CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION COMPUTACION

PROCESADOR EDUCATIVO CON LOS CIRCUITOS AMD29300.

Tesis que presenta el Ing. Guillermo Lara Gómez para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA con opción en COMPUTACION.

Trabajo dirigido por el Dr. Jan Janecek.

México D.F., Junio 1990.

A mis padres y hermano.

AGRADECIMIENTOS:

Primero quiero manifestar mi agradecimiento al Dr. Jan Janecek por todo el apoyo que me proporcionó y sin el cual no hubiese sido posible la realización de este proyecto.

Mi agradecimiento también al Ing. Rodolfo Rosado Martínez por las facilidades prestadas para el uso del laboratorio a su cargo, así como para mi compañero Eduardo Cortés Gil, por su valiosa ayuda en este proyecto. Y un reconocimiento para el Consejo del Sistema Nacional de Educación Tecnológica (COSNET), por el apoyo económico que me brindó.

Guillermo Lara Gómez.

INTRODUCCION:

El presente trabajo describe el proyecto de tesis llamado "Procesador Educativo con los Circuitos AMD29300" (que se le ha dado el nombre de MIRAGE), la cual se compone de dos partes principales: el diseño físico del procesador (tipo RISC) y un modelo computacional del mismo. Teniendo como objetivo principal ayudar a los estudiantes de materias como Arquitectura de Computadoras, a la mejor comprensión de la arquitectura RISC, cuyas características la hacen ser una de las más eficientes de la actualidad. Con este objetivo se previo la implementación de un modelo computacional y no sólo el diseño físico del procesador.

La realización de este trabajo es descrito en los siguientes capítulos:

En el capítulo 1 se hace mención a los orígenes, características y ventajas de la arquitectura RISC. Algunas veces contrastándola con otros tipos de arquitectura.

El capítulo 2 da un panorama de las características RISC del procesador implementado con los circuitos de la línea AMD29300 (MIRAGE), así como información de características propias.

El capítulo 3 nos presenta algunas deficiencias de la arquitectura RISC y la manera en que se trato de solucionarlas en un procesador llamado Clipper.

En el capítulo 4 se describe la solución propuesta en el procesador MIRAGE.

En el capítulo 5 se proprociona una descripción del modelo computacional del procesador MIRAGE.

En el capítulo 6 se dan detalles del diseño físico del procesador MIRAGE.

CONTENIDO

1.-COMPUTADORAS DE CONJUNTO REDUCIDO DE INSTRUCCIONES (RISC)

1.1.-ARQUITECTURA MICROPROGRAMADA.

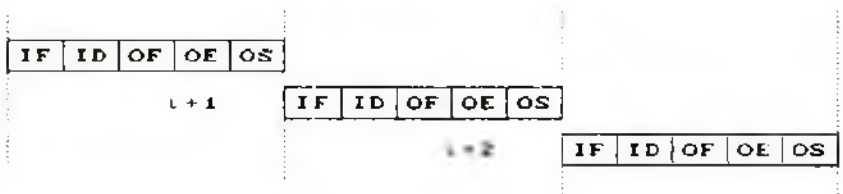
Una de las inovaciones tecnológicas del sistema IBM/360, (uno de los iniciadores de la arquitectura de computadoras moderna) fue la microprogramación. La microprogaramación se auxilio de una pequeña memoria de control, que fue una elegante forma de construir una unidad de control para el procesador con un gran conjunto de instrucciones. Cada palabra de control en la memoria es llamada microinstrucción y los contenidos son esencialmente un intérprete de instrucciones, programado en microinstrucciones. Las memorias principales de estas máquinas fueron núcleos magnéticos y las memorias de control fueron usualmente 10 veces más rápidas que un núcleo magnético.

Las minicomputadoras manufacturadas intentaron seguir la tendencia de los mainframes y la microprogramación se popularizó. Y con el desarrollo de los semiconductores pronto la industria de las minicomputadoras contaba en forma estandar con memoria principal de núcleos magnéticos y con memoria de control de semiconductor tipo ROM (memoria de sólo lectura), con lo cual también hubo un crecimiento en la riqueza y complejidad del conjunto de instrucciones.

Un intento de nuevo diseño fueron los WCSs (Writable Control Store) (1) que en lugar de usar ROMs usan RAMs, con el objeto de brindar una mayor flexibilidad al proporcionar la facultad de que el programador pueda construir sus propias instrucciones. Sin embargo los WCSs no se convirtieron en un diseño popular ya que

la anterior característica también exigía de un mayor esfuerzo por parte del programador. Aunque la motivación para la creación de WCS (las instrucciones no deben ser más rápidas que las microinstrucciones y que los programadores deben escribir operaciones simples que mapeen directamente con microinstrucciones) fue aun válida.

Dado el amplio y poderoso conjunto de instrucciones de estas máquinas se les llamo computadoras de conjunto de instrucciones complejas (CISC). Estas máquinas CISC se caracterizan también por tener un modelo de ejecución secuencial, es decir se ejecuta una nueva instrucción hasta que la anterior instrucción se ha terminado de realizar como se muestra en la fig.1.1, en donde se puede observar que el ciclo de intrucción está dividido en cinco partes fundamentales las cuales son: traer la instrucción desde la memoria (IF), decodificación de dicha instrucción (ID), traer los operandos especificados (OF), ejecución de la operación indicada con los operandos (OE) y el almacenamiento de los operandos (OS).



SECUENCIAL

IF - INSTRUCTION FETCH
ID - INSTRUCTION DECODE
OF - OPERAND FETCH
OE - OPERAND EXECUTE
OS - OPERAND STORE

fig.1.1.- Modelo de ejecución secuencial de un CISC.

Un problema que presentan los sistemas tradicionales VLSI-CISC es el tamaño de su unidad de control, por ejemplo el Motorola 68000, el cual toma el 50% del área del chip para la unidad de control. Debido a que con un gran conjunto de instrucciones, formatos y modos de direccionamiento, la unidad de control debe ser compleja para manejar todas estas opciones. Esto deja poco espacio para los registros internos de la CPU. Siendo esto importante para minimizar el número de accesos a memoria.

Lo anterior trae como consecuencia que sea difícil el hacer uso de tecnologías como la de arseniuro galio (GaAs), cuya principal ventaja es la velocidad, los dispositivos realizados con dicha tecología son alrededor de 5 veces más velores que los de silicio, pero tiene una limitante sobre la densidad de integración de componentes.

1.2.-MAQUINAS RISC.

La arquitectura RISC se ha convertido en un diseño muy popular no únicamente en los laboratorios de investigación de Universidades, sino también en un gran número de industrias de sistemas de computo.

Los principales atributos de los RISCs (4) consisten en la simplicidad y el diseño básico, un reducido menú de selección (pequeño conjunto de instrucciones, formatos y modos de direccionamiento) y un eficiente, lineal y rápido manejo de instrucciones por la unidad de control del sistema (preferentemente la ejecución de una instrucción en un ciclo de máquina). Un RISC puede ser caracterizado por los siguientes puntos:

1- Un relativo bajo número de instrucciones (deseablemente menor a 100).

2- Un bajo número de modos de direccionamiento (deseablemente de 1 a 4).

3- Un bajo número de formatos de instrucción (desablemente de 1 a 4) todos de la misma longitud.

4- Un solo ciclo de máquina para la ejecución de cada una de las instrucciones.

5- Memoria acresada únicamente por instrucciones del tipo LOAD/STORE.

6- Un arreglo de registros de CPU relativamente grande (arriba de 32).

7- Unidad de control alambrada.

8- Extenso soporte por compiladores de lenguajes de alto nivel.

Los puntos 1, 2 y 3 permiten una simplificación, de la unidad de control de cada RISC. Las instrucciones multiciclo tales como instrucciones aritméticas de punto flotante son ejecutadas en software o en coprocesadores. Y modos de direccionamiento más complejos deberán ser desarrollados a partir de los simples.

El punto 3 permite a los RISC una simplificación de la decodificación de instrucción. Los operandos registro del RISC de Berkeley están en el mismo lugar en la palabra de 32 bits, así

que cada acceso a registro puede tomar lugar simultáneamente con la decodificación del código de operación. Las instrucciones de talla simple también simplifican la memoria virtual, puesto que ellas no pueden ser puestas en partes que envuelvan diferentes páginas.

Del punto 5 se desprende que el conjunto de instrucciones, el procesador y el manejo de faltas de página en un medio ambiente de memoria virtual son grandemente simplificados, el tiempo de ciclo de reloj es reducido también.

1.2.1.-PIPELINE.

Un concepto básico de la arquitectura RISC y que en adelante se manejará es el pipeline, que se refiere a un modelo de ejecución de instrucciones. Para la mejor comprensión se muestra la siguiente figura en la cual se puede observar un modelo de ejecución de pipeline, donde se ha particionado el trabajo total de una instrucción y las diferentes piezas de la instrucción son ejecutadas en paralelo.

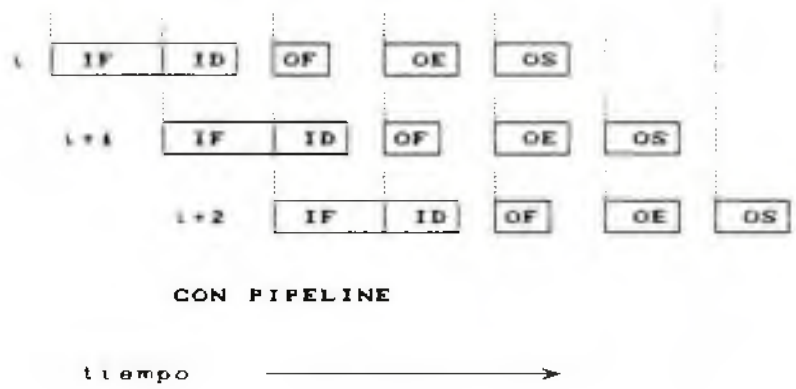


fig. 1.2.1.- Ejecución de tres instrucciones con pipeline.

Un posible problema del pipeline se presentaría por una instrucción de salto, pero se resuelve normalmente retrazando el pipeline hasta que la nueva instrucción de la dirección de salto es traída. Varias máquinas con pipeline han elaborado técnicas para traer la instrucción apropiada después de cada instrucción de salto, pero estas técnicas son demasiado complicadas para los RISC. La solución genérica de los RISC, comunmente usada en conjuntos de microinstrucciones, es redefinir saltos para que ellos no tomen efecto hasta después de la siguiente instrucción; esto es llamado "salto retardado". El retardo del salto permite a los RISC traer siempre la próxima instrucción durante la ejecución de la instrucción actual.

En la fig.1.2.2 se puede observar como los saltos y dependencias de datos reducen la efectividad del pipeline. La instrucción INC A no puede traer el operando A, hasta que la

instrucción previa ADD B,C,A finalice el almacenamiento de B+C en A, forzando a una burbuja (retardo) de dos etapas en el pipeline. Los saltos retrasan el pipeline hasta que la instrucción de la dirección de salto pueda ser traída.

El código de lenguaje de máquina es convenientemente arreglado para que los resultados deseados sean obtenidos. La carga de esta tarea la llevan los programadores del compilador, el optimizador, y el debugger. El retardo de salto también mueve las burbujas de salto normalmente asociadas con la ejecución de pipeline.

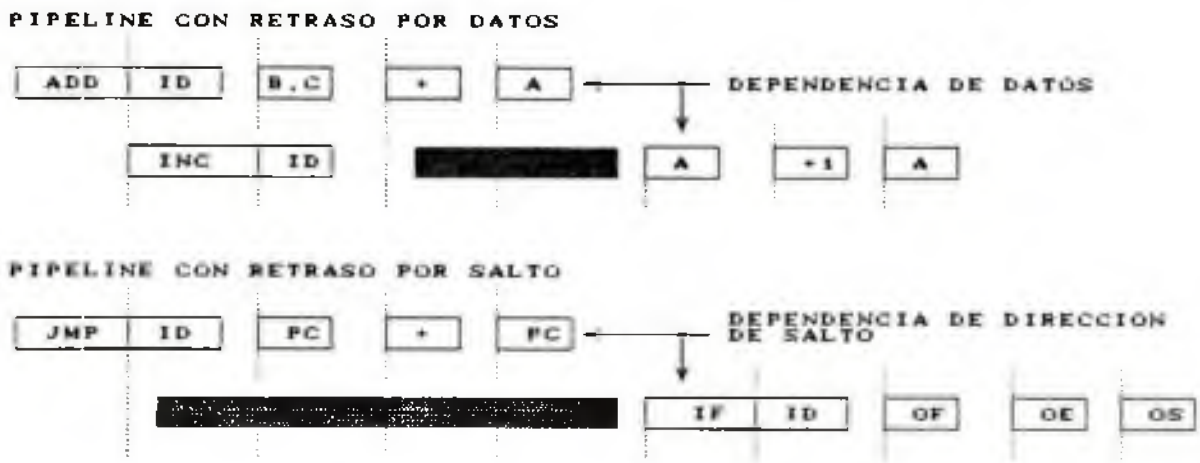


fig.1.2.2.- Retrazo por datos y por salto en el pipeline.

1.3.- PROTOTIPOS DE DESARROLLO RISC.

El primer sistema identificado como un RISC fue el RISC de Berkeley, la primer CPU tipo RISC implementada en un solo chip. Dos modelos fueron desarrollados: RISC I y RISC II (4). Ambos tienen básicamente la misma arquitectura: RISC I tiene 31 instrucciones, mientras que RISC II tiene 39. Ambas máquinas son de 32 bits, contando con un gran arreglo de registros de CPU: 78 registros en el RISC I y 138 registros en el RISC II. Esto no significa que si un programa es escrito, digamos, para el RISC II, los 138 registros estarán disponibles para ser usados libremente. En realidad, cada uno de los procedimientos creados tiene solamente 32 registros a su disposición. En otras palabras, cada procedimiento en un RISC de Berkeley trabaja únicamente con 32 registros de la CPU.

Los registros disponibles para cada procedimiento en el RISC de Berkeley, son subdivididos en dos partes:

- Registros Globales, R0, R1,..., R9: un total de diez. Estos registros pueden ser usados para todos los procedimientos del mismo programa corriendo en un RISC.

- Ventana de Registros, R10, R11,..., R31: un total de 22. Estos registros son únicos para un procedimiento partícular y ellos podrían corresponder a un conjunto diferente de los registros actuales para un procedimiento diferente. Por ejemplo, un procedimiento puede usar los registros R10-R31 como su ventana de registros mientras otro usa R106-R127.

Sin embargo existe una cierta relación entre la ventana de registros de un procedimiento y la ventana de otro procedimiento. Por ejemplo (fig.1.3.2), el procediemiento B, puede llamar al procedimiento C y B puede en su turno ser llamado por el procedimiento A. Cuando un procedimiento llama a otro, puede pasarle ciertos parámetros. Aquí es donde la ventana de registros entra fig.1.3.1 y fig.1.3.2. Los 22 registros de la ventana son subdivididos en tres grupos:

- R10-R15: grupo bajo, seis registros, contiene parámetros de valores pasados al procedimiento llamado.

- R16-R25: grupo local, diez registros, contiene los parámetros locales del procedimiento.

- R26-R31: grupo alto, seis registros, contiene los valores de los parámetros pasados a el procedimiento actual por el procediemiento llamante.

La principal ventaja de una ventana de registros es el hecho de salvar tiempo de cómputo, mientras se pasan parámetros en una llamada a un procedimiento. En los RISC los parámetros permanecen en los mismos registros físicos y no involucran movimientos de datos. En otros sistemas existe un movimiento de datos actuales entre la CPU y memoria, consumiendo un considerable tiempo de cómputo. Teóricamente podría existir un gran arreglo de ventana de registros en un CISC. En la práctica, este no es el caso, dado que una gran parte del área de silicio es ocupada por la unidad de control.

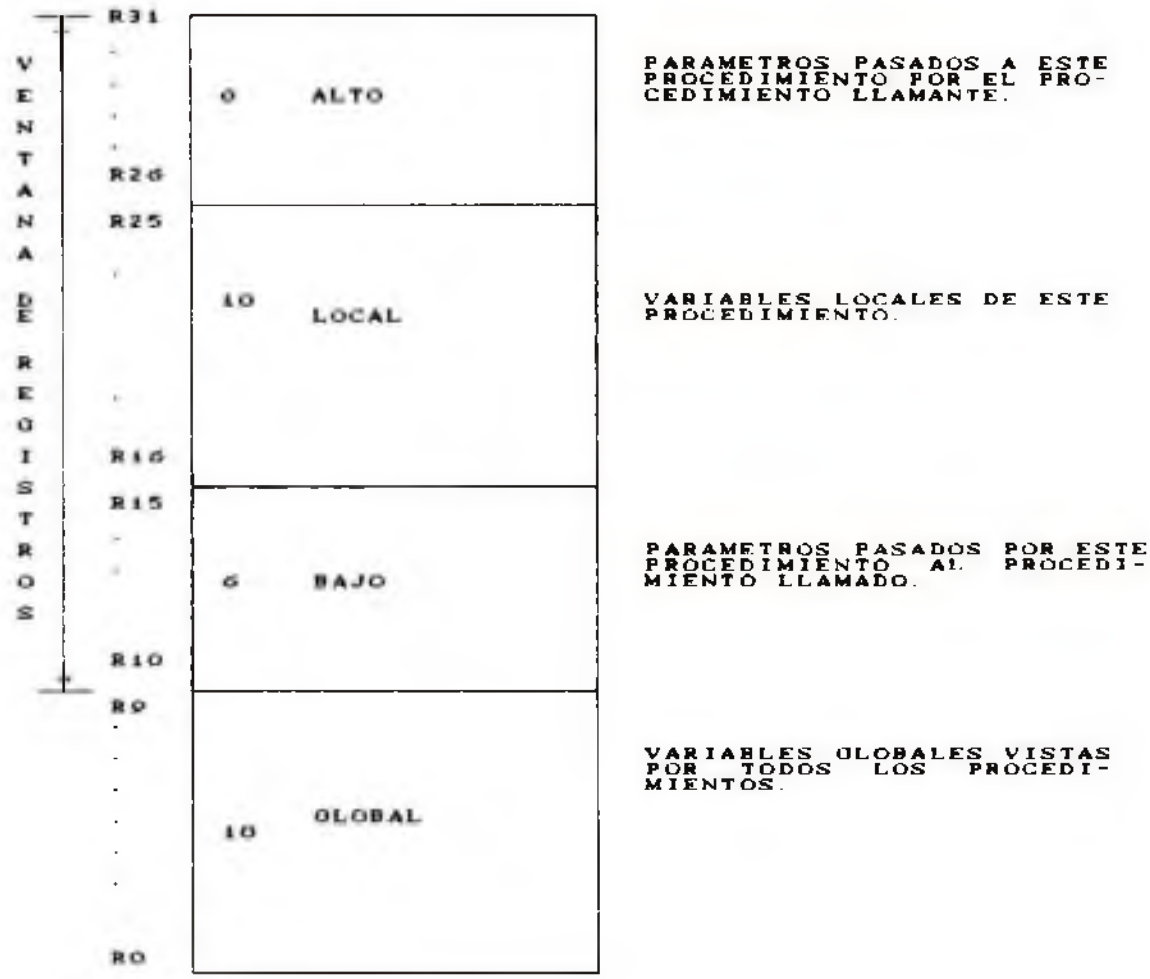


fig. I.3.1.- Registros de la CPU del RISC de Berkeley, vistos por cada procedimiento.

