IMPLEMENTACION DE UNA MEMORIA DE UN SOLO NIVEL

HERNAN VILLANUEVA ARAYA
ING. CIVIL ELEC., MSc., Phd.
Depto. Electrónica, UTA,Chile

RESUMEN

El trabajo hace una proposición para una memoria de un solo nivel, implementada por memorias pseudo-asociativas, formadas por un banco de memorias activas correspondientes a la memoria global de un sistema multi-microcomputador, el cual permite efectuar búsquedas concurrentes para el esquema de reemplazo de páginas de la memoria propuesta.

INTRODUCCION

El concepto de memorias de un nivel (10 fue sugerido para posibilitar un gran espacio de direccionamiento implementado mediante un almacenamiento secundario - lento, tiene acceso automático a sectores activos de este espacio localizados en pequeñas memorias primarias de alta velocidad. Por lo tanto, es un medio - de direccionar unificadamente todo el almacenamiento virtual.

Es conveniente limitar la magnitud de un sector de este almacenamiento a una página de magnitud fija y transferir páginas entre las memorias primarias y secundarias. Cada pagina, en la memoria de alta velocidad tiene asociada la dirección del ocupante presente en la memoria secundaria, la cual se mantiene en una tabla de páginas.

Para accesar una palabra almacenada debe efectuarse una búsqueda en la tabla de páginas, comprobando la presencia de la página requerida. Si ella es encontrada, la palabra puede ser accesada. En caso contrario, debe activarse un mecanismo de transferencia para mover la página desde la memoria secundaria lenta a la memoria primaria.

Se han realizado diversas implementaciones (2,3) que acortan esta búsqueda altamente consumidora de tiempo. Así, es posible construir un arreglo de memoria, direccionable por contenido, que permite una traducción rápida de la dirección virtual a la dirección - física, comparando simultáneamente la página virtual requerida con todas las páginas virtuales contenidas en la memoria primaria. Para ello se agrega a cada elemento almacenador un bit de "matching" el cual - se activa cuando existe correspondencia o se desactiva en caso contrario.

Sin embargo, memorias de millones de bytes no pueden ser actualmente implementadas completamente por arreglos asociativos debido al alto costo de tales dispositivos. Diversas máquinas con memoria virtual intentan resolver el problema de velocidad de las tablas de páginas implementadas en software y el del costo de los arreglos asociativos mediante la construcción de arreglos que mapean parte de la memoria, usando dispositivos denominados buffers de tabla tipo "look aside" respaldados por la localidad de referencia de los programas. Sin embargo, si esta localidad no es total, aparecen otra vez demoras, en - este caso debido a la búsqueda fallida de páginas.

Una solución alternativa usa el hecho que el comportamiento funcional de las memorias asociativas puede simularse en software mediante código de hash, posibilitando implementar grandes memorias pseudo-asociativas. La velocidad de tal implementación puede mejorarse mediante la búsqueda paralela utilizando un sistema multi-microcomputador.

1.- MEMORIAS DIRECCIONABLES POR CONTENIDO. (CAM)

Las CAMs o memorias asociativas (4) son dispositivos de almacenamiento que pueden ser accesadas en - base a su contenido. Las propiedades que las distinguen de los sistemas convencionales son :

- Los items almacenados pueden ser extraidos sin necesidad de conocer su localización física.
- Las operaciones que transforman datos pueden efectuarse simultáneamente sobre muchos conjuntos de argumentos mediante una única instrucción.

Existen diversas organizaciones de CAMs: paralela - total, bit-serie, palabra-serie y orientadas a bloques.

Avances recientes en VLSI posibilitan la implementación de sistemas asociativos utilizando tecnología de semiconductores en vez de criogenia o películas magnéticas delgadas, ya que se han encaminado soluciones para dos problemas importantes : Costo de implementación e incremento en la capacidad de excitación para interrogación de los buses de datos.

Sin embargo, permanece el problema de sus respuestas múltiples. En general, una operación de búsqueda podría obtener un conjunto de respuestas y el propósito de la búsqueda sería identificar un conjunto de argumentos sobre los cuales se desea efectuar - una operación específica.

