como se muestra en la figura 17, siempre se ejecuta un PUSH y una de las dos subrutinas se realiza. En este ejemplo se ejecutará o la subrutina que comienza en al dirección 80, o la que comienza en la dirección 90. Un RETURN FROM SUBROUTINE (instrucción 10) devolverá el flujo de microprograma a la dirección 55.

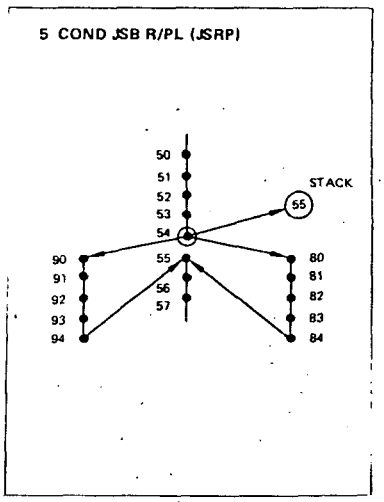


Fig. 17

Para que esta secuencia de control de la microinstrucción opere correctamente, ambos, los siguientes campos de dirección de la instrucción 53 y los siguientes campos de dirección de la instrucción 54 deberían contener el valor adecuado. Supongamos que los campos de dirección de bifurcamiento de la instrucción 53 contienen el valor 90, de modo que estará en el REGISTER/

COUNTER, cuando el contenido de la dirección 54 esté en el registro PIPELINE. Esto requiere que la instrucción de la dirección 53 se cargue en el REGISTER/COUNTER.

Ahora, si durante la ejecución de la instrucción 5 (en la dirección 54), falla el test, el contenido del registro (valor = 90) seleccionará la dirección de la próxima instrucción. Por lo tanto, esta instrucción tiene la facultad de seleccionar una de dos subrutinas para que sea ejecutada, según el test de condición.

- Instrucción 6, es la instrucción CONDITIONAL JUMP VECTOR, la cual tiene la capacidad de tomar la dirección de bifurcación, de una tercera fuente hasta ahora no discutida. Para que esta instrucción sea útil, la salida $\overline{VECT}$ del AM 2910, se usa para controlar la entrada en three-state de un registro, o un buffer, o de una PROM que contenga la dirección del siguiente microprograma. Esta instrucción está provista de una técnica que ejecuta interrupciones del tipo bifurcación a nivel del microprograma.

  Puesto que es una instrucción condicional, si el test pasa, la siguiente dirección se toma de la fuente vector. Si por el contrario, el test no pasa la siguiente dirección de toma del contador de microprograma.

  En el ejemplo de la figura 18, si la instrucción CJV

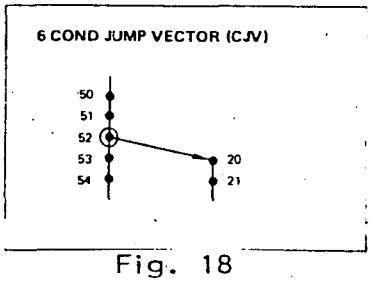


Fig. 18

está contenida en el lugar 52, la ejecución continuará en la dirección 20 del vector; si la entrada TEST es HIGH; si la entrada TEST es LOW, entonces se ejecutará la microinstrucción de la dirección 53.

- Instrucción 7, es la instrucción CONDITIONAL JUMP por medio, o del contenido del REGISTER/COUNTER, o por el contenido del registro PIPELINE. Esta instrucción en muy similar a la instrucción 5 (COND JSB R/PL) la mayor diferencia existente entre estas dos instrucciones es que no se ejecuta ningún PUSH sobre el STACK en la instrucción 7. En la figura 19, se describe esta instrucción como una bifurcación a uno de los dos lugares dependiendo de la condición de test.

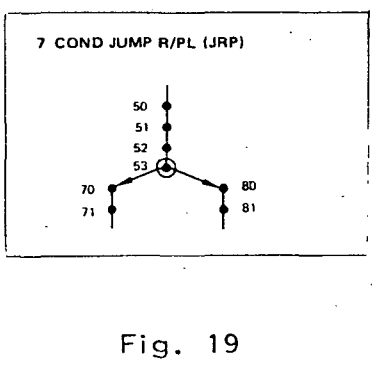


Fig. 19

En el ejemplo se supone que el registro PIPELINE contiene el valor 70 cuando el contenido de la dirección 52 se ejecuta. Como el contenido de la dirección 53 está encerrado dentro del registro PIPELINE, el valor 70 se carga en el REGISTER/COUNTER del AM 2910. El valor 80 estará disponible cuando el contenido de la dirección 53 esté en el registro PIPELINE. Por lo tanto, el control se transfiere o a la dirección 70 o a la direción 80, dependiendo de la condición del test.

- Instrucción 8, es la instrucción REPEAT LOOP, COUNTER ≠ ZERO. Esta microinstrucción hace uso de la capacidad de poder decrecer el REGISTER COUNTER. Para que esta sea útil, algunas instrucciones anteriores, como la 4, debe haber cargado un valor estipulado en el REGISTER/COUNTER. Esta instrucción se detiene para ver si el REGISTER/COUNTER tiene un valor distinto de cero. Si ocurre así, decrece el REGISTER/COUNTER y se toma como próxima microinstrucción el contenido de lo alto del stack. Si por el contrario, el REGISTER/COUNTER tiene un valor cero, se produce la condición de salida del lazo; el control pasa a la seguiente secuencia de microinstrucciones seleccionadas por el μPC; el stack se "POPea" al decrecer el STACK POINTER, y el contenido de lo alto del stack se borra.

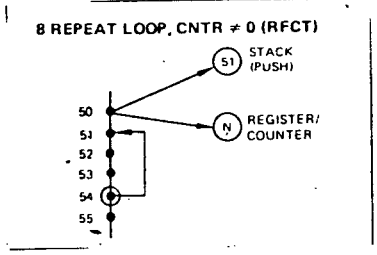


Fig. 20

Un ejemplo de la instrucción REPEAT LOOP, COUNTER ≠ ZERO, es la que se muestra en la figura 20. En este ejemplo, el lugar 50 es el más idóneo para contener una instrucción RFCT, la cual debería originar que la dirección 51 fuese "empujada" en el stack y que el contador se cargue con el valor apropiado para que se generara un lazo, un determinado número de veces.

En este ejemplo, desde que se hace el test de lazo al final de las instrucciones que se repiten (micro dirección 54), el valor apropiado que se carga por la instrucción en la dirección 50 es uno menos que el número de veces que queremos que se genere el lazo. Este método permite que un lazo se ejecute de 0 a 4095 veces.

- Instrucción 9, es la instrucción REPEAT PIPELINE REGISTER, COUNTER ≠ ZERO. Esta instrucción es similar a la anterior, excepto que la dirección de bifurcación proviene ahora del registro pipeline, más bien, de una fila. En algunos casos, se puede pensar que esta instrucción tiene una extensión de una palabra por fila; esto es, al usar esta instrucción, se puede ejecutar todavía un lazo con el contador. La diferencia con la instrucción 8, es que si en esta instrucción se origina un fallo en el test de condición, la fuente que suministra la dirección de la próxima microinstrucción es la entrada D, y cuando se pasa a la condición de test, esta instrucción no ejecuta un POP porque el stack no se está usando.

  En el ejemplo de la fig. 21, la instrucción RPCT es la instrucción 52 y se muestra como un lazo de microinstrucción. La dirección en el registro pipeline, sería 52. La instrucción 51 en este ejemplo, podría ser la instrucción LOAD COUNTER AND CONTINUE (Nº12).

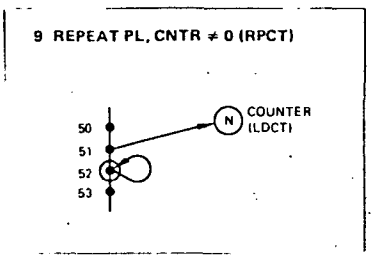


Fig. 21

Mientras que en el ejemplo se muestra un lazo simple de microinstrucción, cambiando simplemente la dirección en el registro pipeline, se podrían ejecutar de esta manera, lazos multi-instrucción para un determinado número de veces, según esté determinado por el contador.

- Instrucción 10, es la instrucción CONDITIONAL RETURN FROM SUBROUTINE. Como su nombre indica, se usa para retornar desde una subrutina a la siguiente dirección del microprograma. Como esta instrucción es condicional, este retorno sólo se ejecuta si se ha pasado el test. Si no ocurre así, se ejecura la siguiente secuencia de microinstrucciones. El ejemplo de la figura 22 describe el uso de la instrucción, tanto en el modo condicional, como en el incondicional.

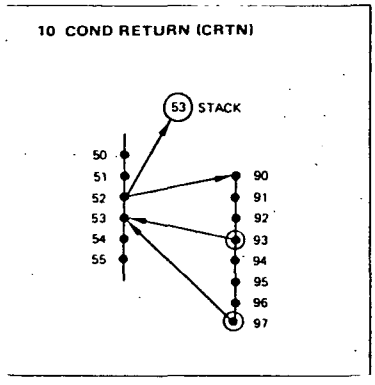


Fig. 22

Tenemos primero un JUMP TO SUBROUTINE en la instrucción localizada en 52, donde el control se transfiere al lugar 90. En el lugar 93 se ejecuta una CRTN. Si el test pasa, se accede al stack y el reprograma se

transferirá a la siguiente instrucción situada en 53. Si el test falla, se ejecutará la siguiente microinstrucción situada en la dirección 94. El programa continuará hasta la dirección 97, donde se completará la subrutina. Para que se realice un RETURN FROM SUBROUTINE INCONDITIONAL, la instrucción condicional RETURN FROM SUBROUTINE se ejecutará incondicionalmente.

La microinstrucción en la dirección 97 se programa para forzar $\overline{CCEN}$ a HIGH, anulando el test y forzando un PASS que origina un retorno incondicional.

- Instrucción 11, es la instrucción CONDITIONAL JUMP PIPE LINE (a la dirección del registro pipeline) y POP stack. Esta instrucción provee otra técnica de terminación de lazo y mantenimiento del stack. El ejemplo de la figura 23 está ejecutando un lazo desde la dirección 55 a la 51. Las instrucciones localizadas en 52, 53 y 54 son todas instrucciones condicionales JUMP o POP. En la dirección 52, si la entrada TEST es HIGH, se hará una bifurcación a la dirección 70 y el stack estará mantenido por medio de POP. Si el test fallara, la instrucción del lugar 53 (la próxima instrucción secuencial) se llevará a cabo.

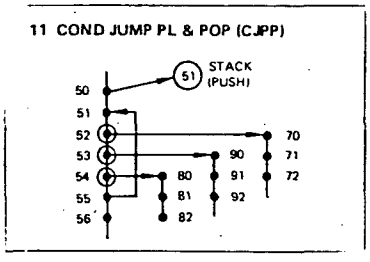


Fig. 2.3

Igualmente, en la dirección 53, se ejecutarán posteriormente o la instrucción situada en 90, o bien la que está localizada 54, según el test pasa o no.
La instrucción en 54 sigue el mismo procedimiento, va la 80 o al 55, según la condición del test.

- Instrucción 12, es la instrucción LOAD COUNTER AND CONTINUE, la cual solamente habilita el contador para que se cargue con el valor de su entrada en paralelo. Esta entrada está normalmente conectada al campo "dirección de bifurcación" del registro pipeline, el cual sirve para suministrar o una dirección de bifurcación, o un valor del contador, según la microinstrucción que se esté ejecutando. En conjunto, tenemos tres maneras de cargar el contador: la carga explícita por la instrucción 12; la carga condicional, incluída como una parte de la instrucción 4; y el uso de la entrada $\overline{RLD}$ junto con cualquier instrucción. El uso de $\overline{RLD}$

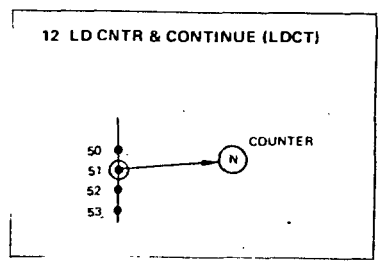


Fig. 24

con una instrucción supedita a que en la instrucción se especifique una cuenta o un decrecimiento para su carga instantánea.
Esta instrucción es equivalente a la combinación de

la instrucción 14 y $\overline{\text{RLD}}$ LOW. El propósito de esta instrucción es la de proveer una pequeña capacidad para cargar el REGISTER/COUNTER, en estas implementaciones en las cuales no está previsto el control del microprograma por $\overline{\text{RLD}}$.

- Instrucción 13, es la instrucción TEST END OF LOOP, la cual tiene la capacidad de salir de un lazo al fin de éste; esto es, es una instrucción condicional que originará un lazo de microprograma a través de la fila, si el test falla y si ocurre lo contrario, continuará a la siguiente instrucción secuencial.

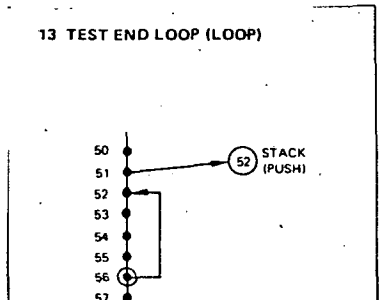


Fig. 25

el ejemplo de la figura 25 muestra la instrucción TEST END OF LOOP situada en la dirección 50. Si el test falla, el microprograma se bifurcará a la dirección 52. La dirección 52 está en el stack porque se ha ejecutado anteriormente una instrucción PUSH en 51. Si pasa el test en 56, el lazo finaliza y se ejecuta la siguiente microinstrucción secuencial situada en 57, lo cual origina que el stack sea POPeado; de este modo se efectúa el mantenimiento requerido en el stack.

- Instrucción 14, es la instrucción CONTINUE, la cual simplemente origina que el contador del microprograma se incremente, una vez que se ejecute la siguiente microinstrucción secuencial.

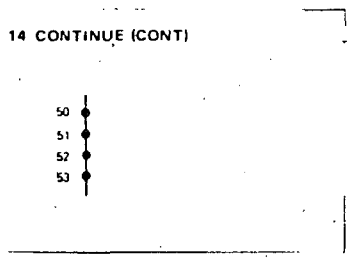


Fig. 26

Esta es la microinstrucción más simple de todas.

- Instrucción 15, es la instrucción THREE WAY BRANCH y es la más compleja de todas. Está provista para testear tanto la condición dependiente de datos como el contador. Además selecciona una entre tres direcciones de microinstrucción, como la siguiente microinstrucción que se va a ejecutar como la instrucción 8, una instrucción previa cargará un número determinado en el REGISTER/COUNTER mientras "empuja" la dirección de una μbifurcación sobre el stack. La instrucción 15 ejecuta una función decrecer -y-bifurcar-hasta-cero, similar a la instrucción 8. La siguiente dirección se toma de lo alto del stack hasta que el número cargado en el contador alcance el valor cero, cuando ocurre esto, la siguiente instrucción viene del registro pipeline. Esta acción continua, tanto en cuanto falle la condición del test. Si en alguna ejecución de la instrucción la condición de test pasa, no se toma ninguna bifurcación; el contador del microprograma provee

la siguiente dirección. Cuando se acaba el lazo, tanto si el contador llega a cero o pasa la condición de test, el stack se "POPea" por decremento del stack pointer, pueto que no nos interesa ya el valor contenido en lo alto del stack.

Como ejemplo consideremos el caso de búsqueda de una instrucción en memoria. Como se ve en la figura 27, la instrucción en la dirección 63 del microprograma, puede ser una instrucción 4 (PUSH), la cual "empuja" el valor 64 sobre el stack del microprograma y cargará el número N, el cual es un número inferior en una unidad al número de lugares de memoria que se examinarán.

