ESCUELA UNIVERSITARIA POLITECNICA DE LAS PALMAS DE GRAN CANARIA

TITULO: SISTEMAS MULTIPROCESADORES. APLICACION.

Autor:

Javier A. Moran Carrera

Tutor:

Sebastian Suarez Gil

PREFACIO

Este trabajo fin de carrera trata sobre el diseño , construccion y verificacion del software y hardware de un sistema multiprocesador teniendo como host a un sistema de desarrollo MDS-221.

Este trabajo consta de tres partes. La primera expone conceptos teoricos y arquitecturas de sistemas multiprocesadores estandard. Se trata de mostrar las ventajas y limitaciones en estos sistemaas asi como las distintas soluciones que diversos fabricantes y organizaciones han propuesto. Se mostraran los resultados del grupo de Torino, sobre la evaluacion de distintas arquitecturas de sistemas multimicro. La segunda se centrara en la arquitectura Multibus y la familia iAPX-86 mostrando distintas configuraciones de soporte de estos sistemas. Se expresaran los resultados obtenidos en la implementacion fisica y se expondran los problemas de depurado que plantean estos sistemas. Se analizaran las distintas opciones para el desarrollo de un controlador de disco para nuestro sistema de desarrollo MDS-221. La tercera parte reune planos y listados.

Cada capitulo contiene una bibliografia parcial. Las conclusiones extraidas y una bibliografia completa estan expuestas al final del trabajo.

Este trabajo fue redactado usando un editor de texto tipo WordStar con sus consiguientes ventajas e inconvenientes. Se emplean las siguientes formulaciones:

<x> referencia x del la bibliografia de ese capitulo.

x/ indica que la senal x es activa a nivel bajo.

Quiero agradecer a todos los "presuntos" implicados, que forman una lista verdaderamente larga, su aportacion material y moral en la realizacion en este proyecto.

LAS PALMAS DE GRAN CANARIA. Julio 1986.

INDICE.

PRIMERA PARTE

CAPITULO 1

CAPITULO 1. INTRODUCCION A LOS SISTEMAS MULTIPROCESADORES

1.1.PROCESO DISTRIBUIDO Y MULTIPROCESADORES.

Los sistemas multiprocesadores son una parte de los llamados sistemas de computacion distribuidos.

Los sistemas distribuidos constan fundamentalmente de un conjunto de elementos de computo PE1, PE2, ... PEn, unidos por una red de interconexion. Cada elemento de computo puede ser un sistema. Vease la figura 1.1.

Se pueden considerar sistemas en el que la cooperacion (intercambio de datos y-o sincronizacion) entre los elementos ocurre muy a menudo, y por el contrario, tenemos los sistemas distribuidos en el que el intercambio de datos es muy frecuente y la sincronizacion ocurre al nivel de instruccion.

Las redes de computadores es la clase mas vieja entre los sistemas distribuidos y tomo su origen de la conexion entre grandes mainframes. Cada procesador tiene una fuerte autonomia local y dedica solamente una parte limitada del su poder de proceso a actividades comunes.

Los sistemas multiprocesador estan formados por varios procesadores totalmente programables que pueden ejecutar su propio programa, pero el conjunto forma una entidad simple. El hecho que distingue a estos de las redes de computadoras es que en el ultimo caso, todos los recursos del sistema estan coordianados a traves de una tarea comun con un mecanismo unico de control (centralizado o distribuido). La cantidad de intercambio de informacion entre los procesadores

puede ahora ser significativamente mayor que en el caso previo. La topologia de la interconexion y la estrategia de comunicacion entre procesadores se convierte en este caso en un puto crucial del sistema.

Los sistemas en que los procesadores no comparten memoria y estan conectados a traves de canales de I/O estan normalemente definidos como multi-computadores o sistemas bajamente acoplados, mientras las estructuras con un espacio de direcciones comun son llamadas multiprocesadores o sistemas altamente acopladas. Las arquitecturas de los primeros puede ser similar a las redes geograficamente distribuidas. El canal de interconexion puede tener una velocidad que varie entre unos pocos kbit por segundo hasta 10 megabit por segundo.

Existen sistemas de proposito general cuya estructura es paralelizable y que se suelen conectar como perifericos a algun sistema de alto poder computacional para acelerar ciertas operaciones frecuentemente necesitadas. Aqui podemos encontrar sistemas que efectuen transformadas discretas de Fourier, o arrays sistolicos para el computo de operaciones matriciales. Existen tambien los llamados procesadores en array que son elementos que efectuan la misma operacion sobre varios grupos diferentes de datos. Son muchas las distintas clasificaciones que se han venido dando a las diferentes arquitecturas, como por ejemplo, SIMD, (single instruction - multiple data), MIMD (multiple instruction - multiple data), etc... Una muestra de estas clasificaciones puede consultarse en <1> y <3>. Existe una gran bibliografia

de descripciones de sistemas distribuidos y sistemas multiprocesadores, como puede consultarse en <1>, <2>, <3>, y <4>.

La estructura mas general de un sistema multiprocesador se muestra en la figura 1.2. Un multiprocesador consiste de un conjunto de modulos master tales como procesadores, y de un conjunto de unidades esclavas tales como memoria conectadas juntas por medio de una estructura de interconexion. La estructura de interconexion es la parte mas importante del sistema porque el intercambio de informacion entre las unidades de proceso depende de esta.

La justificacion del empleo de esta clase de sistemas, se fundamenta en la posibilidad de que numerosos algoritmos son paralelizable o susceptibles de serlo, es decir, de descomponerse en subprocesos que puedan ser ejecutados independientemente en cada procesador con una comunicacion entre ellos. Las tendencias actuales impulsan las investigaciones en torno a la implementacion de sistemas multiprocesadores con elementos VLSI, es decir, la construccion de sistemas multimicroprocesadores. Se estudian actualmente las redes de interconexion entre diversos elementos y sus caracteristicas de adaptacion a, por ejemplo, la filosofia de diseno RISC. Ejemplos de ello se encuentran en <3> cuando trata de elementos tales como el RIMMS, PIPE o el T414 (Transputer).

En <4>, <5> y <6> se describen arquitecturas muy conocidas

(casi academicas) como el Cmmp, el sistema iAPX-432, o el ILLIAC IV. Su lectura, a efectos ilustrativos puede ser provechosa. En <7> pueden verse las distintas tecnicas empleadas en las redes de interconexion entre procesadores tales como redes crossbars, shuffle networks (redes barajadas)....

En <1>, <2> puede tenerse una buena idea de los sistemas basados en un unico bus paralelo que lleva de intercambio de mensajes. Este bus, el de mayor exito comercial, que se aprecia en la figura 1.3, sera el tratado fundamentalemente en este proyecto.

1.2.MOTIVACIONES ARQUITECTONICAS PARA SISTEMAS MULTIMICROPROCESADORES

Los progresos de la tecnologia VLSI que hacen posible la elaboracion de microprocesadores de bajo coste y altas prestaciones y otros circuitos han impulsado la construccion de sistemas de computacion distribuidos, usando un conglomerado de tales componentes.

La idea de proceso distribuido implica dos conceptos; se supone que la tarea a realizar esta distribuida entre varios procesadores con lo que un cierto grado de paralelismo en el proceso existe; por otro lado, las tareas estaran distribuidas geograficamente, es decir, ocuparan situaciones fisicas diferentes, extendiendose entre unas decenas de centimetros o cientos de kilometros.

Se pueden identificar las siguientes motivaciones para la construccion de sistemas multimicroprocesador:

1.-Aumento de prestaciones.
2.-Aumento de seguridad.
3.-Encuentro con los requisitos de aplicaciones distribuidas.
4.-Construccion de computadores de uso general.
5.-Construccion de supercomputadores.

Analizando cada uno de estos puntos mas en detalle:

Aumento de prestaciones.

El empleo de hardware duplicado deberia implicar una mejora en las prestaciones. Esto asume que la tarea a efectuar sea susceptible de descomponer en otras de menor tamaño que puedan ser emplazadas en cada procesador. Ademas se encuentra con los problemas relacionados con la identificacion de las tareas paralelas, su emplazamiento en los microprocesadores, la coordinacion entre estos, etc. Solo la correcta solucion de estos problemas que implican al paralelismo, puede conducir a una mejora real del poder computacional del sistema.

Aumento de la seguridad.

La idea de la duplicacion hardware implica que el sistema podria recuperarse de un fallo unico de cualquiera de sus elementos, ya que otros asumirian sus funciones. El sistema perderia prestaciones pero seguiria funcionando correctamente. Esto es lo que se denomina "fault-tolerant computing". Esto es fundamental en determinadas circunstacias donde un fallo en el funcionamiento pudiera tener graves consecuencias.

Construccion de computadores de proposito general

El aumento de prestaciones antes formulado, y las

caracteristicas atractivas del VLSI, resultaria ventajoso la construccion de un computador de proposito general. Aunque han habido muchos proyectos experimentales, las ventajas comerciales de tal aproximacion no han sido todavia demostradas para la construccion de "mainframes". Aun cuando la operacion multitarea tipica de los mainframe se puede adaptar a los sistemas multimicroprocesador, los problemas debidos a la coordinacion y comunicacion son considerables. El sistema operativo que maneje este computador seria complejo, y la gestion de la memoria se convierte tambien en tarea compleja. Este panorama podria cambiar con las nuevas generaciones de microprocesadores con capacidad de 32 bits, potentes conjuntos de instrucciones y capacidades artimeticas y los nuevos circuitos de manejo de memoria que simplifican el manejo de grandes espacios de memoria.

Construccion de un supercomputador

Eliminando el factor de coste caracterizado en la construccion de mainframes, la idea aqui es desarrollar un sistema de muy alta velocidad de computacion. La idea es tener miles de estos componentes para superar el poder computacional del mas potente sistema computador. Las construccion de tal maquina encuentra mucho de los problemas anteriormente citados, y la contencion debida al uso comun de recursos puede degradar al sistema. Se pueden adoptar redes de conmutacion multinivel que aumentaria el retraso de comunicacion, o bien una organizacion jerarquizada. Esto exige en cualquier caso que la aplicacion este tambien

estructurada de tal manera que se corresponda de la mejor forma con la arquitectura hardware. Esta idea es mas atractiva suponiendo que este compputador se construyera para un proposito especial, un problema con una estructura bien definida, tal como hacen los procesadores en array.

1.3.FACTORES QUE INFLUYEN EN EL DISENO DEL SISTEMA

Tal como ya se ha mencionado, el disemo de un sistema multimicroprocesador implica la consideracion de factores a menudo conflictivos. El hardware y el software de tales sistemas son complejos y es dificil llegar a un disemo optimo evitando otros cuellos de botella. En la siguiente tabla se mencionan algunos de estos.

*TOPOLOGIA
.tamano fisico
.local
.geograficamente distribuido

*PATRON DE CONEXION

.sincrono,asincrono,conmutacion circuito
.homogeneidad de los nodos
.regularidad de la topologia

*CONMUTACION
.centralizada, distribuida

*MEMORIA
.local, compartida
.direccionabilidad

*BUS DATOS
.concurrencia
.ancho
serie,paralelo,mixto
.protocolo transferencia

*DEADLOCK
.prevencion
.evitacion
.deteccion y recuperacion

*SOFTWARE
.sistema operativo
.base de datos

Se trata ahora estos temas con mas detalle.

Topologia y patron de interconexion

La topologia depende del numero de procesadores y el patron de interconexion. En el disemo de un sistema

multimicroprocesador, hay ventajas en el uso de nodos homogeneos en el caso de que el proceso computacional sea homogeneo.

Los patrones de interconexion pueden ser considerados en dos grupos:: estructuras regulares e irregulares. En el primer caso, existe una regla constructiva que defina la estructura. Las redes regulares se emplean en sistemas locales, mientras que las irregulares, propia de sistemas distribuidos geograficamente en el que el patron de interconexion depende de la red.

Conmutacion

La conmutacion que proporciona el intercambio de datos entre varios microprocesadores podria ser centralizado o distribuido.

Memoria

La organizacion de la memoria en una estructura multimicroprocesador es muy importante porque para muchos sistemas proporciona la comunicacion entre varios procesadores. De hecho, para pequemos sistemas multimicro, este es el unico medio para que los procesadores pasen datos entre ellos. A menudo cierta memoria es asignada localmente para cada procesador con un modulo de memoria comun con propositos de comunicacion. Esta tecnica aprovecha la locabilidad de los programas, manteniendos los elementos de comparticion con los otros procesadores en memoria comun. La

direccionabilidad es importante porque normalmente la capacidad del direccionamiento de un procesador es insuficiente en un sistema multimicroprocesador. Normalmente se usan tecnicas de segmentacion.

Bus de datos

Los bus de datos estan caracterizados por el grado de concurrencia, es decir por el numero de lineas que contribuyen a la transmision de datos. Asi tenemos la Ethernet que solo tiene una linea. Los buses de datos comparticdos necesitan resolver contencion. Si la frecuencia de peticiones de bus es alta, entonces se introducen retraso, y de aqui un deterioro de las pretaciones. Los sistemas con lineas de transimision paralelas solo se emplean en sistemas altamente acoplados donde un bus es compartido con varios procesadores. Para distancias mayores, se emplea la alternativa serie. En ciertas circunstancias se efectua la conmutacion de los datos a nivel hardware pero normalmente se emplea una conmutacion a nivel logico por paquetes. Se pueden emplear modos asincronos y sincronos de transmision.

Deadlock

El deadlock es definido como una situacion donde ningun proceso puede proceder sin adquirir un recurso ya mantenido por otro procesadores. El manejos del deadlock es un aspecto importante en el diseno de sistemas multiprocesadores debido a que un cierto numero de recursos estan compartidos y hay un alto grado de complejidad e interdependencia entre procesos

individuales. La prevencion del deadlock requiere mecanismos software y hardware como asignacion circular de recursos. La deteccion del deadlock es a menudo afrontada por medio de un mecanismo de time-out.

Software

El desarrollo del software es una de las consideraciones mas importantes. El aspecto hardware necesita ser reflejado en el software. Una primera consideracion es sobre el sistema operativo que implica inmediatamente un entorno multiproceso. El sistema puede estar centralizado en un procesador encargado de las funciones del scheduling, asignacion de recursos, etc... Otra aproximacion es tener un sistema operativo descentralizado con las funciones distribuidas en cada elemento procesador sin ningun punto central de control. Argumentos similares se aplican al manejos de las bases de datos.

Seguridad

El aumento de complejidad requiere un analisis a la tolerancia a fallos del sistema. En ciertos casos, este aspecto es aun mas importante que las mismas prestaciones.

BIBLIOGRAFIA CAPITULO 1.

<1>: G. Conte y otros.

Multi-microprocessor systems for real-time aplications.

D. Reidel Publishing Company.

<2>: Y. Paker.

Multimicroprocessor systems.

Apic studies in data processing 1978.

<3>: V. Milutinovic

Advanced microprocessors and high-level language computer architectures for VLSI.

Onceavo simposio sobre microproceso y microprogramacion.

<4>: D. Siewiorek y otros.

Computer structures: Principles and examples.

Mac Graw Hill computer science series.

<5>: M. Valero.

Arquitectura de los computadores de alta velocidad.

VII Escuela de verano de informatica. AEIA.

<6>: Intel Corp.

Microsystems components handbook volume 1.

iAPX 432 Micromainframe.

<7>: M. Valero y otros.

Redes de interconexion para sistemas multiprocesadores.

Mundo Electronico.

<8>: Intel Corp.

OEM systems handbook.

AR-55 , AR-72 , AR-229.

<9>: J. Figueras.

Arquitecturas multimicrocomputador.

III Escuela de verano de informatica. AEIA.

En esta referencia se incluyen:

<10>: C. Weitzman.

Distributed micro-mini computer systems.

Editorial Prentice-Hall.

<11>: P. Borrill.

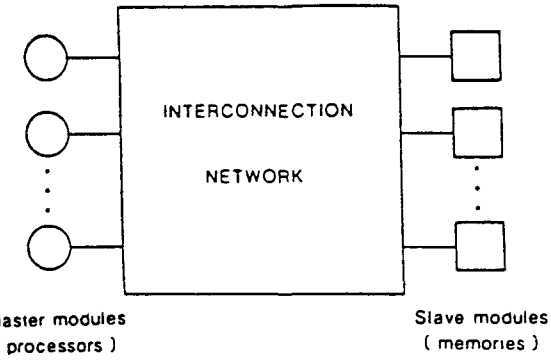
Microprocessor bus structures and standards.

IEEE Micro Febrero 1981.

<12>: A. Allison.

Status report on the P896 backplane bus.

IEEE micro Febrero 1981.



- The most general structure of a multiprocessor system.

Fig. 1.1 Sistema multiprocesador. En los sistemas distribuidos, no se encuentran las memorias, y la red de interconexion tiene un menor caudal. (redes de ordenadores).

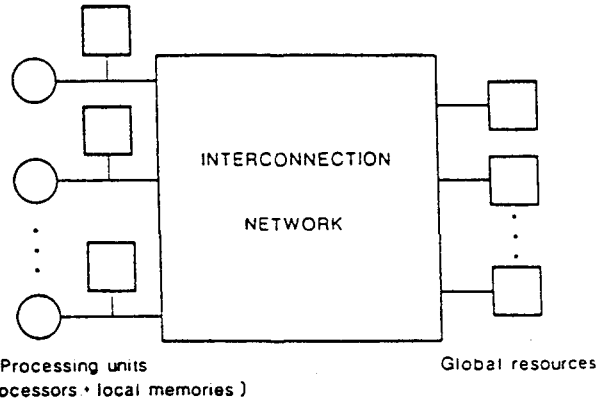
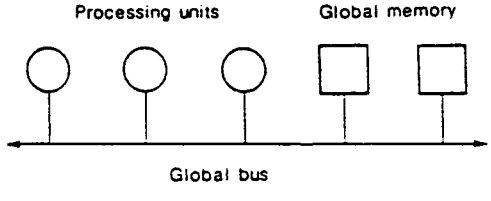


Fig. 1.2 Organizacion usual. Existencia de recursos locales (menos contencion por recursos).



- Single global bus multiprocessor system

Fig. 1.3 Estructura mas comun empleada en industria. Todos los buses que se estudiaran pertenecen a este ramo.

CAPITULO 2

CAPITULO 2. ESTRUCTURAS HARDWARE EN SISTEMAS MULTIPROCESADORES

2.1. INTRODUCCION

Se planteara aqui, las estructuras mas comunes de sistemas multiprocesadores que tienen como red de interconexion, un unico bus comun. Esta ha sido la estructura de mayor exito y aceptacion y por tanto, sera practicamente la unica a la que me refiera el resto del trabajo.

El diseno de sistemas multiprocesadores de acuerdo con unos estandards bien definidos, implica algunas restricciones a la estructura logica y a su implementacion fisica. Estas restricciones definen por ejemplo, el diseno de la circuiteria de interface o las caracteristicas electricas del bus. Sin embargo, dentro de estas restricciones, es posible identificar algunas estructura basicas y reglas generales de diseno. Aqui, analizaremos algunas de las organizaciones y arquitecturas' tipicas de varios tipos de modulos: masters, slaves, mixtos y unidades especiales. Las estructuras aqui descritas no son de ningun bus en particular, sino mas bien, estan relacionadas con la funcion que efectuan. Se tratara de relacionar estas estructuras con algunas implementaciones en determinados buses. Se mostraran solo los niveles que dependen o influyen en el protocolo.

2.2. ORGANIZACION FISICA DE SISTEMAS MULTIPROCESADORES.

En la mayoria de los sistemas multiprocesadores , la estructura de los buses en los sistemas multiprocesadores

esta organizada jerarquicamente como se muestra en la figura 2.1. Asi, se puede identificar un bus que lleva las comunicaciones globales y un confunto de buses que interconectar el procesador con memoria o I/O. El primero recibe el nombre de bus global o bus del sistema, mientras que el segundo suele responder a la denominacion de bus local.

El bus global debe ir a todos las placas, mientra que el bus local solo se conecta con algunos modulos solamente. Un ejemplo claro de esta estructura esta en la arquitectua Multibus I y II de la Intel que se mostraran mas tarde. La solucion usual es agrupar junto los modulos asociados con el mismo bus local, para tener un bus fisicamente local. Estos buses locales pueden estar confinados dentro de una placa. Estando asi disenados para una configuracion fija, este bus local puede estar optimizado para el conjunto de dispositivos usados y se suelen usar protocolos sincronos sin buferizar. El problema fundamental radica en que la expansion suele estar limitada, aunque se puede "montar" alguna expansion encima de la placa tal como hace la expansion iSBX del Multibus.

Una disposicion tipica es la mostrada en la figura x.x en donde un conector es empleado para la interconexion de los buses locales y otro para los globales. El Multibus tiene definido el estandar iLBX para la conexion local. El VME no puede usar esta disposicion porque el segundo conector esta

usado para expansion de bus y I/O.

Otra posibilidad que se verifica en el P896 es poner el interface hacia el bus global en una placa separada tal como se muestra en la figura 2.2.

2.3.ESQUEMAS DE DESARROLLO DE PLACAS.

Las especificaciones del bus definen las caracteristicas electricas de las placas y los buses de conexion, tales como los niveles de voltaje, corrientes de entrada y termporizacioines. La propagacion de las senales, la carga de los drivers, y otros efectos, pueden causar un comportamientos electrico incorrecto (errores de paridad, etc...). Estos efectos dependen de la tecnologia usada en la circuiteria de interface. Un interface ideal forzaria la linea independientemente de la carga (o en otras palabras, seria capaz de drenar cualquier corriente requerida) y podria leer el estado de la linea sin afectar a este (lo que implica que no pide corriente). Esto esta limitado a los drivers actualmente existentes que poseen un fan-out limitado.

En muchos casos una senal del bus debe ir a muchos circuitos distintos en la misma placa, como es el caso de una linea de direcciones-datos multiplexada. Una linea asi conectada drena mucha corriente, lo que puede llegar a introducir en caso de que las pistas sean muy largas, a una carga reactiva. Para limitar estos problemas de adaptacion de impedancias y carga, cada placa deberia usar un driver para

cada linea del bus , localizado lo mas cerca posbile del conector. Esto permite el uso de dispositivos optimizados para la propagacion de semales y aisla la placa del resto del sistema, lo cual puede ser necesario en un sistema fault-tolerant. Una desventaja estriba en el aumento del tiempo de propagacion. El crosstalk debe ser limitado usando las tecnicas de trazado precisas. El problema del ruido impulsivo es mucho mas importante en aquellas lineas que llevan semales sensitivas por flanco. Lineas activas por nivel como las de datos o direcciones que estan habilitadas por los flancos de otras lineas, deben estar fijas solamente en la proximidad de los flancos activos de los comandos. El desacoplo de la fuente ayuda a limitar el ruido.

2.4.MODULOS ESCLAVOS.

Una unidad esclava, puede definirse como una unidad que monitoriza la actividad del bus, y cuando este selecionado, puede llegar a participar en una transferencia de datos. Unidades esclavas tipicas son la memoria y los interfaces de I/O. Un controlador inteligente o un procesador periferico, suelen tambienr ser considerados por el bus como esclavos. Un ejemplo de este caso esta en el controlador de entradas y salidas (IOC) del MDS 221.

2.5.MODULOS MASTER.

Un modulo master es capaz de comenzar y manejar actividades en el bus, para transferir datos desde o hacia modulos esclavos. Tipicos masters son los procesadores y los

controladores de acceso directo a memoria (DMA). Debe ser senalado que incluso si un master contiene una CPU o DMA, en la mayoria de los casos contiene tambien algunas unidades esclavas.

La estructura interna de los masters puede quedar groseramente descrita en la figura 2.3. Se observa como ademas del acceso al bus externo, existe una memoria local e I/O. Los dispositivos que solo pueden ser accedidos por esta CPU, se denominan LOCALES. Es posible que algunos esclavos puedan ser accedidos por una CPU y ademas por cualquier otro master del sistema. En tal caso, se denominan PRIVADOS. Estos modulos master incluyen las unidades de arbitraje que se expusieron en un punto anterior. Normalmente es necesario una logica para que se puedan "traducir" las senales de control de la CPU con la del bus del sistema. Asi mismo, se deben insertar estados de espera hasta que el arbitraje haya sido concluido y el procesador pueda acceder a los recursos comunes del sistema. Se puede dar el caso de que el procesador intente acceder a un recurso que no existe. En tal caso, este quedaria indefinidamente parado y bloqueando al resto del sistema en espera de una trasaccion que nunca se efectuara. Para remediar esto, se suele incluir un circuito de "time-out" que aborta el proceso cuando no se recibe reconocimiento del recurso en un tiempo prudente. El procesador puede entonces efectuar una interrupcion para solventar la excepcion.

2.6.MODULOS ESPECIALES.

En muchas circunstancias, un determinado dispositivo no puede ser considerado como perteneciente a una de las clases antes mencionadas. Algunos de estos dispositivos pueden ser:

-modulos multi-esclavo
-bus windows
-esclavos doble-puerto
-modulos master-slave
-unidades de transferencia de bloques
-modulos supervisores

Paso a describir, muy brevemente la funcion y caracteristicas fundamentales de estos dispositivos.

-Modulos multi-esclavo.

Varios elementos, correspondientes a unidades funcionalmente distintas, coexisten en la misma placa, compartiendo muchas veces los transceivers y la logica de seleccion. Esto suele corresponder, por ejemplo, a modulos con configuraciones complejas de memoria, o con I/O incluido.

-Bus windows.

Para la extension de un bus paralelo mas alla de un unico segmento, se puede emplear dos tecnicas. Una puede ser el uso de una linea de comunicacion comandada como I/O, existiendo un modulo que intercepta los mensajes que van con destino a otro segmento y enviandolo por esta linea de comunicacion, en cuyo extremo, otro modulo hara la operacion inversa. La desventaja radica en la baja capacidad de este canal de comunicacion que no seria apto es sistemas altamente acoplados. Un segundo caso, es la union por un interfase paralelo de casi todas las lineas del sistema. Este interfase actuaria como master para uno de los modulos, mientras para

otro se comportaria como esclavo. Por supuesto, al ser una comunicacion paralela en que el arbitraje es a tiempo real, la capacidad de comunicacion no queda muy degradada, a costa de un hardware mas costoso.

-Esclavos doble-puerto.

Ciertas configuraciones de modulos con master y esclavo en la misma placa, estan organizados de forma tal que el esclavo puede ser accedido tanto por cualquiera de los procesadores del sistema como por el master de su mismo modulo. En el caso de acceso por el master de su modulo, el bus del sistema no sufre perdida de prestaciones, debido a que la comunicacion se hace por camino separado, a no ser que haya contencion por el mismo recurso. Un ejemplo de esto son las placas de soporte de memoria adyacentes al iLBX en el multibus. Por supuesto, existe un sub-arbitraje en el modulo para decidir quien accede al esclavo. Una implementacion de este pequeño esquema de arbitraje puede ser como el planteado en la figura 2.4, en el que cada una de las peticiones, corresponde a uno de los master posibles.

-Unidades de transferencia de bloques

Algunos modulos esclavos son capaces de transmitir datos a muy alta velocidad usando las facilidades que para ello dan algunos buses. Esto suele ser comun, por ejemplo en las unidades de control de disco.

-Supervisores.

Los supervisores son modulos especiales con dos funciones fundamentales: monitorizacion e intervencion. Se encargan de

funciones de depurado y mantenimiento, o bien actuan como vigilantes, generando excepciones en cuanto encuentran una circunstancia anomala en el comportamiento del sistema. Tambien pueden implementar elementos de proteccion a nivel hardware.

2.7. TECNICAS DE ARBITRAJE.

El bus es un unico recurso que puede ser usado por muchos pares de modulos en el intercambio de informacion. Muchos procesadores pueden ser conectados al mismo bus; estas unidades son capaces de pedir el bus y llevar a cabo transferencias independientes de informacion. Sin embargo, el bus solo puede llevar un paquete de informacion a un tiempo. La activacion de muchas peticiones para un unico recurso es llamada CONTENCION. Si varios peticionario, o sea varios procesadores, obtienen y usan el bus, esto causa una COLISION. En una colision, el protocolo es corrompido, la secuencia de operaciones podria volverse inconsistente, y la informacion transmitida es casi siempre perdida.

Para trabajar adecuadamente, un unico procesador debe tener control del bus, o sea que actue como master del sistema, en un determinado instante. Para garantizar esta condicion, el prtocolo del procesador debe incluir las operaciones necesarias para resolver la contencion y seleccionar un unico master del sistema.

Se puede ver una transferencia de informacion completa como una operacion de tres pasos tal como se muestra en la

figura 2.5. En un primer paso, esta la seleccion del master entre los procesadores que piden el uso del bus. Despues, este master selecciona un dispositivo al que va a transmitir la informacion. El ultimo paso el la transferencia de datos punto a punto entre el master y la unidad receptora. La operacion completa (arbitraje + direccionamiento + transferencia) sera llamada una TRANSACCION.

Las tecnicas para la seleccion del master en un sistema con un unico bus, pueden ser clasificadas de acuerdo con el mecanismo basico usado para evitar la perdida de informacion causada por la colision. Las posibilidades son:

1)Evitar colision evitando la contencion (token-passing)

2)Permitir la colision y evitar la perdida de informacion abortando el proceso y recomenzandolo luego (collision detection)

3)Evitar colision resolviendo la contencion entre peticiones (arbitraje)

La tecnica mas simple para identificar uno de los muchos ususarios, es una asignacion a priori del recurso. Un unico derecho de uso es concedido a cada usuario, sin considerar si este actualmente esta haciendo una peticion del recurso o no. El derecho puede ser visto como un TOKEN; solo la unidad que posea el token tendra acceso al recurso.

Con la segunda tecnica, cada peticionario testea el estado

del recurso (busy/free), y empieza a usarlo cuando este libre. Varios peticionarios pueden estar esperando usar el recurso al mismo tiempo, y por lo tanto, todos ellos detectan la liberacion al mismo tiempo. El uso multiple causa una colision, la colision es detectada y fuerza todos los usuarios para liberar el recurso y volver a intentarlo mas tarde. Esto es llamado un acceso multiple con muestreo de portadora y deteccion de colision (CSMA/CD).

Una tercera posibilidad para manejar multiples acceso es evitar la colision asignando el recurso solamente a uno de los peticionarios. Todas las unidades que tienen una peticion pendiente para el recurso, participan en un proceso de contencion que selecciona el proximo usuario. Este proceso es llamado ARBITRAJE, y el subsistema que lo ejecuta es el ARBITRO.

En el caso de sistemas altamente acoplados, el metodo empleado es el de arbitraje pues proporciona el mayor rendimiento. De ahora en adelante, cuando se hable de sistemas multiprocesadores, se hara referencia implicitamente a la tecnica de arbitraje.

La estructura logica de cualquier sistema de arbitraje esta basada en la traduccion de un vector de peticiones R, a un vector de concesiones G. Para evitar colisiones en el acceso al recurso, solo un elemento de G estara activo. Cuando el arbitro es implementdo por una unica unidad fisica, se llama centralizado. El arbitro centralizado mas simple es

un codificado de prioridad que activa solamente la peticion de mas alta prioridad. Un ejemplo de esto se vera en el arbitraje paralelo que se efectua en el multibus. Un arbitro tambien puede estar implementado por un conjunto de unidades identicas que son parte de cada peticionario. En ester caso, el arbitro tiene una organizacion descentralizada. Si, las unidades de arbitraje son identicas, puede clasificarsele como modular.

Varias son las distintas tecnicas para generar las concesiones. En un arbitraje centralizado, se requiere el doble de lineas de arbitraje que masters haya. Esta tecnica no debe emplearse en sistemas con bus comun (lo cual no quiere decir que no se use) debido a la centralizacion de las lineas. Los arbitros distribuidos usan estructuras de conexion modular como un bus.

Asi mismo, el algoritmo que concede el bus puede ser de varios tipos. Algunos algoritmos asignan un nivel de prioridad al master para caso de contencion decidir el usuario. Estos niveles de prioridad pueden ser fijos o variables. En caso de que sea fijo, un determinado procesador de baja prioridad, puede en condiciones de alta carga al bus comun, esperar indefinidamente por la liberacion del bus. Esto, que se conoce tambien en las colas de scheduling en sistemas multitasking, se denomina STARVATION ("morirse de hambre"). Algunas tecnicas han sido desarrolladas para evitar este efecto, denominandose FAIR. Se basan en la reasignacion dinamica de los niveles de prioridad. Con un arbitro de este

tipo, se puede definir el maximo retraso que un procesador soportaria hasta que se hiciera efectiva su peticion.

Una tecnica muy extendida en el arbitraje distibuido, es la llamada DAISY-CHAIN, como se muestra en la figura 2.6, en que las peticiones se acarrean sucesivamente a cada arbitro. La daisy-chain solo usa dos lineas independientemente del numero de procesadores en el bus, pero tiene como defecto, la posibilidad de producirse starvation, y el retraso de propagacion que lo hace inviable en grandes sistemas (en el multibus, se reduce a 3 masters). Asi mismo, un fallo en uno de los arbitros implicaria que los de mas baja prioridad quedarian excluidos.Con algo mas de hardware, se puede implementar un arbitraje tipo round-robin, que evita la starvation y garantiza un limite superior de tiempo de servicio. Una solucion adecuada para sistemas basados en bus, es el arbitro distribuido en el que cada arbitro participa en un proceso de contencion que es basado en la deteccion de la colision de un bit en las lineas del bus. La prioridad puede ser dinamicamente cambiada, y los otros masters pueden conocer quien esta a cargo del sistema. Es posible garantizar que un determinado peticionario recibira servicio en un tiempo conocido. Esta tecnica esta siendo empleada en los buses mas recientes que luego se detallaran.

Se trata de una unidad en cada arbitro conectada por un conjunto de lineas de prioridad y algunas lineas de control como se muestra en la figura 2.7. Cada una de estas unidades

estan compuestas de una red de autoseleccion de prioridad y un bloque de control. Cada bloque de control recibe una peticion de bus del procesador. Se encarga entonces de enviar los comandos apropiados a la red de resolucion de prioridad y a las lineas de control de arbitraje. La red de autoseleccion lleva a cabo el proceso que identifica el modulo de mas alta prioridad. Esta ultima unidad gana entonces el arbitraje y envia una concesion de bus a su procesador, que se convierte en el master del sistema. El arbitraje se conoce por la colision que produce el acceso de todos los procesadores en unas lineas a colector abierto que determinan el elemento de mas alta prioridad. Mas referencias en <1>.

Si la secuencia de arbitraje esta sincronizada por un reloj, recibira entonces el nombre de SINCRONA. En caso contrario (no existe una referencia temporal) recibe el nombre de ASINCRONA. El proceso de arbitraje puede desarrollarse antes o concurrentemente con un ciclo de acceso normal en el bus.

El comportamiento electrico de las lineas del bus es algo muy importante. Se debe examinar la compatibilidad de estados logicos a la entrada y a la salida de los dispositivos enganchados al bus. Se debe cuidar que el numero de dispositivos enganchados al bus caiga dentro de las prestaciones de fan-out. Hay que contar tambien con los retrasos capacitivos que pueden alterar la temporizacion de las lineas, y el efecto de la terminacion de las lineas en su impedancia caracteristica. Esto sin embargo, es bastante

dificil de analizar formalmente, y se suele emplear metodos empiricos.

Debe incluirse tambien el efecto de CROSSTALK entre lineas, debido a un acoplamiento inductivo y capacitivo entre las lineas, que aunque de bajo valor, muestra sus efectos a alta frecuencia. El crosstalk depende del espaciamiento entre lineas, del plano de tierra, de la redes de terminacion... Esto se puede convertir en un problema si el ruido inducido pudiera producir falsos niveles logico. Este error es extremadamente dificil de encontrar en un sistema en ejecucion y por lo tanto su prevencion debe ajustarse al periodo de diseno. Existen varias soluciones para limitar los problemas generados por el crosstalk. Se pueden emplear los llamados tranceivers trapezoidales. Estos proveen una alta inmunidad al ruido impulsivo pero limitando la velocidad de las senales. Otra posibilidad es la inclusion de pares trenzados o drivers diferenciales para minimizar el ruido radiado. Tambien se eliminar el ruido teniendo en cuenta que estos suelen estar localizados en el tiempo, con lo que para su eliminacion se procede a tomar la senal cuando esta este asentada. Los problemas producidos por falso triggering debido a senales activas por flanco, puede tener consecuencias desastrosas (tal como yo pude comprobar personalmente).

En ciertos casos es necesario que los procedimientos de transferencia puedan serndido para permitir el intercambio de

informacion entre tres o mas unidades. Los modulos implicados en la transferencia de la informacion podran tener velocidades diferentes, pero el protocolo debe garantizar una operacion correcta. Para que esto pueda darse, es necesario el conocimiento de dos hechos:

-Al menos un modulo ha respondido a la operacion?

-Han respondido todos los modulos a la operacion?

Estas dos condiciones son satisfechas empleando una operacion OR y una AND sobre las senales de activacion. En los buses se suele implementar con tecnicas de logica cableada (open-collector). Esta tecnica se denomina par de habilitacion desabilitacion (E/D ENABLE/DISABLE pair). Un protocolo usando esta tecnica, la activacion de una determinada condicion se determina porque una linea del par este activa, y la otra inactiva. Se puede entonces saber asi la primera activacion de la accion y la ultima activacion de la misma accion, mirando el estado del par de control.

2.8. CONTROL DE INTERRUPCIONES.

Una interrupcion implica una transferencia de informacion que no es generada por el master del sistema en ese momento. En un ambiente multiprocesador se pueden identificar varios tipos de interrupcion:

-de un dispositivo periferico a un procesador bien definido.

-de un periferico a todos los procesadores (ej. fallo suministro electrico)

-senalizacion especifica procesador a procesador (ej. paso de mensaje)

-senalizacion general procesador a procesador. El procesador que genero la interrupcion no sabe que dispositivo "se dara por aludido", y por tanto no lo puede direccionar.

Un sistema multiprocesador debe soportar varias tecnicas de interrupcion diferentes para permitir la implementacion de los tipos ya comentados. Los dos ultimos casos se emplean en contextos multiprocesador-multitarea de alta eficiencia y se soportan en hardware. En algunos casos, se emplea una linea serie para la implementacion de la interrupcion como es el caso del P896.

En la atencion de interrupciones, se genera un problema similar al del arbitraje previamente comentado. Se encuentra aqui tambien, un esquema centralizado o distribuido en la atencion de interrupciones segun se atiendan. En el capitulo 3, se veran distintos casos de arbitraje y respuesta a interrupciones.

2.9.LINEAS SERIE

Las lineas serie se incluyen para conseguir un mecanismo de transferencia independiente del bus paralelo, lo cual puede ser bastante necesario en caso de fallos. Los controladores de estas lineas serie son manejados por los procesadores como dispositivos I/O y se suele incluir una cierta capacidad de inteligencia a estos controladores. El

uso de estas lineas suele estar relacionado con protocolos de alto nivel. Un ejemplo de ello, son el P896 y el MULTIBUS II.

BIBLIOGRAFIA

<1>: ver referencia <1> en capitulo 1.

<2>: ver referencia <2> en capitulo 1.

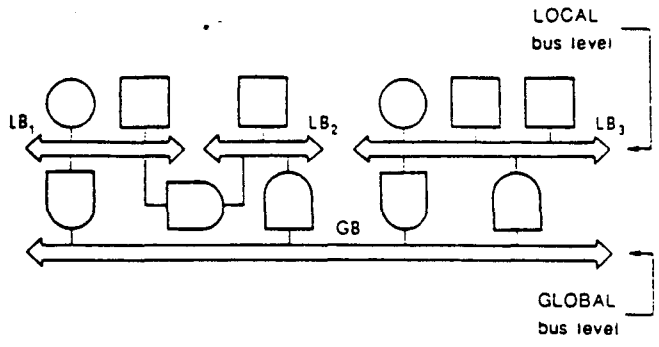


Fig. 2.1 - Bus hierarcy in a multiprocessor system

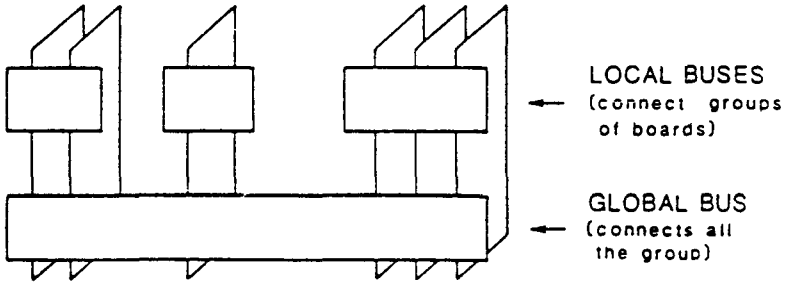


Fig. x.x - Mechanical structure for a hierarchical multiprocessor system.

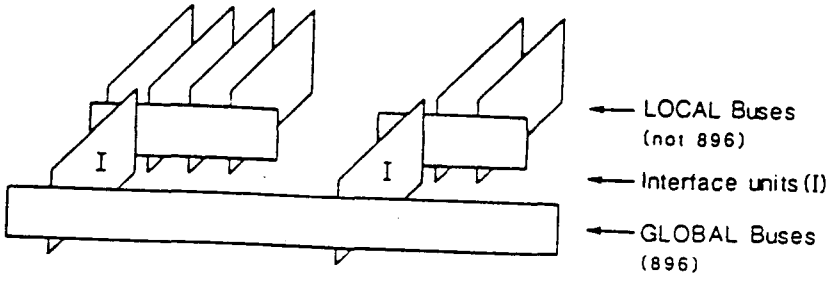


Fig. 2.2

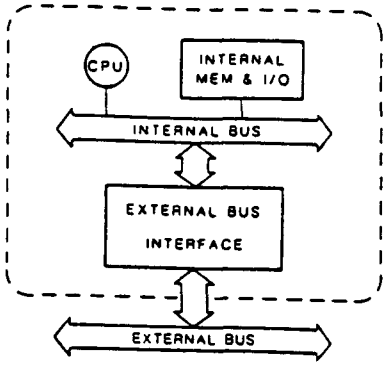


Fig. 2.3

- Organization of a processor board

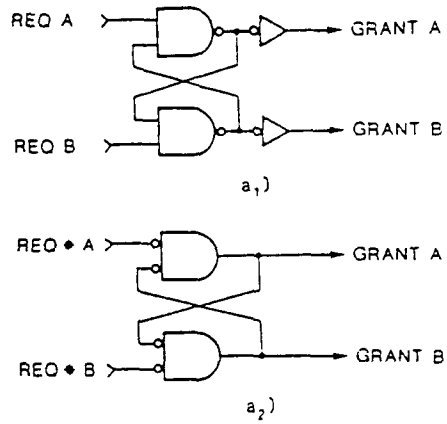


Fig 2.4
Arbitraje FCFS para dos masters.

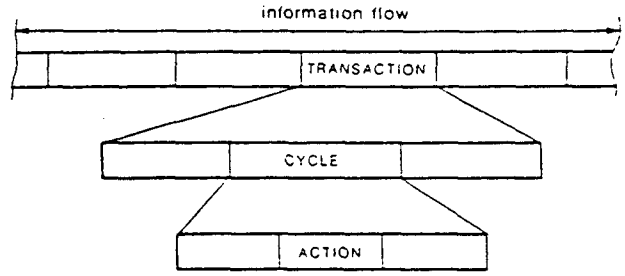


Fig. 2.5 - Hierarchy of bus operations.
Estructura jerarquizada de las operaciones en bus.

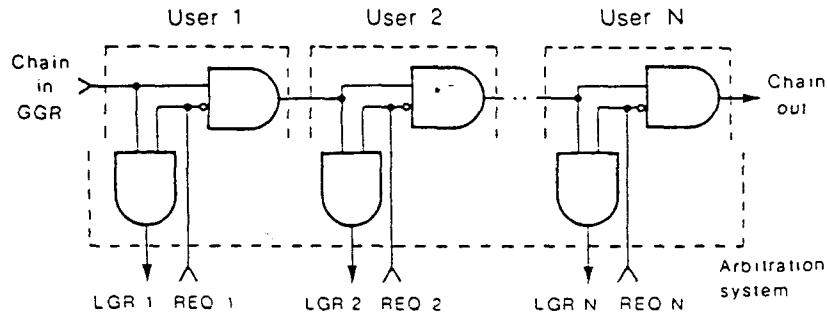


Fig. 2.6 - Distributed daisy-chain arbitration system.
Estructura del arbitraje serie, empleado por ejemplo en un esquema del sistema Multibus-I.

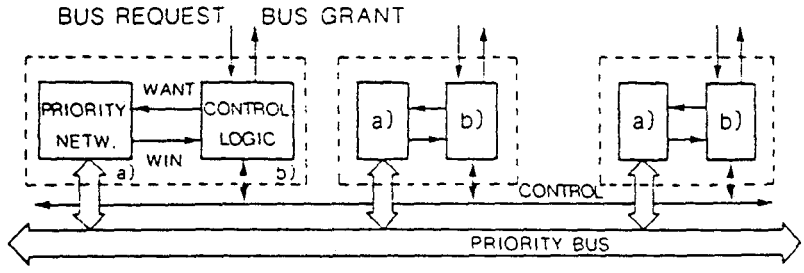


Fig. 2.7 - Distributed self-selection arbitration system.
a) priority network;
b) control logic

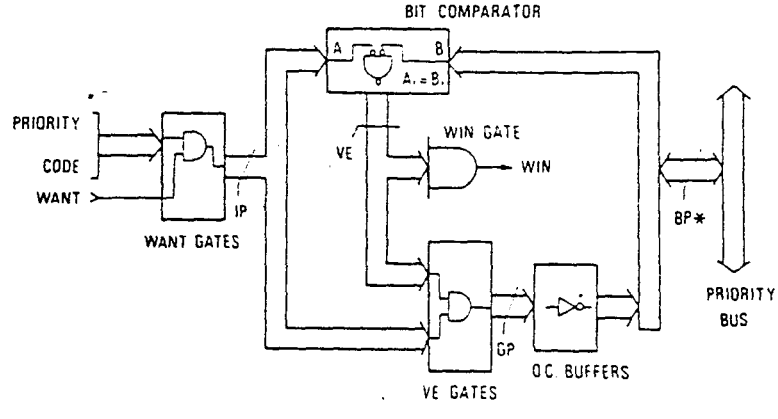


Fig. 2.8 - Block diagram of a self-selection network

CAPITULO 3

CAPITULO 3. EJEMPLOS DE BUSES ESTANDARD

3.1. INTRODUCCION

Se pretende en este capitulo el establecer una relacion entre las estructuras hardware descritas en el capitulo 2, con las actualmente estandarizadas. Se analizaran el VME, el P896 y el MULTIBUS I y II, a nivel puramente descriptivo.

3.2. VME

El bus estandard VME, fue desarrollado en 1981 por Mostek, Motorola y Philips/Signetics para soportar una nueva generacion de microcomputadores de 16 y 32 bits tales como la familia 68000 de Motorola. El objtetivo del diseno fue un bus de alta velocidad y altas prestaciones con manejo de interrupciones y capacidad multiprocesador.

Tres niveles diferentes de complejidad del bus han sido descritos por la especificacion: estandard, reducida, y extendida con distintos anchos de los buses de direcciones y datos. El nivel estandard tiene hasta 16 Megabytes de direccionamiento y 16 bits de datos; el nivel extendido tiene hasta 4 Gigabytes de direccionamiento con 32 bits de datos; el nivel reducido tiene una configuracion de 64 Kbytes de direccionamiento con un bus de datos de 16 bits. Mas de un master puede ser acomodado por el mismo bus, para el soporte de sistemas multiprocesadores. El mecanismo de arbitraje es del tipo "daisy-chain" multinivel.

En sus especificaciones fisicas, el estandard VME usa placas dobles tipo Euro-size con dos conectores DIN de 96

patillas tipo 41612, siendo las hembras las situadas sobre el bus. El rack es de 19", al igual que otros muchos sistemas. La figura 3.1 puede dar una idea de la estructura fisica del bus.

El conector P1 lleva todas las senales del bus. En las configuraciones estandard y reducida, se permiten placas con solamente el conector P1. El conector P2 lleva las lineas del bus extendido, y lineas definibles por el usuario para el soporte de I/O, en caso de que existan. Las placas que usen el conector P2 tienen restringida su posicion a una fija.

Hasta 22 placas pueden ser conectadas en el mismo segmento VME; el primer y el ultimo conector (A0 Y A21) estan reservados para los resistores de terminacion, y el primer conector libre (A1) se reserva para el arbitro del bus y el modulo de reset e inicializacion en encendido. Las senales son TTL estandard, ya sean en tri-estado, totem-pole o colector abierto, dependiendo de la funcion.

Cada transferencia de informacion sobre el VME, consiste en dos ciclos secuenciales: arbitraje, y direccionamiento y transferencia de datos. El primer ciclo es empleado unicamente en sistemas multimaster cuando un procesador pide el bus para convertirse en master del sistema. El arbitraje es siempre ejecutado antes del direccionamiento. Este direccionamiento es llevado a cabo concurrentemente con la transferencia de datos debido a su estructura no multiplexada. El arbitraje, siguiendo el esquema "daisy-chain", es efectuado con cuatros lineas de prioridad que atraviesan todos los masters. Existen otras dos lineas

(BBSY/,BCLR/). Hay varias politicas de arbitraje: daisy-chain simple, prioridad fija y prioridad rotativa. Con el primer esquema, la prioridad de los procesadores es fijada por su posicion geografica con respecto a otros masters y al arbitro. De esta manera, el slot A2 tiene la prioridad mas alta. Las lineas dedicadas entonces al arbitraje son:

-BR/<0..3> Bus request, semales daisy-chain (salida del master)
-BG/<0..3> Bus grant, semales daisy-chain (entrada al master)
-BBSY/ Bus busy, activa cuando el master del sistema haga uso del bus
-BCLR/ Bus clear, usado solamente con prioridad fija, informa que un master de mayor prioridad pide el bus

El ciclo de transferencia y direccionamiento usa conjuntos de lineas de diferentes para el control, las direcciones, y el dato. Las semales de control son:

-AS/ Address strobe

-DS0/ Data strobe 0:cuando esta activa, se habilita el byte impar de datos.

-DS1/ Data strobe 1:cuando esta activa, se habilita el byte par de datos.

-LWORD/ Long word:indica transferencia de 32 bits (modo extendido)

-WRITE/ especifica la direccion de la transferencia.

La informacion transmitida por el master del sistema, consiste en una direccion y 6 bits modificadores de direccion, que especifican el tipo de direccionamiento, el rango de direccionamiento (extendido o estandard) y el nivel

de privilegio. La linea de handshake DTACK/ informa que la actual transferencia ha acabado. BERR/ se activa para informar un error como por ejemplo un time-out.

El VME no tiene lineas de lock, para acceso exclusivo en varios ciclos, necesario para la implementacion de semaforos, pero puede ser emuladoi manteniendo AS/ activa.

Entre las caracteristicas especiales que posee, se incluye una estructura potente para el manejo de interrupciones. Para ello posee siete lineas de peticion de interrupcion IRQ<7..0>. Los modulos esclavos piden un servicio de interrupcion por esa linea. Un esquema daisy-chain es usado para resolver multiples peticiones. Mas de un master con posibilidad de atender interrupciones puede ser emplazado en el bus. Cuando un procesador recibe una interrupcion, se convierte en master del sistema a traves de un ciclo de arbitraje, y entonces efectua el reconocimiento de la interrupcion. El master declarara la interrupcion a ser servida con una operacion tipo direccionamiento y los modulos esclavos usaran un esquema daisy-chain para encontrar el modulo de mas alta prioridad. En el VME, existen dos lineas para un bus serie tambien, que se define como caracteristica opcional, al igual que otros buses. Tambien como opcional, se incluye la capacidad de transferencia en modo de bloque entre un master y un esclavo, pero debido a la naturaleza no multiplexada del bus, esta opcion no implica una mayor velocidad de transferencia. El patillaje del conector P1 del VME se muestra en la figura 3.2.

3.3.BUS 896

El bus P896 fue desarrollado por un comite de la IEEE con el proposito de definir un estandard para sistemas basados en microcomputadores. Sus objetivos en el diseño fueron los de obtener un bus independiente del procesador empleado, capaz de soportar sistemas multiprocesador, con control total distribuido, con transferencias de 32 bits, capacidad de deteccion de error, y una alta velocidad de transferencia con un protocolo asincrono.

Un cuidado especial se dio al comportamiento de las lineas de transmision en el bus. Las reflexiones, crosstalks, "quiños" en or-cableada y el comportamiento electrico com0leto fueron considerados en el diseño del protocolo y en la especificacion de los transceivers. El ultimo documento, con todas las especificaciones, salio en Noviembre de 1983.

El P896 emplea placas con estandard Euro-card, y conectores de 96 patillas DIN 41612, sobre racks de 19". El bus principal corresponde al conector A en la figura 3.3.

Solo existe una linea de alimentacion de 5 voltios; todos los demas voltajes, deberan obtenerse por conversion dc-dc en cada una de las placas. Solo se emplean drivers open-collector. Las especificaciones de los transceivers imponen unas caracteristicas que no permiten el empleo de dispositivos STTL o LSTTL, pero se puede emplear en diseños experimentales.

La transferencia de informacion consiste en tres ciclos secuenciales:

arbitraje, direccionamiento, y transferencia de datos. El arbitraje usa un conjunto separado de lineas, y puede ser efectuado concurrentemente con el direccionamiento y la transferencia de datos. La secuencia completa de las operaciones es mostrada en la figura 3.4.

Un limite superior para el servicio de las peticiones de bus esta grantizado en las aplicaciones en tiempo real porque cada modulo cuede competir otra vez por el control del bus, solamente si todas las peticiones pendientes anteriores han sido satisfechas. Cada unidad que pida el bus, entrega un codigo de prioridad en las lineas AN<6..0>/. AN6/ distingue entre unidades con prioridad y las que tienen un esquema "fair"; AN<5..1>/ es un identificador de modulo, derivado de la posicion geografica del slot; AN0/ es un flag de paridad. El secuenciamiento de las operaciones esta sincronizado con un handshake a 3 hilos, que usa las lineas AP/, AQ/, AR/.El protocolo de direcciones-datos, usa un handshake a 2 hilos para operaciones con un unico destino, o 3 hilos, cuando son multiples destinatarios. Las lineas usadas en el ciclos de direccionamiento son:

-AS/ address strobe
-AK/ address acknoledge
-AI/ inverted address acknowledge (solo en transferencia multi-destino)

La informacion transmitida en el ciclo de direccionamiento consiste en 32 bits, mas 5 bits de modo con el siguiente significado:

-CM4/ read/write
-CM3/ lock/unlock (para operaciones indivisibles)
-CM2/ transferencia simple/bloque
-CM1/ destino simple/multiple
-CM0/ reservado

El esclavo direccionado responde con un reconocimiento y una palabra de estado ST<2..0>, que especifica si la operacion pedida ha sido correctamente ejecutada.