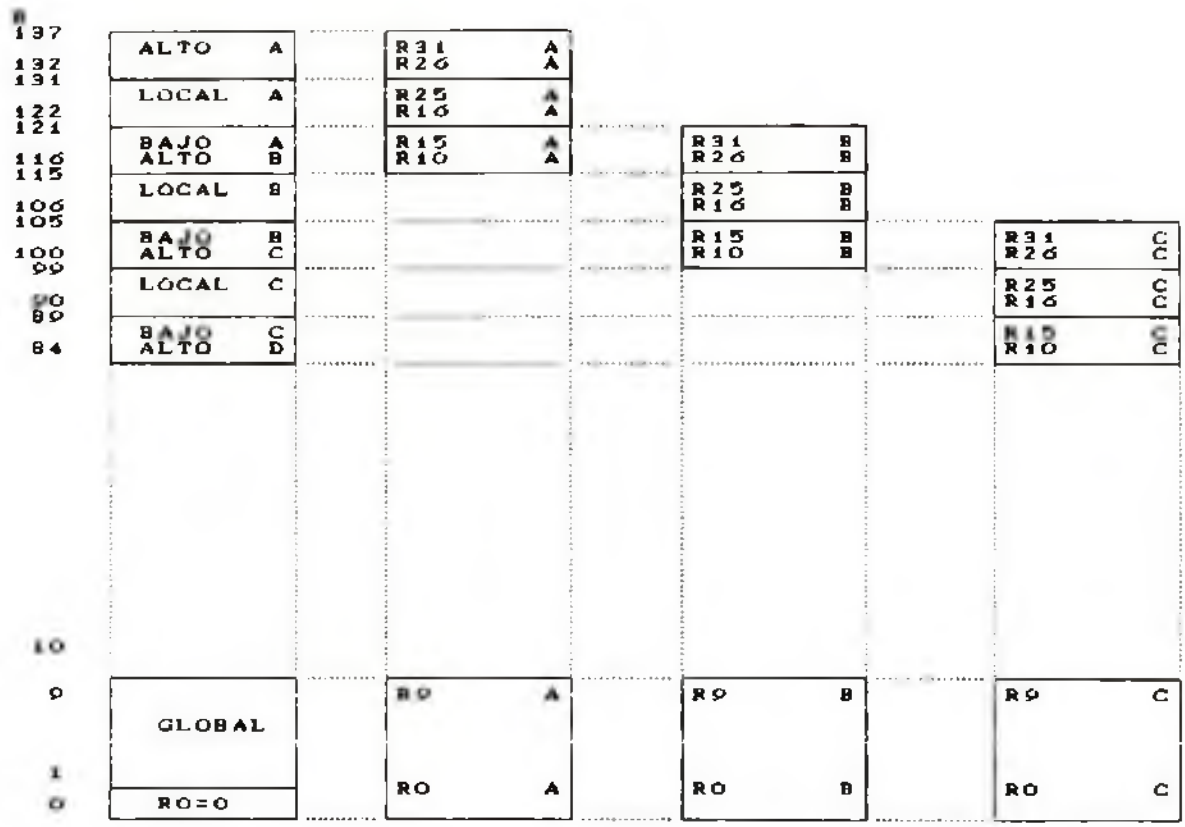


fig. 1.3.2.- Arreglo de registros del RISC II de Berkeley.

El segundo sistema RISC experimental fue desarrollado en la Universidad de Stanford.

Una de las principales diferencias entre el RISC de Stanford y el de Berkeley, es que la CPU tipo RISC de Stanford manejada por el pipeline, puede necesitar un resultado, generado por la instrucción que la precede (i-1), manejada por otra etapa del pipeline.

En el sistema de Stanford este mecanismo es manejado por software en vez de hardware. Específicamente, el MIPS de Stanford usa una sofisticada reorganización de ensamblador. Esta "reorganización" ensambla el código en un código de máquina ejecutable, empaqueta dos instrucciones independientes en una sola, donde sea posible y reorganiza el código. La reorganización de código completa el enlace del pipeline por la sustitución de instrucciones útiles de cualquier lugar en la línea de código, minimizando el uso de NOP (no operación) insertados. Esta es una solución puramente por software. Por esta razón el RISC de Stanford es llamado: microprocesador sin etapas entrelazadas de pipeline (MIPS).

1.3.1.- RISC TIPO MIRIS.

Un nuevo sistema tipo RISC fue anunciado, el MIRIS (7), su desarrollo se inició en la Universidad George Mason (GMU) comenzando el proyecto en Junio de 1986 y siendo desarrollado con los circuitos de la línea AMD2900. Su característica particular es que contrario a muchos de los sistemas RISC ya existentes, su

control está en base a un microcódigo en vez de un alambrado. El microcódigo MIRIS no tiene la estructura estandar con numerosas microsubrutinas tal como en otras máquinas con microcódigo. El microcódigo es almacenado en una memoria PROM de 256 x 64 en la CPU. Los ocho bits más significativos de su palabra de instrucción (32 bits) es el código de operación de cada instrucción que es una dirección de una localidad de la memoria de microcódigo. Cada palabra de control de 64 bits en la memoria de microcódigo corresponde a una instrucción específica de lenguaje de máquina. Cuenta además con un gran arreglo de registros de CPU de 2048 x 32, usando el método de ventana de registros. El diseño del sistema es tal que la característica antes mencionada no afecta la rapidez de ejecución de las instrucciones.

Otra de las características del MIRIS es la simplicidad y regular carácter primitivo de sus intrucciones de lenguaje de máquina. Esto permite una ejecución uniforme de cada instrucción en un solo ciclo de CPU.

2.-CARACTERISTICAS RISC DEL PROCESADOR MIRAGE.

El procesador MIRAGE básicamente será RISC del tipo MIRIS (7), y de las características ya mencionadas en el capítulo 1 el procesador cumple con las siguientes:

- Un número relativamente pequeño de instrucciones (aproximadamente 50).
- Un bajo número de modos de direccionamiento.
- Un bajo número de formatos de instrucciones (5), todos de la misma longitud.
- Ejecución de instrucciones en un solo ciclo.
- Unidad de control en base a una palabra de control de 82 bits, direccionada por una parte de la instrucción traida de la memoria de programa.
- Accesos a memoria a través de instrucciones simples LOAD/STORE.
- Un arreglo de registros relativamente grande (64).

2.1.- DESCRIPCION DE LOS CIRCUITOS DE LA LINEA AMD29300 UTILIZADOS.

Para el diseño del procesador MIRAGE se requirió de tres circuitos de la línea AMD29300 que jugaron un papel preponderante y de los cuales resulta ser ventajoso tener una descripción, como a continuación se ofrece.

2.1.1.-ALU AM29332.

El AM29332 (10) es una Unidad Aritmética y Lógica (ALU) de alto rendimiento. Como se puede observar en la fig.2.1.1 el chip cuenta con dos puertos de entrada (A y B), cada uno con un ancho de 32 bits y un puerto de salida (Y) también de 32 bits.

El AM29332 puede realizar operaciones aritméticas de uno, dos, tres y cuatro bytes contando para ello con el generador de máscaras de la fig.2.1.1. También puede realizar aritmética de multiprecisión y múltiples corrimientos de bits, auxiliándose para ello del funnel shifter (registro de corrimiento de embudo) de la misma figura.

Para operaciones lógicas puede manejar campos de longitud variable. El chip cuenta con hardware para realizar algoritmos de multiplicación y división para datos signados y no signados. Para llevar a cabo estos algoritmos es que se usa el bloque de registro Q y su registro de corrimiento, mostrado en la fig.2.1.1.

El bloque ALU de la figura cuenta con tres entradas y usa a la máscara como un segundo o tercer operando en cada operación. El estado de cada operación (carry, negative, overflow, link) se aplica a el resultado únicamente sobre el ancho especificado. La ALU es capaz de realizar corrimientos a la izquierda en dos bits para el algoritmo de multiplicación, corrimientos a la derecha de un bit para el algoritmo de división y para corrimientos simples. La ALU también puede realizar aritmética BCD.

De la figura 2.1.1 se puede observar que existe un status register (registro de estado) de 32 bits, el cual tiene información necesaria acerca de las operaciones realizadas en la ALU y que es almacenada después de cada ciclo de su reloj.

La lógica de paridad mostrada en la figura 2.1.1 permite que para cada uno de los bytes de las entradas DA y DB haya asociado un bit de paridad (8 en total), si es detectado un error en la paridad se envía una señal de error (parity error en true). Existen también 4 señales de paridad (una por byte) para el bus de salida Y.

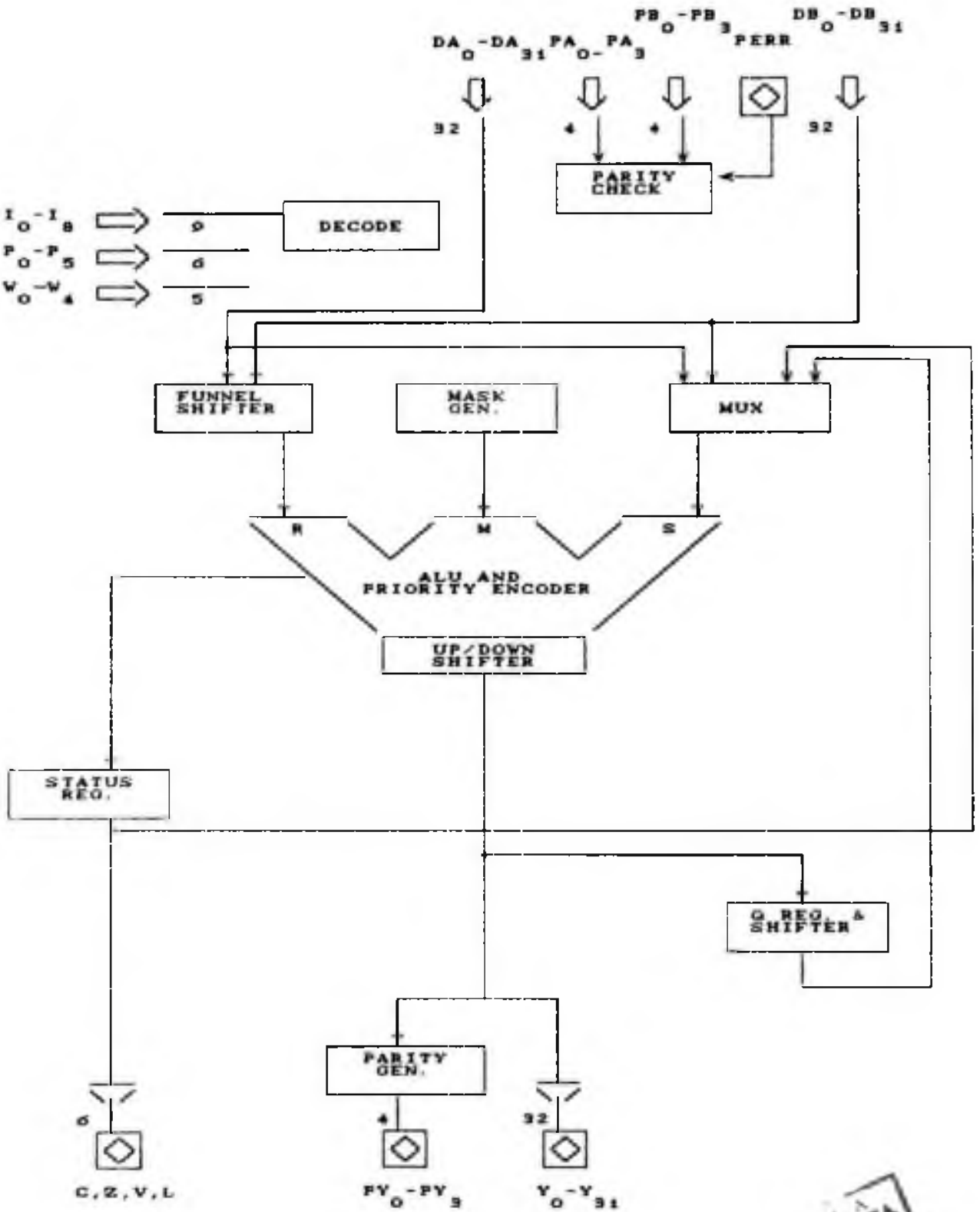


fig.2.1.1.- Diagrama a Bloques del AM29332.

TIPOS DE DATOS QUE PUEDE MANEJAR LA ALU.

La ALU soporta los siguientes tipos de datos (10):

1. Enteros.
2. BCD.
3. Campo de longitud variable de bits.

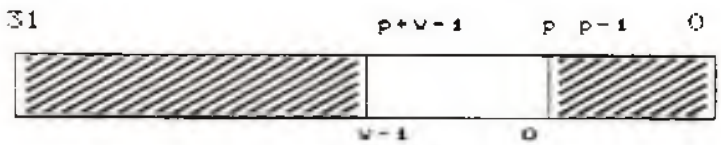
Los primeros dos tipos caen dentro de la categoría de operandos de bytes de límite alineado. La talla del operando puede ser de 1, 2, 3 o 4 bytes. Todos los operandos tienen al bit menos significativo alineado (bit 0). El ancho de bytes está determinado por los bits I_8 y I_7 (que corresponden a los bits más significativos del código de operación a efectuar por la ALU, fig.2.1.1) de la operación (10), como se muestra en la siguiente tabla.

I_8	I_7	Ancho en bytes
0	0	4
0	1	1
1	0	2
1	1	3

El tercer tipo de datos tiene operandos de campo variable (1 a 32 bits) como se muestra en la figura siguiente. El operando es especificado por entradas de ancho (W_0-W_4) y entradas de posición (P_0-P_5), mostradas en la fig.2.1.1. Las entradas de posición indican la posición del bit menos significativo del operando. Dependiendo de los bits I_8 e I_7 de la operación, las entradas de

ancho y posición pueden ser tomadas del Registro de Estado o de las terminales de Ancho y Posición como se muestra en la siguiente tabla.

I_0	I_7	Posición		Ancho	
		Pins	Reg	Pins	Reg
0	0	x		x	
0	1	x			x
1	0		x	x	
1	1		x		x



Campo de Longitud Variable de Bits.

2.1.2.- ARREGLO DE REGISTROS AM29C334.

El AM29C334 (11) es una RAM con palabra de 64x18 bits, con dos puertos de lectura y dos puertos de escritura (A y B). Es factible realizar dos accesos independientes simultáneos y cada acceso pude ser de lectura o escritura.

La razón para la existencia de registros de 18 bits en lugar de 16 bits, es que se pensó en tener los 16 bits más bajos para el dato a ser almacenado y los dos bits más altos para paridad (un bit de paridad por cada byte de datn).

El bloque principal en la fig.2.1.2 es la RAM de doble acceso. Pero también se puede obseravar en esta figura la existencia de dos multiplexores para seleccionar entre lectura y escritura (MUX para los puertos A y SMUX para los puertos B). También se puede ver que existen 2 LATCHES, uno por cada uno de los puertos de salida.

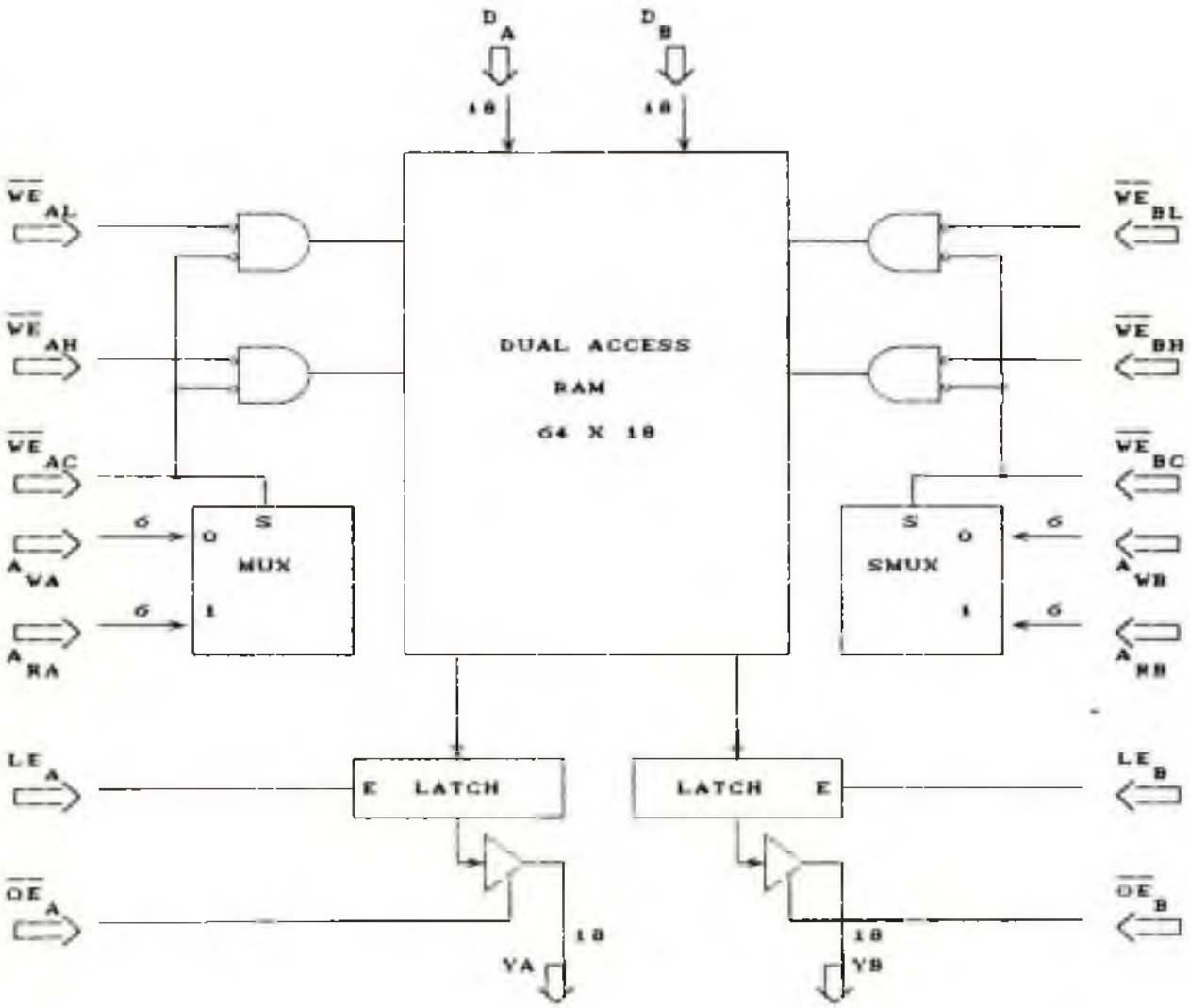


fig.2.1.2.- Diagrama a Bloques del AM29C334.

2.1.3.- SECUENCIADOR AM29C331.

El AM29C331 (9) es un chip de gran rapidez, con un ancho de 16 bits diseñado para el control de secuencia de ejecución de microinstrucciones, almacenadas en una memoria de microprograma. El conjunto de operaciones está diseñado para parecerse a las intrucciones de lenguajes de alto nivel.

El mayor de los bloques de la figura 2.1.3 es el multiplexor de direcciones (ADDRESS MUX) el cual puede seleccionar una dirección de entre alguna de las siguientes fuentes:

1.- Una dirección de salto alimentada por el bus D.
2.- Una diercción de salto alimentada por el bus A.
3.- Una dirección de salto multiforma. La cual está formada por la sustitución de los 4 bits menos significativos de la dirección especificada en el bus D (D3, D2, D1, D0) con uno de los cuatro conjuntos (M0X, M1X, M2X o M3X) de cuatro bits de direcciones de salto multiforma, siendo seleccionado uno de estos cuatro conjuntos por el Multiway Mux.
4.- Un return o dirección de loop del tope de pila.
5.- La del Micro-Program Counter.

De la fig. 2.1.3 se puede observar también una pila (stack) de 33x16 bits, que puede almacenar direcciones de regreso, direcciones de ciclo y valores de conteo. Los datos de salida son pasados al multiplexor de direcciones, al contador o al bus de datos D.

El contador de la fig.2.1.3 puede ser utilizado como contador en ciclos. Y puede ser cargado del bus D, bus A o por un pop de la pila.

El bloque de punto de ruptura (BREAK PT. LOGIC) contiene un comparador de direcciones que permite puntos de ruptura en el microcódigo para sondeo (debugging) y conteo de cuantas veces se ha ejecutado una microinstrucción en una dirección específica.

El bloque de decodificación de operaciones (INSTR DECODE) se encarga de decodificar hasta 64 posibles operaciones y el bloque de lógica de prueba (TEST LOGIC) puede seleccionar de entre 12 bits de prueba. El manejo de interrupciones es realizado por el bloque REAL TIME INT LOGIC.

Algunos ejemplos de operaciones que puede realizar el secuenciador son las siguientes:

FOR D — Inicializa un ciclo, insertando al stack la dirección de la microinstrucción siguiente y carga el contador con lo que está en las entradas del bus de datos D y continúa con la próxima microinstrucción en micromemoria, su código de operación es 31H.

NEXT — Si el contador no es igual a 1, decrementa el contador y salta a la dirección en el tope del stack. Si el contador es igual a uno, entonces decrementa el contador y realiza un POP del stack y continúa con la siguiente microinstrucción, su código de operación es 2FH.

GOTO D — Es un salto incondicional a la dirección señalada en D, su código es 20H.

BRCC D — Si la condición de prueba CC está en alto, salta a la dirección especificada por D. Si CC está en bajo continúa con la siguiente microinstrucción, su código de operación es 00H.

CONT	Continúa con la siguiente microinstrucción, su código es 30H.
RESET_SP	Borra el stack pointer y continúa, su código es 36H.
GOTO A	Es un salto incondicional a la dirección señalada en A, su código es 24H.