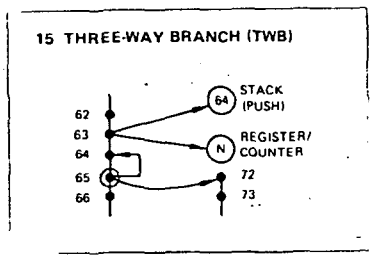


Fig. 27

El lugar 64 contiene una microinstrucción que irá a buscar el siguiente operando desde el área de memoria que se examinará y que lo comparará con el dato pedido. El lugar 65 contiene una microinstrucción que testea el resultado de la comparación y también es un THREE-WAY-BRANCH para el control del microprograma. Si no se encuentra ninguna concordancia, el test falla y el microprograma regresa al lugar 64 para la siguiente dirección del operando. Cuando la cuenta llega a cero, el microprograma bifurca al lugar 72, que hará todo lo necesario en caso de que no se encuentre

ninguna concordancia. Si por el contrario ésta ocurre en cualquier ejecución del THREE-WAY-BRANCH en el lugar 65, el control pasa al lugar 66 conduciendo éste caso. Si la instrucción finaliza, el stack se "POPea" una vez, quitando el valro 64 de lo alto del stack.

5.2.4.- APLICACION DE UN AM 2910 EN UNA CCU.-

El diagrama de bloques de la figura 28, es una unidad de control de la memoria del microprograma, en la que he introducido el AM 2910. Esta estructura es capaz de ejecutar muchas y sofisticadas instrucciones.

Vemos que el registro de instrucción es capaz de cargarse con una instrucción de máquina que proceda de bus de datos. La parte OP CODE de la instrucción se decodifica por medio de una PROM (mapping) , para llegar a la dirección inicial de la secuencia de microinstrucciones requerida para ejecutar la instrucción de máquina. Cuando la primera microinstrucción de la secuencia de instrucción de máquina es un direccionamiento de la memoria del microprograma, el control de la siguiente dirección del AM 2910 selecciona la entrada D del multiplexor y pone en three-state la salida de la PROM.

Cuando esta microinsturcción se ejecuta, se selecciona la siguiente dirección de microinstrucción como una función JUMP, la dirección JUMP estará disponible en al entrada D del multiplexor. Esto se realiza debido a que el AM 2910 selecciona como siguiente dirección a la entrada D del multiplexor y también habilita la salida de "dirección de bifurcación" del regis-

tro pipeline en three-state.

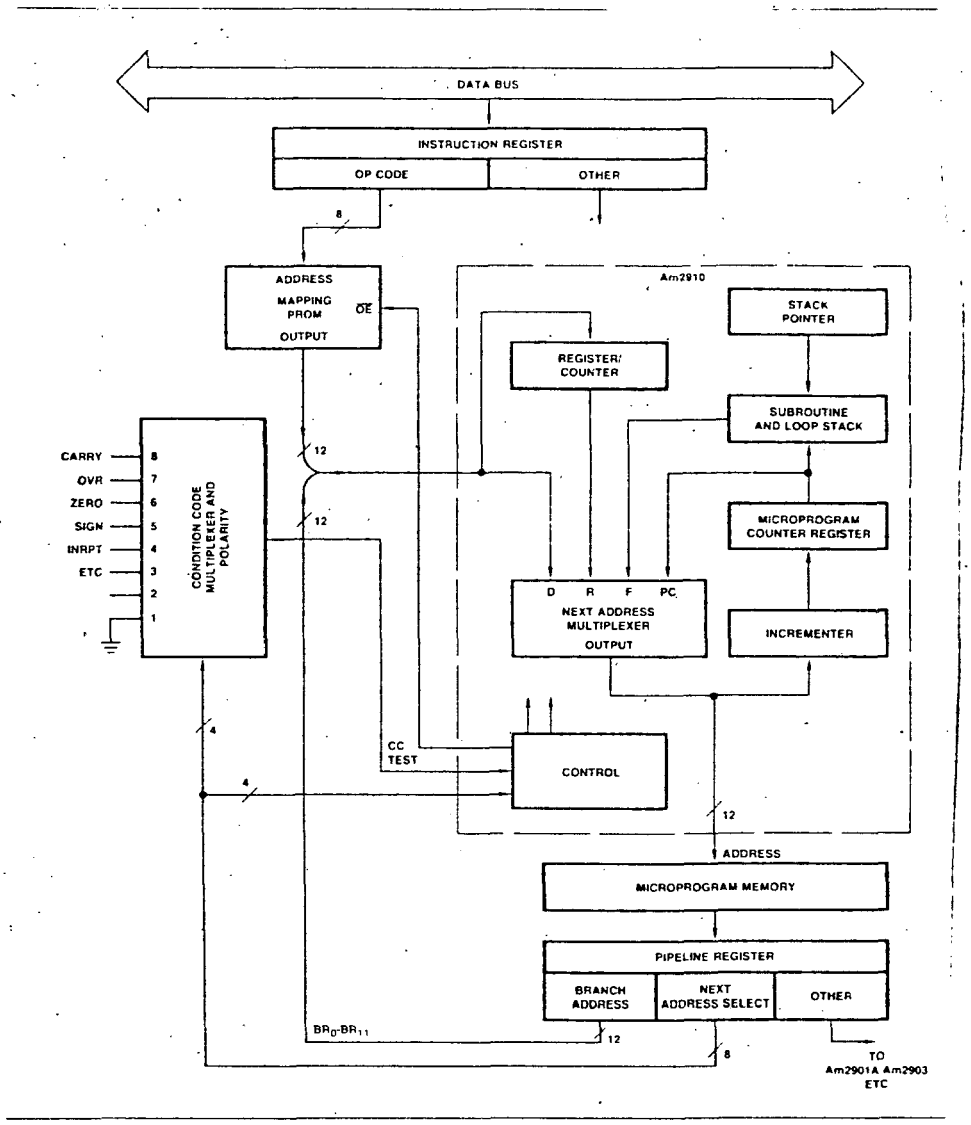


Fig. 28

El registro permite que la entrada al AM 2910 se ponga a tierra, para que este registro cargue el valor en la entrada D del AM 2910. El valor en D se pasa al registro del AM 2910 (R) al final de este microciclo, que hace que el valro D de este microciclo esté disponible como el valor R del siguiente microciclo. De este modo, al utilizar el campo de dirección de

bifurcación de dos secuencias de microinstrucciones, una JUMP-TO-ONE-OF-TWO-SUBRUOTINES CONDITIONAL o una JUMP-TO-ONE-OF-TWO-BRANCHES-ADDRESSES CONDITIONAL, se podrían ejecutar, seleccionando la entrada D o bien la R del multiplexor. Se utiliza el contador de programa del AM 2910, cuando se secuencian varias microinstrucciones continuadas en la memoria del microprograma. Aquí, el control lógico simplemente selecciona la entrada del PC del multiplexor del AM 2910. En resumen, la mayoría de estas instrucciones habilitan las salidas del campo de "dirección de bifurcación" del registro pipeline en three state, lo que permite que el registro del AM 2910 se cargue. El sctack de 5 x 12 se utiliza para hacer lazos y subrutinas en operaciones microprogramadas. Se pueden almacenar hasta cinco niveles de subrutinas o lazos. También se pueden entremezclar, siempre y cuando no se exceda la capacidad de cinco palabras del stack.

Otro punto clave en el estudio de la CCU es su inicialización. Una técnica para llevar esto a cabo es la de utilizar un registro pipeline con una de las entradas borradas para poner a LOW todas las entradas de instrucciones de AM 2910.

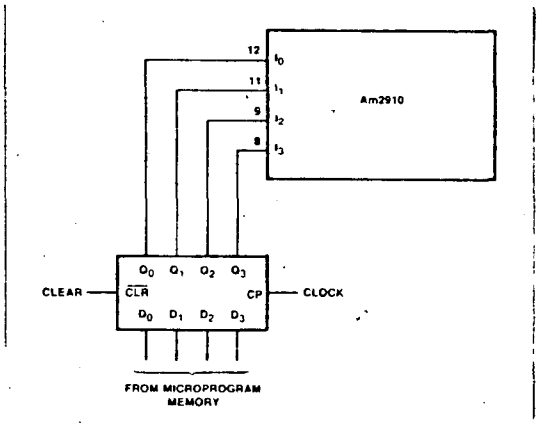


Fig. 29

Esto originará un reset del stack del AM 2910 y forzará las salidas a la palabra cero y al microcódigo, que se puede usar para la rutina de inicialización. Generalmente, el encendido se ejecuta por disparo de un timer que se conecta a la entrada libre del registro pipeline, como muestra la figura 29.

5.2.5.- ESTUDIO DE TIEMPOS EN LA CCU.

El mínimo ciclo de reloj que se puede usar en el diseño de una CCU está generalmente determinado por los retardos de los componentes a través de la ruta "pipeline-register-clock to logic to pipeline-register-clock" más larga. Al principio de cualquier ciclo de reloj, los datos disponibles a la salida de la memoria del microprograma, del contador de estado, o de cualquier otro registro de datos, se guardan en sus registros pipeline asociados. En este punto, comienzan todas las rutas de retardo. Una inspección visual no siempre nos indicará la ruta de retardo más larga de la señal. Las rutas más largas son un buen lugar para comenzar, pero cada ruta definida se podría calcular, componente a componente, hasta que se encuentre verdaderamente la ruta lógica más larga de la señal.

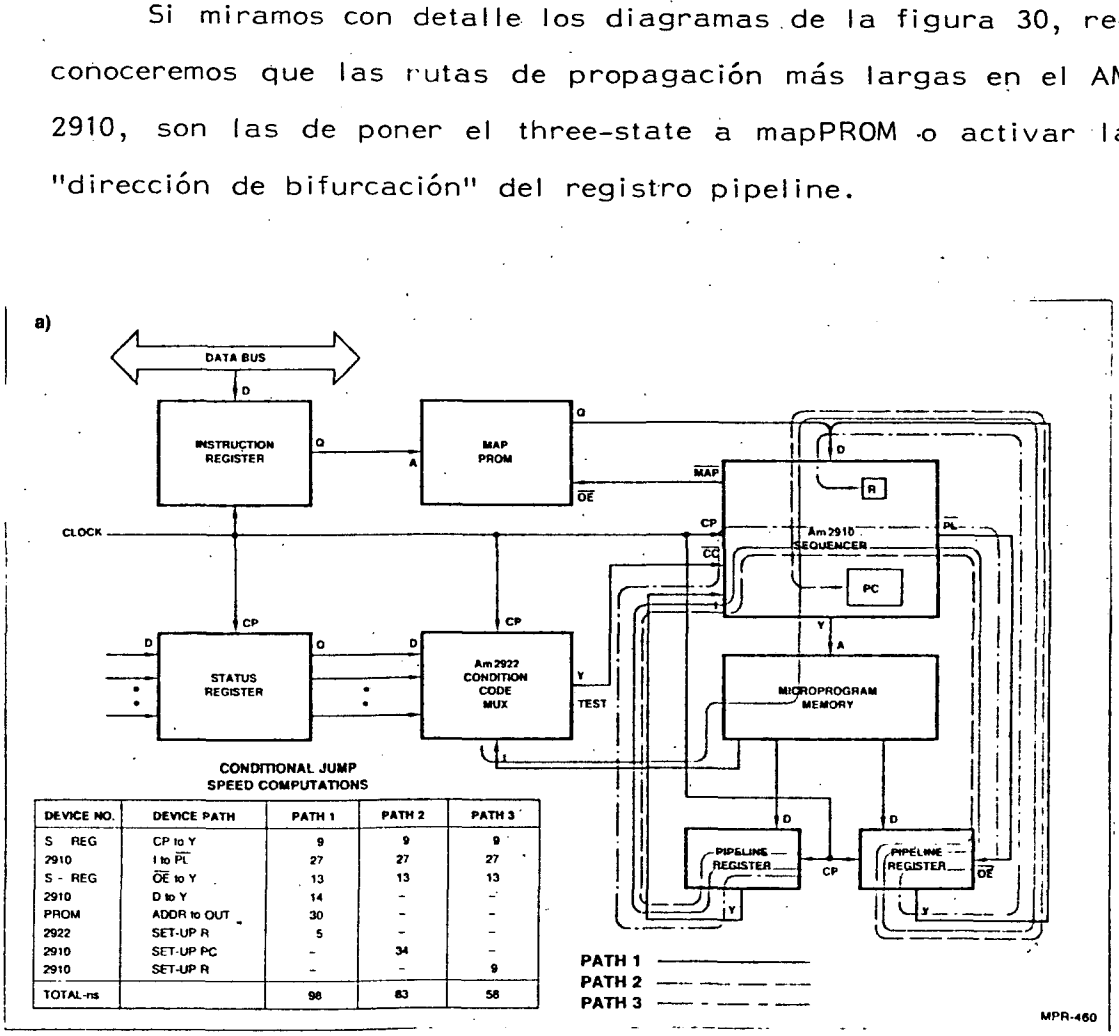
Si observamos la figura 28, podemos identificar varias de estas rutas potencialmente largas. Estas son:

a) Registro de Instrucción -------- Registro Pipeline
b) Registro Pipeline -------- Registro Pipeline, a través del multiplexor del código de condición (Condition Code Multipl.)
c) Registro de Estado -------- Registro Pipeline

Para calcular los tiempos de retardo de estas rutas, he representado los diagramas de la figura 30. De esta manera podremos ejecutar estos cálculos observando los retardos de propagación que en ellos se exponen.

En nuestro caso escogeremos el caso más desfavorable y basándonos en él, tendremos el ciclo mínimo de reloj. Todo esto teniendo en cuenta que estas medidas fueron hechas a 25 ºC y con una alimentación de 5 V.

Si miramos con detalle los diagramas de la figura 30, reconoceremos que las rutas de propagación más largas en el AM 2910, son las de poner el three-state a mapPROM o activar la "dirección de bifurcación" del registro pipeline.