Despues de esta respuesta, el master entra en el ciclo de datos. Las lineas de handshake para datos son:

-DS/ data strobe
-DK/ data acknowledge
-DI/ inverted data acknowledge (solo en multiple destino)

En los ciclos de datos, CM4/ distingue entre operaciones de lectura-escritura, mientras CM<3..0> son usados para determinar la posicion del byte.

En las transferencias de bloques, los flancos de DS/ y DK/ son empleados para mejorar la velocidad. La deteccion de errores es efectuada por medio de 5 bits de deteccion de error (ED), que protegen el dato o direccion y las lineas de comando, mas una linea de validacion de error (EV). Los bits de error son computados como paridad sobre byte.

El 896 no incluye lineas para el soporte de interrupciones. Se supone que cada placa tiene un procesador capaz de manejar los interfaces I/O locales y, asi, la mayoria del trafico en el bus, corresponde a movimiento de bloques de datos o mensajes interprocesador.

Dos lineas son reservadas para un bus serie. Este esta definido como una posibilidad opcional, no esencial para la operacion del sistema, y es una alternativa en caso de fallos. Tambien puede ser empleado como una utilidad generica

de comunicacion en el sistema. El nivel hardware del bus serie es capaz de soportar una amplia variedad de protocolos y tipos de mensajes. La especificacion 896 define diferentes formatos para mensajes, ya sean comandos de sistema, o paquetes definidos por el usuarios de longitud ilimitada, debido a un campo de tipo, que permite multiples usos de este bus, manteniendo la compatibilidad del protocolo estandard con algunas funciones.

Las lineas estan situadas en el conector de 96 patillas de tal manera que se minimizan los errores causados por el crosstalk. Las lineas activas a flanco, estan blindadas por lineas de tierra o estaticas, y los pares criticos, son mantenidos a cierta distancia.

El patillaje completo se muestra en la figura 3.5.

3.5. 3.4.MULTIBUS-I

El interface Multibus es un bus de proposito general que permite la interaccion entre dispositivos de un sistema. Permite que distintos procesadores puedan acceder a unos recursos comunes, teniendo para ello una interaccion tipo master-slave.Puede soportar hasta 16 masters y direccionar directamente hasta 16 megabytes de memoria. Esta estandarizado bajo la norma IEEE 796.

Debe tenerse en cuenta que el concepto de la arquitectura adyacente al Multibus ha cambiado con el paso del tiempo, de tal forma, que se ha reforzado su eficacia para el soporte de sistemas multiprocesadores, tratando de eliminar aquellos accesos inutiles al bus del sistema, que constituye el cuello

de botella del sistema. En la figura 3.6 se puede observar un resumen de la arquitectura tal como actualmente la considera el fabricante (Intel).

Como puede verse, el Multibus es tan solo uno de los buses del sistema, pero por el cual se manejan las principales interacciones entre los distintos dispositivos. No es intencion el describir completamente todos los buses adicionales que rodean al Multibus, sino mostrar la funcion y estructura de cada uno de ellos, y ver como contribuyen al sistema en general.

iLBX: LOCAL BUS EXTENSION.Su funcion es incrementar los recursos locales de un dispositivo del sistema .De esta manera se reducen accesos a recursos que actualmente pertenecen al sistema pero que tienen un uso puramente local. Asi esta extension provee memoria en acceso directo, sin arbitraje pudiendo soportar accesos DMA. Posee un alto ancho de banda (9.5 Mb/s para 8 bits, 19 Mb/s para 16 bits) y dispone la capacidad de intercambio a un master secundario. Puede llegar a soportar hasta 5 dispositivos por bus y direccionar hasta 16 Megabytes. Observese en la figura 3.7 como su conexion es de modo local, no extendiendose a todo el sistema.

MULTICHANNEL I/O BUS:Su funcion fundamental es el proveer de un camino separado para las transferencias DMA en las actividades de entrada-salida. Aislando las transferencias de I/O, el canal reduce las posibilidades de saturacion del bus del sistema. Pueden acceder hasta 16 dispositivos,

direccionando unos 16 Mbytes de memoria o I/O sobre cada uno. Estos dispositivos estan clasificados en una manera similar a la del IEEE 488. Conceptualmente existen 4 tipos: Supervisores, Controladores, Talkers, Listeners. La distancia maxima de conexion es de 15 metros. En la figura 3.7 puede verse su tipo de interconexion.

iSBX I/O EXPANSION BUS:En realidad este bus no contribuye en realidad con otra segunda canalizacion anexa al Multibus. Mas bien, pretende el conferir un standard de expansion para las placas de base iSBC. Actualmente esta bajo la norma IEEE P959. Intel fabrica modulos de (MULTIMODULE) con los mas comunes perifericos de entrada-salida. A traves de este es conectable el link BITBUS.

Este link es un bus serie optimizado para transferencia a alta velocidad de mensajes cortos en un sistema jerarquizado.Puede soportar hasta 250 nodos, con tres velocidades distintas, en modo sincrono o asincrono, y segun el modo hasta 4800 km (con repetidores) de distancia. Esta indicado, por ejemplo, para el control de cadenas de montaje con una gran distancia entre centros de control y accionadores.

3.5.MULTIBUS-II

El MULTIBUS-II es un nuevo bus de interconexion propuesto por Intel como soporte de una proxima generacion de procesadores VLSI de muy altas prestaciones. Estos nuevos procesadores (80386,80387,80286), ofrecen unas prestaciones muy superiores a modelos previos de tal forma que las

arquitecturas de bus anteriores tales como el MULTIBUS-I ya no ofrecen un soporte eficaz que aproveche todas las nuevas caracteristicas.

Veamos las caracteristicas que cumple un sistema basado en la arquitectura MULTIBUS-II. En la figura 3.8 puede observarse un diagrama de los distintos buses que cooperan, y tienen asignados funciones diferentes.

El MULTICHANNEL y el iSBX han sido ya comentados en la discusion sobre la arquitectura basada en el MULTIBUS-I. Como puede observarse, tiene una disposicion global bastante similar a la del MULTIBUS-I pero ahora, se ha tomado aun mas atencion en las caracteristicas del iLBX-II y el iPSB para adaptarlos mejor a las funciones que desenpeman. He aqui un resumen de cada uno de estos subsistemas.

iPSB (PARALLEL SISTEM BUS)

Constituye el fundamento de la arquitectura MULTIBUS-II y es el cauce por donde discurren los mensajes que sincronizan los distintos procesos que corren en varios procesadores. Incluye numerosas caracteristicas que priman directamente, el desarrollo de sistemas multiprocesador de muy altas prestaciones (ver comentario final). Este es un resumen de sus caracteristicas mas notables:

*Ancho de banda muy alto (40 Mbytes/s para bursts,20 Mbytes/seg para transferencias simples). Esto se consigue con componentes estandard, y las lineas estan disenadas para que los problemas de propagacion queden minimizados.

*4 gigabytes de direccionamiento (32 bits)

*Transferencias de 8-16-24-32 bits sobre un bus de 32 bits.

*Arbitraje potente con dos esquemas de asignacion de recursos y con capacidad de hasta 20 masters. El esquema "normal" no permite la "starvation" propia de arbitrajes anteriores que ocurre en sistemas con muchos procesadores. Asi todos los agentes tienen igualdad de derechos de acceso. Tambien es posible un esquema de alta prioridad similar a los esquemas anteriores, de forma que determinados procesadores puedan forzar el acceso. Esto podria ocasionar "starvation" en los procesadores de mas baja prioridad.

*Proteccion de paridad que permite seguridad en las transferencias. Asi mismo, se define un ciclo de excepcion para el caso de deteccion de errores en el bus, de tal forma que el sistema podria llegar a recuperarse. Esto constituye una base para sistemas con posibilidades de "fault tolerant".

*Altamente multiplexado, lo que implica un bus compacto con menos necesidad de drivers.Esto se corresponde con la funcion propia del iPSB de paso de mensajes fundamentalmente y no como bus de muy altas prestaciones desde el punto de vista de la CPU.

*Protocolo para paso de mensajes implementado en hardware, lo que implica un mayor rendimiento a la hora de la construccion de procesos altamente acoplados. Esta base hardware a una funcion normalmente ejercida en software, es una de las estructuras que mas difieren con respecto a buses anteriores con respecto al apoyo al multiproceso.

*Interrupciones potentes, con 255 posibles fuentes para 255 posibles receptores, lo cual supera la tipica limitacion de

buses anteriores proveyendo un esquema muy flexible.

*Posibilidad de identificacion software de los distintos modulos (basado en la posicion del slot), y de reconfigurar el hardware bajo control software (ejemplo: mapeado de la memoria).

*Estandard fisico eurocard.

iLBX-II (LOCAL BUS EXTENSION).

Al igual que el iLBX del MULTIBUS-I, este bus tiene por funcion, el canalizar los accesos locales hacia otra placa de memoria. Esto bus, al estar contacto directo con el procesador, requiere una arquitectura mas elaborada para efectuar transacciones a muy alta velocidad, que no limiten con estados de espera al procesador. Veamos sus caracteristicas fundamentales:

*Ancho de banda de 48 Mbyte/s.

*64 Megabytes de direccionamiento (26 bits).

*Transferencias de 8-16-24-32 bits sobre un bus de datos de 32 bits.Estas transferencias trabajan sobre un bus no multiplexado pensado para soportar un protocolo "pipelined", lo cual mejora sustancialmente el rendimiento ya que este bus soporta un mayor trafico y permite la implementacion de memorias CACHE. Asi mismo, este protocolo pipelined se corresponde con el nuevo protocolo de acceso a memoria en los nuevos procesadores (80386,80286).

*Capacidad de un intercambio de bus entre un master primario y un master secundario (mediante una tecnica similar al HOLD-HLDA).Soporta hasta 6 dispositivos.

*Inclusion de un ciclo de excepcion para prevenir y recuperar posibles errores.

*Posibilidad de identificacion y reconfiguracion hardware al igual que el iPSB.

iSSB (SERIAL SISTEM BUS)

El iSSB es una alternativa de bajo coste para el espacio de direccionamiento de mensajes del iPSB. El interface es identico en ambos buses a nivel software lo que permite una migracion suave entre distintos sistemas. Esto es bajo una limitacion de la eficiencia en los intercambios de mensajes, pero con un precio mucho menor a la solucion del iPSB.Como resumen de sus caracteristicas tenemos:

*Espacio de mensajes logicos equivalente al iPSB.

*2 Megabits/s de velocidad en el canal serie.

*Capacidad del multimaster hasta 32 nodos.

*Protocolo de acceso deterministico.

*Distribucion de hasta 10 metros.

*Deteccion de colision bajo el protocolo CSMA/CD.

Un ejemplo de arquitectura se ve en la fig 3.9

Se ha procedido a un analisis preliminar mas extenso que el de los otros buses. La razon radica en que esta clase de bus sera el que imponga la vanguardia en el desarrollo de las proximas arquitecturas de sistemas multiprocesadores de altas prestaciones. Estos nuevos procesadores de 32 bits (80386,80387) trabajan a unos 3-4 MIPS (y no son maquinas RISC; su codigo es compatible con el 8086) y unos 1.2 MFLOPS, lo que los coloca en el rango de los super-minis. El

desarrollo de un sistemas multiprocesador basado en estas nuevas series de CPUs permitiria el construccion de supersistemas.

BIBLIOGRAFIA DEL CAPITULO 3

<1>: ver referencia <1> en capitulo 1.

<2>: ver referencia <2> en capitulo 1.

<3>: ver referencia <12> en capitulo 1.

<4>: Intel Corp.
OEM systems handbook 1985.
Preliminar de la arquitectura Multibus II.

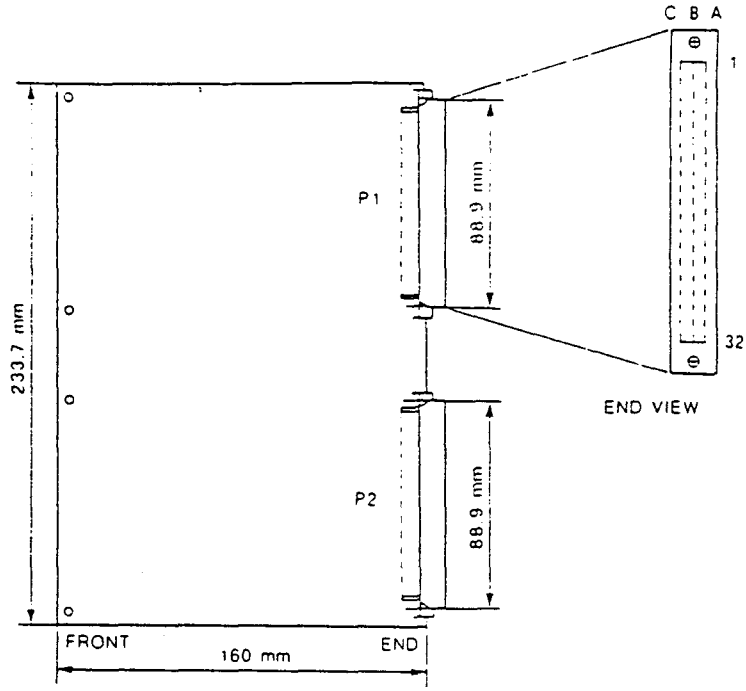


Fig. 3.1 - VME eurocard designation and pin numbering

	A	B	C
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	BG3IN*	SYSFAIL*
11	GND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5*
15	GND	BR3*	A23
16	DTACK*	AM0	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*	(SERCLK)	A17
22	IACKOUT*	(SERDAT)	A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12	+5 STDBY	+12
32	-5	+5	+5

Fig. 3.2 - Pinout of VME bus

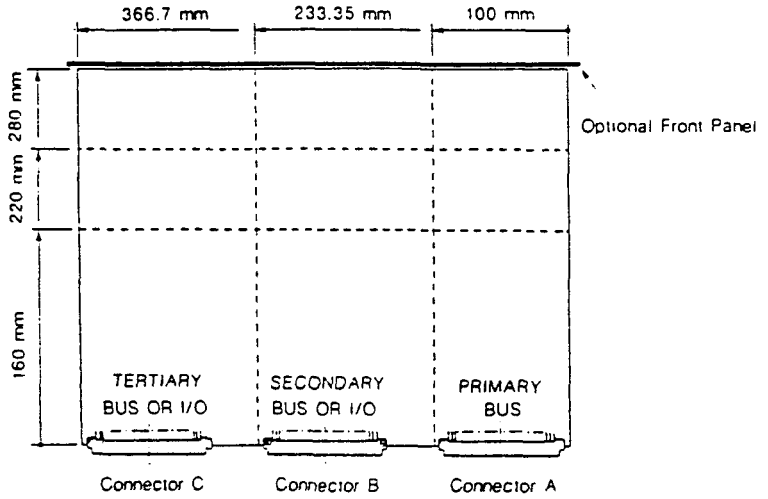
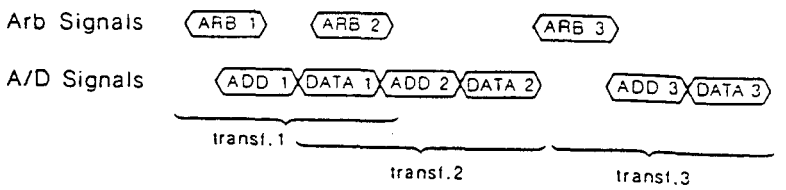


Fig. 3.3 - 896 board family



- Examples of 896 bus operations, showing sequencing of arbitration and A/D transfers

Fig. 3.4 Ciclo de bus en el standard 896

	A	B	C
1	GND	GND	GND
2	+5	+5	+5
3	AD0*	AD1*	AD2*
4	AD3*	GA0*	AD4*
5	AD5*	AD6*	AD7*
6	GND	ED0*	AD8*
7	AD9*	AD10*	GND
8	AD11*	AD12*	AD13*
9	AD14*	GA1*	AD15*
10	ED1*	AD16*	AD17*
11	GND	AD18*	AD19*
12	AD20*	AD21*	GND
13	AD22*	AD23*	ED2*
14	AD24*	GA2*	AD25*
15	AD26*	AD27*	AD28*
16	GND	AD29*	AD30*
17	AD31*	ED3*	GND
18	CM0*	CM1*	CM2*
19	CM3*	GA3*	CM4*
20	CP*	EV*	ST0*
21	GND	ST1*	ST2*
22	AS*	AK*	GND
23	AI*	DS*	DK*
24	DI*	GA4*	AP*
25	AQ*	AR*	AC*
26	GND	AN0*	AN1*
27	AN2*	AN3*	GND
28	AN4*	AN5*	AN6*
29	SB0*	RE*	SB1*
30	RFU0	RFU1	RFU2
31	+5	+5	+5
32	GND	GND	GND

Fig. 3.5 Patillaje del bus 896. La disposición de las líneas es tal que minimiza los efectos de crosstalk y similares.

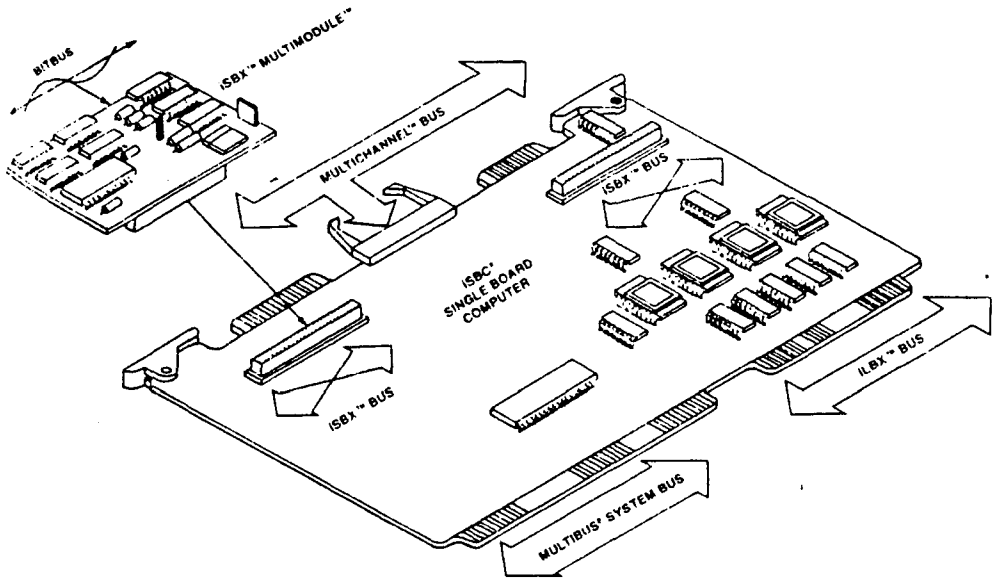


Fig. 3.6 Localización física de los buses en la arquitectura Multibus.

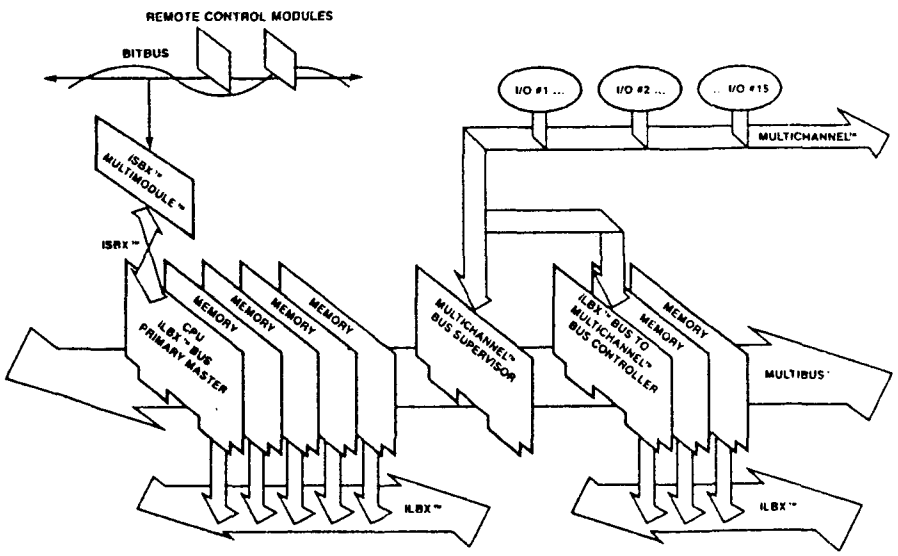


Fig. 3.7 Extensión lógica de los buses en la arquitectura Multibus. Observar disposición local del iLBX. El link bitbus permite la conexión a otros sistemas de control. Las actividades de I/O son descargadas por el multichannel. El iSBX permite flexibilidad en la configuración de recursos de las placas master.

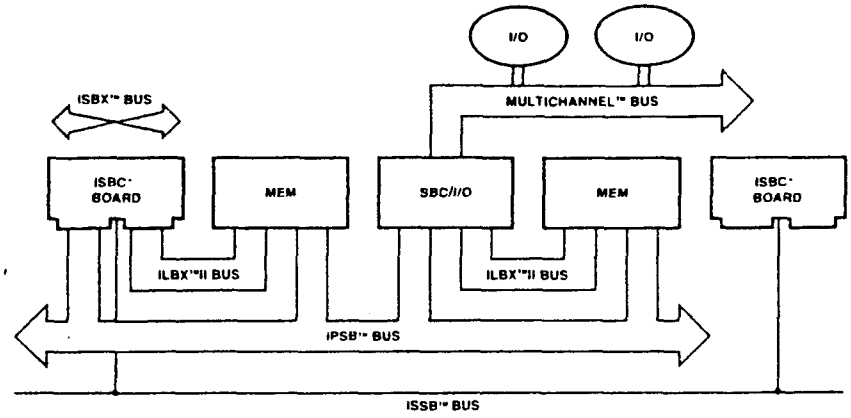


Fig. 3.8 Extensión lógica de los buses en la arquitectura Multibus -II. Ver similitud funcional con Multibus-I. La única novedad es la incorporación del bus serie iSSB.

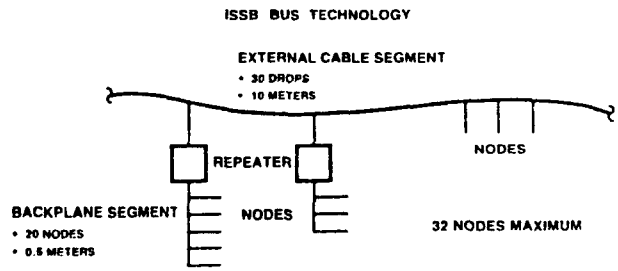


Fig. 3.9 Estructura que puede adoptar el iSSB. Su estructura lógica es equivalente al iPSB del Multibus-II. Esto significa que no hay diferencias software. Esta alternativa de bajo coste al iPSB, baja las prestaciones ya que el bus es serie, pero esto es transparente a la programación. Se puede construir un sistema jerarquizado según la necesidad de comunicaciones entre diversos procesadores.
El iSSB puede conexionar asi, varias agrupaciones de sistemas multiprocesadores. Puede tambien servir de salida de emergencia en caso de error en el iPSB, con lo que se da una cierta tolerancia al error.

CAPITULO 4

CAPITULO 4. ANALISIS DE PRESTACIONES EN SISTEMAS MULTIPROCESADORES.

4.1 INTRODUCCION

El analisis de las prestaciones (performance) de un sistema computador constituye un elemento esencial para la determinacion de la eficiencia de una arquitectura hardware en particular. Esto se aplica con mas razon, al caso de sistemas multiprocesadores, cuya arquitectura ha sido concebida especialmente para un aumento de la capacidad de computacion que en otras condiciones, seria dificil o costoso alcanzar.

Las ventajas de los sistemas multiprocesadores sobre los monoprocesadores, solo pueden ser aprovechadas si se encuentran las siguientes condiciones:

*El problema computacional es susceptible de una descomposicion en subprocesos ejecutables en paralelo, que se adapte convenientemente a la estructura distribuida del computador.

*La sobrecarga (overhead) impuesta por la cooperacion entre diversos procesadores es mantenida a un bajo nivel.

Cada uno de estos puntos, tiene sus propios problemas. Destacando ahora el primer punto, tenemos las siguientes preguntas.

-Hasta que punto, los algoritmos "normales" poseen paralelismo?

-Debe expresarase el paralelismo explicitamente, o el compilador debe extraerlo?

-Cual es la distribucion de procesos ejecutables en paralelo?

-Que uniformidad poseen estos procesos paralelizables?

-Se puede establecer una correlacion eficiente entre arquitectura y proceso?

Las respuestas a estas preguntas son importantes. De nada sirve un sistema multiprocesador con 16 procesadores, si los algoritmos a ejecutar solo poseen un grado reducido de paralelismo. Por otro lado, si los procesos a ejecutar en los distintos procesadores son muy dispares en relacion con la cantidad de memoria usada, tiempo de ejecucion, llamadas a I/O o volumen de sincronizacion-comunicacion, la eficiencia del sistema puede quedar gravemente damada. Asi pues, un sistema regular desde el punto de vista hardware, no encontraria una correspondencia de regularidad con el software. La importancia de la correlacion entre estas dos magnitudes es importantisima para conseguir las maximas prestaciones, tal como se ha demostrado en las investigaciones sobre arquitecturas RISC. La comunicacion entre procesos es un autentico cuello de botella. El soporte hardware que se presta a esta comunicacion , condiciones las prestaciones globales del sistema, pero como se vio ,la interconexion tiene un precio, mayor cuanto mas capacidad tenga el canal y menos contencion se imponga. Es necesario, por tanto, fijar en que rango se registra esta comunicacion entre procesos y desarrollar una arquitectura que se adapte a estas estadisticas. Hasta ahora, no he conseguido ningun tipo de informacion sobre estudios que se hayan afrontado al respecto de la cantidad de paralelismo extraible y su

correspondencia con una estructura hardware determinada.

Con respecto al segundo punto que se especifico en las condiciones iniciales en donde se establece la sobrecarga debido a la cooperacion entre procesos, podemos identificar dos factores:

1)La cooperacion entre procesos es manejada por un programa que no efectua trabajo "util" (como los sistemas operativos).

2)La contencion por el uso de un numero limitado de recursos que implica una perdida de computacion por estados de espera.

Aun la importancia del punto anteriormente tratado, la mayoria de los estudios realizados sobre las prestaciones, trabajan unicamente la degradacion debido a interferencia en el acceso a memoria. Algunos estudios se extienden al nivel de ejecucion de instrucciones. Otros emplean un punto de vista mas abstracto asumiendo operaciones asincronas correspondientes a la ejecucion de diversos segmentos de programas. Algunos estudian el empleo de una red cross-bar en la red de interconexion, mientras que otros emplean otros tipos de redes. En <1> se analiza ligeramente los resultados obtenidos en el estudio del Cmmp. En <2> se analizan consideraciones de ancho de banda en una red de interconexion. En <3> y <4> se estudia la degradacion en un multiprocesador con un unico bus compartido con distintas arquitecturas. Debido a que este fue el sistema desarrollado en este proyecto, se expondran estos resultados.

Las herramientas empleadas para la extraccion de resultados pueden agruparse en dos grupos principales que

emplean:

-Simulacion

-Analisis teorico.

En el primer caso, se evalua un modelo de comportamiento que se simula en software. Por supuesto hay que tener en cuenta las limitaciones de una simulacion pero es una herramienta, de facil uso. Se puede establecer un resultado midiendo una arquitectura con diversos programas que se asemejen en esu comportamiento a programas de uso comun. La posibilidad del analisis teorico, con todas las ventajas y desventajes que el analisis formal trae consigo, se realiza normalmente con redes de Markov, teoria de colas, o variantes de estas ultimas. Una red de Markov es, resumiendo, un tipo de automata de funcionamiento no deterministico sino estocastico. La teoria de colas analiza los retrasos debido a esperas por recursos. Las variantes pueden ser redes de Petri u otras, que se emplean tambien en la especificacion de protocolos. Esto no es un capitulo sobre modelacion de sistemas, sino que se pretende exponer principalmente los resultados obtenidos en <3> y <4>. Mas detalles en la bibliografia.

4.2 RESUMEN DE LOS RESULTADOS OBTENIDOS POR EL GRUPO DE TORINO.

Se planteo el analisis de cuatro arquitecturas basadas en un unico bus. Estas arquitecturas poseen memoria local y memoria privada ademas de la memoria del sistema (global). Los programas residen en memoria privada. Se asume un

contexto multitarea en cada procesador, y el paso de mensajes se realiza mediante las primitivas "send" y "receive" que se describiran en el capitulo 5. El unico bus se emplea casi exclusivamente para el paso de mensajes y asi la carga que se somete al bus es proporcional al numero de mensajes generados por procesadores y tareas.Un procesador estaria en uno de los siguientes cuatro estados:

-ACTIVO: El procesador ejecuta en su memoria privada.

-ACCEDIENDO: El procesador intercambia mensajes con otros procesadores.

-EN COLA: El procesador esta esperando para acceder a memoria global.

-BLOQUEADO: El procesador esta bloqueado porque otro procesador accede a su memoria local.

Los parametros que emplea son la longitud media de un "burst" de cpu, y la duracion media de la transferencia. La comunicacion se realiza por puesta de mensajes en un puerto de entrada de tarea. Las cuatros arquitecturas, que se pueden ver en la figura 4.1 tienen la siguiente estructura:

ARQUITECTURA 1.

La primera arquitectura considerada, esta caracterizada por la existecia de una memoria externa comun a todos los procesadores y accesible unicamente a traves del bus global. Se produce contencion cada vez que un mensaje es leido o escrito en memoria.

ARQUITECTURA 2.

La memoria comun esta distribuida en modulos locales a cada

procesador. Se asume que las memorias locales estan logicamente divididas en areas privadas y comunes. Las areas comunes contienen los puertos de comunicacion de los procesadores asociados. Cada procesador esta conectado a su propio segmento de memoria por un bus local. Un procesador accede a un segmento de memoia no local usando su bus local, el bus global, y el bus local conectado al modulo de memoria comun de destino donde el puerto de entrada del procesador de destino esta localizado. Se necesita arbitraje para manejar buses globales y locales. Se produce contencion cada vez que se usa un bus. Un procesador que consigue acceder al bus global adquiere prioridad para usar cualquier recurso alcanzable y puede desalojar a otros procesadores. Los procesadores desalojados cuando estaban en estado activo quedan bloqueados. Los que estan en estado de espera liberan el bus local y mantienen su estado. Esta politica evita deadlock (ver capitulo 5) y mejora las caracteristicas.

ARQUITECTURA 3.

Consiste en una mejora de la arquitectura 2, usando memorias de doble puerto para implementar la parte comun de la memoria local de cada procesador. Los modulos de memoria comunes son asi directamente accesibles desde procesadores externos a traves del bus global. No se genera contencion en los buses locales o en la memoria de doble puerto que soporta dos accesos simultaneos. Contencion es en este caso solo debida a la comparticion del bus global.

ARQUITECTURA 4.

Esta es una variacion de la arquitectura 3, para el caso de

la no posibilidad de memoria de doble puerto. Como en la arquitectura 3, los procesadores no acceden a los segmentos de memoria comun asociados a cada procesador. Solo un procesador accede al modulo de memoria comun cada vez. La contencion se genera por el uso del bus global y los modulos de memoria comun. Se necesitan mecanismos de arbitraje para el manejo del bus global y los buses de memoia comun. Se da prioridad a las peticiones provenientes del bus global y asi, un procesador accediendo a su modulo de memoria sociada podria ser asi desalojado.

Pasamos a comentar los resultados a la vista de las graficas presentadas en las figuras 4.2 en adelante.

La figura 4.2 muestra la eficiencia de la computacion (se define como poder computacional el tanto por ciento de tiempo que el procesador esta activo) con respecto a la relacion de mensajes producidos para las cuatro arquitecturas en un sistema con dos procesadores. Para bajos indices de carga, la arquitectua 1 mejora el comportamiento de la arquitectura 2 debido al hecho que con baja carga, en la arquitectura 1, el retraso medio debido a la situacion "en cola", es muy baja con lo que la contencion es despreciable. Con la arquitectura 2, cada vez que se accede a un modulo de memoria comun externo, se desaloja un procesador que en condiciones de baja carga, tiene una gran posibilidad de estar activo. Para cargas mayores a 0.5, la arquitectura 2 ya se manifiesta ventajosa respecto a la arquitectura 1. La zona de baja carga debe ser considerada como muy significativa debida al hecho

que en los sistemas multiprocesadores bien disenados, se operaria en esta region ya que la descomposicion de las tareas y la localizacion de estas en los procesadores esta pensada para que se reduzca la sobrecarga debida a la comunicacion. Asi, el comportamiento a baja carga de las arquitecturas 1,3,4 no difieren con mucho y el bus del sistema no se consideraria como el cuello fundamental.

Considerando sistemas mas complejos (vease fig 4.3), se puede observar la eficiencia de un sistema de cinco procesadores organizado en las cuatro arquitecturas. En fig 4.4 se muestra lo mismo para un sistema de diez procesadores. El incremento del numero de procesadores hace que el comportamiento de las arquitecturas 2,3,4 se vuelva muy similar, despreciable en el caso de muy baja carga. El comportamiento similar de estas arquitecturas podria ser intuitivamente explicado por el efecto de cuello de botella en el bus global; en estas condiciones, los procesadores estan principlamente esperando en cola para acceder al bus global y otros fenomenos de contencion y bloqueo desaparecen. Cuando la carga aumenta, la arquitectura se comporta significativamente peor que las otras. En la figura 4.5 se muestra el poder computacional de las arquitecturas 1,2,3 para una carga fija de 0.1. Como se observa, la arquitectura 1 es una mala eleccion para grandes sistemas. Las prestaciones de la arquitectura 4 estara siempre entre la 2 (bajo) y 3 (alto).

Cuando la carga aumenta se puede observar que la contencion en el bus global

casi anula la ventaja esperada de amadir nuevos procesadores. Esto es aun mas evidente para la arquitectura 1. En resumen, para todas las arquitecturas, el aumento de poder computacional debido a nuevos procesadores solo se produce para cargas de comunicacion bajas. Estos resultados han explicitamente despreciado perdidas de prestaciones debido a la sincronizacion entre tareas y/o procesadores. En grandes sistemas multiprocesadores, la asuncion de los procesadores ejecutando un flujo continuo de instrucciones, que el numero de tareas se mucho mayor que el numero de procesadores, no podria esta justificado.

Estos resultados fueron obtenidos mediante modelos formales que explicitan el comportamiento del sistema bajo un modelo markoviano. Muchas de las hipotisis introducidas fueron conservadoras tales como la distribucion exponencial de tiempo de servicio, los esquemas de asignacion de recursos, y la distribucion uniforme de la carga computacional entre todos los procesadores.

Las mediciones realizadas en un prototipo con arquitectura 4 cuadran con las predicciones obtenidas de los modelos formales. Detalles en <4>

BIBLIOGRAFIA CAPITULO 4

<1>: Phil C. C. Yeh y otros.

Shared cache for multiple-stream computer systems.

IEEE Transactions on computers. Enero 1983.

<2>: ver referencia <7> capitulo 1.

<3>: M. Ajmone y otros.

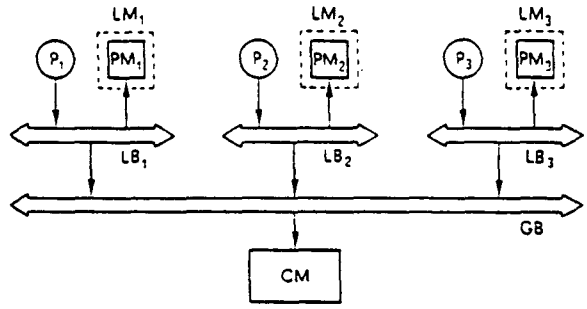
Comparative analysis of single bus multiprocessor architectures.

IEEE Transactions on computers. Diciembre 82.

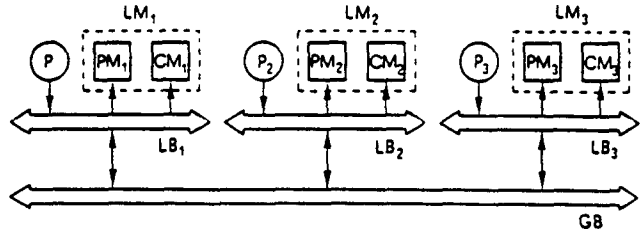
<4>: M. Ajmone y otros.

Modelling bus contention and memory interference in a multiprocessor.

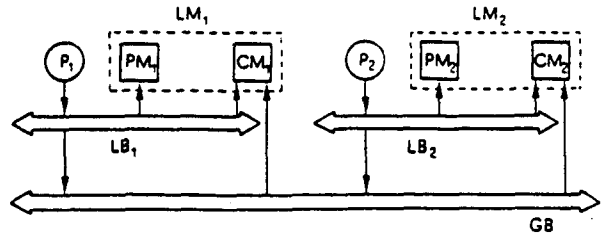
IEEE Transactions on computers. Diciembre 82.



- Structure of architecture 1 with single common memory and 3 processors composed of CPU (P_i) and private memory (PM_i) connected by a local bus (LB_i).



- Structure of architecture 2 with distributed common memory (CM_i)and 3 processors composed of CPU (P_i) and associated private memory (PM_i).



- Structure of architecture 3 with a distributed common memory (CM_i) and 2 processors composed of CPU (P_i) and local memory (LM_i) conected by a local bus (LB_i).

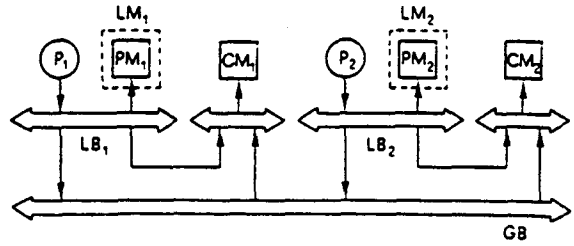


Fig. 4.1

- Structure of architecture 4 with distributed common memories (CM_i) and 2 processors composed of a CPU (P_i) and private memory (PM_i) connected by a local bus (LB_i).

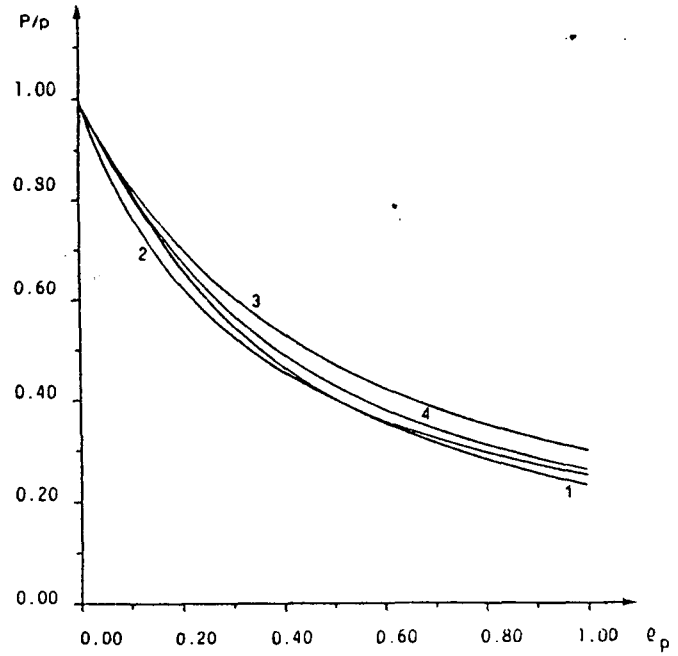


Fig. 4.2 - Two processor system, normalized processing power vs. ϱ_p. Architectures 1 - 4.

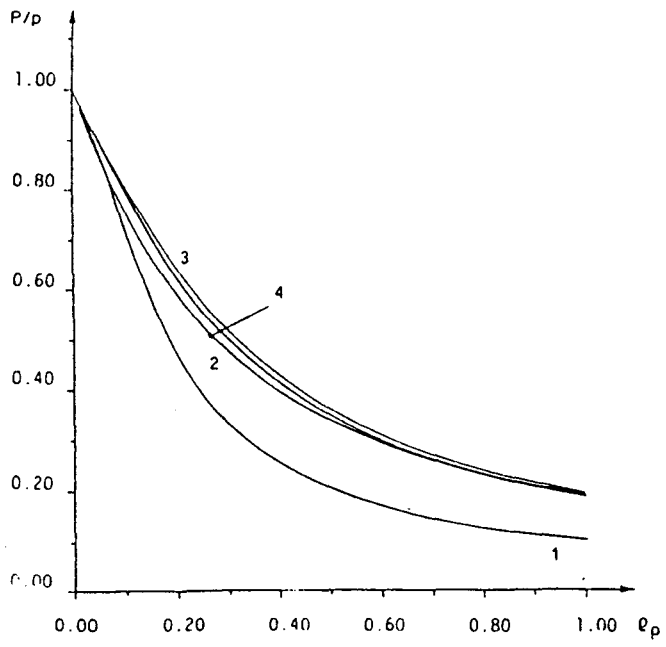


Fig. 4.3 - Five processor system, normalized processing power vs. ϱ_p. Architectures 1 - 4.

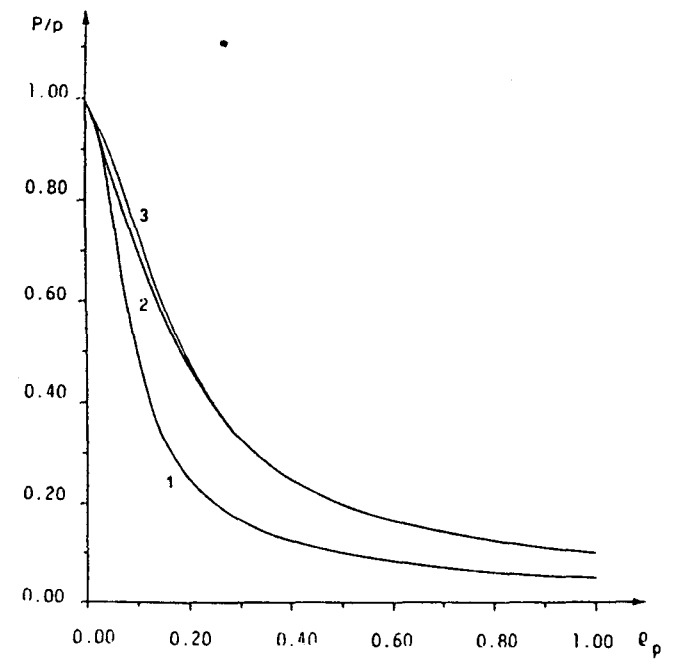


Fig. 4.4 - Ten processor system, normalized processing power vs. ϱ_p. Architectures 1 - 3.

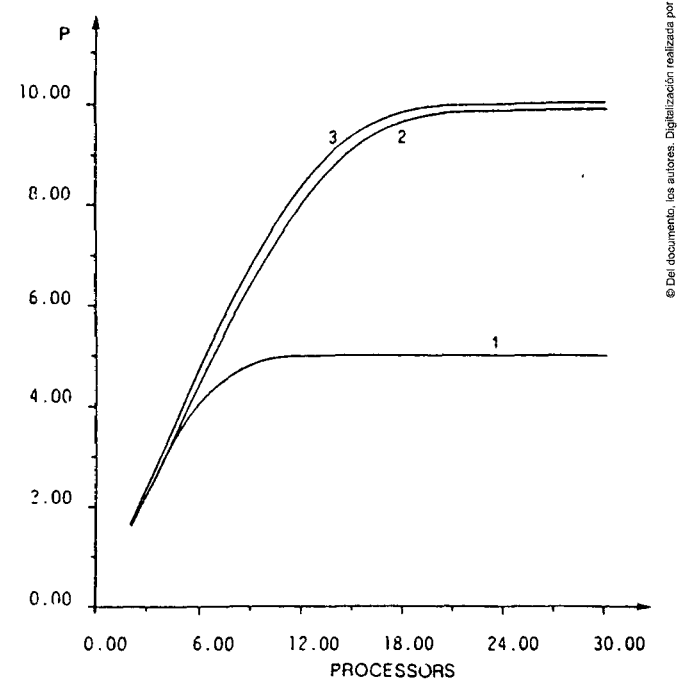


Fig. 4.5 - Processing power vs. number of processors for ϱ_p = 0.1 in the case of architectures 1, 2 and 3.

Degradación de prestaciones según carga y número de procesadores.

CAPITULO 5

CAPITULO 5. EL SOFTWARE EN SISTEMAS MULTIPROCESADOR

5.1. INTRODUCCION

Como se ha visto en el capituo 1, muchos de los problemas mas importantes que se encuentran en los sistemas multiprocesadores, corresponden al software. Existen dos tendencias basicas en la comunicacion de procesos:

1) envio de mensajes.

2) acceso exclusivo a variables.

Cada esquema tiene sus ventajas e inconvenientes. Aqui se tratara sobre todo del segundo esquema, que suele emplearse en sistemas con extraccion de paralelismo. No se introducen ejemplos que se remiten a la bibliografia recomendada. Se exponen brevemente ciertas construcciones de alto nivel usadas en ciertos lenguajes. Para ver ejemplos de acceso exclusivo, remitirse a <1>. En muchos casos, se ve mejor analizando "contraejemplos", o sea, sistemas que no funcionan debido a la falta de acceso exclusivo a sus variables (region critica).

Existen otros problemas tales como el de la estructura del sistema operativo o los algoritmos de scheduling. No he encontrado bibliografia que hable particularmente de estos problemas pero el lector puede remitirse a <4> y <5> en donde se tratan diversos ejemplos. En <6> y <7> se pueden observar implementaciones que se soportan en sistemas multimicro.

5.2.EXTRACCION DE PARALELISMO

La mejora de velocidad de ejecucion de un programa puede lograrse logrando la extraccion de sentencias que, aun formuladas secuencialmente, puedan ejecutarse en paralelo, es decir, simultaneamente. Esto puede ser formalizado con las siguientes restricciones.

Condiciones de Bernstein:

Para 2 sentencias S1,S2 sucesivas, puedan ejecutarse concurrentemente y dar el mismo resultado, debe cumplirse las siguientes condiciones:

1. R(S1) ∩ W(S2)=(Ø)
2. W(S1) ∩ R(S2)=(Ø)
3. W(S1) ∩ W(S2)=(Ø)

Siendo "(Ø)" el conjunto vacio; W(S) el conjunto de variables de la sentencia S que son escritas; R(s) el conjunto de variables de la sentencia S que leidas. La extraccion de paralelismo tiene pues un antecedente formal pero en la mayoria de los compiladores que generan esta clase de codigo, el paralelismo ha de explicitarse (por ejemplo con las construcciones parbegin, parend).

5.3.SOPORTE HARDWARE PARA LA IMPLEMENTACION DE REGIONES CRITICAS.

Hay fundamentalmente dos estructuras que se implementan para la construccion de regiones criticas en sistemas multiprocesador. Veamos la primera:

```
function TEST-AND-SET (var target:boolean): boolean;
  begin
   TEST-AND-SET:=target;
   target:=true;
  end;
```

La funcion carga en TEST-AND-SET el valor de target. Observese que se supone que no han sido implementados otras herramientas de sincronizacion. Target se pone entonces a true, indicando que en caso de que no se estuviera en una region critica, ahora se empezara una. Estas dos instrucciones se ejecutan en un ciclo de memoria continuo, es decir, que ningun otro proceso tendra posibilidad de acceder a la variable target mientras se este efectuando la funcion. Esta capacidad se resuelve por el hardware que no deja la entrada a otro procesador, independientemente de la prioridad de este.

La exclusion mutua usando la funcion TEST-AND-SET se podria implementar como:

```
repeat
     while test-and-set(lock) do;/*nothing*/end;
          :
     seccion critica
          :
     lock:=false;
          :
     seccion no critica
until false;
```

Este metodo de sincronizacion se emplea en el procesador de entradas y salidas 8089 con la instruccion TSL.

Otra forma de implementacion es la que se describe seguidamente:

```
procedure SWAP(var a,b:boolean);
  var temp:boolean;
  begin
    temp:=a;
    a:=b;
    b:=a;
  end
end;
```

Como puede facilmente verse, la funcion de este procedimiento es intercambiar unicamente el valor de la variable a con la de b. Este procedimiento tambien necesita que el acceso sea exclusivo con esas dos variables.

La exclusion mutua podria implementarse entonces como:

```
repeat

    key:=false;
    repeat
       SWAP(lock,key);
    until key=false;
          :
    seccion critica
          :
    lock:=false;

    seccion no critica

until false;
```

Notese que inicialmente la variable global lock esta a false. Este es el procedimiento que emplea el 8086 y el resto de sus compatibles para la implementacion de secciones criticas. Se implementa con una instruccion:

```
LOCK XCHG     A,B
```

El lock es el responsable de el acceso exclusivo a las variables A y B durante la ejecucion de la instruccion XCHG.

5.4.ESTRUCTURAS PARA IMPLEMENTACION DE ACCESO EXCLUSIVO

Un lenguaje debe proporcionar medios para evitar errores

de sincronizacion provenientes de la misma estructura de los programas. Estos errores pueden ocurrir si varios procesos concurrentes se comunican entre si por medio de variables comunes o por mensajes explicitos. Obviamente, se podria no permitir esta comunicacion pero esto reduciria la posibilidad de concurrencia que el sistema operativo podria soportar. Es necesario entonces, llegar a un compormiso donde lo que se intenta es tratar de eliminar estos errores con estructuras del lenguaje que contribuyan a su disminucion.

REGIONES CRITICAS.

Los semaforos son elementos utiles pra resolver de forma efectiva el problema de la seccion critica, asi como esquemas de sincronizacion arbitraria. Todos los procesos comparten una variable semaforo de exclusion mutua "mutex", que inicialmente esta a 1. Cada proceso debe ejecutar una operacion de "cierre" del semaforo mediante la operacion P(mutex) antes de entrar en la seccion critica, y V(mutex) a la salida de ella. Esta secuencia es muy importante, pues puede originar errores de sincronizacion dependientes del tiempo o originar un deadlock (punto muerto). Un ejemplo de estos problemas:

Supongase que por un error, se intercambian las operaciones sobre el semaforo mutex, o sea:

```
V(mutex);
   :
   :
seccion critica
   :
   :
P(mutex);
```

Aqui, podria originarse que al no implementarse correctamente la seccion critica se origine un problema de requerimientos. El error seria dependiente del tiempo y en ocasiones no seria reproducible.

Veamos un error de anidamiento:

```
P(mutex);
   :
seccion critica
   :
P(mutex);
```

En tal caso se produciria un deadlock, ya que se espera la liberacion de la variable sin que haya la posibilidad que esta pueda liberarse.

Estos problemas de estructura en los semaforos indujeron a la introduccion de una estructura de lenguaje nueva: La seccion critica. En esta estructura se declara la variable compartida como "shared" de tipo T, o en otras palabras:

```
var v: shared T;
```

El acceso a la variable v esta condicionado a que este se proceda dentro de una region critica de la forma:

```
region v do S;
```

Esta condicion quiere decir que mientras la sentencia S se esta ejecutando, ningun otro proceso puede acceder a la variable v. De este modo, las sentencias:

```
region v do S1;
   :
   :
region v do S2;
```

aun cuando se ejecuten concurrentemente, no habra problemas con el acceso a la variable v.

La estructura de region critica evita algunos errores simples asociados a la los semaforos pero pueden haber ciertos problemas. El anidamiento de regiones criticas puede ocasionar deadlocks como se prueba en el siguiente ejemplo:

```
var x,y: shared T;
parbegin
  Q:region x do region y do S1;
  R:region y do region x do S2;
parend;
```

Si Q y R entran aproximadamente al mismo tiempo, la variable y y la variable x podrian estar "cerradas" por estar dentro de una region critica, cuando se pide posteriormente el acceso a x e y.Esto implica que el proceso Q, este esperando por la liberacion de y cuando el tiene la variable x cerrada, y asi mismo, el proceso R, esperara por la variable x cuando tiene a y cerrada.

El compilador podria detectar estos errores y proporcionar mensajes de error o bien, podria asumir una ordenacion parcial para en caso de deadlock, saber que variable habria de liberarase.

El compilador podria implementar el codigo de la seccion critica de la forma:

```
P(v-mutex);
S;
V(v-mutex);
```

Cada variable definida en la declaracion como "shared", tendra asociada un semaforo que se "cerrara" o liberara su acceso segun entre en una region critica o salga de ella. Este semaforo es transparente al usuario.

La estructura de la region critica se puede usar para resolver de forma efectiva los problemas previamente mencionados, pero no se puede usar para resolver algunos problemas de sincronizacion en general. Esto introdujo un nuevo concepto: La region critica condicional. La mayor diferencia es que ahora se evalua una expresion booleana B, tal que cuando la expresion es cierta se ejecuta con acceso exclusivo sobre la variable v, la sentencia S. Un ejemplo de su formato seria:

```
region v when B do S;
```

Cuando la expresion B es falsa, el proceso no cumple el acceso exclusivo y se para hasta que la expresion B sea cierta.

El compilador podria implementar la region critica con dos semaforos y dos variables enteras asociada a cada variable con acceso exclusivo condicional. Asi a la variable x, se implementarian de forma transparente al usuario, las siguientes:

```
var x-mutex,x-wait: semaphore;
    x-count,x-temp: integer;
```

La x-mutex proporciona el acceso mutuamente excluyente a la seccion critica. Si un proceso no puede entrar en la seccio critica porque la condicion booleana B es falsa, se esperara entonces en el semaforo x-wait, habiendo liberado previamente con un V(x-mutex) a x, para que se pueda seguir con el computo.

Se controlan el numero de procesos que entran en x-wait

mediante x-count. Cuando un ceso sale de la seccion critica, pdoria haber alterado el valor de alguna condicion booleana B que impedia la entrada de otro proceso en la seccion critica. Por eso, se debe permitir que cada proceso que espera en x-wait, compruebe su condicion booleana B. Se usaria x-temp para determinar cuando hemos permitido a cada proceso comprobar su condicion. Inicialmente x-mutex esta a 1, mientras que x-wait, x-count y x-temp se inicializan a 0. Un ejemplo de implementacion esta en <1>.