Si esta operación debe efectuarse fuera del dispositivo asociativo, algunos miembros del conjunto deben ser extraidos y transferidos. Por lo que debe proveerse el medio para indicar el número de respuestas y seleccionarlas en algún orden.

El primer problema es identificar el número de respuestas, una solución es usar una indicación binaria o terciaria. El segundo problema es seleccionar un miembro del conjunto de respuestas. La selección podría ser al azar o de acuerdo a alguna - prioridad.

2.- IMPLEMENTACION DE MEMORIAS ASOCIATIVAS EN HARDWARE.

Teóricamente, sólo se requieren operaciones de matching exacto con enmascaramiento y operaciones multiescritura para implementar cualquier operación lógica. La figura 1 muestra un diagrama de bloques - de una memoria asociativa, la cual consiste de un - número de celdas almacenadoras equipadas con lógica electrónica para efectuar el matching. (4)

Para buscar un item, este se coloca en el registro de interrogación y la memoria se lleva al modo de - búsqueda. La lógica digital agregada a cada celda compara el contenido del elemento con el contenido del registro de interrogación. El bit de Match es activado o desactivado de acuerdo con el resultado. El registro de enmascaramiento puede seleccionar aquellas partes del registro de interrogación que - se desean comparar.

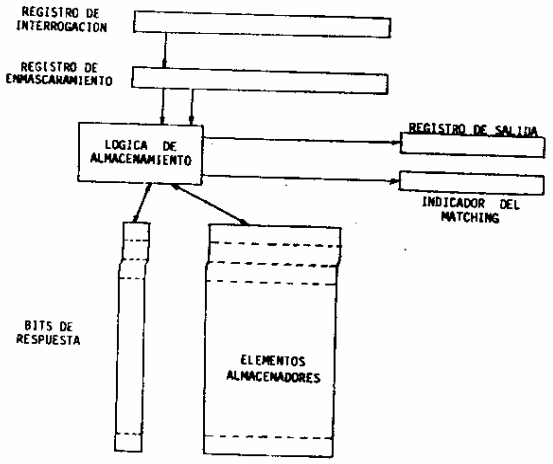


FIGURA 1 : ALMACENAMIENTO ASOCIATIVO

Sin embargo, la ineficiencia de alguna de estas operaciones imponen la implementación en hardware de algunas primitivas. La velocidad de la memoria asociativa depende de qué operaciones se llevan al hardware y de que elementos dispone.

3.- MEMORIAS PSEUDO-ASOCIATIVAS. (5)

Una memoria pseudo-asociativa es una versión simulada en software de una memoria asociativa, cuyo comportamiento funcional teniendo como única operación el matching exacto, puede simularse mediante técnicas de código hashing usuales en implementación de compiladores y bases de datos.

Las memorias pseudo-asociativas consisten de una table look-up con un número de entradas igual al número de items requeridos, y de una técnica de búsqueda. Los elementos en la tabla se identifican mediante alguna llave asociada con sus contenidos. Una implementación alternativa usa memorias RAM convencionales con algoritmos de búsqueda implementados en hardware.

Considerando que las funciones hash producen un número N al azar para usarlo como índice, es conveniente que N sea una potencia de 2. Así, si las transformaciones hash generan "m" bits, el espacio requerido por la tabla será :

$$N = 2^m \quad \text{entradas}$$

También debe disponerse de un modo de distinguir una entrada vacía de otra posiblemente válida. Si el cero puede excluirse como llave válida, entonces cero puede usarse para significar que la entrada está vacía. La tabla completa debe inicializarse a cualquier estado que se elija como señal de entrada vacía.

Luego, los dos tópicos principales para diseñar un algoritmo de hash son :

- Escoger una función de hash, la cual efectúa la transformación desde una llave a una dirección y
- Escoger un buen método para resolver colisiones.