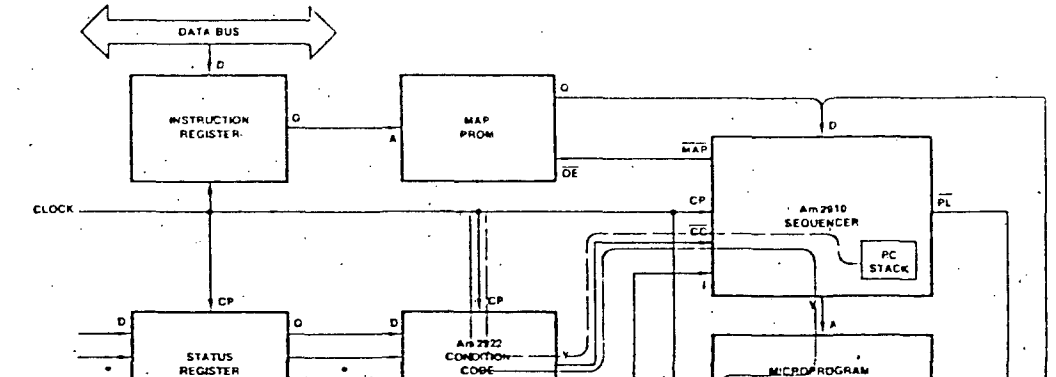


CONDITIONAL JUMP SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S REG	CP to Y	9	9	9
2910	I to $\overline{PL}$	27	27	27
S - REG	$\overline{OE}$ to Y	13	13	13
2910	D to Y	14	–	–
PROM	ADDR to OUT	30	–	–
2922	SET-UP R	5	–	–
2910	SET-UP PC	–	34	–
2910	SET-UP R	–	–	9
TOTAL-ns		98	83	58

Fig. 30

b)



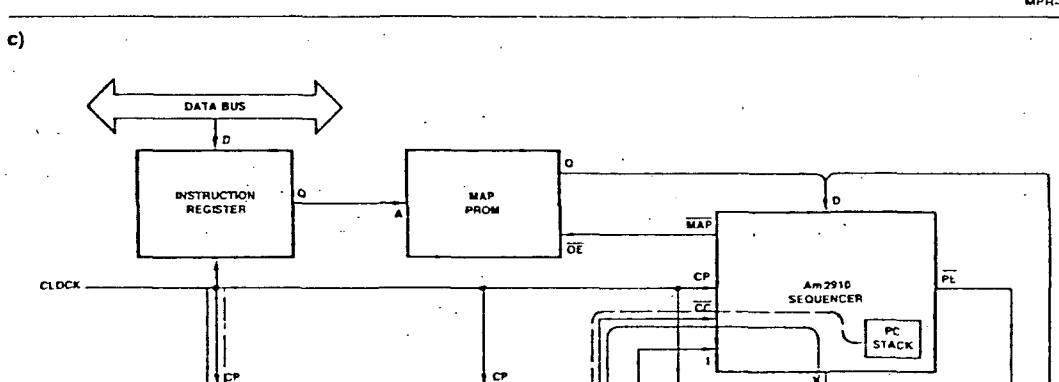
CONDITIONAL JUMP
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2
2922	CP to Y	21	21
2910	$\overline{CC}$ to Y	21	–
PROM	ADDR to OUT	30	–
2922	SET-UP R	5	–
2910	SET-UP PC	–	46
TOTAL-ns		77	67

PATH 1
PATH 2

MPR-461

c)



CONDITIONAL JUMP
SPEED COMPUTATIONS

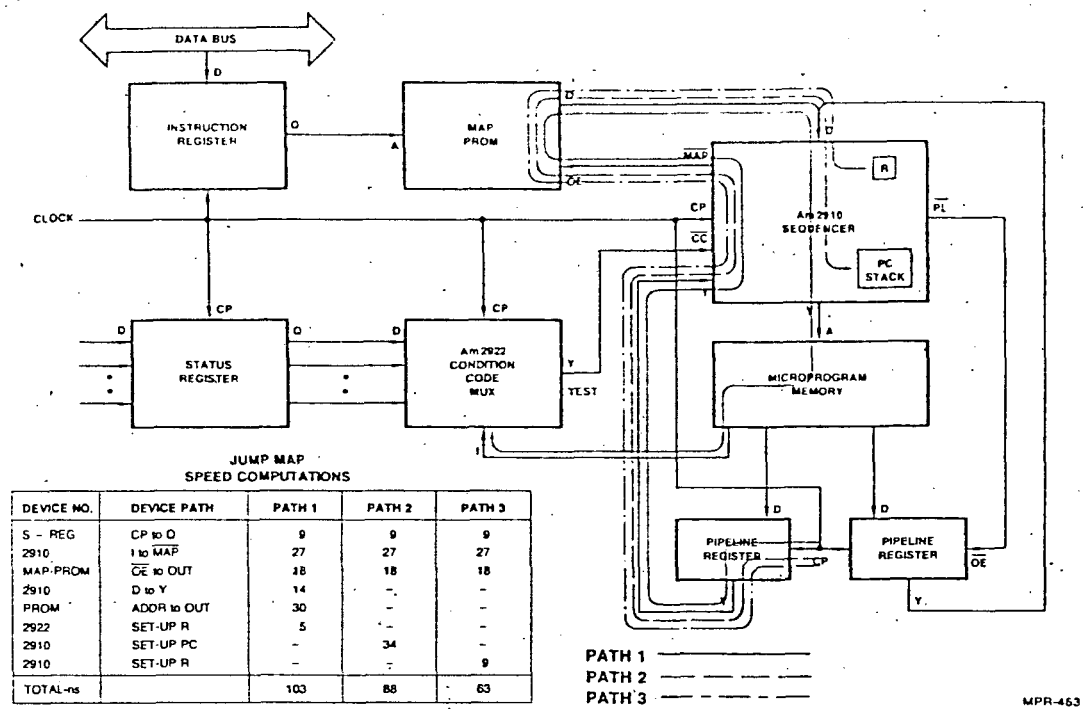
DEVICE NO.	DEVICE PATH	PATH 1	PATH 2
S – REG	CP to Q	9	9
2922	D to Y	13	13
2910	$\overline{CC}$ to Y	21	–
PROM	ADDR TO OUT	30	–
2922	SET-UP R	5	–
2910	SET-UP PC	–	46
TOTAL-ns		78	68

PATH 1
PATH 2

MPR-462

Fig. 30 (Cont.)

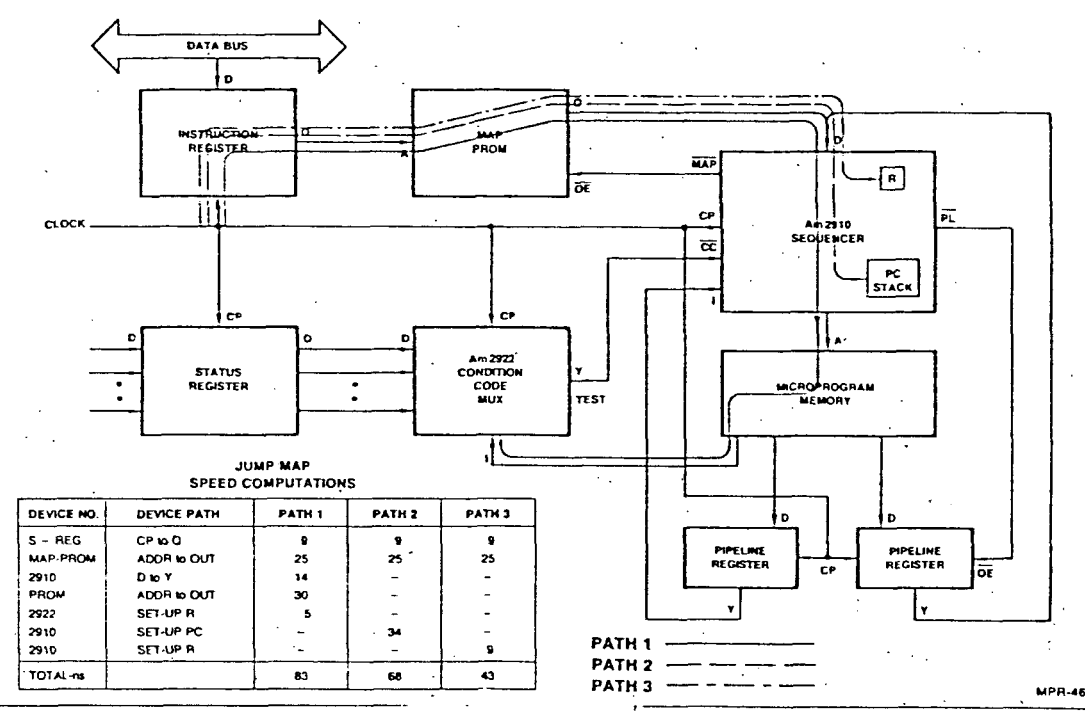
d)



JUMP MAP
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S – REG	CP to O	9	9	9
2910	I to $\overline{MAP}$	27	27	27
MAP-PROM	$\overline{OE}$ to OUT	18	18	18
2910	D to Y	14	–	–
PROM	ADDR to OUT	30	–	–
2922	SET-UP R	5	–	–
2910	SET-UP PC	–	34	–
2910	SET-UP R	–	–	9
TOTAL-ns		103	88	63

e)

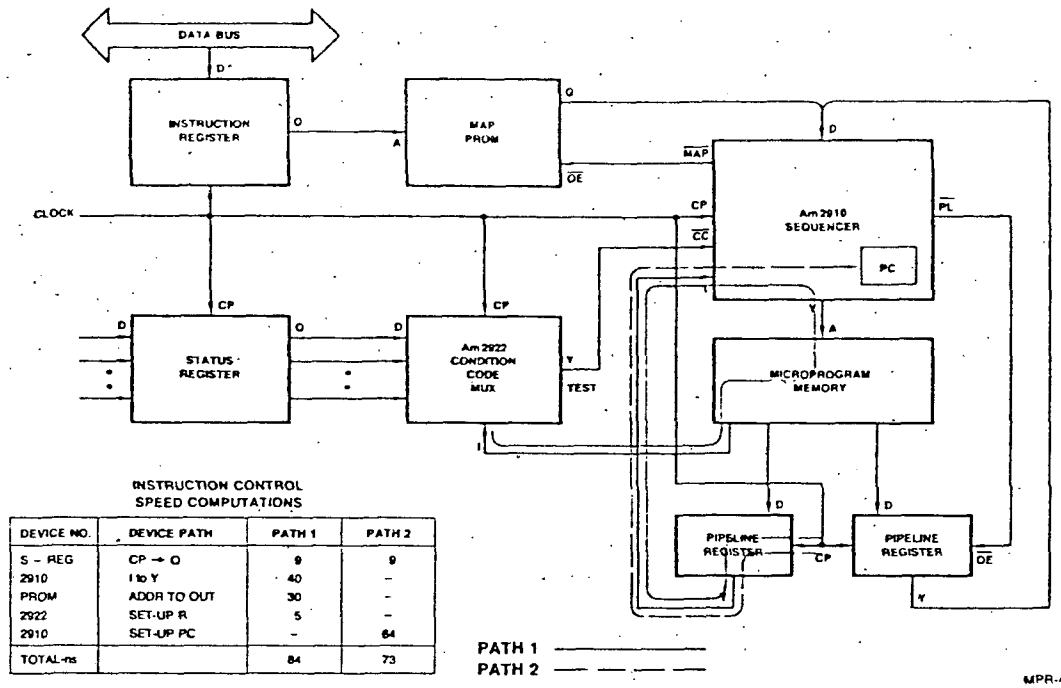


JUMP MAP
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S – REG	CP to O	9	9	9
MAP-PROM	ADDR to OUT	25	25	25
2910	D to Y	14	–	–
PROM	ADDR to OUT	30	–	–
2922	SET-UP R	5	–	–
2910	SET-UP PC	–	34	–
2910	SET-UP R	–	–	9
TOTAL-ns		83	68	43

Fig. 30 (Cont.)

f)

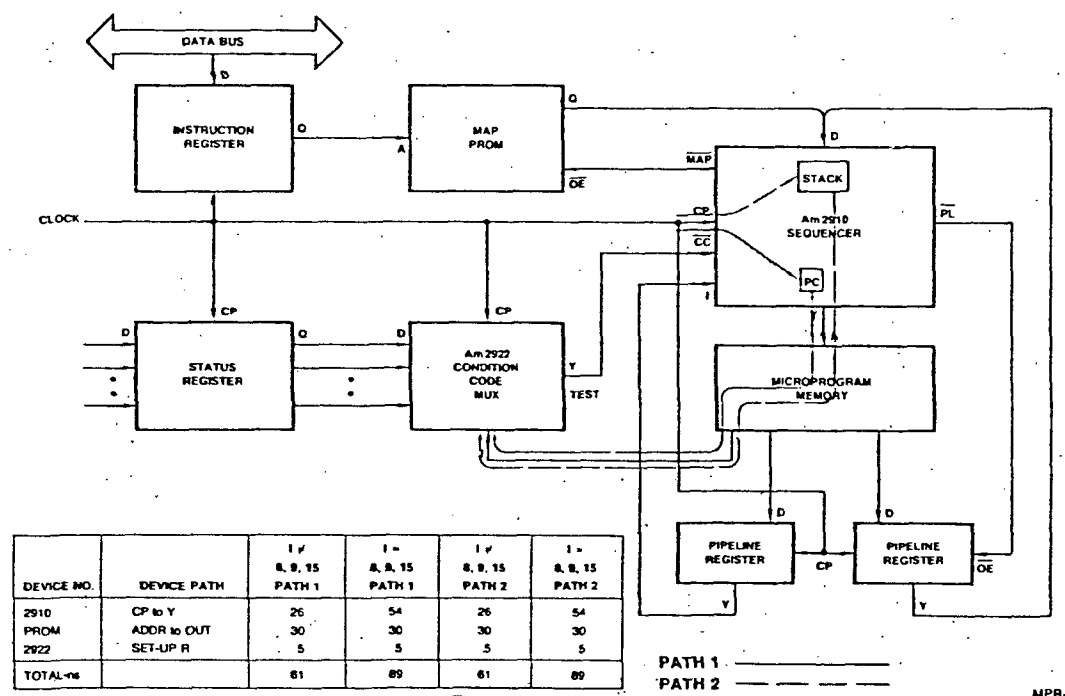


INSTRUCTION CONTROL SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2
S – REG	CP → O	9	9
2910	I to Y	40	–
PROM	ADDR TO OUT	30	–
2922	SET-UP R	5	–
2910	SET-UP PC	–	64
TOTAL-ns		84	73

MPR-465

g)



DEVICE NO.	DEVICE PATH	I ≠ 8, 9, 15 PATH 1	I = 8, 9, 15 PATH 1	I ≠ 8, 9, 15 PATH 2	I = 8, 9, 15 PATH 2
2910	CP to Y	26	54	26	54
PROM	ADDR to OUT	30	30	30	30
2922	SET-UP R	5	5	5	5
TOTAL-ns		61	89	61	89

MPR-466

Fig. 30 (Cont.)

Si se deseasen velocidades más rápidas, estas rutas se podrían eliminar utilizando algunas técnicas. Una de estas es simplemente colocar uno o más bits en el registro pipeline para el control de la habilitación del estado de alta impedancia (three-state) de los varios dispositivos conectados a la entrada D del AM 2910. Por ejemplo, en la figura 29, un solo bit sería suficiente y el registro pipeline se podría implementar con un registro AM 74S175. Esto permitiría qué las salidas, verdadera y complemento, se usaran para gobernar la habilitación de la salida de "dirección de bifurcación" del registro pipeline. De este modo, estas rutas se podrían eliminar y obtendríamos una mejora de unos 30 ns.

5.3.- AM 2903 ALU

5.3.1.- DESCRIPCION GENERAL

El AM 2903 es una Unidad Aritmética Lógica, de tecnología Schottky TTL de baja disipación.

Su principal propiedad es que es un microprocesador bipolar de cuatro bits que puede ampliarse, ya que puede montarse en bit-slice (conectarse varios de ellos en cascada). Está provisto de unas mejoras significativas que son esencialmente útiles en los procesos aritméticos.

El AM 2903 contiene 16 registros internos de trabajo colocados en una arquitectura de dos direcciones y está provisto también de todas las señales necesarias para ampliar externamente el registro de fila utilizando el registro stack AM 29705.

Cualquier número de registros se pueden conectar en cascada al AM 2903, usando esta técnica.

En relación con su juego de instrucción, diremos que está provisto de una serie de instrucciones que facilitan la ejecución de la multiplicación, división, normalización y otras como la generación de paridad y ampliación del signo.

5.3.2.- ARQUITECTURA DEL AM 2903.

La flexibilidad de la microinstrucción del AM 2903 permite que se pueda comparar con la mayoría de las computadoras digitales.

La microinstrucción de nueve bits selecciona las fuentes de la ALU, su función y su destino.

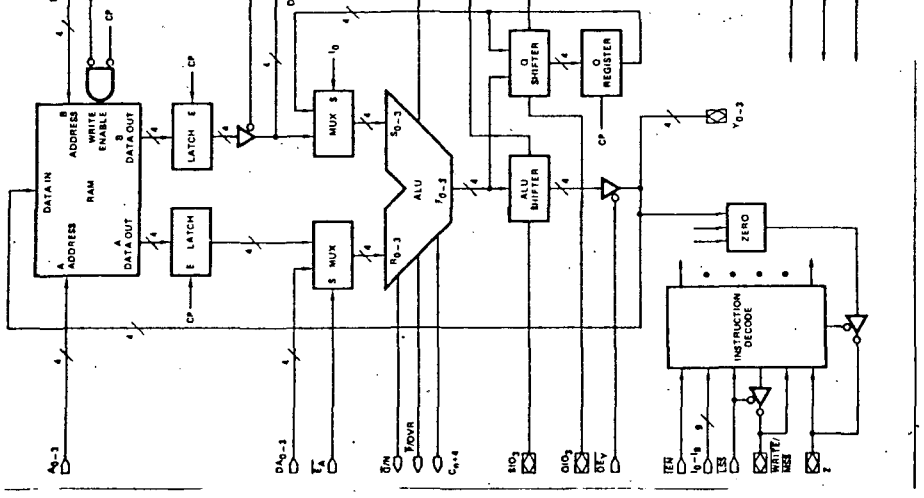


Fig. 31

El AM 2903 se puede conectar en cascada con un sumador de acarreo adelantado que nos proporciona mayor velocidad de cálculo.