Se puede plantear el problema que la evaluacion de todas la expresiones booleanas B puede originar un codigo ineficiente. Hay varios metodos para reducir esta redundancia. Ejemplos se pueden encontrar en (*).

Hay tambien circunstancias en que la sincronizacion debe se debe proceder dentre de cualquier sitio de la region critica y no necesariamente al principio. Esto se realiza con la construccion "await":

5.5. COMUNICACION DE PROCESOS.

Existen dos esquemas fundamentales de comunicacion entre procesos; detallando:

1) memoria compartida (variables compartidas).
2) por mensajes.

La comunicacion por memoria compartida implica que la responsabilidad de establecer la comunicacion entre los procesos recaee en los programadores, sea esta una labor transparente (como es el caso del uso de monitores en el ADA)

o bien especificando las regiones criticas directamente. El sistema operativo solo necesita proporcionar la memoria compartida.

La comunicacion por mensaje implica que la responsabilidad de la comunicacion recae en el sistema operativo. Aun cuando la tarea del paso del mensaje no recaee en el programador sino en el sistema, el programador debera especificar todas el destino de cada mensaje (o sea hacia que tarea se manda el mensaje). Esto implica ciertas restricciones. El programa tiene que tener un esquema general de la comunicacion entre las diversas tareas, y la incorporacion de nuevas tareas implica la modificacion en la estructura de los envios y las recepciones de los mensajes. En la comunicacion por memoria compartida, si la comunicacion se efectua mediante acceso exclusivo, los programas puede que no tengan que ser modificados al agregar otra tarea que hace uso de las variables compartidas. En todo caso la decision pertenece al programador sobre cuando debe usar una tecnica u otra.

Nos centraremos ahora sobre la comunicacion por mensaje.Este esquema es apropiado en sistemas con espacios logicos de direccion disjuntos.

5.5.COMUNICACION POR MENSAJES.

Si dos procesos P y Q desean comunicarse, deben enviarse y recibir mutuamente mensajes. Estas dos operaciones basicas, enviar (send) y recibir (receive) son las que forman la base de un sistema comunicandose por mensajes. Surgen ciertos

esquemas distintos en la implementacion logica de la comunicacion como por ejemplo:

-Comunicacion directa o indirecta.
-Comunicacion a un proceso o a un mailbox (buzon).
-Comunicacion simetrica o asimetrica.
-Buffering automatico o explicito.
-Envio por copia o referencia.
-Mensaje de tamano fijo o variable

Pasamos a comentar, brevemente, estos detalles.

En la comunicacion directa, el proceso debe explicitamente especificar el destinatario o el remitente de la comunicacion. Asi, las primitivas send y receive se definirian como:

```
send (P,mensaje);
receive (Q,mensaje);
```

En el primer caso, se envia un mensaje al proceso P.En el segundo, se recibe un mensaje del proceso Q.

En la comunicacion indirecta, los mensajes son enviados hacia y recibidos desde "buzones" (mailboxes). Un buzon puede verse como un sitio donde pueden dejarse o recogerse mensajes. Un proceso puede comunicarse con otros, mediante un numero de buzones distintos. Asi, las primitivas send y receive se definirian como:

```
send (A,mensaje);
receive (A,mensaje);
```

Pueden existir ciertos problemas en la identificacion del remitente de un determinado mensaje con este metodo, ya que varios procesos pueden compartir un buzon. Hay varias soluciones:

-Permitir que un buzon solo pueda ser asociado como maximo a dos procesos.

-Permitir que un proceso ejecute como maximo una operacion receive cada vez.

-Que el sistema pueda identificar el destinatario al que envia el remitente.

Un buzon puede ser de un proceso o del sistema. Si pertenece a un proceso, entonces se puede distinguir entre el "propietario" y el "usuario" del buzon. En el primer caso, el proceso solo puede enviar mensajes. En el segundo, solo puede recibir. Si cada buzon solo tiene un propietario, no habra confusion en quien recibe un mensaje enviado al buzon. Cuando el "propietario" finaliza, el buzon desaparece, y todas las peticiones posteriores deben notificarse como error.

Un buzon que sea propiedad del sistema operativo, es independiente y no esta asociado a ningun proceso en particular. El sistema operativo nos permitiria:

-Crear un buzon nuevo.
-Enviar/recibir mensajes a traves del buzon.
-Eliminar un buzon.

Los procesos que pueden enviar mensajes son aquellos que hallan creado el buzon, o que otro proceso le asigne esa propiedad. En caso de que un buzon ya no sea accesible a ningun proceso debera ser eliminado para no ocupar mas memoria. Esto requeriria una "garbaje collection" (recoleccion de basura).

Al establecer una comunicacion, ya sea por buzones u otras herramientas, debe pensarse en la cantidad de mensajes que temporalmente pueden residir. Hay tres formas de implementar la cola de mensajes:

-Capacidad cero: no puede tenerse ningun mensjae esperando en el mismo. Ambos procesos deben estar sincronizados para que tenga lugar una transferencia. Esto se llama "rendezvous" (cita).
-Capacidad limitada: la cola tiene una longitud finita n. Si la cola no esta se puede seguir con la ejecucion, y el mensaje se almacenara en la cola. Si la cola esta llena, el proceso tendra que esperar a que halla espacio.
-Capacidad "ilimitada": el proceso nunca tendra que esperar al enviar un mensaje.

Si la capacidad es distinta de cero, un proceso no sabra si el mensaje a llegado a su destino despues de que se halla hecho el send. Si esta informacion es crucial , el proceso que envia , deve comunicarse explicitamente con el receptor para averiguar si ha recibido el mensaje.

Los mensajes pueden ser de tres tipos:

-Tamaño fijo.
-Tamaño variable.
-Mensajes de un tipo determinado.

En el primer caso, la implementacion en el sistema operativo sera sencilla pero la tarea del programador sera mas complicada. En el segundo caso, ocurre justamente al reves. El ultimo caso, asocia un tipo con cada buzon, se aplica unicamente en casos de comunicacion indirecta.

Un mensaje puede ser enviado por referencia, en cuyo caso lo que se envia es un puntero de donde reside el verdadero mensaje, y por copia, con lo que se envia una copia del mensaje. Cada sistema tiene su ventajas e inconvenientes como se vera mas adelante.

El uso de estas tecnicas puede implicar ciertos errores:

-Un proceso puede haber acabado antes de que halla procesado un mensaje.

-Un mensaje puede haberse "perdido".

-Un mensaje puede llega a destino equivocado.

Cada uno de estos problemas, tiene una solucion especifica. La idea aqui es solo enumerarlos.

5.6. DEADLOCK.

En un sistema multiproceso, varios procesos pueden competir por un numero finito de recursos. Si un recurso no esta disponible en un determinado momento, aquel proceso que lo haya requerido entrara en espera. Puede suceder que este recurso este retenido porque otro proceso este tambien en estado de espera. Asi, ambos procesos estaran siempre esperando por la liberacion de un recurso, cosa que nunca ocurrira. Esto se conoce como deadlock y es un problema que en ciertos casos implica soluciones drasticas. Fue reconocido y analizado por primera vez por Dijkstra. Este problema se encuentra en la comparticion dinamica de recursos, programacion paralela, sincronizacion de procesos, comunicacion de procesos etc.

Un ejemplo grafico del deadlock puede verse en la figura 5.1.

DEFINICION DE DEADLOCK.

Un conjunto de procesos se dice que esta en punto muerto

(deadlock) cuando
cada proceso perteneciente a este conjunto esta esperando por un suceso que solo puede ser causado por otro proceso del conjunto. Los sucesos a referidos son la liberacion y peticion de recursos.

Un proceso, cuando usa un recurso, solo puede seguir esta secuencia:

-Peticion:Si el recurso esta ocupado entonces el proceso quedara parado.

-Uso:El proceso hace uso del recurso.

-Liberacion:El recurso queda liberado. Otro proceso puede entonces ocupar el recurso.

Principalmente, existen dos metodos para tratar el deadlock:

-Se puede usar un protocolo de accesos a recursos que garantize que el sistema no caiga en deadlock.

-Se puede dejar que el sistema tenga un deadlock y luego tratarlo de arreglar.

El primer sistema implica un estudio en "estatica" de los procesos que vayan a competir por los recursos e imponer unas condiciones de acceso a los recursos para el peor caso. Esto hace que se pueda llegar a una baja eficiencia del codigo teniendo en cuenta que el codigo debe prevenir todas los posibles problemas, muchos de los cuales no se daran.

El segundo sistema puede llegar a aprovechar de mejor manera los recursos pero con el problema de que el

tratamiento de un sistema en deadlock es bastante dificil. Veamos un resumen de las tecnicas que se emplean en cada caso.

Para que un sistema no llegue a entrar en deadlock existen tres metodos basicos:

-Retencion y espera: Antes de empezar el proceso se deben haber adquirido todos los recursos que se necesitaban.

-No deasignacion: Si un proceso esta reteniendo algun recurso y pide otro, el recurso retenido debe ser liberado antes que se pueda coger el otro.

-Espera circular: Se impone una ordenacion a todos los tipos de recursos. Cada proceso puede pedir recursos solo en orden creciente.

Para evitar deadlock es necesario saber inicialmente cierta informacion sobre los recursos como por ejemplo, el numero de ellos.

En un sistema que no usa ningun protocolo para asegurar que no ocurrira un deadlock, se debe entonces de emplear un esquema de deteccion y recuperacion. Una vez detectado el deadlock, el sistema se debe recuperar deasignando recursos de alguno de los procesos en deadlock. En un sistema que elige "victimas" para deasignarle los recursos. Esto puede producir "starvation", un sindrome que se caracteriza porque un proceso nunca termina de ejecutarse porque siempre se le

deasignan los recursos.(Tambien se emplea esta palabra en el arbitraje en sistemas multiprocesadores cuando un procesador nunca puede adquirir el bus del sistema).

Se ha argumentado que ninguno de estos metodos solos, es apropiado para resolver el amplio espectro de problemas de asignacion de recursos. Los metodos pueden combinarse para seleccionar el metodo optimo.

BIBLIOGRAFIA CAPITULO 5

<1>: E. Rubio y otros.

Apuntes de sistemas operativos.

E. U. Informatica de Las Palmas. U. P. L. P.

<2>: B. Hansen.

Operative sistems principles.

Prentice-Hall.

<3>: ver referencia <1> capitulo 1.

<4>: ver referencia <2> capitulo 1.

<5>: ver referencia <4> capitulo 1.

<6>: Intel corp.

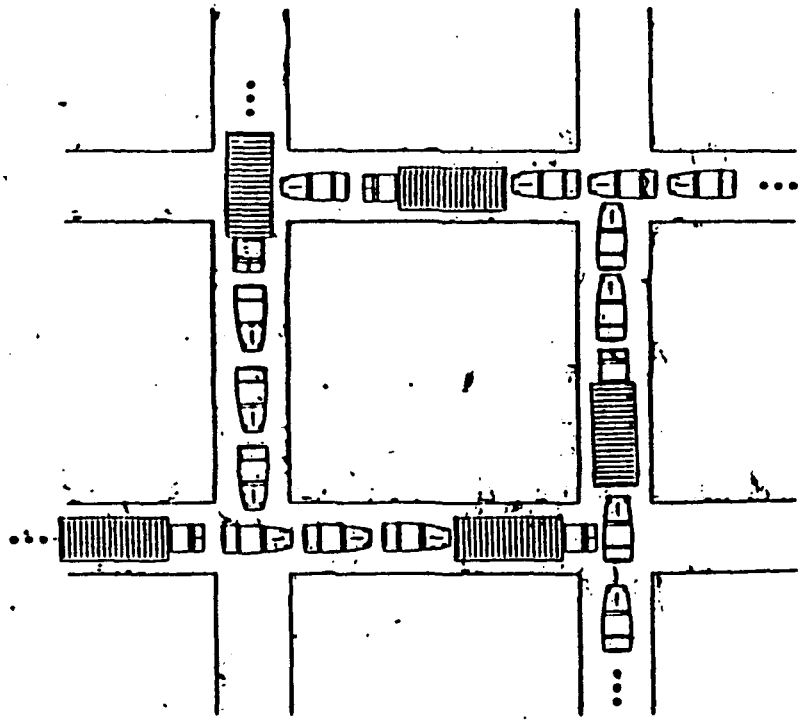
OEM systems handbook 1983

Capitulo 5.

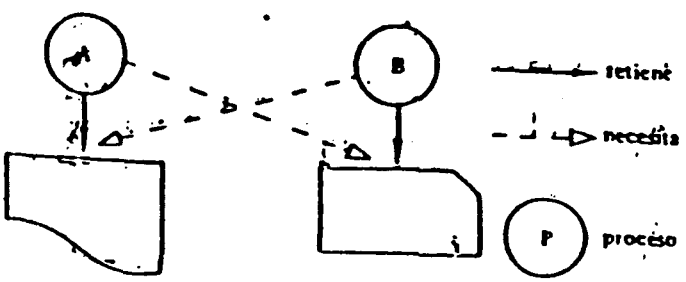
<7>: Intel corp.

iAPX86/88, 186/188 User's manual. Programmer's reference.

Capitulo 8. 80130 OSP.



Traffic deadlock



DEADLOCK ⇒ ESPERA CIRCULAR.

Fig. 5.1

SEGUNDA PARTE

CAPITULO 6

CAPITULO 6. ARQUITECTURA MULTIBUS

6.1. INTRODUCCION

En este capitulo se describe detalladamente el MULTIBUS-I, bus en el cual se engancha el prototipo desarrollado ya que es el que posee el sistema de desarrollo empleado como host, el MDS-221. Para una introduccion mas elemental, remitirse al capitulo 3.

6.2. DESCRIPCION DEL BUS DEL SISTEMA MULTIBUS

Las lineas del MULTIBUS, pueden ser agrupadas en las siguientes categorias: 20 lineas de direcciones, 16 lineas bidireccionales de datos, 8 lineas de interrupcion multinivel, y otras lineas para control de bus, temporizacion y suministro de potencia. Las lineas de direcciones y datos estan conectadas por drivers triestado, mientras las lineas de interrupcion y algunas otras de control son open-collector.

Los modulos que usan el interface multibus, tienen una relacion master-slave. Un modulo master puede acceder y controlar el bus. Un modulo slave no puede controlar el bus, sino aceptar instrucciones que provengan de un master. La arquitectura del multibus, permite soportar dispositivos de 8 o 16 bits.

El arbitraje del bus esta referenciado a un clock (BCLK/), que establece una referencia de temporizacion para resolver la contencion del bus entre multiples peticiones de varios

masters. Esto permite que masters de distinta velocidad compitan por los recursos compartidos en el mismo bus. Las transferencias via bus, son actualente asincronas respecto a BCLK/ y asi la velocidad de transferencia es dependiente unicamente de los dispositivos transmisores y receptores. El diseno del bus permite masters lentos puedan acceder y gann el bus, pero no restringe la velocidad con que los masters rapidos puedan transferir datos por el mismo bus.

6.3. DESCRIPCION DE LAS LINEAS DEL MULTIBUS

La siguiente seccion describe las lineas incluidas en el conector P1 que contiene las lineas de direccionamiento, datos, control de bus e intercambio, interrupcion y suministro de potencia. La mayoria de las senales del bus son activas a nivel bajo.

La figura 4.1 contiene la asignacion de las siguientes senales:

Senal de inicializacion del bus:

INIT/:linea de inicializacion; resetea el sistema enteramente. Puede ser forzado por un master o por una fuente externa como un switch de un panel exterior.

Lineas de direccionamiento e inhibicion:

ADR0/-ADR13/:20 lineas de direccionamiento; usadas para transmitir la direccion de una posicion de memoria o un puerto de I/O que va a ser accedido. La definicion del multibus solo permite usar 12 bits para el direccionamiento de I/O.

BHEN/:byte high enable: es una linea de control de direccionamiento que es usada para especificar que sera trasferido un dato por el byte alto (DAT8/-DATF/), lo que significa que se van a transferir 16 bits.

INH1/:inhibir ram: previene que los dispositivos de memoria RAM, respondan a la direccion emplazada en el bus de direccionamiento del sistema. Permite asi que dispositivos ROM puedan estar emplazados en la misma zona de acceso RAM.

INH2/:inhibir rom: su funcion es analoga a inh1 pero en este caso previene que los dispositivos de memoria rom respondan al direccionamiento.

Lineas de datos:

DAT0/-DATF/: 16 lineas de datos bidireccionales: usadas para transmitir o recibir informacion hacia o desde una posicion de memoria o un puerto de I/O.

Lineas de resolucion de prioridad del bus:

BCLK/:bus constante: el flanco negativo de BCLK/ es empleado para sincronizar la circuiteria de resolucion de prioridad de acceso al bus. Esta linea es asincrona al clock de la CPU. BCLK/ puede ser ralentizada, parada o puesta en "single step" a efectos de depurado.

CCLK/:reloj constante: que provee a los diferentes dispositivos del bus con una senal de frecuencia constante para uso general.

BPRO/:bus priority out signal: usada en los esquemas de resolucion de prioridad serie (daisy chain). BPRO/ es pasada a la entrada BPRN/ del master que tiene la siguiente

prioridad mas baja. Esta senal esta sincronizada con BCLK/, y no es llevada en el backplane.

BPRN/:bus priority in signal; indica a un master en particular que ningun master de mas alta prioridad esta pidiendo el uso del bus del sistema. BPRN/ esta sincronizada con BCLK/ y no es llevada en el backplane.

BUSY/: bus ocupado; es una senal open collector que es forzada por el master que tiene el bus bajo control para indicar que este esta en uso, y prevenir que otros masters puedan ganar el control del bus. BUSY/ esta sincronizada con BCLK/. Esta es una senal muy importante. Si algun tipo de ruido se generara en esta linea, o un determinado master ignorara su estado, provocaria un colision con resultados irreparables a nivel software.

BREQ/: peticion de bus; es usada con una red de resolucion de prioridad paralela para indicar que un master en particular requiere el uso del bus para una o mas transferencias de datos. BREQ/ esta sincronizada con BCLK/, y no es llevada en el backplane.

CBRQ/: peticion de bus comun; es otra linea open collector que es forzada por todos los masters potenciales para informar al master actual que otro desea hacer uso del bus. Si CBRQ/ es alta, indica al master que puede retener el bus. Esto elimina overhead provocado por el arbitraje de acceso al bus del master que actualmente lo posea.

Lineas de protocolo para la transferencia de informacion.

Un master proporciona las lineas MRDC/, MWTC/, IORC/, IOWC/ para indicar una transferencia de lectura o escritura

hacia memoria o I/O. Cuando un comando de lectura escritura este activo, las senales de direccionamiento deben estar estabilizadas para todos los esclavos en el bus. El master tiene que proporcionar el direccionamiento y opcionalmente los datos al menos 50 nseg antes de efectuar el comando de lectura o escritura del bus, iniciando la transferencia. Se debe seguir con este direccionamiento hasta 50 nseg despues de la finalizacion del comando. Un esclavo debe proveer una senal de reconocimiento al master, en respuesta al comando de lectura o escritura. Todos los comandos son asincronos a BCLK/. Se establece una analogia con las senales tipicas de control de los microprocesadores.

MRDC/: comando de lectura de memoria.

MWTC/: comando de escritura en memoria.

IORC/: comando de lectura a I/O.

IOWC/: comando de escritura a I/O.

XACK/: reconocimiento de transferencia, indica que la operacion ha sido acabada.

AACK/: reconocimiento XACK/ avanzado; inidca que la transferencia puede ya abandonarse que el dispositivo esclavo la acabara por su cuenta. Se emplea en modulos con RAM dinamica. Realmente esta linea no esta en el multibus, pero viene implementada en el MDS-221.

Lineas de interrupcion asincrona.

INT0/-INT7/: 8 lineas de peticion de interrupcion paralelas multinivel; usadas por la circuiteria de resolucion de interrupcion paralela. INT0/ tiene la mayor prioridad. Las

lineas son open collector.

INTA/: reconocimiento de interrupcion; es una senal de reconocimiento, forzada por el master del bus, pidiendo informacion sobre la peticion de interrupcion por un dispositivo controlador. La informacion especifica depende en general, de la implementacion del esquema de interrupciones. El flanco positivo de INTA/ indica que el bus de direcciones esta activo, miestras que el flanco negativo indica que el bus datos esta activo.

6.4. CARACTERISTICAS DE OPERACION

Es importante examinar las caracteristicas operativas del bus.

Transferencias de datos.

Las transferencias de datos en el multibus, ocurren con un ancho de banda maximo de 5 Mhz para transferencias simples o multiples. Debido al arbitraje y al tiempo de acceso a memoria, la maxima velocidad de transferencia es a menudo del orden de los 2 Mhz.

Lecturas de datos.

La figura 6.2 presenta un diagrama tipico de lectura. La direccion debe ser estable por un minimo de 50 nseg antes del comando. Este tiempo es el tipico usado por el interface del bus para decodificar las direcciones y asi proveer los selects requeridos para cada dispositivo. El ancho minimo del pulso del comando debe ser de 100 nseg. Las direcciones deben permanecer estables al menos 50 nseg despues de terminar el

comando. Los datos no deben ser emplazados en el bus antes que el comando, y no deben ser eliminados hasta que el comando haya acabado. La senal XACK/ es reponsable de indicar el fin de la operacion, y se activara cuando exista un dato valido y mantenerse hasta que el comando haya acabado.

Escritura de datos.

La figura 6.3 presenta un diagrama para escritura. Durante una transferencia, los datos deben ser presentados simultaneamente con las direcciones. Asi, el tiempo de set-up de los datos debe ser igual al de las direcciones. Este requisito posibilita que el dispositivo a donde se dirige la escritura, pueda latchear los datos con el flanco negativo del comando.

Byte swapping en sistemas de 16 bits.

Un master de 16 bits, puede transferir datos por el multibus ya sea en 8 o en 16 bits dependiendo de como haya sido especificada la operacion. La forma con que se realiza este interfasado de sistemas de 16 bits con dispositivos de 8 bits es completamente analoga a la empleada en la familia iAPX-86.

6.5.OPERACIONES DE INHIBICION

Las operaciones de inhibicion de bus estan requeridas en ciertas configuraciones de bootstrapping y dispositivos de I/O mapeados en memoria. El proposito de la operacion de inhibicio, es permitir una combinacion de RAM, ROM o I/O

mapeada en memoria, ocupando el mismo espacio de direccionamiento. Esto por ejemplo puede ser deseable tras la inicializacion de un sistema en donde el booting puede estar emplazado en una zona que posteriormente puede ser empleada en RAM. Un ejemplo del uso de estas lineas puede observarse en el posicionamiento de la RAM y la ROM en el sistema de desarrollo MDS-221, donde 2 K de EPROM comparten el mapeado con la RAM durante el periodo de diatnostico posterior al encendido o reseteado del aparato.

Hay dos requisitos fundamentales para una operacion de inhibicion exitos. La primera es que la operacion de inhibicion debe ser forzada lo mas pronto posible dentro del maximo de 100 nseg despues de la estabilizacion de direcciones. El segundo requisito es que el reconocimiento debe ser retrasado para permitir asi al esclavo inhibido, terminar cualquier operacion irreversible iniciada por la deteccion de un comando valido antes de la inhibicion. Esta situacion puede surgir porque un comando puede ser emplazado dentro de los 50 nsegs despues de la estabilizacion de las direcciones y todavia la inhibicion no ha sido requerida hasta despues de los 100 nsegs. El retraso del reconocimiento es una funcion del ciclo de la memoria del dispositivo inhibido.

6.6. OPERACIONES DE INTERRUPCION

Las lineas INT0/-INT7/ son usadas por un master para recibir interrupciones desde esclavos, otros masters, o

logica externa. Un master tambien podria tene fuentes de interrupcion internas que no requeririan las lineas de interrupcion del bus. Hay dos esquemas de implementacion de interrupciones, interrupciones vectorizadas e interrupciones no vectorizadas. Una interrupcion no vectorizada no lleva la informacion del vector de interrrupcion sobre el bus. Las interrupciones vectorizadas si lo llevan.

Interrupciones no vectorizadas.

Son aquellas cuyo vector de direccion es generado por el master y no requiere las lineas de direcciones del multibus para la transferencia de la direccion del vector de interrupcion. El vector de interrupcion esta generado por el controlador de interrupciones sobrfe el master y transferido al procesador por un bus local. La fuente de la interrupcion puede ser otro master o otros modulos del bus, y en cada caso se usan las lineas INTØ/ y INT7/.

Interrupciones vectorizadas.

En estas se pide la direccion del vector de interrupcion a traves de las lineas del multibus desde el esclavo al master, usando la linea INTA/ para la sincronizacion. Cuando una interrupcion ocurre, la logica de control de interrupcion interrumpe su procesador. El procesador siendo master en el bus, genera la INTA/, que congela el estado de la logica de interrupcion del esclavo para resolver la prioridad. El master tambien cierra el bus garantizando asi ciclos sucesivos bajo su control a memoria. Despues del primer INTA/, la logica de control de interrupcion del master, pone

un codigo de interrupcion sobre las lineas de direcciones ADR8/-ADRA/. El codigo de interrupcion es la direccion de la linea de interrupcion activa de mas alta prioridad. En este momento, dos tipos de secuencias pueden ocurrir. La diferencia se debe a que la especificacion del multibus puede soportar masters que generen un solo INTA/ adicional como el 8086, o dos, como el 8085. Si el master genera uno solo, esto causa a la logica de control de interrupcion del esclavo, que transmita un vector de interrupcion de 8 bits a traves de las lineas de datos. Este puntero es usado por el master para determinar la direccion de memoria donde esta la subrutina de servicio de la interrupcion. Si el master genera dos INTA/, entonces el esclavo pondra una direccion sobre las lineas de datos del multibus. Esa direccion es empleada para saber asi donde dar servicio a la interrupcion. La especificacion del multibus restringe el uso de las interrupciones vectorizadas a un tipo para un sistema dado.

6.7.OPERACION MULTIMASTER

El multibus tiene dos esquemas de resolucion de prioridades pra el intercambio del bus, una tecnica serie (daisy-chain) y una tecnica paralela, ya comentadas en el capitulo 2. Las figuras 6.4 y 6.5 ilustran estas tecnicas.

Tecnica de resolucion de prioridad serie.

La resolucion de una prioridad serie es efectuada con una tecnica de daisy chain. La entrada de prioridad (BPRN/) del

mastr de mas alta prioridad es conectada entonces a la entrada (BPRN/) del siguiente master con prioridad mas baja, y asi sucesivamente. En la figura 6.6 se aprecia graficamente esta concepcion. Cualquier master generando una peticion de bus, pondra su senal BPRO/ a nivel alto al master de siguiente prioridad. Cualquier master que vea una senal de nivel alto en su BPRN/, pondra la linea BPRO/ a nivel alto, y asi se ira pasando hacia abajo la informacion de la prioridad a los masters de mas baja prioridad. En esta implementacion, la linea de peticion BREQ/ no se emplea. Un numero limitado de masters pueden ser acomodados debido a los retrasos de puerta a traves de la daisy chain. Usando los chips controladores actuales, desarrollados por Intel, hasta 3 ers podrian ser acomodados suponiendo que BCLK/ tiene un periodo de 100 nseg. Si se quieren mas masters, se podra bajar BCLK/ o emplear la siguiente tecnica.

Tecnica de prioridad en paralelo.

En la tecnica de prioridad en paralelo, la prioridad es resuelta en un circuito de resolucion de prioridad en el cual la entrada de mas alta prioridad BREQ/, esta codificada por un codificador de prioridad 74148. Este valor codificado, es entonces decodificado con un chip tal como el 74S138 para activar la linea BPRN/ adecuada. Las lineas BPRO/ no son usadas en el esquema de resolucion de prioridad paralela y su conexion debe ser cortada en el backplane. Un limite practico de hasta 16 masters puede ser acomodado usando esta tecnica

debido a las limitaciones de longitud del bus. En la figura 6.7 se muestra un esquema para resolucion de prioridad paralela.

6.8.OPERACION DE INTERCAMBIO DE BUS EN EL MULTIBUS

Un diagrama de temporizacion de la operacion de intercambio del bus es mostrado en la figura 6.8. Este ejemplo usa un esquema de resolucion de prioridad en paralelo. Este esquema es el que mas se usa y esta implemenado en el backplane multibus del MDS-221. En este ejemplo, el master A tiene una prioridad mas baja que el master B. El intercambio del bus ocurre porque el master B genera una peticion del bus mientras que el master A tiene el control del bus.

El proceso de intercambio comienza cuando el master B requiere el bus para acceder a algun recurso (como por ejemplo un modulo de momoria) mientras el modulo A controla el bus. Esta peticion interna es sincronizada con el flanco negativo de BCLK/ para generar BREQ/. La circuiteria de resolucion de prioridad cambia el BPRN/ del master A de activo a inactivo y el BPRN/ de B se activa. El master A debe entonces de acabar el comando que actualmente esta teniendo lugar, suponiendo que hubiera alguno en operacion. Despues que el master A haya acabado, desactiva la senal BUSY/ en el siguiente flanco negativo de BCLK/. Esto permite que un intercambio de bus tenga lugar, debido a que el master A ha terminado con el control del bus, y el master B lo tiene concedido con la BPRN/. Durante este tiempo, los drivers de A

son desabilitados. El master B debe tomar el control del bus en el siguiente flanco negativo de BCLK/ para completar el intercambio del bus. Toma entonces el control activando BUSY/ y habilitando sus drivers. Se hace evidente los desastrosos efectos que una linea BUSY/ ruidosa puede llegar a causar. Si el maste B ha pedido el bus y el master A no ha terminado el comando que actualmente esta en operacion, en caso de que se produzca un ruido, el master B entendera que el bus ha sido liberado, activando entonces sus drivers ocasionando asi una colision con los drivers del master A que todavia no han sido puestos en triestado.

El master A puede retener el control del bus y prevenir que el master B se haga cargo del bus activando su BUS OVERRIDE o BUS LOCK, que mantiene la senal BUSY/ activa, permitiendo asi que el control del bus sigua con el master A. Esto garantiza que los siguientes ciclos consecutivos seran de este master, siendo asi posible la implementacion de estructuras como regiones criticas o posibilitando transferencias DMA que no pueden ser interrumpidas.

La linea CBRQ/ puede ser usada por un master que tenta control actual sobre el bus para determianr si otro master requiere el bus. Si master que actualmente controla el bus, ve la linea CBRQ/ inactiva mantendra el control del bus entre ciclos adyacentes. Asi se elimina el overhead de readquirir el bus cuando se vuelva a usar un recurso. Si el master ve la linea CBRQ/ activa, entonces si le conviene, liberara el bus y volvera a contender por el bus con otros masters. Las

prioridades relativas de los masters determinaran que master recibira el bus. Notese que a no ser que este activado el BUS OVERRIDE, ningun master podra hacer uso exclusivo del bus. Esto es cierto debido a que es imposible para una CPU pedir constantes accesos a los recursos del sistema. Los otros masters podran siempre acceder al bus mientras el master de mas alta prioridad lo tenga desocupado.

6.9. CONSIDERACIONES DE FALLO DE SUMINISTRO ELECTRICO EN EL MULTIBUS

Como se ha descrito en el capitulo 3, el conector P2 queda reservado para las lineas del iLBX. Sin embargo, en otras circunstancias, se emplea para tener un soporte minimo a los problemas generados por los fallos de suministro. No se entra en detalles, que el lector interesado puede encontrar en la referencia. La asignacion de los pines del conector P2 para la prevencion del fallo de suministro se muestra en la figura 6.9.

BIBLIOGRAFIA CAPITULO 6

<1>: Intel Corp.

OEM systems handbook 1983.

Capitulo 3.

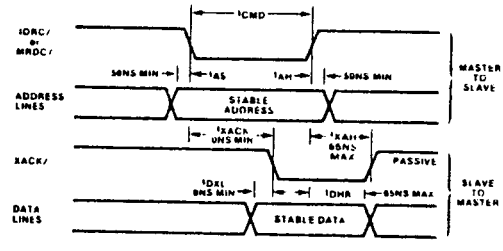
<2>: Intel Corp.

iAPX86/88 User's manual 1981.

	(COMPONENT SIDE)			(CIRCUIT SIDE)		
	PIN	MNEMONIC	DESCRIPTION	PIN	MNEMONIC	DESCRIPTION
POWER SUPPLIES	1	GND	Signal GND	2	GND	Signal GND
	3	+5V	+5Vdc	4	+5V	+5Vdc
	5	+5V	+5Vdc	6	+5V	+5Vdc
	7	+12V	+12Vdc	8	+12V	+12Vdc
	9	-5V	-5Vdc	10	-5V	-5Vdc
	11	GND	Signal GND	12	GND	Signal GND
BUS CONTROLS	13	BCLK/	Bus Clock	14	INIT/	Initialize
	15	BPRN/	Bus Pri. In	16	BPRO/	Bus Pri. Out
	17	BUSY/	Bus Busy	18	BREQ/	Bus Request
	19	MRDC/	Mem Read Cmd	20	MWTC/	Mem Write Cmd
	21	IORC/	I/O Read Cmd	22	IOWC/	I/O Write Cmd
	23	XACK/	XFER Acknowledge	24	INH1/	Inhibit 1 disable RAM
BUS CONTROLS AND ADDRESS	25		Reserved	26	INH2/	Inhibit 2 disable PROM or ROM
	27	BHEN/	Byte High Enable	28	AD10/	
	29	CBRQ/	Common Bus Request	30	AD11/	Address
	31	CCLK/	Constant Clk	32	AD12/	Bus
	33	INTA/	Intr Acknowledge	34	AD13/	
INTERRUPTS	35	INT6/	Parallel	36	INT7/	Parallel
	37	INT4/	Interrupt	38	INT5/	Interrupt
	39	INT2/	Requests	40	INT3/	Requests
	41	INT0/		42	INT1/	
ADDRESS	43	ADRE/		44	ADRF/	
	45	ADRC/		46	ADRD/	
	47	ADRA/	Address	48	ADRB/	Address
	49	ADR8/	Bus	50	ADR9/	Bus
	51	ADR6/		52	ADR7/	
	53	ADR4/		54	ADR5/	
	55	ADR2/		56	ADR3/	
	57	ADR0/		58	ADR1/	
DATA	59	DATE/		60	DATF/	
	61	DATC/		62	DATD/	
	63	DATA/	Data	64	DATB/	Data
	65	DAT8/	Bus	66	DAT9/	Bus
	67	DAT6/		68	DAT7/	
	69	DAT4/		70	DAT5/	
	71	DAT2/		72	DAT3/	
	73	DAT0/		74	DAT1/	
POWER SUPPLIES	75	GND	Signal GND	76	GND	Signal GND
	77		Reserved	78		Reserved
	79	-12V	-12Vdc	80	-12V	-12Vdc
	81	+5V	+5Vdc	82	+5V	+5Vdc
	83	+5V	+5Vdc	84	+5V	+5Vdc
	85	GND	Signal GND	86	GND	Signal GND
All Mnemonics © Intel Corporation 1978						

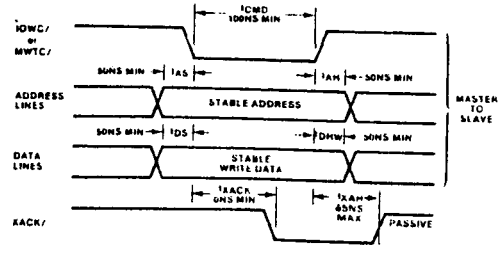
Fig. 6.1 Asignación de lineas en el Multibus-I.

Fig. 6.2



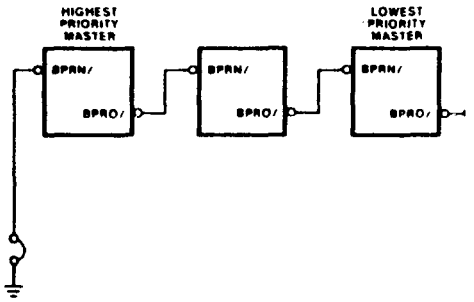
Read AC Timing

Fig. 6.3



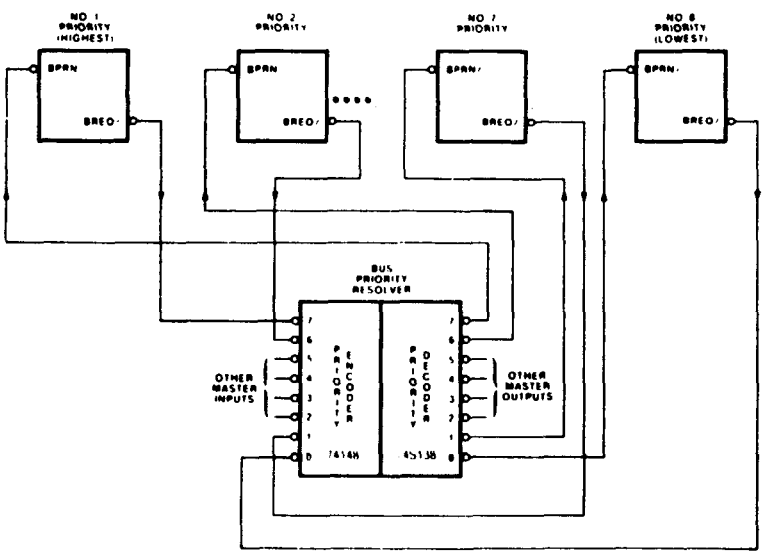
Write AC Timing

Fig. 6.2 Cronogramas de los ciclos de lectura y escritura en
6.3 el Multibus-I. Observar tiempo de establecimiento
(50nseg) antes y despues de efectuar comando.



Serial Priority Technique

FIG. 6.4



Parallel Priority Technique

FIG. 6.5.

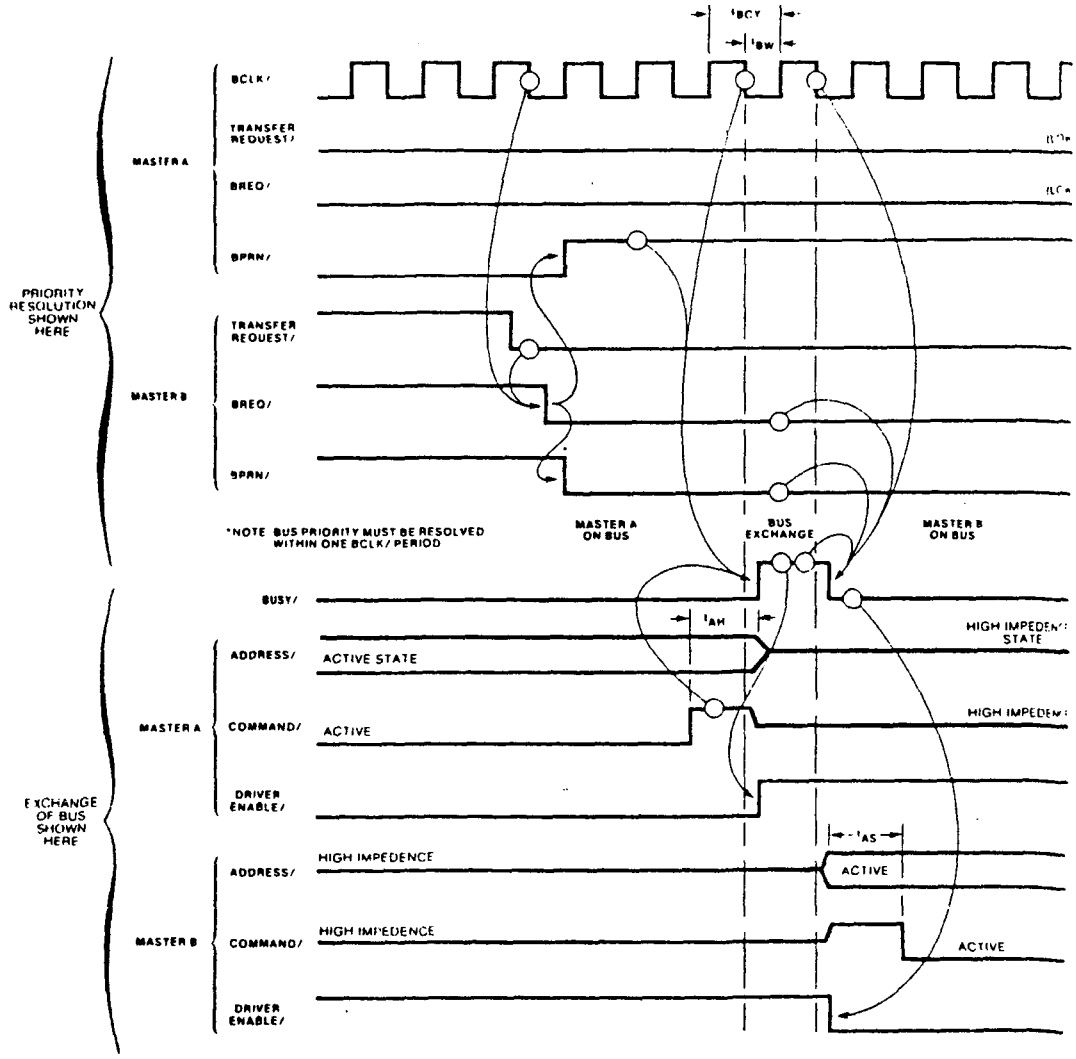


Fig. 6.8 Intercambio de bus. **Bus Control Exchange Operation**

PIN	(COMPONENT SIDE) MNEMONIC	DESCRIPTION	PIN	(CIRCUIT SIDE) MNEMONIC	DESCRIPTION
1	GND	Signal GND	2	GND	Signal GND
3	5 VB	+5V Battery	4	5 VB	+5V Battery
5		Reserved	6	VCCPP	+5V Pulsed Power
7	-5 VB	-5V Battery	8	-5 VB	-5V Battery
9		Reserved	10	Reserved	
11	12 VB	+12V Battery	12	12 VB	+12V Battery
13	PFSR/	Power Fail Sense Reset	14	Reserved	
15	-12 VB	-12V Battery	16	-12 VB	-12V Battery
17	PFSN/	Power Fail Sense	18	ACLO	AC Low
19	PFIN/	Power Fail Interrupt	20	MPRO/	Memory Protect
21	GND	Signal GND	22	GND	Signal GND
23	+15V	+15V	24	+15V	+15V
25	-15V	-15V	26	-15V	-15V
27	PAR1/	Parity 1	28	HALT/	Bus Master HALT
29	PAR2/	Parity 2	30	WAIT/	Bus Master WAIT STATE
31			32	ALE	Bus Master ALE
33			34	Reserved	
35			36	Reserved	
37			38	AUX RESET/	Reset switch
39			40		
40			42		
43	Reserved		44		
45			46		
47			48		
49			50	Reserved	
51			52		
53			54		
55			56		
57			58		
59			60		

Notes:
1. PFIN, on slave modules, if possible, should have the option of connecting to INT0/ on P1.
2. All undefined pins are reserved for future use.

Fig. 6.9 Asignación de lines para prevencion "power-fail".

CAPITULO 7

CAPITULO 7. FAMILIA iAPX Y SOPORTE A MULTIPROCESO

7.1. INTRODUCCION

En este capitulo se exponen las bases de la arqutectura d la familia iAPX empleada en la implementacion del proyecto. No se entra en detalles puesto que han sido ya descritos en numerosa documentacion. Mas bien, se tratara de exponer la filosofia del diseno. El 8089 es tratado con mas profundidad ya que, siendo inicialemente el objetivo del proyecto, se trabajo (teoricamente) mucho con el. Ahora, muy poco del trabajo global es descrito aqui, debido fundamentalmente a la no implementacion de estos. Es importante el arranque para analizar el depurador 89 que se realizao. Lo mas importante, como se ha dicho, es el mostrar la filosofia de compatibilidad HARDWARE que se complementa con la SOFTWARE, que fue empleada en el desarrollo de un sistema facilmente reconfigurable tal como se explicara en el capitulo 8.

7.2. FAMILIA iAPX86, iAPX88, Y SU RELACION CON iAPX186, iAPX188.

Los procesadores 8086 y 8088 desarrollados por Intel, disponen de una amplia documentacion, y ya en anteriores proyectos se vio con detalle los conceptos fundamentales de su estructura y su forma de interfase. No es mi proposito el repetir esto. Unicamente se describira someramente sus prestaciones y su interfase con otros coprocesadores. No se describira su ciclo de bus que se encontrara perfectamente definido en la literatura, sino que se establecera una

analogia con los 80186 y 80188, versiones mejoradas de los ya mencionados. Para mas detalles, se puede consultar <1>,<2>,<3>,<4>.

El 8086 dispone de un bus de 16 bits que se adapta a su arquitectura interna tambien de 16 bits. Tiene una capacidad de direccionamiento de 1Mbyte y su arquitectura fue diseñada para dar un adecuado soporte para lenguajes de alto nivel y ensamblador. Dispone de hasta 24 modos de direccionamiento de operandos y operaciones tales como multiplicar y dividar sobre numero de 8-16 bits con o sin signo. Su estructura de interrupciones esta mas evolucionada que el predecesor 8085 e incluye soporte tanto hardware como software para el uso de coprocesadores con la asignacion del micro en modo maximo. El acceso al bus esta mediatizado por una cola que actua optimizando el comportamiento del microprocesador ya que el acceso a una instruccion y su ejecucion puede ser efectuada en "pipeline". El 8088 tiene como unica diferencia importante, su bus de 8 bits lo cual baja el rendimiento pues se necesitan mas accesos a memoria. Asi mismo, el 8088, que tiene el mismo software, posee un algoritmo de gestion de cola diferente al 8086. Las diferencias se detallaran en un apartado expreso mas tarde.

El 80186 es fundamentalmente un 8086 con ciertas mejoras internas que le permiten un aumento de hasta dos veces las prestaciones del 8086, y que incluye en el chip ciertos dispositivos de uso muy comun en el desarrollo de sistemas y que actualmente estan como soporte externo en el 8086. Asi

mismo, aunque el codigo es completamente compatible "hacia arriba" a nivel de codigo objeto, se disponen de diez nuevas instrucciones. Los nuevos dispositivos incorporados en el 80186 son:

-generador de clock integrado.

-dos canales independientes de DMA de alta velocidad.

-controlador de interrupciones programable.

-logica programable para la seleccion de chip en memoria o perifericos.

-generador de estados de wait programable.

-controlador de bus local.

-tres contadores programables de 16 bits.

Estas mejoras han supuesto un aumento del patillaje del integrado que se pone ahora en 69 patillas en JEDEC tipo A. La asignacion de las patillas, asi como una descripcion interna en bloques se puede observar en las figuras 7.1 y 7.2. El 80188 tiene un bus de 8 bits con el algoritmo de cola cambiado que le proporciona un ancho de banda de 2 Mb/seg. En el caso del 80186, el ancho de banda del bus alcanza los 4 Mb/seg.

El 80186 ya no posee modo maximo. Existen ciertas diferencias con respecto al arbitraje del bus local con respecto al 8086. Estas diferencias se concretan en el siguiente punto.

-dado que el 80186 tiene reloj interno con 50% de cilo de trabajo tenemos que la fase baja del 80186 es mas estrecha y la fase alta mas ancha que el 8086 a misma velocidad

-no existe salida de oscilador disponible en el 80186 como la

que hay en el 8284a.

-no es disponible el PCLK como en el 8284A.

-el 80186 no cualifica internamente AEN con el RDY. Si las entradas ARDY y SRDY son usadas, se debe incluir logica externa paa prevenir dos readys.

-el 80186 solo provee una entrada de ready asincrona y otra sincrona.

-el drive de CLOCKOUT es menor que la salida de CLK del 828aA.

-el cristal empleado es del doble de la frecuencia de cpu, no del triple como en el caso del 8086.

Asi mismo, hay ciertas caracteristicas en el uso del bus. Estas se pueden resumir en:

-ya que el 80186 puede soportar control local y salida de estado simultaneamente, se pdorian realizar sistemas con un bus global que no requirieran dos controladores externos separados.

-la linea de ALE del 80186 se vuelve activa un ciclo antes que la generada por el 8288 (controlador de bus). Esto mejora la propagacion de las direcciones a traves de los latches, pero en el caso de uso con el multibus, se debe emplear el ALE generado por el 8288 para cumplir los requisitos de temporizacion.

-la linea RD/ del 80186 debe ser puesta a tierra para conseguir el modo de estado de cola, indispensable para la sincronizacion con los coprocesadores. El estado de la cola es disponible una fase antes que en el 8086, por lo que es

necesario un cierto retraso.

-el 80186 emplea un protocolo de intercambio de bus local del tipo HOLD/HLDA que permite compatibilidad con los nuevos dispositivos perifericos, en comparacion con el protocolo RQ/-GT/ empleado en el 8086 en modo maximo. Se dispone de un integrado 82188 que efectua el interfase de estos diferentes protocolos, genera los comandos y modifica el compartamiento de la cola para que dispositivos tales como el 8087, puedan ser empleados con el 80186.

Mas informacion sobre las caracteristicas hardware y software de estos dispositivos puede ser encontrada en <4> y <5>.

El 8088 tambien posee ciertas diferencias en su comportamiento con el 8086. Estas diferencias, que resulta importante remarcar debido a que se trata de usar la estructura comun a estos sistemas, son bastantes importantes pues su omision puede provocar la destruccion del componente.

-los pines A8-A15 son unicamente salida en el 8088. Estas lineas de datos estan enclavadas (latcheadas) y se mantienen valida durante un ciclo de bus en una forma similar al 8085.

-BHE/ no tiene significado en el 8088 y ha sido eliminada. La patilla esta a nivel alto en modo maximo.

-SS0/ proporciona la informacion de estado S0/ en modo minimo. Esta salida conjuntamente con DT-R/, e IO-M/ proporcionan informacion completa del ciclo de bus.

-IO-M/ ha sido invertida para mantener la compatibilidad con la MCS-85.

-ALE esta retrasada un ciclo de reloj en modo minimo cuando

se entra en HALT para permitir el enclavamiento del estado.

La mayoria de las diferencias se notan unicamente en modo minimo, pero la diferencia del direccionamiento puede ocasionar ciertos problemas.

7.3.EL CONTROLADOR DE BUS 8288.

Este integrado tiene como funcion el decodificar las lineas de estado de la CPU y generar los comandos que controlaran a la memoria o la I/O. El controlador tiene dos formas fundamentales de trabajo; el modo I/O bus y el modo de sistema. En el primer caso, usado con el coprocesador 8089, el controlador no necesita la cualificacion de las direcciones otorgada por la semal AEN/, ya que considera que el bus de I/O residente que forma parte de la arquitectura del 8089, no posee arbitraje y que esta actualmente disponible. Asi, genera inmediatamente el comando, y activa inmediatamente los drivers, usando para ello las lineas PDEN/ y DT-R/. Las lineas de comandos de I/O no deben ser entonces usadas en el bus del sistema ya que no tienen arbitraje, y por tanto daria lugar a una colision con el acceso de otro master. Al no haber arbitraje, tampoco se imponen estados de espera extra.

En el modo de sistema, la patilla IOB esta a nivel bajo, y entonces ningun comando es generado hasta 115 nseg despues de que la linea AEN/ este activada. Este modo supone que existe un arbitro de bus (8288), que activa la semal AEN/ cuando el bus del sistema esta liberado. Aqui se pueden mezclar los

comandos de entrada y salida, y de acceso a memoria por el mismo bus.

Las salidas de los comandos son las siguientes:

MRDC/:Memory read command,
MWTC/:Memory write command,
IORC/:I/O read command,
IOWC/:I/O write command,
AMWC/:Advanced memory write command (se emplea en RAM dinamicas para no generar estados de espera).
AIOWC/:Advanced I/O write command (idem pero en I/O).
INTA/:Interrupt acknowledge.

La senal INTA/ actua como lectura de codigo en I/O cuando trabaja en modo IOB. Por ello, hace falta la inclusion de una puerta NOR conjuntamente a IORC/ para generar el comando adecuado.

Estas salidas antes mencionadas, estan controladas por la senal CEN/ que actua como cualificador de los comandos. Esto permite implementar particiones de memorias sin problemas de conflictos entre los dispositivos implementados en bus multimaster y un bus residente.

Para controlar los latches y drivers, se emplean las lineas DEN/ y DT-R/ para habilitar y determinar la direccion de los drivers respectivamente. Para el caso del bus de I/O, se emplea la linea PDEN/ para habilitar los drivers del bus residente. El strobe de los latches es generado por ALE.

Existe una linea MCE que habilita un latch en caso de que existan controladores de interrupcion en cascada, para introducir el vector de interrupcion en el bus de datos del

sistema en donde es leido por el procesador.

Un esquema del patillaje se encuentra en el apendice 7.3. Su forma de conexion para el desarrollo de sistemas multiprocesadores se vera en el apartado que trata del arbitro de sistemas multimaster (8289).

7.4. CONTROLADOR DE MULTIBUS 8289.

El arbitro de bus 8289 es un dispositivo que conjuntamente con el controlador de bus 8288, efectua la interface entre un sistema iAPX-8X,iAPX-18X configurado en modo maximo, y un bus multimaster tal como el MULTIBUS-I. El arbitro resulta transparente al procesador y suministra los comandos como si tuviera uso exclusivo del bus. Si el bus multimaster no esta libre, el 8289 desabilita el controlador de bus 8288, y retiene al procesador haciendole ver como si la transferencia a memoria no estuviera lista. El procesador permanecera en estado de wait hasta que el arbitro haya conseguido el uso del bus, en cuyo caso, permitira al 8288 hacer uso del bus, activando sus latchs de direcciones y los drivers del bus de datos. Una vez que el dispositivo esclavo al que se esta accediendo por el multibus, ha terminado su ciclo de lectura escritura, se producira un reconocimiento (XACK/) que correspondera a un READY con respecto al micro y el procesador acabara su ciclo de transferencia. Asi, el arbitro permitira que los distintos procesadores entren sin colision y hagan uso de los recursos del sistema.

Los modos de resolucion de prioridad en el multibus, han

sido ya previamente comentados por los que no volveremos a hacerlo aqui.

Los modos de operacion del 8289, se fundamenta en los dos tipos de procesadores que fundamentalmente tiene la familia iAPX-86. Existen CPUs como tales, y existe ademas, el coprocesador de entradas y salidas (8089), que determina un comportamiento relativamente distinto en el 8289 y el 8288. Este comportamiento "distinto" no es tal, sin embargo, pues puede emularse si se tienen en cuenta ciertas circunstancias que ya comentaremos, lo cual permite la construccion de sistemas muy homogeneos desde el punto de vista hardware.

