te, beta = número total de localidades - 100.

Modo 2. (Escvar). Escritura aleatoria de una sola palabra.
Modo 3. (Ligvar). Ligar nodos (5 palabras contiguas de 32 bits). Se regresan nodos, que se ligan a la lista de nodos libres.
Modo 4. (Celvar). Petición de nodos (desensarta nodos). El mantenedor de listas entrega nodo libre (5 palabras de 32 bits c/u, contiguas)
Modo 5. (Lemvar). Lectura múltiple (5 palabras de 32 bits): lee un nodo.
Modo 6. (Esmvar). Escritura múltiple (5 palabras de 32 bits): escribe un nodo.
Modo 7. (Venvar). Modo ventana.

Nótese que en la memoria de variables, un nodo o celda son 5 palabras contiguas de 32 bits c/u.

Controlador de la memoria pasiva

Los modos de este controlador son los siguientes:

Modo 1. (Leepas). Lectura aleatoria de una palabra de 22 bits.
Modo 2. (Escpas). Escritura aleatoria de una palabra de 22 bits.
Modo 3. (Ligpas). Ligar listas (liberar 2 palabras de 22 bits, o sea un nodo).
Modo 4. (Celpas). Petición de nodo libre (2 palabras de 22 bits).
Modo 5. (Lempas). Lectura múltiple a pasiva (2 palabras de 22 bits).
Modo 6. (Esmpas). Escritura múltiple a pasiva (2 palabras de 22 bits).
Modo 7. (Venpas). Modo ventana.

Nótese que en la memoria pasiva, un nodo o celda son dos palabras contiguas de 22 bits c/u.

Estado actual: actividades realizadas:
- registros: diseño lógico,
  diseño y dibujo del circuito impreso.

Características: tarjeta de registros para formar las funciones de los controladores. Se acopla al control de los controladores en el que corren micro programas para realizar las tareas para el acoplamiento de las memorias pasiva y de variables con los canales 2 y 3.

Estado actual: operativo.
Estado futuro: documentación enero de 1981.

- Controles

Actividades realizadas: diseño lógico, diseño y dibujo del circuito impreso.

Características: contiene la lógica para ordenar secuencias de transferencias de información en la tarjeta de registros de los controladores. Lo hace por medio de microinstrucciones almacenadas en PAL's (programable array logic).

Avance: estado actual: operativo en un 90%
estado futuro: conexión de los PAL's y documentación enero de 1981.

CONTROLADORES

En el plano vertical se muestra un controlador, y en la mesa yace el otro. Los controladores permiten el acceso en los modos preestablecidos a la memoria de variables y a la pasiva.

S I S T E M A DE P R O G R A M A C I O N

Visión Global

En la máquina AHR los programas van íntimamente ligados a los circuitos. Como la electrónica de máquina realiza varias funciones que antaño se hacían por programa, éstos se simplifican.

Las principales componentes de los programas son:

*en los procesadores de Lisp

- código (en lenguaje de Z80) para ejecutar cada una de las funciones primitivas de Lisp.
- código para llevar estadísticas de funcionamiento.

* en el procesador de entrada y salida

- Aquí reside un sistema operativo convencional, que se encarga de dar atención a la terminal, transferir los archivos y usar el editor.
- Un cargador que baja código de Lisp del disco del procesador de e/s a la memoria pasiva de la Computadora AHR.
- Un recolector (serial) de basura de la máquina AHR.
- Protocolos de comunicación del procesador de entrada y salida con los procesadores de Lisp, a través del "canal lento".
- Programa de iniciación del sistema.
- Manejo e impresión de errores.

* en el distribuidor (Versión 0)

- Las funciones del distribuidor de la versión 0 están programadas en una microcomputadora ZDS 1/25.

* en los controladores

- Los controladores de la memoria pasiva y de la memoria de variables son "inteligentes", y están hechos a base de arreglos lógicos programables (PAL's), los que tuvieron que microcodificarse.

Programas de los Procesadores de Lisp (Intérprete de Lisp)

La máquina AHR es capaz de procesar en paralelo cualquier lenguaje expresable en notación lambda. Para la etapa inicial de construcción y prueba de la máquina, se escogió Lisp puro (sin goto's, prog, rplaca, rplacd y otras impurezas) por cumplir con esta característica y además ser un lenguaje ampliamente conocido y utilizado.

El intérprete de Lisp que se está implementando difiere de los intérpretes convencionales, por el simple hecho de que la arquitectura de la máquina AHR es tal, que resuelve por circuitos electrónicos tareas que en otras arquitecturas sólo se pueden atender por programas.

El intérprete de Lisp se encuentra repetido en todos los procesadores de Lisp; en módulos independientes que no necesitan interaccionar entre sí.

EL DISEÑO RECONFIGURABLE DE LA MAQUINA AHR

La máquina AHR se diseñó con dos familias de procesadores de Lisp: [ver informe AHR-76-1]:

a) en caso reconfigurable, cada procesador de Lisp ejecuta una función particular (v.gr. CONS), o un subconjunto pequeño de las funciones de Lisp. Claro que entre todos los procesadores debe cubrirse el espectro total de funciones de Lisp puro.

b) En el caso universal, cada procesador de Lisp ejecuta todas las funciones de Lisp puro.

Inclusive, la máquina AHR puede trabajar con una mezcla de procesadores tipo "particular" (caso 'a' o reconfigurable) y tipo "universal" (caso 'b').

¿Cómo sabemos qué funciones le son permitidas ejecutar a cada procesador de Lisp? Cada procesador tiene asociada una memoria RAME (ram de especialización), que contiene 256 palabras de 1 bit c/u; ese bit nos indica si la función está permitida o inválida. Hay hasta 256 funciones primitivas.

Sin embargo, para agilizar su construcción, las versiones 0 y 1 aquí descritas no son reconfigurables. Luego, no poseen RAME.

Cada procesador de Lisp tiene en su memoria privada una serie de funciones auxiliares de Lisp, y algunas rutinas de control para comunicación con el resto de la máquina. El código de todas las funciones primitivas de Lisp está estructurado como módulos independientes, de manera que no es necesario que radique todo el intérprete de Lisp en cada procesador de Lisp (aunque en las versiones 0 y 1, por no ser reconfigurables, sí es necesario). Para la versión 0 sólo se cuenta con cinco procesadores y es por esto que resulta conveniente que cada procesador contenga todas las funciones primitivas de Lisp. Puesto que los procesadores son lentos, se requiere que el software sea muy eficiente por lo que se decidió programarlo en lenguaje ensamblador.

El intérprete de Lisp para la Versión 0 es una versión "desconfiada", que se preocupa bastante por revisar que la información que maneja sea correcta; esto lo hará lento.

Para la Versión 1 se hará un intérprete "confiado" que supone que todo marcha bien y no verifica la validez de sus argumentos; esto se traduce en mayor rapidez. Para esta versión se implementarán funciones que permitan el uso de arreglos, otras funciones de Lisp que permitan más paralelismo y funciones especializadas localizadas sólo en algunos procesadores de Lisp.

Ciclo de los Procesadores de LISP

1. Petición de burocracia de entrada. (El procesador de Lisp está libre y avisa al distribuidor para que éste le reparta trabajo). Sigue libre mientras no se le asigne trabajo.
2. El distribuidor coloca en la memoria rápida (ram*) del procesador un nodo para procesar. Siempre será un nodo cuya función sea ejecutable por este procesador, pues el distribuidor consulta el RAME de cada procesador para conocer aquellas funciones que es capaz de ejecutar. (Ya se dijo que las Versiones 0 y 1 no poseen RAME, porque un

procesador de Lisp puede ejecutar cualquier función).

3. El procesador de Lisp evalúa el nodo y coloca el resultado en su memoria rápida (ram*). Dependiendo de la función a ejecutar puede ser necesario interaccionar repetidas veces con cualquiera de las tres memorias (parrilla, pasiva y de variables) para lectura y/o escritura de datos.

4. El procesador de Lisp hace una petición de burocracia de salida (avisa al distribuidor que ya terminó la evaluación). Una vez atendida regresa al paso 1.

Estado actual: actividades realizadas: diseño del intérprete.
Codificación de parte.

Diseño terminado. Codificación en un 30%.

Estado futuro: terminación de la codificación.
Depuración y pruebas exhaustivas.
Documentación para febrero de 1981.

Recolector de basura

Debido a que Lisp puro es un lenguaje aplicativo, no existen instrucciones que permitan modificar el valor de las variables. Es por esta razón que para alterar una variable, Lisp crea una copia de esta variable con su nuevo valor. Ello da oportunidad a otras funciones que todavía no hayan sido evaluadas y que vayan a requerir posteriormente el valor anterior de la variable, de consultar el valor original. Sin embargo, cuando una variable ya no va a ser utilizada por ninguna función, es importante liberar la memoria que ocupa. Si se permitiera que se acumularan variables que ya no son necesarias, llegaría un momento en que sería imposible procesar el programa por falta de memoria. Las variables que ya no son útiles reciben el nombre de basura.

En la ejecución normal de Lisp, y como resultado de la actividad de la función CONS, se crean asimismo estructuras (listas) que constituyen resultados parciales, pero que no aparecen como parte del resultado final. Estas listas también son basura: no son útiles.

Durante la ejecución de un programa, se sabe que un átomo o lista ya no es útil cuando no forma parte de ninguna lista; es decir, ninguna función o dato apunta a (contiene la dirección de) el átomo o lista. Esto implica que para localizar la basura de la memoria, es necesario averiguar las direcciones de todas las celdas que no son apuntadas por el programa que está siendo procesado, que es equivalente a consultar toda la memoria. Los sistemas de Lisp más comunes ejecutan un programa y dejan que se acumulen celdas no útiles en la memoria hasta que ésta se llena. Cuando la memoria se agota, se suspende el procesamiento del programa para recoger la basura y posteriormente proseguir con la ejecución. El recolector de basura en los intérpretes de Lisp convencionales es un programa que recorre varias veces la memoria a fin de determinar cuál celda está siendo utilizada todavía y cuál no. La labor del recolector de basura es por tanto una actividad muy prolongada, que requiere varios minutos, para llevarse a cabo en computadoras grandes que posean una cantidad razonable de memoria.