Posee además salidas en three-state. El chip tiene 48 pines y es un circuito LSI.

Todas las rutas de datos dentro del dispositivo son de cuatro bits. Como se muestra en el diagrama de bloques de la figura 31.

El dispositivo tiene una RAM de dos puertas de 16 palabras x 4 bits, con dos latches en ambas salidas, con una ALU de gran ejecución, un registro de desplazamiento para la ALU, un registro Q de muchas aplicaciones con entrada por un registro de desplazamiento y un decodificador para instrucciones de nueve bits.

5.3.2.a.- RAM DE DOS PUERTAS.

Dos palabras cualquiera de la RAM direccionadas en las puertas de dirección A y B, se pueden leer simultáneamente en las respectivas puertas de salida A y B de la RAM. Aparecerán datos idénticos en las dos puertas de salida cuando se aplique al misma dirección en las dos puertas de entrada.

Los latches en las puertas de salida de la RAM se harán transparentes cuando la entrada de reloj CP sea HIGH y mantendrán los datos de salida de la RAM cuando CP sea LOW. Los datos se pueden leer directamente en la puerta de Entrada/Salida DB, bajo el contro de $\overline{OE}_B$.

Los datos externos en la puerta de Entrada/Salida Y se pueden escribir directamente en la RAM, o se pueden habilitar los datos de salida del registro de desplazamiento de la ALU sobre la puerta I/O e introducirlos en la RAM.

Cuando la entrada de permiso $\overline{WE}$, y la entrada de reloj, CP son LOW, los datos se escriben en la RAM en la dirección B.

5.3.2.b.- UNIDAD ARITMETICA LOGICA.

El AM 2903 puede ejecutar siete operaciones aritméticas y nueve lógicas, con dos operandos de cuatro bits. Los multiplexores situados a la entrada de la ALU, posibilitan la capacidad de seleccionar varios pares de fuentes de datos. La entrada $\overline{E}_A$ selecciona, o la entrada DA de datos del exterior o la puerta A de salida de la RAM para usarla como un operando de la ALU. Mientras que las entradas $\overline{OE}_B$ e lo seleccionan, o la puerta B de salida de la RAM, o la entrada de datos del exterior DB, o el contenido del registro Q, para usarlos como el segundo operando de la ALU. De este modo, el AM 2903 puede operar sobre datos de dos fuentes exteriores, sobre una fuente interior y otra exterior, o sobre dos fuentes internas.

Cuando los bits de instrucción I_4, I_3, I_2, I_1 e lo están a LOW, el AM 2903 ejecuta funciones especiales. En la figura 32 están definidas estas funciones y la operación que ejecuta la ALU en cada una de ellas.

I_8	I_7	I_6	I_5	Hex Code	Special Function	ALU Function	ALU Shifter Function	SIO_3 Most Sig. Slice	SIO_3 Other Slices	SIO_0	Q Reg & Shifter Function	QIO_3	QIO_0	$\overline{WRITE}$
L	L	L	L	0	Unsigned Multiply	F= S+C_n if Z=L F=R+S+C_n if Z=H	Log. F/2→Y (Note 1)	Hi-Z	Input	F_0	Log. Q/2→Q	Input	Q_0	L
L	L	H	L	2	Two's Complement Multiply	F=S+C_n if Z=L F=R+S+C_n if Z=H	Log. F/2→Y (Note 2)	Hi-Z	Input	F_0	Log. Q/2→Q	Input	Q_0	L
L	H	L	L	4	Increment by One or Two	F=S+1+C_n	F→Y	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L	H	L	H	5	Sign/Magnitude-Two's Complement	F=$\overline{S}$+C_n if Z=L F=$\overline{S}$+C_n if Z=H	F→Y (Note 3)	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L	H	H	L	6	Two's Complement Multiply, Last Cycle	F=S+C_n if Z=L F=S−R−1+C_n if Z=H	Log. F/2→Y (Note 2)	Hi-Z	Input	F_0	Log. Q/2→Q	Input	Q_0	L
H	L	L	L	8	Single Length Normalize	F=S+C_n	F→Y	F_3	F_3	Hi-Z	Log. 2Q→Q	Q_3	Input	L
H	L	H	L	A	Double Length Normalize and First Divide Op	F=S+C_n	Log. 2F→Y	$R_3 \forall F_3$	F_3	Input	Log. 2Q→Q	Q_3	Input	L
H	H	L	L	C	Two's Complement Divide	F=S+R+C_n if Z=L F=S−R−1+C_n if Z=H	Log. 2F→Y	$\overline{R_3 \forall F_3}$	F_3	Input	Log. 2Q→Q	Q_3	Input	L
H	H	H	L	E	Two's Complement Divide, Correction and Remainder	F=S+R+C_n if Z=L F=S−R−1+C_n if Z=H	F→Y	F_3	F_3	Hi-Z	Log. 2Q→Q	Q_3	Input	L

NOTES: 1. At the most significant slice only, the C_{n+4} signal is internally gated to the Y_3 output.
2. At the most significant slice only, $F_3 \forall OVR$ is internally gated to the Y_3 output.
3. At the most significant slice only, $S_3 \forall F_3$ is generated at the Y_3 output.
4. Op codes 1, 3, 7, 9, B, D, and F are reserved for future use.

L = LOW
H = HIGH
X = Don't Care

Hi-Z = High Impedance
$\forall$ = Exclusive OR
Parity = $SIO_3 \forall F_3 \forall F_2 \forall F_1 \forall F_0$

Fig. 32

Cuando el AM 2903 ejecuta otras instrucciones distintas a estas nueve funciones especiales, la operación se determina con los distintos bits de instrucción I_4, I_3, I_2, I_1 e Io. La figura 33 muestra la operación que realiza la ALU en función de estos cuatro bits de instrucción.

Cuando un número determinado de AM 2903 se conectan en cascada, cada pastilla debe estar programada para ser el chip más significtivo (MSS), el chip intermedio (IS), o le menos significativo (LSS) del conjunto. Las señales $\overline{G}$ (Carry Generate) y $\overline{P}$ (Carry Propagate) requeridas para un esquema de acarreo adelantado se generan por el AM 2093 y están disponibles como salidas del LSS y del IS.

I_4	I_3	I_2	I_1	Hex Code	ALU Functions	
L	L	L	L	0	I_0 = L	Special Functions
					I_0 = H	F_i = HIGH
L	L	L	H	1	F = S Minus R Minus 1 Plus C_n	
L	L	H	L	2	F = R Minus S Minus 1 Plus C_n	
L	L	H	H	3	F = R Plus S Plus C_n	
L	H	L	L	4	F = S Plus C_n	
L	H	L	H	5	F = $\overline{S}$ Plus C_n	
L	H	H	L	6	F = R Plus C_n	
L	H	H	H	7	F = $\overline{R}$ Plus C_n	
H	L	L	L	8	F_i = LOW	
H	L	L	H	9	F_i = $\overline{R}_i$ AND S_i	
H	L	H	L	A	F_i = R_i EXCLUSIVE NOR S_i	
H	L	H	H	B	F_i = R_i EXCLUSIVE OR S_i	
H	H	L	L	C	F_i = R_i AND S_i	
H	H	L	H	D	F_i = R_i NOR S_i	
H	H	H	L	E	F_i = R_i NAND S_i	
H	H	H	H	F	F_i = R_i OR S_i	

L = LOW H = HIGH i = 0 to 3

Fig. 33

El AM 2903 genera también una señal Carry-Out, C_{n+4} , la cual está generalmente dispuesta como salida en cada pastilla. Tanto la señal Carry-in, Cn, como C_{n+4} , son activas en HIGH. La ALU también genera otras dos salidas de estado. Estas son NEGATIVO, N, y OVERFLOW, OVR. La salida N es generalmente el bit más significativo de la salida de la ALU y se usa para determinar si un resultado es positivo o negativo. La salida OVR indica que la operación aritmética que se ha realizado excede el rango disponible del núméto de complemento a dos. Las señales N y OVR están dispuestas como salidas de la pastilla más significativa (MSS).

5.3.2.c.- REGISTRO DE DESPLAZAMIENTO DE LA ALU.

Bajo una instrucción de control, el registro de desplaza-

miento de la ALU corre la salida de la ALU (F) no desplazada, un bit a la izquierda (2F) o un bit a la derecha (F/2). Son posibles tanto el desplazamiento lógico como aritmético. Una operación aritmética de desplazamiento, desplaza los datos rodeando la posición del bit más significativo del MSS, y una operación de desplazamiento lógico desplaza los datos a través de esta posición, como se ve en la figura 34. SIO_0 y SIO_3 son entradas/salidas bireccionales, en serie, de desplazamiento. Durante un desplazamiento hacia la izquierda, SIO_0 es una entrada en serie de desplazamiento y SIO_3 es una salida en serie Durante una operación de desplazamiento hacia la derecha ocurre lo contrario.

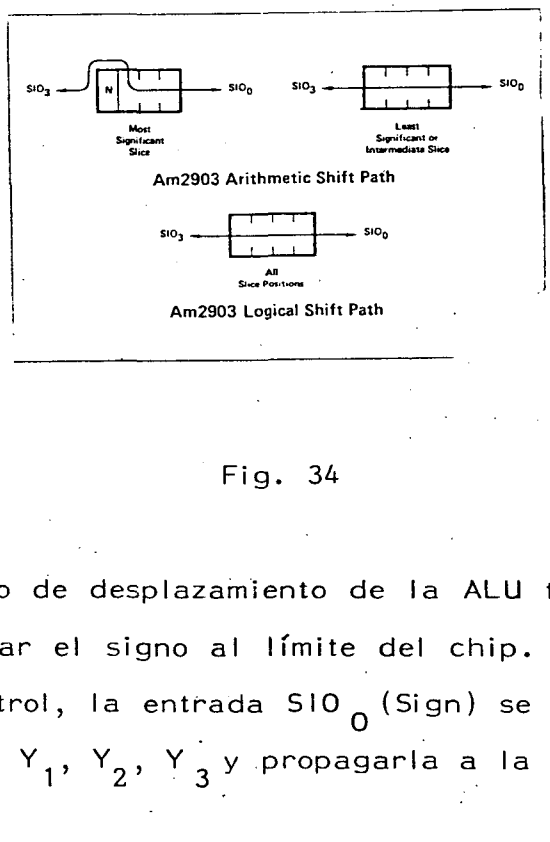


Fig. 34

El registro de desplazamiento de la ALU tiene la capacidad de prolongar el signo al límite del chip. Bajo una instrucción de control, la entrada SIO_0 (Sign) se puede prolongar a través de Y_0, Y_1, Y_2, Y_3 y propagarla a la salida SIO_3.

Dentro del registro de desplazamiento de la ALU, hay diseñado un generador/comprobador de paridad de cinco bits conectado en cascada que proporciona a la ALU la capacidad de detectar errores. Se genera la paridad para las salidas F_0,

F_1, F_2, F_3 de la ALU y la entrada SIO_3, y bajo una instrucción de control se dispone en la salida SIO_0.

Las instrucciones de entrada determinan la operación de desplazamiento.

En la figura 32 están definidas las funciones especiales y la operación que el registro de desplazamiento ejecuta en cada una de ellas. Cuando el AM 2903 ejecuta otro tipo de instrucciones, que no sean esas nueve, la operación que ejecuta el registro de desplazamiento viene determinada por los bits de instrucción I_8, I_7, I_6, I_5. En la figura 35 se definen las operaciones de desplazamiento en función de estos cuatro bits.

I_8	I_7	I_6	I_5	Hex Code	ALU Shifter Function	SIO_3 Most Sig. Slice	SIO_3 Other Slices	Y_3 Most Sig. Slice	Y_3 Other Slices	Y_2 Most Sig. Slice	Y_2 Other Slices	Y_1	Y_0	SIO_0	$\overline{Write}$	Q Reg & Shifter Function	QIO_3	QIO_0
L	L	L	L	0	Arith. F/2→Y	Input	Input	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	L	Hold	Hi-Z	Hi-Z
L	L	L	H	1	Log. F/2→Y	Input	Input	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	L	Hold	Hi-Z	Hi-Z
L	L	H	L	2	Arith. F/2→Y	Input	Input	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	L	Log. Q/2→Q	Input	Q_0
L	L	H	H	3	Log. F/2→Y	Input	Input	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	L	Log. Q/2→Q	Input	Q_0
L	H	L	L	4	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	L	Hold	Hi-Z	Hi-Z
L	H	L	H	5	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	H	Log. Q/2→Q	Input	Q_0
L	H	H	L	6	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	H	F→Q	Hi-Z	Hi-Z
L	H	H	H	7	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	L	F→Q	Hi-Z	Hi-Z
H	L	L	L	8	Arith. 2F→Y	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	L	L	H	9	Log. 2F→Y	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	L	H	L	A	Arith. 2F→Y	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Input	L	Log. 2Q→Q	Q_3	Input
H	L	H	H	B	Log. 2F→Y	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Input	L	Log. 2Q→Q	Q_3	Input
H	H	L	L	C	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	H	Hold	Hi-Z	Hi-Z
H	H	L	H	D	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	H	Log. 2Q→Q	Q_3	Input
H	H	H	L	E	SIO_0→Y_0, Y_1, Y_2, Y_3	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	H	H	H	F	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	L	Hold	Hi-Z	Hi-Z

Parity = $F_3 \forall F_2 \forall F_1 \forall F_0 \forall SIO_3$
∀ = Exclusive OR

L = LOW
H = HIGH

Hi-Z = High Impedance

Fig. 35

5.3.2.d.- REGISTRO Q

El registro Q, es un registro auxiliar de cuatro bits que se activa en la transición LOW a HIGH de la entrada CP. Se

destina primordialmente para las operaciones de multiplicación y división; aunque se puede utilizar como un acumulador o un registro temporal en algunas aplicaciones.

La salida F de la ALU se puede cargar en el registro Q y/o el registro Q puede seleccionarse como fuente de operandos de la ALU. El registro de desplazamiento a la entrada del registro Q proporciona la capacidad de desplazar el contenido del registro Q, un bit a la izquierda (2Q) o un bit a la derecha (Q/2). Sólo se ejecutan desplazamientos lógicos. QIO_0 y QIO_3 son entradas/salidas bidireccionales en serie, de desplazamiento. Durante una operación de desplazamiento a la izquierda del registro Q, QIO_0 es una entrada serie de desplazamiento y QIO_3 es una salida serie de desplazamiento. Durante una operación de desplazamiento a la derecha, ocurre lo contrario. La capacidad de desplazamiento de palabras de doble longitud, tanto aritméticas como lógicas, están previstas por el AM 2903. El desplazamiento de doble longitud se ejecuta conectando el QIO_3 del MSS al SIO_0 del LSS, y realizando una instrucción que desplace tanto la salida de ALU como la del registro Q.

El registro Q y las operaciones de desplazamiento se controlan mediante los bits de instrucción I_8, I_7, I_6, I_5 . En la figura 32 y 35, están definidas las operaciones que ejecutan, tanto el registro Q como el registro de desplazamiento en función de estos cuatro bits.

5.3.2.e.- BUFFERS DE SALIDA.