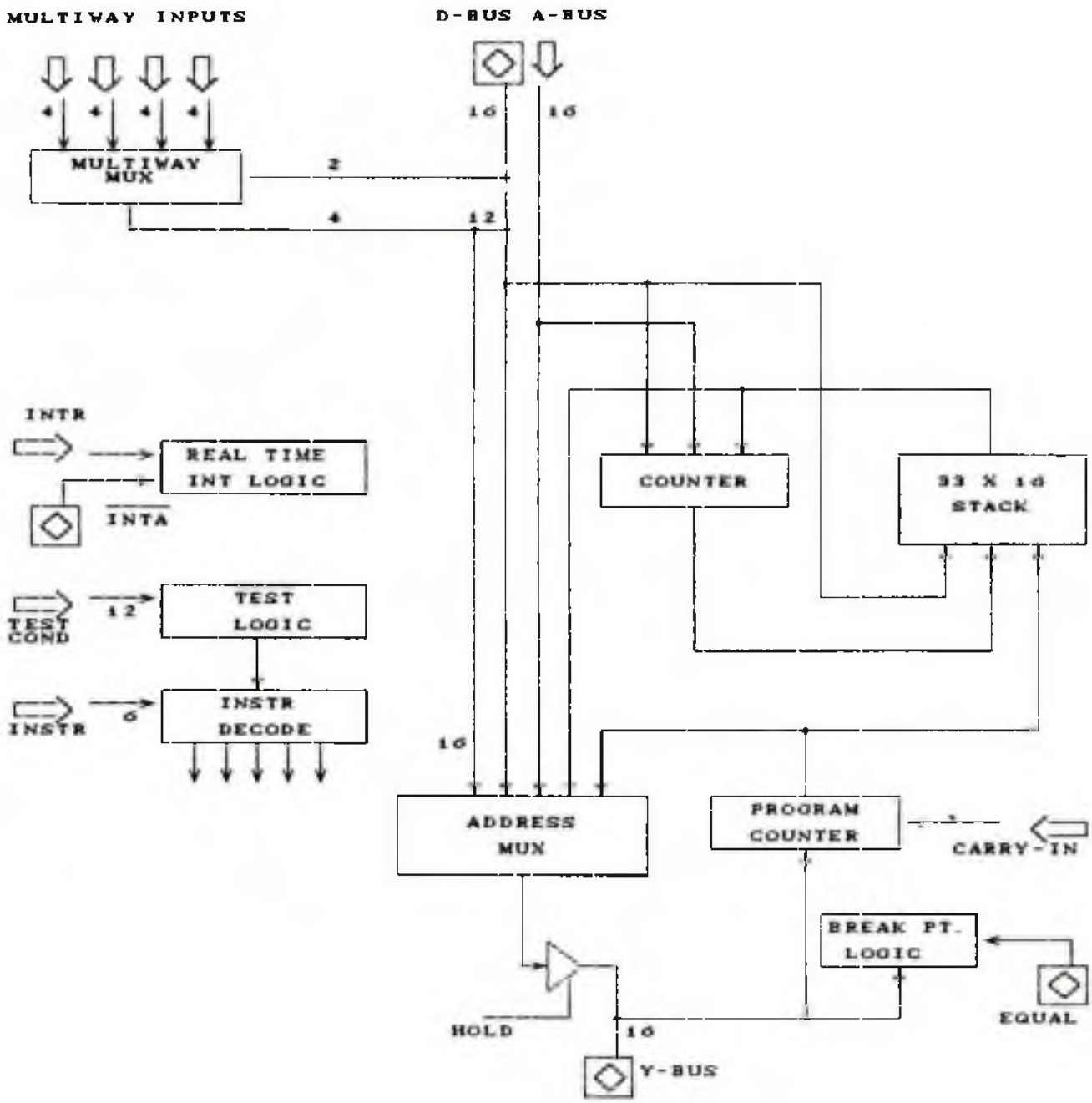


fig.2.1.3.- Diagrama a Bloques del AM29C331.

2.2.- DESCRIPCION GENERAL DEL PROCESADOR MIRAGE.

El diagrama general mostrado en la fig. 2.1 da una visión general de la arquitectura del sistema. Como se puede apreciar este procesador tiene buses de datos de 32 bits y buses de instrucciones de 32 bits, al igual que otros procesadores de tipo RISC. En cambio a diferencia de otros procesadores de su tipo, este tiene control microprogramado el cual no tiene estructura clásica con numerosas microsubrutinas, sino que cada instrucción de máquina corresponde a una palabra de control de 82 bits en la micromemoria. El rápido acceso permite producir una palabra de control de 82 bits, capaz de activar hasta 82 líneas de control en forma simultánea. Este diseño es sumamente flexible ya que permite agregar, quitar o modificar microinstrucciones en forma relativamente fácil.

DIAGRAMA A BLOQUES DEL PROCESADOR MIRAGE

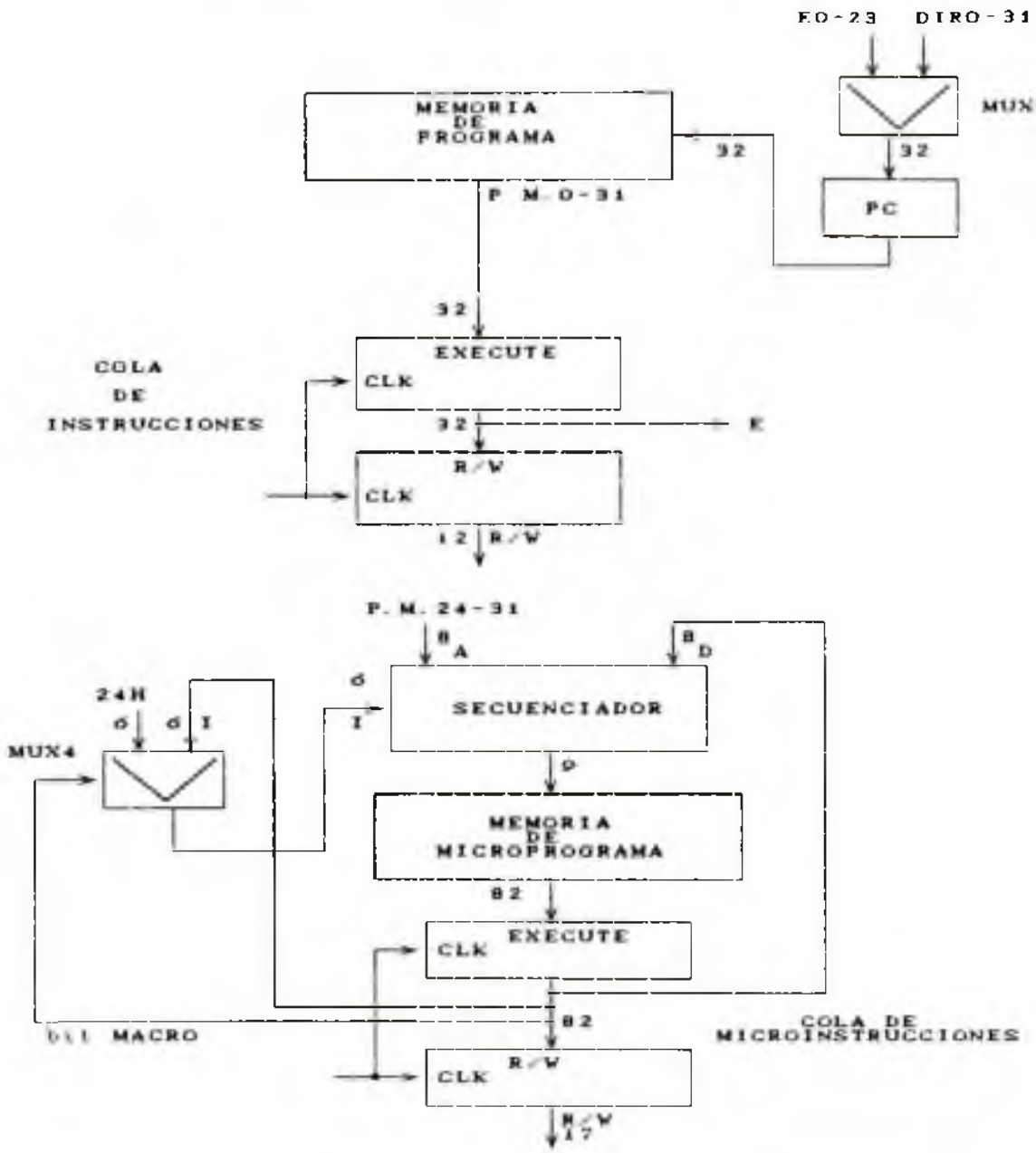


fig.2.1(parte 1).- Sección de Control de Programa.

DIAGRAMA A BLOQUES DEL PROCESADOR MIRAGE.

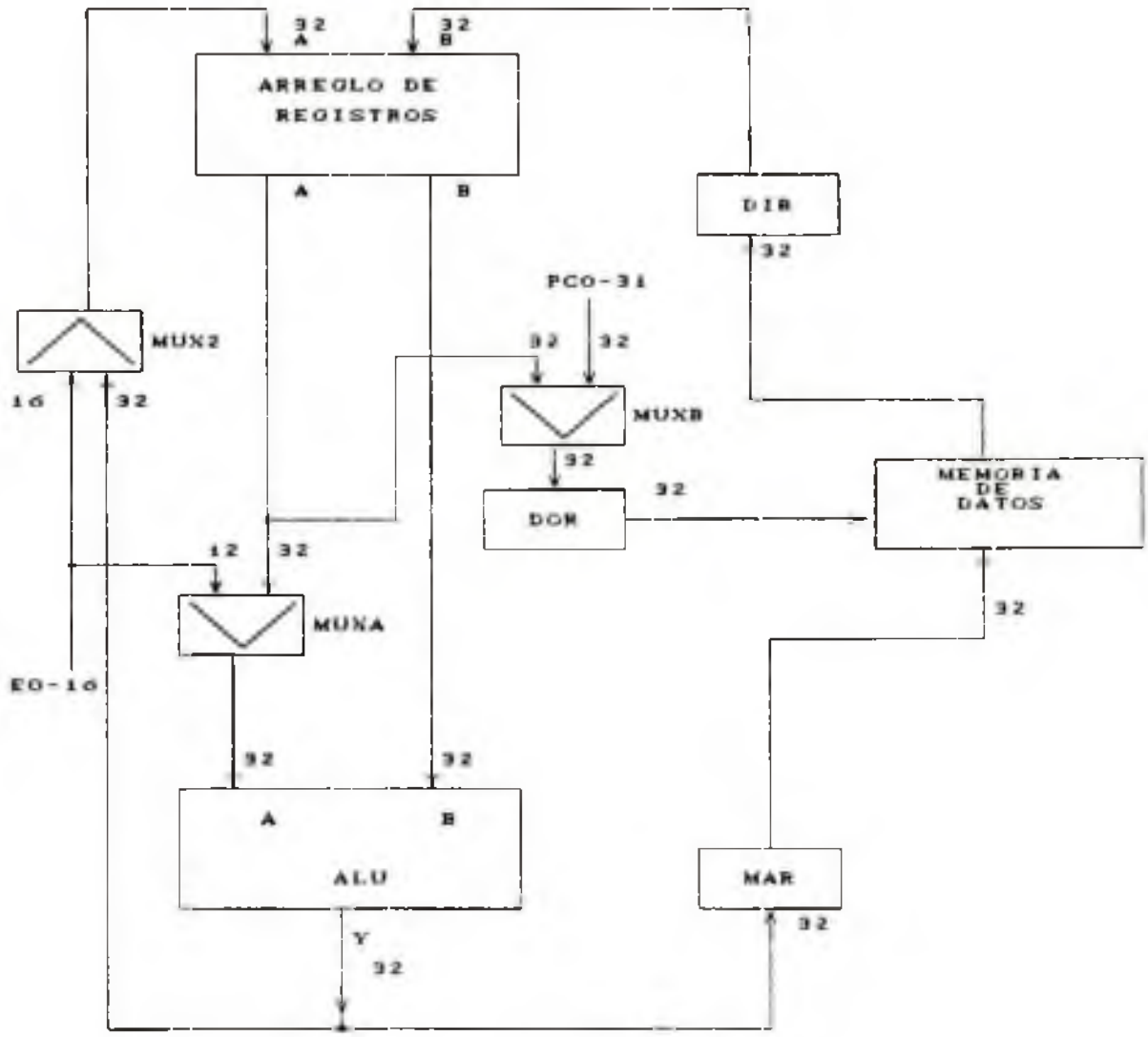


fig.2.1(parte 2).-Sección de Ejecución de Datos.

En la fig. 2.1 puede observarse que el procesador consiste de: un arreglo de registros el cual tiene el puerto A de entrada conectado a un multiplexor MUX2 que selecciona un dato proveniente de la ALU o un dato inmediato, el puerto B de entrada del arreglo es usado cuando se va a alamacenar un dato proveniente de la memoria de datos. Los dos puertos de salida del arreglo de registros van hacia la ALU, pero el puerto A de salida de este arreglo antes de llegar a la ALU se encuentra con un multiplexor MUXA, que puede seleccionar de entre este dato proveniente del arreglo o un dato inmediato.

Se puede notar que existen dos multiplexores MUX(A,B), uno, como ya se explicó (MUXA), para que se seleccione un dato inmediato que servirá para calcular una dirección de una localidad en memoria de datos o un dato proveniente del arreglo. Para cuando este multiplexor selecciona un dato inmediato, el otro multiplexor MUXB selecciona un dato del puerto de salida A del arreglo de registros que se almacena en el registro DOR, el cual se escribirá en la localidad de memoria de datos direccionada por el cálculo realizado en la ALU y almacenado en el registro MAR, de este mismo multiplexor se puede observar que otra de sus entradas sirve para cargar un dato proveniente de PC a memoria de datos.

El multiplexor MUX2 es el encargado de elegir entre un dato inmediato y un dato proveniente de la ALU. El dato seleccionado será enviado al puerto A de entrada del arreglo de registros.

La función del multiplexor MUX3 es cargar a PC con un dato proveniente de memoria de datos con el fin de poder implementar un regreso de subrutinas, al utilizar a la memoria de datos como almacén de una dirección de regreso o bien con un dato inmediato con el objeto de realizar saltos ya sea condicionados o incondicionales, cabe hacer notar que la carga al PC para un

salto se realiza en la fase EXECUTE de tal manera que antes de hacer el salto se realizará la instrucción siguiente a la que define dicho salto.

En cuanto a los registros MAR, DOR y DIR tienen como función respectivamente el ser interfaces para direcciones, datos a ser escritos y para datos de salida de la memoria de datos.

En la fig. 2.1 también se puede notar que existen dos registros de la memoria de programa los cuales tienen nombres: EXECUTE y R/W; dichos registros se utilizan como una cola de instrucciones para poder llevar a cabo el pipeline de tres fases del procesador MIRAGE y que en la siguiente sección se explica más ampliamente.

Por otra parte se tiene un circuito secuenciador que direcciona a alguna localidad dentro de la memoria de microprograma, la cual es parte del procesador MIRAGE y no como la memoria de programa o la memoria de datos. Dicha memoria de microprograma se encarga de almacenar las microinstrucciones (palabras de control de 82 bits). Y juntos memoria de microprograma y secuenciador realizan la función de acceso de secuencias de microinstrucciones.

Los dos registros a la salida de la memoria de microprograma, sirven para crear una cola de microinstrucciones que también ayudará a la implementación del pipeline.

MUX4 se encarga de indicar la operación adecuada al secuenciador para así poder accesar una macroinstrucción o una simple microinstrucción. Si bit MACRO=0, entonces la operación que se indica al secuenciador es 24H que ordena que la dirección entrante por el bus A será la que salga por el secuenciador sin alteración; si bit MACRO=1, entonces la operación a ejecutar por el secuenciador se tomará de la microinstrucción.

2.3.- PIPELINE DEL PROCESADOR MIRAGE.

Como ya se mencionó existe una cola de instrucciones cuya función es el tener las partes necesarias de cada instrucción, para la correcta ejecución de la acción correspondiente a la fase EXECUTE y R/W del pipeline. Es decir, que en la cola de instrucciones no es necesario correr toda la palabra de instrucción de 32 bits sino sólo las partes necesarias para cada fase del pipeline.

Además de la cola de instrucciones se tiene una cola de microinstrucciones, la cual tiene una profundidad de 2 registros. Esta cola es necesaria para llevar a cabo el pipeline, ya que cada instrucción de la memoria de programa tiene en su byte más alto la dirección de una localidad de memoria de microprograma, en otras palabras, a cada instrucción de memoria de programa corresponde una microinstrucción en la memoria de microprograma.

Los registros DOR, DIR y MAR serán utilizados también para este pipeline, en las fases EXECUTE y R/W; ya que en la fase EXECUTE se realiza el cálculo de la dirección de una localidad de memoria de datos pero no es sino hasta la fase de R/W del pipeline que se hace uso de esta dirección calculada y del dato a ser escrito o leído de dicha memoria de datos y que para el caso de escritura de un dato se toma de DOR el cual retiene este dato del ciclo anterior.

Todo lo anteriormente descrito nos permite tener un pipeline de tres fases, ya que mientras una instrucción lee o escribe un dato en la memoria de datos (i), la siguiente instrucción es

ejecutada en la ALU (i+1) y la instrucción que sigue a esta última es traída de la memoria de programa (i+2). Este pipeline es mostrado en la siguiente figura:

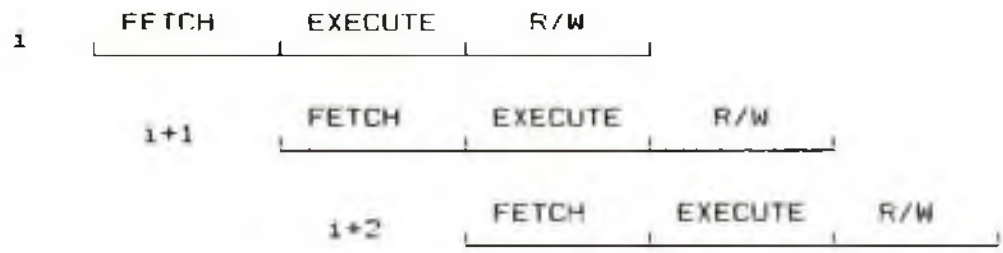


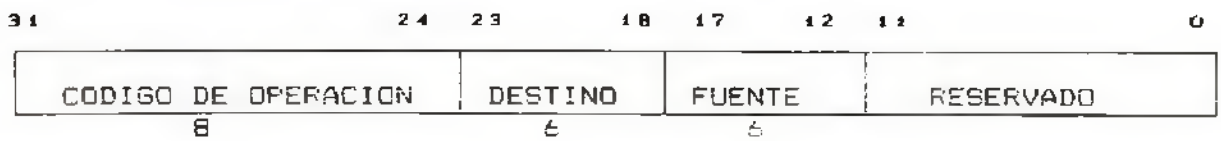
fig.2.3.- Esquema del pipeline de tres etapas del sistema.

2.4.- FORMATO DE INSTRUCCIONES.

Como ya se dijo cada instrucción de programa tiene un tamaño de 32 bits de longitud. Los 8 bits más significativos tienen el código de operación de dicha instrucción, que es la dirección de la microinstrucción que le corresponde. Los restantes 24 bits corresponden a los operandos como se muestra en el siguiente dibujo:

Sin embargo el campo de operandos varía según el tipo de instrucción. A continuación se describen tales campos para cada tipo de instrucción.

2.4.1.-REGISTRO A REGISTRO.



En este tipo se reservan 6 bits para direccionar el registro destino y 6 para el fuente. Según el tipo de operación esta clase de instrucción tiene 3 variantes:

a).Para operaciones binarias. En este tipo de operaciones se realiza una operación entre los dos registros y el resultado se escribe en el registro destino.

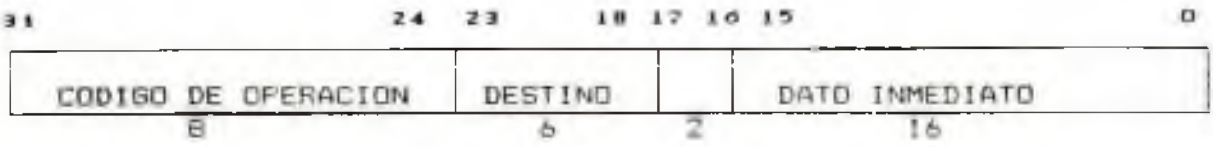
b).Para operaciones unarias que alteran el operando. En este caso la dirección del operando se coloca en la parte del destino, escribiéndose el resultado en ese registro.

c).Para operaciones unarias que no alteran el operando. En este caso la dirección del operando se coloca en la parte que corresponde al fuente y el resultado no se escribe en la dirección indicada en el campo del destino.

2.4.2.-DIRECTO A MEMORIA DE PROGRAMA.

A este tipo de instrucciones corresponden los saltos condicionados e incondicionales y el operando ocupa todos los 24 bits del campo de operando y corresponde a una dirección en la memoria de programa.

2.4.3.-INMEDIATO A REGISTRO.

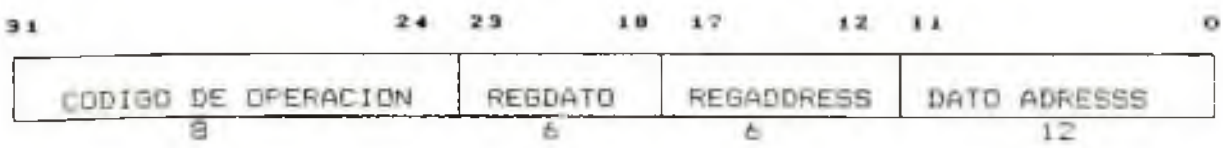


Este tipo de instrucción corresponde a cargar un registro en los bytes inferiores. En este caso se ocupan 6 bits para la dirección del registro destino y 16 bits para el dato inmediato, quedando 2 bits sin usarse entre ellos dos.