Ya que la recolección de basura interrumpe la ejecución del programa por un tiempo considerable, en un sistema que contiene varios procesadores, como es el caso de la máquina AHR, sería deseable que todos ellos recolectaran la basura simultáneamente a fin de que transcurriera el menor tiempo posible. Sin embargo, no se ha encontrado un algoritmo recolector de basura que funcione en paralelo y se adapte fácilmente a la arquitectura de la máquina AHR. Se eligió entonces un algoritmo convencional para recolectar, que es ejecutado solamente por uno de los procesadores. En principio cualquier procesador podría recolectar la basura, pero resultó más práctico que lo hiciera el procesador de entrada y salida, debido a que su ac-

ceso a las memorias compartidas es más ágil con respecto al resto de los procesadores. La recolección de basura se efectúa solamente en la memoria pasiva (que contiene listas) y en la región de números de la memoria de variables. En la zona de medios ambientes de la memoria de variables y en la parrilla no se realiza la recolección de basura debido a que no se genera basura: por la manera en que funcionan estas memorias, se conoce en qué momento se deja de usar una celda de memoria, y en ese mismo momento se coloca en la lista libre. Las listas (que residen en la memoria pasiva) y los números(que se almacenan en la memoria de variables), en cambio, sí producen basura.

La recolección de basura puede iniciarse ya sea porque se acabe la lista libre de la memoria pasiva, o porque se agote la zona de números de la memoria de variables. Los controladores (el de la memoria pasiva y el de la memoria de variables) poseen una señal que activan cuando se llenan sendas memorias. Estas señales llegan al procesador de entrada y salida. En el momento en que cualquiera de esta señales se activa, el procesador de entrada y salida suspende la ejecución del programa (en todos los procesadores de Lisp) y recolecta la basura. La recolección de basura en cada una de las dos memorias no es independiente; ambas se realizan en forma entrelazada. Basta con que una de las dos memorias se llene para que la basura se recolecte en ambas.

En la memoria pasiva cada celda posee un bit reservado para el recolector de basura. Tan pronto se inicia la recolección, el procesador de entrada y salida apaga este bit en todas las celdas. A continuación el procesador de entrada y salida recorre el programa y las listas que contiene; a cada celda que forma parte del programa le prende el bit. Cuando encuentra un número en el programa (que está en la memoria de variables), lo copia a una región de la memoria de variables que esté libre. El recolector de basura copia los números útiles en vez de marcarlos con objeto de que una vez terminada la recolección, la memoria libre no sea una lista (como en la memoria

pasiva), sino una zona contigua de memoria. El algoritmo empleado en la memoria de variables es utilizado por algunos sistemas de Lisp y en ocasiones por el compilador del lenguaje Simula, y se denomina algoritmo compactificador. Una vez que se han marcado las listas y los números útiles, el recolector procede a recorrer una vez más la memoria para recolectar la basura propiamente: forma un lista con las celdas que no estén siendo usadas, que es la lista libre.

En un futuro próximo se elaborará un recolector de basura en paralelo (donde varios procesadores recogen la basura), basado probablemente en el algoritmo de Lamport [33].

Estado actual: actividades realizadas: diseño del programa,
codificación,
prueba.

Características: recolector serial en memoria pasiva y variables; en pasiva liga celdas. En memoria de variables compacta.

Avance: estado actual: operativo en memoria pasiva.

Estado futuro: probar en memoria de variables y documentar, diciembre de 1980.

Programas del procesador de entrada y salida

Las labores de los programas que residen en el procesador de entrada y salida son convertir un programa escrito en Lisp (entrada) a una representación adecuada para que pueda ejecutarse, y copiarlo a la memoria pasiva; así como tomar un resultado de la memoria pasiva y transformarlo a una notación legible, que es el efecto inverso (salida).

Los programas de entrada y salida están escritos en un lenguaje de alto nivel denominado PLZ. Pueden leer o imprimir átomos, listas, números enteros y números de punto flotante. La transferencia de información se efectúa a través de la ventana, con la que el procesador de entrada y salida puede accesar la memoria pasiva o la memoria de variables tal y como si fueran su

propia memoria. El procesador de entrada y salida puede accesar las memorias compartidas sin necesidad de hacer peticiones a los árbitros por cada acceso (como sucede con el resto de los procesadores) de manera que el procesador de entrada y salida se desconecta por iniciativa propia en lugar de que tenga que hacerlo cuando el controlador finaliza de transferir datos a ram*. Es decir, el procesador de e/s hace un acceso al árbitro cuando desea entrar a alguna memoria (pasiva, variables), y la conexión termina por voluntad del procesador de e/s.

Estado actual: actividades realizadas: diseño, codificación.
Características: transforma notación de paréntesis a celdas y viceversa. Para un usuario.
Avance: estado actual: operativo para el manejo de listas, átomos y números enteros.
Estado futuro: probar su operación para números reales. Documentación, diciembre de 1980.

Distribuidor Versión 0

En la Versión 0, las funciones del distribuidor las realiza una microcomputadora programada (en lenguaje de máquina Z80) para tal efecto.

El distribuidor ejecuta los siguientes modos:

* burocracia de entrada.
* burocracia de salida.
* creación de nodos.

DISTRIBUIDOR VERSION CERO

El microprocesador ZDS 1/25 que se muestra en la foto se modificó para servir como distribuidor de la Versión 0 de la Computadora AHR. La pantalla muestra un vaciado de la parrilla.

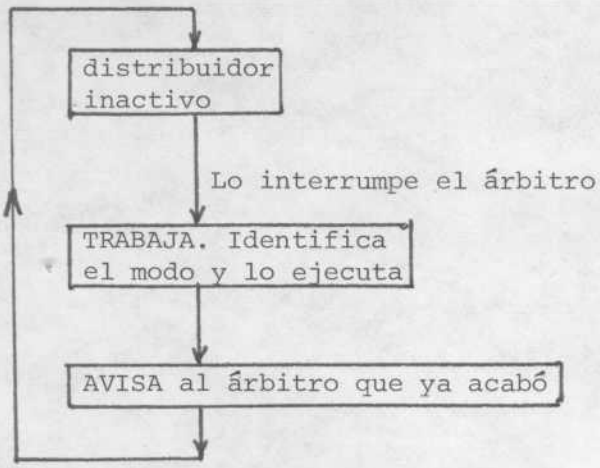
* lectura aleatoria.
* escritura aleatoria.

Además, opera con el procesador de entrada y salida de estas formas:

* crea un nodo maestro en la parrilla.
* suspende la atención a la burocracia de entrada, durante la recolección de basura (suspende la ejecución de los procesadores de Lisp).
* lanza el nodo maestro.
* continúa con el procesamiento después de una recolección de basura.

El distribuidor nunca opera por iniciativa propia, siempre espera a que el árbitro solicite su atención.

Cuando el árbitro lo activa, identifica el modo de la caja que pidió acceso (el distribuidor no sabe qué caja es), ejecuta ese modo y vuelve a dormirse. El diagrama es el siguiente:



Estado actual: actividades realizadas: diseño del programa,
codificación,
pruebas.

Características: permite la comunicación entre procesadores y distribuidor, el manejo del FIFO (pizarrón) y de la parrilla.
Avance: estado actual: operativo.
Estado futuro: documentación noviembre de 1980.

Distribuidor Interactivo para Depuración

Este programa realiza las mismas funciones que el distribuidor de la Versión 0, pero de una manera interactiva con el usuario, a fin de permitirle depurar sus programas. Usa la máquina ZDS 1/25.

El objetivo de este distribuidor interactivo es dar facilidades y herramientas para que el diseñador pueda accesar todos los recursos del distribuidor, FIFO, parrilla, y también la memoria ram* de los procesadores de Lisp. Esto es con el objeto de depurar errores de programación y de circuitería durante el desarrollo de la máquina AHR.

Depuración de errores de circuitería: el distribuidor interactivo juega un papel importante para el desarrollo de los circuitos porque va a permitir probar los procesadores de Lisp y el árbitro desde el distribuidor, a través de un teclado.

Depuración de errores de programación: permite depurar un programa, seguirlo paso a paso, efectuar correcciones y cambios dinámicos durante la ejecución de un programa.

Las funciones del distribuidor interactivo son ocho:

HARDWARE = Permite manejar individual e independientemente las líneas de control de los acopladores de los procesadores de Lisp y del árbitro.

= Permite escribir y leer datos en ram*, que consta de 16 pal de 32 b.

= Ejecuta ciclos de lectura de ram*, controlables en su número de iteraciones.

NODO = Le permite al usuario crear un nodo desde una terminal, extrayendo su dirección de la lista de nodos libres.

= Muestra cualquier nodo cuya dirección se especifique, verificando primero que esta dirección efectivamente corresponda a un nodo.

= Puede matar a un nodo (regresarlo a la lista de nodos libres).

= Lista n nodos libres (sus apuntadores) de la lista de nodos libres.

= Da el tamaño de la lista de nodos libres.

FIFO = Permite insertar en el FIFO un apuntador a un nodo.

= Permite extraer apuntadores del FIFO.

= Muestra el tamaño y contenido del FIFO

= Muestra los extremos (entrada y salida) del FIFO.

DISPLAY = Permite automodificarse cambiando programas y datos.

= Muestra el contenido de posiciones de memoria.

REGISTRO = Permite examinar y cambiar el contenido de los registros internos del distribuidor.

INICIALIZAR = Lanza el nodo maestro que arranca un programa en la máquina AHR.

= Crea y muestra el nodo maestro.

ARBITRAJE = Permite comunicarse con el árbitro.

= A petición del árbitro, ordena la conexión de un procesador de Lisp al canal de parrilla y FIFO.

= Cuando un procesador de Lisp termina su actividad, lo suelta.

EJECUCION = Pasa el control al programa "Distribuidor Versión 0" el que ejecuta las funciones del distribuidor sin intervención del usuario. Así es como opera

```
LINEAS  DE  ENTRADA
MODO     TB       ACNOMA   BARBI    PB       LIMPIO
 F        1         1        1        1        1

LINEAS  DE  SALIDA
DIA      MEA      WEA      NOBUEN   BCONT    IRB      DIR RAM*
 1        1        1        1        1        1        F

.H *E

POSIC. DE RAM*    VALOR      NUEVO

        00        FFFFFFFF   1234
        01        FFFFFFFF   AAAAAAAA
        02        FFFFFFFF   A6
        03        FFFFFFFF   1
        04        FFFFFFFF   9A7B3
        05        FFFFFFFF   27C5
        06        FFFFFFFF   22222555
        07        FFFFFFFF   5
        08        FFFFFFFF   A2345
        09        FFFFFFFF   19A5
        10        FFFFFFFF   BC5F2
        11        FFFFFFFF
```

```
DIREC. ASIGNADA = 7C00

NANE = 1
NUM. DE HIJO = 2
BANDERAS = 3
APUNT. AL MEDIO AMBIENTE = 09A74
TIPO = 15
APUNT. AL CODIGO = 7C140
NUM. DE TAREA = 005
APUNT. AL PADRE = B592A
ARGUMENTO 1 = 00008273
ARGUMENTO 2 = 00091BD7
ARGUMENTO 3 = 00000012
ARGUMENTO 4 = 00000000
```

```
POSICIONES DE MEMORIA
6039  21
603A  60
603B  5B
603C  06
603D  10
603E  BE
603F  20  66
6040  02  15
6041  36
6042  FF
6043  23
6044  10
6045  F8
6046  AF
```

SALIDAS DEL DISTRIBUIDOR
INTERACTIVO

más rápido la versión 0. No regresa al "Distribuidor interactivo para depuración."