Fundamentalmente la diferencia radica en que el 8089, puede tener un bus residente para el tratamiento de I/O, de tal manera que todos los dispositivos en ese bus son tratados y direccionados con comandos de I/O. Esta diferencia es minima a mi juicio, pues el 8086 puede emular perfectamente este comportamiento con solo monitorizar la linea S2/, que define, al igual que el 8089, su acceso a I/O. La opcion de strapping IOB/, configura al 8289 en este modo, lo que le dara a entender que existe un bus residente que sera accedido unicamente en situaciones de acceso a I/O. El 8288, bajo este modo tambien, generara las semales de acceso a I/O y accionara sus drivers. Este comando se emula, decodificando como previamente se ha dicho, la senal S2/ y habilitando con ella la linea SYS-RESB/, que determina el uso de un bus residente.

El 8289 tiene diversos modos de operacion:

1) Interface con un unico bus al multibus.

El modo mas simple se emplea en aquellos sistemas en que se pueda soportar el overhead debido a la no existencia de memoria local. Todo dependera del ancho de banda del bus.

2) Modo IOB.

Es ideal cuando el bus puede ser separado en un bus de I/O y otro de memoria de sistema. Este modo suele ser usado con el 8089 trabajando en modo remoto.

3) Modo residente.

Este es el modo de mayor flexibilidad y permite que el bus principal tenga menos trabajo. Se usa un mapeo de memoria y este modo necesita un segundo 8288.

En la figura 7.4 pueden verse los tres modos en un sistema. En la figura 7.5 puede verse un sistema con dos accesos a multibus.

Existen otros integrados de soporte tales como el 8128 y 8219 para las familias MCS-80 y MCS-85. En este caso difieren un tanto. Ejemplos en <6>. La estructura del 8289 es aplicada casi analogamente al 80286 a efectos macroscopicos como se observa en la ultima figura.

7.5. PROCESADOR DE ENTRADAS Y SALIDAS 8089

El 8089 es un procesador desarrollado por la Intel para proveer con un soporte hardware especifico, los problemas asociados con I/O. Sus caracteristicas y su conjunto de instrucciones estan optimizado pra un manejo eficiente,

flexible y a alta velocidad. Puede trabajar como coprocesador de un 8088 o 8086 (modo local), o bien como procesador autonomo (modo remoto). Puede interfasarse con buses de 8 o 16 bits de forma mixta. El 8089 se comporta como dos procesadores, cada uno con su juego de registros, que proporcionan dos canales de I/O independientes. Se puede conseguir una transferencia de 1.25Mbytes/seg con un reloj de 5Mhz. La comunicacion con el procesadore es a traves de memoria, lo cual aumenta la flexibilidad del sistema y ayuda a la construccion de software modular. Posee un espacio de direccionamiento de 1Mbyte y su DMA posee funciones inteligentes que le permiten traduccion, busqueda, o uso de buses mixtos. Se permiten trasferencias entre I/O-memoria, memoria-memoria, I/O-I/O. Es interfasable con el Multibus siguiendo el mismo concepto de arquitectura que las familias iAPX8X-iAPX18X. En figura 7.6 se puede observar la disposicion del patillaje. Vease que este es extremadamente parecido al del 8086, con la unica diferencia de las patillas que son particulares a este.

El 8089 puede ser aplicado en control de aplicaciones tales como floppy-disk, discos duros, CRT etc. Debido a estas peculiaridades, se considero su uso en el controlador de disco para el MDS. Las referencias en la bibliografia proporcionan una buena documentacion sobre su interfase con otros dispositivos.

7.6.MODOS DE COMUNICACION ENTRE CPU E IOP.

En la comunicacion entre la cpu y el iop, se distinguen

dos fases que pasamos a distinguir y definir:

1) Inicializacion: Comunmente despues del encendido (cold-start). La cpu prepara unos mensajes en memoria y con una senal indica al iop que puede proceder a su lectura. Se determina el modo de trabajo, el ancho de los buses y el acceso.

2) Comandos: Todas las comunicaciones entre cpu e iop pasan por el "channel control block" (CB). Su direccion se pasa en la inicializacion y reside en el espacio de memoria de la cpu. Tiene dos zonas para cada uno de los canales y su contenido es:

a)Flag de busy, operada por el canal para indicar a la cpu si esta trabajando o puede aceptar un nuevo comando. Esencialmente es un semaforo.

b)El channel command block (CCW), que es por donde la cpu escribe que tipo de operacion va a efectuar el iop (empezar y parar un determinado programa, interrupciones, etc...)

c)Un puntero que indica al canal la posicion de un bloque de parametros (PB) que a su vez senala al bloque de tarea (TB). En el PB estan los parametros del programa, mientras que el TB contiene el programa en si. En el PB se pasan todos los parametros de ida y vuelta de la cpu y el iop.

(Los punteros referidos previamente estan en formato estandar 8086, compuesto por un offset y un segmento).

El iop puede generar una interrupcion a la cpu para que esta le atienda, segun sean las lineas SINTR-1 o SINTR-2. Un esquema de la comunicacion podria verse en la figura 7.7.

Una vez que la lista linkada ha sido convenientemente compuesta en la memoria, la cpu activa un CA (channel attention) y un SEL (select), para indicar a un determinado canal que comienze una secuencia de inicializacion basada en los datos de la lista. La actuacion del iop es aproximadamente igual a una interrupcion pero no guarda ningun valor. Cuando se atienda el programa, entonces la flag busy se activara indicando que el proceso esta tomando lugar.

Se puede interpretar el bus del 8089 como formado por otros dos buses con funciones uno para el acceso a memoria y otro para la I/O. En modo remoto, (el que nos interesa), el bus de I/O esta fisicamente separado del sistema multibus. El acceso al sistema por parte del 8089 tiene bastante flexibilidad y asi pueden mezclarse diversas configuraciones de memoria local y publica con I/O. Sin embargo, debe tenerse muy en cuenta que el paso de parametros de comunicacion entre cpu e iop siempre se realiza por el bus mapeado en memoria.

En nuestro caso, se procederia a una emulacion de cpu. Los parametros estarian en memoria local, y la secuencia de inicializacion se generaria por logica hardware.

7.7.ESTRUCTURA DE LOS CANALES.

Cada uno de los canales contiene una seccion que se especializa en la sincronizacion tras las transferencias de DMA.

Su acceso al bus depende de la prioridad conferida a cada uno. En cualquier caso, el bus queda multiplexado en el

tiempo en cuanto al uso de un canal.

Estos canales contiene cada uno, un grupo de registros, inaccesible uno del otro. La mayoria juegan papeles importantes en el transcurso de una operacion de DMA. Estos registros son:

GA..Sirve como registro general o como registro base, y puede servir como puntero en una transferencia DMA.

GB..Igual que GA.

GC..Igual que GA. En una transferencia DMA apunta a la tabla de traduccion.

TP..Puntero de tarea, cargado inicialmente del bloque de parametros. Actua como contador de programa.

PP..Puntero del bloque de parametros que se carga antes de la ejecucion del programa. No puede ser modificado pero es util en el acceso a parametros.

IX..Registro Indice.

BC..Registro general. En transferencias DMA actua como contador.

MC..Registro general. En transferencias DMA actua como mascara y comparador.

CC..Control de canal. Controla la DMA y la prioridad de ejecucion por lo que no no debe usarse como registro general.

PSW.Informa a la cpu del estado del canal. El canal no tiene acceso a este.

Tag bits.Definen si el acceso es a I/O o a memoria. Todo depende de la inicializacion y de las operaciones que se efectuen con ese registro.

7.8.ESTRUCTURA HARDWARE 8089 E INICIALIZACION.

La arquitectura hardware resulta completamente compatible con la del 8086,8088 a excepcion de las lineas particulares que controlan la sincronizacion de la DMA y teniendo en cuenta que el bus puede tener un comportamiento mixto para permitir el uso de dispositivos de 8 y 16 bits. Se describira aqui unicamente el interfase hardware para la comunicacion con la cpu y los tipos y peculiaridades asociadas a la transferencia DMA. La configuracion de los controladores de bus y arbitro (8288 y 8289) para actuacion en modo bus I/O se ha explicado previamente mostrandose el bus I/O local.

La inicializacion del iop es labor del procesador host, ya sea un 8086, o un master en una multibus. La inicializacion del iop en si empieza con la activacion del RESET. Esta linea suele proceder del generador de clock 8284 y debe cumplir los requisitos impuestos tambien para el 8086. Despues del reset, el host avisa al iop para que comienze su secuencia de inicializacion activando la entrada del 8089 CA (channel attention). El 8089 no reconocera el pulso por lo menos un ciclo de reloj despues de que el reset quede inactivo. Coincidiendo con el flanco positivo del primer CA, el 8089 muestrea la entrada SEL (select) para determinar su estado master-slave para la circuiteria de request-grant. Si la entrada SEL esta a nivel bajo, el 8089 es asignado como master, y viceversa. Cuando el 8089 funciona como master, accede al bus directamente como master y sera el encargado de hacer el manejo de este. Esta es la configuracion que se

emplea en modo remoto, la que se emplearia para el caso del controlador de disco. En caso de que fuera designado como esclavo, tendria que hacer una peticion de bus por medio de RQ/-GT/ cada vez fuera a hacer una transaccion. La cpu se encargaria entonces de concederselo en el tiempo preciso.

Ademas de determinar el estado master-slave, el pulso de CA causa que el 8089 comienze una secuencia de inicializacion de una ROM interna. El 8089 debe tener el control del bus del sistema, no de I/O para realizar esta inicializacion. En la ejecucion de la secuencia de inicializacion, el iop lee varios bytes que indican la configuracion global del sistema. Especificamente, lee el byte SYSBUS emplazado en la 0FFFF6H, en donde se especifica el ancho del bus del sistema. Dependiendo del ancho especificado, se accede entonces al bloque de configuracion del sistema (SCB) contenido en las posiciones 0FFFF8H a 0FFFFBH. El puntero especificado en el SCB inica la posicion del comando de operacion de sistema (SOC). En este ultimo acceso, se conoce el ancho fisico del bus de I/O. Despues se accede lee el bloque de control de canal (CB). Para indicar al host que la secuencia de inicializacion ha sido completada, el 8089 pone a cero la flag de busy del primer canal. Despues que el iop haya sido inicializado, el (SCB) puede ser alterado para inicializar otro iop. Una vez inicializado, el CCB (channel control block) no puede ser alterado ya que su direccion esta internamente guardada y esta es automaticamente accedida en cada pulso de CA.

La activacion de las lineas de SEL y CA suele estar a cargo de un puerto de I/O.

Durante la operacion normal, el supervisor de I/O que corre en el host, recibiria la peticion de efectuar una operacion deteminada en uno de los canales del 8089. En respuesta a esta peticion, el programa supervisor normalmente efectuaria la siguiente secuencia de operaciones:

*Testear la disponibilidad del canal especificado examinando el flag de busy del bloque de control de canal. Es posible que varios procesadores accedan al canal, implementandose el acceso con una operacion de semaforo.

*Cargar los parametros variables requeridos para la operacion.

*Cargar el CCW (channel command word)

*Establecer los linkajes necesarios para escribir la direccion de comienzo del programa de cnal en los primeros de los cuatro bytes, y escribiendo esta en el bloque de parametros del bloque de control de canal.

*Mandar un CA (channel attention) al canal especificado.

Esto es igual para modo remoto o local.

Las transferencias de dma consisten en al menos dos ciclos de bus; uno de lectura del dato desde la fuente hacia el iop, y un bus ciclo de escritura desde el iop al destino. Notese que para todas las transferencias, los datos pasan a traves del iop para permitir las operaciones de comparacion-enmascaramiento y traduccion que opcionalmente pueden ser efectuadas durante la transferencia, asi como para adaptar los distintos anchos fisicos de bus que pudieran tener el bus

del sistema y el bus de I/O.

El iop realiza transferencias dma en uno de los siguientes tres modos: no-sincronizada, sincronizada en fuente, sincronizada en destino. El modo no-sincronizado es usado cuando los dispositivos fuente y destino no proporcional la senal de peticion de transferencia al IOP como de transferencias memoria-memoria. En los modos de trasferencia sincronizada en fuente o destino, el dispositivo inicia el ciclo de transferencia activando la linea DRQ1 o DRQ2 segun el canal a usar. La entrada DRQ es asincrona y suele estar originada por un controlador de I/O tal como un controlador de disco. El iop empezaraa a efectuar el ciclo de lectura en caso de sincronizacion en fuente o de escritura en caso de sincronizacion en destino, direccionando el dispositivo de I/O. La direccion debe ser decodificada por circuiteria externa de forma analoga a la del chip select, para generar el reconocimiento de DMA , DACK. El controlador normalmente retirara la peticion al recibo del DACK. Una transferencia DMA puede acabar cuando ocurra una de las siguientes condiciones que previamente estarian habilitadas por el software:

*transferencia de un unico ciclo.

*terminacion de la cuenta.

*condicion de mascara y comparacion.

*un hecho externo.

Las condiciones son especificadas en el registro de control del canal. Se puede especificar mas de una condicion.

El resultado seria un salto a una tabla de JMPs en funcion de la condicion que se ha cumplido. La terminacion por un hecho externo se produce por un nivel en las lineas EXT1 o EXT2 segun el canal implicado.

7.9.ARQUITECTURA ACTUAL DEL MDS.

El sistema de desarrollo que se empleo para el desarrollo de software y el soporte del sistema, fue un MDS 221 de Intel, con 64k RAM y 4k ROM. La arquitectura del sistema esta orientada hacia un interfase multibus alrededor del cual gira enteramente el sistema. Se pueden distinguir claramente 3 procesadores en el sistema, con funciones completamente distintas. Estos procesadores forman parte de los siguientes subsistemas:

IPB: Integrated processor board, donde reside el 8080A-2 responsable del proceso en si del sistema.

IOC: Input Output Controller, encargado del control de entradas y salidas, manejando la unidad de disco, el CRT y el teclado. El teclado usa un microcomputador para la decodificacion de este.

PIO: Parallel Input Output, encargado de las comunicaciones paralelas (impresora, puncher, programador de PROM).

La informacion de la IPB a la PIO o a la IOC, se pasa en forma de comandos de un byte que pueden tener asociados un byte mas de datos. El control de la comunicacion se efectua mediante unos flags del buffer DBB. Este byte es el que indica el estado de ejecucion de la PIO o el IOC, y su control es por software unicamente. La finalizacion de una

tarea por parte de la IOC o la PIO, se notifica con un bit en el DBB; no se efectua ninguna interrupcion. En el caso de teclado y las lineas de entrada y salida serie, necesitan un polling casi constante.

La organizacion del sistema en torno al multibus, le confiere una gran flexibilidad. Otros masters pueden hacerse cargo de los recursos del sistema. Estos recursos son: la RAM, el monitor en ROM, las funciones totales de la IOC, y lo mismo de la PIO.

El uso de estos recursos se debe efectuar por medio de los puertos de comunicacion ya previstos. Atencion que la IPB no permite el acceso a: los controladores del bus y del sistema, el puerto de control, los dos canales serie, los contadores programables (timers), y los controladores de interrupciones y el monitor de diagnostico.

El direccionamiento generado por el IPB trabaja sobre ADR0/-ADRF/. Las lineas ADR10/-ADR13/ se comprueban para ver si el direccionamiento corresponde a su segmento de memoria. La linea CBRQ/ no esta implementada. EL esquema de arbitraje es paralelo, siendo la IPB el master de mas baja prioridad. La IPB trabaja introduciendo un estado de espera en cada acceso a memoria. La IPB trata de mantener uso exclusivo del bus hasta que otro master se lo pida o ejecute un HALT. Se puede efectuar un LOCK al bus del sistema activando el override provisto y accesible por un puerto. Este override se activa en el arranque del sistema para que la IPB pueda

llevar a cabo tareas de diagnostico sin ningun interferencia de otro master. Una vez terminado el diagnostico, se desactiva el override y se entra propiamente dicho en el monitor. El mapeado de la ROM se comparte con el de la RAM por un procedimiento denominado SHADOWING, con auxilio de las lineas INH1/ e INH2/.

BIBLIOGRAFIA CAPITULO 7

<1>: Intel corp.

iAPX 86/88, 186/188 User's manual. Hardware reference.

<2>: Intel corp.

Microsystems components handbook. Volumen 1 y 2.

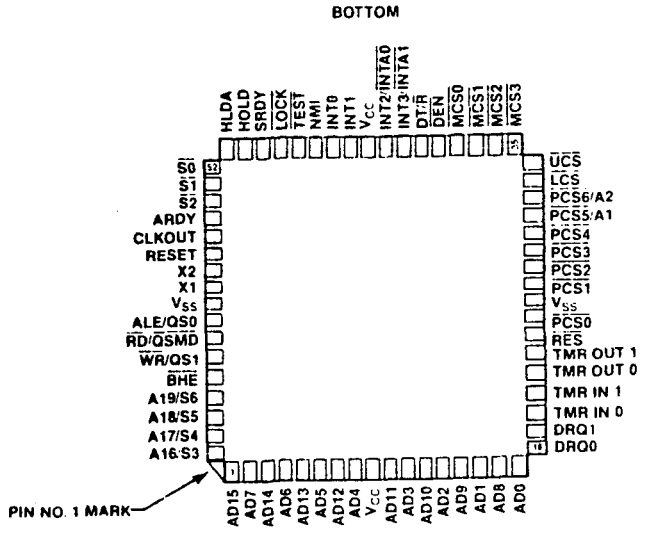


Fig. 7.1 Patillaje del 80186. Obsérvese que no existe la patilla de modo y que tampoco existen las líneas de petición y concesión de bus tal como se conciben en el 8086 en modo máximo.

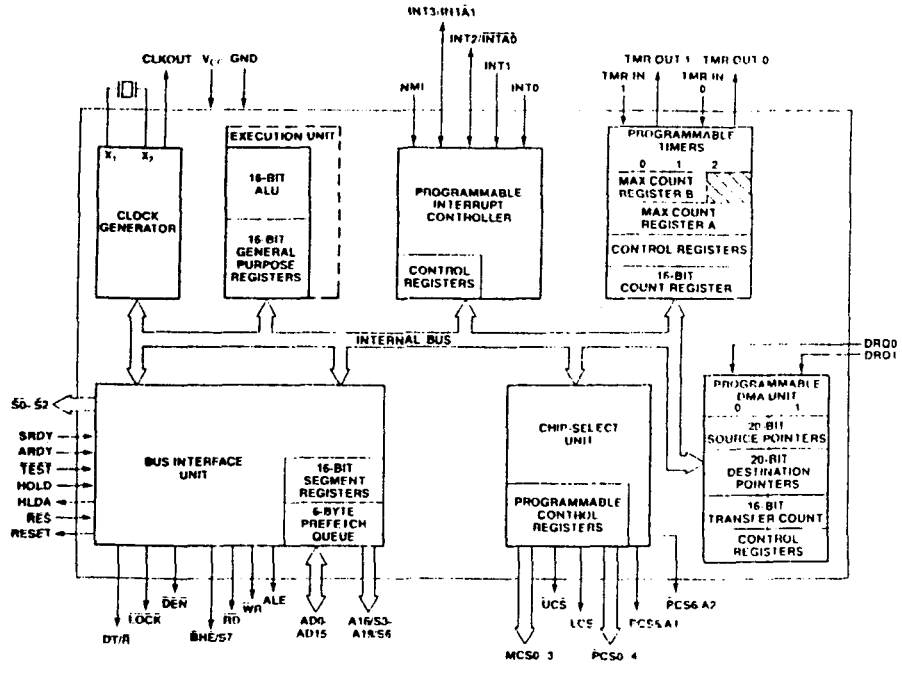
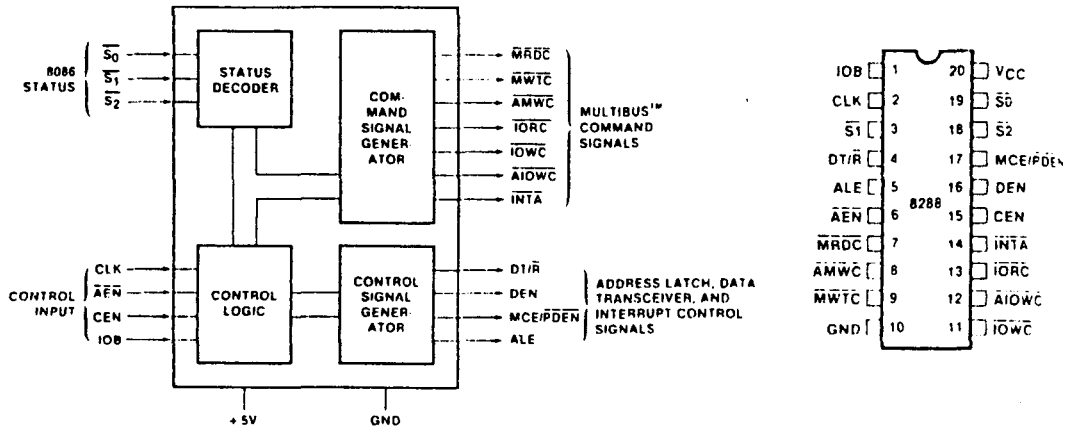


Fig. 7.2 Estructura de bloques del 80186.



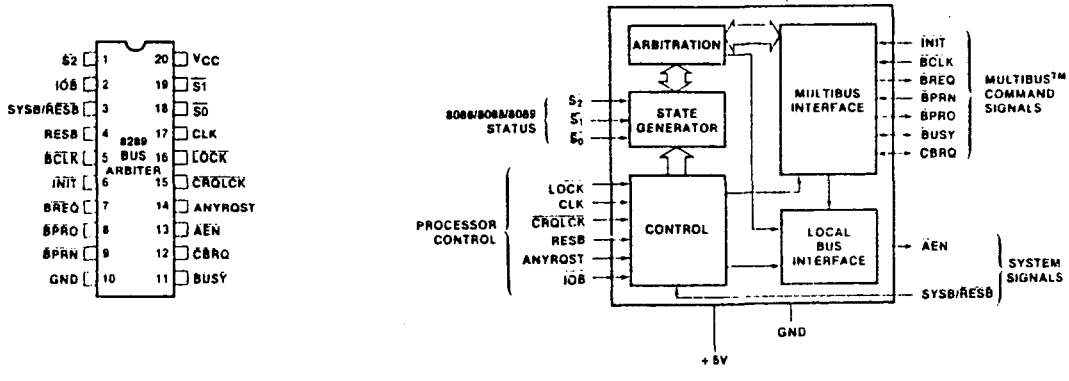
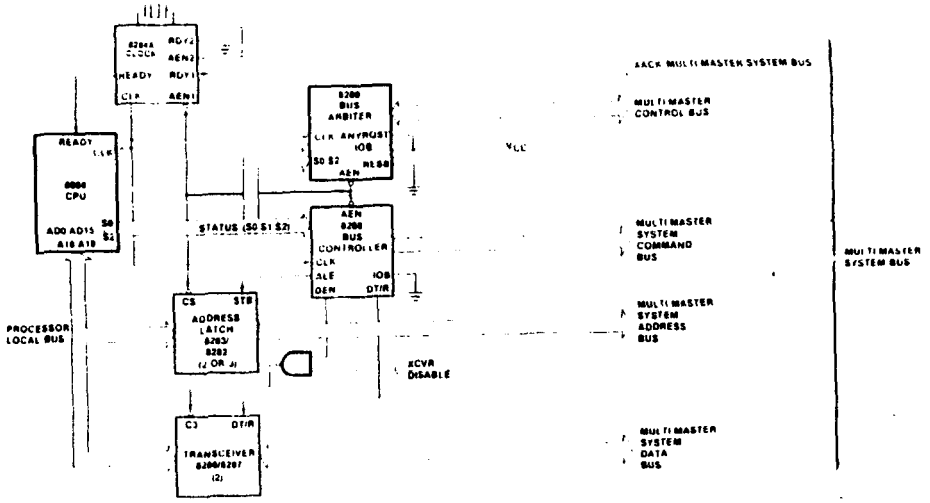
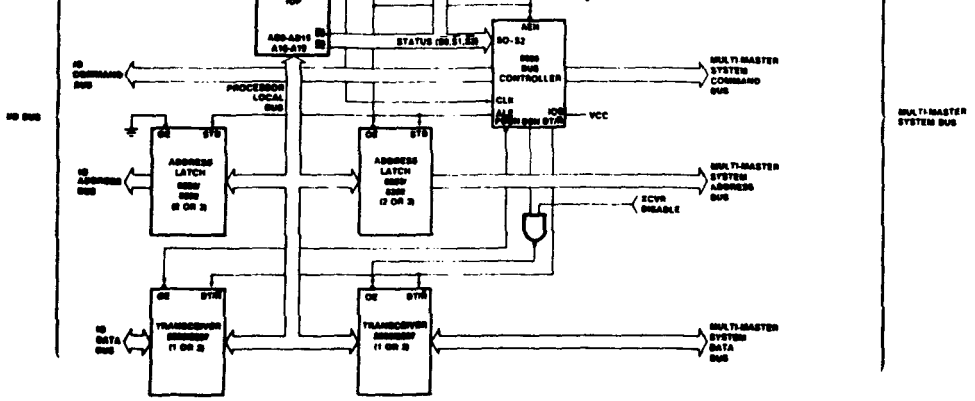


Fig. 7.3 Patillaje y estructura 8288 y 8289.

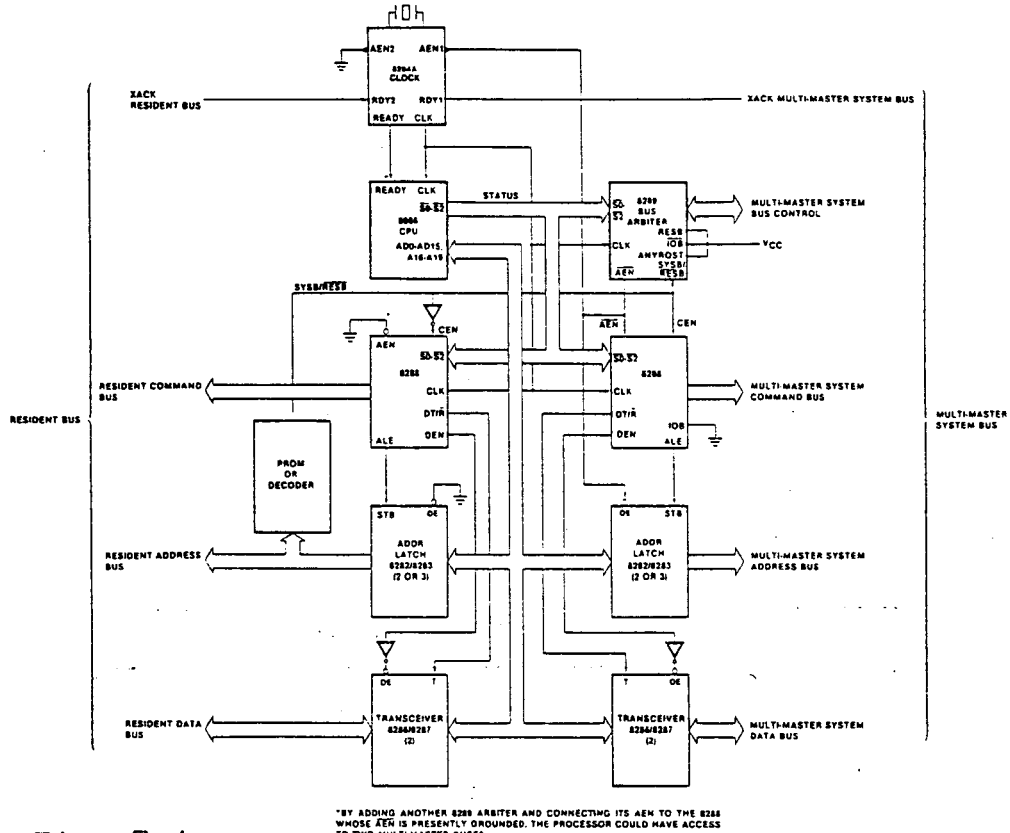




8289 Configured in I/O Bus Mode With 8089 I/O Processor

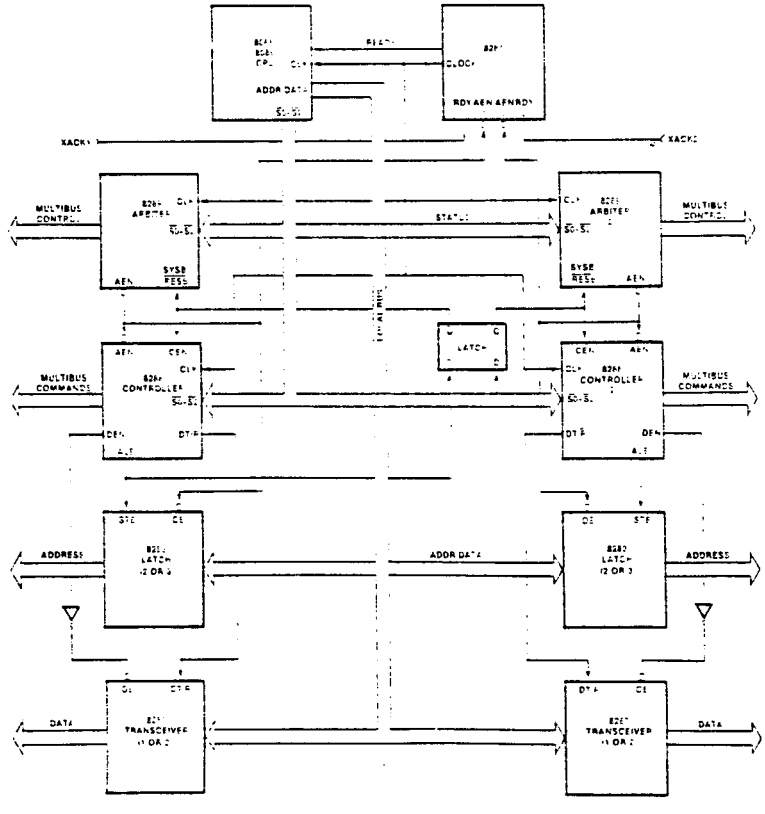
Fig. 7.3

Fig. 7.4



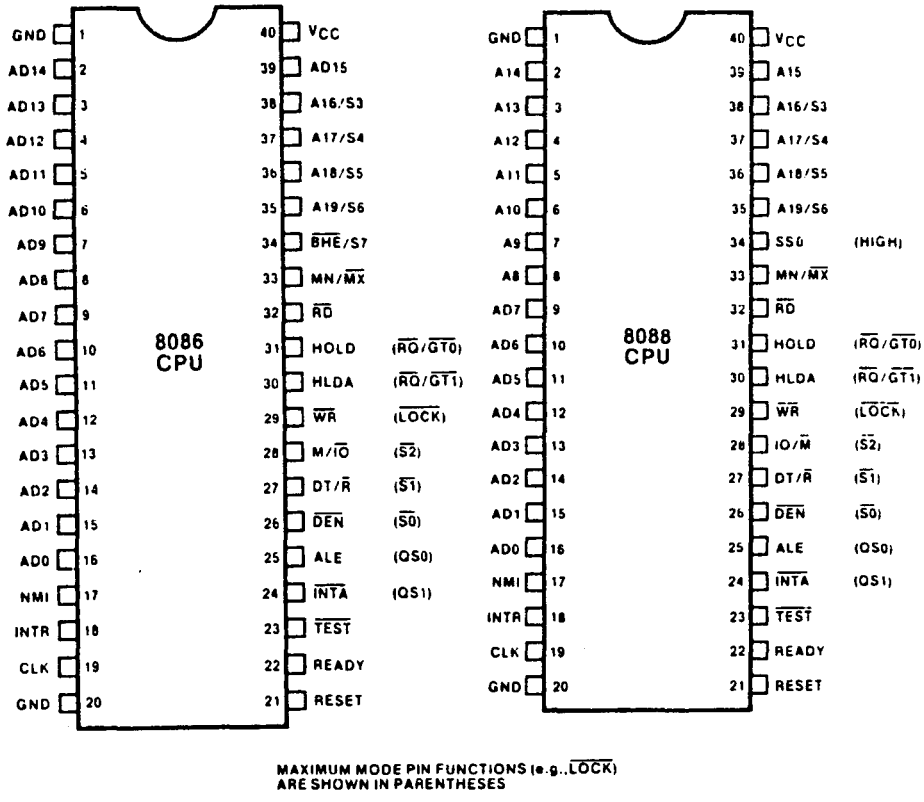
"BY ADDING ANOTHER 8289 ARBITER AND CONNECTING ITS AEN TO THE 8288 WHOSE AEN IS PRESENTLY GROUNDED, THE PROCESSOR COULD HAVE ACCESS TO TWO MULTI-MASTER BUSES.

Fig. 7.4.c



MEMORY MAPPING DECODING IS SHOWN TAKING PLACE DIRECTLY OFF OF THE PROCESSOR'S LOCAL MULTIPLEXED ADDRESS-DATA BUS.

Fig. 7.5



Patillaje del 8086 y el 8088. Establecer relaciones.

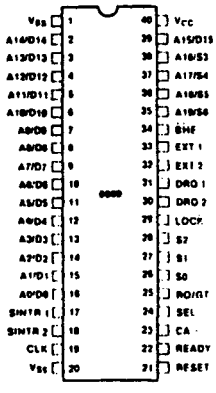


Fig. 7.6 Patillaje del 8089. Ver similitud con el 8086.

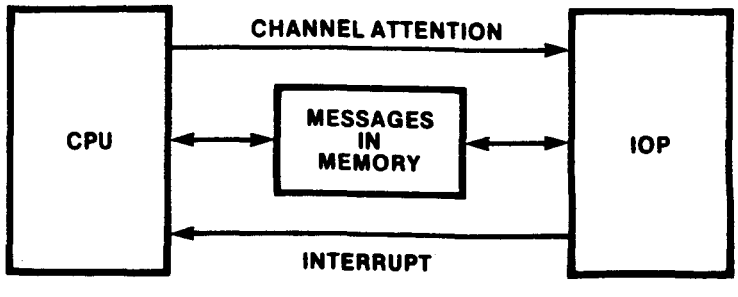


FIG. 7.7

CPU/IOP Communication

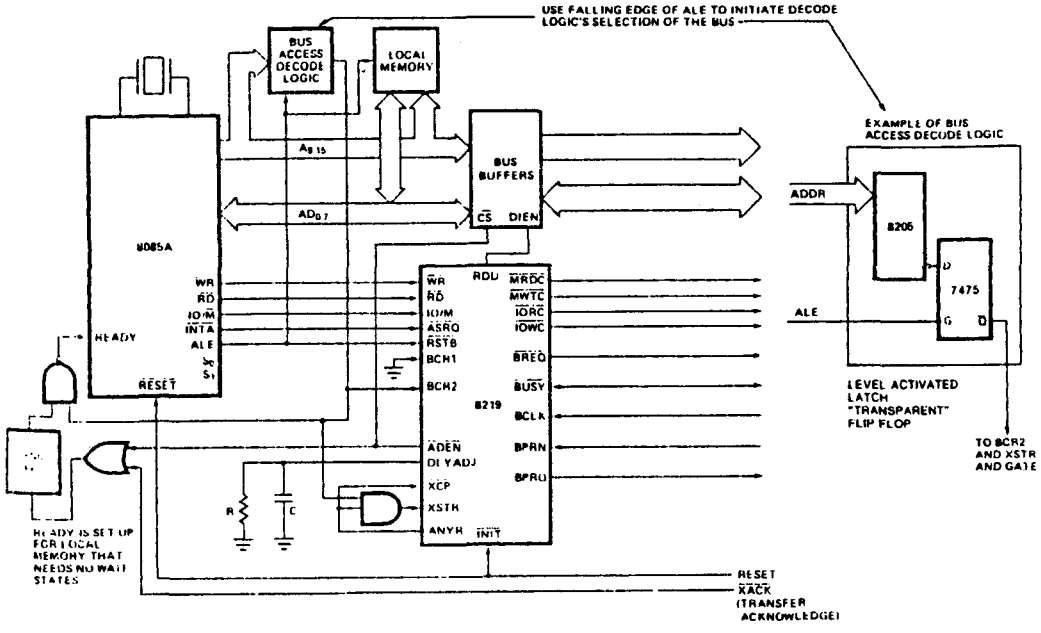
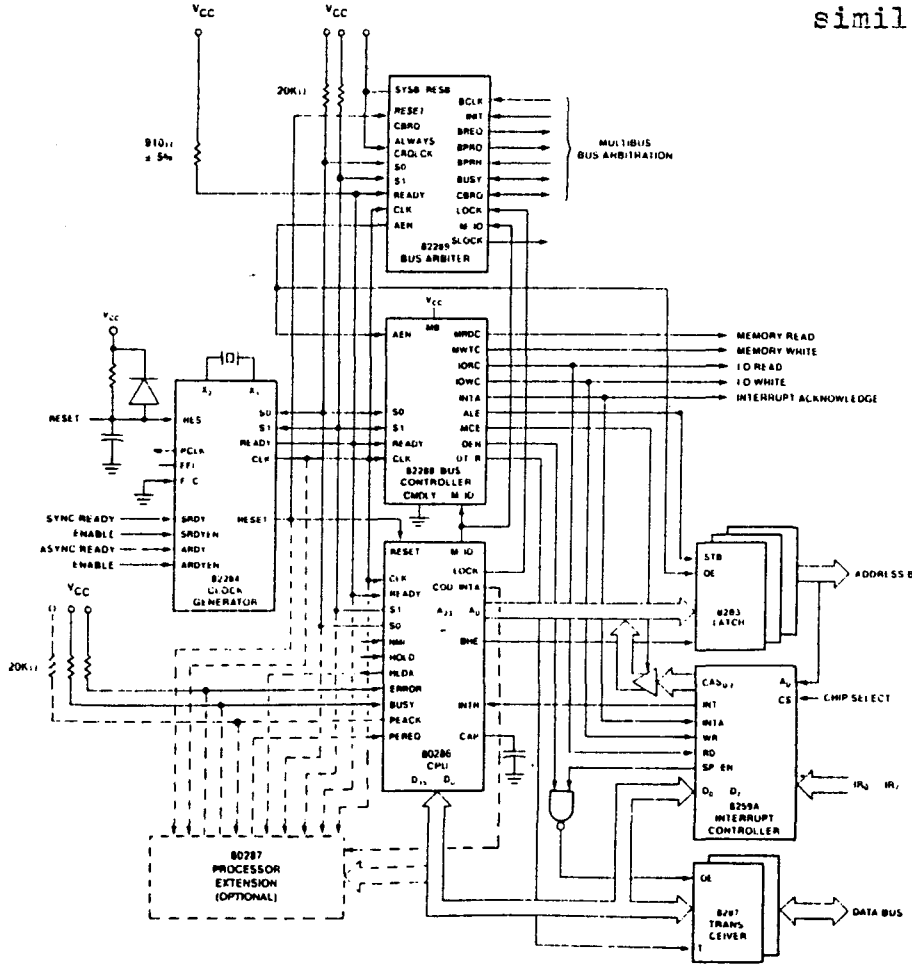


Fig. 7.8 Estructura de interfase para la familia MCS-85 al Multibus-I. No existe correlación con la iAPX.

Fig. 7.9 Con la familia del 80286 se sigue una similar estructura.



CAPITULO 8

CAPITULO 8. IMPLEMENTACION DEL PROYECTO

8.1. INTRODUCCION

Se procede a la discusion del dispositivo multiprocesador basado en el 8088 desarrollado en este proyecto, del software escrito para el, de las tecnicas de depuracion hardware y software empleadas, y de un pequeño analisis de prestaciones que se realizo.

Todos los detalles sobre la arquitectura hardware propios de la familia iAPX se remitiran al capitulo 7. Para algun detalle sobre el empleo del semaforo, remitirse al capitulo 5.

8.2. ESTRUCTURA HARDWARE.

La familia de procesadores iAPX-8X, iAPX-18X tiene una estructura muy apta para la implementacion de sistemas multiprocesadores. Para ello, se configuran en modo maximo (iAPX-8X), y se obtiene una cierta homogeneidad desde el punto de vista hardware, que se complementa con la compatibilidad "hacia arriba" del software. Esta propiedad ha sido empleada como base del desarrollo del hardware y se ha procedido a la separacion del bus de la CPU de la placa que contiene la logica de arbitraje y control del BUS LOCAL y PRIVADO. La razon fundamental reside en que una arquitectura asi, provee una muy flexible eleccion del microprocesador a usar con sus coprocesadores, sin cambiar la logica de arbitraje. Este bus de CPU consta de lo siguiente:

ADo AD7 .
A8 A19 . En sistemas de 8 bits. Atencion que las lineas A8-A15 permanecen "latcheadas" durante un ciclo de acceso a memoria, no como en los 16 bits. Esto implica la desabilitacion de los drivers que pueden en la placa base. Esto es muy importante. Es la unica diferencia Hardware realmente importante y caso contrario, se produciria una colision que destruiria el compontente.

BHEN/ . siempre a nivel alto en sistemas de 8 bits.

ADo AD15. En sistemas de 16 bits. Estas lineas son bidireccionales y tienen sus propios buffers.

BHEN/ . bus high enable. Senal que permite la compatibilidad con dispositivos de 8 bits, en un bus de 16 bits.

LOCK/,S0/,

S1/,S2/ . Lineas de estado a decodificar por el control del bus y el arbitraje.

RDY,CLK . Senales de ready y clock del sistema.

Debe tenerse en cuenta que este RDY es diferente en las familias iAPX-8X Y iAPX-18X, ya que la primera usa un chip externo (generador de clock 8284). Esta diferencia implica el cambio de un strapping en la placa de arbitraje para producir la funcion correcta del ready, que es combinacion del ready local, y el del sistema.

Las posibles combinaciones que actualmente se pueden soportar (combinaciones de procesadores y coprocesadores que

utilizan este bus de CPU) son:

CPUs	COPROCESADORES
8086	8089 I/O coprocessor
8088	8087 floating point coprocessor
80186	80130 iRMX-86 coprocessor
80188	80150 CP/M-86 coprocessor
8089	82586 Local Area Network coprocessor
	82730 Texts coprocessor

Para permitir el arbitraje en el bus de la CPU, se tienen las siguientes lineas:

QS0/,QS1/ : estado de la cola de prefetching (busqueda anticipada de instrucciones)

RQ0/-GT0/,
RQ1/-GT1/ : lineas de peticion y concesion de bus. Familia iAPX-8X.

HOLD/ : linea de peticion de bus. Familia iAPX-18X.

HLDA/ : linea de concesion de bus. Familia iAPX-18X.

TEST/,
BUSY/ : lineas de sincronizacion de procesadores.

El disemo implementado fisicamente consta de 3 placas, que pasare a detallar su funcion:

a) Placa de soporte al multiproceso.

Contiene los drivers, los controladores de bus (8288), el arbitro de acceso al multibus (8289), el clock (8284) y alguna logica adicional. Esta parte permanece inalterable con el cambio de micros, excepto la deshabilitacion del driver responsable de las lineas A8-A15, que no se usa en los sistemas de 8 bits y que puede provocar una colision en el bus local. La arquitectura es completamente convencional. Se puede soportar tambien el procesador de i/o 8089 en modo master, validando el SYS-RESB/ (senal que controla el acceso al bus local o al del sistema) con la salida S2/ del micro.

Tiene numerosos strappings que le permite:

Variar la politica de accesso al multibus,
incluir interrupciones a 2 niveles,
posibilitar la ampliacion a mas buses locales,
o que el bus local tenga el mismo arbitraje (bus local con soporte al multiproceso),
variar la prioridad en el acceso al multibus,
otras caracteristicas que le confieren gran flexibilidad.

b) Placa del procesador.

Contiene el microprocesador (cualquiera de los previamente comentados) y la logica de seleccion de bus (a partir de las lineas de direccionamiento y el estado S2/). Para el soporte de la familia iAPX-18X, habra que retocar algun pequeno problema en la placa de soporte. El acceso esta mapeado de tal forma que los 500k de memoria superiores y todos los puertos, esten asignados al bus local. El resto esta en memoria de sistema, excepto 2k situados en la segmento 0, para soportar las interrupciones.

c) Placa de memoria.

Contiene la logica de chip select, generacion de ready local, y los 2k RAM y 2k EPROM que contienen las interrupciones y el "booting". Esta actualmente configurada para sistemas de 8 bits, pero no hay ningun problema en la ampliacion a 16 bits.

La pauta de arranque exige la carga del lanzador y su arranque en el MDS-221. El sistema 8088 debe estar desabilitado (patilla de wait a nivel alto). En la eprom local se encuentra un programa que consiste en una instruccion de wait para contener el micro hasta que se haya procedido al arranque del MDS y la carga de los programas, y un salto entre-segmentos a la direccion 0A00:0000, donde se

encuentra otro salto hacia el comienzo real del programa.

El sistema esta enganchado a los conectores de expansion del multibus que posee el MDS. Este conectores son para expansion, no para enganchar sistemas. Los problemas implicados son evidentes, pero no se poseia de placa de pruebas multibus.

El hardware ha planteado numerosisimos problemas de fiabilidad y depuracion. Se solo un analizador logico de 16 canales y 1 nivel de trigger. Los testeos iniciales fueron realizados inyectando senales simuladas por un uC SDK85 a baja frecuencia, ya que la circuiteria de control de bus es estatica. Para solventar la baja memoria del analizador, se uso una tecnica de "reset pulsado", que consiste en aplicarle un tren de pulsos al reset, para conseguir una monitorizacion comoda.

Problemas de implementacion de la circuiteria, debidos a muchos factores pero fundamentalmente a la falta de un soporte fiable, elevaron varias veces, el tiempo de depuracion y puesta a punto de prototipo, e indujo a una actitud de "no mas hardware si seguimos asi".

Personalmente pienso que el uso de un emulador, y una implementacion mas fiable, hubieran podido rebajar el tiempo de desarrollo del prototipo unas 4 veces. El tiempo dedicado a la depuracion (no al diseno de las placas o la circuiteria) fue de aproximadamente 4 meses y medio.

Se incluye en un anexo la estructura de bloques del prototipo, mostrandose claramente, en que placas estan

situadas cada uno de los componentes basicos de control, memoria, cpu, etc...

En el esquema, aun cuando esta actualmente ilustrado para una interface con el 8089, no representa diferencias significativas para cuando se conecta el 8088. Tener en cuenta que, aunque se muestran los drivers en la placa de arbitraje para las lineas A8-A15, estos drivers no estan montados (solo estan los zocalos para su emplazamiento). Se puede ver tambien todas las opciones de strapping que permiten multiples configuraciones en el control.

8.3. SOFTWARE DESARROLLADO.

Se pretende unicamente describir funcionalmente el software mas importante que se desarrollo. No se entra en explicaciones pormenorizadas ya que la mayoria del codigo esta redactado en lenguaje de alto nivel PLM-80 o PLM-86, y los programas han sido suficientemente comentados para que puedan ser considerados como auto-explicativos. En ciertas circunstancias, se daran unas breves instrucciones de uso.

En las distintas fases del proyecto, se generaron dos clases de programas.

1) para posibilitar el manejo facil del 8089, se desarrollo un ensamblador 89 y un programa de ayuda a la depuracion.

2) un monitor de depuracion 8088, soporte ISIS-II, traductores de codigo hexadecimal, reubicador de tope de memoria logica del sistema MDS-221, y lanzadores de los programas multiprocesador.

Este software aqui descrito es una parte del total que se desarrollo. En particular el ensamblador 8089 paso numerosos programas de prueba para contrastar su fiabilidad. Para el 8088, se desarrollaron varios programas cortos que fueron grabados en EPROM, para verificar el correcto acceso a la memoria, o las transacciones con el multibus, es decir, a efectos de depuracion hardware. Asi mismo, se desarrollo un pequeño programa que corria en un SDK85 para simular el comportamiento de un ciclo de acceso a memoria por parte de un dispositivo master de la familia iAPX. En resumen, este programa era un inyector de señales que enviaba secuencias previamente almacenadas en una tabla. Igualmente se desarrollo un modulo de comunicacion 8088-8080 simplificado antes de pasar a la version definitiva a efectos de facilitar la depuracion que fue bastante ardua. Se pasa a describir mas detalladamente cada parte del software generado.

ESTRUCTURA SOFTWARE GENERADA PARA EL 8089.

Para el 8089 se desarrollo un ensamblador 89 en PLM80 para que se facilitara la labor de programacion con este. Este ensamblador fue desarrollado antes que los primeros problemas se hubieran planteado, ya que de otra forma, probablemente no se hubiera hecho. El ensamblador es muy modular. Esta dividido en tres modulos, XASM89.SRC, XCOD89.SRC y XLIB89.SRC, siendo el primero, el programa principal, el segundo el generador de codigo, y el tercero, las subrutinas de soporte y utilidades. La entrada es ensamblador 8089 con ciertas limitaciones expuestas en el listado, y se genera una

salida listada, y un fichero compatible con el formato hexadecimal 8080.

Realmente, no se penso inicialmente que se pudiera haber desarrollado en otro lenguaje de programacion de los que actualmente soporta el MDS. Hubiera sido mas factible emplear el PASCAL80.

Para ayudar a la depuracion, se genero un monitor cruzado para control del 8089, el cual era una adaptacion del mostrado en <1>, por lo que se puede remitirse alli, ya que, en estos ultimos dias se produjo una perdida del listado fuente de ese programa y no se esta en condiciones de repetirlo unicamente a efectos ilustrativos.

ESTRUCTURA SOFTWARE GENERADA PARA EL 8088.

Se describira inicialmente la estructura global del monitor 8088 y luego se entrara en los otros programas generados.

Fundamentalmente, hay dos tipos de programas, los que se ejecutan por el 8088 y los que se ejecutan por el 8080 del INTELLEC MDS221. El software fue construido de forma que el MDS asuma las operaciones de sistema operativo que otros procesadores le manden,a traves de una interface comun independiente del codigo,que posteriormente se detallara. El ISIS-II, sistema operativo del MDS no esta pensado para el soporte de sistemas multiprocesadores, y por tanto, sus posibilidades estan algo restringidas. Todos los programas se ejecutan en la memoria del MDS, compartiendo por tanto este

recurso. Se desarrollo todo el codigo del 8088 en PLM-86, por su comodidad, y el codigo del 8080, por razones de eficacia y brevedad, esta en ensamblador 80. Para el 8088, hay un monitor,basado en el SDK86, que contiene ciertas mejoras en los drivers de entrada y salida, y tiene las facilidades adicionales que le provee un sistema operativo.La interface con el sistema operativo tambien fue redactada en PLM86, implementando el semaforo de acceso a la region critica por medio de la funcion LOCKSET. Para el 8080, se desarrollo el software de comunicacion que permite pasar parametros desde el monitor 88 y el ISIS-II. Este punto se vera con mas detenimiento.

Mapeado

Como ya se menciono, todos los programas residen en los 62k RAM del sistema de desarrollo. Hay dos excepciones para el codigo del 8088. Existe un "booting" en una memoria local de 2k EPROM en la zona alta de memoria (0FF800H-0FFFFFH) para inicializar el sistema y contener al 8088 para que se puedan cargar el monitor 88 y el interface del ISIS-II. Existe tambien una memoria de 2K RAM, local tambien, mapeada entre (00000H-007FFH),para soportar pequenos programas locales y la tabla de interrrupciones necesaria para soportar ciertas funciones del monitor.

Interface de comunicacion

La comunicacion del monitor 88 y el ISIS-II se realiza a traves de mensajes en memoria y la activacion de flags y un semaforo que posibilitan asi la independencia del codigo usado en cada uno de los procesos.Seguidamente se describe

someramente la naturaleza de la interface, usando las primitivas de Dijkstra P( ), y V( ), para indicar el cierre o liberacion de un semaforo.

Codigo de otro procesador (ej 8088) Codigo interface ISIS-II (ej 8080)

```
.                                          .
.                                          DESACTIVA FLAG
P(SEMAFORO);                               V(SEMAFORO);
CARGA PARAMETROS AL ISIS-II          LOOP1: DO WHILE FLAG  NO
                                              ACTIVADA;END;
ACTIVA FLAG DE ARRANQUE                    CALL ISIS-II;
DO WHILE FLAG ACTIVA;END;                  DESACTIVA FLAG;
LEE ERROR                                  GOTO LOOP1;
V(SEMAFORO);                               .
.                                          .
```

Observese como el arranque del ISIS-II, se efectua mediante la activacion de una flag; que el acceso de la zona del paso de parametros, esta bajo un lock (P(SEMAFORO)); que el codigo 8080 se ejecuta primero, y por tanto inicialmente el semaforo esta liberado (V(SEMAFORO)) y la flag desactivada. La interface no es complicada y el uso de semaforos permite la expansion a un mayor numero de procesadores.

Posibilidades y Mejoras

Debido a que la comunicacion entre procesos se efectua enteramente por software,esto implica un uso del bus para muchos ciclos potencialmente inutiles, como por ejemplo,el testeo de la activacion del flag de arranque del ISIS-II. Una mejora sustancial podria implementarse haciendo que el ISIS se activase bajo una interrupcion comandada por el master que actualmente este en la region critica. Sin embargo el ISIS-II tiene, segun el manual, dificultades para la correcta

operacion bajo interrupciones, y el tema habria que estudiarlo mas de cerca. El problema de la infiabilidad del hardware, es lo suficientemente fuerte como para hacer desistir cualquier ampliacion de este.

El sistema, ya de por si, podria cargar inicialmente, un programa de codigo 8088 y posteriormente un programa 8080 para la realizacion de investigaciones en la comunicacion entre procesos, pero eso probablemente desbajustaria la interface con el sistema operativo, con los consiguientes problemas.

Deberia plantearse el estudio de estos problemas, ya no bajo la perspectiva actual frente a un sistema limitado, sino con un sistema operativo capaz, (probablemente iRMX 86) que permita perfectamente el ernvio de mensajes, el control de multiprocesos, y otras funciones tipicas en sistemas operativos disenados especificamente para ese soporte.

Los otros programas desarrollados fueron:

-un lanzador (RUN88.SRC) que carga directamente los distintos programas del 8088 y el 8080 en la memoria del MDS-221, y arranca el segundo. La mision de este lanzador es simplemente, la de facilitar el lanzamiento del sistema (carga de los dos programas de control).