4.- TECNICAS DE BUSQUEDA TIPO HASH. (6,7)

Un código hash es un algoritmo que asocia a cada item una dirección en una tabla. Este código calcula la localización del item en vez de efectuar una búsqueda exaustiva de él y es posible definirlo como el siguiente mapping :

hash : {todas las llaves posibles} $\rightarrow$ {1,2,3,..., N; localizaciones}

Esta transformación llave-a-dirección, bien diseñada, esparse las direcciones calculadas en forma uniforme a través de las direcciones disponibles.

Una evaluación comparativa (8) de algoritmos de hashing utilizando los métodos direccionamiento abierto y revalse encadenado para resolver colisiones, indica que el código de hash por división presenta el mejor desempeño con direccionamiento abierto. También, muestra que no existen mayores diferencias entre diversos métodos de hashing con resolución de colisiones por revalse encadenado; que el número de pruebas para encontrar un item disminuye consistentemente a medida que el número de items por entrada a la tabla crece y que las técnicas con resolución con direccionamiento abierto muestran un comportamiento errático.

Se comprueba que para todas las técnicas el número de pruebas y el número de colisiones crecen a medida que el factor de carga aumenta.

Un método ideal de transformaciones debe insertar todo el conjunto de llaves en direcciones diferentes. Sin embargo, la uniformidad en la distribución de direcciones no es sinónimo con el mapping de llaves en direcciones con igual probabilidad. Por eso, una transformación eficiente debe preservar cualquier uniformidad existente en las llaves. Tampoco es posible dejar de lado alguna técnica de hash debido a un desempeño más bajo cuando las colisiones se resuelven con revalse encadenado.

Cuando las colisiones se resuelven por direcciona miento abierto, el codigo hash por división y el de la función cuadrado-media son los mejores.

5.- TECNICAS PARA RESOLVER COLISIONES

Una vez que se han insertados algunos items en la tabla, es posible computar las mismas direcciones para llaves diferentes, causando colisiones entre localizaciones asignadas, por lo que debe encontrar se otro lugar para dichos items. Para ello se han desarrollados diversos métodos para soluciones el problema de las colisiones y los más conocidos son los algoritmos de encadenamiento y los de direccionamiento abierto.

Los algoritmos que resuelven colisiones mediante encadenamiento con listas separadas, consisten en M listas unidas, una para cada código hash.

Debe incluirse un campo lista en cada item, teniendo así M cabezas de listas. Despues de efectuar el hashing, es necesario hacer una búsqueda secuencial en la lista. La búsqueda es rápida porque las listas son cortas. Si hay N llaves y M listas, el promedio de la magnitud de las listas es N/M; por lo tanto el hashing en este método disminuye la cantidad de trabajo secuencial en aproximadamente un factor M. Si se requiere aumentar la velocidad, la magnitud M de la tabla debe ser grande. En tal situación es probable que existan listas que se encuentren vacías y una gran cantidad del espacio no se use.

- El método alternativo se denomina encadenado con listas coalecentes y se usa cuando las dimensiones de los items es pequeña.

El almacenamiento de los items se traslapa con la cabeza de las listas y no existe un área de revalse para cada lista. Algunas versiones disponen de un área bodega común para todas las colisiones. Como el espacio de almacenamiento es fijo, es posible que una cabeza de lista choque con otro item insertado por lo que también debe disponerse de una técnica para encontrar una localización vacía.

Los métodos con direccionamiento abierto no requieren del campo de link para resolver las colisiones. Por lo tanto requieren menos memoria para la tabla look-up y la localización de los items que chocan se calculan independientemente de la información dada por los item en colisión, como es el caso de la alternativa encadenada. Las técnicas de direccionamiento abierto resuelven las colisiones generando nuevas direcciones a probar en la tabla. La secuencia de pruebas termina cuando esta se recicla y la tabla se declara llena.

El algoritmo que resuelve las colisiones debe generar una secuencia exaustiva. Cada localización de la tabla debe ser probada exactamente una vez antes de que la tabla se declare llena y el promedio de prueba por búsqueda debe ser mínimo.

Esto es posible ya sea acortando el tiempo promedio de computo de h(k), la función de hashing, o acortando la longitud promedio de la secuencia.