Las puertas Y y DB son puertas bidireccionales de Entrada/Salida, gobernadas por BUFFERS de salida que se ponen en

Cuando $\overline{LSS}$ se pone a HIGH, el pin $\overline{WRITE}/\overline{MSS}$ llega a ser un pin de entrada. Manteniendo el pin $\overline{WRITE}/\overline{MSS}$ en HIGH, se programa el chip para operar como IS y manteniendo en LOW se programa como MSS.

NOTA: Las tablas características de las rutas de retardo de la AM 2903 están especificadas en el Anexo 1.

6.- DISEÑO DE LA CPU DE 16 BITS.

Lo primero que debemos tener en cuenta es el número de bits de operandos con los que va a trabajar la ALU Superslice. En este proyecto he supuesto que cada operando va a tener un número máximo de 16 bits. Por lo tanto necesitamos 4 AM 2903, ya que cada una trabaja con cuatro bits por cada operando.

La forma de conectarlas entre ellas y con el AM 2902, (generador de acarreo adelantado) se ve en el Plano 1, según lo especificado en el apartado 5.3.2.g. En segundo lugar, debemos poner una serie de componentes, exteriores a la ALU SL, que realicen una serie de canalizaciones (interface) entre la ALU SL y la CCU, una matriz de control de desplazamiento, que gobierne el encadenamiento de los desplazamientos interiores de la ALU SL (SIO_0-SIO_{15}, QIO_0- QIO_{15}), así como unos registros de estado para las señales CY, OVR, N y Z. El diagrama de bloques correspondiente a esta configuración se describe en el Plano 2.

En el Plano 3, tenemos la CCU ampliada, su conexión con la memoria del microprograma, el registro pipeline, el multiplexor de estado y las PROMS para las señales $\overline{VECT}$ y $\overline{MAP}$.

Uniendo estas tres partes obtenemos un esquema general de la CPU, que se representa en el Plano 4.

6.1.- ESQUEMA DE LA CPU.

Basándonos en el esquema general del Plano 4 y en la estructura del microprograma, podemos ya desarrollar la CPU.

6.1.1.- CCU.

Empezando por el registro de instrucción, necesitaremos tres registros. Dos registros de 4 bits (U2 y U3), que contendrán, o la dirección de los registros de los operandos, o los operandos en sí y otro de 8 bits que contendrá el OP CODE de la microinstrucción (U1).

Para la decodificación del OP CODE de la instrucción de máquina, utilizaremos tres PROMS AM 27S21 (U4, U5 y U6) que constituirán la MAP PROM, poniendo $\overline{CS}_2$ a tierra y conectando $\overline{CS}_1$ a la patilla $\overline{MAP}$ del AM 2910 (secuenciador) y las salidas de la MAP PROM a las entradas D. Como acabamos de decir, utilizaremos el AM 2910 como secuenciador del microprograma. Para habilitar las entradas D del AM 2910 para un posible salto (JUMP) del microcódigo, utilizaremos tres registros AM 2918, (U8, U9, U10) conectando sus entradas a los bits 56-67 (PL) del microprograma. Estos tres registros forman la parte "dirección de bifurcación" (BRANCH ADDRESS) del registro pipeline. Para las instrucciones de entrada al AM 2910 (I3 - Io), empleamos un registro AM 25LS399 (U14) que forma parte del registro pipeline, conectando su entrada a los bits 52-55 del microprograma. También como parte del registro pipeline necesitamos dos registros que puedan seleccionar DA y DB del AM 2903 desde el microprograma, mediante las micro-órdenes 24-31 del microprograma. Para ello he escogido otros dos registros AM 2919 (U11 y U12), conectándolos a la parte correspondiente del registro de instrucción.

Para controlar estos registros de selección, utilizo un decodificador AM 74S139 (U13), gobernado por las micro-órdenes 32

y 33 del microprograma.

Como parte también del registro pipeline, utilizo un inversor AM 2921 (U15) que me da a su salida, tanto el complemento como el valor real del bit 11 del microprograma, ya que esta micro-orden forma parte, no sólo del control de las entradas $\overline{IEN}$, $\overline{EA}$ y $\overline{OEB}$, sino que también se conecta a los registros de entrada (U20 y U21) para que no surjan problemas en los pines DB.

También necesitamos un decodificador "uno de dieciseis" cuya salida se conecta al bus DA que permite deducir todas las operaciones de los bits. Para ello tenemos dos decodificadores AM 2921 (U16 y U17). Donde el bit 40 del microprograma controla la polaridad del decodoficador y el bit 45, cuando está a LOW aplica la salida del decodificador a la puerta DA del Am 2903.

Por último necesitamos dos PROMS que tengan la capacidad de introducir una de treinta y dos constantes preprogramadas sobre el bus DA, usando para su direccionamiento los bits 56-60 del microprograma. Para ello utilizaremos dos AM 27S19 (U18 y U19) gobernadas por el bit 45 del microprograma, que habilitará la salida de estos PROMS sobre el bus DA.

6.1.2.- ZONA DE TRATAMIENTO DE DATOS.

Necesitamos dos registros de entrada de ocho bits cada uno que hagan la función de registros de datos. Para esto empleamos dos AM 2920 (U20 y U21) que lo conectaremos al bus de datos y su salida a la entrada directa de datos de la ALU S, los

cuales estarán gobernados por los bits del microprograma $\overline{PL11}$ y PL22.

Tendremos también las cuatro ALUs AM 2903 y el generador de acarreo adelantado AM 2902; U26, U27, U28, U29 y U30, respectivamente.

Como registro de estado tenemos dos AM 74S153 y un registro AM 2919 (U22, U23 y U24, respectivamente), teniendo la capacidad de guardar y almacenar el contenido de los flags en la memoria principal.

La U25, AM 2922, es el multiplexor del código de condición para el secuenciador AM 2910, gobernado por las micro-órdenes 48-51 del microprograma.

También necesitamos cuatro multiplexores que encadenen los desplazamientos internos del AM 2903 S. Para ello hemos utilizado cuatro AM 2922 (U33, U34, U35 y U36) conectándolos con los bits del microprograma 12-17 y a los pines SIO_3, QIO_3 del MSS, y SIOo, QIOo de LSS.

Tenemos también otro inversor AM 25LS175 (U37), que a su salida nos da el complemento, así como el valor real, de los bits del microprograma, μ8, μ34, μ44 y μ45. Este inversor forma parte del registro pipeline.

También nos hace falta un multiplexor para las operaciones aritméticas de dobre longitud, que se encargue del control del acarreo. Este es un AM 74S157 (U38), que estará gobernado por el μ35 del microprograma.

Necesitamos también otros registros AM 2910 (U31 y U32) que sean los registros de los datos de salida, conectando su entrada al bus Y (salida de datos de la ALUs) y su salida al bus de datos. Estarán gobernados por los bits 20 y 21 del microprograma.

Así mismo necesitaremos otros dos registros AM 2920 (U39 y U40) que hagan de MAR.

6.1.3.- ZONA DEL MICROPROGRAMA

Para almacenar el microprograma, utilizaremos cuatro PROMs de 512 palabras x 8 bits, AM 29775 (U41-U44) y nueve PROMs de 512 palabras x 4 bits, AM 27S13 (U45-U53).

El esquema final del montaje se muestra en el Plano 5.

6.1.4.- LISTA DE ELEMENTOS UTILIZADOS.

4 ..... AM 2903 ALUs
1 ..... AM 2910 Secuenciador de microprograma.
1 ..... AM 2902 Generador de acarreo adelantado.
5 ..... AM 2919 Registros.
4 ..... AM 2918 Registros.
6 ..... AM 2920 registros.
1 ..... AM 25LS399 Registros.
3 ..... AM 27S21 PROMs.
2 ..... AM 27S19 registros.
9 ..... AM 27S13 Registros.
4 ..... AM 29775 Registros.
1 ..... AM 74S139 Decodificador.

2 ..... AM 2921 Decodificador.

2 ..... AM 74S153 Decodificadores.

2 ..... AM 25LS175 Inversores.

5 ..... AM 2922 Multiplexores.

1 ..... AM 74S157 Multiplexores.

TOTAL

53 UNIDADES

==========

6.2.- ESTRUTURA DEL MICROPROGRAMA.-

Para ello vamos a emplear un microprogramación por campos. Debemos estudiar cuáles son las señales que hay que gobernar, el número de bits de instrucción de ALU, el número de bits de instrucción del secuenciador, así como un número de bits que indiquen la siguiente dirección del microprograma.

Estas señales son:

AM 2910	AM2903	MAR	Y - D	OTRAS
$\overline{RLD}$ .. 1 bit	C_n ... 1 bit	$\overline{OE}$ .. 1 bit	$\overline{DDBE}$ .. 1 bit	Flags .. 2 bits
$\overline{CLEN}$. 1 bit	B ... 1 bit	$\overline{E}$ ... 1 bit	$\overline{OE}$ ..... 1 bit	CY=0 ... 1 bit
$\overline{CC}$ ... 3 bit	A ... 2 bits		$\overline{E}$ ..... 1 bit	FDOE ... 1 bit
$\overline{CI}$ ... 1 bit	R_2... 4 bits			MMW .... 1 bit
CCP .. 1 bit	R_1 ... 4 bits			MMR .... 1 bit
I ..... 4 bits	Q .... 3 bits			POL .... 1 bit
	S .... 3 bits			$\overline{IRE}$ .... 1 bit
	$\overline{OEB}$. 1 bit			PL .....12 bits
	$\overline{EA}$ ... 1 bit			DA
	$\overline{IEN}$ .. 1 bit			CONS ... 1 bit
	$I-I_{o\ 8}$ .. 9 bits			BIT .... 1 bit
11 bits	30 bits	2 bits	3 bits	TOTAL PARCIAL: 22 bits

TOTAL DE BITS: 68.

Luego debemos tener una longitud de 68 bits de microprograma. Organicemos el microprograma de la siguiente manera:

PLD-PL8: Las 9b. instrucción del AM 2903.

PL9-PL10-PL11: Entradas de control referentes $\overline{IEN}$, $\overline{EA}$, $\overline{OEB}$. PI11 se conecta también a los registros de entrada de datos (habilitación de las salidas). Esta conexión garantiza que no habrá problemas con los pines de DB.

PL12-PL13-PL14: Seleccionan la fuente para SI0 del AM 2903, tanto para operaciones de desplazamiento a la derecha como a la izquierda.

$\mu 12$ --- $\mu 14$

14	13	12	SI0 (Shift down)	SI0o (Shift up)
L	L	L	0	0
L	L	H	SI0o	SI0
L	H	L	Q10o	Q10
L	H	H	Acarreo	Acarreo
H	L	L	Cero	Cero
H	L	H	Signo	Signo
H	H	L	No disponible	No disponible
H	H	H	1	1

$\mu 15$ --- $\mu 17$

PL15-PL16-PL17: Seleccionan la fuente para Q10 del AM 2903, tanto para las operaciones de desplazamiento a la

derecha como a la izquierda.

PL18: Cuando está LOW, activa la entrada de reloj del MAR. Los datos que aparezcan a la salida Y serán introducidos en el MAR en la transición LOW ---→ HIGH del pulso de reloj.

PL19: Cuando está a LOW, habilita la salida del MAR sobre el BUS de dirección de memoria.

PL20: Cuando está a LOW, habilita el reloj del registro de los datos de salida. Por ejemplo, los datos que aparecen en los PINES DE SALIDA Y del Am 2903 se introducirán en los registros de salida en la transición LOW-HIGH del ciclo de reloj.

PL21: Cuando está a LOW, habilita los datos de salida de los registros sobre el BUS de datos.

PL22: Cuando está a LOW, habilita el reloj del registro de entrada de datos. Por ejemplo, los datos que aparecen en el bus de datos se introducirán en registros de entrada de datos en la transición LOW a HIGH del pulso de reloj.

PL23: Es la entrada CI del secuenciador AM 2910.

PL24-PL27: Es un campo de 4 bits de ancho, que se puede usar para la "dirección - A" ó "dirección - B", o para ambas, A y B, del AM 2903.

PI28-PL31: Es un campo de 4 bits de ancho, que se puede usar para la "dirección A" del AM 2903 o para designar uno de los dieciséis bits para las entradas DA del AM 2903 a través de los AM 2921s (decodificadores 1/16)

PL32-33: Seleccionan la fuente "dirección A" del AM 2093 según la tabla:

33	32	Fuente de "Dirección A"
L	L	Bits del bus de datos 0 --- 3
L	H	Bits del microprogarma 28 --- 31
H	L	Bits del bus de datos 4 --- 7
H	H	Bits del programa 24 --- 27

PL 34: Seleccionan las fuentes del "B-ADDRESS" del AM 2903 según la tabla:

34	Fuente de "B ADDRESS"
L	Bits 4 --- 7, del bus de datos
H	Bits 24 --- 27 del μprograma

PL35: Es la entrada C_n del LSS del AM 2903 a través del mux. AM 74S157 (μ38).

PL36-PL37: Afecta a las señales de entrada del registro de estado, según la tabla:

37	36	Prox. acarreo.	Prox. cero, sign OVER FLOW
L	L	Acarreo previo.	Previo cero, sign, OVR
L	H	Previo $S10_{15}$	" "
H	L	Salida del AM 2903. S.S.	
H	H	Bits del bus de datos 0 a 3.	

PL38: Selecciona el flip-flop de acarreo o el bit PL35 para acarreo de entrada.

PL39: Cuando está a LOW, habilita la salida del registro de es-

tado a los bits del bus de datos 0, 1, 2 y 3.

PL40: Controla la polaridad de salida de uno de dieciseis bits de selección lógica.

PL41: Cuando está a LOW, habilita el reloj del registro de instrucción (U1, U2, U3). Los datos presentes entre los bits 0 y 15 del bus de datos serán guardados en el registro de instrucción en la siguiente transición LOW a HIGH del pulso de reloj.

PL42: Es una señal de salida. Cuando está a HIGH, señala a la memoria principal que se requiere una lectura memoria.

PL43: Es una señal de salida. Cuando está a HIGH señala a la memoria principal que se requiere una escritura.

PL44: Selecciona la fuente de los decodificadores, uno de dieciseis bits (U16 y U17). Cuando está a LOW la salida del registro AM 2919 (U12) que previamente se había cargado con los bits 28, 29, 30 y 31 del microprograma, se aplicará a los decodificadores. Cuando está a HIGH la salida del registro AM 2919 (U13) que previamente se había cargado con los bits 0, 1, 2 y 3 del bus de datos, se aplicará a los decodificadores.

PL45: Selecciona la fuente de la puerta DA del AM 2903 SL. Cuando está a LOW, la salida de los deocdificadores, uno de 16 bits (U16 y U17) se aplicará a la puerta. Cuando está a HIGH, la salida de las PROMs (U18 y U19) AM 29771, se aplicarán a las puertas DA del AM 2903 SL.

PL46-PL47: Son las entradas de control $\overline{RLD}$ y $\overline{CCEN}$ del secuenciador AM 2910.

PL48-PL50: Seleccionan el código de condición según la tabla:

50	49	48	Código condición seleccionado
L	L	L	Acarreo
L	L	H	Signo
L	H	L	Cero
L	H	H	Overflow

No asignados

PL51: Es el control de polaridad del código de condición cuando está a HIGH, el código de condición seleccionado no pasará invertido, y cuando esté a LOW se complementará.

PL52-PL55: Son las entradas I del secuenciador AM 2910.

PL56-PL67: Es un campo de 12 bits de ancho y sirve generalmente como la siguiente dirección de micorprograma. Sin embargo, los cinco bits menos significativos de este campo (bit 56 --- 60) sirven también como un campo de dirección de las AM 29771 (PROM "constante") U18 y U19.

6.3. EJEMPLOS DE MICRORUTINAS.-

- I N I T .