-un traductor de codigo para transformar el codigo hexadecimal generado por el comando OH86 (conversor de codigo objeto absoluto a hexadecimal formato 8086) al formato hexadecimal tratable por el ISIS-II. El formato fuente esta descrito en el listado de (LOADER.SRC). La razon de la

existencia de este programa se fundamenta en la necesidad de compatibilizar los codigos generados por el PLM-80 (y ASM-80), y el PLM-86, para que se puedan cargar (que no ejecutar) varios programas sin necesidad de usar cargadores distintos. Su tratamiento esta basado en las subrutinas de manejo hexadecimal del monitor serie del SDK-86.

8.4. DEPURACION

Debido a la falta de herramientas adecuadas para la depuracion hardware y software de esta clase de sistemas iAPX en nuestra escuela, y a la falta de repuesto lo que implicaba que la destruccion accidental de un componente podia conducir a la paralizacion (como de hecho llevo) del proyecto, se procedio a la depuracion de forma extremadamente cauta. Asi, por ejemplo, en el desarrollo de la placa de arbitraje prototipo, se procedio a los siguientes pasos en la depuracion hardware:

1)Prueba de cortocircuitos y continuidad.

2)Verificacion conmutacion de buses en los drivers del multibus.

3)Comprobacion funcionamiento reloj.

4)Sometimiento del controlador de bus local 8288 a pruebas de comportamiento, simulando S0/, S1/, S2/, y CLK con un SDK-85 como generador programable de secuencias. No hay problemas de velocidad dado que la circuiteria es estatica.

5)Montaje parte local del sistema, con la logica de conmutacion de buses desabilitada. Programas locales.

6)Verificacion de las corrientes en los drivers para cotejar los efectos que una posible colision podria tener en el MDS-

221. Esto era un factor muy importante, ya que todas las modificaciones HARDWARE de agregacion, debian tener plenamente probadas su incapacidad de producir damos permanentes al MDS. Para ello se midio la corriente de cortocircuito en dos drivers de salida, forzados en colision. La prueba, pasada satisfactoriamente, elimino la precaucion de una posible red resistiva limitadora de la corriente.

7)Comprobacion acceso a multibus en lectura. Analisis de las senales de peticion y concesion de bus.

8)Acceso total al sistema.

En ciertos casos, como el apartado 5 de la depuracion, se procedio a una tecnica de reset pulsado, consistente en introducir un tren de pulsos al reset para conseguir del micro un comportamiento "cuasi-estable" cuando este trabajara en modo local, y se facilitaran las tareas de monitorizacion (se pueden comprobar asi facilmente los accesos). El analizador logico que se posee, solo tiene 16 canales y un nivel del disparo, lo que limita las posibilidades de encontrar los fallos. Estaba enganchado a las lineas AD0-AD7 y S0/, S1/, S2/, LOCK/, y CLOCK del micro, y en otros puntos criticos del circuito. Se exponen a continuacion algunos de los errores (y consecuencias) que se detectaron:

-Error en conector P14.

-Error logica de control de los buses del sistema.

-Error en bus de CPU. No se habian desabilitado los drivers correspondientes a AD8-AD15, y se produjo una colision con el 8088 empleado. Esto significo la destruccion de este

componente. Afortunadamente, se poseia otro 8088 (solo uno mas).

-Error frecuencia clock. Se bajo al minimo con un cristal de 6.144Mhz.

-Error direccionamiento eprom local.

-Error generacion ready local.

-Errores fantasmas debido a falsos contacos, debidos a las "peculiaridades" de la implementacion. Aun subsisten.

Para la depuracion del software, se siguio el siguiente esquema:

-Estudio en estatica del comportamiento del programa.

-Uso de programas de depuracion que comprueban el comportamiento correcto de las subrutinas en un "ambiente cerrado". Eso a veces tiene sus problemas. Algunos quebraderos de cabeza no estaran en las subrutinas en si, sino en los programas que se elaboraban lo mas simple posible para eliminar posibles errores en ellos.

-Desarrollo siguiendo la metodologia "TOP-DOWN".

La lista de errores software detectados siguiendo este esquema es lo suficientemente extensa como para que no tenga cabida.

Como ya se ha comentado previamente, es practicamente imprescindible un conjunto mas potente de dispositivos de depuracion (analizadores logicos, emuladores, etc...) para que el desarrollo de arquitecturas, tenga un cierto nivel de "productividad".

8.5.ANALISIS DE PRESTACIONES

Se procedio, al final del proyecto, a un rudimentario analisis de la degradacion del sistema. La idea inicial era la de contrastar los resultados mostrados en el capitulo 4. Usando una trampa, podemos llegar a lograr que hasta 3 procesadores trabajen concurrentemente. El primer procesador seria el de la IPB del MDS, pero al no posee memoria privada, no es susceptible de empleo como "generador de carga". El segundo procesador es el 8088 desarrollado. El tercero, y aqui viene la trampa, seria el emulador 8085 que se posee. Con la ayuda de un pequeno sistema minimo, se puede tener memoria local, y la memoria del sistema o compartida se mapearia como de intellec en las instrucciones de emulador. Una vez que el programa de carga empieze a funcionar con el comando go, observese que el led indicador del estado de run del 8080 de la IPB, esta apagado, y que unicamente se enciende cuando se entra en modo comando, lo que lleva a sospechar que es el ISIS-II el que asume las operaciones de control del sistema, y que el procesador 8080 esta desconectado cuando funciona el emulador para impedir contencion que bajen la respuesta de este a circunstancias en tiempo real. El paso se efectua mediante interrupcion al tocar el teclado (INT7). Si procedemos a una interrrupcion i mientras el emulador esta en go, se produce una nueva carga del ISIS-II, pero el emulador SIGUE FUNCIONANDO y accediendo a la memoria del sistema, aunque no tiene lock en su acceso. Esto fue comprobado y se hicieron algunos trucos. La forma de pararlo es aplicandole un reset al sistema. Se puede asi simular una nueva carga. De todos modos, hay que analizar los

estados de wait inducidos unicamente por el emulador. Eso no ha sido estudiado.

Se procedio a la realizacion de un programa que corria bajo el 8088 que efectuaba un cierto numero de bucles en memoria privada, y otro en memoria del sistema. Se analizo como variando la carga hacia el sistema (se ejecutan el mismo numero de instrucciones aproximadamente, pero ahora hay mas accesos a memoria del sistema) se bajaba considerablemente el rendimiento. (El tiempo invertido casi subia en una magnitud comparado con el proceso en memoria puramente privada). Para amedicion se procedio al analisis de la senal de conmutacion de buses SYS-RESB/ (remitirse a los planos). Estas medidas no pueden considerarse como exponente real del comportamiento en condiciones normales, y ademas, no encontraria los resultados expuestos en el capitulo 4. Las razones para esta afirmacion son:

-En los resultados de <4> se supone una distribucion uniforme de las cargas software, y procesadores iguales. Esta claro que aqui no lo son.

-El comportamiento del sistema debe ser extraido como media en un cierto rango estocastico que simule desde el punto de vista estadistico, el comportamiento de programas reales. Este software no es en absoluto estocastico.

Las asumciones de estos resultados estan contenidas en la bibliografia del capitulo 4.

BIBLIOGRAFIA CAPITULO 8

<1>: ver referencia <1> capitulo 7.

<2>: ver referencia <2> capitulo 7.

Las siguientes referencias pertenecen al manual de uso del MDS-221, editado por la Intel corp.

<3>: Intellec series II microcomputer development system hardware reference manual.

<4>: Intellec series II microcomputer development system schematic drawings.

<5>: Intellec series II microcomputer development system interface manual.

<6>: Intellec series II microcomputer development system monitor listings

<7>: ICE85 in-circuit emulator operating instructions for ISIS-II users.

<8>: PL/M-86 programming manual for 8080/8085 based development systems.

<9>: PL/M-86 User's guide.

CAPITULO 9

CAPITULO 9. ELEMENTOS PARA EL DESARROLLO DE UN CONTROLADOR FLOPPY PARA EL MDS-221

9.1. INTRODUCCION

En este capitulo, se expone los factores fundamentales y los trabajos desarrollados para la construccion de un controlador de floppys para el sistema de desarrollo MDS-221. La explicacion de no haber efectuado este diseno sera comentada en las conclusiones.

9.2. REQUISITOS DE LA UNIDAD DE DISCOS PARA EL MDS 221

Todas las operaciones deben haber sido iniciadas por la unidad central de proceso. Una vez iniciada, el controlador completa la operacion especificada sin ninguna posterior intervencion de la cpu. Desde el punto de vista de la cpu, hay tres pasos para realizar una operacion en disco.

-La CPU debe preparar y guardar en memoria de sistema un bloque de parametros de I/O (IOPB) para cada operacion a ser realizada. Un IOPB consta de siete bytes y especifica la operacion en particular y los parametros requeridos para la operacion.

-La CPU debe pasar la direccion de memoria del IOPB al controlador.

-La CPU debe procesar el resultado de la operacion proveniente del controlador despues de la finalizacion de la operacion.

Estos siete parametros que forman el IOPB deben adaptarse al siguiente formato:

byte 1: comando de canal
" 2: instruccion a disco " 3: numero de records
" 4: direccion de track
" 5: direccion de sector
" 6: direccion de buffer (baja)
" 7: direccion de buffer (alta)

La preparacion del IOPB por la CPU no requiere interaccion con el canal controlador de diskette. El paso de la direccion de memoria donde reside el IOPB, requiere, sin embargo, interaccion entre ambos. Se implementan 6 comandos de canal que permiten realizar estas operaciones interactivas. Tres de los comandos son el resultado de ejecutar una operacion de salida en un puertos I/O dedicado, mientras que otros 3 comandos son el resultado de una operacion de entrada en otros puertos dedicados. Estos puertos radican a partir de una direccion de base. El puerto resulta de la suma de un dezplazamiento a esta base. En operacion con el ISIS-II, esta base debe estar fijada en la posicion 78H. Los seis comandos de canal son:

1-Escribir direccion de memoria baja (salida)
2-Escribir direccion de memoria alta y comenzar la operacion (salida)
3-Resetear el canal (salida)
4-Leer el estado del subsistema (entrada)
5-Leer tipo de resultado (entrada)
6-Leer byte de resultado (entrada)

El comando de canal permite que el controlador pueda:

-Determinar el metodo de asignacion de direcciones logicas de sector.

-Habilitar o desabilitar una serie de posibles interrupciones.

-Determinar la longitud de los datos a ser transferidos.

La CPU transmite la direccion del IOPB ejecutando los comando 1 y 2. Despues de la ejecucion del comando 2, el

controlador pedira el control de master del Multibus, extraera la instruccion de disco y los parametros asociados del IOPB, y procedera a la realizacion de la determinada operacion. Las operaciones que se realizan serian:

1-Recalibrar (ir a track 0)
2-Posicionar cabeza
3-Formatear un track
4-Escribir datos (con marcas de datos)
5-Escribir datos (con marcas borradas)
6-Leer datos
7-Verificar CRC

El controlador puede interrumpir la CPU cuando la operacion finaliza o cuando el estado de ready del disco cambia. El software de sistema del host puede implementar su interrupcion ya sea por interrupcion hardware directa o testeando reiteradamente ("polling") el comando de canal de lectura de estado.Se pueden asignar distintas prioridades de interrupcion. En el caso del ISIS-II la prioridad de la interrupcion debe ser 2 (INT2/). Cuando la CPU haya determinado la finalizacion de la operacion, la CPU debera ejecutar los comandos 5 y 6 en este orden para determinar si la operacion ha sido correctamente ejecutada, y si no, que tipo de error ha ocurrido.

Asi, en resumen, ciertos comandos de canal son ejecutados por la CPU para senalar al controlador un IOPB residente en la memoria del sistema, e inician la secuencia de operacion. El controlador de canal, entonces, accede al IOPB para realizar la operacion especificada en el byte de instruccion. El controlador generara, suponiendo que estuviese habilitada por la CPU, una interrupcion indicando la finalizacion de la operacion o la deteccion de un error. La CPU, entonces,

ejecuta otros comandos de canal para determinar el resultado de la operacion del disco.

Un ejemplo practico de una secuencia de este tipo, se encuentra en los listados del monitor del MDS-221 <2>, concretamente en las lineas 611 hasta 642.

Debe tenerse en cuenta que la inclusion de la unidad de discos, reasigna el dispositivo que actualmente recibe el nombre :F0: y lo transforma en :F4:. Una muestra de las distintas configuraciones de floppys y otros perifericos que pueden soportarse bajo el ISIS-II esta en <1>. Uno de los nuevos discos tomaria entonces la denominacion de :F0: y ahi residiria el disco de sistema.

9.3. RESUMEN DE LA ESTRUCTURA DEL CONTROLADOR DE DISCOS.

El controlador estaria basado fundamentalmente en una pastilla que se encargara de los problemas asociados con los formatos de las pistas en las unidades de disco, que distan mucho de ser evidentes, como podra observarse en la bibliografia reservada. Se emplearia un microprocesador que o bien tuviera velocidad suficiente para hacer la transferencia en modo "polling", o que existiera un dispositivo que pudiera hacer una transferencia DMA. Estas alternativas se tratan con mas detalle en el siguiente punto donde se toman en consideracion distintos factores que influyen en el diseño. Ademas de estos dos elementos basicos (con lo que se evita la necesidad de un hardware o software fuera de lo "normal"), se tendria que incluir un controlador de multibus, ya sea el 8289 para la familia iAPX, o bien el 8218 para la familia

MCS-85. Se necesitaria una cierta memoria local donde residirian los programas de control e interfase. Esta memoria local seria en su mayoria ROM pues se necesitan muy pocos bytes para acumular todas las variables en juego. Existiria una logica de conmutacion de buses segun el direccionamiento y de seleccion de pastillas. En funcion del controlador de disco empleado, se podria necesitar un PLL para extraer los datos, ya que estos van en doble densidad y el procedimiento de extraccion es mas complicado que en simple densidad.

9.4.ALTERNATIVAS PARA EL DESARROLLO DE UNA UNIDAD DE DISCOS PARA EL MDS-221.

En la realizacion de la circuiteria de control multimaster y transferencia DMA de la unidad de discos, se plantean multiples posibilidades a ser consideradas. Estas posibilidades no fueron vistas en la primera fase del proyecto, sino mas bien avanzado este, una vez visto y analizado multiples aspectos que se implican mutuamente. Fundamentalmente, y en base a la experiencia que se posee, nos podemos plantear dos soluciones:

1) Solucion basada en la MCS-85.

2) Solucion basada en familias iAPX.

Se plantea ahora el analisis de cada una de las posibles opciones que se pueden encontrar.

1)SOLUCION.

El problema fundamental que se encuentra con el uso del procesador 8085 y su familia, es el reducido espacio de direccionamiento que se dispone. Los requisitos que se tienen

impuestos incluyen un espacio de un total de 64 K. Por supuesto, se necesita espacio de direccionamiento local para la ejecucion del programa de interfase y control. En tal caso, esta claro que no se podria afrontar esta solucion con un 8085 sin exponernos a grandes complicaciones en la logica de seleccion de bus y chip. Hay una alternativa en el que se supone bastante razonablemente, que el espacio de direccionamiento es excesivo en la gran mayoria de los casos. Podemos arriesgarnos tratando de usar un sistema con menor capacidad de direccionamiento asumiendo que en muy pocas ocasiones puderan surgir problemas. En particular, seria muy extraño que los 2 K superiores del sistema que no son RAM, sino que es donde reside el monitor en ROM, no sera n cargados de disco, lo cual es algo muy creible. Asi se dispondria de una zona de 0 a 62K-1 de direccionamiento disponible para el acceso a la memoria del sistema, y los ultimos 2K se tendrian como memoria local para el programa de control. Se plantea un problema ahora. El 8085 arranca en 0000H con lo que en tal caso, se trataria de ejecutar algo que no es codigo local. Sin embargo, el MDS carga en esta posicion un salto a la posicion FED4H donde se encuentra el monitor para dar respuesta a la interrupcion 0 del panel frontal, y alli se encuentra tambien la memoria local. Pero cuando se enciende por primera vez el MDS, esas posiciones no han sido cargadas todavia cuando el 8085 hace uso de ellas. Una solucion se encontraria modificando el arranque del sistema. Nuestra placa incluiria un interrupcion de activacion que permitiria el arranque o no de la unidad de

control de disco. Entonces, inicialmente, se cargaria con nuestra placa desactivada, el monitor sin ningun disco en la unidad :F4: (disco integrado). Esto cargaria el salto en RAM. Posteriormente solo volveriamos a activar nuestra placa y resetear el sistema con lo cual, se produciria un arranque correcto de todo el sistema. Por supuesto, la unidad :F0: tendria cargado el disco con el ISIS-II. No habria problema con los procedimientos de arranque del MDS. Para una descripcion de estas, remitirse a <3>. Es recomendable leersela pues es bastante curiosa. Tambien es plausible un arranque tal como lo hace el MDS-221, lo cual eliminaria la parte manual del arranque.

Ademas de esta alternativa se podria tratar de efectuar un "remapeado" con una PROM bipolar que reasignara el direccionamiento de acceso al sistema, de tal manera que no hubieran interferencias con este. Podemos describir sus caracteristicas fundamentales:

1.ALTERNATIVA.

-poco hardware.

-el soporte de las interrupciones (en caso de que las hubiera) implicaria escribir los vectores de salto en la memoria del sistema, concretamente en las posiciones de memoria 24H,2CH,34H,3CH correspondientes al TRAP y otras. Afortunadamente hay espacio para todo sin interferir en los vectores del sistema en si, pero el usuario tendria que tener cuidado con el empleo de estos.

-el arranque se complica un poco.(susceptible de eliminarse)

2.ALTERNATIVA.

-inclusion de hardware.

-soporte transparente de interrupciones.

-arranque transparente.

-calculo de "aliasing" o dezplazamiento debido al remapeado de la memoria en todos los accesos a memoria del sistema.

Para el soporte de la transferencia DMA, suponiendo que empleamos el integrado 8272, tenemos otras dos alternativas principales:

1)implementacion con un dispositivo DMA especializado tal como el 8237 o el 8257. Con el primero no habria problemas; con el segundo tendriamos que fraccionar los bloque ya que la maxima transferencia que soporta es de 16k seguidos, lo que implicaria una sobrecarga del software.

2)emplear un 8085 AH-1 o 8085 A-2 que corren a 6 y 5 Mhz respectivamente. El paso datos se haria previo testeo de un bit que posee el 8272 indicando la presencia de uno de estos. La interrupcion solo se usaria para deteccion de condiciones anomalas en el disco. Podria haber problemas con estados de espera en la memoria del sistema, o de sincronizacion incluso. Por eso, esta solucion es un tanto mas arriesgada, aunque tiene menos hardware.

En todos los casos, el acceso DMA a memoria del sistema implicaria un lock para prevenir la contencion impuesta por los demas dispositivos. La ventaja fundamental del empleo del 8085, es que se posee todas las herramientas de desarrollo a nuestro alcance, y por tanto, los problemas de depuracion podrian quedar reducidos. Sin embargo, y debido a los

problemas de respuesta en tiempo-real del emulador, probablemente las pruebas en tiempo real con el 8085 AH-1 se harian por separado.

2)SOLUCION

El empleo de la alternativa iAPX elimina casi todos los problemas que implican el uso de la MCS-85 y que se derivaban de su baja capacidad de direccionamiento, pero anade el nada desdenable problema de la depuracion, ya que se carece actualmente de soporte adecuado para el desarrollo de sistemas con esta tecnologia. Hay tres posibilidades:

1)8089 2)8088 3)80188

PRIMERA POSIBILIDAD.

El empleo del 8089 es, a mi juicio, la solucion mas razonable pues posee todas las caracteristicas para una correcta operacion. Solo podrian haber problemas en la inicializacion, facilmente solucionables. No se cuenta apenas con soporte por lo que se realizo un depurador y un ensamblador. Tiene un elevado precio.

SEGUNDA POSIBILIDAD.

Emplear un 8088-1 con solucion analoga al 8085 AH-1. Tambien se podria emplear un 8088 sincronizando las transferencias con WAIT. Es una alternativa arriesgada pero podria funcionar. El uso del 8088 radica en que todos los dispositivos en juego son de 8 bits. Podria haber problemas con las interrupciones que se resolverian de forma analoga al 8085.

TERCERA POSIBILIDAD.

Es casi matar una mosca con un cañon. El 80188 posee dos canales de DMA de alta velocidad, pero se malgastan otros recursos. Sin embargo, en caso de que se decidiera tener incluso otros perifericos enganchados a este, podria ser razonable. Incluso esta tendencia se destaca en diferentes placas OEM que fabrica Intel, y que estando relacionadas con dispositivos de I/O, emplea el 80188 o el 186. Tambien es caro.

9.5.SOPORTE HARDWARE ADICIONAL.

El controlador debe ser capaz de dar determinadas respuestas que no requieran necesariamente la atencion del microprocesador encargado del control del dispositivo. Estas respuestas se consiguen con un soporte adicional hardware integrado por logica SSI y MSI fundamentalmente. Dado que no se implemento en hardware esta parte, no considero que sea planteable la exposicion de la circuiteria que desarrolle sobre el papel puesto que no esta depurada y no conozco los problemas que pudiera ocasionar. Sin embargo, esta circuiteria es muy parecida al soporte que la unidad microprogramada que actualmente tiene el controlador. Para una referencia a ella, remitirse a <4> y <5>.

9.6.SOPORTE SOFTWARE.

La inclusion de una pastilla controladora de floppy, elimina muchos de los problemas de programacion que son debidos a la peculiar estructura de las pistas. Un resumen sobre la estructura de estas pistas puede ser encontrada en

<1>. A su vez, podria consultarse <7> en donde se muestra la construccion del software de control para un sistema basado en el procesador 6800. Asi mismo, en <5> pueden observarse los diagramas de flujo del microcodigo que se encarga de la operacion del controlador. En esencia, el software del sistema, independientemente del microprocesador empleado, tendria la misma estructura de bloque, a excepcion de una zona de inicializacion del dispositivo. Otras diferencias vendrian de la interpretacion de los datos provenientes del controlador para adaptarlos al formato que necesita el ISIS-II.

9.7. CONTROL Y CONTROLADORES DE FLOPPY.

En general, las senales que se precisan para el control de un floppy, podrian quedar definidas asi:

-carga de cabeza: controla la posicion de la cabeza para que empieze a leer datos del disco. La cabeza debe estar arriba cuando se dezplaza pista a pista.

-seleccion de cabeza: controla cual de las cabezas recibira la orden. Esta linea es opcional en muchos casos.

-linea de ready: indica que el floppy esta listo para enviar o recibir datos. En ciertos floppys (minifloppys) no existe esta linea.

-direccion de movimiento de cabeza: controla la direccion del movimiento de la cabeza, ya sea hacia dentro o hacia fuera del disco. Esta senal suele estar cualificada por otra.

-indice: los floppys tienen una senal que indica el comienzo de una pista. Esta sirve de referencia para que el

controlador sepa en que posicion esta.

-write protect: indica que el disco introducido tiene una marca "hardware" de proteccion a la escritura.

-seleccion de drive: actua de forma analoga al chip select. En este caso se selecciona el drive a usar.

-datos a leer: evidente. Actualmente, la mayoria de los controladores necesitan que los datos esten cualificados por una "ventana". Para eso, se disponen distintas senales para habilitar y controlar el PLL que decodifica los datos.

-datos a escribir: idem. En muchos casos, el controlador necesita logica externa para que los datos puedan ser escritos correctamente. Se emplea una logica de precompensacion con sus propias lineas y un clock.

-low current: baja la corriente de escritura en las pistas mas internas. En ciertos floppys esto no existe.

-testeo de fault condicion: examina la condicion de fallo del floppy. En ciertos casos se incluye un reseteo de este flip-flop.

Inicialmente hubieron dos opciones principales para la eleccion del controlador de disco:

1) 2797 de WESTERN DIGITAL

2) 8272 de INTEL, o bien uPD765 de NEC.

El problema que principalmente se queria evitar era el de la depuracion. El 2797 es para ello un integrado ideal debido a su PLL interno y a su logica de precompensacion interna que elimina practicamente todos los problemas encontrables y que por tanto era un firme candidato. Hubieron dos razones por

las que no se opto.

-dificultad de conseguirlo.

-solo posibilita el control de un drive (pero es expandible)

Dado que solo controla un dirve, es necesario tener una relacion del posicionamiento de cada drive y que el control de seleccion se hiciera por el micro. A pesar de esta desventaja, los problemas eliminados son bastante mas importantes. La "dificultad" de conseguirlo fue el factor mas real.

El empleo del 8272 (o uPD765) tiene ciertas ventajas entre las que se dispone:

-transferencias en dos modos (DMA - no DMA).

-capacidad de transferencia multisector y multipista.

-control de hasta cuatro floppys o minifloppys de dos caras.

-operaciones de busqueda en paralelo en los cuatro drives.

A su vez lo teniamos disponible inmediatamente. El principal problema era la logica de PLL para la decodificacion de los datos del disco, no por compleja, sino por dificil de depurar con los instrumentos que se poseen. Ejemplos de PLL pueden verse en <6> y <10>.

Se contaba con el uso del dispositivo WD1691 para la implementacion del PLL. Razones, la mayoria ajenas a mi voluntad, impidieron afrontar esta parte del proyecto. Patillaje y referencias del 8272 en <10>. El WD1691 y otros integrados de la WESTERN DIGITAL pueden encontrarse en <11>. Para una referencia de la unidad de discos OEM referirse a <12>. No se disponen de elementos para la calibracion del disco (discos de alineamiento, etc..).

BIBLIOGRAFIA CAPITULO 9

<1>: Intel corp.

ISIS-II user's guide.

<2>: ver referencia 6 capitulo 8.

<3>: ver referencia 3 capitulo 8.

<4>: Intel corp.

Intellec double density diskette operating system hardware reference manual.

<5>: Intel corp.

SBC 202 double density diskette controller hardware reference manual.

<6>: varios autores.

Interconexion de perifericos a microprocesadores.

Editorial Marcombo.

<7>: Motorola.

M6800 microprocessor applications manual.

Capitulo 5.

<8>: ver referencia <1> capitulo 7.

<9>: ver referencia <2> capitulo 7.

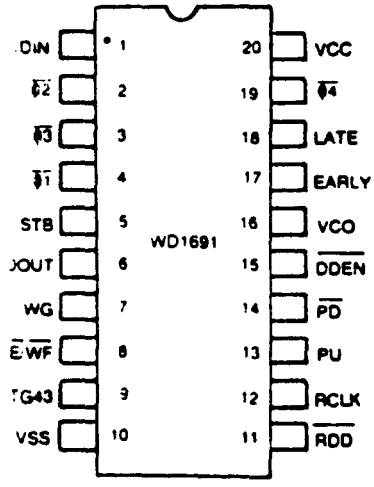


FIG. 8.1

CONCLUSIONES

CONCLUSIONES.

En el desarrollo de este proyecto, si algun elemento tuvo una importancia capital, este fue sin duda la depuracion. Los problemas con la implementacion fisica llevaron a un aumento del tiempo de desarrollo. El empleo de un emulador hubiera bajado varias veces este tiempo. Asi mismo, la necesidad de una implementacion mas fiable, es tambien una consideracion fundamental. Esta es una breve historia de la evolucion del proyecto.

1-Estudio de los requisitos necesarios. Desecho de la opcion MCS-85. En esta etapa no se habia pensado en las opciones de direccionamiento a 62 K. Se elige alternativa 8089. Se habia empezado el desarrollo del control de la unidad de discos en un proyecto complementario.

2-Esperando por la llegada de los componentes (mas de 3 meses!), se empieza a desarrollar software para el 8089. Ensamblador y depurador, y disemar las placas.

3-Implementacion del proyecto con muchos problemas en la fiabilidad de las conexiones y ruido en la linea BUSY/. Se emplea para los tests un 8088, ya que solo se poseia un 8089, y aprovechando la compatibilidad descrita previamente que fue buscada expresamente. El desarrollo del control de la unidad de discos queda paralizado.

4-La paralizacion del proyecto de control, conlleva un abandono del empleo del 8089. Se sigue con el 8088. El objetivo ahora se convierte en el desarrollo de un monitor de

depuracion software para sistemas iAPX, usando los recursos del host, el sistema de desarrollo MDS-221, tales como unidad de discos, mayor memoria, etc...

5-Se consigue compilador , ensamblador y utilidades para el desarrollo del software 8088. Increiblemente, la informacion de las utilidades no se tiene. La deduccion de su modo de funcionamiento se baso en algun ejemplo aislado que dio la sintaxis, y en el analisis del modulo objeto que dio los comandos.

6-Se consigue la comunicacion ISIS-II - 8088. Se desarrolla totalmente el monitor. Se hace algun analisis de prestaciones. Se sigue, sin embargo con problemas de fiabilidad en las conexiones.

El empleo de la familia iAPX conlleva muchas ventajas desde todos los puntos de vista, pero en cualquier caso, y como regla general que se aplica a todos estos sistemas, la necesidad de herramientas hardware y software como soporte a la depuracion y desarrollo de software, es absolutamente imprescindible, si se quiere conseguir una cierta "productividad". De otro modo, los problemas asociados, si bien no insolubles, se convierten facilmente en intratables, y este hecho se agiganta en este caso, en donde se tienen varios procesadores distintos coordinados en hardware y software, con los problemas derivados de ello en el interfase (codigo diferentes, velocidades diferentes, hardware diferentes, etc...).

Asi pues, para el desarrollo de un controlador de discos para nuestro sistema de desarrollo que fue el objetivo inicial, la alternativa empleando la familia MCS-85 se convierte asi en la mas atractiva teniendo en cuenta las suposiciones hechas en el capitulo 9, para el ambito de nuestra escuela en las condiciones actuales, y la unica razon real e importante para efectuar este cambio de alternativa radica en las facilidades para la depuracion que actualmente se posee.

BIBLIOGRAFIA

G. Conte y otros.

Multi-microprocessor systems for real-time aplications.

D. Reidel Publishing Company.

Y. Paker.

Multimicroprocessor systems.

Apic studies in data processing 1978.

V. Milutinovic

Advanced microprocessors and high-level language computer architectures for VLSI.

Onceavo simposio sobre microproceso y microprogramacion.

D. Siewiorek y otros.

Computer structures: Principles and examples.

Mac Graw Hill computer science series.

M. Valero.

Arquitectura de los computadores de alta velocidad.

VII Escuela de verano de informatica. AEIA.

M. Valero y otros.

Redes de interconexion para sistemas multiprocesadores.

Mundo Electronico.

J. Figueras.

Arquitecturas multimicrocomputador.

III Escuela de verano de informatica. AEIA.

En esta referencia se incluyen:

C. Weitzman.

Distributed micro-mini computer systems.

Editorial Prentice-Hall.

P. Borrill.

Microprocessor bus structures and standards.

IEEE Micro Febrero 1981.

A. Allison.

Status report on the P896 backplane bus.

IEEE micro Febrero 1981.

Intel Corp.

OEM systems handbook 1985.

Phil C. C. Yeh y otros.

Shared cache for multiple-stream computer systems.

IEEE Transactions on computers. Enero 1983.

M. Ajmone y otros.

Comparative analysis of single bus multiprocessor architectures.

IEEE Transactions on computers. Diciembre 82.

M. Ajmone y otros.

Modelling bus contention and memory interference in a multiprocessor.

IEEE Transactions on computers. Diciembre 82.

E. Rubio y otros.

Apuntes de sistemas operativos.

E. U. Informatica de Las Palmas. U. P. L. P.

B. Hansen.

Operative sistems principles.

Prentice-Hall.

Intel corp.

iAPX86/88, 186/188 User's manual. Programmer's reference.

Intel Corp.

OEM systems handbook 1983.

Intel Corp.

iAPX86/88 User's manual 1981.

Intel corp.

iAPX 86/88, 186/188 User's manual. Hardware reference.

Intel corp.

Microsystems components handbook. Volumen 1 y 2.

Las siguientes referencias pertenecen al manual de uso del MDS-221, editado por la Intel corp.

Intellec series II microcomputer development system hardware reference manual.

Intellec series II microcomputer development system schematic drawings.

Intellec series II microcomputer development system interface manual.

Intellec series II microcomputer development system monitor listings

ICE85 in-circuit emulator operating instructions for ISIS-II users.

PL/M-86 programming manual for 8080/8085 based development systems.

PL/M-86 User's guide.

ISIS-II user's guide.

Intel corp.
Intellec double density diskette operating system hardware reference manual.

Intel corp.
SBC 202 double density diskette controller hardware reference manual.

varios autores.
Interconexion de perifericos a microprocesadores.
Editorial Marcombo.

Motorola.
M6800 microprocessor applications manual.
Capitulo 5.

Western Digital.
Data Catalog.

Intel corp.
SDK-86 system design kit monitor listings.

Texas Instruments.
TTL Data catalog.