Para acortar el tiempo promedio de computo de las funciones de hashing h(k), es conveniente hacer los cálculos de la secuencia en base a la primera evaluación, esto es :

$$hi(k) = hO(k) + k(i)$$

para $i >= 1$ y k = llave del item

aquí d(i) es una simple función desplazamiento independiente de la llave y la suma se efectúa módulo la magnitud de la tabla. En efecto hO(k) es la única función hashing y d(i) sólo resuelve colisiones o sea aquellos casos en que :

$$hi(k) = hj(k') \quad ; k <> k'$$

El problema crucial de los algoritmos con direccionamiento abierto es el diseño de la ley incremental debido al amontonamiento de items en racimos.

Los racimos, primarios y secundarios, son el gran obstáculo de un hashing efectivo debido a que aumenta la probabilidad de encontrar una localización ocupada.

7.- DISEÑO DE ALGORITMOS DE BUSQUEDA PARALELA

Dos de las mejores técnicas de búsqueda son el método encadenado y doble hashing con resolución de direccionamiento abierto.

Un análisis de implementaciones secuenciales de algoritmos de búsqueda indica que pueden caracterizarse por construcciones de lazo en donde se repiten acciones de comparación entre llaves.

Es posible, que disponiendo de un hardware de multiprocesadores y utilizando un control apropiado, permitir que la búsqueda proceda concurrentemente en vez de secuencialmente, considerando que estos conjuntos de comparaciones operan en cada interacción sobre datos distintos.

Dos tipos de estructuras de software son especialmente útiles en la implementación de memorias pseudo-asociativas, ellas son estructuras traslapadas y las pipelines.

Las técnicas de pipeline aplican a un conjunto de acciones independientes que se presentan seriamente en un lazo, actuando en cada iteración sobre datos distintos. Normalmente más de un conjunto de datos son procesados por operaciones diferentes y simultáneas. Las acciones independientes pueden ser controladas por un constructor pipeline pero las acciones que afectan el fin del lazo deben efectuarse en el orden original. En aquellos casos en que no se puede aplicar el constructor pipeline, podría considerarse la solución traslapada, la cual requiere varias copias distintas de código y el control permite operaciones concurrentes de códigos distintos con datos diferentes.

7.- ORGANIZACION PARA BUSQUEDA PARALELA

Una prueba de hash para una llave determinada se logra mediante la siguiente secuencia de operaciones:

a) Computar la dirección de hash hj(k), seleccionada desde una secuencia hash h1(k), h2(k),...
b) Extraer kj desde la dirección de hash hj(k).
c) Comparar la llave k con kj.
d) Decidir en base el resultado de la comparación.

En un hashing puramente secuencial, los pasos a) hasta d) se repiten iterativamente. Cuando cada paso se ejecuta en hardware, el último factor que determina el desempeño es el número de llaves accesadas en el paso b). i.e. J llaves se leen simultaneamente en b) y J comparaciones pueden efectuarse en paralelo.

La estructura de la organización propuesta en este trabajo, tiene una tabla look-up implementa en la memoria global un sistema multi-microcomputador.

El hardware de hash paralelo consiste de J módulos de bancos de memoria, cada uno equipado con M palabras de almacenamiento, un generador de direcciones de hash y una unidad de control.

En la memoria de un solo nivel, las transformaciones de llave-a- dirección se efectuan secuencialmente, cada vez que el programa requiere una nueva instrucción. Por lo tanto, el proceso requiere una nueva instrucción. Por lo tanto, el proceso concurrente se restringe a la rarea de búsqueda.

Como cada vez, una sola página virtual se debe buscar, insertar o borrar, se requiere de un solo generador de direcciones hash, el cual es implementado por uno de los procesadores del sistema multi-microcomputador.

Cada banco de memoria tiene su propia unidad comparadora mas algunos buffers controladores de interlocks. Cada uno de ellos implementado por otro procesador. La figura 2 muestra la estructura propuesta para la memoria pseudo-asociativa. La arquitectura de memoria con multi-bancos aparece naturalmente, debido al requerimiento de busqueda simultánea, la cual debe efectuarse concurrentemente por varios procesadores.