		2910					DA								2903					2910	Y-D			MAR		2903								
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags	Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I3	I5-I4	IO
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4	4	1
Bit Nº	56-67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8	1-4	0
UNIT	X	E	X	X	X	I	X	X	0	0	X	X	1	0	2	X	1	X	X	F	1	X	1	X	X	0	X	X	X	X	0	F	8	X

1. Los campos de cuatro bits en hex., los demás en octal.
2. X = Indiferente.

Esta microrutina estaría en la dirección 0 y cuando la máquina esta en RESET, el AM 2910 comenzará a ejecutar desde aquí. El propósito de esta ubicación es para poner a cero el contador de programa de la máquina (Registro 15) Al ser los bits 1, 2, 3 y 4 (AM 2903 11-14) 8_h (0001) el SL AM 2903 generará todos los ceros en los puntos F (Internos). Los bits 5, 6, 7 y 8 (15 --- 18, AM 2903) se han puesto a F_h (1 1 1 1), lo que causará que todos estos ceros originados anteriormente aparezcan en las salidas Y. El bit 9 ($\overline{IEN}$) es LOW y por lo tanto, $\overline{WRITE}$ será LOW también y estos datos se escribirán en el registro interno seleccionado por las entradas de "dirección B". Como el bit 34 es HIGH, por lo tanto, los bits 24-27 se seleccionarán como la fuente de "dirección B". Puesto que F_h (1 1 1 1) está en estos bits, en el contador de programa se escribirán todos los ceros. El bit 18 está LOW,

INIT (Cont.)

por lo tanto los datos en las salidas Y (todos cero) se cargarán en el MAR en el siguiente ciclo de reloj. Los bits 36 y 37 actualizarán los flags, poniendo CY = N = OVF = Ø , Z = 1.

Los bits 42 y 43, ambos en LOW, harán que no se envíe ninguna señal de referencia a la memoria principal (el MAR está todavía en un estado indeterminado). Los bits 52-55 (AM 2910) están puestos a E_h (Ø 1 1 1) lo cual forzará a que el secuenciador continue con la siguiente dirección secuencial, ya que CI (bit 23) está a HIGH. Los bits 21 y 39 están a HIGH para asegurar que no haya ningún conflicto en el bus de datos; si en algún caso alguno de ellos fuese una X, el bit 38 podría ser también una X, ya que el acarreo es anulado por la ALU. Teniendo un HIGH en el bit 46, se ejecuta este microprograma sin perturbar el registro interno del AM 2910. Todos los demás bits, son X.

- FETCH.

		2910					DA							2903					2910	Y-D			MAR		2903										
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags	Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I2	I5-I4	IO
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4	4	1
Bit Nº	56-67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8	1-4	0
FETCH	X	E	X	X	X	1	X	X	0	1	0	X	1	0	0	X	X	X	X	X	1	1	1	1	0	1	X	X	0	X	1	X	X	X

Este es el primer paso en la rutina FETCH (ir a por) de la instrucción de máquina. En este paso, la memoria principal está direccionada por el MAR. Se emite una señal de lectura (bit 42 = HIGH) y la instrucción de máquina (macroinstrucción) se coloca en el bus de datos por la memoria. Esta se guarda en el registro de instrucción (U1, U2, y U3) en la siguiente transición de reloj, de LOW a HIGH, (bit 41 = LOW). El bit 9 ($\overline{IEN}$ AM 2903) está en HIGH; de este modo, no nos importa lo que esté haciendo la ALU durante este micropaso. Poniendo los bits 36-38 a LOW, prevenimos a los flags de algún cambio que se pueda realizar. También los registros en la salida Y tienen la entrada $\overline{E}$ a HIGH (bits 18-20). Los bits 21 y 39 están ambos a HIGH, por lo tanto el bus de datos está libre para aceptar datos desde la memoria principal. (El bit 42 está a HIGH, señalando a la memoria una demanda de lectura). El MAR se habilita para el bus de dirección (bit 19 = LOW) y en el siguiente ciclo de reloj, la

FETCH (Cont.)

siguiente macroinstrucción se guardará en el registro de instrucción (bit 41 = LOW). El secuenciador AM 2910 continuará a la siguiente instrucción (bits 52-55 = E_h).

FETCH + 1

		2910					DA								2903					2910	Y-D			MAR		2903											
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags	Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	EA	$\overline{IEN}$	I5-I8		I5-I4		I0
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4		4		1
Bit Nº.	56-67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8		1-4		0
FETCH	X	E	X	X	X	1	X	X	0	1	0	X	1	0	0	X	X	X	X	X	1	1	1	1	0	1	X	X	0	X	1	X		X		X
FETCH + 1	X	2	X	X	X	1	X	X	0	0	1	X	1	0	0	1	1	X	X	F	1	1	1	1	0	0	X	X	0	X	0	F		4		0

Este es el segundo paso de la rutina FETCH de la macroinstrucción. La instrucción siempre está en el registro de instrucción (U1, U2, U3).

El AM 2910 recibe una instrucción JUMP MAP (bits 52-55 = 2). La siguiente microinstrucción comenzará a ejecutar la presente macroinstrucción, según la MAP PROM.

Se usa este micropaso para incrementar el contador del programa (registro 15). Estando el bit 34 a HIGH, los bits 24-27 (= F_h) del microprograma serán la "dirección de B". $\overline{OEB}$ e Io son LOW, por lo tanto, el contenido del registro 15 servirá como el operando S de la ALU. C_n está a HIGH, luego si en I1-I4 tenemos un 4_H (Ø Ø 1 Ø), se incrementará este valor. Como $\overline{IEN}$ = LOW, con I5 - I8 = F se escribirá este valor (incrementado) dentro del mismo registro (R15). Al mismo tiempo, se incrementa el MAR (bit 18 = LOW).

ADD

		2910					DA							2903					2910	Y-D			MAR		2903									
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags	Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I3	I5-I4	IO
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4	4	1
Bit Nº	56-67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8	1-4	0
ADD	FETCH + 1	7	X	X	1	1	X	X	0	1	0	X	1	0	2	0	0	0	X	X	1	1	1	1	0	1	X	X	0	0	0	F	3	0

Esto es una operación registro-a-registro. Los dos operandos residen en los registros internos señalados por los dos campos de cuatro bits de la macroinstrucción.

15 8	7 4	3 0
OP CODE	OPERANDO Nº del Reg. de destino	2º OPERANDO Nº de Regist.

Los bits 32 y 33 se ponen a LOW, los bits 0-3 del registro de instrucción se seleccionan como "dirección A". El bit 34, LOW, selecciona los bit 4-7 del registro de instrucción como "dirección B". El bit 1 (Io), bit 10 ($\overline{EA}$) y el

ADD (Cont.)

bit 11 ($\overline{OEB}$) están también a LOW, por lo tanto, los contenidos de los registros seleccionados se presentarán en las entradas R y S de la ALU. Los bits 1-4 (11-14) = 3, luego entonces la ALU ejecutará

$$F = R \text{ más } S \text{ más } C_n .$$

Hay que tener en cuenta que tanto el bit 35 como el 38 son LOW. Con 15-18 (bits 5-8) = F_h e $\overline{IEN}$ (bit 9) = LOW, el registro se escribirá en el registro interno señalado por las líneas de "dirección B".

Los bits 18 y 20 son HIGH, inhibiendo el MAR y los registros de los datos de salida que son afectados, mientras que los bits 36 y 37 (= 2) permiten que los flags asuman valores en función de los resultados de la operación.

Durante la ejecución de la función requerida (ADD en este ejemplo), se va en busca (fetch) del siguiente OP CODE en la memoria principal. Se habilita el MAR al bus de direcciones (bit 19 = LOW) y se pide una lectura de memoria (bit 42 = HIGH). Al final del micropaso la siguiente macroinstrucción se cargará en los registros de instrucción (bit 41 = LOW).

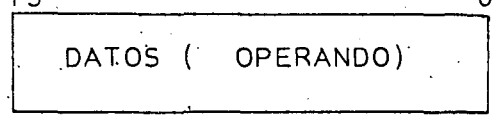
El AM 2910, secuenciador del microprograma, se prepara para seleccionar a los bits 56-67 del registro pipeline como la siguiente dirección del microprograma (bits 52-57 = 7, bit 47 = HIGH) donde se escribirá FETCH + 1 (2 en este ejemplo). El próximo paso será un JUMP MAP e incrementar el PC.

ADD IMMEDIATE (ADDIMM)

		2910					DA							2903					2910	Y-D			MAR		2903									
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags	Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I8	I5-I4	IO
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4	4	1
Bit Nº	56-67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8	1-4	0
ADDIMM	X	E	X	X	X	1	X	X	0	1	1	X	1	0	0	1	1	X	X	F	1	0	1	1	0	0	X	X	0	X	0	F	4	0
ADDIMM + 1	FETCH + 1	7	X	X	1	1	X	X	0	1	0	X	1	0	2	0	0	0	X	X	1	1	1	1	0	1	X	X	1	0	0	F	3	0

Esta microrutina de sos pasos, suma el contenido de un registro interno señalado por los bits 0-3 de la macroinstrucción con su segunda palabra, situando el resultado en el registro interno señalado por los bits 4-7 del OP CODE.

PRIMERA PALABRA DE LA MACROINSTRUCCION



SEGUNDA PALABRA(CONSEC.) DE LA MACROINSTRUCCION

El primer paso es para leer el primer operando en la memoria (bit = 19 LOW, bit 42 = HIGH) y para guardarlo en

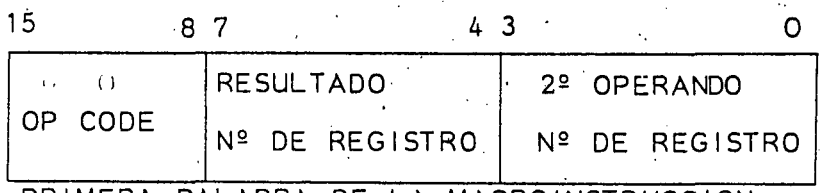
ADD IMMEDIATE (ADDIMM) (Cont.)

el registro de entrada de datos (U20 y 21) (bit 22 = LOW). Al mismo tiempo la ALU incrementa el contador de programa y el MAR (bit 18 = LOW). El secuenciador AM 2910 continuará con la siguiente dirección del microprograma. El ADDIMM + 1 es el segundo paso de esta macroinstrucción. Es muy similar al anterior, la única diferencia estriba en que el bitt 11 ($\overline{OEB}$) es HIGH, seleccionando el registro de entrada de datos como fuente de operandos para la ALU SL.

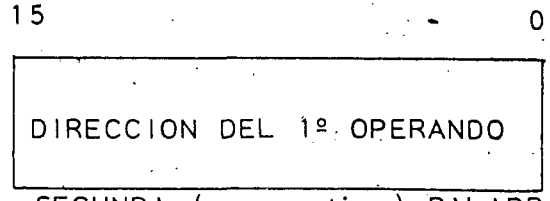
ADD DIRECT (ADD DIR)

		2910								DA														2903					2910	Y-D			MAR		2903											
	PL	I		CCP		$\overline{CC}$		$\overline{CLEN}$		$\overline{RLD}$		CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags		Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q		S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I8		I5-I4		IO
Nº de Bits	12	4		1		3		1		1		1		1		1		1				1	1	2	4		1	1		1		1		3		3	1	1	1		4		4		1
Bit Nº	56-67	52-55		51		48-50		47		46		45		43	42	41	40	39	38		36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8		1-4		0	
ADD DIR	X	E		X		X		X		1		X		0	1	1	X	1	0		0	1	1	X	X	F	1	0	1	1	0	X	X	X	0	X	0	F		4		0	

Este es el lugar de comienzo para ejecutar una macroinstrucción donde la segunda palabra es la dirección del operando.



PRIMERA PALABRA DE LA MACROINSTRUCCION



SEGUNDA (consecutiva) PALABRA DE LA MACROINSTRUCCION.

El primer paso es para leer la segunda palabra de la macroinstrucción en el registro de entrada de datos. Esta palabra es idéntica a la escrita en el lugar ADDIMM

ADD DIR + 1

		2910					DA							2903					2910	Y-D			MAB		2903								
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags	Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I8	I5-I4	I0
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4	4	1
Bit Nº	56-67	52-55		51	48-50	47	46	45	44	43	42	41	40	39	38	36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8	1-4	0
ADD DIR	X	E	X	X	X	1	X	X	0	1	1	X	1	0	0	1	1	X	X	F	1	0	1	1	0	X	X	X	0	X	0	F	4	0
ADD DIR + 1	X	E	X	X	X	1	X	X	0	0	1	X	1	0	0	0	X	X	X	X	1	1	1	1	X	0	X	X	1	X	1	X	4	0

El registro de datos de entrada contiene ahora la dirección del operando. Esta se deberá de transferir al MAR. Con el bit 0 (Io) a LOW y $\overline{OEB}$ en HIGH, lo operandos de la ALU S estarán en el bus DB. I1-I4 (bits 1-4) = 4 pasarán esta entrada a su salida, ya que C_n (bit 3) es LOW.

Con $\overline{IEN}$ (bit 9) = HIGH, la línea $\overline{WRITE}$ estará a HIGH también, asegurando que los registros internos mantengan su contenido. Como I5 - I8 (bits 5-8) = F_h, la salida de la ALU aparecerá en los pines Y de la AM 2903. Estos datos son las direcciones de los operandos y se transferirán al MAR en el siguiente ciclo de reloj. El AM 2910 continua al siguiente micropaso.

ADD DIR + 2

		2910					DA							2903					2910	Y-D			MAR		2903								
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags	Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I3	I5-I4	IO
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4	4	1
Bit Nº	56-67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8	1-4	0
ADD DIR																																		
ADD DIR + 1																																		
ADD DIR + 2	ADDIMM + 1	7	X	X	1	1	X	X	0	1	1	X	1	0	0	0	X	3	X	F	1	0	1	1	0	0	X	X	X	0	1	F	6	X

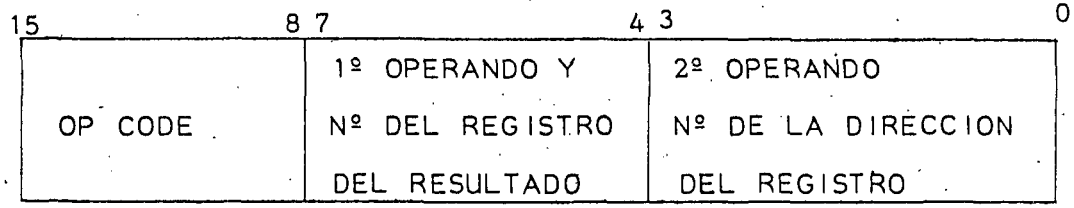
Ahora se lee el operando desde la memoria principal. El MAR se habilita al bus de dirección (bit 19 = LOW), una señal de lectura en memoria se emite (bit 42 = HIGH) y se habilita el reloj del registro de entrada de datos (bit 22 = LOW). En la siguiente transición de LOW a HIGH del ciclo de reloj, el operando se colcará en el registro de entrada de datos.

Mientras tanto, se necesita recargar la dirección de la siguiente macroinstrucción en el MAR. Los bits 32-33 = 3 seleccionan los bits 24-27 del microprograma como "dirección A"; por lo tanto el contador de programa interno, se direccionará, ya que $\overline{EA}$ (bit 10) = LOW. La ALU ejecuta F = R + C_n (bit 35) a LOW, pasando el contenido del contador de programa a la salida. El bit 9 = HIGH ($\overline{IEN}$) previene alguna posible complicación en los registros del AM 2903 y el bit 18 habilitará el MAR para recibir la dirección de la siguiente macroinstrucción.

ADD RR1

		2910					DA						2903					2910	Y-D		MAR		2903								
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags	Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I8	I5-I4	I0
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4	4	1
Bit Nº	56-67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8	1-4	0
ADD RR1	X	E	X	X	X	1	X	X	0	0	1	X	1	0	0	0	X	0	X	X	1	X	1	1	X	0	X	X	X	0	1	F	6	X

La macroinstrucción que será ejecutada aquí, señala el registro en el que se escribirá el primer operando, y también donde se escribirá el resultado. El segundo campo de 4 bits del OP CODE (bits 0-3) señala la dirección del registro donde está almacenado el segundo operando.