TERCERA PARTE

LISTADOS 88

```
ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE MONITOR
OBJECT MODULE PLACED IN :F4:VER4.OB J
COMPILER INVOKED BY:
:F1:PLM86 :F4:VER4.SRC PRINT(:F1:VER4.LST) LARGE OPTIMIZE(2) DEBUG


$TITLE('MONITOR SISTEMA MULTIPROCESADOR 8088')
$NOINTVEC TOR


/* MONITOR SISTEMA MULTIPROCESADOR 8088. V2.1
                  JUNIO 1986.

                  DESARROLLADO EN LA
                     ESCUELA UNIVERSITARIA POLITECNICA DE  LAS PALMAS.
                     RAMA: TELECOMUNICACION, ESPECIALIDAD: EQUIPOS ELECTRONICOS.
                      UNIVERSIDAD POLITECNICA DE LAS PALMAS.

                  AUTOR
                      JAVIER ALBERTO MORAN CARRERA

                  A EFECTOS DE
                      PROYECTO FIN DE CARRERA.

                  NOTA: DEBIDO A EFE CTOS DE ESPACIO QUE IMPOSIBILITA LA
                  COMPILACION, REMITIRSE AL PROYECTO PARA EXPLICACION DE
                   LA ESTRUCTURA Y CAPACIDADES DE ESTE MONITOR, ASI COMO
                  LOS REQUIS ITOS HARDWARE Y SOFTWARE QUE SE PRECISAN.
                  ESTE MONITOR ES UNA ADAPTACION DEL QUE POSEE EL MONIT OR
                  SERIE DEL KIT MICROCOMPUTADOR SDK-86. OTRA INFORMACION
                  PUED E EXTRAERSE DE SU LISIADO FUENTE.                    */
```

```
            $EJECT

  1         MONITOR:DO;


              /* MODULO DE DECLARACION DE VARIABLES GLOBALES ###########*/


  2   1     DECLARE INT$VECTOR(5) POINTER
                          AT (00000 H);        /* VECTORES DE INTERRUPCION */
  3   1     DECLARE MONITOR$STACKPTR WORD,
                  MONITOR$STACKBASE WORD;
  4   1     DECLARE (BRK1$FLAG,                 /* TRUE SE HAY BREAKPOI NT */
                  BRK1$SAVE,                    /* RESERVADA PARA INSTRC. BREAK */
                   CHAR,                         /* CARACTER SIGUIENTE BUFFER */
                  CHECK$SU M,                    /* CHECKSUM FORMATO HEXADECIMAL */
                  I,                            /* I NDICE */
                  WORD$MODE,                    /* FLAG ACTIVACION MODO 8086 */
                   LAST$COMMAND                  /* ULTIMO COMANDO */
                             ) BYTE;


  5   1     DECLARE END$OFF WORD;               /* DIRECCION FIN DE OFFSE T */
  6   1     DECLARE FILEIN(*) BYTE DATA(':CI: '); /* CONSOLA DE ENTRADA */
  7   1      DECLARE BUFFERIN  (80) BYTE;        /* BUFFER DE ENTRADA */
  8   1     DECLARE LONG$BUF$IN  BYTE;           /* PUNTERO BUFFER ENTRADA */
  9   1     DECLARE AFTN$IN WORD;               /* AFTN DE CONSOLA DE ENTRADA  */
 10   1     DECLARE AFTN$OUT LITERALLY '0H';    /* AFTN DE CONSOLA DE SAIDA */
 11   1      DECLARE STATUS WORD;                /* ESTADO DE ERROR DE ISIS-II */
 12   1     DECLARE LIMIT$IN WORD;               /* NUM CARACTS VALIDOS BUFFERIN */


 13   1     DECLARE TRUE           LITERALLY '0FFH',
                FALSE          LITERALLY '000H',
            BREAK$INST    LITERALLY '0CCH',
               STEP$TRAP     LITERALLY '0100H',
                USER$INIT$SP  LITERALLY '0100H',
               60$COMMAND    LITERALLY '2',
                SS$COMMAND    LITERALLY '3',
               STANDARD$LEN  LITERALLY '16',
                ASCR          LITERALLY '0DH',
               ASLF          LITERALLY '0 AH',
               ASBL          LITERALLY '20H';


 14    1      DECLARE SIO$BREAK$MSG(*) BYTE DATA ('BR ',0),
                SIO$SIGNON(*) BYTE DATA ('8088 PRO TOTIPO MULTIPROCESADOR',0),
                ASCII(*)      BYTE DATA ('0123456789ABCDEF'),
                 SIO$CMND(*)   BYTE DATA ('SXGNMDIORWCE');
```

```
$EJECT

15  1   DECLARE MEMORY$ARG1$PTR POINTER,          /* ARGUM ENTO 1 */
            ARG1 STRUCTURE (OFF WORD,SEG WORD) AT (@MEMORY$ARG1$PTR),
             MEMORY$ARG1 BASED MEMORY$ARG1$PTR BYTE,
            MEMORY$WORD$ARG1 BASED MEMORY$ARG1 $PTR WORD;

16  1   DECLARE MEMORY$ARG3$PTR POINTER,          /* ARGUMENTO  3 */
            ARG3 STRUCTURE (OFF WORD,SEG WORD) AT (@MEMORY$ARG3$PTR),
             MEMORY$ARG3 BASED MEMORY$ARG3$PTR BYTE,
            MEMORY$WORD$ARG3 BASED MEMORY$ARG3$PTR  WORD;

17  1   DECLARE MEMORY$BRK1$PTR POINTER,          /* BREAKPOINT 1  */
            BRK1 STRUCTURE (OFF WORD,SEG WORD) AT (@MEMORY$BRK1$PTR),
             MEMORY$BRK1 BASED MEMORY$BRK1$PTR BYTE;

18  1   DECLARE MEMORY$ CSIP$PTR POINTER,          /* CS:IP */
            CSIP STRUCTURE (OFF WORD,SEG WORD) AT (@MEMORY$CSIP$ PTR),
            MEMORY$CSIP BASED MEMORY$CSIP$PTR BYTE;

19  1    DECLARE MEMORY$USERSTACK$PTR POINTER,
            USERSTACK STRUCTURE (OFF WORD,SE G WORD) AT (@MEMORY$USERSTACK$PTR),
            MEMORY$USERSTACK BASED MEMORY$USERSTACK$PTR WORD;

20  1   DECLARE REG(*) BYTE DATA                 /* NOMBRE DE LOS REGISTROS */
                        ('AXBXCXDXSPBPSIDICSDSSSESIPFL'),
             REG$INDEX   WORD,                  /* INDICE DE REGISTROS */
            REG $SAV(14) WORD,                  /* ZONA DE GUARDA DE REGISTROS */
            REG$ORD(*) BYTE DATA (7,6,1,3 ,2,0,9,11,12,8,13),
            SP  LITERALLY 'REG$SAV(4)',
            BP      LITERALLY 'REG$SAV(5)',
            CS  LITERALLY 'REG$SAV(8)',
             DS      LITERALLY 'REG$SAV(9)',
            SS  LITERALLY 'REG$SAV(10)',
             ES      LITERALLY 'REG$SAV(11)',
            IP  LITERALLY 'REG$SAV(12)',
             FL      LITERALLY 'REG$SAV(13)';
```

```
                $EJECT

                  /* SECCION DE I/O################################### ###*/
                  /* SE INCLUYE EN ESTA SECION LOS PROCEDIMIENTOS EMPLEADOS
                      POR EL MONITOR PARA EL TRATAMIENTO DE SUS ENTRADAS Y
                     SALIDAS. SE INCLUYE UN GRUPO DE SU BRUTINAS EXTERNAS
                     EMPLEADAS PARA LA COMUNICACION CON EL ISIS-II         */


                  /* SUBSECCION DE COMUNICACIONES CON ISIS-II              */
                   /* LAS SUBRUTINAS
                     EXOPEN,EXWRITE,EXREAD,EXEXIT,EXCONSOLE
                      INCLUYEN EL ACCESO EXCLUSIVO A LAS SUBRUTINAS DEL
                     SISTEMA PARA PERMITIR E L EMPLEO DE OTRO PROCESADOR.
                     EN ELLAS SE INCLUYEN LAS COMUNICACIONES CON EL ISIS-II
                      Y EL CALCULO DE LAS DIRECCIONES REALES.
                     MAS COMENTARIOS EN EL LI STADO DEL PROGRAMA MDSLIB.LST.*/

 21   1      EXOPEN:PROCEDURE(AFTN$PTR,FILE$PTR,ACCESS,MODE,S TS$PTR) EXTERNAL;
 22   2             DECLARE (AFTN$PTR,FILE$PTR,STS$PTR) POINTER,
                            (ACCESS,MODE) WORD;
 23   2             END;

 24   1      EX WRITE:PROCEDURE(AFTN,BUFFER$PTR,COUNT,STS$PTR) EXTERNAL;
 25   2              DECLARE (BUFFER$PTR,STS$PTR) POIN TER,
                            (AFTN,COUNT) WORD;
 26   2              END;

 27   1      EXREAD:PROCEDURE(AFTN,BUFFER$PTR,COUNT,ACTUAL$PTR,STS$PTR) EXTERNAL;
 28   2             DECLARE (BUFFER$PTR,ACTUAL$PTR,STS$PTR) POINTER,
                           (AFTN,COUNT) WORD;
 29   2             END;

 30   1      EXCONSOL:PROCEDURE(INFILE,OUTFILE,STS$PTR) E XTERNAL;
 31   2              DECLARE (INFILE,OUTFILE,STS$PTR) POINTER;
 32   2               END;

 33   1      EXEXIT:PROCEDURE EXTERNAL;
 34   2             END;

                  /*SUBRUTINAS INTERNAS DE COMUNICACION CON I/O           */


 35   1      SIO$OUT$CHAR:
                  /*ESTA SUBRUTINA SE EMPLEA EN LA SALIDA D E DATOS HACIA LA
                    CONSOLA DEL SISTEMA QUE ESTE ACTUALMENTE ASIGNADA */
                           PROCEDURE (C);
 36   2                  DECLARE C BYTE;
 37   2                        CALL EXWRITE(AFTN$OUT,@C,1,@STATUS);
 38   2                  END;


 39   1      SIO$GET$CHAR:
                  /*ESTA RUTINA EXTRAE UN CARACTER PROVENIENTE D EL BUFFER
                    QUE ENTREGA LA CONSOLA DE ENTRADA DEL SISTEMA */
                           PROCEDURE;
 40   2                  IF LIMIT$IN>LONG$BUF$IN THEN DO;
```

```
 42  3                    CHAR=BUFFERIN(LONG$BUF$IN);
 43  3                    LONG$BUF$IN=LONG$BUF$IN+1;
 44  3                END;
 45  2                ELSE CHA R=0;
 46  2                END;
```

```
            $EJECT

 47   1     SIO$OUT$BYTE:
              /*ESTA SUBRUT INA RECOGE COMO PARAMETRO UN BYTE Y LO EMITE
                COMO DOS CARACTERES */
                       PROCEDURE(B);
 48   2                DECLARE B BYTE;
 49   2                 CALL SIO$OUT$CHAR(ASCII(SHR(B,4) AND 0FH));
 50   2                CALL SIO$OUT$CHAR(ASCII(B AND 0FH)) ;
 51   2                CHECK$SUM=CHECK$SUM-B;
 52   2                END;


 53   1     SIO$OUT$WORD:
              /*ESTA SUBRUTINA RECOGE COMO  PARAMETRO UNA PALABRA DE 16
                BITS Y LO ENVIA COMO 2 BYTES A SIO$OUT$BYTE QUE SE
                 ENCARGARA DE PASARLO COMO CARACTERES */
                       PROCEDURE(W);
 54   2                 DECLARE W WORD;
 55   2                CALL SIO$OUT$BYTE(HIGH(W));
 56   2                 CALL SIO$OUT$BYTE(LOW(W));
 57   2                END;


 58   1     SIO$OUT$BLANK:
              /*ESTA SUBRUTINA ENVIA UN BLANCO A  LA SUBRUTINA DE SALIDA*/
                        PROCEDURE;
 59   2                 CAL L SIO$OUT$CHAR(ASBL);
 60   2                 END;


 61    1    SIO$OUT$STRING:
              /*ESTA SUBRUTINA ENVIA UNA STRING DE CARACTERES POR LA
                 CONSOLA ASIGNADA ACTUALEMENTE. LA DIRECCION DE LA STRING
                SE PASA COMO PUNTE RO. EL FIN DE STRING SE DETECTA POR UN
                CERO */
                         P ROCEDURE(PTR);
 62   2                  DECLARE PTR POINTER,STR BASED PTR(1) BYTE;
 63   2                    I=0;
 64   2                   DO WHILE STR(I)<>0;
 65   3                       CALL SIO$OUT$CHAR(STR(I));
 66   3                      I=I+1;
 67   3                    END;
 68   2                  END;


 69   1     SIO$OUT$HEADER:
              /*ESTA SUBRUTINA ENVIA A LA CONSOLA DE SALIDA, UN
                 ENCABEZAMIENTO DE FORMATO  HEXADECIMAL CONSISTENTE EN
                ':', SEGUIDO POR LA LONGITUD  DEL RECORD, LA DIRECCION DE
                CARTA Y EL TIPO DE RECORD. SE INICIALIZA EL CHECKSUM A CERO*/
                          PROCEDURE (LENGTH,LOAD$ADDR,REC$TYPE);
 70   2                   D ECLARE (LENGTH,REC$TYPE) BYTE, LOAD$ADDR WORD;
 71   2                   CALL SIO$OUT$CHAR(':');
 72   2                    CHECK$SUM=0;
 73   2                   CALL SIO$OUT$BYTE (LENGTH);
 74   2                    CALL SIO$OUT$WORD (LOAD$ADDR);
```

```
 7 5   2                    CALL SIO$OUT$BYTE (REC$TYPE);
 7 6   2                   END;
```

```
            $EJECT

 77   1     SIO$GET $STRING:
              /*ESTA SUBRUTINA EXTRAE DE LA CONSOLA DEL SISTEMA, UNA
                 LINEA DE FORMATO EDITADO. LAS CAPACIDADES DE EDICION NO
                PERTENECEN AL 8088, SINO LO EJECU TA EL MDS221.
                SIO$GET$CHAR RECOGE LOS CARACTERES DEL BUFFER QUE ESTA
                 SUBRUTINA HA RECOGIDO DEL MDS. */
                        PROCEDURE;
 78   2                   DECLARE ACTUAL WORD;
 79   2                  CALL EXREAD(AFTN$IN,@BUFFERIN(0),80,@ACTUAL, @STATUS);
 80   2                  LONG$BUF$IN=0;
 81   2                  LIMIT$IN=ACTU AL;
 82   2                  END;
```

```
            $EJECT

              /*UTILIDADES ######################################### */
              /* LAS SIGUIENTES SUBRUTINAS EFECTUAN TRABAJOS SUCIOS DE
                  USO COMUN */


 83   1     SIO$VALID$HEX:
              / *ESTA SUBRUTINA TESTEA EL PARAMETRO DE ENTRADA
                DEVOLVIENDO TRUE O FALSE EN CASO DE QUE EL DAT O SEA UN
                VALOR ASCII HEXADECIMAL CORRECTO */
                        PROC EDURE (H) BYTE;
 84   2                 DECLARE H BYTE;
 85   2                 DO I=0 T O LAST(ASCII);
 86   3                 IF H=ASCII(I) THEN RETURN TRUE;
 88   3                  END;
 89   2                 RETURN FALSE;
 90   2                 END;


 91   1     SIO$HEX:
              /*ESTA SUBRUTINA CONVIERTE UNA  ENTRADA ASCII A SU
                EQUIVALENTE BINARIO */
                   PROCEDURE (C)  WORD;
 92   2                 DECLARE C BYTE;
 93   2                 IF C<='9' THEN RETURN DOUBLE( C-030H);
 95   2                 ELSE RETURN DOUBLE(C-037H);
 96   2                 END;


 97   1     SIO$VALID$REG$FIRST:
              /*ESTA SUBRUTINA CHE QUEA SI EL CARACTER DE ENTRADA ES LA
                PRIMERA LETRA DE UN REGISTRO VALIDO Y RETORNA TRUE EN
                 ESTE CASO */
                    PROCEDURE BYTE;
 98   2                  DO I=0 TO 26 BY 2;
 99   3                 IF CHAR=REG(I) THEN RETURN TRUE;
101    3                 END;
102   2                 RETURN FALSE:
103   2                 END;


104   1     SIO$VALID$REG:
              /*ESTA SUBRUTINA CHEQU EA SI LOS DOS PARAMETROS DE ENTRADA
                PUESTOS JUNTOS FORMAN UN NOMBRE DE UN REGISTRO VALIDO.
                 BUSCA ENTONCES EN LA TABLA DE REGISTOS Y SI SE ENCUENTRA,
                PO NE LA VARIABLE GLOBAL 'REG$INDEX' CON EL INDICE DEL
                REGISTRO. EN CASO DE NO ENCONTRARLO, DEVU ELVE FALSE  */
                         PROCEDURE (C1,C2) BYTE;
105   2                   DECLARE (C1,C2) BYTE;
106   2                  DO REG$INDEX=0 TO 13;
107   3                   IF C1=REG(REG$INDEX*2) AND C2=REG(REG$INDEX*2+1) THEN
108   3                                                    RETURN TRUE;
109   3                  END;
110   2                  RETURN  FALSE;
111   2                  END;
```

```
                *EJECT

112   1       SIO$CRLF:
                  /*ESTA SUBRUT INA ENVIA UN 'CARRIAGE RETURN & LINE FEED' A
                    LA CONSOLA DE SALIDA */
                        PROCEDURE;
113   2                 CALL SIO$OUT$CHAR(ASCR);
114   2                  CALL SIO$OUT$CHAR(ASLF);
115   2                 END;


116    1      SIO$TEST$WORD$MODE:
                  /*ESTA SUBRUTINA TESTEA LA CONTINUACION DE UNA 'W' DESPUES
                     DE UN COMANDO PARA INDICAR MODO DE 16 BITS. EN TAL CASO
                    RETORNA T RUE. */
                              PROCEDURE:
117   2                           WORD$MODE =FALSE;
118   2                           CALL SIO$GET$CHAR;
119   2                            IF CHAR='W' THEN DO;
121   3                           WORD$MODE=TRUE;
122   3                            CALL SIO$GET$CHAR;
123   3                           END;
124   2                            IF CHAR=ASBL THEN CALL SIO$GET$CHAR;
126   2                         END;


127   1       SIO$SCAN$BLANK:
                  /*ESTA SUBRUTINA SE LLAMA DESPUES DE UN CO MANDO PARA
                    ELIMINAR EL CARACTER BLANCO OPCIONAL */
                             PROCEDURE;
128   2                    CALL SIO$GET$CHAR;
129   2                    IF CH AR=ASBL THEN CALL SIO$GET$CHAR;
131   2                    END;
```

```
                $EJECT

                   /*EVALUACION DE ARGUMENTOS # ##############################*/


132   1       SIO$GET$WORD:
                  /*ESTA SUBRUTINA LEE CARACTERES DE LA CONSOLA Y EVALUA
                   UNA EXPRESION CONSISTEN TE DE '+-' Y OPERANDOS DE NUMEROS
                   HEXADECIMALES Y NOMBRES DE REGISTROS */
                          PROCEDURE WORD;
133   2                  DECLARE (SAVE,W) WORD, (OPER,T) BYTE;
 134   2                  OPER='+';
135   2                  W=0;
136   2                   DO WHILE TRUE;
137   3                  T=CHAR;
138   3                  SAVE=0;
 139   3                  IF SIO$VALID$REG$FIRST THEN DO;
141   4                   CALL SIO$GE T$CHAR;
142   4                   IF SIO$VALID$REG(T,CHAR) THEN DO;
144   5                    SAVE=REG$SAV(REG$INDEX);
145   5                   CALL SIO$GET$CHAR;
146   5                    GOTO EVAL;
147   5                   END;
                          ELSE
 148   4                    SAVE=SIO$HEX(T);
149   4                   END;
150   3                    IF NOT(SIO$VALID$HEX(T)) THEN GOTO ERROR;
152   3                   DO WHILE SIO$VALID$HEX(CHAR);
153   4                   SAVE=SHL(SAVE,4)+SIO$HEX(CHAR);
154   4                   CALL SI O$GET$CHAR;
155   4                   END;
156   3            EVAL:  IF OPER='+' THEN W=W+SA VE;
158   3                   ELSE W=W-SAVE;
159   3                   IF CHAR=ASCR OR CHAR= ':' OR CHAR=',' THEN RETURN W;
161   3                   IF CHAR='+' OR CHAR='-' THEN OPER=CHAR;
 163   3                   ELSE GOTO ERROR;
164   3                   CALL SIO$GET$CHAR;
 165   3                   END;
166   2                  END;
```

```
                $EJECT


167   1     SIO$GET$ADDR:
                /*ESTA SUBRUTINA ACEPTA UNA EXPRESION DE DIRECCION VALIDA
                 COMPUESTA POR UN SEGMENTO OPCIONAL <SEG>: Y UN
                 DESPLAZAMIENTO. CADA UNO DE ESTOS ELEMENTOS PUEDE  SER
                 UNA EXPRESION DE LAS COMPUTADAS POR SIO$GET$WORD */
                        PROCEDURE (PTR,DEFAULT$BASE);
168   2                 DECLARE PTR POINTER, DEFAULT$BASE WORD,
                            ARG BASED PTR STRUCTURE (OFF WORD, SEG WORD);
169   2                     ARG.SEG=DEFAULT$BASE;
170   2                    ARG.OFF=SIO$GET$WORD;
171   2                     IF CHAR=':' THEN DO;
173   3                    CALL SIO$GET$CHAR;
174   3                     ARG.SEG=ARG.OFF;
175   3                    ARG.OFF=SIO$GET$WORD;
176   3                     IF CHAR=':' THEN GOTO ERROR;
178   3                     END;
179   2                   END;


180   1     SIO$UPDATE$IP:
                /*ESTA SUBRUTINA ES LLAMADA POR LOS COMANDOS SINGLE STEP
                 Y GO PARA EXHIBIR POR LA CONSOLA D E SALIDA EL VALOR
                 ACTUAL DE IP Y EL DATO APUNTADO POR ESTE, Y ABRE EL IP
                  PARA LECTURA */
                        PROCEDURE;
181   2                  CALL SIO$OUT$BLANK;
182   2                 CALL SIO$OUT$WORD(IP);
183   2                  CSIP.SEG=CS;
184   2                 CSIP.OFF=IP;
185   2                 CALL SIO$ OUT$CHAR('-');
186   2                 CALL SIO$OUT$BLANK;
187   2                 CALL  SIO$OUT$BYTE(MEMORY$CSIP);
188   2                 CALL SIO$OUT$BLANK;
189   2                  CALL SIO$GET$STRING;
190   2                 CALL SIO$GET$CHAR;
191   2                  IF CHAR<>',' AND CHAR<>ASCR THEN
192   2                                    CALL SIO$GET$ADDR(@CSIP,CS) ;
193   2                 END;
```

```
                $EJECT

                  /* SECCION TRATAMIENTO FORMATO HEXADECIMAL ###############*/


                  /*LAS SIGUIENTES SUBRUTINAS SE EMPLEAN PARA EL TRATAMIENTO
                     DE DATOS EN FORMATO HEXADECIMAL A LA CONSOLA DE SALIDA
                     O EN TRADA ACTUALMENTE ASIGNADA */


194   1       SIO$READ$CHAR:
                  /*ESTA SUBRUTINA EFECTUA LA MISMA LABOR QUE SIO$GET$CHAR,
                   PERO EL RESULTADO LO  DEVUELVE ESTA A LA VEZ QUE
                   ACTUALIZA LA VARIABLE 'CHAR' */
                            PROCEDURE BYTE;
195   2                    CALL SIO$GET$CHAR;
196   2                     RETURN CHAR;
197   2                    END;


198   1        SIO$READ$BYTE:
                  /*ESTA SUBRUTINA LEE UN BYTE Y AJUSTA EL CHECKSUM*/
                            PROCEDURE BYTE;
199   2                    DECLARE T BYTE;
200   2                     T=LOW(SIO$HEX(SIO$READ$CHAR));
201   2                    T=SHL(T,4)+LOW(SIO$HEX(SIO$READ$CHAR));
 202   2                     CHECKSUM=CHECKSUM+T;
203   2                    RETURN T;
 204   2                     END;


205   1       SIO$READ$WORD:
                  /*ESTA SUBRUTINA LEE UNA PALABRA DE 16 BITS */
                            PROCEDURE WORD;
206   2                    DECLARE T BYTE;
207   2                    T=SIO$READ$BYTE;
 208   2                     RETURN SHL(DOUBLE(T),8)+DOUBLE(SIO$READ$BYTE);
209   2                     END;


                  /*SECCION DE TRATAMIENTO DE INTERRUPCIONES ############ ###*/


                  /* LA SIGUIENTE SECCION CONTIENE LAS SUBRUTINAS RESPONSA BLES
                     DEL TRATAMIENTO DESPUES DE UNA INTERRUPCION Y DEL SALVADO
                      Y RECUPERACION DE REGISTROS */


210   1       SAVE$REGISTERS:
                  /*ESTA SUBRUTINA PROCEDE A SALVAR LOS REGISTROS DEL
                   USUARIO EN EL AREA  DEL MONITOR RESERVADA PARA ELLA. SE
                   EFECTUA DESPUES DE UNA INTERRUPCION */
                             PROCEDURE;
211   2                    BP=MEMORY$USERSTACK;
212    2                    USERSTACK.OFF=USERSTACK.OFF+4;
213   2                    DO I=0 TO 10;
 214   3                     REG$SAV(REG$ORD(I))=MEMORY$USERSTACK;
```

```
                215   3                     USERSTACK.OFF=USERSTACK.OFF+2;
                 216   3                      END;
 217   2                        SS=USERSTACK.SEG;
 218   2                         SP=USERSTACK.OFF;
 219   2                        END;
```

```
          $EJECT

220  1    RESTORE$EXECUTE:
            /*ESTA SUBRUTINA CARGA EL ESTADO DE LA MAQUINA DEL USUARIO
             Y  DEVUELVE EL CONTROL AL PROGRAMA DEL USUARIO. CONTIENE
             UNA SUBRUTINA EN CODIGO MAQUINA PARA RE ALIZAR LA
             RESTAURACION DE LOS REGISTROS DEL MONITOR A LA PILA Y
              LUEGO EJECUTA UN 'IRET' */
                        PROCEDURE;
221  2                   DECLARE RESTORE$EXECUTE$CODE(*) BYTE DATA
                        (08BH,0ECH,             /* MOV B P,SP */
                         088H,046H,002H,        /* MOV AX,/BP/.PARM2 */
                          08BH,05EH,004H,        /* MOV BX,/BP/.PARM1 */
                         08EH,0D0H,             /* MOV SS,AX */
                         08BH,0E3H,             /* MOV SP,BX */
                          05DH,                  /* POP BP */
                         05FH,                  /* POP DI */
                         05EH,                  /* POP SI */
                          05BH,                  /* POP BX */
                         05AH,                  /* POP DX */
                         059H,                  /* POP CX */
                          058H,                  /* POP AX */
                         01FH,                  /* PO P DX */
                         007H,                  /* POP ES */
                          0CFH                   /* IRET   */);
222  2          DECLARE  RESTORE$EXECUTE$CODE$PTR WORD DAT A (.RESTORE$EXECUTE$CODE);
223  2                   USERSTACK.SEG=SS;
224  2                    USERSTACK.OFF=SP;
225  2                   DO I=0 TO 10;
226  3                     USERSTACK.OFF=USERSTACK.OFF-2;
227  3                    MEMORY$USERSTACK=REG$SAV(REG$ORD(10-I)) ;
228  3                   END;
229  2                   USERSTACK.OFF=USERSTACK.OFF -2;
230  2                   MEMORY$USERSTACK=BP;
231  2                   CALL REST ORE$EXECUTE$CODE$PTR(USERSTACK.OFF,USERSTACK.SEG);
232  2                   END;


233  1    INTERRUPT1$ENTRY:
           /*ESTA SUBRUTINA ES LLAMADA CUANDO LA  CPU ES INTERRUMPIDA
             DEBIDO A LA EJECUCION DE UNA INSTRUCCION CON EL TRAP
              BIT ACTIVO (SINGLE STEP) */
                        PROCEDURE INTERRUPT 1:
 234  2                   USERSTACK.OFF=STACKPTR;
235  2                   USERSTACK.SEG=STACKBASE;
236  2                   STACKPTR=MONITOR$STACKPTR;
237  2                   STACKBA SE=MONITOR$STACKBASE;
238  2                   CALL SAVE$REGISTERS;
239  2                    FL=FL AND (NOT STEP$TRAP);
240  2                   IF LAST$COMMAND<>SS$COMMAND THEN CALL RESTORE $EXECUTE;
242  2                   CALL SIO$CRLF;
243  2                   CALL SIO$ UPDATE$IP;
244  2                   IF CHAR=',' THEN DO;
246  3                   IP =CSIP.OFF;
247  3                   CS=CSIP.SEG;
248  3                   FL=FL OR S TEP$TRAP;
249  3                   CALL RESTORE$EXECUTE;
```

```
 250   3                    END;
251    2                   IF CHAR<>ASCR THEN GOTO ERROR;
 253   2                    GOTO NEXT$COMMAND;
254    2                   END;
```

```
                $EJECT

255   1         INTERRUPT3$ENTRY:
                   /*ESTA SUBRUTINA ES LLAMADA CUANDO LA CPU EJECUTA UNA
                     INTER RUPCION TIPO 3 PRODUCIDA POR UNA INSTRUCCION. EL
                     MONITOR INSERTA ESTA PARA PRODUCIR EL BREAKP OINT.
                     TAMBIEN UNA INTERRUPCION EXTERNA O UNA INTERRUPCION DEL
                      USUARIO PODRIAN CAUSAR QUE ESTA SUBRUTINA FUESE LLAMADA */
                            PROCEDURE INTERRUP T 3;
256   2                     USERSTACK.OFF=STACKPTR;
257   2                     USERS TACK.SEG=STACKBASE;
258   2                     STACKPTR=MONITOR$STACKPTR;
259   2                      STACKBASE=MONITOR$STACKBASE;
260   2                     CALL SAVE$REGISTERS;
2 61   2                     CALL SIO$CRLF;
262   2                     GOTO AFTER$INTERRUPT;
 263   2                     END;


264   1         INIT$INT$VECTOR:
                  /*ESTA SUBRUTINA INICIALIZA EL VECTOR DE INTERRUPCION
                    COMO SIGUE: EL D ESPLAZAMIENTO DE LA DIRECCION DE
                    'INT$ROUTINE' ES CORREGIDA POR EL NUMERO DE BYTES
                     APROPIADOS DEL PROLOGO DE INTERRUPCION GENERADO POR EL
                    PL/M. OBSERVES E QUE ESTE PROLOGO ES DEPENDIENTE DE LA
                    OPCION DE COMPILADOR, Y EN ESTE CASO, LA OPCION DEBE
                    SER 'LARGE'. EL SEGMENTO DEL REGISTRO ACTUAL CS ES
                    DETERMI NADO POR UNA RUTINA CODIFICADA EN CODIGO
                    MAQUINA */
                     PROCEDURE(INT$VECTOR$PTR,INT$ROUTINE$OFFSET);
265   2                DECLARE INT$VECTOR$PTR POINTER, INT$ROUTINE$OFFSET WORD,
                       VECTOR BASED INT$VECTOR$PTR STRUCTURE(OFF WORD,SEG WORD),
                       CORRECTION LITERALLY '19H',
                       INIT$IN T$VECTOR$CODE(*) BYTE DATA
                       (055H,                    /* PUSH BP           */
                        089H,0ECH,               /* MOV BP,SP         */
                        08CH,0C8H,               /* MOV AX,CS         */
                       0C4H,05EH,004H,           /* LES BX,/BP/.PARM1 */
                       026H,089H,007H,           /* MOV ES:W/BX/,AX   */
                        05DH,                    /* POP BP            */
                        0C2H,004H,000H           /* RET 4             */),
                       INIT$INT$VECTOR$CODE$ PTR WORD DATA (.INIT$INT$VECTOR$CODE);
266   2                CALL INIT$INT$VECTOR$CODE$PTR(@VECTOR.SEG);
267   2                VECTOR.OFF=INT$ROUTINE$OFFSET-CORRECTION;
268   2                 END;
```

```
                $EJECT

                    /*MODULO DE COMANDOS##################################*/


                    /* EN ESTE MODULO RADICAN LAS SUBRUTINAS RESPONSABLES DE
                      LAS FU NCIONES DEL MONITOR PROPIAMENTE DICHAS */


269   1       SIO$GO:
                  /*SE IMPLEMENTA EL COMANDO 'GO'. EL USUARIO PODRIA
                   ESPECIFICAR UN NUEVO IP:P C Y UN BREAKPOINT OPCIONAL */
                     PROCEDURE;
270   2                       CALL  SIO$UPDATE$IP;
271   2                       IF CHAR=',' THEN DO;
273   3                       CA LL SIO$GET$CHAR;
274   3                       CALL SIO$GET$ADDR(@BRK1,CSIP.SEG);
275   3                        IF CHAR<>ASCR THEN GOTO ERROR;
277   3                       BRK1$SAVE=MEMORY$BRK1;
 278   3                       MEMORY$BRK1=BREAK$INST;
279   3                       IF MEMORY$BRK1<>BREAK$INST TH EN GOTO ERROR;
281   3                       BRK1$FLAG=TRUE;
282   3                       END;
 283   2                       ELSE IF CHAR<>ASCR THEN GOTO ERROR;
                              CALL  SIO$CRLF;
286   2                       IP=CSIP.OFF;
287   2                       CS=CSIP.SEG;
 288   2                       FL=FL AND (NOT STEP$TRAP);
289   2                       CALL RESTORE $EXECUTE;
290   2                       END;


291   1       SIO$ SINGLE$STEP:
                  /*SE IMPLEMENTA EL COMANDO 'SINGLE STEP'. SE MUESTRA EL IP
                    Y EL DATO APUNTADO. SE ABRE EL CS:IP PARA ENTRADA. UNA
                   COMA SIGNIFICA UN NUEVO 'S INGLE STEP'*/
                             PROCEDURE;
292   2                       CALL SIO$UP DATE$IP;
293   2                       IF CHAR<>',' THEN GOTO ERROR;
295   2                        IP=CSIP.OFF;
296   2                       CS=CSIP.SEG;
297   2                       FL=FL O R STEP$TRAP;
298   2                       CALL RESTORE$EXECUTE;
299   2                       E ND;
```

```
                $EJECT

 300   1      SIO$EXAM$MEM:
                /*SE IMPLEMENTA EL COMANDO DE ACCESO A MEMORIA */
                        PROCEDURE;
 301   2                DECLARE W WORD;
 302   2                 CALL SIO$TEST$WORD$MODE;
 303   2                CALL SIO$GET$ADDR(@ARG1,CS);
 304   2                 IF CHAR<>',' THEN GOTO ERROR;
 306   2                DO WHILE TRUE;
 307   3                 CALL SIO$OUT$BLANK;
 308   3                IF WORD$MODE THEN CALL SIO$ OUT$WORD(MEMORY$WORD$ARG1);
 310   3                ELSE CALL SIO$OUT$BYTE(MEMORY$ARG1);
 31 1  3                 CALL SIO$OUT$CHAR('-');
 312   3                CALL SIO$OUT$BLANK;
 313   3                 CALL SIO$GET$STRING;
 314   3                CALL SIO$GET$CHAR;
 315   3                 IF CHAR=ASCR THEN RETURN;
 317   3                IF CHAR<>',' THEN DO;
 319   4                    W=SIO$GET$WORD;
 320   4                   IF (CHAR<>',') AND (CHAR<>A SCR) THEN GOTO ERROR;
 322   4                   IF WORDMODE THEN DO;
 324   5                      MEMORY$WORD$ARG1=W;
 325   5                     IF MEMORY$WORD$ARG1<>W THEN GOTO ERROR;
 327   5                      END;
 328   4                   ELSE DO;
 329   5                      MEMORY$ARG1=LOW(W);
 330   5                     IF MEMORY$ARG1<>LOW(W) THEN GOTO ERROR;
 332   5                    END;
 333   4                   END;
 334   3                 IF CHAR=ASCR THEN RETURN;
 336   3                IF WORD$MODE THEN ARG1.OFF=ARG1.OFF+2;
 338   3                 ELSE ARG1.OFF=ARG1.OFF+1;
 339   3                CALL SIO$CRLF;
 340   3                 CALL SIO$OUT$WORD(ARG1.OFF);
 341   3                END;
 342   2                 END;
```

```
                $EJECT

343   1     SIO$EXAM$REG:
               /* SE IMPLEMENTA EL COMAND O DE EXAMEN DE REGISTROS. SE
                BUS UN NOMBRE DE REGISTRO VALIDO Y SE EXHIBE EL VALOR
                 DE ESTE REGISTRO, Y SE ESPERA UNA ENTRADA. UNA COMA DA
                PASO A UN NUE VO REGISTRO EXCEPTO PARA EL CASO DE 'FL'
                QUE DA FIN AL COMANDO */
                       PROCEDURE;
344   2                DECLARE (T,I) BYTE, SAVE WORD;
345   2                 CALL SIO$SCAN$BLANK;
346   2                IF CHAR=ASCR THEN DO;
348   3                 CALL SIO$CRLF;
349   3                DO I=0 TO 13;
350   4                  CALL SIO$OUT$BLANK;
351   4                 CALL SIO$OUT$CHAR(REG(I*2));
352   4                  CALL SIO$OUT$CHAR(REG(I*2+1));
353   4                 CALL SIO$OUT$CHAR('=');
 354  4                  CALL SIO$OUT$WORD(REG$SAV(I));
355   4                 IF I=6 THEN CALL SIO$CRLF ;
357   4                END;
358   3                RETURN;
359   3                 END;
360   2                IF NOT(SIO$VALID$REG$FIRST) THEN GOTO ERROR;
36 2   2                T=CHAR;
363   2                CALL SIO$GET$CHAR;
364   2                  IF NOT(SIO$VALID$REG(T,CHAR)) THEN GOTO ERROR;
366   2                 I=REG$INDEX;
 367  2                 DO WHILE TRUE;
368   3                CALL SIO$OUT$CHAR('=');
 369  3                 CALL SIO$OUT$WORD(REG$SAV(I));
370   3                CALL SIO$OUT$CHAR('-');
 371  3                 CALL SIO$OUT$BLANK;
372   3                CALL SIO$GET$STRING;
 373  3                 CALL SIO$GET$CHAR;
374   3                IF CHAR<>',' AND CHAR<>ASCR  THEN DO;
376   4                 SAVE=SIO$GET$WORD;
377   4                 IF (CHAR<>'< ') AND (CHAR<>ASCR) THEN GOTO ERROR;
379   4                 REG$SAV(I)=SAVE;
380   4                  END;
381   3                IF CHAR=ASCR OR I=13 THEN RETURN;
383   3                 I=I+1;
384   3                CALL SIO$CRLF;
385   3                CALL SIO$ OUT$CHAR(REG(I*2));
386   3                CALL SIO$OUT$CHAR(REG(I*2+1));
387   3                 END;
388   2                END;
```

```
                $EJECT

389   1     SIO$MOVE:
              /* SE EF ECTUA UN COMANDO DE MOVIMIENTO MEMORIA-MEMORIA.
                 SE ACEPTAN 3 ARGUMENTOS QUE INDICAN INICIO
                  DIRECCION-FUENTE, FINAL DIRECCION-FUENTE, INICIO
                 DIRECCIO N-DESTINO. UN ERROR EN LOS DATOS O UNA NO
                 CONFIRMACION DEL MOVIMIENTO CAUSA ERROR */
                    PROCEDURE;
390   2               CALL SIO$SCAN$BLANK;
391   2                CALL SIO$GET$ADDR(@ARG1,CS);
392   2               IF CHAR<>',' THEN GOTO ERROR;
 394  2                CALL SIO$GET$CHAR;
395   2               END$OFF=SIO$GET$WORD;
396   2                IF END$OFF<ARG1.OFF THEN GOTO ERROR;
398   2               IF CHAR<>',' THEN GOTO ERROR;
 400  2                CALL SIO$GET$CHAR;
401   2               CALL SIO$GET$ADDR(@ARG3,CS);
 402  2                IF CHAR<>ASCR THEN GOTO ERROR;
404   2               CALL SIO$CRLF;
 405  2     LOOP:      MEMORY$ARG3=MEMORY$ARG1;
406   2               IF MEMORY$ARG1<>MEMORY$ARG3 THEN GOTO ER ROR;
408   2               IF ARG1.OFF=END$OFF THEN RETURN;
410   2               ARG1.OFF=ARG 1.OFF+1;
411   2               ARG3.OFF=ARG3.OFF+1;
412   2               GOTO LOOP;
 413  2                END;


414   1     SIO$DISPLAY:
              /* SE IMPLEMENTA EL COMANDO DE EXHIBICION DE MEMORIA. SI
                 ES LLAMADO CON UN  UNICO PARAMETRO DE ENTRADA, SE
                 EXHIBE UN UNICO BYTE. SI SE LLAMA CON DOS, SE EXHIBE
                  EL RANGO ENTRE ELLOS */
                       PROCEDURE;
415    2                DECLARE T BYTE;
416   2                CALL SIO$TEST$WORD$MODE;
417   2                 CALL SIO$GET$ADDR(@ARG1,CS);
418   2                IF CHAR=ASCR THEN END$OFF=ARG1.OFF;
 420  2                 ELSE DO;
421   3                 IF CHAR<>',' THEN GOTO ERROR;
 423  3                  CALL SIO$GET$CHAR;
424   3                END$OFF=SIO$GET$WORD;
 425  3                  IF END$OFF<ARG1.OFF  THEN GOTO ERROR;
427   3                IF CHAR<>ASCR  THEN GOTO ERROR;
429   3                END;
430   2          NEWLINE: CALL SIO$CRLF;
 431  2                   CALL SIO$OUT$WORD(ARG1.OFF);
432   2            LOOP:  CALL SIO$OUT$BLANK ;
433   2                   IF WORD$MODE THEN DO;
435   3                     CALL SIO$OUT$WOR D(MEMORY$WORD$ARG1);
436   3                     IF ARG1.OFF=END$OFF THEN RETURN;
438   3                      ARG1.OFF=ARG1.OFF+1;
439   3                     END;
440   2                   ELSE CALL SIO$OUT$BYTE(MEMORY$ARG1);
441   2                  IF ARG1.OFF>=END$OFF THEN RETURN;
 443  2                   ARG1.OFF=ARG1.OFF+1;
```

```
444   2                  T=ARG1.0FF AND 00FH;
445   2                 IF T=0 OR (WORD$MODE AND T=1) THEN GOTO NEWLINE;
 447   2                 GOTO LOOP;
448   2                 END;
```

```
            $EJECT


449   1     SIO$INPUT:
                /* SE IMPLEMENTA EL COMANDO DE ENTRADA POR PUERTO. EL
                 USUARIO ES PECIFICA EL PUERTO EL DATO DE ESTE ES LEIDO */
                    PROCEDURE;
450   2                  DECLARE PORT WORD;
451   2                 CALL SIO$TEST$WORD$MODE;
452   2                  PORT=SIO$GET$WORD;
453   2               LOOP:IF CHAR<>',' THEN GOTO ERROR;
455   2                  CALL SIO$CRLF;
456   2                 IF WORD$MODE THEN CALL SIO$OUT$WORD(INWORD(PORT));
 458  2                  ELSE CALL SIO$OUT$BYTE(INPUT(PORT));
459   2                 CALL SIO$GET$CHAR;
 460  2                  IF CHAR=ASCR THEN RETURN;
462   2                 GOTO LOOP;
4 63  2                  END;


464   1     SIO$OUTPUT:
                /*ESTE COMANDO IMPLEMENTA UNA SALIDA A PUERTO. EL USUARIO
                 ENTREGA EL DATO Y EL PUERTO  DE SALIDA */
                      PROCEDURE;
465   2                  DECLARE (DATUM,PORT)  WORD;
466   2                  CALL SIO$TEST$WORD$MODE;
467   2                  PORT=SIO$GET$WO RD;
468   2                  IF CHAR<>',' THEN GOTO ERROR;
470   2                  CALL SIO$GET $CHAR;
471   2                LOOP:DATUM=SIO$GET$WORD;
472   2                  IF CHAR<>':' THEN G OTO ERROR;
474   2                  IF WORD$MODE THEN OUTWORD(PORT)=DATUM;
476   2                   ELSE OUTPUT(PORT)=LOW(DATUM);
477   2                  IF CHAR=',' THEN DO;
479   3                   CALL SIO$CRLF;
480   3                  CALL SIO$OUT$CHAR('-');
481   3                   CALL SIO$OUT$BLANK;
482   3                  CALL SIO$GET$CHAR;
483   3                  IF CH AR<>ASCR THEN GOTO LOOP;
485   3                  END;
486   2                  END;
```

```
                $EJECT

487   1       SIO$WRITE:
                /* SE IMPLEMENTA UNA ESCRITURA A LA CONSOLA DE SALIDA
                    ACTUALMENTE ASIGNADA DE LAS POSICIONES DE MEMORIA
                   ESPECIFICADAS EN FORMATO HEXADECIMAL. SE ENVIA EL
                   RECORD DE DIRECCION EXTENDIDA (MODO 8086 SOLAMENTE),
                    LOS RECORDS DE DIRECCION DE COMIENZO (MODO 8086
                   SOLAMENTE), LOS RECORDS DE DATOS, Y EL RECORD DE EOF */
                    PROCEDURE;
488   2               DECLARE (START$REC,MODE$8086)BYTE, (LEN,INDEX)WORD;
489   2               CALL SIO$GET$CHAR;
490   2                MODE$8086=TRUE;
491   2               IF CHAR='X' THEN DO;
493   3                 MODE$8086=FALSE;
494   3                CALL SIO$GET$CHAR;
495   3               END;
 496   2               IF CHAR=ASBL THEN CALL SIO$GET$CHAR;
498   2               CALL SIO$GET$ADDR(@ARG1,CS);
499   2               IF CHAR<>',' THEN GOTO ERROR;
501   2               CALL SIO$GET$CHAR;
502   2               END$OFF=SIO$GET$WORD;
503   2               IF END$OFF <ARG1.OFF THEN GOTO ERROR;
505   2               IF CHAR<>ASCR THEN DO;
507   3                 START$REC=TRUE;
508   3                CALL SIO$GET$CHAR;
509   3                CALL SIO $GET$ADDR(@ARG3,CS);
510   3               END;
511   2               ELSE DO;
 512   3                START$REC=FALSE;
513   3               ARG3.OFF=0;
514   3                END;
515   2               IF CHAR<>ASCR THEN GOTO ERROR;
517   2               CALL  SIO$CRLF;
518   2               DO I=1 TO 60;
519   3                CALL SIO$OUT$CHAR(0);
520   3               END;
521   2               CALL SIO$CRLF;
522   2                IF MODE$8086 THEN DO;
524   3                IF START$REC THEN DO;
526   4                  CALL SIO$OUT$HEADER(04,0,03);
527   4                 CALL SIO$OUT$WORD(ARG3.SEG);
 528   4                  CALL SIO$OUT$WORD(ARG3.OFF);
529   4                 CALL SIO$OUT$BYTE(CHECK$SUM);
 530   4                  CALL SIO$CRLF;
531   4                 ARG3.OFF=0;
5 32   4                 END;
533   3              CALL SIO$OUT$HEADER(02,0,02);
534   3               CALL SIO$OUT$WORD(ARG1.SEG);
535   3              CALL SIO$OUT$BYTE(CHECKSUM);
536   3               CALL SIO$CRLF;
537   3              END;
538   2              LEN=STANDARD$LEN;
539   2       LOOP:INDEX=END$OFF-ARG1.OFF;
543   2            IF INDEX<STANDARD$LEN THEN  LEN=INDEX+1;
542   2            CALL SIO$OUT$HEADER(LEN,ARG1.OFF,00);
543   2            DO  I=1 TO LEN;
```

```
544   3              CALL SIO$OUT$BYTE(MEMORY$ARG1);
 545   3              ARG1.OFF=ARG1.OFF+1;
546   3             END;
547   2              CALL SIO$OUT$BYTE(CHECKSUM);
548   2             CALL SIO$CRLF;
549   2              IF END$OFF<>ARG1.OFF-1 THEN GOTO LOOP;
551   2             CALL SIO$OUT$HEADER(00,ARG3.OFF,01);
 552   2              CALL SIO$OUT$BYTE(CHECKSUM);
553   2             CALL SIO$CRLF;
554   2              DO I=1 TO 60;
555   3               CALL SIO$OUT$CHAR(0);
556   3             END ;
557   2             END;
```

```
                $EJECT

558   1     SIO$READ:
              /* SE EFECTUA UNA LECTURA  DE LA CONSOLA DE ENTRADA DEL
                 SISTEMA ACTUALMENTE ASIGNADA DE DATOS EN FORMATO
                  HEXADECIMAL */
                    PROCEDURE;
559   2               DECLARE BIAS WORD, (REC$TYPE,LEN,I,T) BYTE, OFFSET WORD;
560   2               BIAS=0;
561   2                ARG1.SEG=0;
562   2               CALL SIO$SCAN$BLANK;
563   2               IF CHAR<>ASCR THEN BIAS=SIO$GET$WORD;
565   2               IF CHAR<>ASCR THEN GOTO ERROR;
567   2                CALL SIO$CRLF;
568   2            LOOP:CALL SIO$GET$STRING;
569   2                 IF SIO$READ$CHAR<>':' THEN GOTO ERROR;
571   2               CHECK$SUM=0;
572   2                LEN=SIO$READ$BYTE;
573   2               OFFSET=SIO$READ$WORD;
574   2               ARG1.OFF=OFFSET+BIAS;
575   2               REC$TYPE=SIO$READ$BYTE;
576   2               IF REC$TYP E=03 THEN DO;
578   3                  CS=SIO$READ$WORD;
579   3                  IP=SIO$READ$ WORD;
580   3               END;
581   2               IF REC$TYPE=02 THEN ARG1.SEG=SIO$READ$W ORD;
583   2               IF REC$TYPE=01 THEN IF OFFSET<>0 THEN IP=OFFSET;
586   2                IF REC$TYPE=0 THEN DO I=1 TO LEN;
588   3                  T=MEMORY$ARG1=SIO$READ$BYTE;
58 9   3                  IF MEMORY$ARG1<>T THEN GOTO ERROR;
591   3                  ARG1.OFF=ARG1.OFF+1;
 592   3                END;
593   2               T=SIO$READ$BYTE;
594   2                IF CHECK$SUM<>0 THEN GOTO ERROR;
596   2               IF REC$TYPE<>01 AND LEN<>0 THEN GOTO LOOP;
 598   2                CALL SIO$OUT$CHAR(0);
599   2               CALL SIO$OUT$CHAR(0);
 600   2                END;


601   1     SIO$CONSOLE:
              /* SE ASIGNA LA CONSOLA DE SALIDA Y ENTRADA DEL MDS-221,
                 LO QUE PERMITE ACCEDER A  TODAS LAS POSIBILIDADES DE
                                COMUNICACION DEL ISIS-II. */
                                      PROCEDURE;
                602   2                DECLARE IMODE BYTE;
                 603   2                 IMODE=FALSE;
604   2                  CALL SIO$GET$CHAR;
605   2                  IF CHAR<>'I' AND CHAR<>'O' THEN GOTO ERROR;
607   2                   IF CHAR='I' THEN IMODE=TRUE;
609   2                  CALL SIO$GET$CHAR;
610   2                   IF CHAR<>'=' THEN GOTO ERROR;
612   2                  IF IMODE THEN DO;
614   3                      CALL EXCONSOL(@BUFFERIN(LONG$BUF$IN),@(':CO: '),@STATUS);
615   3                  END;
 616   2                   ELSE DO;
617   3                     CALL EXCONSOL(@(':CI: '),@BUFFERIN(LONG$BUF$ IN),@STATUS);
```

```
618   3                 END;
619   2                  END;
```

```
                $EJECT

620   1         SIO$END:
                   /* SE EFECTUA UNA SALIDA 'GRACIL' DEL SISTEMA */
                        PROCEDURE;
621   2                 CALL EXEXIT;
622   2                  END;
```

```
                $EJECT

                   /*PROGRAMA PRINCIPAL#####################################*/


                   /* EN ESTE MODULO, SE TRATA EL DESPACHO DE COMANDOS */


 623   1                  DISABLE;
624    1                 CALL SIO$OUT$STRING(@SIO$SIGNON);
 625   1                  CS,SS,DS,ES,FL,IP=0;
626    1                 SP=USER$INIT$SP;


                   /*MENSAJE DE SALIDA*/
627    1                 CALL EXOPEN(@AFTN$IN,@F ILEIN,1,0FF00H,@STATUS);


                   /*SE PROCEDE A LA ESCRITURA EN MEMOR IA LOCAL DE LOS
                     VECTORES DE INTERRUPCIONES */
628    1                 CALL INI T$INT$VECTOR(@INT$VECTOR(1),.INTERRUPT1$ENTRY);
629    1                 CALL INIT$INT$VECTOR(@INT$VECTOR(2),.INTE RRUPT3$ENTRY);
630    1                 CALL INIT$INT$VECTOR(@INT$VECTOR(3),.INTERRUPT3$ENTRY);
6 31   1                  BRK1$FLAG=FALSE;
632    1                 MONITOR$STACKPTR=STACKPTR;
633    1                  MONITOR$STACKBASE=STACKBASE;

634    1       NEXT$COMMAND:
                    /* EN ESTE BUCLE SE PROCEDE AL DESPACHO DE LOS COMANDOS
                      QUE SE RECIBEN A CADA PROC EDIMIENTO QUE SE ENCARGA DE
                      SU EJECUCION */
                          CALL  SIO$CRLF;
635    1                  CALL SIO$OUT$CHAR(0);
636    1                  CALL SIO$O UT$CHAR('>');
637    1                  CALL SIO$GET$STRING;
638    1                  CALL SI O$GET$CHAR;
639    1                  DO I=0 TO LAST(SIO$CMND);
640    2                       IF CHAR=SIO$CMND(I) THEN GOTO DISPATCH;
642    2                  END;
643    1                   GOTO ERROR;
644    1       DISPATCH:
                       LAST$COMMAND=I;
6 45   1                   DO CASE I;
646    2                       CALL SIO$EXAM$MEM;
647    2                        CALL SIO$EXAM$REG;
648    2                       CALL SIO$GO;
649    2                        CALL SIO$SINGLE$STEP;
650    2                       CALL SIO$MOVE;
651    2                        CALL SIO$DISPLAY;
652    2                       CALL SIO$INPUT;
653    2                       CALL S IO$OUTPUT;
654    2                       CALL SIO$READ;
655    2                       CALL SIO$WRITE ;
656    2                       CALL SIO$CONSOLE;
657    2                       CALL SIO$END;
 658   2                    END;
659    1                   GOTO NEXT$COMMAND;
```

```
660   1       ERROR:
                /*SE MANEJAN TODOS LOS ERRORE S DETECTADOS POR EL
                  MONITOR Y SE EXHIBE SIMBOLO DE ERROR */
                    CALL SIO$OUT$CHAR('#');
661   1             GOTO NEXT$COMMAND;
```

```
            $EJECT

662   1     AFTER$INTERRUPT:
              /* ESTA SUBRUTINA ES LLAMADA DESPUES DE UNA INTERUPCION
                 PARA MOSTRAR EL CS:IP Y RESTAURAR  LA INSTRUCCION DE
                 BREAKPOINT */
                        IF BRK1$FLAG T HEN DO;
664   2                 MEMORY$BRK1=BRK1$SAVE;
665   2                 BRK1$ FLAG=FALSE;
666   2                 IF ((IP-1)AND 000FH)=(BRK1.OFF AND 000FH) AND
                              (SHR(IP-1,4)+CS)=(SHR(BRK1.OFF,4)+BRK1.SEG) THEN DO;
668   3                     IP=IP-1;
669   3                    CALL SIO$OUT$STRING(@SIO$BREAK$MSG);
 670   3                  END;
671   2                 END;
672   1                  CALL SIO$OUT$CHAR('@');
673   1                 CALL SIO$OUT$WORD(CS);
674   1                  CALL SIO$OUT$CHAR(':');
675   1                 CALL SIO$OUT$WORD(IP);
 676   1                  GOTO NEXT$COMMAND;

677   1     END MONITOR;
             EOF
```

MODULE INFORMATION:

```
     CODE AREA SIZE     = 0F08H   3851D
    CONSTANT AREA SIZE = 0000H      0D
    VARIABLE  AREA SIZE = 00BCH    188D
    MAXIMUM STACK SIZE = 0046H     70D
    1962 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-86 COMPILATION
2

```
ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE LIBRERIAS
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM86 MDSLI1.SRC WORKFILES(:F0:,:F0:) NOOBJECT LARGE OPTIMIZE(2) RAM PAGEWIDTH(110)


  1          LIBRERIAS:DO:

             /* ESTAS SUBRUTINAS DEL PROGRAMA VER4.SRC, EFECTUAN LA
               COMUNICACION CON ACCESO EXCLUSIVO,  A LOS RECURSOS
               PROVISTOS POR EL ISIS-II. ESTE PROGRAMA ES EJECUTADO
                POR EL 8088, MIENTRAS QUE EL PROGRAMA DE INTERFASE CON
               EL ISIS-II, MANAG, ES EJECUTADO  POR EL 8080.


               SE PROPORCIONA SOPORTE A LAS SUBRUTINAS OPEN,READ,
               WRITE,CONSOLE,EXIT. PARA LA COMUNICACION CON MANAG,
               SE EMPLEA UNA ZONA RESERVADA QUE RADICA ENTRE A000H
               Y A00AH, EN EL QUE ESTAN EL SEMAFORO Y LAS DISTINTAS
               VARIABLES DE CONTROL DE ACCESO.


                EL PASO DE PARAMETROS A LAS SUBRUTINAS EXOPEN, EXREAD
               EXWRITE, EXCONSOLE, Y EXEXIT, ES  ANALOGO A SUS
               EQUIVALENTES DEL ISIS-II. LA UNICA DIFERENCIA ESTRIBA
                QUE LAS VARIABLES ADDRESS DE LOS SUBRUTINAS ISIS-II QUE
               HACIAN FUNCIONES DE PUNTERO, EST AN AHORA DE LA FORMA
               POINTER.                                              */


             /* MODULO DE DECLARACION DE VARIABLES ######################*/


  2   1      DECLARE BUSYFLAG BYTE AT (0A005H),    /* FLAG DE OCUPACION DEL ISIS-II*/
                      PAR$ISIS BYTE AT (0A006H),    /* OPERACION A REALIZAR */
                     D IR$ISIS WORD AT (0A007H),    /* DIRECCION BLOQUE PARAMETROS */
                     SEMAFORO BYTE AT (0A009H);     /* SEMAFORO DE CONTROL DE ACCESO
                                                       EXCLUSIVO AL ISIS-II.  SI SE
                                                       PRETENDE EMPLEAR MAS PROCESADORES
                                                        CON ACCESO AL ISIS-II, SE
                                                        SU ACCESO DEBE PASAR POR ESTA
                                                       VA RIABLE                   */


  3   1      DECLARE BUFFER$INS T(5) WORD,         /* BUFFER DE PARAMETROS DE PASO */
                     ERROR          BYTE;         /* FLAG  DE ERROR DIRECCION ILEGAL */


  4   1      DECLARE TRUE LITERALLY '0FFH',
                     FALSE LITERALLY '00H';

  5   1      DECLARE ILLEGAL*ADDRESS  LITERALLY '0101'; /*ERROR DE DIRECCION ILEGAL*/
```

```
                $EJECT

                /* MODULO DE SUBRUTINAS ######################################*/


 6   1          AJUSTE:
                   /* ESTA SUBRUTINA CALCULA LA DIREC CION REAL QUE VIENE EN
                      FORMA DE SEGMENTO MAS DESPLAZAMIENTO PARA QUE PUEDA
                       EMPLEARLA EL 8080. EN CASO DE QUE LA DIRECCION SE
                      MAYOR DE 64K SE GENERA  UN ERROR DE DIRECCION ILEGAL
                      (ERROR 101) */
                       PROCEDURE ( PUNTERO) WORD;
 7   2                 DECLARE PUNTERO POINTER;
 8   2                 DECLARE DATAPT R STRUCTURE(OFF WORD,SEG WORD) AT (@PUNTERO);
 9   2                 IF PUNTERO>0FFFFH THEN DO;
 11   3                                ERROR=TRUE;
12   3                               RETURN 0;
 13   3                               END;
14   2                 ELSE RETURN SHL(DATAPTR.SEG,4)+DATAPTR.OFF;
 15   2                 END;


16   1          EXTERNAL$ISIS:
                   /* SUBRUTINA MAS IMPORTANTE DEL PROGRAMA. AQUI SE PROCEDE
                      A LA COMU NICACION CON EL ISIS-II A TRAVES DE INTERFASE
                      MANAG. COMO PUEDE VERSE, PRIMERO SE EFECTUA EL  ACCESO
                      EXCLUSIVO POR MEDIO DE LA VARIABLE 'SEMAFORO' Y DE LA
                       FUNCION PLM-86 LOCKSET. UNA VEZ CONSEGUIDO EL ACCESO,
                      SE PASAN LOS PARAMETROS DE COMAN DO Y DIRECCION DEL
                      BUFFER DE INSTRUCCIONES Y SE ARRANCA EL PROCESO DEL
                       ISIS-II CON LA ACTIVACION DE 'BUSY$FLAG'. EN CASO DE
                      QUE EL COMANDO SEA DE SA LIDA, SE PARA EL PROCESO. SE
                      ESPERA A LA FINALIZACION DEL PROCESO DEL ISIS-II
                       (BUSY$FLAG=FALSE) PARA LIBERAR EL ACCESO EXCLUSIVO A
                      ESTE Y ACABAR EL  COMANDO */
                             PROCEDURE (COMANDO);
17   2                          DECLARE  COMANDO BYTE;

18   2                          DO WHILE LOCKSET(@SEMAFORO,0FFH);   /*P(MUTEX)*/
 19   3                             END;

23   2                          PAR$ISIS=COMAND 0;
21   2                          DIR$ISIS=AJUSTE(@BUFFER$INST(0));
22   2                          B USY$FLAG=TRUE;                    /*ARRANCA EL ISIS*/
23   2                          IF COMANDO=9 THEN
 24   2                                              HALT;           /*FIN DE PROCESO. EXIT*/
25   2                           DO WHILE BUSY$FLAG=TRUE;
26   3                             END;                           /*ESPERA FIN ALIZACION ISIS*/

27   2                          SEMAFORO=0;/*V(MUTEX*/

 28   2                          END;
```

```
            $EJECT

            /* SUBRUTINAS REALES DE INTERFASE CON 8088 ##################*/


            /* EL COMPORTAMIENTO DE TODAS ESTAS SUBRUTINAS ES ANALOGO.
                INICIALMENTE SE PONE EL FLAG DE ERROR, Y SE VAN PASANDO
               AL BUFFER DE PARAMETROS, LOS P ARAMETROS AJUSTADOS AL
               FORMATO 8080 QUE ENTREGA EL 8088. EN CASO DE QUE NO HAYA
                HABIDO ERROR, SE LANZA EL COMANDO. DE OTRO MODO SE RETORNA
               ERROR 101. */


29   1      EXOPEN:
               /* SUBRUTINA OPEN PARA 8088 . */
                  PROCEDURE (AFTN$PTR,FILE$PTR,ACCESS,MODE,STS$PTR) PUBLIC:
30   2              DECLARE (AFTN$PTR,FILE$PTR,STS$PTR) POINTER;
31   2             DECLARE (ACCESS,MODE) WORD;
 32  2              DECLARE ERR BASED STS$PTR WORD;
33   2             ERROR=FALSE;
3 4  2              BUFFER$INST(0)=AJUSTE(AFTNPTR);
35   2             BUFFER$INST(1)=AJUSTE(FILE$PTR);
 36  2              BUFFER$INST(2)=ACCESS;
37   2             BUFFER$INST(3)=MODE;
 38  2              BUFFER$INST(4)=AJUSTE(STS$PTR);
39   2             IF ERROR=FALSE THEN CALL EXTERNAL$ISIS(0 );
41   2                    ELSE ERR=ILLEGAL$ADDRESS;
42   2             END;


43   1      EXWRITE:
               /* SUBRUTINA WRITE PARA 8088 */
                     PROCEDURE (AFTN,BUFFER$PTR,COUNT,STS$PTR) PUBLIC:
44   2              DECL ARE (BUFFER$PTR,STS$PTR) POINTER;
45   2              DECLARE (AFTN,COUNT) WORD;
46   2               DECLARE ERR BASED STS$PTR WORD;
47   2              ERROR=FALSE;
48   2               BUFFER$INST(0)=AFTN;
49   2              BUFFER$INST(1)=AJUSTE(BUFFER$PTR);
50   2               BUFFER$INST(2)=COUNT;
51   2              BUFFER$INST(3)=AJUSTE(STS$PTR);
52   2               IF ERROR=FALSE THEN CALL EXTERNAL$ISIS(4);
54   2                     ELSE ERR=ILLEGAL$ADDRESS;
 55  2               END;


56   1      EXREAD:
                /* SUBRUTINA READ PARA 8088 */
                  PROCEDURE (AFTN,BUFFER$PTR,COUNT,ACTUAL$PTR, STS$PTR) PUBLIC;
57   2              DECLARE (BUFFER$PTR,ACTUAL$PTR,STS$PTR) POINTER;
58   2               DECLARE (AFTN,COUNT) WORD;
59   2              DECLARE ERR BASED STS$PTR WORD;
60    2              ERROR=FALSE;
61   2              BUFFER$INST(0)=AFTN;
62   2              BUF FER$INST(1)=AJUSTE(BUFFER$PTR);
63   2              BUFFER$INST(2)=COUNT;
64   2               BUFFER$INST(3)=AJUSTE(ACTUAL$PTR);
```

```
 65   2          BUFFER$INST(4)=AJUSTE(ST S$PTR);
 66   2          IF ERROR=FALSE THEN CALL EXTERNAL$ISIS(3);
 68   2                         ELSE ERR=ILLEGAL$ADDRESS;
 69   2          END;
```

```
                $EJECT

 70   1     EXEXIT:
              /*  SUBRUTINA EXIT PARA 8088 */
                  PROCEDURE PUBLIC;
 71   2             BUFFER$ INST(0)=0A02AH;
 72   2             CALL EXTERNAL$ISIS(9);
 73   2             END;


 74   1     EXLOAD:
              /* SUBRUTINA PARA CARGA DE PROGRA MAS 8080 EN MEMORIA DEL
                 ISIS-II. EN REALIDAD EL MONITOR NO LA EMPLEA PERO SE
                  PROPORCIONA A EFECTOS DE COMPLITUD */
                  PROCEDURE (FILE,BIAS,SWITCH,ENTR Y,STS$PTR) PUBLIC;
 75   2             DECLARE (FILE,STS$PTR) POINTER;
 76   2             DEC LARE (BIAS,SWITCH,ENTRY) WORD;
 77   2             DECLARE ERR BASED STS$PTR WORD;
 78   2              ERROR=FALSE;
 79   2             BUFFER$INST(0)=AJUSTE(FILE);
 80   2             BUF FER$INST(1)=BIAS;
 81   2             BUFFER$INST(2)=SWITCH;
 82   2             BUFFER$INST(3 )=ENTRY;
 83   2             BUFFER$INST(4)=AJUSTE(STS$PTR);
 84   2             IF ERROR=FALS E THEN CALL EXTERNAL$ISIS(6);
 86   2                             ELSE ERR=ILLEGAL$ADDRESS;
 87    2             END;

 88   1     EXERROR:
              /* SUBRUTINA DE E XHIBICION ERRORES DEL ISIS-II */
                  PROCEDURE (ERRNUM) PUBLIC;
 89   2               DECLARE ERRNUM WORD;
 90   2              BUFFER$INST(0)=ERRNUM;
 91   2               BUFFER$INST(1)=0A00AH;
 92   2              CALL EXTERNAL$ISIS(12);
 93   2              END;


 94   1     EXCONSOL:
              /* SUBRUTINA DE REASI GNACION DE CONSOLA DEL ISIS-II */
                   PROCEDURE (INFILE,OUTFILE,STS$PTR) PUBLIC;
 95    2               DECLARE (INFILE,OUTFILE,STS$PTR) POINTER;
 96   2              DECLARE ERR BASED ST S$PTR WORD;
 97   2              ERROR=FALSE;
 98   2              BUFFER$INST(0)=AJUSTE(INF ILE);
 99   2              BUFFER$INST(1)=AJUSTE(OUTFILE);
100   2              BUFFER$INST( 2)=AJUSTE(STS$PTR);
101   2              IF ERROR=FALSE THEN CALL EXTERNAL$ISIS(8);
103   2                              ELSE ERR=ILLEGAL$ADDRESS;
104   2              END;
```

```
            $EJECT


 105   1    EXCLOSE:
               /* SUBRUTINA PARA CIERRE DE FICHEROS */
                    PROCEDURE(AFTN,S TS$PTR) PUBLIC:
 106   2                 DECLARE STS$PTR POINTER:
 107   2                 DECLARE AFT N WORD:
 108   2                 DECLARE ERR BASED STS$PTR WORD:
 109   2                 ERROR=FALSE:
 110   2                 BUFFER$INST(0)=AFTN:
 111   2                 BUFFER$INST(1)=AJUSTE(STS$PTR) :
 112   2                 IF ERROR=FALSE THEN CALL EXTERNAL$ISIS(1):
 114   2                                 ELSE ERR=ILLEGAL$ADDRESS:
 115   2                 END:


 116   1     END LIBRERIAS:
            EOF



MO DULE INFORMATION:

     CODE AREA SIZE     = 0288H    696D
     CONSTANT AREA SIZ E = 0000H      0D
     VARIABLE AREA SIZE = 0008H     11D
     MAXIMUM STACK SIZE = 0022H     34 D
     233 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COM PILATION
3
```

```
ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE LOADER86
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM86 LOADER.SRC WORKFILES(:F0:,:F0:) NOOBJECT


            $TITLE('CARGADOR DE CODIGO 8086 ')
            $PAGEWIDTH(110 )
   1        LOADER86:DO;


               /* EL SIGUIENTE  MODULO EFECTUA UNA TRASLACION ENTRE
                  CODIGO HEXADECIMAL GENERADO POR EL PROGRAMA OH86, A
                   CODIGO COMPATIBLE CON EL COMANDO HEX03J DEL ISIS-II.
                  ESTO PERMITE QUE PROGRAMAS 8088 SEAN CARGADOS POR EL
                  ISIS-II SIN PROBLEMAS. EL CODIGO HEXADECIMAL D EL OH86
                  INCLUYE DOS NUEVOS CAMPOS, ORDEN 02, ORDEN 03. LA
                   PRIMERA ESPECIFICA EL SEGMENTO QUE EL CODIGO QUE SIGUA
                  SE UBICARA. EL CODIGO 03 ESPCIFICA  SEGMENTO Y OFFSET
                  PARA QUE COMIENZE LA EJECUCION. EL CARGADOR LOADER86
                   AJUSTA ESTE DEZPLAZAMIENTO PARA QUE LAS DIRECCIONES
                  PARA EL 8080 SEAN CORRECTAS, Y GENERA CODIGO PARA
                  GUARDAR EL SALTO EN LA POSICION RESERVADA A000H QUE
                   SE EMPLEA EN LOS OTROS PROGRAMAS (VER4.SRC POR EJEMPLO) */


                /* LOS FICHEROS DE ENTRADA Y SALIDA DEL LOADER, SON
                   INDEFECTIBLEMENTE DATAI Y DATAO, EN :F0:.EL
                   PROCEDIMIENTO PARA TRABAJAR ENTONCES ES EL SIGUIENTE:

                  -EFECTUAR TRASLACION DE CODIGO OBJETO 8088 A HEXADECIMAL
                   88 CON EL COMANDOOH86, ESPECIFICANDO COMO FICHERO DE
                   SALIDA, EL "DATAIN".

                  -LANZAR EL LOADER, QUE GENERARA EL FICHERO "DATAO".
                   CONVERTIR CODIGO OBJETO 8085 CON COMANDO HEX0BJ.