= Idéntico a lo anterior, pero mostrando al usuario el número de nodos que se van ejecutando.

= Permite programar el número de nodos a ejecutar; luego, cede control al usuario.

= Ejecuta paso a paso.

La Comunicación Técnica AHR-80-9 de Luis Hugo Peñarrieta describe en detalle el funcionamiento del distribuidor y la detección de errores con la ayuda del mismo.

Estado actual del Distr. Versión 0 y del Distr. Interactivo:

- 100% programado, compilado y corre.
- Falta probar los comandos de ejecución y arbitraje, los que se podrán verificar cuando se integren los procesadores de Lisp con el árbitro.
- A la fecha se conectó un procesador de Lisp, el árbitro y el sistema que emula el distribuidor. Se probaron con la opción HARDWARE, habiéndose encontrado satisfactorios.

En la siguiente hoja se muestra un diagrama de flujo del distribuidor interactivo.

Microprogramación de los controladores

Los controladores de la memoria pasiva y de la memoria de variables son "inteligentes", en cuanto a que responden a varios modos de acceso a esas memorias. Estos modos son decodificados y ejecutados por un grupo de PALs (arreglos lógicos programables), que están microcodificados para este fin.

El controlador puede verse como una caja que recibe un número de 4 bits (el "modo") y responde a cada modo de entrada con cierto número secuencial de señales de 11 bits.

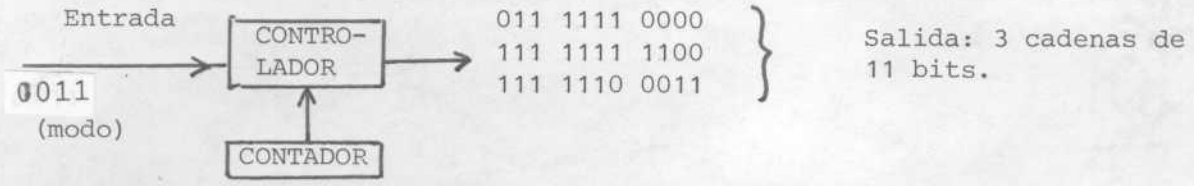


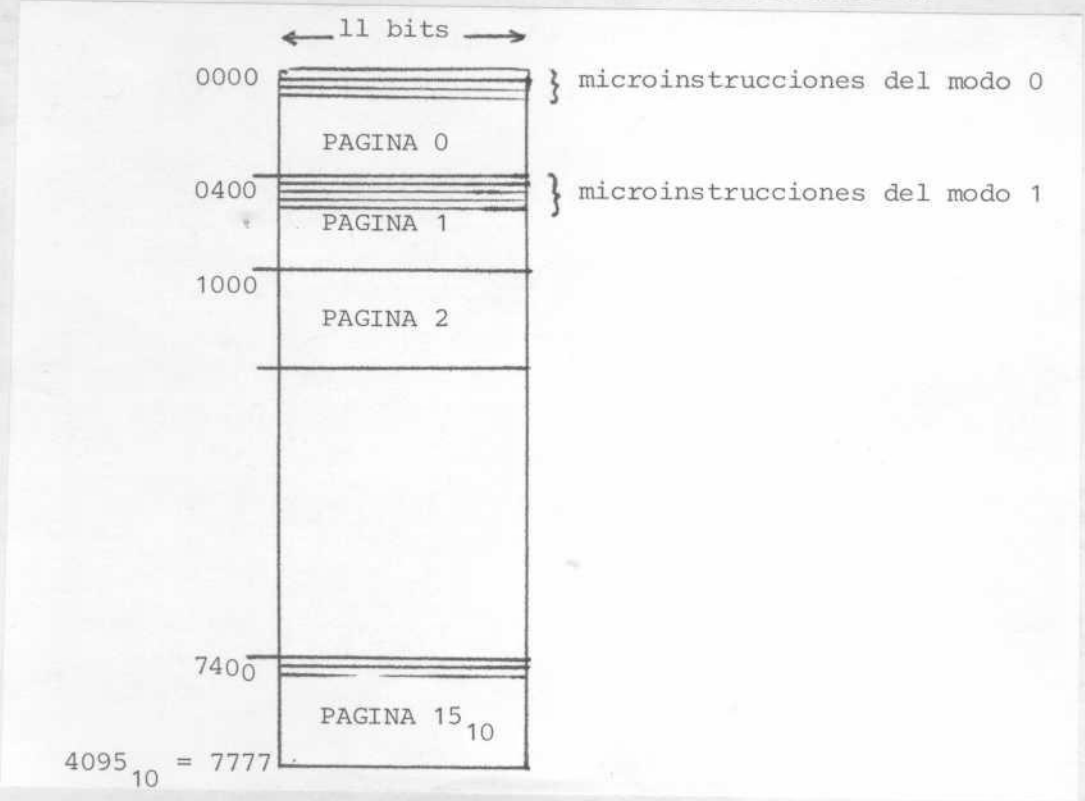
FIGURA "EL CONTROLADOR DE LAS MEMORIAS"

El modo 0011 excita al controlador, el que contesta con tres microinstrucciones (niveles en 11 líneas) que son las que, alambradas a las compuertas correspondientes, ejecutarán al modo 0011.

Cada cadena de 11 bits se usa como una microinstrucción para abrir y cerrar compuertas y efectuar operaciones digitales. En el ejemplo de la figura, la primera salida es 011 1111 0000, que irá a efectuar operaciones según la lógica del alambrado. Esta salida es seguida de otras dos. Este modo tuvo tres salidas.

No todos los modos producen el mismo número de salidas.

La decodificación de un modo puede verse como el acceso a palabras sucesivas de 11 bits en un espacio de direccionamiento de $2^{12} = 4096$. Esto se aclara a continuación.



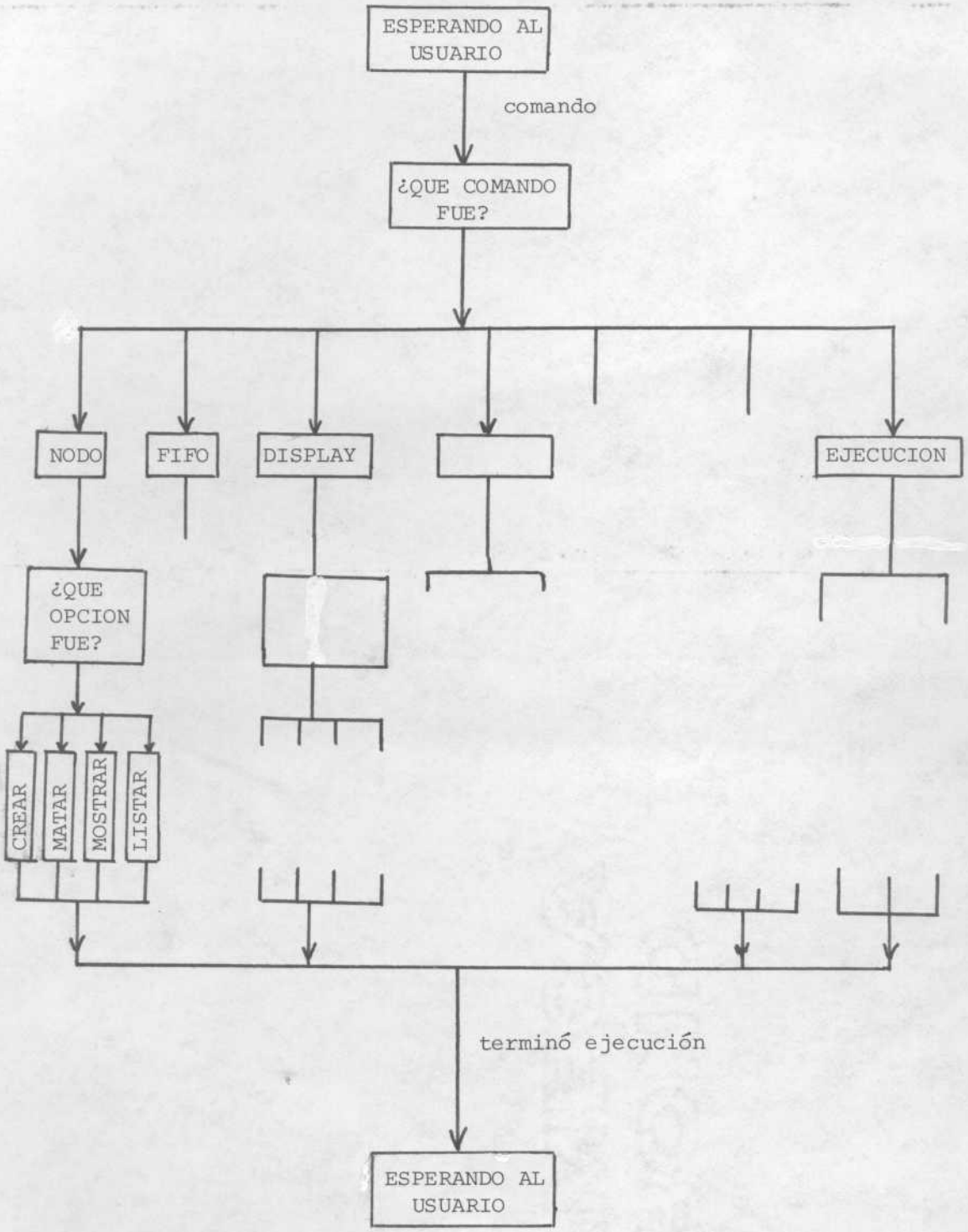


DIAGRAMA DE EJECUCION DEL DISTRIBUIDOR INTERACTIVO PARA DEPURACION

Cuando llega un modo (4 bits), se forma una dirección de 12 bits anexándole a su derecha ocho bits ceros:

0011 0000 0000
modo
dirección inicial

Esta dirección se refiere a un espacio de direccionamiento de 4096 palabras. Se accesa esa palabra (su contenido son 11 b) y se saca por las líneas de salida. Aquí sale la primera respuesta del PAL. Luego, se incrementa en 1 la dirección,

0011 0000 0001,

se accesa otra vez esa dirección, y su contenido (otros 11 b) se saca por las líneas de salida.