Los bits 32 y 33 están a LOW, por lo tanto, los bits 0-3 del registro de instrucción formarán la "dirección A". Entonces, se coge el contenido de este registro y se coloca en el MAR, exactamente de la misma manera que se hizo

ADD RR1 (Cont.)

en ADD DIR + 2 con el contador de programa. El AM 2910 continua.

ADD RR1 + 1

		2910					DA											2903					2910	Y-D			MAR		2903									
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDOE	CY=0	Flags		Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I3		I5-I4		IO
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2		1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4		4		1
Bit Nº	56- 67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37		35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8		1-4		0
ADD RR1																																					
ADD RR1 + 1	X	E	X	X	X	1	X	X	0	1	1	X	1	0	0		0	X	3	X	F	1	0	1	1	0	0	X	X	X	0	1	F		6		X

Aquí se va en busca del operando y se le coloca en el registro de la entrada de datos. Al mismo tiempo, se carga el contador de programa en el MAR.

ADD RR1 + 2

		2910					DA									2903					2910	Y-D			MAR		2903									
	PL	I	CCP	$\overline{CC}$	$\overline{CLEN}$	$\overline{RLD}$	CONS	BIT	MMW	MMR	$\overline{IRE}$	POL	FDGE	CY=0	Flags	Cn	B	A	R2	R1	CI	$\overline{DDBE}$	$\overline{OE}$	$\overline{E}$	$\overline{OE}$	$\overline{E}$	Q	S	$\overline{OEB}$	$\overline{EA}$	$\overline{IEN}$	I5-I8	I5-I4	I0
Nº de Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4	4	1
Bit Nº	56-67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8	1-4	0
ADD RR1																																		
ADD RR1 + 1																																		
ADD RR1 + 2	FETCH + 1	7	X	X	1	1	X	X	0	1	0	X	1	0	2	0	0	2	X	X	1	1	1	1	0	1	X	X	1	0	0	F	3	0

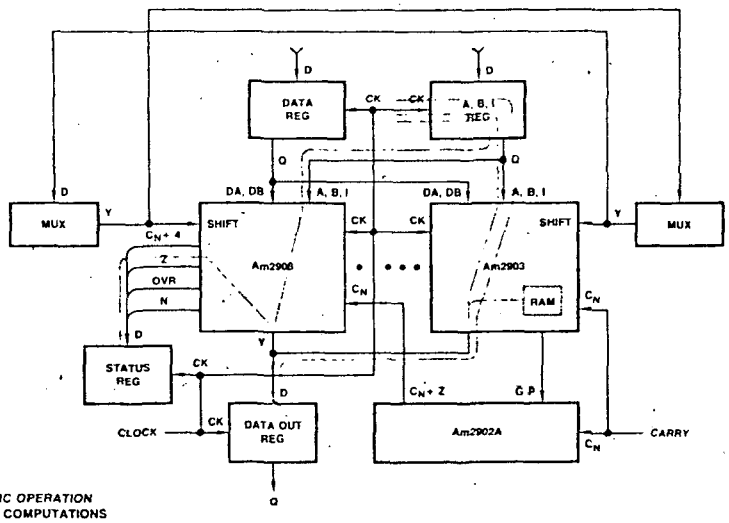
Los bits 32, 33 = 2 y los bits del registro de instrucción, 4-7, se utilizan como la "dirección A". Con el bit 34 = LOW, los mismos bits del registro de instrucción se utilizan como la "dirección B" también. Hay que tener en cuenta que $\overline{OEB}$ (bit 11) es HIGH; por lo tanto, la fuente de R será el registro de entrada de datos y la fuente de S será el registro direccionado por la "dirección A". El resultado (suma), sin embargo, se escribirá en el registro correcto, ya que $\overline{IEN}$ (bit 9) es LOW.

ANEXO

ANEXO 1

RUTAS DE RETARDO DEL AM 2903

a)



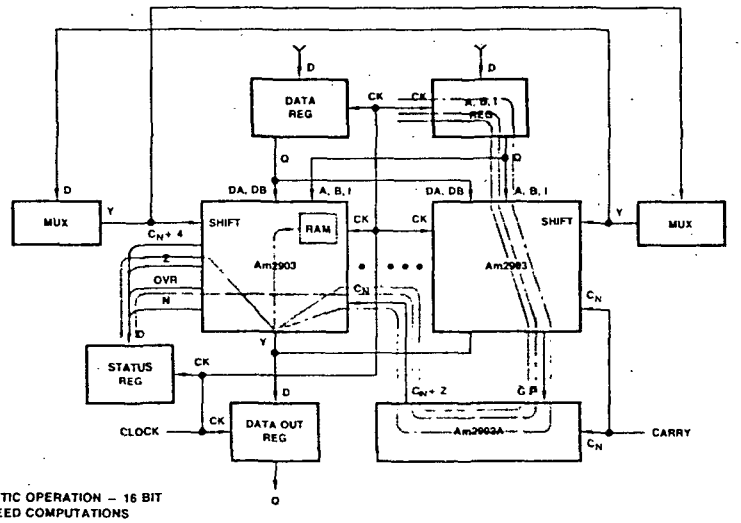
LOGIC OPERATION
SPEED COMPUTATIONS

DEVICE NO	DEVICE PATH	PATH 1	PATH 2	PATH 3
S - REG	CP to O	9	9	9
2903	A, B to Y	56	56	56
2903	Y to Z	-	16	-
S-REG	SET-UP D	2	2	-
2903	SET-UP Y	-	-	9
TOTAL-ns		67	83	74

PATH 1
PATH 2
PATH 3

MPR-540

b)



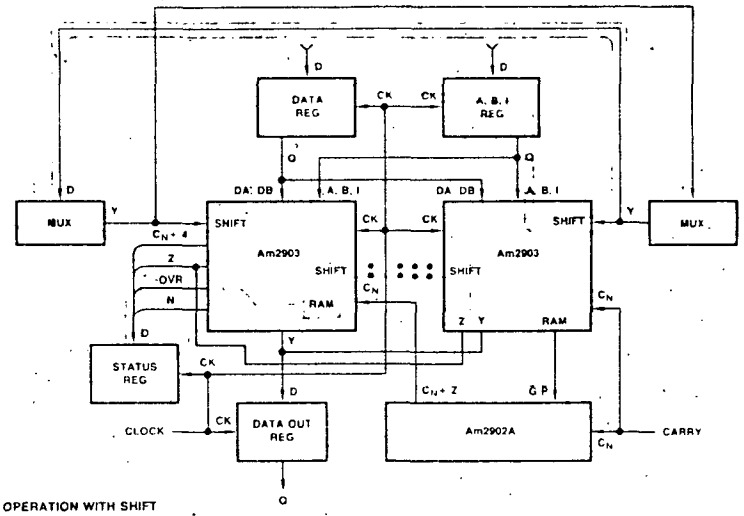
ARITHMETIC OPERATION – 16 BIT
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S-REG	CP to O	9	9	9
2903	A, B to G, P	56	56	56
2902A	G, P to C_{n+xyz}	5	5	5
2903	C_n to Y	25	-	25
2903	C_n to FLAG	-	38	-
2903	Y to Z	16	-	-
S-REG	SET-UP D	2	2	-
2903	SET-UP Y	-	-	9
TOTAL-ns		113	110	104

PATH 1
PATH 2
PATH 3

MPR-541

c)

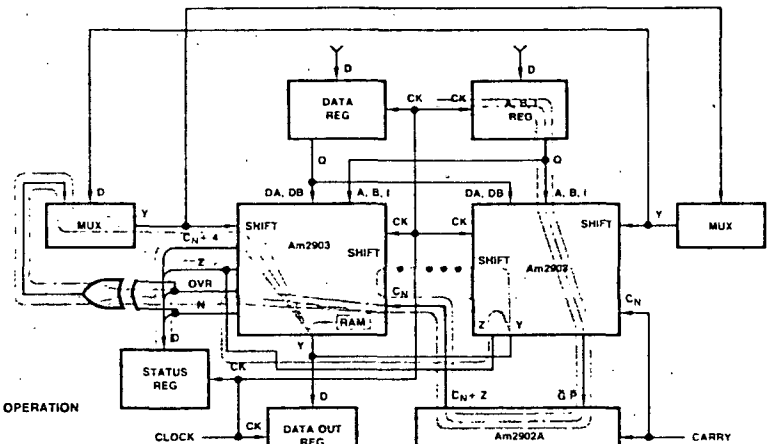


LOGIC OPERATION WITH SHIFT
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S - REG	CP to O	9	9	9
2903	A, B to S_0	64	64	64
MUX	D to Y	5	-	5
2903	S_3 to Y	13	13	13
2903	Y to Z	16	16	-
S-REG	SET-UP D	2	2	-
2903	SET-UP Y	-	-	9
TOTAL-ns		109	104	100

PATH 1
PATH 2
PATH 3

d)



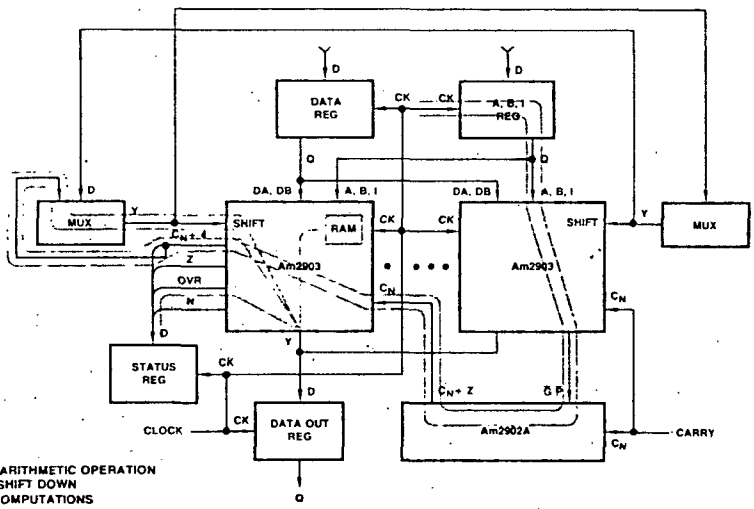
TWO'S COMPLEMENT ARITHMETIC OPERATION
WITH SHIFT DOWN – 16 BIT
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S - REG	CP to O	9	9	9
2903	A, B to G, P	56	56	56
2902A	GP to C_{n+xyz}	5	5	5
2903	C_n to SIO_0	21	-	-
2903	SIO_3 to Y	13	-	-
2903	C_n to N, OVR	-	38	38
S-EXOR	IN to OUT	-	7	7
S-MUX	D to Y	-	5	5
2903	SIO_3 to Y	-	13	13
2903	Y to Z	16	16	-
2903	SET-UP Y	-	-	9
S-REG	SET-UP D	2	2	-
TOTAL-ns		122	151	142

PATH 1
PATH 2
PATH 3

MPR-543

e)



MAGNITUDE ONLY ARITHMETIC OPERATION
WITH SHIFT DOWN
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2
S - REG	CP to O	9	9
2903	A, B to G, P	56	56
2902A	GP to C_{n+xyz}	5	5
2903	C_n to C_{n+4}	21	21
S-MUX	D to Y	5	5
2903	SIO_3 to Y	13	13
2903	Y to Z	16	-
S-REG	SET-UP D	2	-
2903	SET-UP Y		9
TOTAL-ns		127	118

PATH 1
PATH 2

MPR-544

ANEXO 2

CARACTERISTICAS DE CONMUTACION DEL AM 2903

A. Am2903 SWITCHING CHARACTERISTICS (TYPICAL ROOM TEMPERATURE PERFORMANCE) – (MAY 18, 1978)

Tables IA, IIA, and IIIA define the nominal timing characteristics of the Am2903 at 25°C and 5.0V. The Tables divide the parameters into three types: pulse characteristics for the clock and write enable, combinational delays from input to output, and set-up and hold times relative to the clock and write pulse.

Measurements are made at 1.5V with V_{IL} = 0V and V_{IH} = 3.0V. For three-state disable tests, C_L = 5.0pF and measurement is to 0.5V change on output voltage level.

TABLE IA – Write Pulse and Clock Characteristics

Time	
Minimum Time CP and $\overline{WE}$ both LOW to write	15ns
Minimum Clock LOW Time	15ns
Minimum Clock HIGH Time	35ns

TABLE IIA – Combinational Propagation Delays (All in ns)
Outputs Fully Loaded. CL = 50pF (except output disable tests)

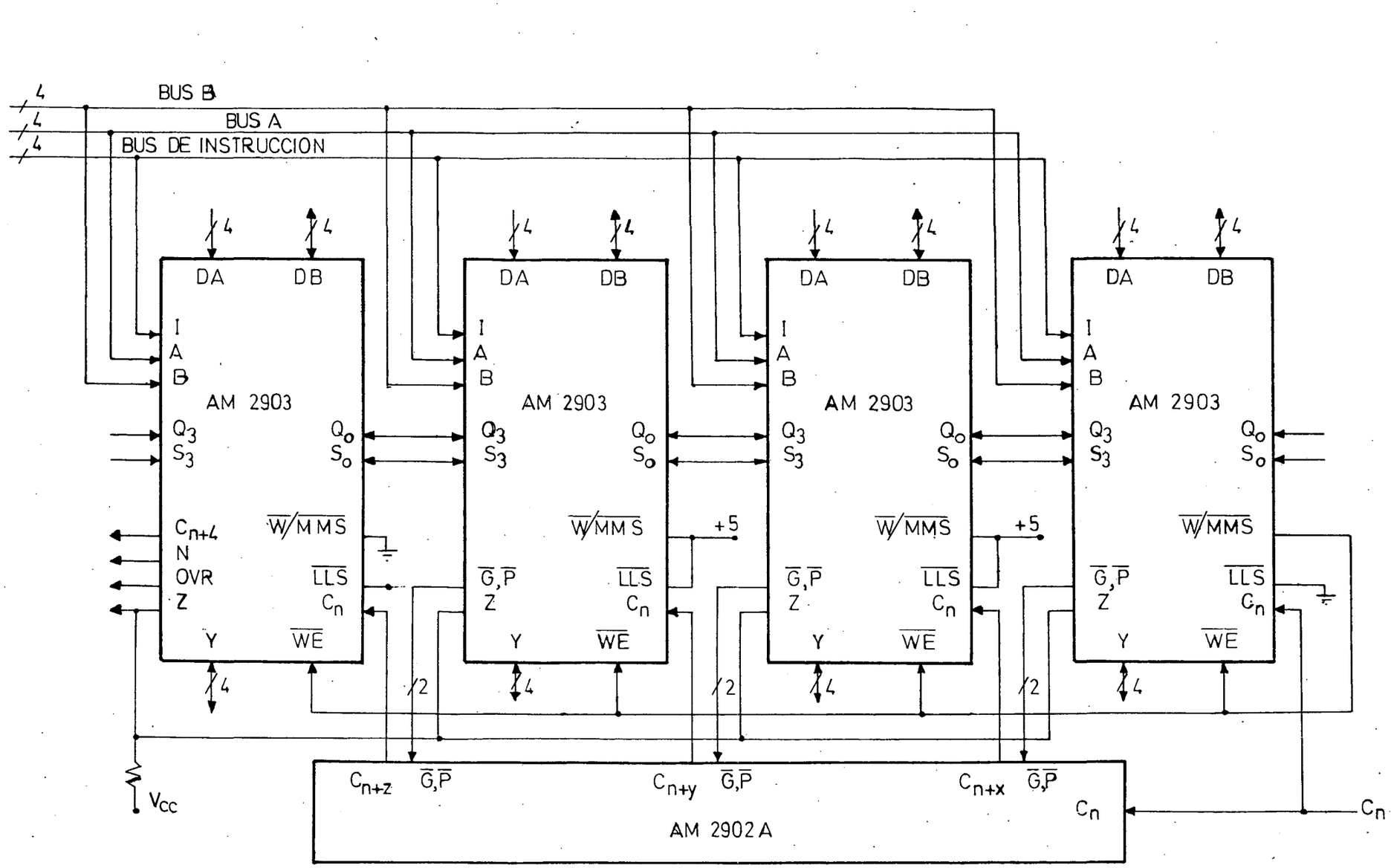
From Input \ To Output	Y	C_{n+4}	$\overline{G}$, $\overline{P}$	(S) Z	N	OVR	DB	$\overline{WRITE}$	QIO_0, QIO_3	SIO_0	SIO_3	SIO_0 (Parity)
A, B Addresses (Arith. Mode)	65	60	56	–	64	70	33	–	–	65	69	87
A, B Addresses (Logic Mode)	56	–	46	–	56	–	33	–	–	55	64	81
DA, DB Inputs	39	38	30	–	40	56	–	–	–	39	47	60
$\overline{EA}$	38	33	26	–	36	41	–	–	–	36	41	58
C_n	25	21	–	–	20	38	–	–	–	21	25	48
I_0	40	31	24	–	37	42	–	15(1)	–	41	39	63
I_{4321}	45	45	32	–	44	52	–	17(1)	–	45	51	68
I_{8765}	25	–	–	–	–	–	–	21	22/29(2)	24/17(2)	27/17(2)	24 17(2)
$\overline{IEN}$	–	–	–	–	–	–	–	10	–	–	–	–
$\overline{OEB}$ Enable/Disable	–	–	–	–	–	–	12/15(2)	–	–	–	–	–
$\overline{OEY}$ Enable/Disable	14/14(2)	–	–	–	–	–	–	–	–	–	–	–
SIO_0, SIO_3	13	–	–	–	–	–	–	–	–	–	19	20
Clock	58	57	40	–	56	72	24	–	28	56	63	76
Y	–	–	–	16	–	–	–	–	–	–	–	–
$\overline{MSS}$	25	–	25	–	25	25	–	–	–	24	27	24

Notes: 1. Applies only when leaving special functions.
2. Enable/Disable. Enable is defined as output active and correct. Disable is a three-state output turning off.
3. For delay from any input to Z, use input to Y plus Y to Z.