                   ATENCION!. ESTE CODIGO NO ES EJECUTABLE POR EL 8085.
                   SOLO CARGABLE POR EL ISIS-II. */


                /* ZONA DE DECLARACION DE VARIABLES */


   2    1       DECLARE (AFTNOUT,AFTNIN) ADDRESS;                /* AFTN FICHEROS */
   3    1       DECLARE FILEIN(*) BYTE  DATA(':F0:DATAI ');      /* FICHERO DE ENTRADA */
   4    1       DECLARE FILEOUT(*) BYTE DATA (':F0:DATAO ');     /* FICHERO DE SALIDA */
   5    1       DECLARE MIDBUFFER(128) BYTE ,
                        BUFFER OUT(128) BYTE ,
                        BUFFERIN(128) BYTE;                     /* BUFFERS TRATAMIENTO */
   6    1       DECLARE (CHECKSUM$IN,CHECKSUM$OUT) BYTE;         /* CHECKSUMS */
   7    1       DECLARE STAT US ADDRESS;
   8    1       DECLARE LIMITE ADDRESS;                          /* DIRECCION LIMITE */
   9    1       DECLARE ASCII(*) BYTE DATA ('0123456789ABCDEF');/* TABLA TRADUCCION */
```

```
                $EJECT

                    /* SUBRUTINAS EX TERNAS DEL ISIS-II */


10   1      OPEN:       PROCEDURE (AFTNPTR,FILE,ACCESS ,MODE,STATUS) EXTERNAL;
11   2                  DECLARE (AFTNPTR,FILE,ACCESS,MODE,STATUS) ADDRESS;
 12   2                  END OPEN;

13   1      READ:       PROCEDURE (AFTN,BUFFER,COUNT ,ACTUAL,STATUS) EXTERNAL;
14   2                  DECLARE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
 15   2                  END READ;

16   1      WRITE:      PROCEDURE (AFTN,BUFFER,COU NT,STATUS) EXTERNAL;
17   2                  DECLARE (AFTN,BUFFER,COUNT,STATUS) ADDRESS;
18    2                  END WRITE;

19   1      ERROR:      PROCEDURE (ERRNUM) EXTERNAL;
 20   2                  DECLARE (ERRNUM) ADDRESS;
21   2                  END ERROR;

22   1      EXIT:       PROCEDURE EXTERNAL;
23   2                  END EXIT;

                   /* LAS SIGUIENTES SUBRUTINAS EFECTUAN UTILIDADES. PARA LA
                       EXPLICACION DE ESTAS REMITIRSE AL LISTADO VER4.SRC, YA
                      QUE LA MAYORIA SON COPIADAS DE AL LI. SOLO CAMBIA LA
                      DECLARACION DE LAS VARIABLES. */


 24    1      SIO$RESET$READ:PROCEDURE;
25   2                   CALL READ(AFTNIN,.BUFFER$IN,128,. LIMITE,.STATUS);
26   2                   PTRIN=0;
27   2                   END SIO$RESE T$READ;


28   1      SIO$HEX:PROCEDURE(C) ADDRESS;
29   2               DECLARE C BYTE;
30   2              IF C<='9' THEN RETURN DOUBLE(C-30H);
32   2               ELSE RETURN DOUBLE (C-37H);
33   2              END SIO$HEX;


 34    1      SIO$READ$CHAR:PROCEDURE BYTE;
35   2                   IF LIMITE<PTRIN THEN  CALL  SIO$RESET$READ;
37   2                   PTRIN=PTRIN+1;
38   2                   RETURN B UFFERIN(PTRIN-1);
39   2                   END SIO$READ$CHAR;
```

```
                $EJECT

 40   1      SIO$OUT$CHAR:PROCEDURE (C) ;
 41   2                    DECLARE C BYTE;
 42   2                   BUFFEROUT(PTROUT)=C;
 43   2                    PTROUT=PTROUT+1;
 44   2                   IF PTROUT=128 THEN DO;
 46   3                    CALL WRITE(AFTNOUT,.BUFFEROUT,128,.STATUS);
 47   3                   PTROUT=0;
 48   3                    END;
 49   2                   END SIO$OUT$CHAR;


 50   1      SIO$OUT$BYTE:PROCEDURE (B) ;
 51   2                   DECLARE B BYTE;
 52   2                    CALL SIO$OUT$CHAR(ASCII(SHR(B,4) AND 0FH));
 53   2                   CALL SIO$OUT$CHAR(ASCII(B AND 0FH));
 54   2                   CHECKSUM$OUT=CHECKSUM$OUT-B;
 55   2                    END SIO$OUT$BYTE;


 56   1      SIO$OUT$WORD:PROCEDURE  (W);
 57   2                   DECLARE W ADDRESS;
 58   2                   CALL SIO$OUT$BYTE( HIGH(W));
 59   2                   CALL SIO$OUT$BYTE(LOW(W));
 60   2                   END;


 61   1      SIO$READ$BYTE:PROCEDURE BYTE;
 62   2                    DECLARE T BYTE;
 63   2                   T=LOW(SIO$HEX(SIO$READ$CHAR));
 64   2                    T=SHL(T,4)+LOW(SIO$HEX(SIO$READ$CHAR));
 65   2                   CHECKSUM$IN=CHECKSUM$IN+T;
 66   2                   RETURN T;
 67   2                   END SIO$READ$BYTE;


 68   1      SIO$READ$WORD:PROCEDURE ADDRESS;
 69   2                    DECLARE T BYTE;
 70   2                   T=SIO$READ$BYTE;
 71   2                   RETUR N SHL(DOUBLE(T),8)+DOUBLE(SIO$READ$BYTE);
 72   2                   END SIO$READ$WORD;


 73   1      SIO$CRLF:PROCEDURE;
 74   2              CALL SIO$OUT$CHAR(0AH);
 75   2               CALL SIO$OUT$CHAR(0DH);
 76   2              END SIO$CRLF;
```

```
          $EJECT

             /*PROGRAMA  PRINCIPAL. EL TRATAMIENTO DE CONVERSION DE
               CODIGO SE HACE DE FORMA COMPLETAMENTE ANALOGA A CO MO
               LA EFECTUA VER4.SRC, SOLO QUE SE COMPUTAN DOS CHECKSUM
                PARA LA ENTRADA Y LA SALIDA A LA VEZ */


 77   1      DECLARE BIAS ADDRESS, (R EC$TYPE,LEN,I,T) BYTE, OFFSET ADDRESS;
 78   1      DECLARE (START$REC,MODE$8086)BYTE, INDEX ADDRESS;
 79   1       DECLARE (PTROUT,PTRIN,DIRBEGIN) ADDRESS;
 80   1      DECLARE (CS,IP) ADDRESS;


 81   1              CALL OPEN(.AFTNIN,.FILEIN,1,0FF00H,.STATUS);
 82    1              CALL OPEN(.AFTNOUT,.FILEOUT,2,0,.STATUS);
 83   1              BIAS=0;
 84    1              PTROUT=0;
 85   1         LOOP1:CALL SIO$RESET$READ;
 86   1          LOOP:D O WHILE SIO$READ$CHAR<>':';END;
 88   1              CHECK$SUMIN=0;
 89   1              LEN =SIO$READ$BYTE;
 90   1              OFFSET=SIO$READ$WORD;
 91   1              REC$TYPE=SIO $READ$BYTE;
 92   1              IF REC$TYPE=03 THEN DO;
 94   2                 CS=SIO$READ $WORD;
 95   2                 IP=SIO$READ$WORD;
 96   2                 DIRBEGIN=SHL(CS,4)+ IP;
 97   2                 LEN=LEN-02H;
 98   2              END;
 99   1               IF REC$TYPE=02 THEN DO;
101   2                                  BIAS=SHL(SIO$READ$WORD,4);
 102   2                                  GOTO LOOP1;
103   2                                  EN D;
104   1              IF REC$TYPE=0 THEN DO I=1 TO LEN;
106   2                 MIDBUFFER( I)=SIO$READ$BYTE;
107   2              END;
108   1              T=SIO$READ$BYTE;
 109   1               IF CHECKSUMIN<>0 THEN CALL ERROR(0208);
111   1            CALL SIO$OUT$CHAR(':') ;
112   1            CHECKSUMOUT=0;
113   1            CALL SIO$OUT$BYTE(LEN);
 114   1             CALL SIO$OUT$WORD(BIAS+OFFSET);
115   1            CALL SIO$OUT$BYTE(REC$TYPE);
 116   1             IF REC$TYPE=03 THEN CALL SIO$OUT$WORD(DIRBEGIN);
118   1            ELSE DO I=1  TO LEN;
119   2               CALL SIO$OUT$BYTE(MIDBUFFER(I));
120   2            END;
 121   1             CALL SIO$OUT$BYTE(CHECKSUMOUT);
122   1            CALL SIO$CRLF;
 123   1             IF RECTYPE<>01H AND LEN<>0H THEN GOTO LOOP;
125   1            CALL WRITE(AFTNOUT,.B UFFEROUT,PTROUT,.STATUS);
126   1            CALL EXIT;
127   1      END LOADER86;
```

MODULE INFORMATION:

```
     CODE AREA SIZE      = 02EDH    749D
     CONSTANT AREA SIZE = 0024H     36D
     VARIABLE AREA SIZE = 01A2H    41 8D
     MAXIMUM STACK SIZE = 0020H     32D
     233 LINES READ
     0 PROGRAM ER ROR(S)

END OF PL/M-86 COMPILATION
3
```

```
ASM80 MANAG.SR1 PAGEWIDTH(110)


ISIS-II 8080/8085 MACRO ASSEMBLER V4.0        MODULE   PAGE    1


LOC  OBJ          LINE        SOURCE STATEMENT

                     1        EXTRN   ISIS
0004                 2 WRITE   EQU     4
A005                 3 PTR     EQU   0A005H    ;ZONA RESERVADA PARA EL PASO DE PARAMETROS AL
 A007                 4 DIRP    EQU   0A007H    ;MANAGER DESDE OTROS SUBSISTEMAS
A009                 5 SEMAP   EQU   0A009H
                     6 ;##################################################### ############
                     7 ;
                     8 ;EN A000H----->SALTO AL COMIENZO  REAL DEL PROGRAMA
                     9 ;EN A005H----->BUSYFLAG
                    10 ;EN A 006H----->CALL NUMBER
                    11 ;EN A007H----->PUNTERO DE BLOQUE DE PARAMETROS
                     12 ;EN A009H----->SEMAFORO DE ACCESO AL SISTEMA
                    13 ;EN A00AH----->RETORNO  DE ERROR
                    14 ;
                    15 ;################################## ####################################
                    16        CSEG
0000 112A00    C    1 7        LXI     D,BLOCK
0003 0E04           18        MVI     C,WRITE
0005 CD0005    E    1 9        CALL    ISIS
0008 3E00           20        MVI     A,00    ;INICIALIZAR SEMAFORO
 000A 3209A0        21        STA     SEMAP
000D 2F             22        CMA             ;PUESTA A FF PARA COM PROBAR EL BUSYFLAG
000E 2105A0         23        LXI     H,PTR
0011 3600           24        MVI     M,00H
0013 BE             25 LOOP:  CMP     M       ;SE ACTIVA EL ARRANQUE
0014 C21 300   C    26        JNZ     LOOP
0017 23             27        INX     H
0018 4E             28        MOV     C,M     ;CARGA DEL COMANDO
0019 2A07A0         29        LHLD    DIRP
 001C EB            30        XCHG
001D CD0005    E    31        CALL    ISIS    ;ARRANQUE DEL ISIS-II
 0020 2105A0        32        LXI     H,PTR
0023 3E00           33        MVI     A,0H
 0025 77            34        MOV     M,A
0026 2F             35        CMA
0027 C31300    C    36        JMP     LOOP
002A 0000           37 BLOCK: DW      0H      ;SALIDA CO
002 C 3200     C    38        DW      MESS    ;DIRECCION BUFFER
002E 2700           39        DW      039D    ;LONGITUD BUFFER
0030 5900      C    40        DW      STS             ;DIRECCION VOLCADO ERROR
 0032 1B            41 MESS:  DB      027D,'J'        ;LIMPIADO DE PANTALLA ESC 'J'
0033 4A
 0034 4D414E41      42        DB      ('MANAGER DE RECURSOS DE ISIS-II V1.0')
0038 47455220
 003C 44452052
0040 45435552
0044 534F5320
0048 4445 2049
004C 5349532D
0050 49492056
0054 312E30
```

```
 0057 0D            43          DB      0DH,0AH
 0058 0A
 0059               44 STS :    DS      2


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0        MODULE   PAGE     2


  LOC  OBJ         LINE        SOURCE STATEMENT

                     45        END


PUBLIC SYMBOLS


EXTERNAL SYMBOLS
ISIS   E 0000

USER SYMBOLS
 BLOCK  C 002A    DIRP   A 0007    ISIS   E 0000    LOOP   C 0013    MESS   C 0032    PTR    A 0005
 SEMAP  A 0009    STS    C 0059    WRITE  A 0004

ASSEMBLY COMPLETE,  NO ERRORS
 P
```

```
ASM80 INTERC.SRC

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0        MODULE   PAGE    1
 INTERCEPTADOR DE MEMTOP

 LOC  OBJ          LINE        SOURCE STATEMENT

                     1 $TITLE(' INTERCEPTADOR DE MEMTOP ')
                     2
                     3 ;FUNCION:  REDEFINICION DEL MEMTOP DEL SISTEMA, ES DECIR,
                     4 ;          DE LA CANTIDAD DE MEMORIA LOGICA QUE DISPONDRA
                     5 ;          EL SISTEMA PARA SU USO. EL RESTO DE LA MEMORIA
                     6 ;          NO SUFRIRA INTERACCIONES POR PARTE DEL MDS-221.
                     7 ;          ESTO POSIBILITA QUE OTRO PROCESADOR ESTE
                     8 ;          TRABAJANDO EN ESE AREA LIBRE, SIN PREOCUPACIONES
                     9 ;          DE LO QUE HAGA EL MDS.
                    10
                    11 ;REQUISITOS: RELANZAMIENTO DEL SISTEMA (NUEVA CARGA DEL
                    12 ;            ISIS-II, PERO EL SISTEMA NO SE RESETEA DESDE
                    13 ;            EL PUNTO DE VISTA HARDWARE.
                    14
                    15 ;LIMITACIONES: NO DEBE EMPLEARSE CUALQUIER TIPO DE COMANDO
                    16 ;              QUE RECOMPUTE Y REASIGNE EL VALOR DE LA
                    17 ;              VARIABLE MEMTOP. DEBE DEJARSE LA SUFICIENTE
                    18 ;              RAM DISPONIBLE PARA SOPORTAR EL ISIS-II
                    19 ;              Y ALGUNA OTRA APLICACION
                    20
 4000               21                  ORG     4000H
                    22
 4000 3E0C          23                  MVI     A,0CH   ;SELECCION DE BOOT/DIAGNOSTIC
 4002 D3FF          24                  OUT     0FFH    ;SALIDA POR EL PUERTO DE CONTROL
                    25                                  ;CPUC. A PARTIR DE AHORA Y HASTA
                    26                                  ;LA NUEVA CARGA DEL ISIS-II, LA
                    27                                  ;ZONA E800H CONTIENE EL BOOTING
                    28
 4004 3E02          29                  MVI     A,02H   ;SELECCION DIRECCION E800H
 4006 D3FF          30                  OUT     0FFH
                    31
 4008 21FFBF        32                  LXI     H,0BFFFH;PARA ESTE CASO, LA MEMORIA DISPONIBLE
                    33                                  ;PARA EL MDS SERA DE 48K. LOS 14K
                    34                                  ;SUPERIORES PUEDEN SER EMPLEADOS SIN
                    35                                  ;INTERFERENCIA. ESTO NO QUIERE DECIR
                    36                                  ;QUE EL 8080 NO PUEDA ACCEDER A ESTA
                    37                                  ;ZONA, SINO QUE EN CONDICIONES NORMALES
                    38                                  ;NO DEBERIA HACERLO.
                    39
 400B C34EE8        40                  JMP     0E84EH  ;SALTO A BS2. PARA REFERENCIAS VER LOS
                    41                                  ;LISTADOS DEL MONITOR. ATENCION CON
                    42                                  ;REASIGNACION DE DIRECCIONES!!
                    43
                    44                  END

PUBLIC SYMBOLS
```

EXTERNAL SYMBOLS

USER SYMBOLS

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0 MODULE PAGE 2
INTERCEPTADOR DE MEMTOP

ASSEMBLY COMPLETE, NO ERRORS

LISTADOS 89

```
ISIS-II 8089 CROSS-ASSEMBLER VER1.0

LOC  OBJ                  LINE

                          0001  ;ESTO ES UNA PRUEBA DEL FUNCIONAMIENTO DE UN PROGRAMA
                          0002  ;DE PRUEBA PARA EL ENSAMBLADOR 8089. OBSERVESE
                          0003  ;QUE LOS DISTINTOS CAMPOS ES TAN SITUADOS EN CIERTAS
                          0004  ;COLUMNAS. SOLO ESTAN PERMITIDOS ACTUALMENTE 256 SIMBOLOS
                          0005  ;ETIQUETAS. NO SE PERMITE TAMPOCO EL TRATAMIENTO
                          0006  ;DE LA INSTRUCCION DE MOVIMIENTO DE MEMORIA A MEMORIA.
                          0007  ;LOS SIMBOLOS  DEBEN EMPEZAR POR UN CARACTER ALFA,
                          0008  ;MIENTRAS QUE LOS DATOS NUMERICOS EMPIEZAN POR  UN NUMERO.
                          0009  ;
                          0010  ;123456789012345678
0000 0180                 0011  BEGIN:  MOV    GA,[GA]
0002 438503               0012        MOV    [GB].05H,GC
0005 A683                 0013        MOVB   IX,[PP+IX+]
0007 6484                 00 14        MOVB   [GA+IX],BC
0009 883025               0015        MOVBI  TP,025H
000C 0A4D3A28             0016         MOVBI  [GB].0AH,028H
0010 D1302400             0017        MOVI   CC,024H
0014 114E0A00             0018         MOVI   [GC],0AH
0018 838C27               0019        MOVP   TP,[GA].02 7H
001A 039827               0020        MOVP   [GA].027H,GA
001D 038827               0021         LPD    GA,[GA].027H
001F 11980FFF000F         0022        LPDI   GA,0FFF0FH
0025 E3A027               0023         ADD    MC,[GA].027H
0027 63D027               0024        ADD    [GA].027H,BC
002A 02A027               0025         ADDB   GA,[GA].027H
002C A2D027               0026        ADDB   [GA].0 27H,IX
002F 112000FF             0027        ADDI   GA,0FF00H
0033 13C027F00F           0028         ADDI   [GA].027H,0FF0H
0038 2820FF               0029        ADDBI  GB,0FFH
003B 0AC027A0             0030         ADDBI  [GA].027H,0A0H
003F 0038                 0031        INC    GA
0041  03E827              0032         INC    [GA].027H
0044 02E827               0033        INCB   [GA].027H
0047 003C                 0034         DEC    GA
0049 03EC27               0035        DEC    [GA].027H
004C 02EC                 0036        DECB   [GA]
004E E3A827               0037        AND    MC,[GA].027H
0050 63D827               0038        AND    [GA].027H,BC
0053 02A827               0 039        ANDB   GA,[GA].027H
0055 A2D827               0040        ANDB   [GA].027H,IX
0058  112800FF            0041         ANDI   GA,0FF00H
005C 13C827F00F           0042        ANDI   [GA].027H,0FF0H
0061 2828FF               0043         ANDBI  GB,0FFH
0064 0AC827A0             0044        ANDBI  [GA].027H,0A0H
0068 E3A427               0045        OR     MC,[GA].027H
006A 63D427               0046         OR     [GA].027H,BC
006D 02A427               0047        ORB    GA,[GA].027H
00 6F A2D427              0048         ORB    [GA].027H,IX
0072 112400FF             0049        ORI    GA,0FF00H
0076 13C427F00F           0050         ORI    [GA].027H,0FF0H
007B 2824FF               0051         ORBI   GB,0FFH
007E 0AC427A0             0052        ORBI   [GA].027H,0A0H
0082 002C                 0053         NOT    GA
0084 03DC27               0054        NOT    [GA].027H
0087 00DC                 0055         NOTB   [GA]
0089 62F427               0056        SETB   [GA].027H,3
0 08C E2F827              0057         CLR    [GA].027H,7
```

```
003F 899C6E              0058           CALL   [GA],BEGIN
0092 919D6AFF            0059           LCALL  [GB],BEGIN
0096 882067              0060           JM P    BEGIN
0099 912063FF            0061           LJMP   BEGIN
009D 084460              0062            JZ     GA,BEGIN
00A0 08E4995C            0063           JZ     [GA].099H,BEGIN
00A4 10445BFF             0064           LJZ    GA,BEGIN
00A8 13E49953FF          0065           LJZ    [GA].099H,BEGIN
 00AD 0AE4994F            0066           JZB    [GA].099H,BEGIN
00B1 12E4994AFF          0067           LJZB   [G A].099H,BEGIN
00B6 684047              0068           JNZ    BC,BEGIN
00B9 08E09943            0069            JNZ    [GA].099H,BEGIN
00BD D3403FFF            0070           LJNZ   CC,BEGIN
00C1 13E0 993AFF          0071           LJNZ   [GA].099H,BEGIN
00C6 0AE39936            0072           JNZB   [GA].099H,BEGIN
00CA 12E099FBFF          0073  FINAL:   LJNZB  [GA].099H,FINAL
00CF 0AB099F7            0074            JMCE   [GA].099H,FINAL
00D3 12B099F2FF          0075           LJMCE  [GA].099H,FINAL
00D8 0AB 499EE            0076           JMCNE  [GA].099H,FINAL
00DC 12B499E9            0077           LJMCNE [GA].099H,FINA L
00E1 EABC99E5            0078           JBT    [GA].099H,7,FINAL
00E5 12BC99E9FF          0079            LJBT   [GA].099H,0,FINAL
00EA 8AB899DC            0080           JNBT   [GA].099H,4,FINAL
0 0EE 72B899D7FF          0081           LJNBT  [GA].099H,3,FINAL
00F3 14949924D2          0082           TSL    [GA]. 099H,024H,FINAL
00F8 A000                0083           WID    8,16
00FA 6000                0084            XFER
00FC 4000                0085           SINTR
00FE 2048                0086            HLT
0100 0000                0087           NOP
                         0088            END

TABLA DE SIMBOLOS
BEGIN      0000    LAB
FINAL      00C A    LAB
ENSAMBLADO TERMINADO -NUMERO ERRORES 00
```

```
:1000000001824385054683648498302504D0A282B
:0B001000D1302400114E0A00838C2721
:060214000398270388276C
:09001F0011080FFF000FE3A027F8
:0600270063D02702A02780
:0302C00A2D027112000FF13C027F00F2820FF0AB1
:10023C00C027A0023803E8270200B27003C03EC2780
:05004C0300ECE3AB2711
:0600500063D02702A82777
:13005500A2D827112800FF13C827F00F2828FF0A68
:0600650C827A0E3A42758
:06006A0063D42702A42765
:13006F00A2D427112400FF13C427F00F2824FF0A5E
:13007F00C427A0302C03DC2703DC62F427E2F8275A
:13008F00899C6E919D6AFF98206791206JFF0844C9
:10009F00603BE4995C104458FF13E49953FF0AE492
:13004F00994F12E4994AFF68404708E09943D040B8
:1300BF003FFF13E0993AFF0AE0993612E099FBFFF0
:1300CF00AB099F712B099F2FF0AB499EE12B499E7
:0100DF00E937
:1300E100EABC99E512BC99E0FFBAB899DC72B89923
:1000F100D7FF1A949924D2A9036090400020460044
:0101010000FD
:00000001FF
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE ENSAMBLADOR89
OBJECT MODULE PLACED IN  XASM89.OBJ
COMPILER INVOKED BY:  PLM80 XASM89.SRC WORKFILES(:F0:,:F0:)


            $TITLE('MODULO PRINCIPAL ENSAMBLADOR 8089')
            $PAGEWIDTH(110)
  1         ENSAMBLADOR89:DO;

              /* EL SIGUIENTE PROGRAMA ES E L MODULO PRINCIPAL DEL
                 ENSAMBLADOR 8089 DESARROLLADO PARA EL SOPORTE DE
                  ESTE. LA ENTRADA DE ESTE ENSAMBLADOR ES UN FICHERO
                 EDITOR EN DONDE ESTA LA D ESCRIPCION DEL PROGRAMA.
                 EN EL LISTADO PRUEBA.LST, SE ENCUENTRA UNA SALIDA
                  GENERADA POR ESTE ENSAMBLADOR, EN DONDE SE PONEN
                 LAS EQUIVALENCIAS DE CAD A INSTRUCCION, Y SE OBSERVA
                 EL FORMATO QUE DEBE CUMPLIRSE, EN LAS COLUMNAS
                  DE LOS CAMPOS TAL COMO SE HACE EN FORTRAN. POR
                 AHORA, NO SE SOPORTAN OTRO S SIMBOLOS QUE LAS
                 ETIQUETAS. SIN EMBARGO, TODAS LAS VARIABLES Y
                  LA ESTRUCTURA DEL PROGRAMA ESTA CONCEBIDA PARA
                 UNA FACIL INCORPORACION DE ESTAS QU E NO SE HIZO
                 POR FALTA DE TIEMPO */

              /* MOD ULO DE DECLARACION DE SUBRUTINAS EXTERNAS
                 PERTENECIENTES A LOS MODULOS XLIB89.SRC Y XCOD89.S RC */


              /* LAS DESCRIPCIONES DE FUNCION DE CADA UNA DE LAS
                  SIGUIENTES SUBRUTINAS SE ENCUENTRAN EN EL PROLOGO DE
                 ESTAS * /


  2   1                 COMANDO$ASM:PROCEDURE EXTERNAL;
  3   2                             END COMANDO$ASM;


  4   1                  LEE$LINEA:PROCEDURE  EXTERNAL;
  5   2                           END LEE$LINEA;


  6   1                 COMENTARIO:PROCEDURE (PUNTERO) BYTE EXTERNAL;
  7   2                             DECLARE PUNTERO ADDRESS;
  8   2                            END COMENTARIO;


  9   1                 VE$ETIQUETA:PROCEDURE (PUNTERO) EXTERNAL;
 10   2                             DECLARE PUNTERO ADDRESS;
 11   2                            END  VE$ETIQUETA;


 12   1                 INC$SYMBOL:PROCEDURE (SIMBOLO,VALOR, TIPO) EXTERNAL;
 13   2                            DECLARE (SIMBOLO,VALOR) ADDRESS;
 14   2                             DECLARE TIPO BYTE;
 15   2                            END INC$SYMBOL;
```

```
                $EJECT

 16  1          TRATA$CODE:PROCEDURE (PUNTERO) EXTERNAL;
 17  2                DECLARE PUNTERO ADDR ESS;
 18  2                END TRATA$CODE;


 19  1          GEN$CODE:PROCEDURE (PUNTERO)EXTERNAL;
 20  2                DECLARE PUN TERO ADDRESS;
 21  2                END GEN$CODE;


 22  1          INICIAL:PROCEDURE EXTERNAL;
 23  2                END INICIAL;


 24  1          REINICIA:PROCEDURE  EXTERNAL;
 25  2                END REINICIA;


 26  1          FINALIZ AR:PROCEDURE EXTERNAL;
 27  2                END FINALIZAR;


 28  1          LONGSYM:PROCEDURE (PUNTERO) BYTE EXTERNAL;
 29  2                DECLARE PUNTERO ADDRESS;
 30  2                END LONGSYM;


 31  1          IMPRESION:PROCEDURE  EXTERNAL;
 32  2                END IMPRESION;


        /* MODULO DE DECLARACIO N DE VARIABLES GLOBALES */


 33  1          DECLARE EOFILE BYTE EXT ERNAL;          /* FLAG EOFILE */
 34  1          DECLARE NFICHERO(10) BYTE EXTERNAL;    /* NOMBRE FIC HERO */
 35  1          DECLARE LONGITUD BYTE EXTERNAL;
 36  1           DECLARE ETIQUETA BYTE EXTERNAL;        /* FLAG ETIQUETA */
 37  1          DECLARE (LINEA1,LINEA2)BYT E EXTERNAL; /* NUM. LINEA LISTADO */
 38  1          DECLARE FIN BYTE EXTERNAL;
 39  1           DECLARE VALOR ADDRESS EXTERNAL;        /* VALOR SIMBOLO */
 40  1          DE CLARE TRUE LITERALLY '0FFH';
 41  1          DECLARE FALSE LITERALLY '0H';
 42  1           DECLARE (AFTN1) ADDRESS EXTERNAL;      /* AFTN FICH. SOURCE */
 43  1          DEC LARE NEWAD ADDRESS EXTERNAL;        /* PUNTERO ACCESO BUFFER*/
 44  1          DECLARE INICIALIZACION  BYTE EXTERNAL; /* FLAG INICIALIZACION */
```

```
            $EJECT

               /* MODULO PRINCIPAL. LLAMADA A LOS COMANDOS */


               /* SE EFECTUAN DOS PASADAS AL ESTILO TRADICIONAL DE LOS
                   ENSAMBLADORES. LA PRIMERA PASADA EXTRAE EL VALOR DE LOS
                  SIMBOLOS QUE SE EMPLEARAN PARA LA GE NERACION DE CODIGO
                  DE LA SEGUNDA PASADA. ASI MISMO, SE HACE UN PREANALISIS
                   SINTACTICO */

               /* INICIALIZACION DE LAS VARIABLES Y OBTENCIO N OPCIONES
                  DE COMPILACION */

45   1                  CA LL COMANDO$ASM;
46   1                  CALL INICIAL;
47   1                  VALOR=0;
 48   1                  EOFILE=FALSE;



                /* MIENTRAS NO HAY FINAL DE TEXTO, DESCARTAR COMENTARIOS,
                  EXTR AER ETIQUETAS, COMPROBAR EL NUMERO DE BYTES DE
                  CADA INSTRUCCION, Y DETERMINAR EL VALOR DE LO S
                  SIMBOLOS */


49   1     PASADA1:      DO WHILE NOT(EOFILE);
50   2                  CALL LEE$LINEA;
51   2                  IF  COMENTARIO(NEWAD)=FALSE THEN DO;
53   3                     CALL VE$ETIQUETA(NEWAD);
54    3                     IF ETIQUETA THEN
55   3                     CALL INC$SYMBOL(NEWAD,VALOR,(80H OR
                                        (LONGSYM(NEWAD) AND 0FH)));
56   3                     CALL TRATA$CODE(NEWAD+8);
57   3                     VALOR=VALOR+LONGITUD;
58   3                     IF FIN THEN EOFILE=TRUE;
60   3                   END;
61   2                   INICIALIZACION=FALSE;
62   2                   END /*WHILE*/;


               /* FIN DE LA PRIMERA PASADA. COMIENZA ZONA DE GENERACION
                   DE CODIGO */


               /* REINICIALIZACION DE LAS VARIABLES  */


63   1                  EOFILE=FALSE;
64   1                   FIN=FALSE;
65   1                  CALL REINICIA;
66   1                   LINEA1=0;
67   1                  LINEA2=0;
68   1                  VALOR=0;
```

```
            $EJECT
               /* MIENTRAS NO H AYA FIN DE FICHERO, EXTRAER COMENTARIOS Y
                  GENERAR CODIGO SEGUN LOS SIMBOLOS RESULTANTES DE L A
                  PRIMERA PASADA */


 69   1     PASADA2:        DO WHILE NOT(EOFILE);
 70   2                     CALL LEE$LINEA;
 71   2                      IF COMENTARIO(NEWAD)=FALSE THEN DO;
 73   3                        CALL TRATA$CODE(NEWAD+8);
 74   3                         CALL GEN$CODE (NEWAD+15);
 75   3                        VALOR=VALOR+LONGI TUD;
 76   3                     END /*COMENTARIO*/;

                /* GENERAR NUMERO DE LINEA DE FICHERO LISTADO */


 77   2                      IF FIN THEN EOFILE=TRUE;
 79   2                     IF LINEA1>98H THEN LINEA2=LINEA2+1;
 81   2                      LINEA1=DEC(LINEA1+1);
 82   2                     CALL IMPRESION ;
 83   2                      INICIALIZACION=FALSE;
 84   2                     END /*WHILE*/;

               /* FIN DE PROGRAMA. CIERRE DE FICHEROS. RETORNO A ISIS-II */


 85   1                     CALL FINALIZAR;


 86   1      END ENSAMBLADOR89;


MODULE INFORMATION:

    CODE AREA SIZE     = 0108H    267D
    VARIABLE AREA SIZE = 0003H      3D
     MAXIMUM STACK SIZE = 0004H      4D
    191 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
1
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE GENERACIONCODIGO
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 XCOD89.SR1 WORKFILES(:F0:,:F0:) NOOBJECT


            $TITLE('GENERACION CODIGO 89')
            $PAGEWIDTH(110)
    1        GENERACION$CODIGO:DO;

               /* ENSAMBLADOR 89                * /
               /* MODULO DE GENERACION DE CODIGO */


                /* ZONA DE DECLARACION DE VARIABLES */
               /* LAS VARIABLES EXTERNAS SON EXPLICADAS EN E L MODULO DE
                  PROCEDENCIA */


    2   1                    DECLARE TRUE LITERALLY '0FFH';
    3   1                   DECLARE FALSE LITERALLY '00H';
    4   1                    DECLARE LONGITUD BYTE EXTERNAL;
    5   1                   D ECLARE VALOR ADDRESS EXTERNAL;
    6   1                   DECLARE FIN BYTE EXTERNAL;
    7   1                    DECLARE NEWAD ADDRESS EXTERNAL;
    8   1                   DECLARE (AFTN2) ADDRES S EXTERNAL;
    9   1                   DECLARE STS ADDRESS;
   10   1                    DECLARE AA BYTE;                  /* CAMPO INSTR. AA*/
   11   1                   DECLARE W8 BYTE;                  /* CAMPO INSTR. WB*/
   12   1                   DECLARE FOUND BYTE EXTERNAL;
```

```
$EJECT

   /* NEMOTEC NICOS EMPLEADOS EN EL ENSAMBLADOR 89. LO QUE
      VIENE ES UNA LISTA LINKADA EN DONDE SE CONTIENE  EL
      NEMOTECNICO, SU TIPO, Y EL NUMERO DE BYTES QUE EMPLEA */


13  1                                   DECLARE NMOTECNICOS (53) STRUCTURE(
                                              LETRA(7) BYTE,
                                             CODIGO   BYTE,
                                             DIRCODE BYTE,
                                              DIROPER BYTE,
                                             B8$16 BYTE ) DATA(
 'ADD    ',022H,00,30,0FH,'ADDB   ',022H,00,00,00H,
'ADDBI  ',022H,02,0 6,02H,'ADDI   ',022H,04,10,0FH,
'AND    ',022H,06,00,0FH,'ANDB   ',022H,06,00,00H,
 'ANDBI  ',022H,08,06,02H,'ANDI   ',022H,10,10,0FH,
'CALL   ',012H,12,20,0FH,'CLR    ',012H,13,14,00H,
'DEC    ',021H,14,00,0EH,'DECB   ',011H,15,01,00H,
 'HLT    ',010H,00,00,30H,'INC    ',021H,16,00,0EH,
'INCB   ',011H,17,01,00H,'JBT    ',013H,18,14 ,00H,
'JMCE   ',012H,19,20,00H,'JMCNE  ',012H,20,20,00H,
'JMP    ',0 11H,21,21,00H,'JNBT   ',013H,22,14,00H,
'JNZ    ',022H,23,24,0EH,'JNZB   ',012H,24,26,00H,
 'JZ     ',022H,25,24,0EH,'JZB    ',012H,26,26,00H,
'LCALL  ',012H,27,22,0FH ,'LJBT   ',013H,28,17,00H,
'LJMCE  ',012H,29,22,00H,'LJMCNE ',012H,30,20,00H,
 'LJMP   ',011H,31,23,0FH,'LJNBT  ',013H,32,17,00H,
'LJNZ   ',022H,33,28,0EH,'LJNZB  ',01 2H,34,30,00H,
'LJZ    ',022H,35,28,0EH,'LJZB   ',012H,36,30,00H,
'LP D   ',012H,37,37,0FH,'LPDI   ',012H,38,41,0FH,
'MOV    ',022H,39,00,0FH,'MOVB   ',022H,39,00,00H, /*ATENCION MOV*/
'MOVBI  ',022H,42,06,00H,'MOVI   ',022H,44,10,0FH,
 'MOVP   ',022H,46,00,0FH,'NOP    ',010H,01,00,00H,
'NOT    ',021H,48,00,0EH,'NOTB   ',011H,49,01,0 0H,/*ATENCION NOT*/
'OR     ',022H,51,00,0FH,'ORB    ',022H,51,00,00H,
 'ORBI   ',022H,53,06,00H,'ORI    ',022H,55,10,0FH,
'SETB   ',012H,57,14,00H,'SINTR  ',010H,02,0 0,30H,
'TSL    ',013H,58,32,00H,'WID    ',012H,59,35,00H,
'XFER   ', 010H,03,00,30H);
```

```
$EJECT

    /* CODIGO DE CADA UNO DE LOS NEMOTECNICOS. ESTA TABLA
      ESTA INDEXADA  POR LA ANTERIORMENTE DECLARADA */


14  1                              D ECLARE COD89(60) BYTE DATA(
/*ADC0*/101000$00B,110100$00B,/*ADDBI2*/001000$01B,110000$01B,
 /*ADDI4*/001000$10B,110000$10B,/*AND6*/101010$003,110110$00B,
/*ANDBI8*/001 010$01B,110010$01B,/*ANDI10*/001010$10B,110010$10B,
/*CALL12*/100111$01B,/*CLR13*/111110$00B,/*DEC 14*/001111$00B,111011$00B,
/*INC16*/001110$00B,111010$00B,/*JBT18*/101111$01B,/*JMCE19*/101100$01B ,
/*JMCNE20*/101101$01B,/*JMP21*/001000$01B,/*JNBT22*/101110$01B,
/* JNZ23*/010000$01B,111000$01B,/*JZ25*/010001$01B,111001$01B,
/*LCALL27*/100111$10B,/*LJBT28*/101111 $10B,/*LJMCE29*/101100$10B,
/*LJMCNE30*/101101$10B,/*LJMP31*/001000$10B,/*LJNBT32*/101110$10B,
 /*LJNZ33*/010000$10B,111000$10B,/*LJZ35*/010001$10B,111001$10B,
/*LPD37 */100010$00B,/*LPDI38*/000010$10B,
/*MOV39*/100000$003,100001$00B,100100$00B,/*MOVBI42*/001100$01B ,010011$01B,
/*MOVI44*/001100$10B,010011$10B,/*MOP46*/100011$00B,100110$00B,
 /*NOT48*/001011$00B,110111$00B,101011$00B,/*OR51*/101001$00B,110101$00B,
/*ORBI53*/001001 $01B,110001$01B,/*ORI55*/001001$10B,110001$10B,
/*SETB57*/111101$00B,/*TSL58*/100101$11B,/*WID59*/ 000000$00B);
/* */


   /* LAS SIGUI ENTES INSTRUCCIONES SON EXCEPCIONES A LA
      REGLA QUE PRECISAN UN TRATAMIENTO DIFERENTE */


15  1                           DECLARE XCEPTION(*) ADDRESS DATA(
 /*HLT*/00100000$01001000B,/*NOP*/0H,/*SINTR*/01000000$00000000B,
/*XFER*/0110 0000$00000000B);


   /* TIPOS DE LAS INSTRUCCIONES */


/*ATENCION 1-REGISTRO,2-PTR REG,3-INMED8,4-INMED16,5-INMED-32,6-MEMORIA*/
 /*7-LONG LABEL,8-SHORT LABEL,9-07,10-8/16,11-'DO NOTHING'*/


16  1                                DECLARE FORMAT(*) BYTE DATA(
/*00*/1,6,6,1,6 ,6,/*06*/1,3,6,3,/*10*/1,4,6,4,/*14*/6,9,8,/*17*/6,9,7,
/*23*/6,8,/*22*/6,7,/*24*/1,8,6,8,/*28*/1, 7,6,7,/*32*/6,3,8,/*35*/10,10,
/*37*/2,6,6,2,/*41*/2,5,/*43*/1,11,6,11,1,6);


   /* PSEUDO-INSTRUCCIONES */


17   1                            DECLARE PSEUCOD(7) STRUCTURE(
                                 LETRAS( 7) BYTE) DATA(
                                   'ORG    ','END    ','DS     ','EQU    ',
                                    'DB     ','DW     ','DD     ');
```

```
$EJECT

/* REGISTROS DE 20 BITS */

18  1                       DECLARE PTR(5) STRUCTURE(
          L ETRAS(2) BYTE) DATA('GA','GB','GC',0,0,'TP');

/* REGISTROS DE  16 BITS */

19  1                       DECLARE REGISTRO(8) STRU CTURE(
          LETRAS(2) BYTE) DATA('GA','GB','GC','BC','TP','IX','CC',
                               'MC');

/* V ALORES PARA LA INSTRUCCION WID */

20  1                       DE CLARE B07(8) BYTE DATA('01234567');

/* ESQUEMA DE DIRECCIONAM IENTO PRIMERO */

21  1                       DECLARE PTR$REG(4)  STRUCTURE(
          LETRAS(4) BYTE) DATA('[GA]','[GB]','[GC]','[PP]');

/* ESQUEMA DE DIRECCIONAMIENTO SEGUNDO */

22  1                       DECLARE PTR$REG$IX(4) STRUCTURE(
          LETRAS(7) BYTE) DATA('[GA+IX]','[GB+IX]','[GC+IX]',
                                                   '[PP+IX]');

/*  ESQUEMA DE DIRECCIONAMIENTO TERCERO */

23  1                        DECLARE IPTR$REG$IX(4) STRUCTURE(
          LETRAS(8) BYTE) DATA('[GA+IX+]','[GB+IX+]','[GC +IX+]',
                                                   '[PP+IX+]');

24  1                       DECLARE (NCODE,ETIQUETA,INICIALIZACION,LONGST) BYT E
                                                          EXTERNAL;
25  1                        DECLARE ASCII(16) BYTE EXTERNAL;
26  1                        DECLARE BUFFERLST(24) BYTE EXTERNAL;
27  1                       DECLARE (ERRORFLAG,TYERR,NERROR ) BYTE EXTERNAL;
```

```
$EJECT

   /* DECLARACION DE SUBRUTINAS EXTERNAS. LA EXPLICACION DE
       ESTAS SUBRUTINAS ESTA EN XLIB89.SRC EN EL PROLOGO DE
      CADA UNA DE ELLAS */


28  1          TRATA$ST:PROCEDURE (PUNTERO) ADDRESS EXTERNAL;
 29  2                   DECLARE PUNTERO ADDRESS;
30  2                 END TRATA $ST;

31  1          CHARACTER:PROCEDURE (PUNTERO,CARACTER) BYTE EXTERNAL;
 32  2           DECLARE PUNTERO ADDRESS;
33  2           DECLARE C ARACTER BYTE;
34  2           END CHARACTER;

35  1          SYM$IN$TAB:PROCEDURE (PUNTERO,LONGITUD) BYTE EXTERNAL;
36  2            DECLAR E PUNTERO ADDRESS;
37  2            DECLARE LONGITUD BYTE;
38  2             END SYM$IN$TAB;

39  1            ERRFATAL:PROCEDURE EXTERNAL ;
40  2             END ERRFATAL;

41  1          LONG$SYM:PROCEDURE (PUNTERO) BYTE EXTERNAL;
42  2            DECLARE PUNTERO ADDRESS;
 43  2             END LONG$SYM;

44  1          ALFA:P ROCEDURE (PUNTERO) BYTE EXTERNAL;
45  2               DECLARE PUNTERO ADDRESS;
46  2                END ALFA;

47  1          NUM:PROCEDURE (PUNTERO) BYTE EX TERNAL;
48  2              DECLARE PUNTERO ADDRESS;
49  2              E ND NUM;

50  1          HEXCONV:PROCEDURE (CARACTER) BYTE EXTERNAL;
 51  2            DECLARE CARACTER BYTE;
52  2            END HEXCONV;

53  1          WRITE:PROCEDURE (AFTN,BUFFER,COUNT,STATUS) EXTERNAL;
54  2            DECLARE (AFTN,BUFFER,COUNT,STATUS) ADDRESS;
55  2           END WRI TE;

56  1        GET$ADDRESS:PROCEDURE (PUNTERO) ADDRESS EXTERNAL;
57  2            DECLARE PUNTERO ADDRESS;
58  2           END GET$ADDRESS;

59  1          FIND:PROCEDURE(PUNTERO1,PUNTERO2,LONGITUD) BYTE EXTERNAL;
 60  2           DECLARE (PUNTERO1,PUNTERO2) ADDRESS;
61  2          DECLARE LO NGITUD BYTE;
62  2          END FIND;

63  1          HEXCOD:PROCEDURE (PUNTERO,DIRLD,LONGITUD,FINITO) EXTERNAL;
64  2            DECLARE (PUNTERO,DIR LD,LONGITUD) ADDRESS;
65  2            DECLARE FINITO BYTE;
66  2             END HEXCOD;
```

```
            $EJECT

                /* SUBRUTINA DE TRATAMIENTO INICIAL DE CODIGO. AQUI SE
                  COMPUTA LA VALIDEZ DE U NA DETERMINADA INSTRUCCION, EL
                  NUMERO DE BYTES QUE OCUPA INDEPENDIENTEMENTE DE
                   LOS SIMBOLOS, Y SE DAN RESULTADOS A LA SUBRUTINA DE
                  GENERACION DE COD IGO */

 67   1           TRATA$CODE:PROCEDURE (PUNTERO) PUBLIC;
 68   2                       DECLARE FINAL LABEL;
 69   2                      DECLARE PUNTERO ADDRESS;
 70   2                       DECLARE AUXAD ADDRESS;
 71   2                      DECLARE DUMMYCHAR BASED AUXA D ADDRESS;
 72   2                      DECLARE I BYTE;
 73   2                      LONGITU D=2;                    /* LONGITUD INICIAL */
 74   2                      DO NCODE=0 TO 52;              /*  BUSQUEDA INSTRUCCION */
 75   3                         IF FIND(PUNTERO,.NMOTECNICOS(NCODE).LETRA(0),7) THEN D O;
 77   4                         IF (NCODE=52) OR (NCODE=12) OR (NCODE=41)
                                          OR (NCODE=49) THEN GOTO FINAL;

                                /* SE ENCUENTRA OFFSET */
 79   4                         IF CHARACTER(PUNTERO+11,'.') THEN AA=01H;ELSE AA=0H ;
 82   4                         W8=(0003$0011B) AND (COD89(NMOTECNICOS(NCODE).DIRCODE));

                                /* AJUSTE DE LA LONGITUD DE INSTRUCCION SEGUN CAMPOS */
 83   4                          IF AA=01H THEN LONGITUD=LONGITUD+1;
 85   4                         IF W8=01H THEN L ONGITUD=LONGITUD+1;
 87   4                         IF W8>01H THEN LONGITUD=LONGITUD+2;
 89   4                          IF NCODE=35  THEN LONGITUD=LONGITUD+2;
 91   4                         GOTO FINAL;
 92   4                        END;
 93   3                      END;/*DO*/
 94   2                       IF ETIQUETA THEN GOTO FINAL;


                               /* TRATAMIENTO DE LOS PSEUDOCODIGOS ACTUALMENTE
                                 IMPLEMENTADOS                                  */
 96   2                      DO I=0 TO 2;
 97   3                       IF FIND(PUNTERO,.PSEUCOD(I).LETRAS(0),7) THEN DO;
 99   4                                    DO  CASE I;


                                           /* PSEUCOD ORG */
100   5                                     DO;
101   6                                      AUXAD=GETADDRESS(PUNTERO+8);
102   6                                     VALOR=DUMMYCHAR;
103   6                                      LONGITUD=0;
104   6                                    EN D;


                                           /* PSEU-COD END */
105   5                                    DO;
106   6                                     FIN= TRUE;
107   6                                     LONGITUD=0;
108   6                                     END;
```

```
                                        /* PSEU-COD EQU */
109   5                                 DO;
 110   6                                  AUXAD=GETADDRESS(PUNTERO+3);
111   6                                  LONGITUD=DUMMYCHAR
                                           /* ATENCION LA INCLUSION  DEL SIMBOLO
                                              DEPENDIENTE DE PRIMERA O SEGUNDA
                                               PASADA*/;
112   6                                  END;
113   5                               END /*CASE*/;
114   4                            END /*IF FIND*/;
115   3                     END /*DO*/;
116   2                  FINAL:END TRATA$CODE;
```

```
$E JECT

117  1          GEN$CODE:

             /* ESTA SUBRUTINA SE ENCARGA DE GENERAR EL CODIGO
              APROPIADO SEGUN LA INFORMACION QU E LA SUBRUTINA
              TRATA$CODE EXTRAE */

                        PROCEDURE (PUNTERO) PUBLIC;

           /* DECLAR ACION VARIABLES LOCALES */

118  2                  DECLARE ALLI LABEL;
119  2                  DECLARE GENCO LABEL;

123  2                   DECLARE (PTRLN,              /* PUNTERO DE LINEA */
                              PAUX,                  /* PUNTERO AUXILIAR */
                             PUNTERO) ADDRESS;

121  2                  DECLARE (RBP,              /* CAMPO RBP INSTRUCCION * /
                             OFFSET,               /* CAMPO DE OFFSET  */
                              DATA8FLAG,            /* FLAG DATO INMEDIAT. 8 BITS */
                              DATA16FLAG,           /* IDEM 16 BITS */
                             DATA32FLAG,           /* IDEM 32  BITS */
                             H)BYTE;               /* AUXILIAR */
122  2                   DECLARE (DISP16FLAG,       /* FLAG DEZPLAZAMIENTO 16 BITS */
                              DISP8FLAG) BYTE;     /* IDEM 8 BITS */

123  2                  DECLAR E (DATA8,I,J,MM,B) BYTE; /*VARIABLES AUXILIARES */
124  2                  DECLARE (DISP) ADDRESS;         /*DEZPLAZAMIENTO */
125  2                  DECLARE DATA32(4) BYTE;        /*DATO 32 BITS */
 126  2                  DECLARE DATA16(2) BYTE;        /*DATO 16 BITS */
127  2                  DECLARE DISP8 BYTE;            /*DEZPLAZAMIENTO 8 BITS */
128  2                  DECLARE CODE(6) BYT E;         /*CODIGO */
129  2                  DECLARE DUMMYCHAR BASED PAUX BYTE;

           /* SE PROCEDE AL ANALISIS DE EXCEPCIONES */

 130  2                   IF NCODE=53 THEN GOTO ALLI;
132  2                  I=0;
 133  2                   FOUND=FALSE;
134  2                  IF NCODE=52  OR  NCODE=49 / *HALT,SINTR,XFER,NOP*/
                        OR NCODE=12  OR NCODE=41 THEN DO;
136  3                         CODE(0)=HIGH(XCEPTION(NMOTECNICOS(NCODE).DIRCODE));
137  3                         CODE(1)=LOW(XCEPTION(NMOTECNICOS(NCODE).DIRCODE));
138  3                        I=2;
 139  3                         GOTO GENCO;
140  3                        END;
```

```
141   2                     DO WHILE (I<SHR(NMOTECNICOS(NCODE).CODIGO,4)) AND NOT(FOUND);

            /* INICIALIZACION DE VARIABLES Y FLAGS */

142   3                     PTRLN=PUNTERO;
143   3                     J=0;
144   3                     DATA8FLAG=FALSE;
145   3                     DATA16FLAG=FALSE ;
146   3                     DATA32FLAG=FALSE;
147   3                     FOUND=TRUE;
 148   3                     MM=0;
149   3                     DISP16FLAG=FALSE;
 150   3                     DISP8FLAG=FALSE;


151   3                     IF NCODE=8/*CALL*/ OR NCODE=24/*LCALL*/
                           OR NCODE=18/*JMP*/ OR NCODE=28/*LJMP */ THEN RBP=04H;
153   3                     ELSE RBP=0H;
154   3                     IF N CODE=51 /*WID*/ THEN RBP=01H;
156   3                     /*ATENCION RESETEADO DE OTRAS FLAGS PARA GENCO*/;
```

```
$EJECT

    /* SE PROCEDE A UNA DETERMINACION DE CASOS PARA UN
      TRATAMIENTO ADECUADO A CADA TIPO D E DIRECCIONAMIENTO
      O DE OPERANDO USADO */

 157  3                    DO WHILE (J<(NMOTECNICOS(NCODE).CODIGO AND 0FH)) AND FOUND;
158  4                        FOUND=FALSE;H=0;
160  4                   COMCASE: DO CASE (FORMAT(NMOTECNICOS(NCODE).DIROPER + J+
                                      I*(NMOTECNICOS(NCODE).CODIGO AND 0FH)))-1;


161  5                        A1:DO /*REGISTRO*/;
162  6                            DO  WHILE H<=7 AND NOT(FOUND);
163  7                             IF FIND(.REGISTRO(H).LETRAS(0),PTR LN,2) THEN DO;
165  8                                        RBP=H;
166  8                                         PTRLN=PTRLN+2;
167  8                                        FOUND=TRUE;
 168  8                                        END;
169  7                             H=H +1;
170  7                            END /*WHILE*/;
171  6                           EN D  /*CASE 1*/;


172  5                        A2:DO /*PTR-REG*/;
 173  6                           DO WHILE H<=3 AND NOT(FOUND);
174  7                            IF FIND(.PTR(H).LETRAS(0),PTRLN,2) THEN DO;
176  8                                  RBP=H;
 177  8                                   PTRLN=PTRLN+2;
178  8                                  FOU ND=TRUE;
179  8                                  END;
180  7                             H=H+1;
181  7                            END /*WHILE*/;
182  6                            END /*CASE 2*/;


183  5                        A3:DO /*INMED8*/;
 184  6                           PAUX=TRATA$ST(PTRLN);
185  6                            DATA8 FLAG=TRUE;
186  6                            DATA8=DUMMYCHAR;
187  6                             PTRLN=PTRLN+LONGST;
188  6                           END /*CASE 3*/;


189  5                        A4:DO /*INMED16*/;
190  6                            PA UX=TRATA$ST(PTRLN);
191  6                            DATA16FLAG=TRUE;
192  6                             DATA16(0)=DUMMYCHAR;
193  6                            PAUX=PAUX+1;
194  6                             DATA16(1)=DUMMYCHAR;
195  6                            PTRLN=PTRLN+LONGST;
 196  6                            END /*CASE 4*/;
```

```
$EJECT

197  5                    A5: DO /*INMED32*/;
198  6                         PAUX=TRATA$ST(PTRLN);
199  6                          DATA32FLAG=TRUE;
200  6                         DO H=0 TO 3;
201  7                          DATA32(H)=DUMMYCHAR;
202  7                         PAUX=PAUX+1;
203  7                          END;
204  6                         PTRLN=PTRLN+LONGST;
205   6                        END /*CASE 5*/;


206  5                     A6:DO /*MEMORIA*/;
207  6                        DO WHILE H<=3 AND NOT(FOUND);
208   7                        IF FIND (PTRLN,.IPTR$REG$IX(H).LETRAS(0),8) THEN DO;
210  8                                                    FOUND=TRUE;
211  8                                                    MM=H;AA=0000$00118;
213  8                                                   PTRLN=PTRLN+8;
 214  8                                                   END;
215  7                          H=H+1;
216  7                        END /*WHILE*/;
217  6                         H=0;
218  6                         DO WHILE H<=3 AND NOT(FOUND);
219  7                          IF FIND (PTRLN,.PTR$REG$IX(H).LETRAS(0),7) THEN DO;
             221  8                                                          FOUND=TRUE;
             222  8                                                          MM=H;AA=0020$00108;
             224  8                                                          PTRLN=PTRLN+7;
225  8                                                   END;
226  7                          H=H+1;
227  7                        END /*WHILE*/;
228  6                         H=0;
229  6                         DO WHILE H<=3 AND NOT(FOUND);
 230  7                          IF FIND (PTRLN,.PTR$REG(H).LETRAS(0),4) THEN DO;
232  8                                                    FOUND=TRUE;
233  8                                                    MM=H;AA=0H;
235  8                                                   PTRLN=PTRLN+4;
 236  8                                 IF CHARACTER(PTRLN,'.') THEN DO;
238  9                                       PTRLN=PTRLN+1;AA=01H;
240  9                                      PAUX=TRAT A$ST(PTRLN);
241  9                                      OFFSET=DUMMYCHAR;
242  9                                       PTRLN=PTRLN+LONGST;
243  9                                                 END;
244  8                                            END;
245  7                          H=H+1;
246  7                        END /*WHILE*/;
247  6                         END /*CASE 6*/;
```

```
                $EJECT


 248   5                        A7:DO /*LONG-LABEL*/;
 249   6                             PAUX=TRATA$ST(PTRLN);
 250   6                            DISP16FLAG=TRUE;
  251   6                             DISP=DOUBLE(DUMMYCHAR);
 252   6                            PAUX=PAUX+1;
 253   6                            DISP=SHR(DOUBLE(DUMMYCHAR),8) OR DISP;
 254   6                             DISP=DISP-(VALOR+LONGITUD);
 255   6                            PTRLN=PTRLN+LONGST;
  256   6                            END /*CASE 7*/;


 257   5                         A8:DO /*SHORT-LABEL*/;
 258   6                            PAUX=TRATA$ST(PTRLN);
  259   6                             DISP8FLAG=TRUE;
 260   6                            DISP8=DUMMYCHAR ;
 261   6                            DISP8=DISP8-(VALOR+LONGITUD);
 262   6                             PTRLN=PTRLN+LONGST;
 263   6                           END /*CASE 8*/;


 264   5                        A9:DO/*07*/;
 265   6                            DO W HILE H<=7 AND NOT(FOUND);
 266   7                             IF FIND (PTRLN,.B07(H),1) THEN DO;
  268   8                                                         FOUND=TRUE;
 269   8                                                         R8P=H;
 270   8                                                        PTRLN=PTR LN+1;
 271   8                                                        END;
 272   7                               H=H+1;
 273   7                           END /*WHILE*/;
 274   6                           END /*CASE 9*/;