Se incrementa otra vez la dirección, etc.

El controlador continúa leyendo las direcciones del PAL, hasta que un contenido especial, que significa "ALTO", aparezca.

¿Para qué sirven estos contenidos? En cada conjunto de 11 señales (ciclo de PAL) se van a efectuar once operaciones (microinstrucciones) distintas, según a donde inhiban o exciten las señales. En nuestro ejemplo, fueron necesarios tres conjuntos de 11 excitaciones para acabar de ejecutar el modo 0011.

¿Porqué PAL's en vez de EPROM's? No se usaron memorias para depositar allí las microinstrucciones y luego accesarlas. Se usaron PAL's. Hay tres razones:

1. La rapidez de PAL es de aprox. 30 ns, versus 470 ns de una memoria EPROM.
2. No se desperdicia memoria que no se ocupa. Aunque el espacio de direccionamiento de PAL es de 4096 posiciones, ¡no todas existen!
3. No habíamos trabajado con PAL's, y decidimos aprender.

Modos del controlador de memoria de variables. A continuación se listan los modos de funcionamiento del controlador, con una breve explicación de cada uno.

MODO	CODIGO	DESCRIPCION
MI	0000	LECTURA ALEATORIA. Lectura de una palabra. En este modo, el controlador accesa la localidad 0 de la memoria ram* (que contiene la dirección de memoria de variables en donde se encuentra almacenado el dato que se requiere) y el contenido de esta localidad es depositado en su registro de direcciones. Después, el controlador accesa la localidad de la memoria de variables a la que apunta dicho registro de direcciones y transfiere su contenido hacia la localidad 1 de la memoria ram*.
M2	0001	ESCRITURA ALEATORIA. Escritura de una palabra. En este modo el controlador accesa la localidad 0 de la mem. ram* (que contiene la direcc. de mem. de var. en donde se desea escribir una pal.) y el contenido de esta localidad se deposita en su registro de direcciones. Después el controlador accesa la localidad siguiente de memoria ram* y transfiere su contenido a la localidad de memoria de variables apuntada por el registro de direcciones.
M3	0010	ENSARTAR NODOS. Ligar una celda o nodos (5 palabras de 32 bits en variables; 2 palabras de 22 bits en pasiva) a la lista de nodos libres. En este modo, el controlador lee el contenido de la localidad 0 de la memoria ram* y lo deposita en su registro de direcciones. Depués, el controlador escribe el contenido de su registro cabeza de lista (HIR) en la localidad de memoria de variables apuntada por el registro de direcciones. Después, el contenido de este último es copiado en el registro cabeza de lista.
M4	0011	DESENSARTAR NODOS. Liberar una celda o nodo de la lista de nodos libres. En este modo, el controlador escribe el contenido de su registro cabeza de lista en la localidad 0 de ram* y también en su registro de direcciones, para después, transferir la información contenida en la localidad de la memoria de variables apuntada por el registro de direcciones, hacia el registro cabeza de lista.
M5	0100	LECTURA MULTIPLE. Lectura de 5 palabras contiguas en la memoria de variables o 2 palabras contiguas en la pasiva. En este modo, el controlador lee el contenido de la dirección 0 de memoria ram* (que representa la dirección de memoria de variables en donde se empieza a leer) y el contenido de dicha localidad se deposita en su registro de direcciones; después, el controlador efectúa 5 lecturas consecutivas de la memoria de variables a partir de esa dirección y transfiere los datos a cinco localidades contiguas de ram* a partir de la localidad 1.

M6 0101 ESCRITURA MULTIPLE. Escritura de 5 palabras contiguas en la de variables o 2 palabras contiguas en la memoria pasiva. En este modo el controlador lee el contenido de la dirección 0 de la memoria ram* (que representa la dirección de memoria de variables en donde se emplieza a escribir) y el contenido de dicha localidad es depositado en su registro de direcciones. Después el controlador efectúa 5 lecturas consecutivas de la memoria ram* a partir de la localidad 1, y transfiere estos datos a cinco localidades contiguas de la memoria de variables a partir de la localidad apuntada por el registro de direcciones.

M7 0110 VENTANA DEL PROCESADOR E/S. En este modo, el controlador efectúa un intercambio de sus canales de direcciones, datos y control, con los canales respectivos provenientes del procesador de entrada y salida.

En este modo, las lecturas y/o las escrituras se efectúan por bytes o sea palabras de 8 bits, para lo cual el controlador provee el ruteo adecuado de la información. Por otra parte, el procesador de e/s envía sus direcciones a través del canal de datos de AHR , de la misma forma, el controlador provee el ruteo adecuado, tomando en cuenta además que los dos bits menos significativos de esta dirección proveen el sector de la palabra de 32 bits que se va a accesar mientras que los 19 siguientes representan la dirección de la localidad accesada por la ventana del procesador de e/s.

M8 0111 RECOLECCION 11. Este modo se prevee para uso posterior y aquí el controlador sólo realiza accesos de lectura-escritura en memoria de variables hacia la memoria ram*, pero forzando la escritura del bit menos significativo a un estado bajo.

Estado actual: actividades realizadas: diagramas de flujo, diseño de microcódigo.

Características: mediante la ejecución de los microprogramas se ordenan las transferencias de información entre las memorias pasiva y de variables a los respectivos buses, que conectan a los procesadores de Lisp.

Avance: estado actual: código definido.
Estado futuro: programación de los PAL's con el código, enero de 1981.

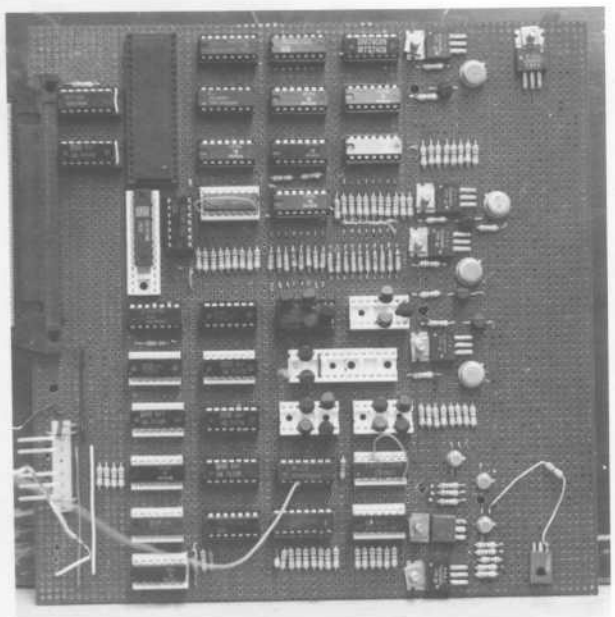
Programador de PAL's

Se tuvo que diseñar un dispositivo, que no forma parte de la máquina AHR, necesario para microprogramar los PAL's.

Un PAL tiene todas sus posible conexiones (fusibles) vírgenes, y el usuario quema varias de ellas, según el mapeo que quiera efectuar. El programador de PALs deduce la secuencia de quemado (a partir de tablas de entrada) y conduce a un autómata que produce una secuencia de voltajes y corrientes necesarias para fundir el fusible escogido.

Estado actual: diseño lógico. Armado con alambre enrollado. Operativo 95%.

Estado futuro: conexión a ZDS 1/25, prueba del sistema. Nov. de 1980.



PROGRAMADOR DE PAL's

Aunque no forma parte de la Computadora AHR, el programador de PAL's hizo posible la microprogramación de los controladores y facilitará la manufactura del distribuidor de la Versión 1 de la Computadora AHR, en caso de que éste use PAL's.

Iniciación del Sistema

En la máquina AHR existen ciertas variables que deben tener un valor determinado antes de que se empiece a evaluar un programa. El procesador de entrada y salida inicia los valores de estas variables.

Las principales variables que deben prepararse al iniciar un programa son: la lista de celdas libres en la memoria pasiva, indicadores del tamaño de la zona de números en la memoria de variables (que está vacía), así como el diccionario de átomos y funciones primitivas del intérprete. Además debe transmitirse el intérprete a todos los procesadores, ya que éste se borra al apagar el sistema pues se almacena en memoria volátil.

También hay que llenar el RAME de cada uno de los procesadores de Lisp, para indicar sus especialidades.(Como las versiones 0 y 1 de la máquina no poseen RAME,esto no es necesario en ellas).

Comunicación a través del canal lento

El procesador de entrada y salida tiene la capacidad de comunicarse con los procesadores de Lisp a través del canal lento. Este canal transmite información a baja velocidad pues no se utiliza con frecuencia.

El canal lento se emplea para enviar el código del intérprete de Lisp a los procesadores al iniciar la ejecución de un programa, o para suspender la evaluación del programa, si el usuario decide que no debe continuar su ejecución. El procesador de entrada y salida se comunica con todos los procesadores simultáneamente en ambos casos (en modo broadcast).

En ocasiones, el procesador de entrada y salida se comunica solamente con un solo procesador en particular; como es el caso de la generación de errores durante la ejecución, o la extracción de estadísticas sobre el programa cuando finaliza su ejecución. En estos casos, el procesador de entrada y salida tiene la habilidad de dirigirse a un solo procesador a la vez.

La comunicación por el canal lento se realiza siempre por iniciativa del procesador de entrada y salida, de manera que a pesar de que no existe un sistema de arbitraje en este canal, no hay colisiones en él.

El canal lento es de 16 bits. Por 8 de ellos el procesador de e/s indica el número del procesador de Lisp con el que se comunica, y los otros 8 transmiten datos. Este formato puede no ser el mejor, pero se escogió por su simplicidad y baja burocracia en la transmisión/recepción de mensajes.