TABLE IIIA – Set-Up and Hold Times (All in ns)
CAUTION: READ NOTES TO TABLE III. NA = Not Applicable; no timing constraint.

Input	With Respect to to this Signal	HIGH-to-LOW		LOW-to-HIGH		Comment
		Set-up	Hold	Set-up	Hold	
Y	Clock	NA	NA	9	–3	To store Y in RAM or Q
$\overline{WE}$ HIGH	Clock	5	Note 2	Note 2	0	To Prevent Writing
$\overline{WE}$ LOW	Clock	NA	NA	15	0	To Write into RAM
A,B as Sources	Clock	19	–3	NA	NA	See Note 3
B as a Destination	Clock and $\overline{WE}$ both LOW	–4	Note 4	Note 4	–3	To Write Data only into the Correct B Address
QIO_0, QIO_3	Clock	NA	NA	10	–4	To Shift Q
I_{8765}	Clock	2	Note 5	Note 5	–18	
$\overline{IEN}$ HIGH	Clock	10	Note 2	Note 2	0	To Prevent Writing into Q
$\overline{IEN}$ LOW	Clock	NA	NA	10	–5	To Write into Q

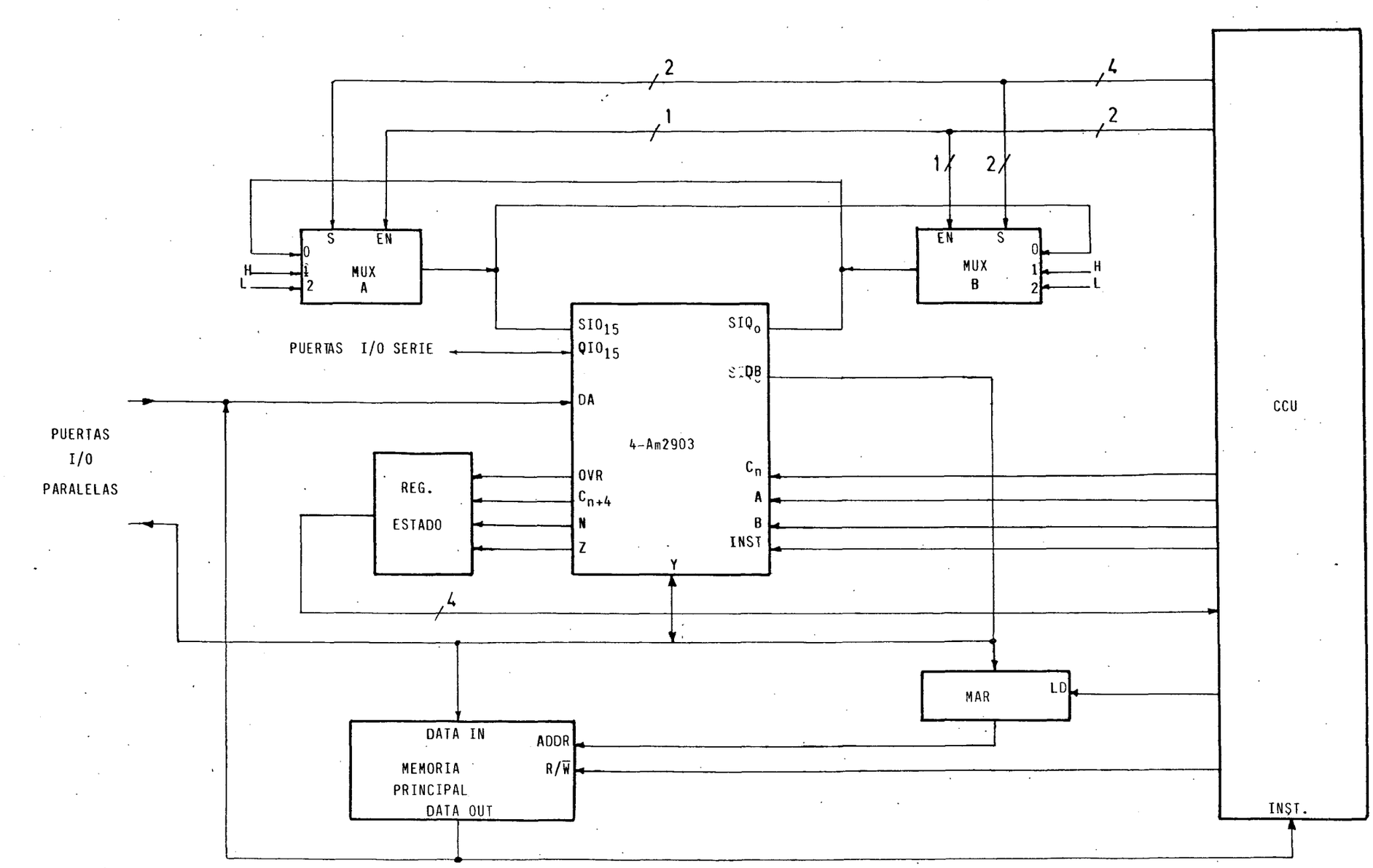
PLANOS



PLANO 1

CONEXION DE 4 Am 2903 CON UN Am 2902 A

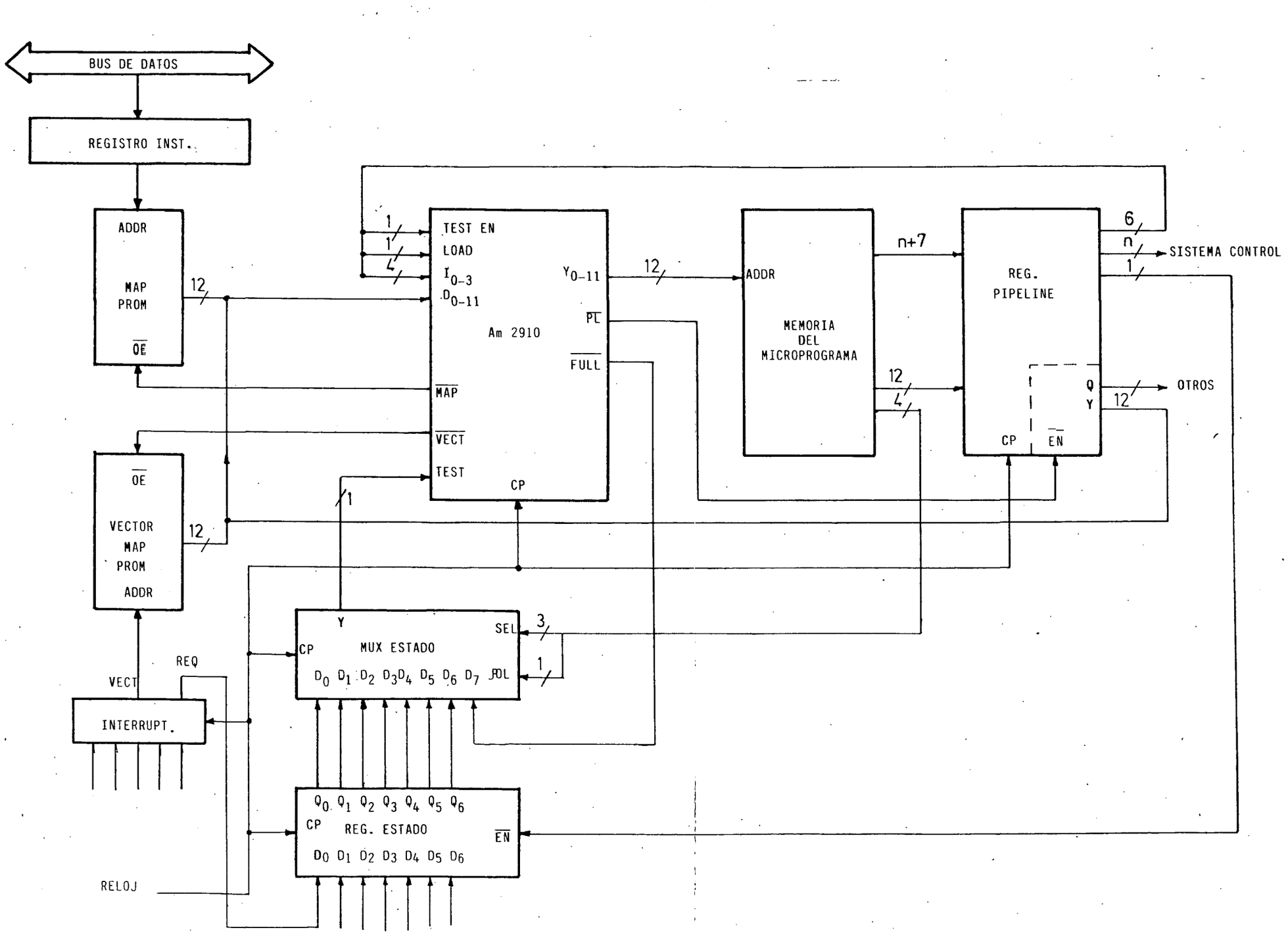
PLANO	Nº1	CONEXION DE 4 Am 2903 con un 2902 A
PROYECTO		CPU 16 BITS CON MICROPROCESADORES MICROPROGRAMABLES
AUTOR: ANGEL GALLO LUÑO		TUTOR: SEBASTIAN SUAREZ GIL
FECHA 6/2/82		E. U. I. T. T.
ESCALA s/n		LAS PALMAS DE GRAN CANARIA



PLANO 2

ZONA DE TRATAMIENTO DE DATOS

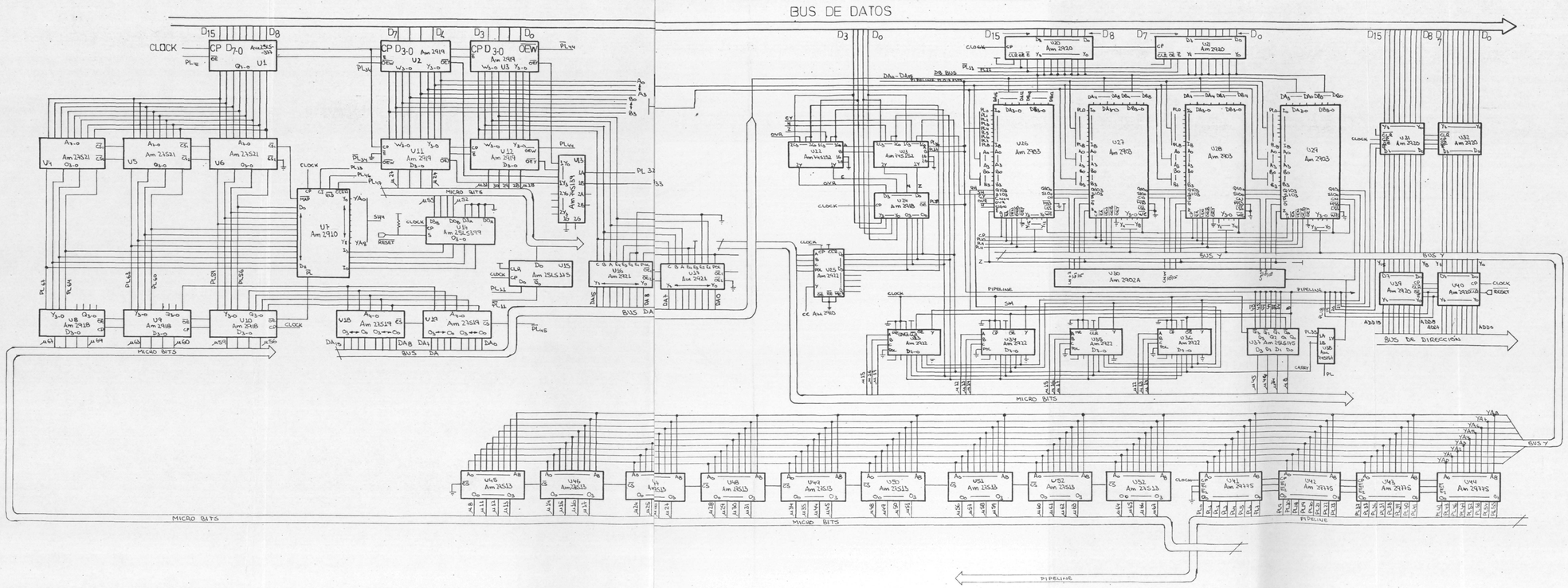
PLANO	Nº 2	ZONA DE TRATAMIENTO DE DATOS
PROYECTO:		CPU DE 16 BITS CON MICROPROCESADORES MICROPROGRAMABLES
AUTOR: ANGEL GALLO LUÑO		TUTOR: SEBASTIAN SUAREZ GIL
FECHA: 6/2/82		E. U. I. T. T.
ESCALA: s/n		LAS PALMAS DE GRAN CANARIA



PLANO 3

UNIDAD DE CONTROL DEL MICROPROGRAMA

PLANO	Nº 3	UNIDAD DE CONTROL DEL MICROPROGRAMA
PROYECTO: CPU 16 BITS CON MICROPROCESADORES MICROPROGRAMABLES		
AUTOR: ANGEL GALLO LUÑO		TUTOR: SEBASTIAN SUAREZ GIL
FECHA: 6/2/82	E. U. I. T. T.	
ESCALA: s/n	LAS PALMAS DE GRAN CANARIA	



PLANO	Nº 5	ESQUEMA GENERAL DE LA CPU
PROYECTO		CPU 16 BITS CON MICROPROCESADORES MICROPROGRAMAB
AUTOR: ANGEL GALLO LUÑO		TUTOR: SEBASTIAN SUAREZ GIL
FECHA 5/2/82		E. U. I. T. T.
ESCALA s/n		LAS PALMAS DE GRAN CANARIA