2.4.4.-DE MEMORIA DE DATOS A REGISTRO Y VICEVERSA.

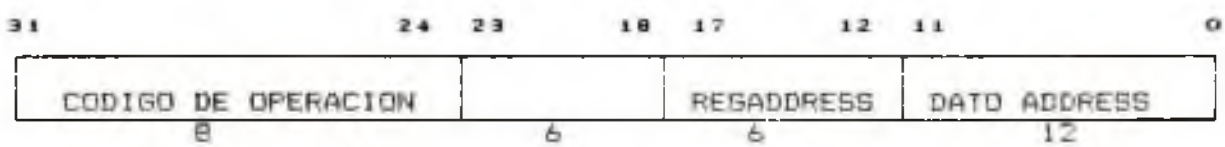
En este tipo de instrucciones se tiene dos variantes:

a).De memoria de datos a registro de CPU y viceversa.



En este caso se tienen 6 bits para la dirección del registro al cual se va a guardar un dato o del que se va a cargar un dato a memoria de datos, al cual aquí llamamos REGDATO, 6 bits para la dirección de un registro empleado para calcular la dirección del dato en la memoria de datos al que llamamos REGADDRESS y 12 bits para un dato inmediato que aquí hemos llamado DATO ADRESS y el cual es empleado para calcular la dirección en memoria de datos mediante una operación con el registro REGADDRESS en la ALU.

b).De memoria de datos a PC y viceversa.



Este tipo de instrucción tiene la misma estructura que el anterior solo que no se emplea el campo de REGDATO. Este tipo de instrucción sirve para la llamada y retorno de subrutinas ya que se puede implementar una pila en la memoria de datos mediante estas instrucciones.

3.- PROBLEMAS QUE PRESENTA LA ARQUITECTURA RISC Y LA MANERA EN QUE SE INTENTO RESOLVERLOS .

3.1.- PLANTEAMIENTO DE LOS PROBLEMAS.

El problema que presenta el tener un conjunto de instrucciones reducido, puede resultar no ser tan grave si se considera que recientes estudios, han demostrado que únicamente alrededor del 10% del total de instrucciones son usadas en un 90% de las veces. El resto de las instrucciones son usadas menos del 10% de las veces. Es por ello que los diseñadores de máquinas RISC han incluído sólo las instrucciones que son utilizadas más frecuentemente (datos, aritméticas, movimientos, lógicas). Como resultado de esta aclaración, se puede afirmar que los programas RISC son aproximadamente 25% más grandes que su contraparte en un CISC en lugar de 200% ó 300%.

Las instrucciones que no se encuentran en un sistema RISC pero que sí se pueden encontrar en un CISC, deben de ser remplazadas por varias con las que sí se cuenta. En otras palabras, algunas de las operaciones que toman únicamente una instrucción en un CISC, podrían llevarse dos o tres instrucciones para llevar a cabo dicha operación en un RISC. Esto haría parecer que los programas en un RISC serían dos o tres veces más largos que en un CISC, llevando a cabo la misma tarea. La programación en lenguaje ensamblador de un RISC implica un gran esfuerzo. Un programa muy largo también requeriría un gran espacio de memoria para almacenamiento, lo cual incrementaría el costo del sistema así como su volumen.

3.2.-ARQUITECTURA DEL PROCESADOR CLIPPER DE FAIRCHILD.

A continuación se presenta la solución que se planteo en el procesador Clipper diseñado por la División de Procesadores Avanzados de Fairchild, para resolver el problema antes mencionado, a través de la definición de macroinstrucciones.

El Clipper de Fairchild es un microprocesador de 32 bits de alto nivel de rendimiento, el cual combina aspectos de arquitectura de computadoras de conjunto reducido de instrucciones y de computadoras de conjunto de instrucciones complejas. Cuenta con características derivadas de supercomputadoras y mainframes, especialmente el diseño de cache de estilo mainframe.

Implementado en tres chips, la arquitectura Clipper incluye un chip CPU conteniendo una unidad de punto flotante y dos manejadores de memoria cache plus. Una CAMMU (Unidad de Manejo de Memoria Cache) para procesos de instrucciones y uno para procesos de datos. La arquitectura del Clipper integra todas las funciones computacionales en un chip y todas las funciones de acceso de memoria en dos CAMMUs. Los tres chips reemplazan arriba de 40 chips encontrados en diseños con otros microprocesadores. Estos chips están integrados en un módulo de 3 x 4.5 pulgadas que tiene una efectividad de ejecución aproximadamente igual a una VAX8600.

En la fig.3.2.1 se muestra una diagrama del Clipper. La CPU contiene una unidad de punto flotante y una unidad de pipeline de tres etapas. Cada CAMMU contiene una MMU (Unidad de Manejo de Memoria), un cache de 4K-byte y un control lógico de cache. Los cálculos de punto flotante pueden

ser procesados en paralelo con instrucciones de enteros y traslación de direcciones concurrentemente con un escudriño al cache. La arquitectura del Clipper está separada en tres arquitecturas, cada una definida por límites entre niveles del sistema y diseñadas con diferentes metas en mente. Las inovaciones internas y las arquitecturas del conjunto de instrucciones fueron diseñadas para mayor rapidez de ejecución de programa. Las aplicaciones de la arquitectura fueron diseñadas para compatibilidad con la base instalada de software. Tal compatibilidad no fue posible sino hasta recientemente que emergió el Sistema Unix V como un sistema operativo estándar y programas de aplicaciones que fueron escritos en lenguaje de alto nivel.

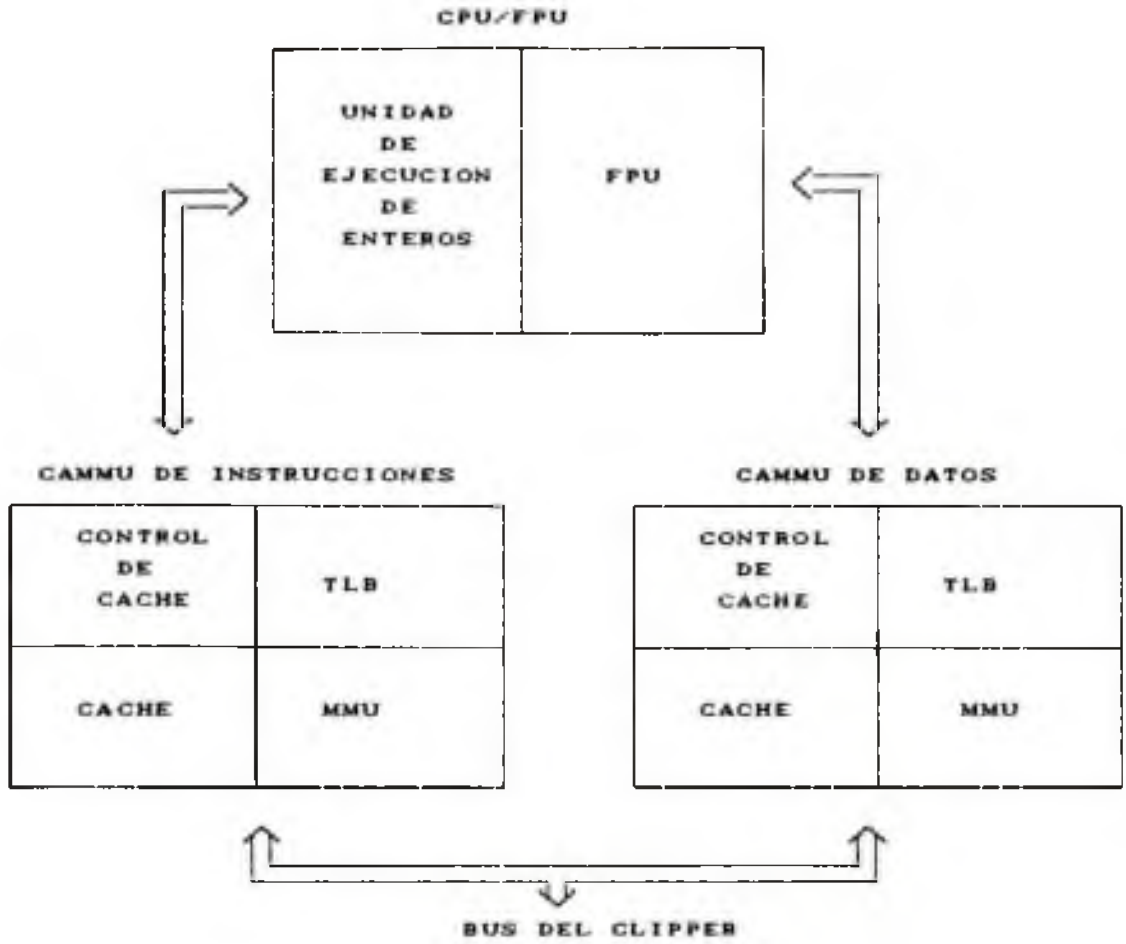
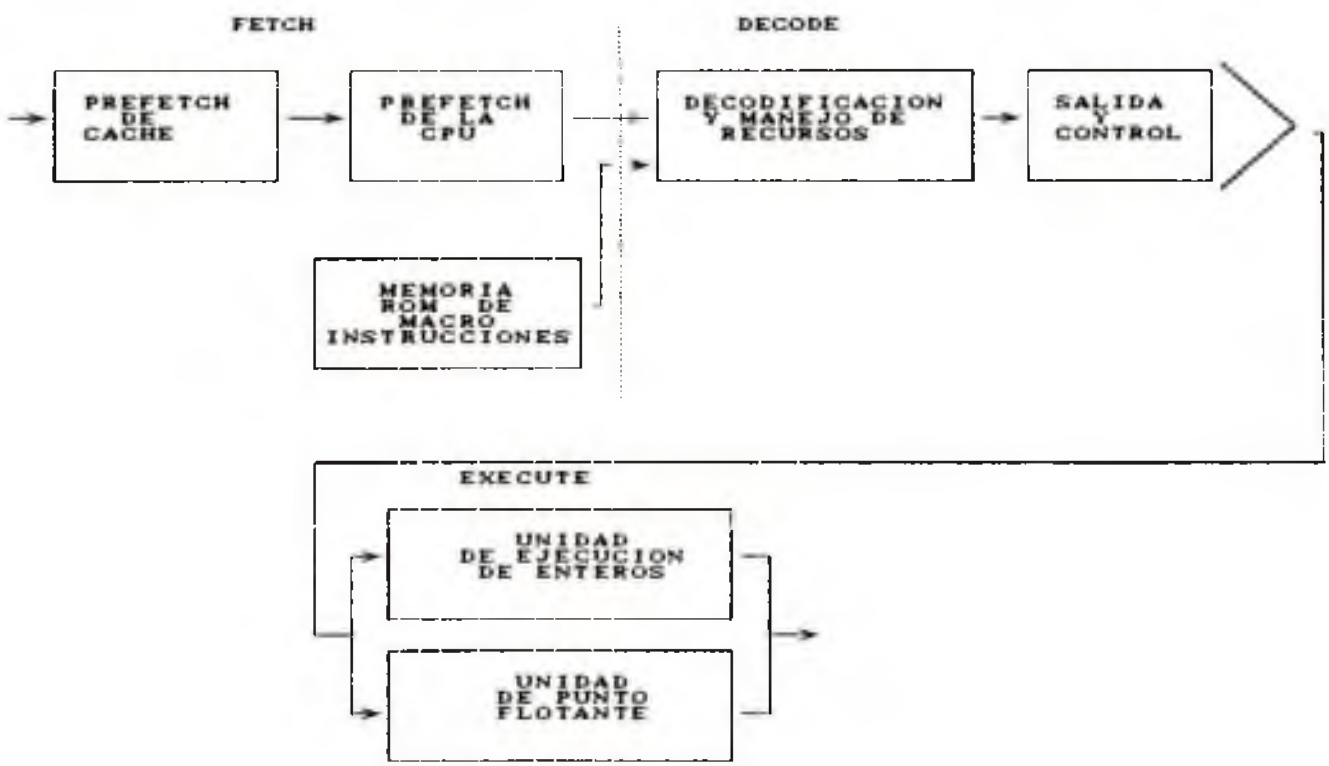


fig.3.2.1.- Diagrama a bloques del Clipper.

PIPELINE DEL CLIPPER.

El sistema de pipeline del Clipper combina los beneficios de un pipeline simple de tres fases con el uso extenso de concurrencia y paralelismo entre cada fase del pipe, especialmente en la fase de ejecución.

Pipeline de tres fases. La figura siguiente muestra las tres fases del pipeline del Clipper: fetch, decode y execute. Cada una de las fases procede independientemente sin tener que esperar a que la primera instrucción finalice todas la fases. Ya que las instrucciones aritméticas y lógicas del Clipper operan únicamente sobre registros, no hay necesidad de direcciones especiales de pipeline o cálculos efectivos de dirección de fases. Las instrucciones que requieren de cálculos de dirección efectiva simplemente hacen un paso a través de la ejecución en la ALU. Soló un paso es necesario porque estas instrucciones no requieren de operaciones aritméticas o lógicas de la ALU.



PIPELINE DEL CLIPPER.

Nótese que la figura muestra una subfase de prefetch de cache dentro de la fase de fetch. El prefetch ayuda a que la CPU mantenga lleno el buffer de instrucciones y mejore el rendimiento de cache.

Unidades de ejecución doble. La CPU contiene dos unidades de ejecución usadas en la última fase del pipeline. La unidad de ejecución de enteros maneja todas las operaciones aritméticas enteras y cálculos de direcciones y la unidad de ejecución de punto flotante que maneja todos los procesos de aritmética de punto flotante.

Las instrucciones de la unidad de macroinstrucciones hacen uso de sus propios arreglos de registros, así que ellas no hacen uso de los recursos de arreglos generales que están siendo ordinariamente usados por otras instrucciones.

MODELO DE PROGRAMACION

Un objetivo primario de diseño del Clipper fue proveer un conjunto de instrucciones por arquitectura que soportase bien a sistemas operativos y compiladores.

Las instrucciones deben ser ejecutadas rápidamente, preferentemente una por ciclo y éstas deben ser fáciles para escribir programas usando las instrucciones, especialmente para sistemas operativos y para generadores de código de compiladores.

La primera meta orienta el sistema a un conjunto muy simple de instrucciones, con pocos conceptos de alto nivel soportados en hardware. Este acercamiento es algunas veces llamado computadora de conjunto reducido de instrucciones, o filosofía RISC. La segunda meta lleva al diseñador en dirección totalmente opuesta, hacia un conjunto de instrucciones más rico el cual soporta conceptos de lenguajes de alto nivel y sistemas operativos. A ellos nos podemos referir como computadora de conjunto de instrucciones complejas, o filosofía CISC.

El conjunto de instrucciones del Clipper está en balance entre las características del RISC y CISC, un balance que también es característico de arquitecturas de supercomputadoras. En las siguientes líneas se mencionan algunas características del Clipper:

CARACTERISTICAS RISC Y CISC DEL CLIPPER.

RISC — Las instrucciones aritméticas y lógicas operan sobre registros; básicamente los accesos a memoria son cargas, almacenamientos y saltos, además de instrucciones de manipulación de la pila.

Existe una planta de registros: registros de 32 bits. 16 para la operación del sistema y 16 para el usuario.

CISC — Las instrucciones para acceso a memoria son provistas con un conjunto de modos de direccionamiento que facilita el acceso a estructuras de datos de alto nivel

Contiene un soporte explícito para pilas y cadenas.

Contiene modos de operación para el usuario y para el sistema operativo, cada uno teniendo sus propios recursos y privilegios.

INSTRUCCIONES

El conjunto de instrucciones del Clipper contiene 101 instrucciones alambradas y 67 macroinstrucciones para la operación de los tipos de datos básicos. Cada una de las instrucriones especifica una operación a ser ejecutada, y el tipo y localización de los operandos. Estos operandos pueden estar localizados en memoria, en un registro, o en la instrucción misma.

Los formatos de instrucciones son de dos tipos: los que tienen direcciones y los que no. Los formadores son las instrucciones que necesitan accesos a memoria; las más tardadas son las instrucciones aritméticas y lógicas aunque generalmente se ejecutan en un ciclo de reloj. Las instrucciones del Clipper tienen cero, uno o dos operandos, pero sólo uno puede ser accesado por una dirección de memoria.

Las 67 macroinstrucciones están implementadas en la unidad de macroinstrucciones, como secuencias de instrucciones alambradas. Excepto por su formato distintivo, nada distingue a

las macroinstrucciones de la instrucciones alambradas, para lo que concierne al usuario. Se puede mencionar que todas las instrucciones de manejo de caracteres son macros, así como muchas de las de manejo de pila (excepto PUSH y POP), y también una o dos de las instrucciones aritméticas, transferencia e instrucciones de control. Seis de estas macros son privilegiadas, estos es que sólo pueden ser accesadas por un programa en modo supervisor.

4.- SOLUCION PROPUESTA EN EL PROCESADOR MIRAGE.

4.1.- DESCRIPCION DE LAS MACROS EN EL PROCESADOR MIRAGE.

Para resolver el problema planteado en el capítulo 3 se propuso la implementación de macroinstrucciones que para este procesador son definidas como secuencias de microinstrucciones.

Con tal motivo se propuso el uso de un secuenciador el cual puede ser observado en el diagrama general a bloques del procesador. Este secuenciador trabaja directamente con la micromemoria y con el se pueden realizar saltos, ciclos y definir rutinas dentro de un espacio reservado en la micromemoria que será a partir de 100H hasta 1FFH. Para el manejo de parámetros de dichas macroinstrucciones se han reservado 8 registros dentro del arreglo de registros y que son los 8 últimos, cabe mencionar que estos registros pueden ser utilizados de manera general pero bajo responsabilidad del usuario.

Como más adelante se verá, existe un bit dentro de la microinstrucción que nos indica si dicha microinstrucción pertenece a una macro, si es fin de la macro o que también puede ser una instrucción normal. La macro será llevada a cabo totalmente en la fase de ejecución. También cabe hacer notar que la instrucción en la memoria de programa que llama a la macro en cuestión, sólo sirve para contener la dirección de inicio en micromemoria de la secuencia de microinstrucciones que forman a la macro.

El bit de control para las macroinstrucciones llamado MACRO también hará la deshabilitación del contador de programa (PC) para que no se incremente y ya no direccione nuevas localidades de memoria de programa, pues la ejecución de la macroinstrucción se llevara a cabo únicamente en la fase EXECUTE del pipeline, esto no significa que la macroinstrucción se llevará a cabo en un solo ciclo de reloj, sino que simplemente no se traerán nuevas instrurciones hasta que no se termine la ejecución de la macroinstrucción (fig.4.1).

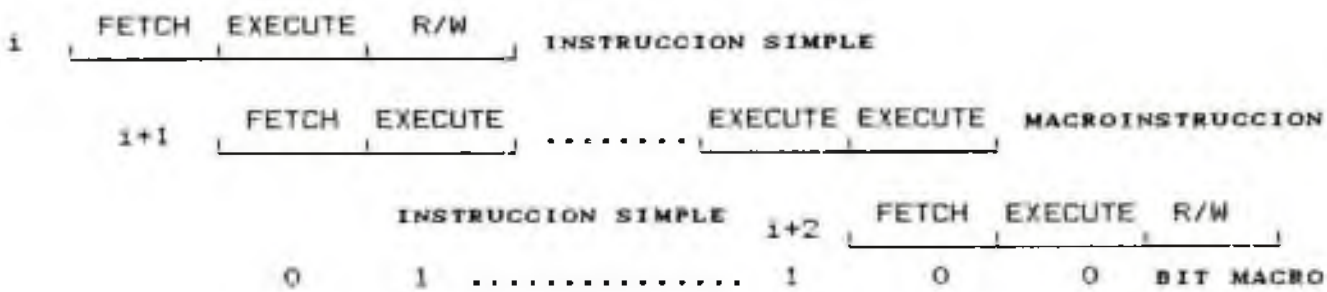
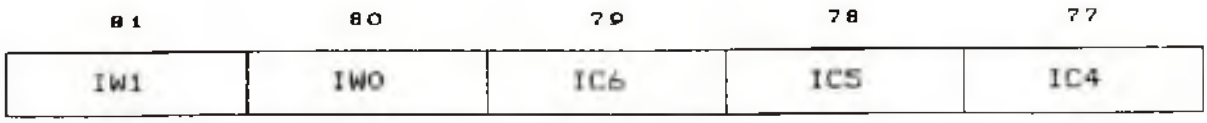
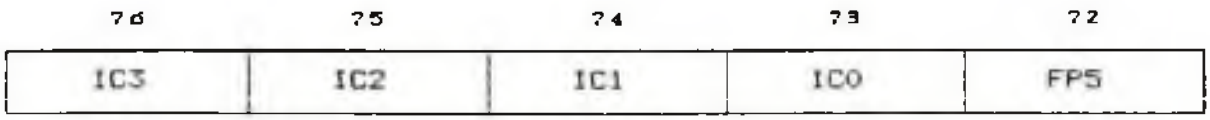


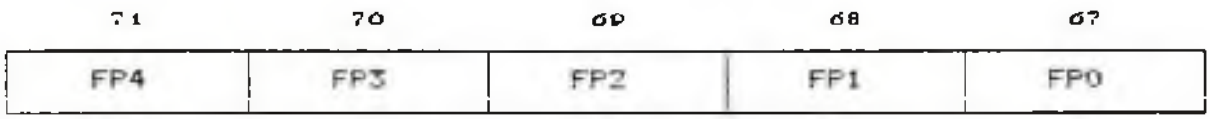
fig.4.1.- Comportamiento del pipeline cuando se lleva a cabo una macroinstrucción.