Manejo e impresión de errores

Los errores que se pueden producir durante la evaluación de un programa son de diversa índole. Cuando cualquiera de ellos se genera, se suspende la ejecución, pues implica deficiencias en el programa. Los errores más importantes son: acceso a memoria inexistente, error de paridad en las memorias pasiva y de variables, división por cero, argumentos inadecuados para una función y sobreflujo en la parrilla. La mayoría de ellos son atendidos por el procesador de entrada y salida. Al producirse un error, el procesador pertinente envía un mensaje por el FIFO[2] indicando al procesador de entrada y salida la naturaleza del error; éste puede entonces interrumpir el programa.

AVANCES TECNICOS LOGRADOS Y GENERACION DE NUEVOS CONOCIMIENTOS

Avances en Programación

* Se ha empezado una teoría o punto de vista sobre máquinas heterárquicas. Ver publicaciones AHR-76-1 a AHR-80-13.
* Se han aclarado y generalizado varios puntos de vista sobre máquinas que computan sobre flujos de datos (data flow machines), tuberías (pipelines) y máquinas pra procesamiento en paralelo. Ver Apéndice I.
* Se entienden mejor los procesos secuenciales de lectura y escritura dentro del contexto de computación en paralelo.
* Se ha firmado un convenio entre el Laboratorio AHR del IIMAS y el Instituto de Ciencias de Control de la Unión Soviética, para estudiar efectos del paralelismo en la relación memoria-tiempo-número de procesadores.
* Se sabe, puesto que se ha hecho, cómo hacer un recolector de basura adaptado a un ambiente de multiprocesamiento.
* Se ha adquirido experiencia con los problemas de depuración de errores en un ambiente de procesamiento en paralelo. Ver publicación AHR-80-9.
* Se han diseñado programas reconfigurables. Ver publicación AHR-80-12.
* Se han diseñado y construido sistemas de información reconfigurables. Ver publicaciones AHR 80-8 y AHR-80-12

Avances en circuitos electrónicos

* Se tiene experiencia en la interconexión de varias microcomputadoras, y las diferentes señales que deben cursarse entre las mismas, a fin de sincronizarlas.
* Se conoce más de la estructura de los árbitros que administran recursos comunes.

* Se conocen algunas relaciones de optimización entre número de procesadores, memoria y tiempo. Ver reportes AHR-79-4 y AHR-79-5.

* Se diseñó una máquina heterárquica para procesar imágenes. Ver Apéndice II.

* Se posee una visión alterna de procesos equivalentes a programas Lisp. Ver reporte AHR-79-4. En este tema contribuyó el Prof. Kemer Norkin, visitante del Instituto de Ciencias de Control de la URSS.

* Se entienden mejor los problemas de la computación distribuida. Ver Apéndice III.

Formación de recursos humanos

Nivel: egresados de licenciatura.

- Entrenamiento en el diseño de sistemas digitales y en diseño de sistemas microprogramados (2 personas).

Nivel: en curso de estudios de Licenciatura

- Entrenamiento en diseño de circuitos impresos, así como en técnicas de prueba de tarjetas (2 personas).
- Entrenamiento en diseño de interfases para periféricos, construcción y prueba (1 persona).

Nivel: MAESTRIA

- Se perfeccionó el curso de Arquitectura de Computadoras, en la Maestría de Computación del IIMAS-CCH.
- Se generaron los cursos de
  - Lenguajes de Programación en Paralelo
  - Computación Distribuida

  para la Maestría en Computación del IIMAS-CCH y la de UPIICSA-IPN.
- Se recolectaron e imprimieron artículos sobre estructuras de computadoras digitales. Ver publicación AHR-80-11.

Nivel: LICENCIATURA.

- Se publicaron (Ver Comunicación AHR-80-7) una serie de proyectos y temas de tesis (licenciatura y maestría) en Computación.

Nivel: PROFESIONALES Y ESPECIALISTAS

- Miembros del Laboratorio AHR organizaron y presentaron ponencias en la "Conferencia sobre Microprocesadores y Microcomputadoras 1980". Esto ayudó a difundir nuestro proyecto.

Avances en Paralelismo

* Se creó, desarrolló y encontró útil el concepto de heterarquía.
* Se conoce cómo ejecutar un programa de propósito general en paralelo, sin instrucciones ni comandos específicos de parte del programador.
* Se sabe cómo administrar, coordinar, controlar y usar los recursos necesarios para llevar a cabo un cómputo por varias computadoras simultáneamente.
* Se tiene una herramienta poderosa para la experimentación con lenguajes y sistemas de programación en paralelo.
* Se ha formado un grupo de personas capaces de proseguir investigaciones de frontera en procesamiento distribuido, procesamiento en paralelo, bases de datos por hardware, etc.

C O N C L U S I O N E S Y R E C O M E N D A C I O N E S

La manufactura de la Computadora AHR muestra que es posible procesar programas de propósito general (que no han sido escritos explícitamente para un ambiente de paralelismo) en un multiprocesador con bastante simultaneidad.

Se ha logrado un entendimiento teórico y práctico de importantes efectos en Computación; se conoce cómo administrar, coordinar, controlar y supervisar la ejecución de varias corrientes de instrucciones (instruction streams), con un control hecho en su mayoría por circuitos.

Se ha aprendido cómo construir computadoras orientadas a lenguaje de máquina de alto nivel.

El esfuerzo del Proyecto AHR ha hecho posible que el IIMAS-UNAM y la comunidad académica cuenten con una computadora de tamaño mediano, de propósitos generales, diseñada y construida en México y por técnicos nacionales.

La creación de la Computadora AHR fue posible gracias a que se desarrolló en el Laboratorio AHR,con ayuda económica de CONACYT e IIMAS y en el apoyo entusiasta de los integrantes del Proyecto AHR. La infraestructura lograda en este laboratorio, en cuanto a aparatos, componentes electrónicos, experiencia y recursos humanos hacen que el Laboratorio AHR constituya un recurso valioso para la acometida de nuevos proyectos de investigación en Computación, Comunicaciones y Electrónica.

El conjunto de elementos humanos del Laboratorio AHR (ver Apéndice IV) constituye un grupo especializado y valioso que continuará emprendiendo investigaciones de alta calidad y elevada originalidad, donde los estudiantes asimilen conocimientos en el devenir de su trabajo cotidiano.

Recomendaciones

1. El fortalecimiento del Laboratorio AHR, mediante la adquisición de equipo adicional, el ingreso de nuevos elementos humanos y la continuación del programa de estudiantes becarios.

2. Continuar con la Versión 0 de la máquina AHR, hasta su terminación.

3. Emprender la construcción de la Versión 1 de la Máquina AHR.

4. Crear un grupo de trabajo ad-hoc para estudiar la factibilidad de desarrollar investigaciones en máquinas para bases de datos y bases de datos y computación distribuida (Ver Apéndice III).

5. Difundir las actividades y logros del Proyecto AHR.

6. Poner a disposición de la UNAM y del público programador la máquina AHR.

7. Continuar investigaciones en programas y sistemas para procesamiento en paralelo (nuevos lenguajes).

8. Incorporar al programa de Maestría de Computación del IIMAS-CCH los siguientes cursos:
   a. Diseño de arquitecturas de cómputo en paralelo.
   b. Lenguajes de programación en paralelo.
   c. Máquinas de bases de datos.
   d. Computación distribuida.

   Los cursos (a), (c) y (d) pueden ofrecerse como curso-proyecto, con una gran parte del aprendizaje efectuándose a través del diseño y construcción de componentes específicas en el Laboratorio AHR, guiados por profesores del Laboratorio.

9. En general, incorporar a la Maestría de Computación del IIMAS-CCH, ciertas actividades "creativas" y de diseño del Proyecto AHR.

E L F U T U R O D E L A M A Q U I N A A H R

* Más rápida. La versión 1 será mucho más rápida que la Versión 0, al substituirse el distribuidor.

* Más completa. La Versión 1 tendrá más memoria, más procesadores de Lisp y otras mejoras que la experiencia señalará. Ver tabla 'Especificaciones'.

* Más utilizada. Se hará accesible al público programador del IIMAS y de la UNAM. Con esto esperamos un buen número de programas no triviales que la usen y pongan a prueba sus límites de diseño, bajo la ejecución "pesada" de problemas fuertes, que demanden muchos recursos en memoria, tiempo de ejecución y complejidad de los algoritmos.

  Esto dará una buena experiencia del desempeño de la máquina en un ambiente de uso sistemático.

* Mejoras a la máquina. El uso fuerte de la máquina pondrá al descubierto lugares débiles, cambios y diseños alternos que la hagan realizar mejor, más rápido, con menos memoria, o con más elegancia, sus funciones y trabajos. Estas mejoras vendrán poco a poco, como fruto de la experiencia en su uso; algunas mejoras se echarán a andar, mediante la modificación de la arquitectura de la máquina en sus circuitos y/o programas.

* Su uso en la enseñanza. Las Licenciaturas y Maestría en Computación contarán con una computadora en paralelo, es decir, con un multiprocesador, donde los alumnos podrán hacer ejercicios, cambios al diseño, simulaciones, mediciones y obtener experiencias en varios campos de la

Computación.

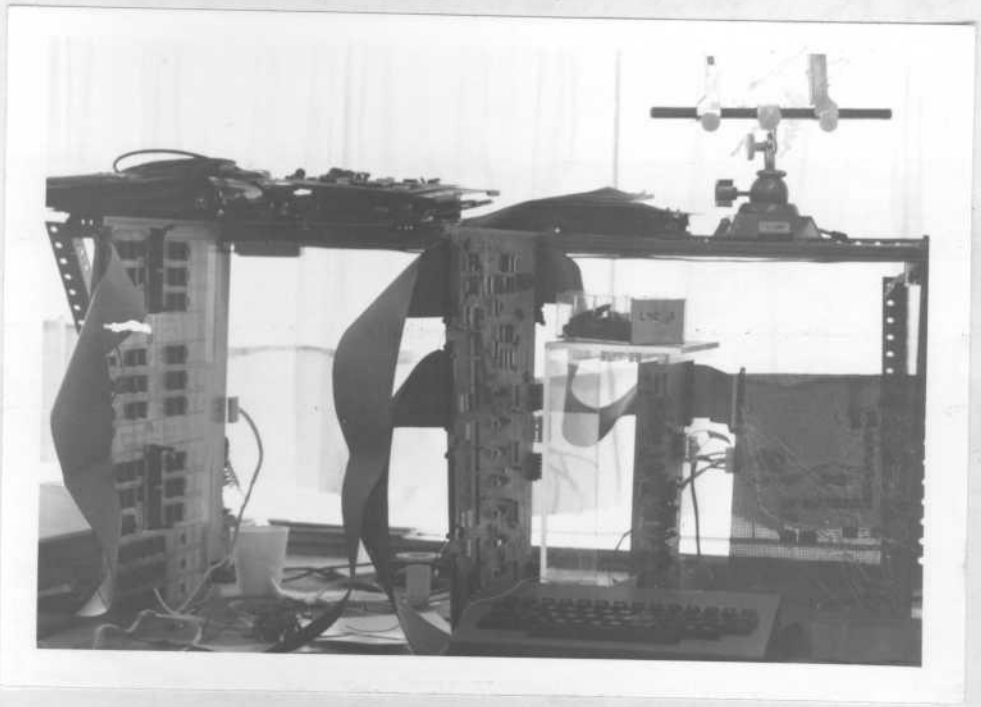
Los siguientes cursos podrán tener ejercicios prácticos en la máquina AHR:

CURSOS DE LA LICENCIATURA EN COMPUTACION	CURSOS DE LA MAESTRIA EN COMPUTACION
Arquitectura de Computadoras.	Arquitectura de Computadoras.
Simulación.	Simulación.
	Lenguajes de Programación.
Diseño Lógico.	Diseño Lógico.
	Lenguajes de Programación en Paralelo.