En orden a evitar más de una prueba por entrada de la tabla, se hace necesario dividir la tabla en secciones dentro de un espacio de direcciones fijo, asignando un espacio de direcciones propio a cada procesador. De esta manera, cada espacio de memorias con su procesador asociado constituye un banco de memoria activo.

Considerando que el espacio de memoria de la tabla look-up es constante, el espacio de memoria de cada banco es más pequeño a medida que el número de procesadores buscadores aumenta.

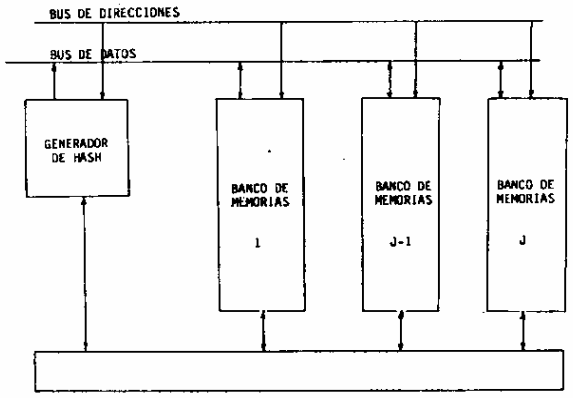


FIGURA 2 : ESTRUCTURA DE LA MEMORIA PSEUDO-ASOCIATIVA

La configuración general propuesta en la figura 2 - tiene varias posibilidades de acuerdo con el modo en que los datos y la información de control fluyan dentro de ella. La figura 3 muestra tres alternativas estudiadas :

a) Búsqueda asincrónica. Muestra a Pn = 0 como el generador de hash, el cual emite la página virtual a buscar.

El procesador de cada banco, procede asincronicamente respecto de los otros procesadores buscadores, generando enteramente la secuencia de hash a su propia velocidad, seleccionando aquellos índices que están en el espacio de direcciones de su propio banco de memoria.

El resultado del procesador buscador es devuelto a Pn = 0, el que detiene la tarea ya sea cuando ha recibido todas las respuestas desde los bancos de memoria o cuando uno encuentra la llave que hace matching.

b) Búsqueda sincrónica. Muestra a Pn = 0 como el generador de hash, al igual que en a). Este procesador emite la página virtual y genera la secuencia de hash como sigue :

Una dirección de hash es enviada a cada procesador, cada uno de los cuales procede a efectuar la búsqueda usando esa dirección. Los resultados de la búsqueda se envían a Pn = 0. Si el item no es hayado Pn = 0 genera otra dirección de hash y la transmite a todos los procesadores. Así todos los procesadores buscan sincrónicamente. Esta búsqueda continúa hasta que algún procesador encuentre el item o todos los procesadores fallen.

Esta alternativa tiene la ventaja de una implementación más simple, pero no posee una sobrecarga adicional en el proceso de búsqueda. Esto se debe a - que cada vez que un nuevo índice de la secuencia es enviado el [illegible] sólo uno de los buscadores hará comparaciones ya que los otros tendrán localizado [illegible] su espacio de direcciones.

c) Búsqueda [illegible]. Muestra a Pn = 0 como el ge[illegible] adicional agregado a cada [illegible]

[illegible]

Pn = 0 emite la página virtual a cada procesador. - Este entonces produce la secuencia de hash de la página virtual y selecciona que procesador puede usar el índice particular de la secuencia hash producida. El índice seleccionado se envía al procesador y se almacena en el buffer correspondiente.

Esta alternativa tiene la ventaja de traslapar el - proceso de búsqueda en los módulos del banco de memoria. El procesador de cada banco puede realizar comparaciones sin comprobar si la localización referida por el índice está o no fuera de su espacio de direcciones, lo cual es una ventaja sobre las alternativas a) y b).

Las acciones en la "pipe" son la generación del índice de hash y las comparaciones del contenido de - la entrada con la página virtual buscada.