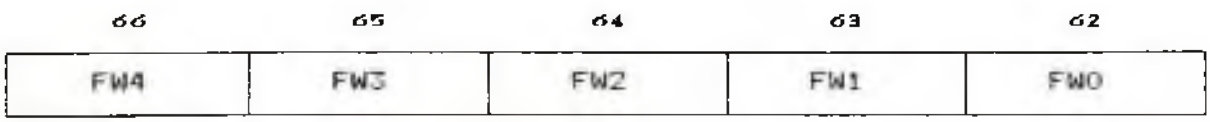
4.2.-DESCRIPCION DE LAS LINEAS DE CONTROL.

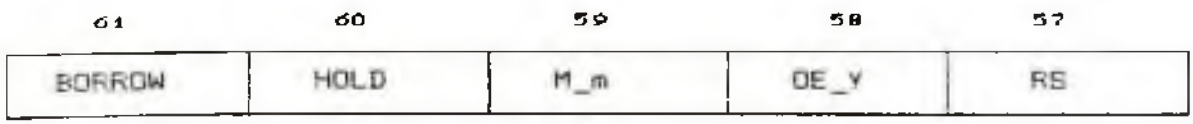
Como ya se mencionó cada microinstrucción tiene una longitud de 82 bits cada uno de los cuales corresponde a una señal de control para el sistema. La estructura para todas las microinstrucciones es la misma y es la que se muestra en los siguientes dibujos.

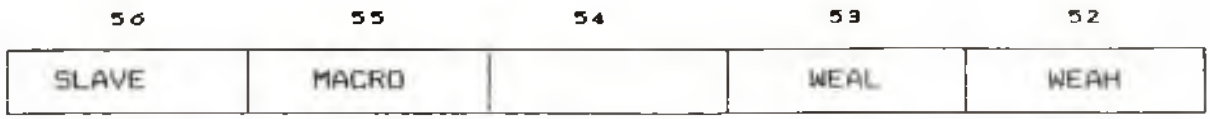

81
80
79
78
77
IW1
IW0
IC6
IC5
IC4

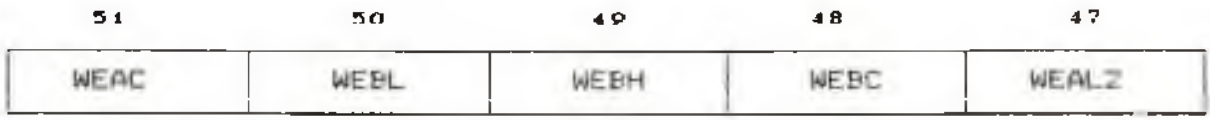

76
75
74
73
72
IC3
IC2
IC1
IC0
FP5

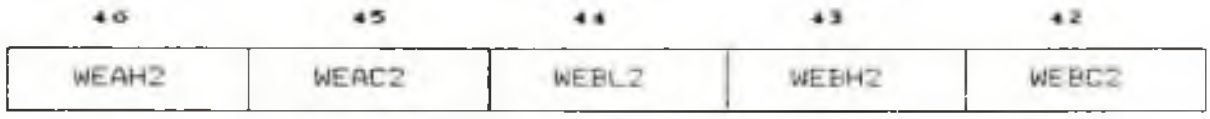

71
70
69
68
67
FP4
FP3
FP2
FP1
FP0

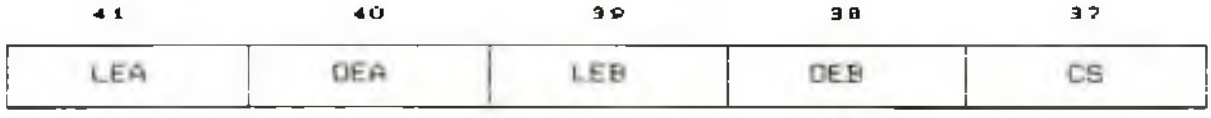

66
65
64
63
62
FW4
FW3
FW2
FW1
FW0

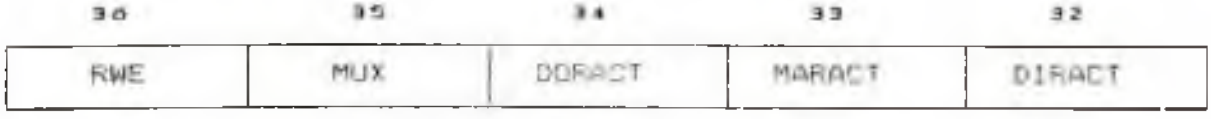

61
60
59
58
57
BORROW
HOLD
M_m
OE_Y
RS

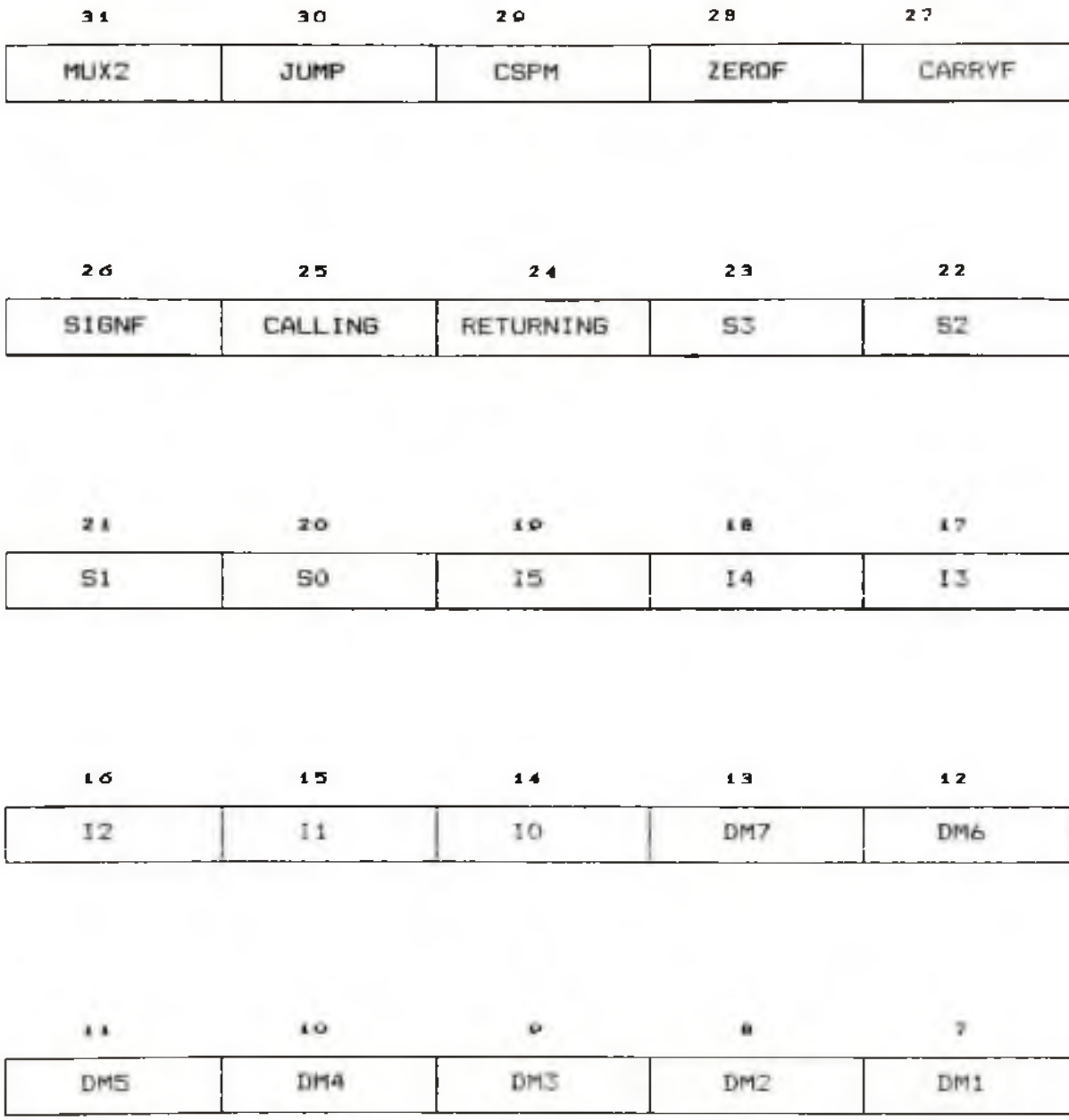

56
55
54
53
52
SLAVE
MACRO
WEAL
WEAH

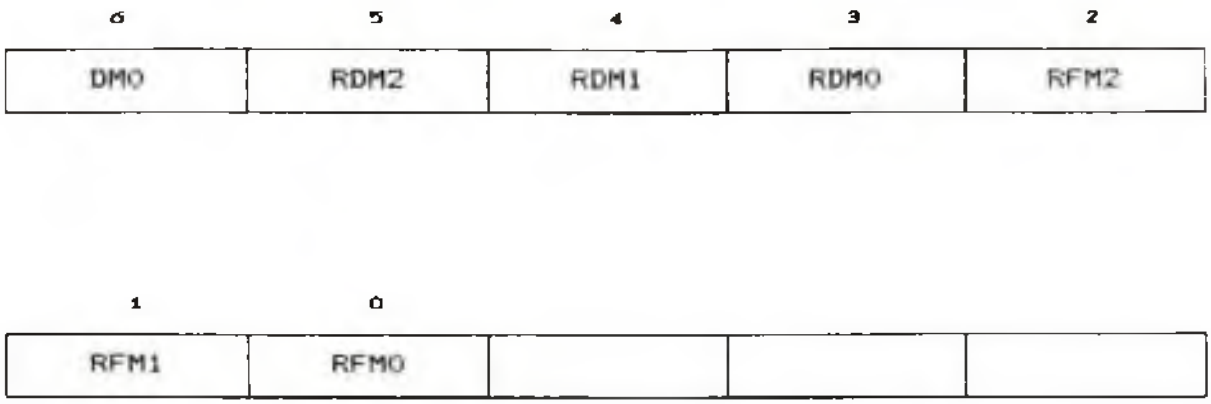

51
50
49
48
47
WEAC
WEBL
WEBH
WEBC
WEAL2


46
45
44
43
42
WEAH2
WEAC2
WEBL2
WEBH2
WEBC2


41
40
39
38
37
LEA
OEA
LEB
OEB
CS


36
35
34
33
32
RWE
MUX
DDRACT
MARACT
DIRACT


31
30
29
28
27
MUX2
JUMP
CSPM
ZERDF
CARRYF
26
25
24
23
22
SIGNF
CALLING
RETURNING
S3
S2
21
20
19
18
17
S1
S0
I5
I4
I3
16
15
14
13
12
I2
I1
I0
DM7
DM6
11
10
9
8
7
DM5
DM4
DM3
DM2
DM1



A continuación damos el significado de cada uno de los campos de la microinstrucción los cuales como recordamos corresponden a señales de control.

IW.-Ancho de bytes para el formato de operandos de límite alineado, y para el formato de campo de longitud variable decide de donde se tomarán el ancho y la posición (registro de estado o terminales).

IC.-Código de intrucción para la ALU.

FP.-Campo de posición para el formato de longitud variable de bits.

FW.-Campo de ancho para el formato de longitud variable de bits.

BORROW.-Cuando está en alto se invierte el carry de entrada y el de salida (ALU).

HOLD.-En alto inhibe la actualización del registro de estado y el registro Q de la ALU.

M_m.-En alto selecciona las terminales macro carry y macro link en bajo toma el carry y link de entrada del registro de estado (ALU).

OE_Y.-En alto el bus Y es deshabilitado (ALU).

RS.-En alto selecciona el status de la ALU y en bajo el registro de estado.

SLAVE.-En alto la ALU se pone en modo slave.

MACRO.-En alto indica que la microinstrucción pertenece a parte de una secuencia y también el inicio de la misma, y en bajo nos indica el final de una secuencia o que se trata de una microinstrucción simple.

$\overline{\text{WEAL}}$.-Habilitar la escritura en el byte menos significativo del registro direccionado para escritura en el arreglo de registros, si $\overline{\text{WEAC}}$ está habilitado, en el puerto A de entrada.

$\overline{\text{WEAH}}$.-Habilitar la escritura en el byte segundo menos significativo del registro direccionado para escritura en el arreglo de registros, si $\overline{\text{WEAC}}$ está habilitado, en el puerto A de entrada.

$\overline{\text{WEAC}}$.-Habilitar la escritura en los dos bytes menos significativos del registro direccionado para escritura en el arreglo de registros, en el puerto A de entrada.

$\overline{WEBL}$.-Habilitar la escritura en el byte menos significativo del registro direccionado para escritura en el arreglo de registros, si $\overline{WEBC}$ está habilitado, en el puerto B de entrada.

$\overline{WEBH}$.-Habilitar la escritura en el segundo byte menos significativo del registro direccionado para escritura en el arreglo de registros, si $\overline{WEBC}$ esta habilitado, en el puerto B de entrada.

$\overline{WEBC}$.-Habilitar la escritura en los dos bytes menos significativos del registro direccionado para escritura en el arreglo de registros, en el puerto B de entrada.

$\overline{WEAL2}$.-Habilitar la escritura en el tercer byte menos significativo del registro direccionado para escritura en el arreglo de registros, si $\overline{WEAC2}$ está habilitado, en el puerto A de entrada.

$\overline{WEAH2}$.-Habilitar la escritura en el byte más significativo del registro direccionado para escritura en el arreglo de registros, si $\overline{WEAC2}$ esta habilitado, en el puerto A de entrada.

$\overline{WEAC2}$.-Habilitar la escritura en los dos bytes más significativos del registro direccionado para escritura en el arreglo de registros, en el puerto A de entrada.

$\overline{WEBL2}$.-Habilitar la escritura en el tercer byte menos significativo del registro direccionado para escritura en el arreglo de registros, si $\overline{WEBC2}$ esta habilitado, en el puerto B.

$\overline{WEBH2}$.-Habilitar la escritura en el byte más significativo del registro direccionado para escritura en el arreglo de registros, si $\overline{WEBC2}$ esta habilitado, en el puerto B de entrada.

$\overline{WEBC2}$.-Habilitar la escritura en los dos bytes más significativos del registro direccionado para escritura en el arreglo de registros, en el puerto B de entrada.

LEA.-Mantener un dato en el puerto A de salida del arreglo de registros.

$\overline{OEA}$.-Habilitar salida en el puerto A de salida del arreglo de registro.

LEB.-Mantener un dato en el puerto B de salida en el arreglo de registros.

$\overline{OEB}$.-Habilitar salida del arreglo de registros en su puerto B de salida.

$\overline{CS}$.-Habilitar a la memoria de datos.

RWE.-Habilitar escritura (0) o lectura (1)en la memoria de datos.

MUX.-Habilitar a un dato inmediato (0) o al dato en el puerto A de salida del arreglo de registros (1) como entrada al puerto A de entrada de la ALU.

DORACT.-Activar al registro DOR .

MARACT.-Activar al registro MAR.

DIRACT. Activar al registro DIR.

MUX2.-Habilitar a un dato inmediato (0) o al dato en el puerto Y de la ALU (1) como entrada al puerto A del arreglo de registros.

$\overline{\text{JUMP}}$.-Habilitar a PC para cargarse con dato inmediato.

CSPM.-En alto habilita la lectura de la memoria de programa.

$\overline{\text{ZEROF}}$.-Habilitar a PC para cargarse con dato inmediato si la bandera de cero está en 1.

$\overline{\text{CARRYF}}$.-Habilitar a PC para cargarse con dato inmediato si la bandera de carry está en 1.

$\overline{\text{SIGNF}}$.-Habilitar a PC para cargarse con dato inmediato si la bandera de signo está en 1.

$\overline{\text{CALLING}}$.-Almacenar a PC en la memoria de datos.

$\overline{\text{RETURNNIG}}$.-Habilitar a PC para cargarse con un dato proveniente de la memoria de datos.

Sn.-Se refiere a que condición de prueba se usará.

In.-Es el código de la instrucción que deberá ejecutar el secuenciador.

DMn.-Es la dirección en micromemoria de la macro definida con anterioridad.

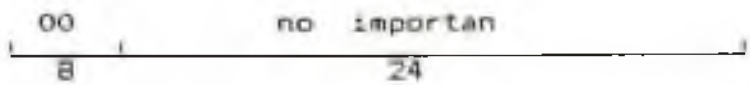
RDMn.-El registro destino usado dentro de una macro que como se dijo anteriormente es uno de los ocho registros finales dentro del arreglo de registros.

RFMn.-El registro fuente dentro de una macro.

4.3.-JUEGO DE INSTRUCCIONES Y MICROINSTRUCCIONES DISEÑADAS.

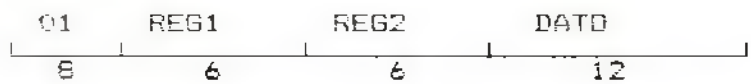
En seguida se decribe el conjunto de instrucciones diseñadas y sus correspondientes microinstrucciones, todas las cuales se dan en su código hexadecimal.

NO OPERACION.- Como su nombre lo dice no realiza ninguna operación.



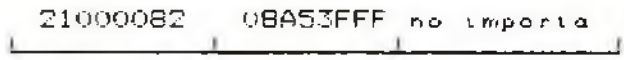
La microinstrucción que le corresponde está en la localidad cero de la micromemoria y es como sigue:

CARGA A REGISTRO UN DATO DE MEMORIA.- Esta operación tiene el siguiente formato:

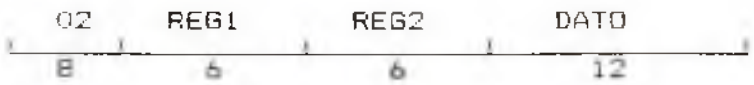


Lo que hace esta instrucción es cargar el dato en la memoria de datos cuya dirección se calcula con la suma del registro cuya dirección es "REG2" y el dato inmediato "DATO" en el registro cuya dirección es "REG1".

La microinstrucción que le corresponde está en la localidad 1 de la memoria de microprograma y es la siguiente:

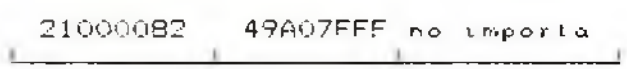


ALMACENAR UN REGISTRO EN MEMORIA.- Esta operación tiene el siguiente formato:

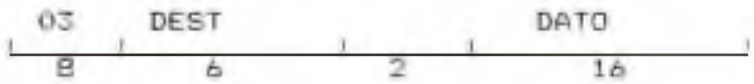


Lo que hace esta instrucción es guardar el contenido del registro cuya dirección es "REG1" en la localidad de datos cuya dirección se calcula con la suma del registro cuya dirección es "REG2" y el dato inmediato "DATO".

La microinstrucción que le corresponde está en la localidad 2 de la memoria de microprograma y es la siguiente:

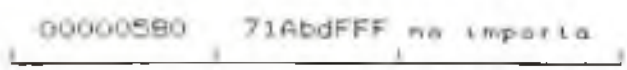


CARGA INMEDIATA A REGISTRO EN BYTES INFERIORES.- El formato de esta instrucción es el siguiente:

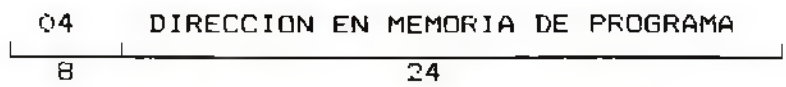


Esta instrucción carga en los dos bytes menos significativos del registro cuya dirección es "DEST" el valor inmediato "DATO".