* Su uso en Investigación. La máquina AHR abre perspectivas reales de hacer investigación relevante, original y altamente creativa en los siguientes campos:
  - Lenguajes y sistemas de programación en paralelo
  - Heterarquía
  - Sistemas de Actores
  - Computación distribuida
  - Control y comandos distribuidos

  Estas áreas son todas temas actuales en Computación, pero su avance se dificultaba por la carencia de un Laboratorio y/o una computadora que fuese la herramienta adecuada para que el investigador pudiese experimentar, medir y ver, además de las labores teóricas de pensar, calcular, dibujar y simular.

* Comercialización. Aunque es temprano para buscarle una salida comercial (hacer que se vendan muchas de estas máquinas), en colaboración con algunos industriales del ramo, se puede crear un grupo ad-hoc que investigue esta posibilidad.



EL CANAL RAPIDO

Detalle de interconexión entre varias tarjetas de la computadora AHR, Versión 0.

APENDICE I. TRABAJOS PREVIOS RELACIONADOS

General

Hay mucha actividad en el campo de la computación en paralelo. Algunos libros sobre el tema son [5,15 y 36].

La mayoría de los proveedores de equipo de cómputo ofrecen máquinas con algún tipo de paralelismo[18,27,31,45, 49,50].

Otra forma de desarrollar computación en paralelo es a través de redes de computadoras [1,24,48,53] y de computación distribuida [Ver Apéndice III].

Ha habido un número creciente de reuniones y conferencias sobre paralelismo [16,35,43,44,46].

Máquinas para el cálculo lambda

En 1971, un artículo [7] formaliza la manera de evaluar expresiones lambda en una máquina. Por supuesto, el artículo de McCarthy [29] es un clásico en el tema.

Máquinas con lenguajes de máquina de alto nivel

La idea de usar un lenguaje de alto nivel como lenguaje de máquina no es nueva. Se ha propuesto [4] una máquina de Fortran. Hewlett-Packard provee una máquina de Basic. La B5500 y la B6700 [22] están inspiradas en Algol. La máquina PERQ [42] procesa Pascal como lenguaje de máquina. A continuación siguen las máquinas de Lisp.

Máquinas de Lisp

Peter Deutch [13] presenta una arquitectura especialmente apropiada para Lisp. Ver también [2 y 52]. Todas éstas

son monoprocesadores de Lisp.

Keller et al [32] presentan una máquina muy parecida a la nuestra en cuanto a que ejecuta una especie de Lisp puro en paralelo.

[8] presenta una revisión de máquinas para Inteligencia Artificial, incluyendo máquinas de Lisp.

En el lado de la programación, [33] se presenta un algoritmo para recolectar basura en paralelo e incrementalmente.

Tuberías (pipelines)

La idea de muchas unidades que juntamente transforman un conjunto de datos se usa en arquitecturas de tuberías [31,45]. Podemos pensar que la tubería es un tipo especial de nuestra parrilla, donde el flujo de los resultados parciales sigue trayectorias rígidas.

Máquinas con etiquetas (tag machines)

La B6700 [22] ya posee bits que se usan sólo para etiquetar, tal como nuestra máquina AHR. Feustel [17] describe y generaliza este concepto.

Computación asíncrona

Patil [41] estudió la evaluación asíncrona de expresiones lambda. Esta línea de trabajo continúa en M.I.T. Rumbaugh [47] diseñó una máquina altamente paralela para programas expresados en un lenguaje de flujo de datos. Posee programas tanto durmientes como activos; sus estructuras de datos son vectores de valores. Una estructura es compartida (en vez de copiada) por varias activaciones concurrentes.

Máquinas de flujo de datos

El modelo de gráficas de flujo por donde las ejecuciones

"caminan" abandona el concepto clásico de usar un contador de programa para ejecución secuencial. Ver [20,21,51]. Nuestra máquina AHR se parece a las máquinas de flujo de datos en que en ambos diseños la disponibilidad de los datos dicta las siguientes operaciones (funciones, en Lisp) a ser ejecutadas, y es posible obtener un paralelismo fuerte.

Módulos para transferencia de registros

Bell [6] postula cajas que pueden fácilmente conectarse y reconectarse para desempeñar computaciones arbitrarias.

Arquitecturas que se parecen a la nuestra

Miller [39,40] describe una máquina reconfigurable basada en un buscador que alimenta las unidades operacionales con nuevos trabajos que realizar. Este buscador se parece al distribuidor nuestro, excepto en que nuestro distribuidor no hace búsquedas, sino que encuentra en el FIFO los nodos ya listos para ser evaluados. Su máquina no es reconfigurable en el sentido nuestro. El usa una red de n por n interconexiones para obtener reconfiguración.

Glushkov [19] presenta una arquitectura recursiva de computadora que es similar a la nuestra en el sentido de que todos los elementos de programa para los cuales están disponibles los operandos entrarán a unas cajas (similares a nuestros procesadores de Lisp) para ser ejecutados. Tal como en nuestra arquitectura, la suya permite la remoción de programas que manejan interrupciones.

Los actores de Hewitt [25] forman un formalismo especialmente interesante alrededor de un concepto único fundamental: la computación mediante el pase de mensajes.

Si nosotros vemos a los nodos (que el distribuidor de

AHR expele a los procesadores de Lisp) como mensajes "a cualquier procesador de Lisp", y los resultados como otros mensajes "a la parrilla", entonces la máquina AHR puede verse como una arquitectura que intercambia mensajes entre dos tipos de actores: los "procesadores de Lisp" (elementos activos) y la parrilla (memoria). Es interesante notar que no hay intercambio explícito de mensajes entre dos procesadores de Lisp.

Kautz [30] también coloca lógica en la memoria de su máquina.

Una teoría que es relevante a nuestros trabajos es la de Landin [34].

APENDICE II. RELACION DE LA MAQUINA AHR CON PROCESAMIENTO DE IMAGENES

El procesamiento de imágenes es un buen campo para comenzar aplicaciones de procesamiento en paralelo

El procesamiento en paralelo es un campo excitante, en parte porque el programador o el usuario pueden dar órdenes a muchas computadoras, para ser ejecutadas simultáneamente. El tiene muchos sirvientes al mismo tiempo. Cómo coordinar, sincronizar y prevenir puntos muertos entre todas estas máquinas no es fácil. La situación está bajo control en el caso de máquinas de una sola corriente de instrucciones y múltiples corrientes de datos (single-instruction multiple-data machines), por ejemplo la Illiac IV, donde un conjunto de órdenes (esto es, un programa) se ejecuta simultáneamente por muchos procesadores; en cualquier instante todos los procesadores ejecutan la misma instrucción, aunque sobre diferentes celdas de memoria.

Cuando el programador tiene que dar órdenes a máquinas con múltiples corrientes de instrucciones y de datos (multiple-instruction multiple-data machines), como la máquina AHR, "alguien" (lo más probable es que sea él mismo) tiene que poner atención en la ejecución correcta (en el tiempo) de cada programa. Este trabajo es necesario para poder sincronizar las acciones de cada procesador; así, por ejemplo, los argumentos de una función deberán estar listos (evaluados) antes de que comience el procesamiento (evaluación) de esa función.

Si el algoritmo a ser ejecutado es complejo (en el sentido de ser difícil a primera vista cómo dividirlo en n programas diferentes, uno para cada máquina) entonces la carga que se le impone al programador es pesada: esencialmente, él tiene que inventar un algoritmo paralelo desde el comienzo.

Pudiese ser que él tuviera en su bolsillo un programa serial (para mono-procesador), pero esto le daría poca ayuda.

¿Cómo podemos ayudar al programador a ir de un algoritmo serial a un algoritmo en paralelo? Nuestra posición es que, en el caso de procesamiento de imágenes, será más fácil que en otras áreas con tareas más estructuradas. Esto es así porque las imágenes no tienen "mucha" estructura: un procesador puede comenzar su análisis de la esquina inferior izquierda y avanzar bastante, antes de que necesite información de la esquina superior derecha. Esto es, mucha de la computación es "local" o limitada en diámetro (el resultado en un pixel depende solamente de una pequeña vecindad a su alrededor). Así, el programador puede todavía pensar serialmente casi todo el tiempo y obtener suficiente trabajo del multiprocesador MIMD.

Si mi posición es correcta, esperaremos ver avances en la programación en paralelo de máquinas MIMD aplicadas al análisis de imágenes, antes de que veamos estos avances en otros campos [35] .

La máquina heterárquica para analizar imágenes

En la publicación AHR-79-3, se describe una arquitectura de computadora, apta para procesamiento digital a altas velocidades de imágenes LANDSAT, y de otros tipos. Este es un diseño "de papel" solamente, y no se han hecho simulaciones para explorar la bondad de la arquitectura propuesta.

La máquina comprende varias (hasta 128) unidades activas o procesadores de imágenes (probablemente, microprocesadores), cada uno de ellos desarrollando un trabajo sobre una subimágen. Como en la máquina AHR, la coordinación entre estos procesadores de imágenes se maneja por un distribuidor, que contiene un FIFO o pizarrón. Este FIFO es una memoria que contiene direcciones de los trabajos listos para ser eje-

cutados, así como los destinos de los resultados de los cálculos ya terminados. El nombre "heterarquía" se usa porque no hay jerarquía entre los procesadores de imágenes.