La alternativa c) es la más rápida para implementar la memoria pseudo-asociativa, pero el precio es la memoria adicional requerida para implementar lo buffers. Como la tabla de look-up tiene N entradas,- la magnitud total del buffer es de N bytes.

El contenido de cada buffer es la sebsecuencia de - índices, que el buscador asociado con ese buffer - usa para formar comparaciones dentro de su espacio de direcciones.

FIGURA 3.a) BUSQUEDA ASINCRONA

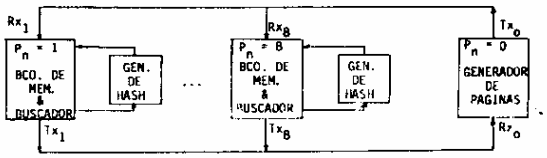


FIGURA 3.b) BUSQUEDA SINCRONICA

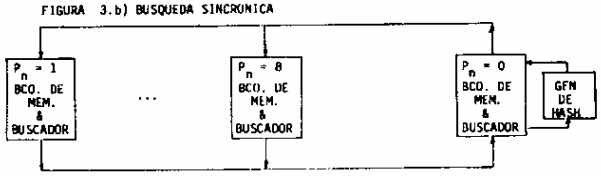
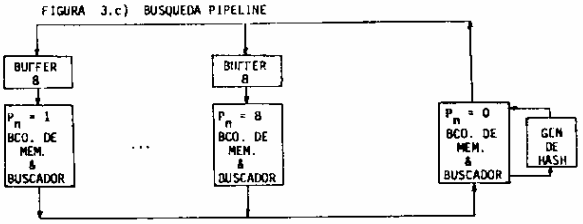


FIGURA 3.c) BUSQUEDA PIPELINE



8.- SELECCION DE ALGORITMOS PARA BUSQUEDA PARALELA

Después de decidir la estructura de la memoria - pseudo-asociativa, se debe elegir el algoritmo de hashing más apropiado para búsqueda paralela.

De acuerdo con la referencia 8, el algoritmo más - rápido es el método encadenado con listas separadas, pero este requiere una cantidad de memoria - adicional variable en donde la table look-up almacene a las colisiones. Por ello, se debe escoger

el algoritomo coalecente de hashing, ya que el espacio para la tabla debe ser fijo y además tan pequeño como sea posible.

Sin embargo, este método tiene una característica que lo hace muy rápido para sistemas uniprocesadores pero no apropiado para una busqueda mediante multiprocesadores.

Cuando se usa un algoritmo coalecente para inicializar la tabla de look-up, el algoritmo une varias cadenas de items dentro de la tabla, haciendo más rápido el proceso de búsqueda, porque este proceso busca sobre cadenas más cortas en vez de buscar en toda la tabla.

El item inicial es referido por la dirección de hash dada por la transformación llave-a-dirección.- La información en donde encontrar el item siguiente está contenida en el campo link de la entrada de la tabla. Asi, no es necesario efectuar otro computo de hashing para obtener la dirección del item siguiente.

Esta característica que lo hace un algoritmo rápido no opera en un ambiente de multiprocesadores por las siguientes razones :

- Considerando que la tabla look-up está formada por varios bancos de memoria, cada uno de ellos recorridos por diferentes procesadores, habrán casos donde las cadenas pasan de un banco a otro. Por lo tanto, las cadenas pasan de un espacio de direcciones a otro, apareciendo discontinuidades y ambos procesadores que están desarrollando la búsqueda no pueden hacerla correctamente.

La solución podría ser implementar un mensaje de comunicación entre procesadores, para pasar la dirección del siguiente item, pero esto agrega una sobre carga al sistema, perdiendo velocidad en la búsqueda.

La solución alternativa para un algoritmo de búsqueda paralelo es uno del tipo de direccionamiento abierto. Estos algoritmos no requieren conocer el contenido del campo de link para conocer la localización de la siguiente prueba. En vez de ello, estos algoritmos producen una secuencia de direcciones de hash para cada llave que cubren toda la tabla de look-up.