La microinstrucción que le corresponde está en la localidad 3 y es la siguiente:

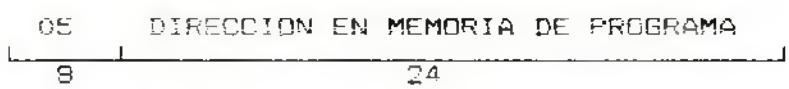


SALTO INCONDICIONAL.- Esta instrucción realiza un salto incondicional a la dirección de memoria indicada en el campo de operando. Antes de realizarse el salto se realiza la instrucción que sigue. Su estructura es la siguiente:

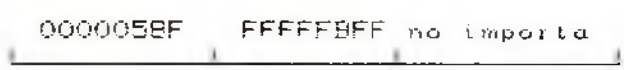


La microinstrucción que le corresponde está en la dirección 4 y es la que sigue:

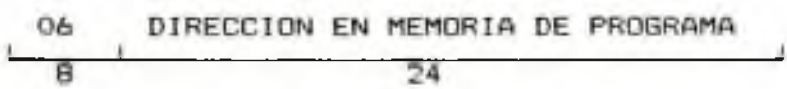
SALTO CONDICIONAL SEGUN BANDERA CERO.- Esta instrucción realiza un salto si la bandera de cero está en 1, a la dirección de memoria indicada en el campo de operando. Antes de realizarse el salto se realiza la instrucción que sigue. Su estructura es la siguiente:



La microinstrucción que le corresponde está en la dirección 5 y es la que sigue:

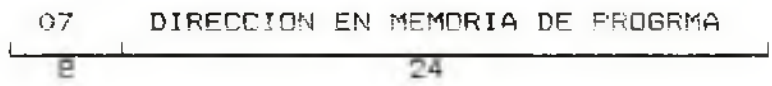


SALTO CONDICIONAL SEGUN BANDERA CARRY.- Esta instrucción realiza un salto si la bandera de carry está en 1, a la dirección de memoria indicada en el campo de operando. Antes de realizarse el salto se realiza la instrucción que sigue. Su estructura es la siguiente:



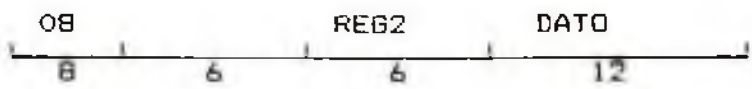
La microinstrucción que le corresponde está en la dirección 6 y es:

SALTO CONDICIONAL SEGUN BANDERA SIGNO.- Esta instrucción realiza un salto si la bandera de signo esta en 1, a la dirección de memoria indicada en el campo de operando. Antes de realizarse el salto se realiza la intrucción que sigue. Su estructura es la siguiente:

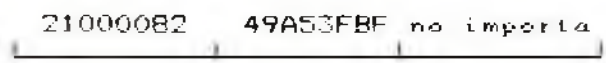


La microinstrucción que le corresponde está en la dirección 7 y es:

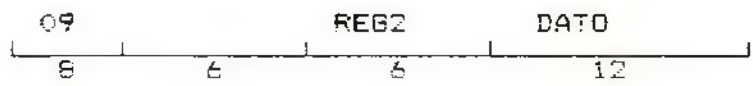
CARGAR PC CON UN DATO DE MEMORIA.- Esta instrucción tiene el siguiente formato:



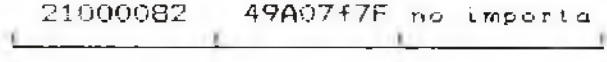
Lo que hace es cargar al registro PC con el dato en la memoria de datos cuya dirección se calcula sumando el contenido del registro en el arreglo de registros cuya dirección es "REG2" y el valor inmediato "DATO". La microinstrucción correspondiente está en la localidad 8 y es la que sigue:



GUARDAR PC EN MEMORIA.- Esta instrucción tiene el siguiente formato:

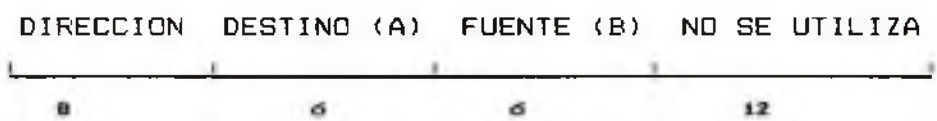


Esta instrucción se encarga de guardar el registro PC en la localidad de la memoria de datos cuya dirección se calcula sumando el contenido del registro en el arreglo de registros cuya dirección es "REG" y el valor inmediato "DATO". La microinstrucción correspondiente está en la localidad 9 y es la que sigue:



OPERACIONES ENTRE REGISTROS.

A continuación se muestra el formato:



En base a la extensa cantidad de instrucciones que puede ejecutar el ALU AM29332, se diseñaron algunas micro-instrucciones las cuales se han clasificado en tres tipos y que han sido realizadas en el formato de operandos de límite alineado y con un ancho de 4 bytes:

- unarias
  - que afectarán al operando de entrada, TABLA 1.
  - que no afectan al operando de entrada, TABLA 2.
- binarias, las cuales afectan a su operando A, TABLA 3.

En las tablas 1, 2 y 3 se muestra una descripción de las instrucciones.

NOTA:Las tablas sólo hacen mención a las microinstrucciones que no son parte de una secuencia y no se incluyó los primeros 18 bits de la parte baja pues no importa su contenido para este caso. Para las microinstrucciones que son parte de una macro, los primeros 18 bits de la parte baja son variables y el bit llamado MACRO debe estar en alto.

TABLA 1

MNEMONICOS	MICRO - CODIGO 64 BITS	DESCRIPCION	SALIDA Y		ESTATUS							DIR. DE MEM
			NO-SEL	SEL	S	M	L	Z	V	N	C	
NOT-A	04000080 41ABFFFF	COMPLEMENTO A UNO	A	$\overline{A}$				*				15
NEG-A	05000080 41ABFFFF	COMPLEMENTO A DOS	A	$\overline{A}+1$				*	*	*	*	16
INCR-A	09000080 41ABFFFF	INCREMENTO POR UNO	A	$A+1$ (1)				*	*	*	*	17
DECR-A	08000080 41ABFFFF	DECREMENTO POR UNO	A	$A-1$ (1)				*	*	*	*	18
SUM-CORR-A	24000080 41ABFFFF	CORRECCION SUMA BCD	A	A CORREGIDO				*	*	*	*	19
DIFF-CORR-A	25000080 41ABFFFF	CORRECCIOM RESTA BCD	A	A CORREGIDO				*	*	*	*	1A
LDSTAT-A	0E000080 41ABFFFF	CARGAR EL REGISTRO DE ESTADO	S	A	+	+	+	+	+	+	+	1B
DN1-0F-A	10000080 41ABFFFF	CORR. DESCENDENTE CON LLENADO A 0	A	$Y_i = A_{i+1}$ $Ymsb=0$			*	*		*		1C
DN1-1F-A	12000080 41ABFFFF	CORR. DESCENDENTE CON LLENADO A 1	A	$Y_i = A_{i+1}$ $Ymsb=1$			*	*		*		1D
UP1-0F-A	18000080 41ABFFFF	CORR. ASCENDENTE CON LLENADO A 0	A	$Y_i = A_{i-1}$ $Ymsb=0$			*	*		*		1E
UP1-1F-A	1A000080 41ABFFFF	CORR. ASCENDENTE CON LLENADO A 1	A	$Y_i = A_{i-1}$ $Ymsb=1$			*	*		*		1F
DN1-0F-AQ	11000080 41ABFFFF	CORR. DESCENDENTE CON LLENADO A 0		$0 \gg A \gg Q$			*	*		*		20
DN1-1F AQ	13000080 41ABFFFF	CORR. DESCENDENTE CON LLENADO A 1		$1 \gg A \gg Q$			*	*		*		21
UP1-0F-AQ	19000080 41ABFFFF	CORR. ASCENDENTE CON LLENADO A 0		$A \ll Q \ll 0$			*	*		*		22
UP1-1F-AQ	1B000080 41ABFFFF	CORR. ASCENDENTE CON LLENADO A 1		$A \ll Q \ll 1$			*	*		*		23

NOTA (1) Los bits de acarreo entre nibbles son actualizados en estas instrucciones.

* Se actualiza la bandera.

\+ Se actualiza sólo si el ancho de bytes es 3 o 4.

TABAL 2

MNEMONICOS	MICRO - CODIGO 64 BITS	DESCRIPCION	SALIDA Y		ESTATUS							DIR DE MEM
			NO-SEL	SEL	S	M	L	Z	V	N	C	
NOT-B	04800080 41ABFFFF	COMPLEMENTO A UNO	B	$\overline{B}$				*				24
NEG-B	05800080 41ABFFFF	COMPLEMENTO A DOS	B	$\overline{B}+1$				*	*	*	*	25
INCR-B	09800080 41ABFFFF	INCREMENTO POR UNO	B	$B+1$ (1)				*	*	*	*	26
DECR-B	08800080 41ABFFFF	DECREMENTO POR UNO	B	$B-1$ (1)				*	*	*	*	27
SUM-CORR-B	24800080 41ABFFFF	CORRECCION SUMA BCD	B	B CORREGIDO				*	*	*	*	28
DIFF-CORR B	25800080 41ABFFFF	CORRECCIOM RESTA BCD	B	B CORREGIDO				*	*	*	*	29
LDSTAT-B	0E800080 41ABFFFF	CARGAR EL REGISTRO DE ESTADO	S	B	+	+	+	+	+	+	+	2A
DN1-0F-B	10800080 41ABFFFF	CORR. DES-CENDENTE CON LLENADO A 0	A	$Y_i = B_{i+1}$ $Ymsb=0$			*	*		*		2B
DN1-1F-B	12800080 41ABFFFF	CORR. DES-CENDENTE CON LLENADO A 1	A	$Y_i = B_{i+1}$ $Ymsb=1$			*	*		*		2C
UP1-0F-B	18800080 41ABFFFF	CORR. AS-CENDENTE CON LLENADO A 0	A	$Y_i = B_{i-1}$ $Ymsb=0$			*	*		*		2D
UP1-1F-B	1A800080 41ABFFFF	CORR. AS-CENDENTE CON LLENADO A 1	A	$Y_i = B_{i-1}$ $Ymsb=1$			*	*		*		2E
DN1-0F-BQ	11800080 41ABFFFF	CORR. DES-CENDENTE CON LLENADO A 0		$0 \gg B \gg Q$			*	*		*		2F
DN1-1F-BQ	13800080 41ABFFFF	CORR. DES-CENDENTE CON LLENADO A 1		$1 \gg B \gg Q$			*	*		*		30
UP1-0F-BQ	19800080 41ABFFFF	CORR. AS-CENDENTE CON LLENADO A 0		$B \ll Q \ll 0$			*	*		*		31
UP1-1F-BQ	1B800080 41ABFFFF	CORR. AS-CENDENTE CON LLENADO A 1		$B \ll Q \ll 1$			*	*		*		32

NOTA (1) Los bits de acarreo entre nibbles son actualizados en estas instrucciones.

* Se actualiza la bandera.

\+ Se actualiza sólo si el ancho de bytes es 3 o 4.

TABLA 3

MNEMONICOS	MICRO - CODIGO 64 BITS	DESCRIPCION	SALIDA Y		ESTATUS							DIR. DE MEM.
			NO-SEL	SEL	S	M	L	Z	V	N	C	
ZERO	1E000080 41ABFFFF	PUESTA A CERO	B	0				1		0		33
OR	1F000080 41ABFFFF	OR	B	A OR B				*		*		34
XOR	1F800080 41ABFFFF	EXOR	B	A XOR B				*		*		35
AND	20000080 41ABFFFF	AND	B	A AND B				*		*		36
XNOR	20800080 41ABFFFF	XNOR	B	A XNOR B				*		*		37
ADD	21000080 41ABFFFF	SUMA	B	A + B				*	*	*	*	38
ADDC	21800080 41ABFFFF	SUMA CON CARRY	B	A + B + C				*	*	*	*	39
SUB	22000080 41ABFFFF	RESTA	B	A + $\bar{B}$ + 1				*	*	*	*	3A
SUBC	22800080 41ABFFFF	RESTA CON BORROW	B	A+$\bar{B}$+1+C				*	*	*	*	3B

NOTA: Las últimas 4 microinstrucciones también afectan a las banderas NC0-NC7 de la ALU.

4.4.- EJEMPLO DE UNA MACROINSTRUCCION.

La siguiente macroinstrucción es para realizar una suma BCD.

Lo primero es definir una instrucción en memoria de programa que sea la llamada a esta macroinstrucción suma BCD. La cual puede ser:

0b00 0000H

en esta instrucción se puede apreciar que lo único que se indica es la dirección de la microinstrucción la cual para este ejemplo se encuentra en la localidad 00bH de la memoria de microprograma. En esta localidad se encuentra una microinstrucción que tiene como función el hacer un salto a las localidades de memoria reservadas para definiciones de secuencias de palabras de control que conforman la macroinstrucción. En este ejemplo tal microinstrucción es:

0000 05af ffff fffe 0000 0H

La cual indica la ejecución de una no operación y el salto a la localidad 100H de la memoria de microprograma y además anuncia el comienzo de la macroinstrucción (con el bit MACRO=1). En dicha localidad se encuentra la microinstrucción que efectuará la suma binaria que se nencesita como primer paso a la suma BCD, dicha microinstrucción también tendrá al bit MACRO=1 y trabajará con los registros reservados para las macroinstrucciones, que para este ejemplo el dato A usara al registro 3 reservado para macros (R59 del arreglo de registros) que será donde se almacene el resultado y para el dato B se usa el registro 4 reservado para

macros (R60 del arreglo de registros), esto supone que antes de la llamada a la macroinstrucción se ha hecho la carga de los datos en los registros. La microinstrucción que corresponde a esta suma binaria es la siguiente:

2100 00a0 41ab ffff 0001 0H

Cabe hacer notar que esta microinstrucción debe también indicar al secuenciador que debe realizar un CONTINUE (11). Por tanto la siguiente microinstrucción a ejecutar será la localizada en la dirección 101H de la memoria de microprograma, esta instrucción es una corrección de suma BCD (10), la cual tiene como operando el dato A es decir al registro 3 (R59), esta microinstrucción tiene al bit MACRO=0 e indica que la macro ha finalizado. La microinstrucción es la siguiente:

2400 0080 41ab ffff 0007 0H

5.-DESCRIPCION DEL MODELO COMPUTACIONAL DEL PROCESADOR MIRAGE

5.1.- DESCRIPCION GENERAL DE LAS PANTALLAS.

En la fig.5.1 se puede observar que existe una ventana que es la más grande y en su parte superior derecha tiene la leyenda REG-ARRAY, esta ventana corresponde al arreglo de registros. La ventana tiene en su parte superior dos salientes, que son los dos puertos de entrada A y B de izquierda a derecha respectivamente; igual en la parte inferior de esta ventana de registros hay dos salientes que son los dos puertos de salida A y B de izquierda a derecha respectivamente. Dentro de la ventana existen los 64 registros de que se compone el arreglo de registros y para determinar su posición se utilizaron coordenadas de base octal, dichas coordenadas están en el primer renglón y columna, de tal manera que los registros van creciendo en orden de izquierda a derecha y de arriba hacia abajo.

Bajo la ventana de registros existe una ventana más pequeña que corresponde al ALU la cual en su parte superior muestra los datos de entrada A y B. En la parte inferior de la ventana está un dato que es precedido por una "S" que nos muestra como han sido afectadas las banderas de la ALU que se encuentran en el registro "S" de dicha ALU (10) y a la derecha se encuentra otro dato con un prefijo "Q" que indica como ha sido afectado el registro "Q" de la ALU (10). En la parte inferior a esta ventana de la ALU se encuentran dos salientes, la de la izquierda con etiqueta "S" es la palabra de estado de banderas de la ALU y del lado derecho se encuentra la saliente con etiqueta "Y" que corresponde al resultado de la operación efectuada por la ALU.

Al lado derecho de la ventana correspondiente al ALU, se encuentra una ventana rectangular que muestra los datos de los registros MAR, DIR y DOR que corresponden a los registros MAR, DIR y DOR utilizados como interface con la memoria de datos.

La ventana al lado izquierdo de la ALU nos muestra información de la instrucción traída en la fase FETCH del pipeline. En primer lugar se tiene a "IA" que es la dirección de la instrucción , esta instrucción es expresada en base hexadecimal. Después sigue "IR" que corresponde al código de operación de la acción a ejecutar en la ALU (10) y que es expresada en base binaria. El siguiente renglón corresponde al código de ancho de los datos a manejar en la ALU (10) para el formato de operandos de bytes de límite alineado, este dato es expresado en base binaria. "FP" corresponde a la posición y "FW" al ancho del operando para el formato de operandos de campo variable de la ALU (10) y también son expresados en base binaria.

NOTA: Todos los valores que son mostrados en el simulador están expresados en base hexadecimal excepto para los que se indique lo contrario.

5.2.- COMANDOS DEL MODELO COMPUTACIONAL.

El la pantalla del simulador del procesador MIRAGE puede apreciarse una línea de ayuda en la parte inferior con los comandos principales para manejar el simulador. Oprimiendo la primera letra (indicada con otro color o tono de gris para el caso de monitor monocromático) de cada comando, éste se ejecutará. Ahora se procederá a describir cada uno de los comandos en forma separada.

Comando "Memory".- Este comando se ejecutará al oprimir la tecla **M** y sirve para ver y modificar el contenido de la memoria de programa. Cuando se ejecuta este comando aparece la ventana de memoria fig. 5.2 mostrandonos las primeras 10 localidades. Como puede verse existe una flecha apuntando a la primera localidad. Para desplazarnos a lo largo de la memoria basta con oprimir la tecla de flecha hacia abajo (cursor) y nos moveremos hacia las localidades más altas, oprimiendo la tecla de flecha hacia arriba nos moveremos a las localidades menores y con la teclas PgUp y PgDn se avanza o se regresa por pagina. La ventana cuenta con un scrolling para poder movernos a lo largo de toda la memoria. Si se desea modificar una localidad basta con movernos y posicionarnos sobre la localidad que deseamos modificar, oprimir la tecla **m** (modificar), entonces desaparere la flecha y aparece el cursor justo arriba del primer nibble de la localidad a modificar, en este momento podemos comenzar a modificar tecleando los nuevos valores que queremos en forma hexadecimal. En caso de teclear algun caracter que no sea hexadecimal el programa no lo toma en cuenta para así protegerse contra errores. Para terminar la modificación es suficiente con oprimir la tecla de retorno o automáticamente se termina al modificar el último nibble, apareciendo nuevamente la flecha sobre esta localidad indicando que de nuevo podemos movernos en la memoria. Para salir de la memoria simplemente oprimimos la tecla ESC.

Comando "Umemory".- Este comando se ejecuta al oprimir la tecla **U** y nos sirve para ver y modificar la micromemoria (o memoria de microprograma) fig. 5.3 y su uso es similar al de la memoria de programa, la diferencia es que cada localidad es más grande, por lo que se ha dividido en tres partes separadas por un espacio (estas tres partes pueden soportar más de 82 bits para posibles ampliaciones de la palabra de control, pero para la actual palabra de control los bits sobrantes no son tomados en

cuenta por el simulador) para apreciarlas mejor. Cuando se pasa de una de las partes a otra al estar modificando automáticamente se salta el espacio entre partes.

Comando "Data".- Este comando se ejecuta al oprimir la tecla D y sirve para ver y modificar la memoria de datos (fig. 5.4) y se emplea de la misma manera que el comando "Memory".

Comando "Registers".- Este comando se invoca con la tecla R. Se emplea para modificar el contenido de algun registro del arreglo de registros en forma manual. Cuando ejecutamos este comando nos posicionamos sobre el primer registro, indicandolo por un cambio de color en el registro. Para movernos en el arreglo de registros empleamos las teclas de flechas de hacia arriba, abajo, izquierda y derecha. Cuando nos movemos por el arreglo de registros se nos indica por un cambio de color o tono de gris. Para modificar el contenido de algún registro primero nos colocamos sobre él y oprimimos la tecla m (modificar) y comenzamos a escribir los nuevos valores, estos se nos indica cambiando de color los nibbles que se van actualizando. Cuando actualizamos el último nibble automáticamente termina el modo modificar regresando el registro de nuevo al color que indica que estamos posicionados en él. Para terminar con el comando simplemente se oprime la tecla de retorno volviendo a quedar todos los registros con el mismo color.

Comando "Keep".- Este comando se emplea para guardar en un archivo el contenido que en ese momento tenga la memoria de programa, la memoria de datos o la memoria de microprograma. Cuando invocamos este comando aparece un menú fig. 5.5 y se nos pide que indiquemos que memoria guardar, en que archivo y que localidades queremos guardar, es decir tanto la localidad inicial

como la localidad final del rango que deseamos guardar. Cuando damos la dirección de la última localidad automáticamente desaparece la ventana y regresamos como estabamos.

Comando "Load".- Cuando queremos cargar a una de las memorias, con la información que guardamos previamente con el comando "Keep" invocamos el comando "Load" el cual también nos despliega un menú (fig. 5.6) y nos pregunta cual memoria queremos cargar, con que archivo y desde que localidad, terminando automáticamente al dar la localidad inicial de carga.