Los principales componentes de la máquina son: una memoria para la imagen, donde reside la fotografía a ser procesada; los procesadores de imágenes, que realizan el cómputo; el distribuidor, que reparte trabajo que aún falta por hacerse; y un canal que conecta la máquina a una computadora de propósito general, haciéndola ver como un periférico de ésta. Nótese la similaridad con la computadora AHR.

APENDICE III. RELACION DE LA MAQUINA AHR CON PROCESAMIENTO DISTRIBUIDO

Los avances alcanzados en el diseño y construcción de la máquina AHR inducen a pensar en atacar otros dos problemas actuales, verdaderos temas de investigación en donde, para tener éxito hay que dominar la programación, el diseño lógico y tener un espíritu creativo no despegado del criterio del ingeniero en cuanto a optimización de soluciones viables. Me refiero al procesamiento distribuido y a los sistemas de información por hardware.

Los sistemas de procesamiento distribuido

Después de haber fabricado enormes computadoras que son usadas a través del tiempo compartido y la multiprogramación, las economías de la integración en gran escala inducen al hombre a pensar en multitud de máquinas modestas (microcomputadoras) que colaboran para desarrollar las mismas tareas que hasta ahora han venido desarrollando sus hermanas mayores.

Es la división y repartición del trabajo en múltiples unidades de procesamiento; es la intercomunicación, colaboración y coordinación de tales unidades; es el abaratamiento de la transmisión de datos, al solo transmitirse la información requerida; es el procesamiento local de la información local; éstos son los atributos principales de sistemas de procesamiento distribuido [Reporte AHR-79-6].

El denominador común de los sistemas distribuidos

El tema común que liga los diferentes tipos de sistemas distribuidos es el uso de múltiples elementos de cómputo que colaboran entre sí para realizar trabajos que hasta ahora habían sido desarrollados en grandes computadoras, o no habían sido desarrollados en ningún lado.

Los sistemas distribuidos se han desarrollado como alternativas a grandes instalaciones de cómputo porque, en muchas circunstancias, las instalaciones centrales no han sabido proveer soluciones óptimas o efectivas a problemas importantes.

Los grandes y rápidos cambios en la tecnología de semiconductores han tenido un impacto fuerte en los costos relativos de las diferentes partes de un sistema de cómputo que posee comunicaciones. Mientras que los costos de lógica y de almacenamiento interno han disminuido muy rápidamente, los costos de almacenamiento (memoria) masivo han disminuido menos rápidamente, en tanto que los costos de comunicación han permanecido prácticamente constantes. Además, la programación se ha convertido en un porcentaje creciente del costo total del sistema de cómputo. Ya no hay, pues, una penalidad económica severa en contra de distribuir la capacidad de procesamiento a las terminales individuales.

En sistemas de tiempo compartido y de multiacceso, hay varias razones sólidas funcionales para centralizar la base de datos, particularmente en aquellos casos donde varios usuarios pueden alterarla y accesarla. Sin embargo, no existen fuertes razones funcionales para centralizar las funciones mismas de procesamiento. La decisión de centralizar el procesamiento del usuario es una decisión económica. Con las tendencias actuales en los costos de equipo digital por una parte y los de comunicación por otra, es económicamente atractivo efectuar tanto como sea posible de cómputo y procesamiento localmente, para así minimizar las necesidades de transmisión de datos y por ende los costos de comunicación.

Los sistemas de información por circuitos

Dentro del área de programación, los productos más comercializados son los sistemas de manejo de información. A la fecha se ha desarrollado una teoría extensa sobre procedimientos para adquirir, recuperar, consultar y actualizar información en bancos de datos, cada vez mayores.

Los problemas tradicionales de acceso a esa información por lo general han sido tratados desarrollando algoritmos com-

plejos que los solucionaban parcialmente, sacrificando áreas de almacenamiento unas veces, o consumiendo mucho tiempo en otras. Problemas tales como:

- manejo de listas invertidas
- clasificación (sorting)
- empalme (merging)
- recuperación de información dependiente de parámetros internos

y muchos otros, que aún son un grave cuello de botella en las grandes bases de datos.

La idea de tratar de resolver varios de esos problemas desde el punto de vista de circuitos no es nueva. Sin embargo las soluciones propuestas y algunas desarrolladas no son del todo satisfactorias. El dominio del mercado en esa área lo sigue teniendo la programación.

Algunos ejemplos de estos sistemas son DBC de la Universidad de Ohio, la máquina de la Universidad de Braunschweig, Array Processors (STARAN) de la Universidad de Syracuse, DBCP de la Universidad Complutense de Madrid, SPIRIT de la Universidad de Kiev en Japón, CASSM de la Universidad de Florida, RARES de la Universidad de Utah, INFOPLEX de M.I.T., RAP de la Universidad de Toronto.

Algunos de estos sistemas exploran técnicas en bases de datos distribuidas como INFOPLEX, en cambio otras tienen su estructura centralizada como DBC, Braunschweig, etc.

En la actualidad el desarrollo tecnológico está demostrando que las grandes unidades de disk-pack están perdiendo bonos ante la nueva tecnología Winchester que permite gran densidad a bajo costo y a bajo consumo de potencia. Vale la pena explorar esta posibilidad teniendo en cuenta que los disk-packs no irán mucho más allá de lo que se tiene en la actualidad; en cambio, los discos de tecnología Winchester son muy nuevos y no es fácil predecir, en este momento, la tendencia en esa área.

Propuesta

De acuerdo con este bosquejo general se propone enfocar parte de los recursos del Laboratorio AHR hacia la búsqueda de elementos de programación y circuitos que, combinados, optimicen el manejo de Sistemas de Información.

Actividades

A. Evaluar en base a una aplicación tipo y a los avances tecnológicos la conveniencia de un sistema centralizado o distribuido.
B. Si la decisión es un sistema centralizado:
   Cambiar el acceso serial de un disk-pack a uno paralelo, es decir, obtener o almacenar información en todas sus cabezas simultáneamente.
   Si la decisión es un sistema distribuido:
   Sincronizar el acceso a varias unidades de disco simultáneamente y estudiar la manera de procesar la información lo más rápido posible.
C. En base a toda la experiencia anterior, estructurar un diseño funcional con lujo de detalle sobre una Computadora de Informática que vendrá a substituir los sistemas de Manejo de Información convencionales.
D. Efectuar una simulación que permita manejar los parámetros del sistema a voluntad.

Tiempo

A. Lo que resta de 1980.
B, C y D durante 1981.

APENDICE IV. INTEGRANTES DEL LABORATORIO AHR

Norma Apodaca de Rosenblueth. Estudia la carrera de Ingeniero en Computación, Fac. de Ingeniería, UNAM. Actualmente colabora en el Laboratorio AHR, IIMAS. Sus intereses profesionales: circuitos digitales, computación en paralelo.

Manuel Correa Síntora. Ingeniero Mecánico Electricista, Fac. de Ingeniería, UNAM, 1973-77. Tesis "Diseño de un generador programable de funciones", realizado con una microcomputadora. Presentada en la Spring Conference on Microcomputers, Philadelphia, March 1980. Actualmente colabora en el IIMAS, en el Laboratorio AHR. Intereses profesionales: arquitectura de computadoras.

Raúl Gómez Romero. Ingeniero Mecánico Electricista, Fac. de Ingeniería, UNAM, 1973-77. Tesis "Diseño de un generador programable de funciones", realizado con una microcomputadora. Presentada en la Spring Conference on Microcomputers, Philadelphia, March 1980. Ayudante de la Maestría en Computación, IIMAS. Actualmente colabora en el Laboratorio AHR, IIMAS. Intereses profesionales: microprogramación, electrónica digital, arquitectura de computadoras.

Dora Luz Gómez Sotomayor. Pasante de la carrera de Actuario, UNAM. Merecedora de la Medalla Gabino Barreda otorgada al mejor promedio de la Generación.

Forma parte del personal del IIMAS desde 1974 como miembro del Departamento de Computación. Su experiencia en el ramo de la Computación data de 1972 cuando trabajó con el M. en C. Manuel Alvarez en la elaboración de un sistema de información para la Biblioteca Central de Ciudad Universitaria.

En Mayo de 1978 es invitada a presentar su trabajo "Tessellation of triangles of variable precision as an economical

representation for digital terrain models" en el Symposium que sobre Modelos Digitales del Terreno, organiza la American Society of Photogrammetry, en San Luis, Missouri.

Ha impartido cursos en computación (Computación I y II, Programación de Sistemas, Estructura de datos) a nivel de Licenciatura y Lenguajes de Programación I a nivel de Maestría.

Sus intereses profesionales: programación en paralelo, sistemas de información, computación distribuida.

Adolfo Guzmán Arenas. Egresado del IPN-ESIME (Ingeniero en Comunicaciones y Electrónica, tesis "CONVERT", un lenguaje para careo de expresiones y manipulación de símbolos). Es Doctor en Ciencias de la Computación del Massachusetts Institute of Technology (1968, tesis "Decomposition of a visual scene into 3-dimensional bodies"), y fue profesor del M.I.T. (Depto. Ing. Eléctrica, 1969-70). Fue Director del Centro Nacional de Cálculo del IPN, Investigador Titular del Centro de Investigación y Estudios Avanzados del IPN, Director del Centro Científico IBM de América Latina, Jefe del Departamento de Computación del IIMAS-UNAM. Actualmente es Investigador Titular "C" de dicho Instituto. Intereses profesionales: reconocimiento de formas, bases de datos, procesamiento distribuido, inteligencia artificial.

Juan López Fragoso. Estudiante de Ingeniería en Electrónica, Fac. de Ingeniería, UNAM. Actualmente es becario del proyecto AHR en el IIMAS. Sus intereses profesionales: circuitos digitales, microprocesadores.

Luis Lyons Vargas. Realizó estudios de Ingeniero Mecánico Electricista en la Facultad de Ingeniería, UNAM. Trabajó en Hewlett Packard en instrumentación. Con amplia experiencia en Electrónica, colaboró en el Instituto de Ingeniería (UNAM)

en el Laboratorio de Automatización, en los proyectos "Reconocimiento de voz" y "Procesamiento de Imágenes."

Durante 1978 trabajó en la SPP, en el Sistema Nacional de Información.

Desde 1979 a la fecha es colaborador del Departamento de Sistemas de Cómputo del IIMAS.