Para proceder a la búsqueda paralela, toda la secuencia producida por una llave puede dividirse en un número de subsecuencias iguales al número de procesadores de búsqueda.

Los elementos de estas subsecuencias son las direcciones de hash de las localizaciones donde el item buscado podría estar inserto. Ya sea cuando existe un matching durante la comparación o la localización probada esta vacía, la búsqueda es exitosa y el buscador entrega la respuesta apropiada.

Si no hay matching, la secuencia entrega la dirección siguiente a probar. Si el proceso de búsqueda ha barrido todo su banco de memoria, se detiene y responde que ha tenido lugar una búsqueda infructuosa.

De los algoritmos de direccionamiento abierto, el mejor es el de prueba lineal con técnica de resolución de colisiones de doble hashing. Este algoritmo tiene la ventaja sobre el resto de los algoritmos de direccionamiento abierto en que es el más rápido y no sufre de racimos secundarios.

La mejor implementación de algoritmo de doble hashing es el algoritmo de búsqueda lineal con incrementos ponderados, propuestos por F. Luccio (9) modificado de acuerdo con sugerencias hechas por S.K. Bandyopadhyay (10). Esta modificación evita el racimo secundario y hace más eficiente la codificación debido al computo de índices de hashing complementarios.

Finalmente puede indicarse que los métodos de direccionamiento abierto imponen la condición que la tabla de look-up no debe llenarse completamente. El rendimiento expresado en términos del número promedio de pruebas y la velocidad de búsqueda se degrada con el factor de carga cuando este supera el 70 por ciento. Esta situación puede mejorarse al agregar un colector de basura para limpiar aquellas entradas de la tabla que no se usen. Resta para terminar, la selección de la función que transforme la llave a dirección. Se ha demostrado que para algoritmos de direccionamiento abierto, con factores de carga bajo el 75%, la función de transformación llamada cuadro-medio junto con la función de división son las mejores. Sin embargo, la función de cuadrado-medio muestra un error estandar mejor para diferentes tipos de carga.

9.- CONCLUSIONES

Se propone una organización de un conjunto de bancos de memoria pseudo-asociativa, emulada sobre un sistema multi-microcomputador como implementación de una memoria de un solo nivel.

Para ello se analizan los factores que intervienen en el diseño de dispositivos pseudo-asociativos utilizando técnicas de hash apropiadas para una operación concurrente. Se determina que las técnicas de direccionamiento abierto operan sin restricciones en un ambiente multi-microcomputador y que para obtener un rendimiento e implementación eficiente, el algoritmo adecuado es el de pruebas lineales con técnicas de resolución con doble hashing. El análisis también determina que el algoritmo que hace el mapping de las llaves-a-direcciones, más propicio, es el denominado cuadrado-medio.

El trabajo siguiente consiste en medir la proposición presente, corriendo las soluciones con un número creciente de módulos, en un sistema multimicrocomputador.

10.- BIBLIOGRAFIA

1.- Kilburn T. et al "One-level storage system" -- IRE transactions n Electronic Computers,April 1962.

2.- Hoare C.A.R. and McKeag R.M. "Asurvey of storage management techniques, Parts 1 and 2". In International on Jperating Systems Techniques. Belfast 1971, Academic Press, 1973.

3.- McKeag R.M. "Survey of virtual storage techniques" Virtual Storage Infotech State od the art report, 1976.

4.- Garside R.G. "The Architecture of digital computers". Oxford University Press, 1980.

5.- Da Silva and Watson I. "Pseudo-asociative store with hardware hashing". IEE Proc. Vol 130,- Jan, 1983.

6.- Bell J.R. "The quadratic quotient method : a hash code eliminating secondary clustering". COM of the ACM, Vol 13, N°2, 1970.

7.- Maurer W. & Lewis T. "Hash table methods". Computing Surveys, Vol 7, N° 1, March 1975, pp. 5 - 19.

8.- Lum et al. "Key-to-address transform techniques : a fundamental perfomance study on large existing formatted files". COM of ACM, April 1971, Vol 14, pp. 228 - 237.