Comando "Simulate".- Usamos este comando para iniciar una simulación cuando lo hacemos aperece un menú fig. 5.7 donde se nos pide que indiquemos que velocidad de simulación deseamos. si indicamos paso a paso entonces al momento de la simulación tenemos que oprimir una tecla (excepto ESC) para cada ciclo de reloj. Si elegimos la velocidad lenta entonces cada ciclo de reloj toma aproximadamente 2 segundos. Si elegimos la opción rápida entonces el proceso corre mucho más rápido y la duración de cada ciclo depende de la velocidad de la máquina donde se corra el simulador. en cualquier momento podemos terminar la simulación con orpimir la tecla ESC.

Comando "End".- Este comando sirve para salir del simulador.

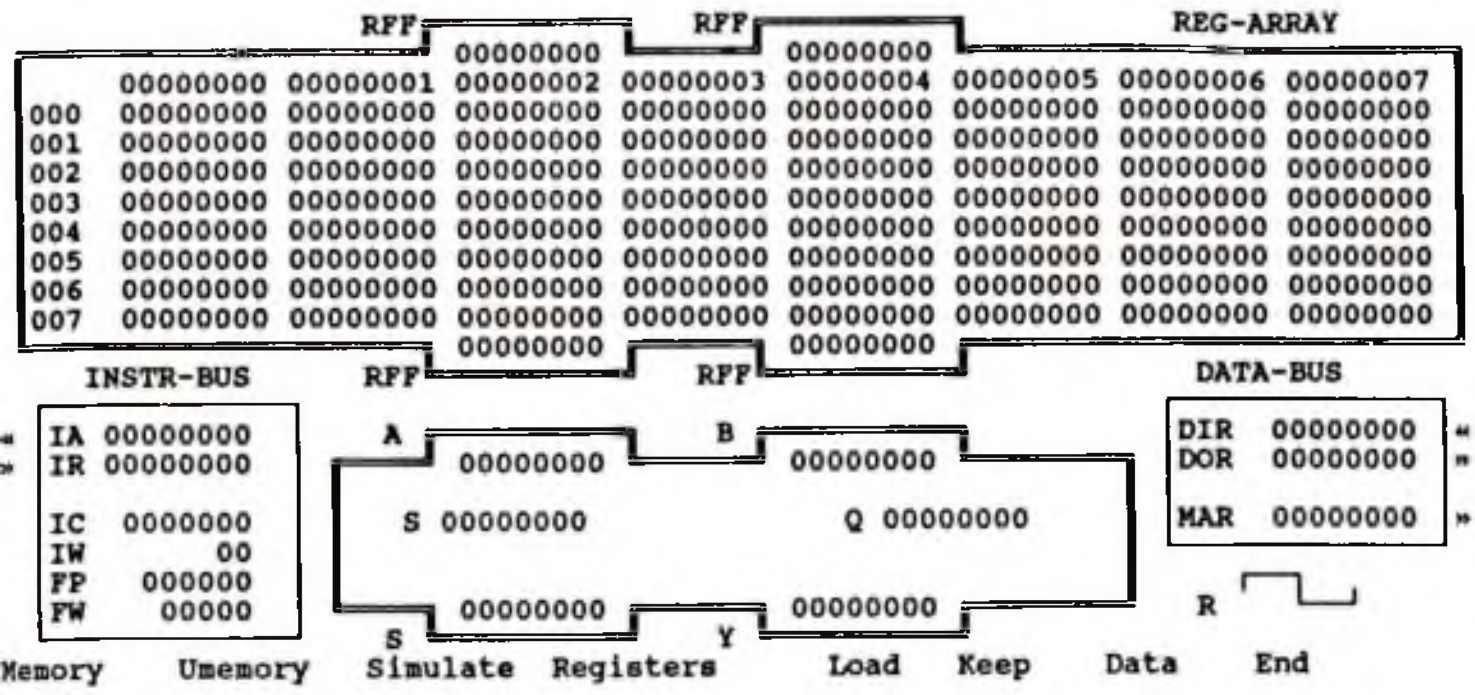

AMD 29300 Processor Simulator Processing Unit CIEA IPN 1990
RFF
RFF
REG-ARRAY
00000000 00000001 00000002 00000003 00000004 00000005 00000006 00000007
000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
002 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
003 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
004 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
005 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
006 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
007 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
INSTR-BUS
RFF
RFF
DATA-BUS
IA 00000000
IR 00000000
IC 0000000
IW 00
FP 000000
FW 00000
A
B
S 00000000
Q 00000000
DIR 00000000
DOR 00000000
MAR 00000000
R
S
Y
Memory
Umemory
Simulate
Registers
Load
Keep
Data
End

fig.5.1

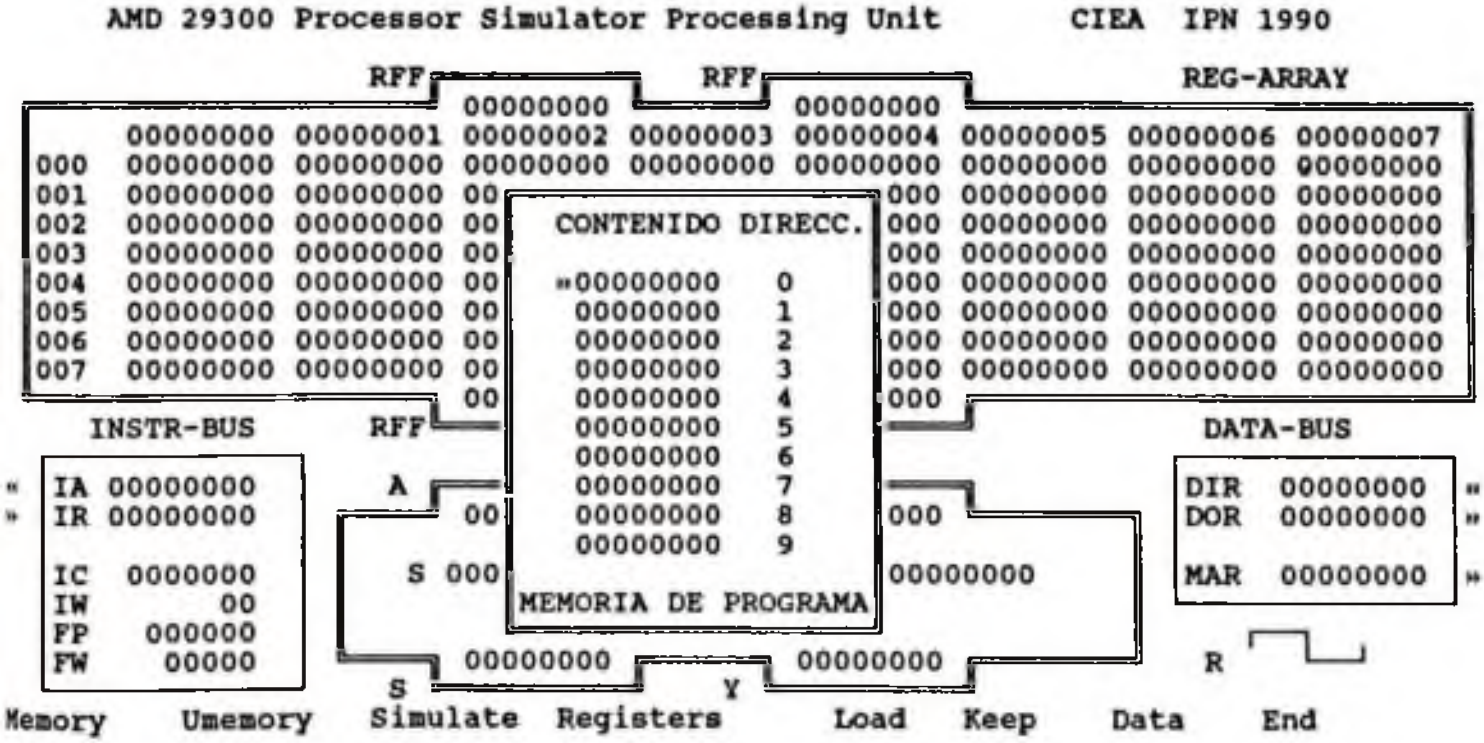

AMD 29300 Processor Simulator Processing Unit CIEA IPN 1990
RFF
RFF
REG-ARRAY
CONTENIDO DIRECC.
MEMORIA DE PROGRAMA
INSTR-BUS
RFF
DATA-BUS
IA 00000000
IR 00000000
IC 0000000
IW 00
FP 000000
FW 00000
A
B
DIR 00000000
DOR 00000000
MAR 00000000
R
S
Y
Memory
Umemory
Simulate
Registers
Load
Keep
Data
End

fig.5.2

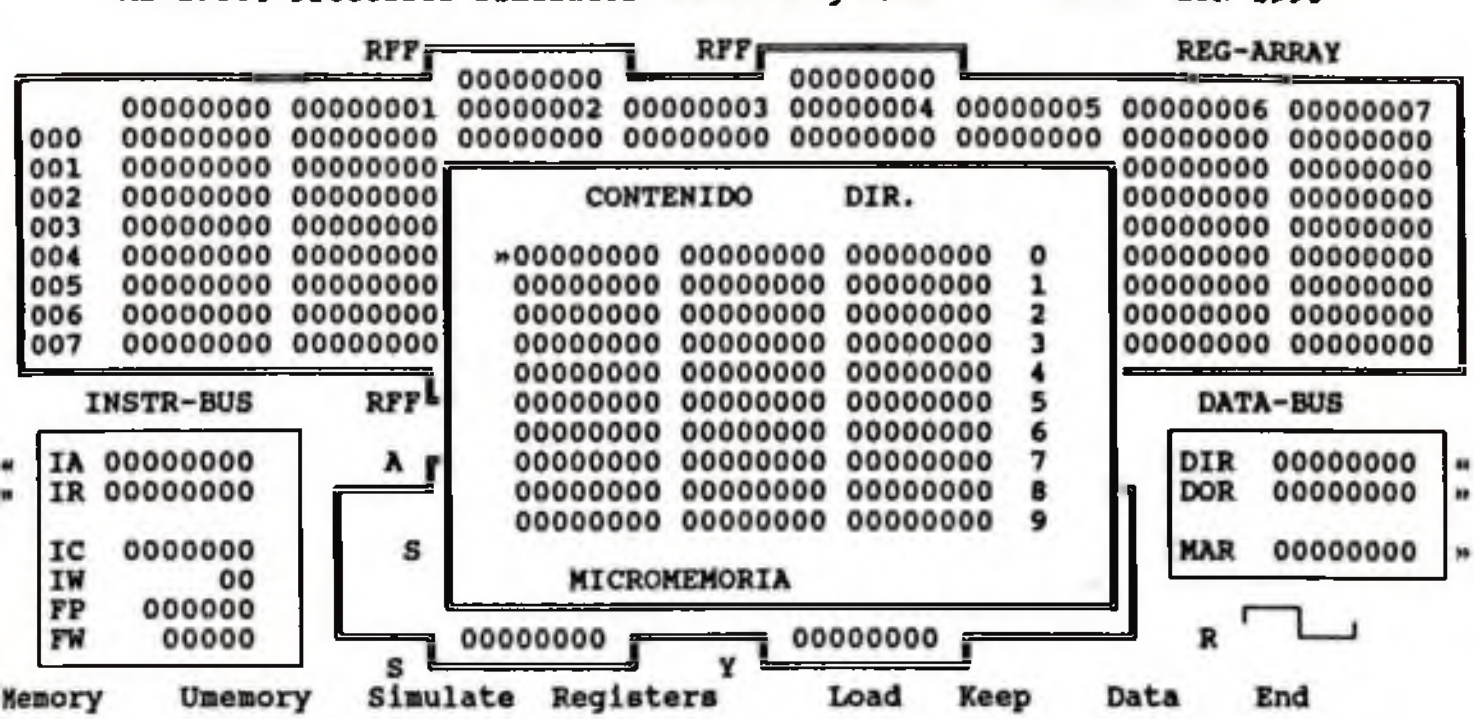

AMD 29300 Processor Simulator Processing Unit CIEA IPN 1990
RFF
RFF
REG-ARRAY
CONTENIDO DIR.
MICROMEMORIA
INSTR-BUS
RFF
DATA-BUS
IA 00000000
IR 00000000
IC 0000000
IW 00
FP 000000
FW 00000
DIR 00000000
DOR 00000000
MAR 00000000
A
S
R
Memory
Umemory
Simulate
Registers
Load
Keep
Data
End

fig.5.3

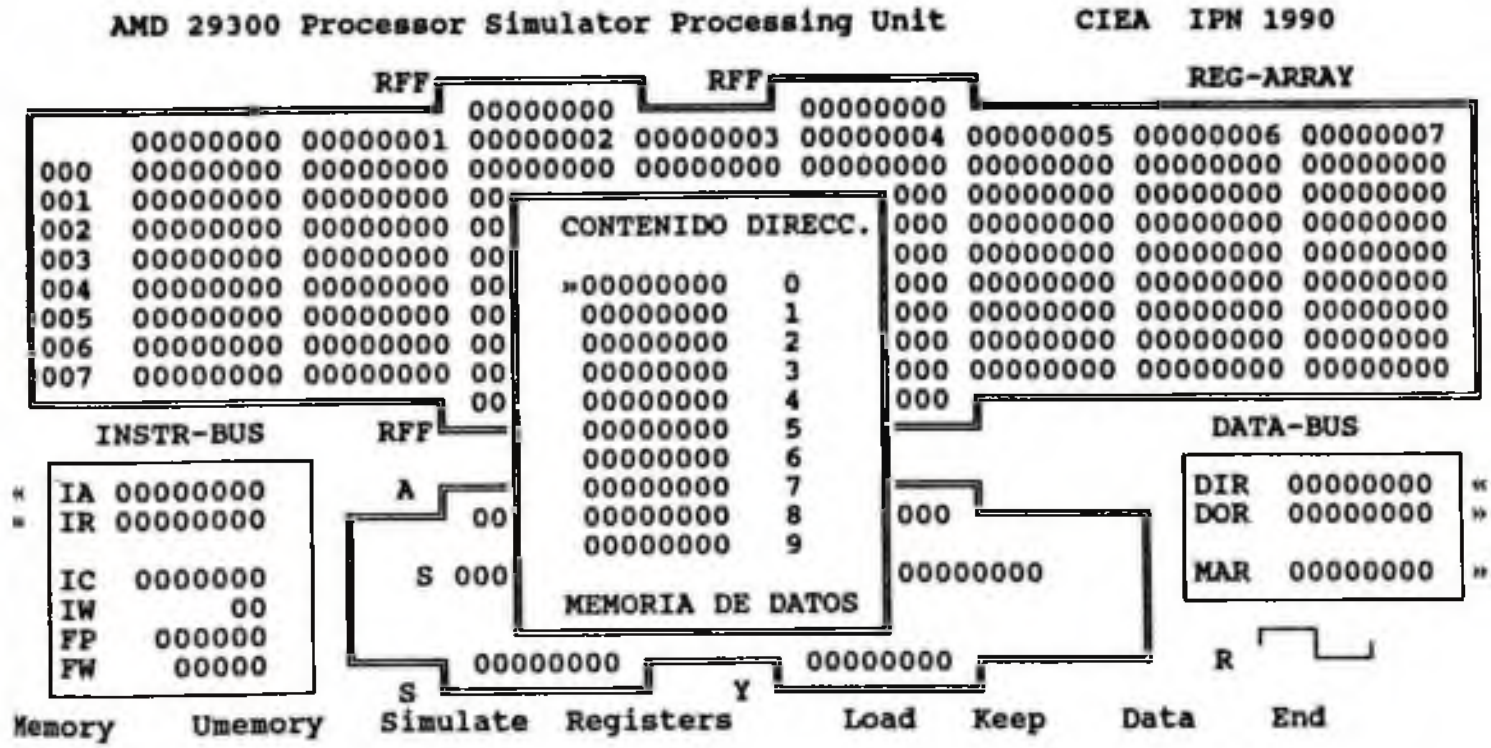

AMD 29300 Processor Simulator Processing Unit CIEA IPN 1990
RFF
RFF
REG-ARRAY
CONTENIDO DIRECC.
MEMORIA DE DATOS
INSTR-BUS
RFF
DATA-BUS
IA 00000000
IR 00000000
IC 0000000
IW 00
FP 000000
FW 00000
DIR 00000000
DOR 00000000
MAR 00000000
A
S
R
Memory
Umemory
Simulate
Registers
Load
Keep
Data
End

fig.5.4

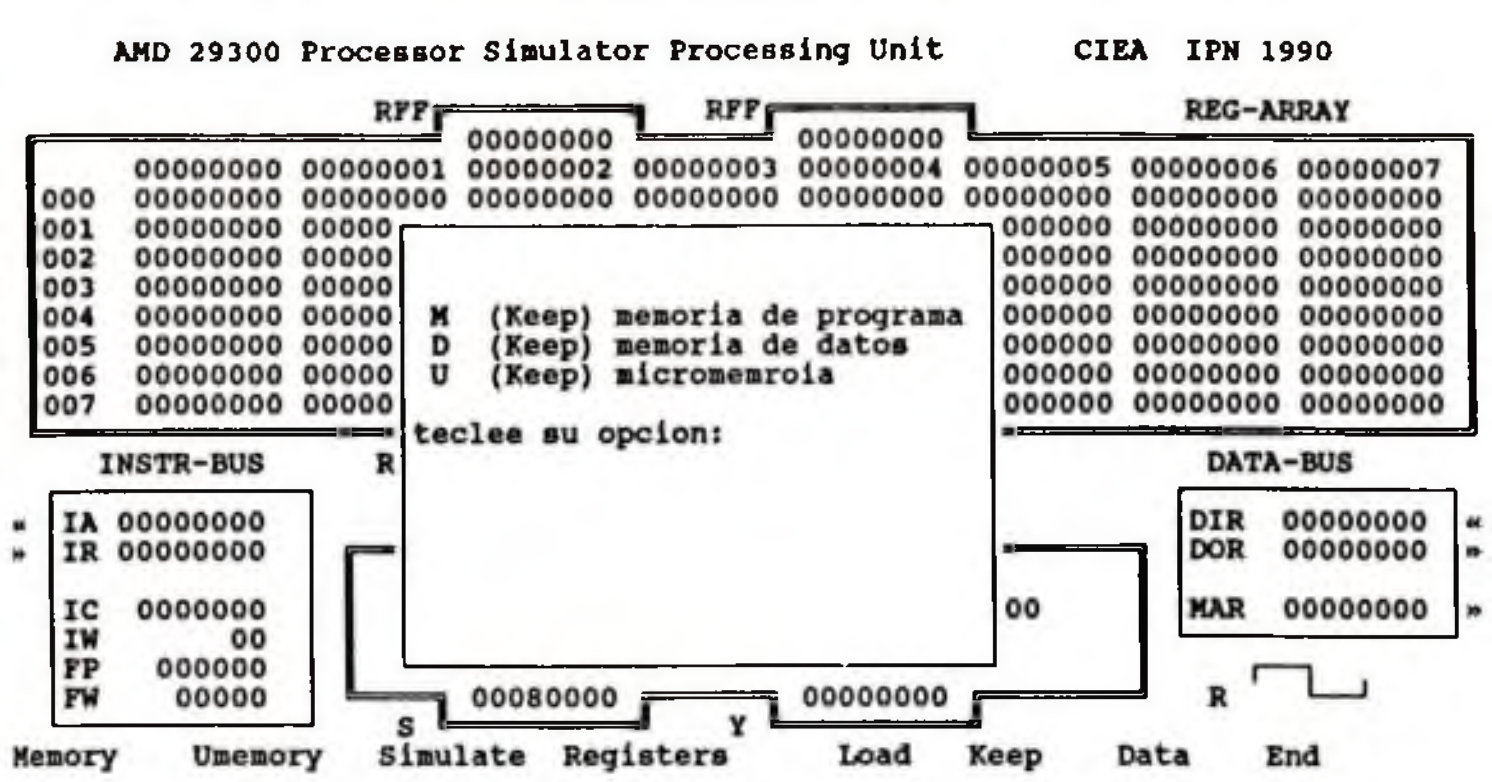

AMD 29300 Processor Simulator Processing Unit CIEA IPN 1990
RFF
00000000
RFF
00000000
REG-ARRAY
00000000 00000001 00000002 00000003 00000004 00000005 00000006 00000007
000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
001 00000000 00000 000000 00000000 00000000
002 00000000 00000 000000 00000000 00000000
003 00000000 00000 000000 00000000 00000000
004 00000000 00000 000000 00000000 00000000
005 00000000 00000 000000 00000000 00000000
006 00000000 00000 000000 00000000 00000000
007 00000000 00000 000000 00000000 00000000
M (Keep) memoria de programa
D (Keep) memoria de datos
U (Keep) micromemroia
teclee su opcion:
INSTR-BUS
R
DATA-BUS
IA 00000000
IR 00000000
IC 0000000
IW 00
FP 000000
FW 00000
00
DIR 00000000
DOR 00000000
MAR 00000000
00080000
00000000
S
Y
R
Memory
Umemory
Simulate
Registers
Load
Keep
Data
End