Es poseedor de varias patentes.

Intereses profesionales: manejo digital de imágenes, reconocimiento de voz, electrónica digital y analógica.

Pablo Martínez Suárez. Estudiante de Ingeniería en Electrónica. Facultad de Ingeniería, UNAM. Actualmente, becario del Proyecto AHR del IIMAS. Intereses profesionales: circuitos digitales, microprocesadores.

Isauro Morales Flores. Estudiante de Ingeniería en Electrónica. Facultad de Ingeniería, UNAM. Actualmente, becario del Proyecto AHR del IIMAS. Intereses profesionales: microprocesadores, circuitos digitales.

Luis Hugo Peñarrieta Echenique. Ingeniero Mecánico Electricista, Facultad de Ingeniería, UNAM, 1975. Tesis "Diseño y Construcción de un simulador de funciones lógicas". Maestría en Electrónica, Div. de Estudios Superiores, Fac. Ing., UNAM, 1978. Tesis "Multiprocesamiento basado en una memoria compartida por n procesadores." Actualmente es investigador del IIMAS - UNAM, Profesor de la Maestría en Computación y de la Licenciatura en Computación. Intereses actuales: procesamiento distribuido, sistemas de información por hardware, microprocesadores.

David Rosenblueth Laguette. Ingeniero Mecánico Electricista, Facultad de Ingeniería, UNAM, 1978. Tesis "Un sintetizador de voz." Estudiante de la Maestría en Computación, IIMAS-CCH. Actualmente colabora en el Laboratorio AHR del IIMAS. Intereses profesionales: miltiprocesamiento.

B I B L I O G R A F I A

INFORMES TECNICOS DEL LABORATORIO AHR

Serie y No.	AUTOR	TITULO	AÑO
NA 133	Adolfo Guzmán Raymundo Segovia	A parallel configurable Lisp machine. AHR-76-1.	1976
NA 200	David Rosenblueth Carlos Velarde	La máquina AHR para procesamiento en paralelo. 1a. etapa. AHR-79-2	1979
NA 206	Adolfo Guzmán	Heterarchical architectures for parallel processing of digital images. AHR-79-3	1979
NA 214	Kemer Norkin Dora Gómez	A new description for data transformation in the AHR computer AHR-79-4	1979
NA 215	Kemer Norkin David Rosenblueth	Towards optimization in AHR AHR-79-5	1980
NA 216	Adolfo Guzmán	La computación distribuida como una alternativa del futuro AHR-79-6	1979
AM 15	Adolfo Guzmán	Ciento veinticinco proyectos y temas de tesis en computación AHR-80-7	1980
NA 229	Adolfo Guzmán	Reconfigurable geographic data bases AHR-80-8	1980
	Luis Hugo Peña-rrieta	Detección de errores en la máquina AHR AHR-80-9	1980
NA 253	Adolfo Guzmán, Luis Lyons,et al.	La computadora AHR: Construcción de un multiprocesador con Lisp como lenguaje principal AHR-80-10	1980
VE 17	Adolfo Guzmán, David Rosenblueth, recop.	Estructuras de computadoras digitales AHR-80-11	1980

	Víctor Germán Sánchez	Sistema de Información General de estructura reconfigurable AHR-80-12	1980.
NA 246	Adolfo Guzmán	A parallel heterarchical machine for high level language processing AHR-80-13	1980.

REFERENCIAS

1. Abramson, N. The ALOHA System. Computer Communication Networks. Prentice Hall, N. J. 1973.

2. Alvarez, Manuel. A LISP growing machine. M. Sc. Thesis, Moore School of E. E., U. of Pennsylvania. 1967.

3. Auerbach Computer Technology Reports. Auerbach Publishers Inc. Philadelphia, Penn.

4. Bashkow, T. R.; Sasson, A., and Kronfeld, A. System design of a FORTRAN machine. Chapter 31, pp 363-381. In [5].

5. Bell, C. G., and Newell, A. (Eds). Computer Structures: Readings and Examples. McGraw Hill. 1971.

6. Bell, C. G.; Eggert, J. L.; Grason, J., and Williams, P. The description and use of register transfer modules (RTM's), IEEE Trans. on Computers, C-21, 5, 495-500, May 1972.

7. Berkling, Klaus. A computing machine based on tree structures. IEEE Trans. Computers, Vol. C-20, 4, April 1971.

8. Boley, Harold. A preliminary survey of artificial intelligence machines. SIGART Newsletter 72, July 80. 21-28.

9. Bobrow, D. G., and Wegbreit, B. A model for control structures for artificial intelligence programming languages. Proc. Third Intl. Jt. Conf. on Art. Intelligence, Stanford Research Institute, Stanford, Calif. 1973. pp 246-253.

10. Control Data 6000 Series Computer Systems. Reference Manual. Control Data Co. Minneapolis, Minn.

11. Cornell, J. A. Parallel processing of ballistic missile defensive radar data with PEPE. COMPCON 72, Sept. 1972. pp 69-72. We refer to the PFOR language of this machine.

12. Dennis, J. B., and Van Horn, E. C. Programming semantics for multiprogramming computations. Comm ACM, 9, 3, March 1966. 143-155

13. Deutsch, L. Peter. A LISP machine with very compact programs. Proc. Third Intl Jt Conf on Art Int, S.R.I., Stanford, Cal. 1973. 697-703

14. Digital Equipment Co. PDP-11 Peripherals Handbook 1975.

15. Enslow, P. H. (ed). Multiprocessors and parallel processing. John Wiley. New York 1974

16. Enslow, Philip H. (ed) Proceedings of the 1976 International Conference on Parallel Processing. IEEE Publication No. 76 CH 1127-0C

17. Feustel, E. A. On the advantages of a tagged architecture. IEEE Trans Comp C-22, 7, Jul 73, 644-656.

18. Gleary, J. G. Process handling on Burroughs B6700. Proc. Fourth Australian Comp Conf 1969.

19. Glushkov V. M., et al. Recursive machines and computing technology. Proc. IFIP 1974 North Holland 65-70

20. Gostelow, K, P., and Thomas, R. E. A view of dataflow. AFIPS 1979 Conference Proceedings, Vol 48, 629-636

21. Gurd, John, and Watson, Ian. Data driven system for high speed computing. Part 1: structuring software for parallel execution. Computer Design, June 80, 91-100.

22. Hauck, E. A., and Dent, B. A. Burroughs B6500/B7500 stack mechanism, Proc. SJCC 1968.

23. Guzmán, A., and McIntosh, H. V. CONVERT. Comm ACM 9, 8, Aug. 66.

24. Heart, F. E., et al. A new minicomputer/multiprocessor for the ARPA network. Proc AFIPS 1973 Natl Comp Conf. AFIPS Press, Montvale, N. J. 1973

25. Hewitt, Carl; Bishop, Peter; and Steiger, Richard. A universal modular ACTOR formalism for artificial intelligence. Proc Third Intl Jt Conf on Art Int, S.R.I., Stanford, Cal. 1973 235-245.

26. Higbie, L. C. The OMEN computers: associative array processors. COMPCON 72, 287-290. We refer to Fortran and Basic of this machine.

27. IBM Systems 370 Principles of operation. GA22 7000 4.

28. McCarthy, John, et al. LISP 1.5 Programmer's Manual. MIT Press. 1962

29. John McCarthy. Recursive Functions of symbolic expressions and their computation by machine. COMM ACM 3, 4, April 60.

30. Kautz, W. H., and Pease, M. C. Cellular logic-in-memory array. National Tech Information Serv, Nov 71, AD763710

31. Keller, Robert E. Look ahead processors. Computer Surveys 7, 4, Dec 75.

32. Keller, R. M.; Lindstrom, G., and Patil, S. A loosely-coupled applicative multi-processing system. AFIPS 1979 Conf Proceedg Vol.48, 613-622

33. Lamport, Leslie. Garbage collection with multiple processes: an exercise in parallelism. Pages 50-54 in [16].

34. Landin, P. J. A program machine symmetric automata theory. in Machine Intelligence 5, Edinburgh Univ. Press. 99-120

35. Levialdi, Stefano and Duff, Michael. Proceedings of the workshop on new computer architectures and image processing. Ischia, Italy, May 1980. To appear as a book by Academic Pr.

36. Lorin, Harold. Parallelism in hardware and software: real and apparent concurrency. Prentice-Hall, N. J. 1972

37. Lawrie, D. Hl, et al. GLYPNIR- A programming language for Illiac IV. Comm ACM March 75, 157-164

38. Magidin, M., and Segovia, R. LISP B6700 Implementation. Technical Report 5, 70, Computer Science Dept., IIMAS, National University of Mexico. 1974

39. Miller, Raymond E. A comparison of some theoretical models of parallel computation. IEEE Trans Computers Aug 73 710-717

40. Miller, R., and Cocke, J. Configurable computers: a new class of general purpose machines. Intl Symp on Theoretical Programming. Ershov and Nepomniaschy (Eds) Springer Verlag New York 1974 285-298

41. Patil, S. S. An abstract parallel-processing system. S. M. Thesis, E. E. Dept. M.I.T. June 67

42. PERQ- a landmark computer systems. (Brochure). Three Rivers Computer Co. 160 North Craigh St. Pittsburgh, Pa.

43. Proc. Symp on Computer Architecture Dec 73. Avail from IEEE

44. Proc. 1975 Sagamore Computer Conference on Parallel Processing Available from IEEE

45. Raytheon Data systems array transform processor. Raytheon Data Systems. Norwood, Mass.

46. Record of Project MAC Conference on concurrent systems and parallel computation. 1970. Available from ACM

47. Rumbaugh, J. E. A parallel asynchronous computer architecture for data flow programs. Ph. D. Th, EE Dept., M.I.T

48. Segovia, Raymundo. Computer networks for load sharing. To appear as a technical report, Computer Sci. Dept, IIMAS.

49. Thornton, J. E. Design of a computer: the Control Data 6600 Scott, Foreman & Co.

50. Thurber, J. K. Associative and parallel processors. Computer Surveys (ACM) 7, 4, Dec 75

51. Watson, Ian, and Gurd, John. A prototype data flow computer with token labeling. AFIPS 1979 Conference Proceedings, Vol. 48, 623-628.

52. Weinreb, D., and Moon, D. Lisp Machine manual. M.I.T. A. I. Laboratory. 1979.

53. Wulf, W., and Bell, C. G. "C.mmp - a multi-mini processor" Proc. AFIPS 1972 FJCC AFIPS Press.