fig.5.5

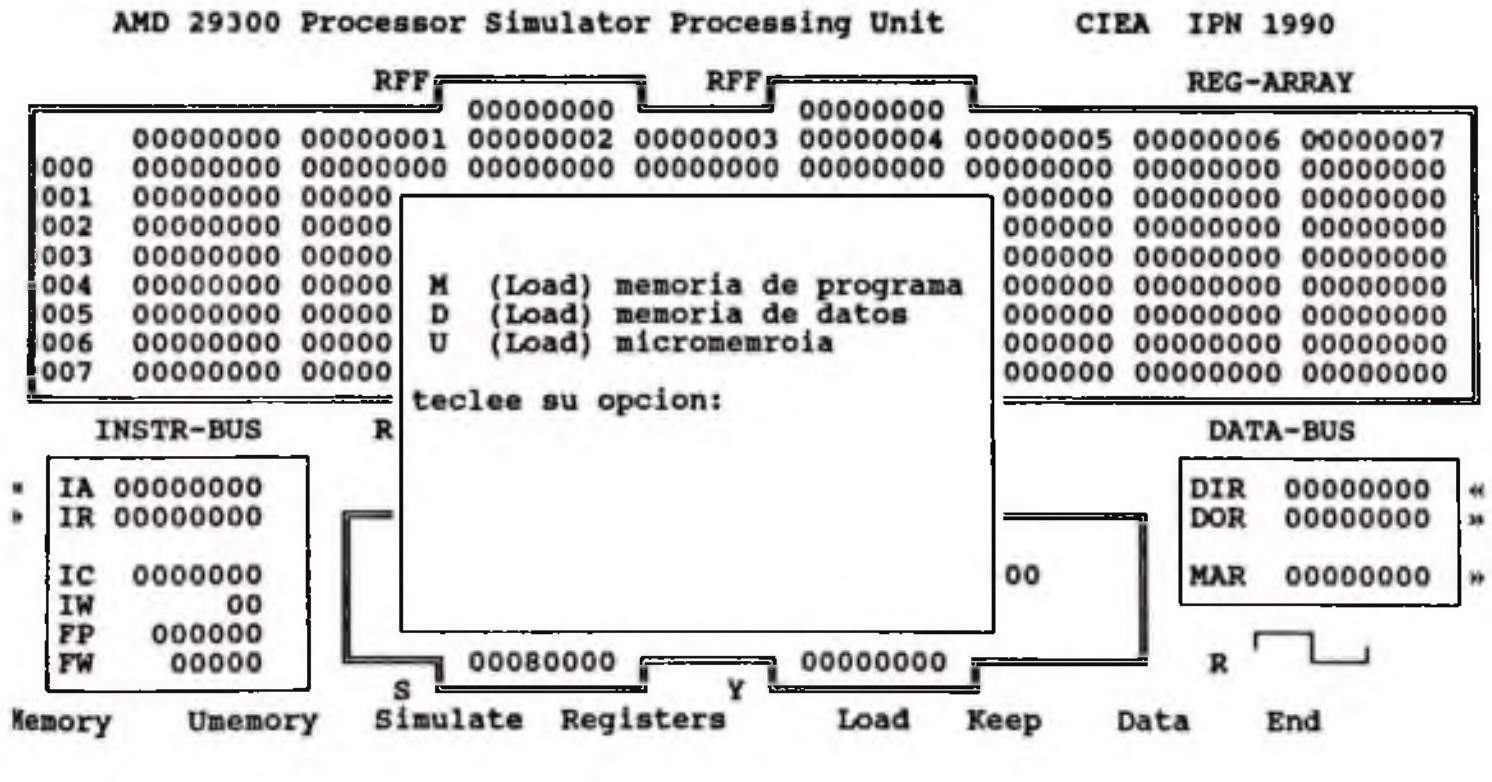

AMD 29300 Processor Simulator Processing Unit CIEA IPN 1990
RFF
00000000
RFF
00000000
REG-ARRAY
00000000 00000001 00000002 00000003 00000004 00000005 00000006 00000007
000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
001 00000000 00000 000000 00000000 00000000
002 00000000 00000 000000 00000000 00000000
003 00000000 00000 000000 00000000 00000000
004 00000000 00000 000000 00000000 00000000
005 00000000 00000 000000 00000000 00000000
006 00000000 00000 000000 00000000 00000000
007 00000000 00000 000000 00000000 00000000
M (Load) memoria de programa
D (Load) memoria de datos
U (Load) micromemroia
teclee su opcion:
INSTR-BUS
R
DATA-BUS
IA 00000000
IR 00000000
IC 0000000
IW 00
FP 000000
FW 00000
00
DIR 00000000
DOR 00000000
MAR 00000000
00080000
00000000
S
Y
R
Memory
Umemory
Simulate
Registers
Load
Keep
Data
End

fig.5.6

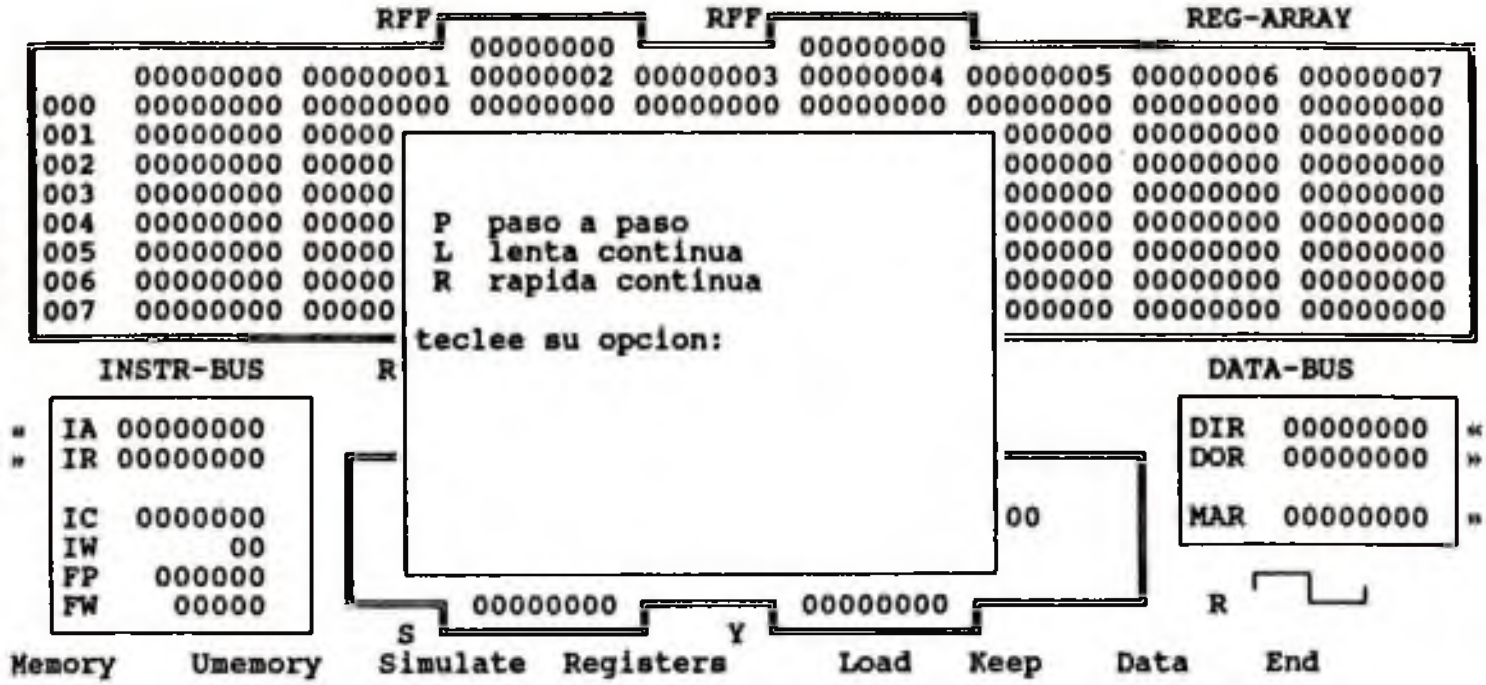

RFF
00000000
RFF
00000000
REG-ARRAY
00000000 00000001 00000002 00000003 00000004 00000005 00000006 00000007
000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
001 00000000 00000 000000 00000000 00000000
002 00000000 00000 000000 00000000 00000000
003 00000000 00000 000000 00000000 00000000
004 00000000 00000 000000 00000000 00000000
005 00000000 00000 000000 00000000 00000000
006 00000000 00000 000000 00000000 00000000
007 00000000 00000 000000 00000000 00000000
P paso a paso
L lenta continua
R rapida continua
teclee su opcion:
INSTR-BUS
R
DATA-BUS
IA 00000000
IR 00000000
IC 0000000
IW 00
FP 000000
FW 00000
00
DIR 00000000
DOR 00000000
MAR 00000000
R
00000000
00000000
S
Y
Memory
Umemory
Simulate
Registers
Load
Keep
Data
End

fig.5.7

6.- DISEÑO FISICO DEL PROCESADOR MIRAGE.

6.1.-ORGANIZACION DEL PROCESADOR MIRAGE.

El procesador MIRAGE está organizado en dos secciones básicas:

1.- Sección de Control de Programa.
2.- Sección de Ejecución de Datos.

La Sección de Control de Programa (fig.2.1) está compuesta de las siguientes partes:

- Contador de Programa (P.C.)
  El P.C. que contiene la dirección de la próxima instrucción a ejecutar y está compuesto de circuitos contadores TTL 74S162 con los cuales se puede direccionar con 32 bits de capacidad. El tiempo máximo de carga es de 15 ns.

- Cola de Instrucciones.
  La cola de instrucciones formada por los registros EXECUTE y W/R, están compuestos por 9 chips TTL 74S374. Estos registros son cargados con una nueva instrucción cada ciclo de reloj (130 ns).

- Interface con la Memoria de Programa (P.M.).
  La P.M. tiene conectado al bus de direcciones el P.C. y en su bus de datos (32 bits) se encuentra conectado en el byte más alto el secuenciador, los otros tres bytes están conectados a la cola de instrucciones. La memoria de programa está compuesta (para prueba) de 4 chips CMOS Fast Static RAM MCM6164CC70 de 8k x 8, con un tiempo de acceso de 70 ns.

- Cola de Microinstrucciones.
  La cola de microinstrucriones formada por los registros EXECUTE y W/R, están compuestos por 14 chips TTL 74S374. Estos registros son cargados con una nueva microinstruicción cada ciclo de reloj (130 ns).

- Memoria de Microprograma (M.M.).
  La memoria de microprograma está compuesta por 11 chips TTL PROM DM87SR474N de 518 x 8, con un tiempo de acceso de 20 ns.

- Secuenciador (SEQ).
  El circuito secuenciador de microprograma es un chip CMOS AMD29C331, con un tiempo de retardo de propagación combinacional de 25 ns.

- Multiplexores.
  Todos los multiplexores de la fig.2.1 (parte1 y parte2) están compuestos por multiplexores TTL 74S257.

La Sección de Ejecución de Datos está compuesta de las siguientes partes:

- Unidad Aritmética y lógica (ALU).
  La ALU lleva a cabo todas las operaciones de proceso de datos. La ALU es un chip AMD29332 con ancho de 32 bits y con un retardo de propagación combinacional de 32 ns para datos y de 45 ns para banderas.

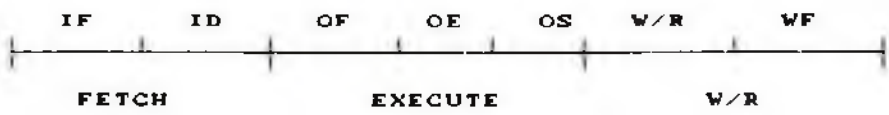
- Arreglo de Registros (F.R.).
  El arreglo de registros está compuesto de dos chips CMOS AMD29C334 de 64 x 18 bits para así formar 64 registros con ancho de 36 bits, con tiempos de acceso de lectura de 35 ns y de escritura de 30 ns.

Registros Interface con la D.M.

MAR.- Registro de dirección de D.M.
DIR.- Registro de Entrada de datos al procesador.
DOR.- Registro de Salida de datos del procesador.
Memoria de Datos (D.M.).La D.M. es igual que la memoria de programa.

Los tres registros anteriormente mencionados tienen un ancho de 32 bits, realizados con chips TTL 74S374 (MAR,DOR) y TTL 74S244 (DIR). El registro MAR contiene la dirección de localidad de D.M. accesada. El registro DOR sirve como buffer para los datos ha ser almacenados en D.M., mientras DIR es un buffer para el dato traído de D.M.

6.2.-DIAGRAMAS DE TIEMPO.



IF - TRAER UNA INSTRUCCION DE P.M.

ID - DECODIFICACION DE INSTRUCCION.

OF - TRAER LOS OPERANDOS.

OE - EJECUCION CON OPERANDOS TRAIDOS.

OS - ALMACENAMIENTO DE OPERANDOS.

W/R- LECTURA O ESCRITURA A D.M.

WF - ESCRITURA AL ARREGLO DE REGISTROS DE UN DATO DE D.M.

fig.6.1.- Subdivisiones de las 3 fases de una instrucción del procesador MIRAGE.

La fase FETCH de una instrucción del procesador MIRAGE está dividida en dos partes como se puede observar en la fig.6.1. La primera parte es el IF, la cual en el diagrama de tiempos de la fig.6.2 corresponde al tiempo máximo de retardo por actualización del P.C. (15 ns) más el tiempo de acceso para lectura a la P.M. (70 ns).

Después de transcurrido el IF se realiza el ID, el cual consta de dos partes, una que es el tiempo que tarda el secuenciador en dar un resultado en sus salidas (25 ns) y la segunda parte que consiste en el tiempo de acceso a la M.M. (20 ns).

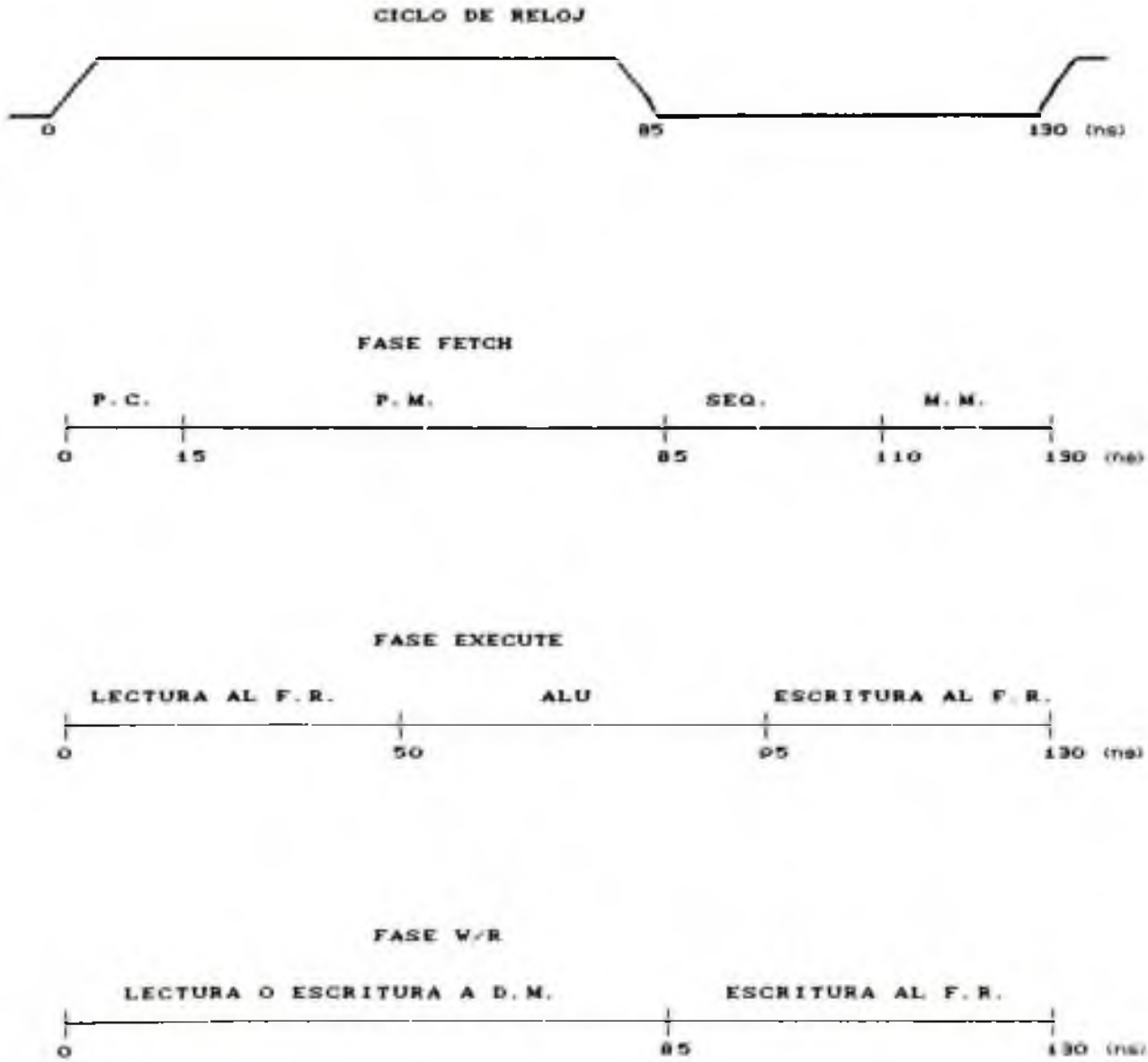


fig.6.2.- Diagramas de tiempo.

Cuando el procesador MIRAGE, se encuentra realizando una macroinstrucción toma lugar un prefetch de microinstrucción, posible ya que la actual microinstrucción tiene la información necesaria para hacer que el secuenciador indique que microinstrucción será ejecutada en el proximó ciclo de reloj, aunque esta operación en el secuenciador sea de salto se dispone del tiempo necesario después de la actualización de banderas (de la ALU) para traer la próxima microinstrucción a ejecutar.

De la fig.6.1 se puede ver que la fase EXECUTE se ha dividido en tres partes (OF, OE y OS). OF corresponde al tiempo de acceso de lectura del arreglo de registros (fig.6.2) y retardos por corrimientos en la cola de instrucciones y de microinstrucciones. La parte OE corresponde al tiempo (fig.6.2) en que la ALU realiza una operación y actualiza sus banderas (carry, overflow, zero, negative y link). La parte OS se compone del tiempo en el cual se realiza un acceso de escritura al arreglo de registros. Los niveles del ciclo de reloj combinados con algunos bits de control de la microinstrucción, controlan la escritura y lectura al F.R.

La fase W/R de la fig.6.1 se divide en dos partes (fig.6.2), una donde se hace el acceso de lectura o escritura a D.M. y la otra donde se realiza un acceso de escritura al F.R. cuando de la D.M. se ha leido un dato.

CONCLUSIONES:

Lo esperado del procesador MIRAGE se puede resumir en los siguientes puntos:

- Un manejo simple de instrucciones primitivas.

- Decodificación simple y rápida.

- Una memoria rápida de microintrucciones horizontales (82 bits).

- Un arreglo de registros de CPU de doble acceso, el cual permite accesar dos registros y almacenar un resultado en un solo ciclo de 130 ns.

- Almacenamiento de datos, en el arreglo de registros, en vez de la memoria principal.

- Rápido acceso a la memoria.

- Todas las instrucciones son ejecutadas en un solo ciclo de 130 ns.

- Capacidad de definición de macroinstrucciones.

- Retardo de un solo ciclo para la ejecución de un salto.

Los puntos antes citados, han sido realizados en un diseño con circuitos de la línea AMD29300.

REFERENCIAS.

1.- Patterson, D.A. "Reduced Instruction Set Computers". Communications of ACM, vol.28, número 1 (Enero 1985), 8-20.

2.- Patterson, D.A. "RISC watch". Computer Architecture News, vol. 12, número 1 (Marzo 1984), 11-19.

3.- Hunter, C.B. "Introduction to Clipper Architecture". IEEE Micro (Agosto 1987), 7-26.

4.- Tabak, D. "RISC Systems". Microprocessors and Microsystems, vol. 12, número 4 (Mayo 1988), 179-185.

5.- Wilson, R. "RISC CPUs tune up for embedded computing". Computer Design (Mayo 1989), 36-38.

6.- Tabak, D. "High-performance RISC systems". Microprocessors and Microsystems, vol.13, número 6 (Julio 1989), 355-371.

7.- Dubose D.K., Fotakis D.K., Tabak D. "A Microcoded RISC". Computer Architecture News, vol. 14, número 3 (Junio 1986), 5-16.

9.- AMD29C331 Handbook, Advanced Micro Devices (Septiembre 1987).

10.- AMD29332 Handbook, Advanced Micro Devices (Julio 1987).

11.- AMD29C334 Handbook, Advanced Micro Devices (Agosto 1987).

El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobo esta tesis el 19 de Junio de 1990.

Dr. Adriano de Luca Pennacchia

Dr. Jan Janecek

M. en C. Andrés Vega García