 275   5                       A10:DO /*8$16 */;
 276   6                           DECLARE I8 BYTE DATA('8');
 277   6                            DECLARE I16(2) BYTE DATA('16');
 278   6                           IF FIND(PTRLN,.I8,1) THEN DO;
  280   7                                                    FOUND=TRUE;
 281   7                                                    PTRLN=PTRLN+1;
 282   7                                                   R8P=SHL(R8P ,1);
 283   7                                                   END;
 284   6                            ELSE DO;
 285   7                             IF FIND(PTRLN,.I16,2) THEN DO;
 287   8                                                    FOUND=TRUE;
 288   8                                                    PTRLN=PTRLN+2;
 289   8                                                   R8P=(SHL(R8P,1) OR 01H);
  290   8                                                    END;
 291   7                            END;
 292   6                           END /*CASE 10*/;
 293   5                         EN D /*BLOQUE CASE*/;
```

```
                $EJECT

 294   4                FINCASE: IF J+1<(NMOTECNICOS(NCODE).CODIGO AND 0FH) THEN DO;
296   5                           IF CHARACTER(PTRLN,',') THEN PTRLN=PTRLN+1;
298   5                          ELSE FO UND=FALSE;
299   5                        END;
300   4                         J=J+1;
 301   4                        END /*WHILE*/;
302   3                        I=I+1;
 303   3                        END /*WHILE*/;
304   2                        I=I-1;
 305   2                  TEST1: IF NOT(FOUND) THEN DO;
307   3                           COD E(0)=0010$0020B;
308   3                           CODE(1)=0100$1000B;
309   3                            I=2;
310   3                           ERRORFLAG=TRUE;
311   3                            TYERR=01H;
312   3                           NERROR=NERROR+1;
3 13   3                          GOTO GENCO;
314   3                         END;

              /* COMPOSICION DE LOS CAMPOS QUE FORMAN LA INSTRUCCION */

315   2                          IF I=0 THEN B=(01H AND NMOTECNICOS(NCODE).B8$16);
 317   2                               ELSE B=(01H AND (SHR(NMOTECNICOS(NCODE).B8$16,I)));
318   2                         CODE(0)=SHL(RBP,5) OR SHL(WB,3) OR SHL(AA,1) OR B;
319   2                        CODE( 1)=((COD89(NMOTECNICOS(NCODE).DIRCODE+I)
                                 AND 1111$1100B) OR MM);
 320   2                        I=2;

321   2                        IF AA =01H THEN DO;
323   3                           CODE(I)=OFFSET;
324   3                            I=I+1;
325   3                           END;

326   2                         IF DATA8FLAG THEN DO;
328   3                           CODE(I)=DATA8;
 329   3                            I=I+1;
330   3                           END;

331   2                        IF DATA16FLAG THEN DO;
333   3                            CODE(I)=DATA16(0);
334   3                           CODE(I+1)=DATA16(1);
335   3                            I=I+2;
336   3                           END;

 337   2                         IF DATA32FLAG THEN DO;
339   3                           DO H=0 TO 3;
340   4                           CODE(I+H)=DATA32(H);
341   4                            END;
342   3                           I=I+4;
343   3                            END;
```

```
            $EJECT

 344   2                          IF DISP8FLAG THEN DO;
 346   3                             CODE(I)=DISP8;
 347   3                              I=I+1;
 348   3                             END;

 349   2                          IF DISP16FLAG THEN DO;
 351   3                             CODE(I)=LOW(DISP);
 352   3                             CODE(I+1)=HIGH(DISP);
 353   3                              I=I+2;
 354   3                             END;


              /* PREPARACION BUFFER SALIDA LISTADO*/


 355   2                     GENCO: CALL HEXCOD(.CODE(0),VALOR,I,0);
 356   2                            BUFFERLST(0 )=ASCII(SHR(HIGH(VALOR),4) AND 0FH);
 357   2                            BUFFERLST(1)=ASCII(HIGH(VALOR) AND  0FH);
 358   2                            BUFFERLST(2)=ASCII(SHR(LOW(VALOR),4) AND 0FH);
 359   2                             BUFFERLST(3)=ASCII(LOW(VALOR) AND 0FH);
 360   2                             DO J=0 TO SHL(I-1,1) BY 2;
 361   3                            BUFFERLST(5+J)=ASCII(SHR(CODE(SHR(J,1)), 4) AND 0FH);
 362   3                            BUFFERLST(6+J)=ASCII(CODE(SHR(J,1)) AND 0FH);
 363   3                             END /*DO*/;
 364   2                      ALLI:  END GEN$CODE;


 365   1      END GENERACION$CODIGO;


MODULE INFORMATION:

     CODE AREA SIZE     = 0CE5H   3301D
      VARIABLE AREA SIZE = 002BH     43D
     MAXIMUM STACK SIZE = 0006H      6D
     623 LINES R EAD
     3 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
3
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SUBRUTINAS89
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 XLIB89.SRC WORKFILES(:F0:,:F0:) NOOBJECT


            $TITLE(' SUBRUTINAS ENSAMBLADOR 8089')
            $PAGEWIDTH(110)
     1       SUBRUTINAS89:DO;


              /* ESTE MODULO CONTIENE LAS SUBRUTINAS DE UTILIDADES DE
                 LOS MODULOS ANTERIORES XASM89.SRC Y XCOD89.SRC. AQUI
                  SE OFRECE LA EXPLICACION DE LA MAYORIA DE LAS
                 SUBRUTINAS * /


              /* ZONA DE DECLARACION DE VARIABLES */


     2   1                  DECLARE LONGITUD BYTE PUBLIC;  /* NUM. BYT. INSTRU */
     3   1                   DECLARE (AFTN1,AFTN2,AFTN3     /* AFTNS FICHEROS */
                                    ) ADDRESS PUBLIC;
     4   1                  DECLARE STS ADDRESS PUBLIC;    /* ER ROR ISIS */
     5   1                  DECLARE BUFFER(256) BYTE ;     /* BUFFER FICHERO */
     6   1                   DECLARE NFICHERO(12) BYTE PUBLIC;  /* NOMBRE FICHERO */
     7   1                   DECLARE NFICHEX (12) BYTE ;    /* NOMBRE FICHERO HEXA */
     8   1                  DECLARE NFI CHLST(12) BYTE ;    /* IDEM LISTADO */
     9   1                  DECLARE ETIQUETA BYTE PUBLIC;  /* FLAG D ETECCION */
    10   1                  DECLARE TRUE LITERALLY '0FFH';
    11   1                   DECLARE FALSE LITERALLY '0H';
    12   1                  DECLARE INICIALIZACION BYTE PUBLIC;  /* FLAG INCIAL*/
    13   1                  DECLARE INICIALIZACION1 BYTE;
    14   1                   DECLARE SLENGTH BYTE PUBLIC;    /* LONGITUD STRING */
    15   1                  DECLARE  NEWAD ADDRESS PUBLIC;
    16   1                  DECLARE EOFILE BYTE PUBLIC;     /* FLAG DETECCION EOF */
    17   1                  DECLARE LONGTABLA BYTE PUBLIC;  /* LONGITUD TABLA SIMBOLOS*/
    18   1                   DECLARE TABLAOK BYTE PUBLIC;    /* STATUS TABLA SIMBOLOS */
    19   1                   DECLARE SYMBUX BYTE PUBLIC;     /* PUNTERO SIMBOLO */
    20   1                  DECLARE VALOR  ADDRESS PUBLIC;   /* VALOR SIMBOLO */
    21   1                  DECLARE FIN BYTE PUBLIC;        /* FLAG  DETECCION FINAL */
    22   1                  DECLARE NCODE BYTE PUBLIC;      /* NUM. CODIGO INST. TABLA*/
    23   1                  DECLARE (LINEA1,LINEA2) BYTE PUBLIC; /* NUM. LINEA */
    24   1                   DECLARE BUFFERLST(24) BYTE PUBLIC;   /* BUFFER LISTADO */
    25   1                   DECLARE (ERRORFLAG,                  /* DETECCION ERROR*/
                                  TYERR,                 /* TIPO ERROR */
                                  NERROR) BYTE PUBLIC;   /* NUMERO ERROR  */
    26   1                  DECLARE ASCII(*) BYTE PUBLIC
                                   DATA ('0123456789ABCDEF'); /*TABLA TRADUCCION */
    27   1                  DECLARE LONGST BYT E PUBLIC;     /* LONGITUD STRING */
    28   1                  DECLARE FOUND BYTE PUBLIC;      /* FLAG ENC ON. SIMBOLO */
```

```
                $EJECT

 29   1                        DECLARE ERRMESS(5) STRUCTURE (
                              MENSAJE(2 0) BYTE) DATA (
              '*****ERROR<<<1>>>***',0DH,0AH,
              '*****ERROR<<<2>>>***',0 DH,0AH,
              '*****ERROR<<<3>>>***',0DH,0AH,
              '*****ERROR<<<4>>>***',0DH,0AH,
              '*****ERROR<<<5>>>***',0DH,0AH);                    /* MENSAJES DE ERROR */
 30   1                         DECLARE TABSYM(256) STRUCTURE(
                                  LETRAS(7) BYTE,
                                   VALUE  ADDRESS,
                                   TIPO      B YTE) PUBLIC;      /* TABLA DE SIMBOLOS */


 31   1                             DEC LARE FOUNDLT LITERALLY
                          'CHARACTER(NEWAD+SLENGTH,0DH) AND CHARACTER(NEWAD+SLENGTH+1,0A H)';


 32   1                            DECLARE EOBUFFER LITERALLY
                                   'NEWAD+SLENGTH-(.BUFFER(0))>EOBUFFERIND';


                    /* SUBRUTINAS DEL ISIS-II QUE SE EMPLEAN */


 33   1                      OPEN:PROCEDURE (AFTNPTR,FILE,ACCESS,MODE,STATUS)EXTERNAL;
 34   2                          DECLARE (AFT NPTR,FILE,ACCESS,MODE,STATUS)ADDRESS;
 35   2                          END OPEN;

 36   1                      READ:PROCEDURE (AFTN,BUFFER,COUNT,ACTUAL,STATUS)EXTERNAL;
 37   2                           DECLARE (AFTN,BUFFER,COUNT,ACTUAL,STATUS)ADDRESS;
 38   2                          END READ;

 39   1                     WRITE:PROCEDURE (AFTN,BUFFER,COUNT,STATUS)EXTERNAL;
 40   2                            DECLARE (AFTN,BUFFER,COUNT,STATUS)ADDRESS;
 41   2                           END WRITE;

 42   1                     CLOSE:PROCEDURE (AFTN,STATUS)EXTERNAL;
 43   2                            DECLARE (AFTN,STATUS) ADDRESS;
 44   2                           END CLOSE;

 45   1                      ERROR:PROCEDURE (ERRNUM) EXTERNAL;
 46   2                           DECLA RE ERRNUM ADDRESS;
 47   2                           END ERROR;

 48   1                      EXIT:PROCEDURE EXTERNAL;
 49   2                          END EXIT;
```

```
$EJECT

    /* MODULO DE SUBRUTINA S DE UTILIDADES */


50  1       CHARACTER:
          /* ESTA SUBRUTINA COMPARA EL CARACTER INDEXADO POR
            PUNTERO, CON EL OTRO PARAME TRO, RETORNA TRUE SI
            'GUALES, Y VICEVERSA. */
                        PROCEDURE (PUNTERO,CARACTER) BYTE PUBLIC REENTRANT;
51  2                   DECLARE PUNTER O ADDRESS;
52  2                   DECLARE CARACTER BYTE;
53  2                    DECLARE DUMMYCHAR BASED PUNTERO BYTE;
54  2                   IF DUMMYCHAR=CARACTER  THEN RETURN TRUE;
56  2                   RETURN FALSE;
57  2                    END CHARACTER;


58  1       COMENTARIO:
          /* ESTA SUBRUTINA DETECTA SI LA SENTENCIA EN CURSO ES
            UN COMENTARIO * /
                        PROCEDURE (PUNTERO) BYTE PUBLIC;
59  2                    DECLARE PUNTERO ADDRESS;
60  2                   RETURN CHARACTER(PUNTERO,';');
 61  2                   END COMENTARIO;


62  1           ERRFATAL:
          /* ESTA SUBRUTINA ABORTA EL ENSAMBLADO Y SALE FUERA DEL
             SISTEMA EN CASO DE ERROR */
                         PROCEDURE PUBLIC;
63  2                    CALL ERROR(STS);
64  2                     CALL EXIT;
65  2                   END ERRFATAL;
```

```
                $EJECT

 66   1                    COMANDO$ASM:
                        /* ESTA SUBRUTINA DETECTA LOS COMANDOS ENTRADOS EN EL
                          LANZAMIENTO DEL ENSAMB LADOR, Y EXTRAE EL NOMBRE DEL
                           FICHERO FUENTE Y GENERA LOS NOMBRES DEL HEX Y LST */
                                        PROCEDURE PUBLIC;
 67   2                                 DECLARE  (ACT) ADDRESS;
 68   2                                 DECLARE (I,J) BYTE;
 69   2                                  DECLARE APHEX(*) BYTE DATA('.HEX ');
 70   2                                 DECLARE APLST(*) BY TE DATA('.LST ');
 71   2                                 CALL READ(1,.NFICHERO,12,.ACT,.STS);
  72   2                                 IF STS<>0 THEN CALL ERRFATAL;
 74   2                                 I=1;
  75   2                                 DO WHILE NFICHERO(I)<>' ' AND NFICHERO(I)<>'.'
                                                 AND NFICHERO(I)<>0DH;
 76   3                                 NFICHEX(I)=NF ICHERO(I);
 77   3                                 NFICHLST(I)=NFICHERO(I);
 78   3                                  I=I+1;
 79   3                                 END;
 80   2                                  DO J=0 TO 4;
 81   3                                    NFICHEX(I+J)=APHEX(J);
 82    3                                   NFICHLST(I+J)=APLST(J);
 83   3                                 END /*DO*/;
 84   2                                 END COMANDO$ASM;


 85    1                   HEXCONV:
                        /* ESTA SUBRUTINA CONVIERTE UN DATO ASCII A HEXADECIMAL */
                                       PROCEDURE (DATO) BYTE PUBLIC;
 86   2                                  DECLARE DATO BYTE;
 87   2                                DECLARE I BYTE;
 88   2                                 DO I=0 TO LAST(ASCII);
 89   3                                IF DATO=ASCII(I) THEN RETURN I;
  91   3                                 END;
 92   2                                RETURN TRUE;
  93   2                                 END HEXCONV;
```

```
$EJECT


 94   1              GET$ADDRESS:
                   /* ESTA  SUBRUTINA RECOGE UNA DIRECCION DE 20 BITS Y LA
                      TRANSFORMA EN UN OFFSET Y UN SEGMENTO */
                                 PROCEDURE (PUNTERO) ADDRESS PUBLIC;
 95   2                           DECLARE PUNTERO ADDRESS;
 96   2                          DECLARE BUFF BYTE;
 97   2                           DECLARE INDAT(4) BYTE;
 98   2                          DECLARE DATAX BASED PUNTERO BYTE;
 99   2                          LONGST=0;
100   2                          INDAT(0)=0;
 101   2                          INDAT(1)=0;
102   2                          INDAT(2)=0;
 103   2                          INDAT(3)=0:
104   2                          BUFF=0;
1 05   2                          DO WHILE BUFF<> TRUE;
106   3                          INDAT(3)=SHR(INDAT(1),4)  AND 0FH;
107   3                          INDAT(1)=(SHL(INDAT(1),4) OR (SHR(INDAT(0),4) AND 0FH));
 108   3                          INDAT(0)=(SHL(INDAT(0),4) OR (BUFF AND 0FH));
109   3                          LONGST=LONGST+1;
110   3                          BUFF=HEXCONV(DATAX);
111   3                          PUNTERO=PUNTERO+1:
112   3                          END /*WHILE*/;
113   2                          IF DATAX='H' THEN LONGST=LONGST+1;
115   2                          RETURN .INDAT(0);
 116   2                          END GET$ADDRESS:


117   1                    INICIAL:
                   /* ESTA SUBRUTINA PROCEDE A LA APERTURA DE FICHEROS Y A
                       LA INICIALIZACION DE LAS VARIABLES DEL SISTEMA, Y
                      ENVIO DE MENSAJE * /
                                  PROCEDURE PUBLIC;
118   2                           DECL ARE MENSAJE (*) BYTE DATA(0DH,0AH,
                          'ISIS-II 8089 CROSS-ASSEMBLER.VER1.',0DH,0AH);
 119   2                           DECLARE (AUX1,AUX) BYTE;
120   2                          C ALL WRITE(0,.MENSAJE,38,.STS);
121   2                          IF STS<>0 THEN CALL ERRFATAL;
 123   2                           CALL OPEN(.AFTN1,.NFICHERO(1),1,0,.STS);
124   2                           IF STS<>0 THEN CALL ERRFATAL;
126   2                          INICIALIZACION=TRUE;
127   2                           LONGTABLA=0;
128   2                          NERROR=0;
129   2                           TABLAOK=TRUE;
130   2                          FIN=FALSE;
131   2                           DO AUX=0 TO 0FFH;
132   3                          DO AUX1=0 TO 6;
 133   4                           TABSYM(AUX).LETRAS(AUX1)=' ';
134   4                          END;
 135   3                           END;
136   2                          END INICIAL;
```

```
                $EJECT

137   1                 LEE$LINEA:
                  /* ESTA SUBRUTINA EXTRAE UNA SENTENCIA DEL DISCO. SE
                      PODIA HABER REALIZADO POR EL PROCEDIMIENTO DE LINEA
                     EDITADA */
                                 PROCEDURE PUBLIC;
138   2                         DECLARE OLDA D ADDRESS :
139   2                         DECLARE ACTUAL ADDRESS;
140   2                          DECLARE PREVEOFILE BYTE:
141   2                         DECLARE EOBUFFERIND ADDRESS;
 142  2                    LECTURA:PROCEDURE;
143   3                     CALL READ (AFTN1,.BUF FER(SLENGTH),256-SLENGTH,.ACTUAL,.STS);
144   3                           IF STS<>0 THEN CALL ERRFATAL;
 146  3                           IF ACTUAL<256-SLENGTH THEN DO;
148   4                                    PREVEOFILE=TRUE;
149   4                                   EOBUFFERIND=ACTUAL+SLE NGTH;
150   4                                   END;
151   3                           OLDAD=.BUFFER(0);
152   3                 END LECTURA;
153   2                           SLENGTH=0;
154   2                          IF INICIALIZACION THEN DO;
156   3                              PREVEOFILE=FALSE;
157   3                            EOBUFFERIND=255;
 158  3                            EOFILE=FALSE;
159   3                            CALL LECTU RA;
160   3                          END;
161   2                     START:NEWAD=OLDAD ;
162   2                          IF CHARACTER(NEWAD+SLENGTH,0AH) THEN DO;
164   3                                    SLENGTH=SLENGTH-1;
165   3                                   GOTO C ASIFIN;
166   3                                   END;
167   2                           DO WHILE NOT(FOUNDLT) AND NOT(EOBUFFER);
                                          /*FOUNDLT=FOUND L INE TERMINATOR*/
                                          /*EOBUFFER=END OF BUFFER*/
168   3                              SLENGTH=SLENGTH+1;
169   3                          END /*WHILE*/;
 170  2                           IF NOT(EOBUFFER) THEN GOTO CASIFIN;
172   2                           IF  PREVEOFILE=TRUE THEN DO;
174   3                                       EOFILE=TRUE;
 175  3                                        GOTO FINLINEA;
176   3                                       END;
177   2                          CALL MOVE (SLENGTH,NEWAD,.BUFFER(0));
 178  2                           CALL LECTURA;
179   2                          GOTO START ;
180   2                    CASIFIN: OLDAD=NEWAD+SLENGTH+2;
181   2                    FINL INEA:END LEE$LINEA;
```

```
                $EJECT

 182   1                    ALFA:
                 /* ESTA SUBRUTINA INDICA SI LO APUNTADO ES UNA LE TRA */
                            PROCEDURE (PUNTERO) BYTE PUBLIC;
 183   2                           DECLARE PUNTERO ADDRESS;
 184   2                          DECLARE I BYTE;
 185    2                          DECLARE ALFACHAR(*) BYTE DATA
                                   ('ABCDEFGHIJ KLMNOPQRSTUVWYXZ');
 186   2                          DO I=0 TO LAST(ALFACHAR);
 187   3                           IF CHARACTER(PUNTERO,ALFACHAR(I))=TRUE THEN
 188   3                                                RETURN TRUE;
 189   3                          END;
 190   2                           RETURN FALSE;
 191   2                          END ALFA;


 192   1                    NUM:
                 /* ESTA SUBRUTINA INDICA SI LO APUNTADO ES UN NUMERO  */
                            PROCEDURE (PUNTERO) BYTE PUBLIC:
 193   2                           DECLARE PUNTERO ADDRESS;
 194   2                          DECLARE I BYTE;
 195   2                          DECLARE NUMCHAR(*) BYTE DATA
                                   ('0123456789');
 196    2                          DO I=0 TO LAST(NUMCHAR);
 197   3                           IF CHARACTER(PUNTERO,NUMCHAR(I))=TRUE THEN
 198   3                                                RETURN TRU E;
 199   3                          END;
 200   2                          RETURN FALSE;
 201   2                          END NUM;


 202   1                    FIND:
                 /* ESTA SUBRUTINA COMPARA DOS PALABRAS APUNTADAS POR
                     PUNTERO1 Y PUNTERO2 Y RETORNA TRUE EN CASO DE QUE
                    SEAN IGUALES */
                            PROCEDURE(PUNTERO1,PUNTERO2,LONGITUD) BYTE PUBLIC;
 203   2                          DECLARE (PUNTERO1,PUNTERO2) ADDRESS;
 204   2                         DECLARE LONGITUD BYTE;
 205    2                          DECLARE H1 BYTE;
 206   2                         DECLARE DUMMYCHAR1 BASED PUNTER 01 BYTE;
 207   2                         DECLARE DUMMYCHAR2 BASED PUNTER02 BYTE;
 208   2                          DO H1=1 TO LONGITUD;
 209   3                            IF DUMMYCHAR1=DUMMYCHAR2 THEN DO;
 211    4                                      PUNTERO1=PUNTERO1+1;
 212   4                                       PUNTERO2=PUNTERO2+1;
 213   4                                      END;
 21 4   3                             ELSE RETURN FALSE;
 215   3                         END /*DO*/;
 216    2                          RETURN TRUE;
 217   2                         END FIND;
```

```
            $EJECT

218   1                 LONG$SY M:
                  /* ESTA SUBRUTINA EXTRAE LA LONGITUD DE UN SIMBOLO */
                           PROCEDURE (PUNTERO) BYTE  PUBLIC;
219   2                  DECLARE PUNTERO ADDRESS;
220   2                  DECLARE I BYTE;
221   2                  I=0;
 222   2                  DO WHILE ALFA(PUNTERO+I) AND I<=6;
223   3                  I=I+1;
224   3                  END /*WHILE*/;
225   2                  RETURN I-1;
226   2                  END LONG$SYM;


 227   1                 EX$SYMBOL:
                  /* ESTA SUBRUTINA INDICA SI LO APUNTADO ES  UN SIMBOLO */
                           PROCEDURE  (PUNTERO) BYTE PUBLIC;
228   2                   DECLARE PUNTERO ADDRESS;
229   2                  RETURN ALFA(PUNTERO)  ;
230   2                  END EX$SYMBOL;


231    1                 VE$ETIQUETA:
                  /* ESTA SUBRUTINA DETECTA LA PRESENCIA DE UNA ETIQUETA */
                              PROCEDURE (PUNTERO)  PUBLIC;
232   2                     DECLARE PUNTERO ADDRESS;
233   2                     IF EX$SYMBOL(PUNTERO) THEN DO;
 235    3                 IF CHARACTER(PUNTERO+LONG$SYM(PUNTERO)+1,':') THEN ETIQUETA=TRUE;
237    3                      ELSE ETIQUETA=FALSE;
238   3                     END;
 239   2                      ELSE ETIQUETA=FALSE;
240   2                     EN D VE$ETIQUETA;


241   1                 INC$SYMBOL:
                  /* ESTA SUBRUTINA INCLUYE UN SIMBOLO A LA TABLA */
                            PROCEDURE (P UNTERO,VAL,TIPO) PUBLIC;
242   2                   DECLARE (PUNTERO,VAL)ADDRESS;
2 43   2                   DECLARE TIPO BYTE;
244   2                   DECLARE I BYTE;
 245   2                   DECLARE CAREX BASED PUNTERO BYTE;
246   2                   IF LONGTABLA<0FFH THEN TABLAOK=TRUE;
248   2                                  ELS E DO;
249   3                                       TABLAOK=FALSE;
250   3                                        GOTO ADIOS;
251   3                                    END;
252   2                   DO I=0 TO (0FH AND TIPO);
253   3                    TABSYM(LONGTABLA).LETRAS(I)=CAREX;
254   3                   PUNTERO=PUNTERO+1;
255   3                   END /*DO*/;
256   2                   TABSYM (LONGTABLA).VALUE=VAL;
257   2                   TABSYM(LONGTABLA).TIPO=TIPO;
258    2                   LONGTABLA=LONGTABLA+1;
259   2                ADIOS:END INC$SYMBOL;
```

```
$EJECT

260  1                    SYM$IN$TAB:
                  /* ESTA SUBRUTINA BUSCA UN DETERMINADO SIMBOLO EN LA
                      TABLA. SI LO ENCUENTRA RETORNA TRUE */
                                       PROCEDURE (PUNTERO,LONGI TUD) BYTE PUBLIC;
261  2                                   DECLARE PUNTERO ADDRESS;
262  2                                    DECLARE LONGITUD BYTE;
263  2                                   DECLARE I BYTE;
 264  2                                    SYMBUX=0;
265  2                                   DO WHILE (SY MBUX<LONGTABLA) AND (TABLAOK);
266  3                                   IF ((TABSYM(SYMBUX).TIPO)AND(0FH))=LONGIT UD THEN DO;
268  4                       IF FIND(PUNTERO,.TABSYM(SYMBUX).LETRAS(0),LONGITUD) THEN RETURN TRUE;
 270  4                                    END;
271  3                                   SYMBUX=SYMBUX+1;
272  3                                   END /*DO*/;
273  2                                   RETU RN FALSE;
274  2                                   END SYM$IN$TAB;
```

```
                $EJECT

275   1                 HEXCOD:
                      /* ESTA SUBRUTINA GENERA EL CODIGO HEXADECIMAL PARA EL
                        FICHERO HEX */
                                 PROCEDURE (PUNTERO,DIRLD,LONGITUD,FINITO) PUBLIC;
276   2                          DECLARE (PUNTERO,DIRLD,DIRACT) ADDRESS;
277   2                         DECLARE (LONGITUD,CHECKSUM,LONGLI NE,FINITO) BYTE;
278   2                         DECLARE (I,AUX) BYTE;
279   2                          DECLARE BUFFEROUT(48) BYTE;
280   2                         DECLARE DUMMYCHAR BASED PUNTERO BYTE;
 281   2                          DECLARE SALIDA LABEL;

                      /* ESTAS SUBRUT INAS SE EMPLEAN DENTRO DE HEXCOD */

282   2                       CIERRE:
                        /* ESTA SUBRUTINA TERMINA UNA LINEA HEXA */
                                 PROCEDURE:
 283   3                          DECLARE AUX BYTE;
284   3                         CHECKSUM=-(CHECKSUM+LONGLINE);
285   3                         BUFFEROUT(1)=ASCII(SHR(LONGLINE,4) AND 0FH);
                      28 6   3                    BUFFEROUT(2)=ASCII(LONGLINE AND 0FH);
                      287   3                     AUX=9+SHL(LONGLINE,1) /*ATENCION MULTIPLICACION x2 */;
                      288   3                    BUFFEROU T(AUX)=ASCII(SHR(CHECKSUM,4) AND 0FH);
289   3                         BUFFEROUT(AUX+1)=ASCII(CHECKSUM AND 0FH);
290   3                         BUFFEROUT(AUX+2)=0DH;
291   3                         BUFFEROUT (AUX+3)=0AH;
292   3                         CALL WRITE (AFTN2,.BUFFEROUT(0),AUX+4,.STS);
293    3                          IF STS<>0 THEN CALL ERRFATAL;
295   3                         LONGLINE=0;
 296   3                          END CIERRE;

297   2                        APERT:
                        /* ESTA SUBRUTINA HACE LA CABECERA DE LINEA HEXA */
                                 PROCEDU RE;
298   3                         BUFFEROUT(0)=':';
299   3                         BUFFEROUT( 3)=ASCII(SHR(HIGH(DIRACT),4) AND 0FH);
300   3                         BUFFEROUT(4)=ASCII(HIGH(DIRACT) AND 0FH);
301   3                         BUFFEROUT(5)=ASCII(SHR(LOW(DIRACT),4) AND 0FH);
302   3                          BUFFEROUT(6)=ASCII(LOW(DIRACT) AND 0FH);
303   3                         BUFFEROUT(7)='0';
 304   3                          BUFFEROUT(8)='0';
305   3                         CHECKSUM=HIGH(DIRA CT)+LOW(DIRACT);
306   3                         LONGLINE=0;
307   3                         END  APERT;


308   2                      FINFILE:
                      /* E STA SUBRUTINA GENERA UN FIN DE FICHERO HEXA */
                                 PROCEDURE:
309    3                          DECLARE EOFMESS(*) BYTE DATA (':00000001FF',0DH,0AH);
310   3                          CALL WRITE(AFTN2,.EOFMESS,SIZE(EOFMESS),.STS);
311   3                         IF STS<>0 THEN CALL ERRFATAL;
 313   3                          END FINFILE;
```

```
                $EJECT
                  /* MODULO PRINCIPAL DE LA SUBRUTINA HEXCOD */
314   2                        IF FINITO THEN DO;
316   3                       CALL CIERRE;
317   3                        CALL FINFILE;
318   3                       GOTO SALIDA;
319   3                        END;
320   2                       IF INICIALIZACION1 THEN DO;
322   3                                          INICIALIZACION1=FALSE;
323   3                                             DIRA CT=DIRLD;
324   3                                             CALL APERT;
325   3                                              END;


326   2                       IF DI RACT<>DIRLD THEN DO;
328   3                                            CALL CIERRE;
329   3                                             DIRACT=DIRLD;
330   3                                             CALL APERT;
331   3                                            END;
332   2                        DO I=1 TO LONGITUD;
333   3                            IF LONGLINE<16 THEN DO;
335    4                               AUX=9+SHL(LONGLINE,1);
336   4                                  BUFFEROUT(AU X)=ASCII(SHR(DUMMYCHAR,4) AND 0FH);
337   4                                  BUFFEROUT(AUX+1)=ASCII(DUMMYCHAR  AND 0FH);
338   4                                  CHECKSUM=CHECKSUM+DUMMYCHAR;
339   4                                   DIRACT=DIRACT+1;
340   4                                  LONGLINE=LONGLINE+1;
 341   4                                   PUNTERO=PUNTERO+1;
342   4                               END;
343   3                              ELSE DO;
344   4                                CALL  CIERRE;
345   4                                CALL APERT;
346   4                                 I=I-1;
347   4                              END;
348   3                             END /*DO*/;
349   2              SALIDA:  END HEXCOD;
```

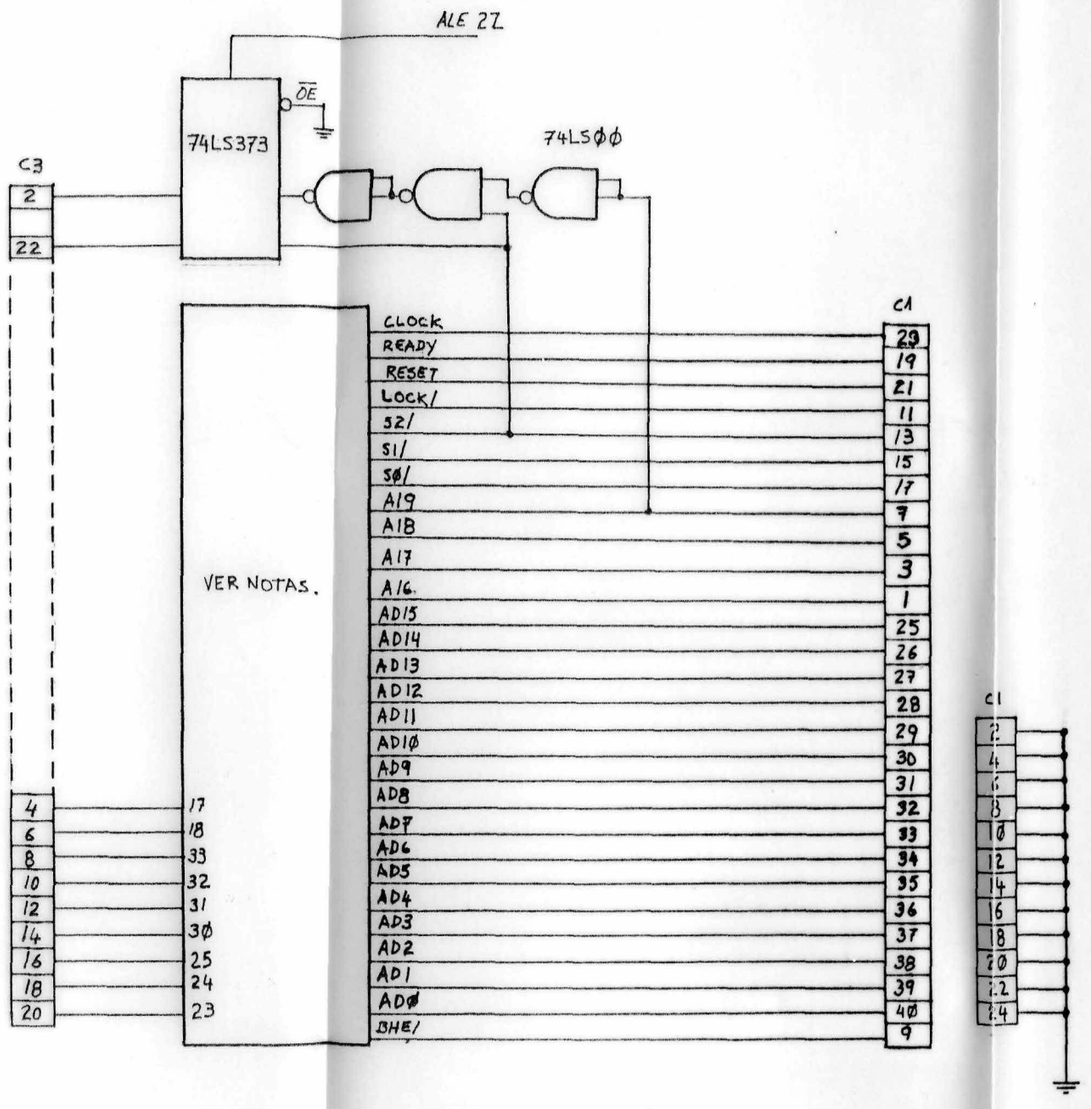
```
                *EJECT


 350   1                      IMPRESION:
                   /* ESTA SUBRUTINA PROCEDE A LA IMPRESION DE UNA LINEA DEL
                     FICHERO LISTADO */
                                      PROCEDURE  PUBLIC;
 351   2                                 DECLARE A BYTE:
 352   2                                CALL WRITE (AFTN3,.BUFFERLST,24,.STS);
 353   2                                 IF STS<>0 THEN CALL ERRFATAL:
 355   2                           BUFFERLST(0)=ASCII(SHR(LINEA2,4) AND 0FH);
 356   2                          BUFFERLST(1)=ASCII(LINEA2 AND 0FH):
 357   2                           BUFFERLST(2)=ASCII(SHR(LINEA1,4) AND 0FH);
 358   2                           BUFFERLST(3)=ASCII(LINEA1 AND 0FH):
 359   2                                    BUFFERLST(4)=' ';
 360   2                                     BUFFERLST(5)=' ':
 361   2                                    CAL L WRITE (AFTN3,.BUFFERLST,6,.STS);
 362   2                                    IF STS<>0 THEN CALL ERRFATAL;
 364   2                                     CALL WRITE (AFTN3,NEWAD,SLENGTH+2,.STS);
 365   2                                     IF STS<>0 THEN CALL ERRFATAL:
 367   2                                    IF ERRORFLAG THE N DO;
 369   3                                 CALL WRITE (AFTN3,.ERRMESS(TYERR).MENSAJE(0),20,.STS);
 370   3                                  IF STS<>0 THEN CALL ERRFATAL;
 372   3                                       ERRORFLAG=FALSE;
 373   3                                    END;
 374   2                                     DO A=0 TO 23;BUFFERLST(A)=' ';END:
 377   2                                    END IMPRESION;


 378   1                      REINICIA:
                  /*ESTA  SUBRUTINA SE ACTIVA DESPUES DE LA PRIMERA PASADA Y
                    SE ENCARGA DE LA APERTURA DE LOS FICHEROS  HEXADECIMAL
                    Y LISTADO Y DE ALGUNA OTRA TAREA */
                                      PROCEDURE PUBLIC;
 379   2                                 DECLARE A BYTE;
 380   2                                  DECLARE MEN$ISIS(*) BYTE
          DATA('ISIS-II 8089 CROSS-ASSEMBLER VER 1.0  ',0DH:0AH,0DH:0AH,
          'LOC  08J                LINE',0DH,0AH,0DH,0AH);
 381   2                                    INICIALIZACION=TRUE:
 382   2                                    INICIALIZACION1=T RUE;
 383   2                                    ERRORFLAG=FALSE;
 384   2                                     CALL CLOSE(AFTN1,.STS);
 385   2                                    IF STS<>0 THEN CALL ERRFATAL;
 387   2                               CALL OPEN(.AFTN1,.NFICHERO(1),1,0,.STS);
 388   2                               CALL OPEN(.AFTN2,.NFICHEX(1),2,0,.STS);
 389   2                              CALL OPEN(.AFTN3,.NFICHLS T(1),2,0,.STS);
 390   2                              IF STS<>0 THEN CALL ERRFATAL:
 392   2                               DO A=0 TO 23;BUFFERLST(A)=' ';END;
 395   2                        CALL WRITE(AFTN3,.MEN$ISIS,S IZE(MEN$ISIS),.STS);
 396   2                              END REINICIA;
```

```
                $EJECT

397   1                 FINALIZAR:
                    /* ESTA SUBRUTINA GENERA EL ULTIMO MENSAJE Y TERMINA LA
                      EJECUCION DEL EN SAMBLADOR 8089 */
                                  PROCEDURE PUBLIC;
398   2                              DECLARE ULTMENS(*) BYTE DATA
                ('ENSAMBLADO TERMINADO -NUMERO ERRORES ');
 399   2                             DECLARE BUFFER(4) BYTE ;
400   2                             DECLARE PTS  BYTE;
401   2                            CALL WRITE(AFTN3,.(0DH,0AH),2,.STS);
402   2                             CALL WRITE (AFTN3,.('TABLA DE SIMBOLOS',0DH,0AH),19,.STS);
403   2                            I F LONGTABLA<>0 THEN DO;
405   3                             DO PTS=0 TO LONGTABLA-1;
406   4                               CALL WRITE(AFTN3,.TABSYM(PTS).LETRAS(0),7,.STS);
407   4                               CALL WRITE(AFTN3,.('    '),4,.STS);
408   4                              BUFFER(0)=ASCII(SHR(HIGH(TABSYM(PTS).VA LUE),4) AND 0FH);
409   4                              BUFFER(1)=ASCII(HIGH(TABSYM(PTS).VALUE) AND 0FH);
 410   4                              BUFFER(2)=ASCII(SHR(LOW(TABSYM(PTS).VALUE),4) AND 0FH);
411   4                              BUFFER(3)=ASCII(LOW(TABSYM(PTS).VALUE) AND 0FH);
412   4                              CALL WRITE(AFTN3,.BUFFER(0),SIZE(BUFFER),.STS);
413   4                             CALL WRITE(AFTN3,.('    ') ,4,.STS);
414   4                             IF (TABSYM(PTS).TIPO AND 80H)=80H  THEN
415   4                              CALL WRITE(AFTN3,.('LAB',0DH,0AH),5,.STS);
416   4                             ELS E CALL WRITE(AFTN3,.('SYM',0DH,0AH),5,.STS);
417   4                             END /*DO*/;
4 18   3                           END /*DO*/;
419   2                           CALL HEXCOD (0,0,0,0FFH);
 420   2                            CALL WRITE (AFTN3,.ULTMENS(0),SIZE(ULTMENS),.STS);
421   2                            CALL WRITE (0,.ULTMENS(0),SIZE(ULTMENS),.STS);
422   2                           BUFFER(0 )=ASCII(SHR(NERROR,4) AND 0FH);
423   2                           BUFFER(1)=ASCII(NERROR AND 0FH);
 424   2                            BUFFER(2)=0DH;
425   2                           BUFFER(3)=0AH;
 426   2                            CALL WRITE (AFTN3,.BUFFER(0),SIZE(BUFFER),.STS);
427   2                            CALL WRITE (0,.BUFFER(0),SIZE(BUFFER),.STS);
428   2                           CALL EXIT;
429   2                           END FINALIZAR;
```

```
                $EJECT

 430   1                  TRATA$ST:
                        /* DADA UNA STRING, DEVUELVE EL VALOR DE UN SIMBOLO O UN
                          DATO INMEDIATO */
                                    PROCEDURE (PUNTERO) ADDRESS PUBLIC;
 431   2                                DECLARE PUNTERO ADDRESS;
 432   2                                IF ALFA(PUNTERO) THEN DO;
 434   3                                       IF SYM$IN$TAB(PUNTERO,LONG$SYM(PUNTERO))THEN DO;
 436   4                                                                FOUND=TRUE;
 437   4                                                               RETURN .TABSYM(SYMBUX).VALUE(0);
                                   END /*IF*/;
 439   3                                 END /*IF*/;
 440   2                                IF NUM(PUNTERO) THEN DO;
 442    3                                FOUND=TRUE;
 443   3                                RETURN GETADDRESS(PUNTERO);
  444   3                                END /*DO*/;
 445   2                                RETURN 0;
  446   2                                END TRATA$ST;
 447   1       END SUBRUTINAS89;



MODULE INFORMATION:

     CODE AREA SIZE     = 0D65H    3429D
     VARIABLE AREA SIZE = 0BD5H    3029D
     MAXIMUM STACK SIZE = 000AH      10D
      596 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
 4
```

```
ISIS-II MCS-86 LINKER, V1.3, INVOKED BY:
LINK86 VER3.OBJ,MDSLIB.OBJ TO VER3.LNK
 LINK MAP FOR VER3.LNK(MONITOR)

LOGICAL SEGMENTS INCLUDED:
LENGTH ADDRESS  SEGME NT          CLASS
 0F0FH  ------  MONITOR+CODE     CODE
 028CH  ------  MONITOR+DATA      DATA
 0068H  ------  STACK            STACK
 0000H  ------  MEMORY           ME MORY
 0304H  ------  LIBRERIAS+CODE   CODE
 0000H  ------  LIBRERIAS+DATA   DATA

INPUT MODULES INCLUDED:
VER3.OBJ(MONITOR)
MDSLIB.OBJ (LIBRERIAS)
```

PLANOS



NOTAS:

DADA LA COMPATIBILIDAD HARDWARE, SE PUEDE EMPLEAR LOS PROCESADORES 8Ø88 - 8Ø86 y 8089. SOLO CAMBIAN LAS ASIGNACIONES DE LAS LINEAS DEL CONECTOR C3 QUE CAPACITAN UNA GRAN FLEXIBILIDAD.
EN EL SISTEMA DESARROLLADO, LAS LINAS TIENEN LA SIGUIENTE ASIGNACION (PROCESADOR USADO 8Ø88)

17 - NMI : A INTERRUPTOR - PULSADOR.
18 - INTR : A MASA.
33 - MN-$\overline{MX}$: A MASA. (MODO MAXIMO).
32 - $\overline{RD}$: NO CONECTADO.
31 - $\overline{RQ}$-$\overline{GT}$Ø: A Vcc.
3Ø - $\overline{RQ}$-$\overline{GT}$1 : A Vcc. (NO COPROCESADORES).
25 - ΦSØ : NO CONECTADO.
24 - ΦS1 : NO CONECTADO.
23 - TEST/ : A INTERRUPTOR - PULSADOR. SE EMPLEA PARA PERMITIR INICIALIZACION MDS-221 (HOST).

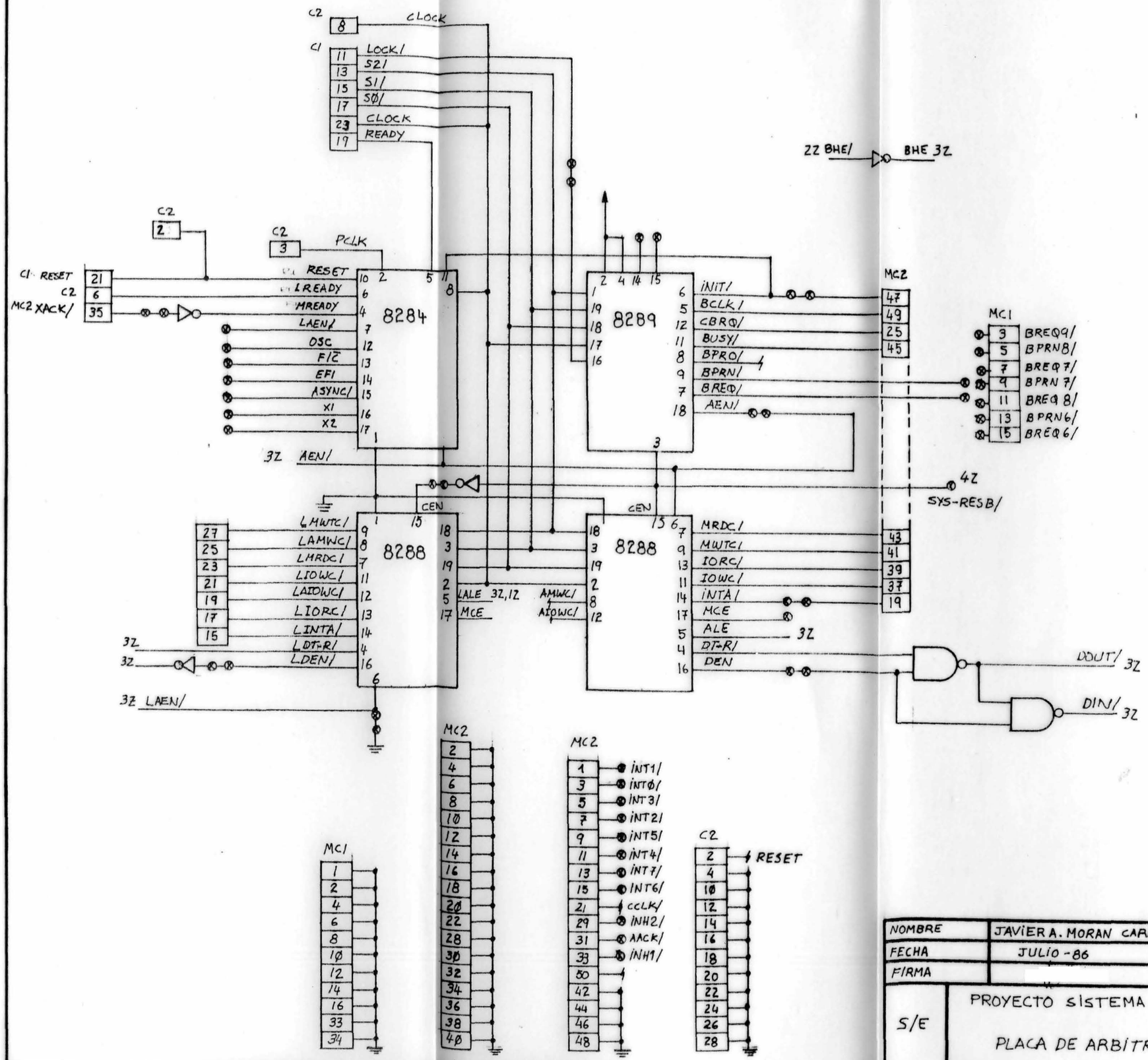
SE DESACOPLA Vcc CON UN CONDENSADOR DE 10μF.

LA SALIDA 2 DEL CONECTOR C3 SE EMPLEA PARA LA GENERACION DE SYS-$\overline{RES8}$.

NO SE MUESTRAN LAS OTRAS ASIGNACIONES DEL CONECTOR C3, TOTALMENTE DEFINIBLES PARA OTRAS APLICACIONES.

SE PUEDE EMPLEAR UN PROCESADOR 8Ø186 o 8Ø188 EN ESTA PLACA. LAS ASIGNACIONES DEL CONECTOR C1, QUE VAN A LA PLACA DE ARBITRAJE Y CONTROL, PERMANECERAN INVARIABLES.

NOMBRE	JAVIER A. MORAN CARRERA	PETICIONARIO:
FECHA	JULIO-86	E. U. POLITECNICA. TELECOMUNICACION. U.P.L.P.
FIRMA		
S/E	PROYECTO SISTEMA MULTIMICROPROCESADOR 88 PLACA DE CPU Y LOGICA DE SELECCION DE BUS.	Nº 1 DE 4



NOTAS:
EL SIMBOLO "●" INDICA PUNTO DE CONEXION PARA HACER MEDIDAS O VARIAR LAS CARACTERISTICAS MEDIANTE "JUMPERS".

LOS CONECTORES MC1 Y MC2 CONECTAN EL PROTOTIPO AL "HOST" MDS-221, POR EL BUS DE EXPANSION (MC2≡J16, MC1≡J14).

EL CONECTOR C1 INTERCONECTA ESTA PACA CON LA DE CPU Y LOGICA DE SELECCION DE BUS. EL C2 CONECTA CON LA PLACA DE MEMORIA LOCAL.

PARA EL SOPORTE DE LOS PROCESADORES 80186 Y 80188, SE DEBERA ELIMINAR EL 8284 E INCORPORAR LOGICA PARA LA COMPOSICION DE LOS READY LOCAL Y DEL SISTEMA.

LOS INTEGRADOS IC12, IC10, IC15 DEBEN SER ELIMINADOS CUANDO SE USEN CPU(S) DE 8 BITS.

ACTUALMENTE, LA PRIORIDAD DE ESTE SISTEMA ES LA MAXIMA PERMITIDA (BREQ/≡BREQ9/, BPRN/≡MASA)

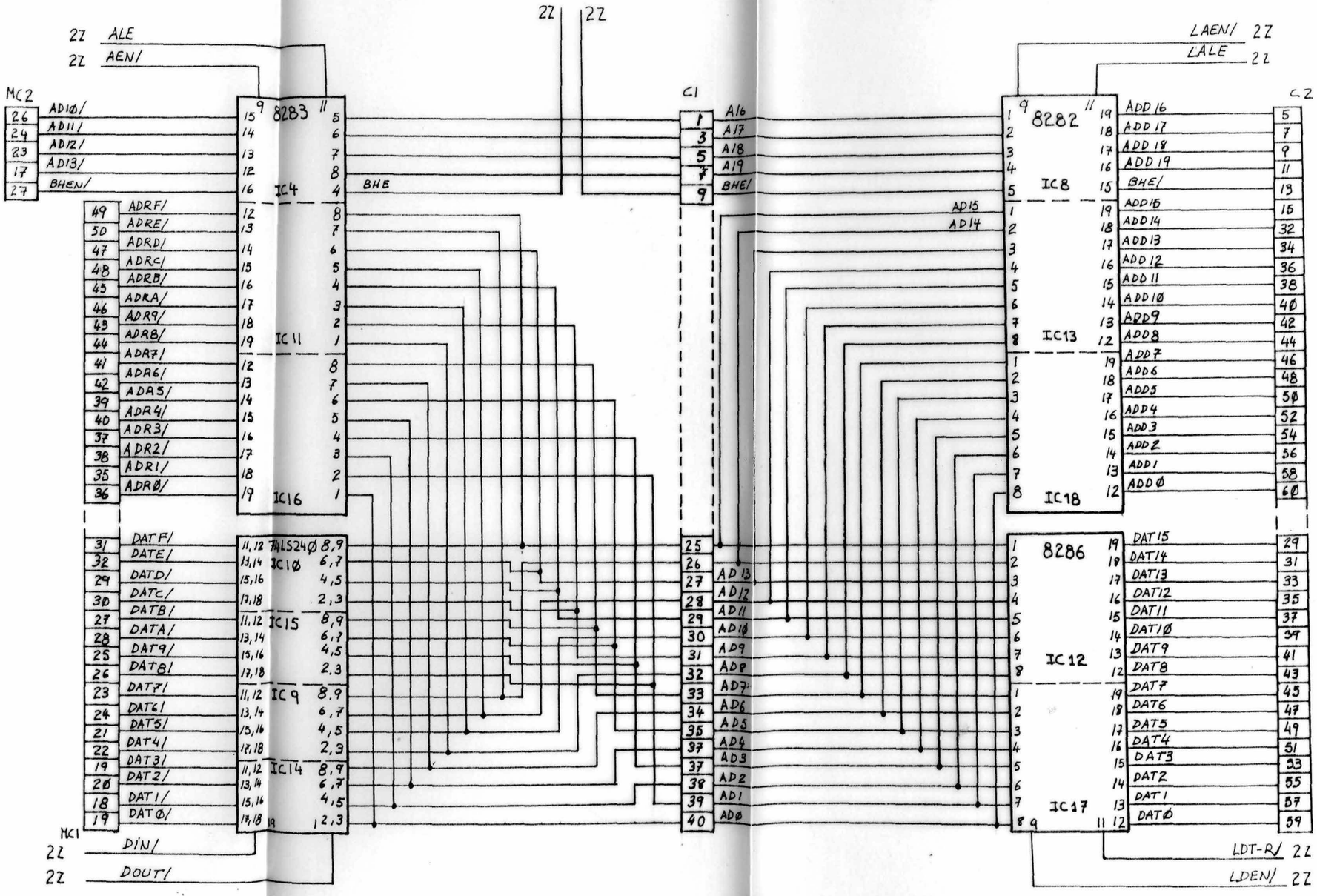
SE EMPLEA OPCION CRISTAL A 6.144 MHZ.

SE DESACOPLA V_{cc} CON UN CONDENSADOR DE 10μF.

PARA EVITAR "STARVATION" EN EL MDS-221, ANYRQST≡ACTIVO, CBRQ/≡ACTIVO

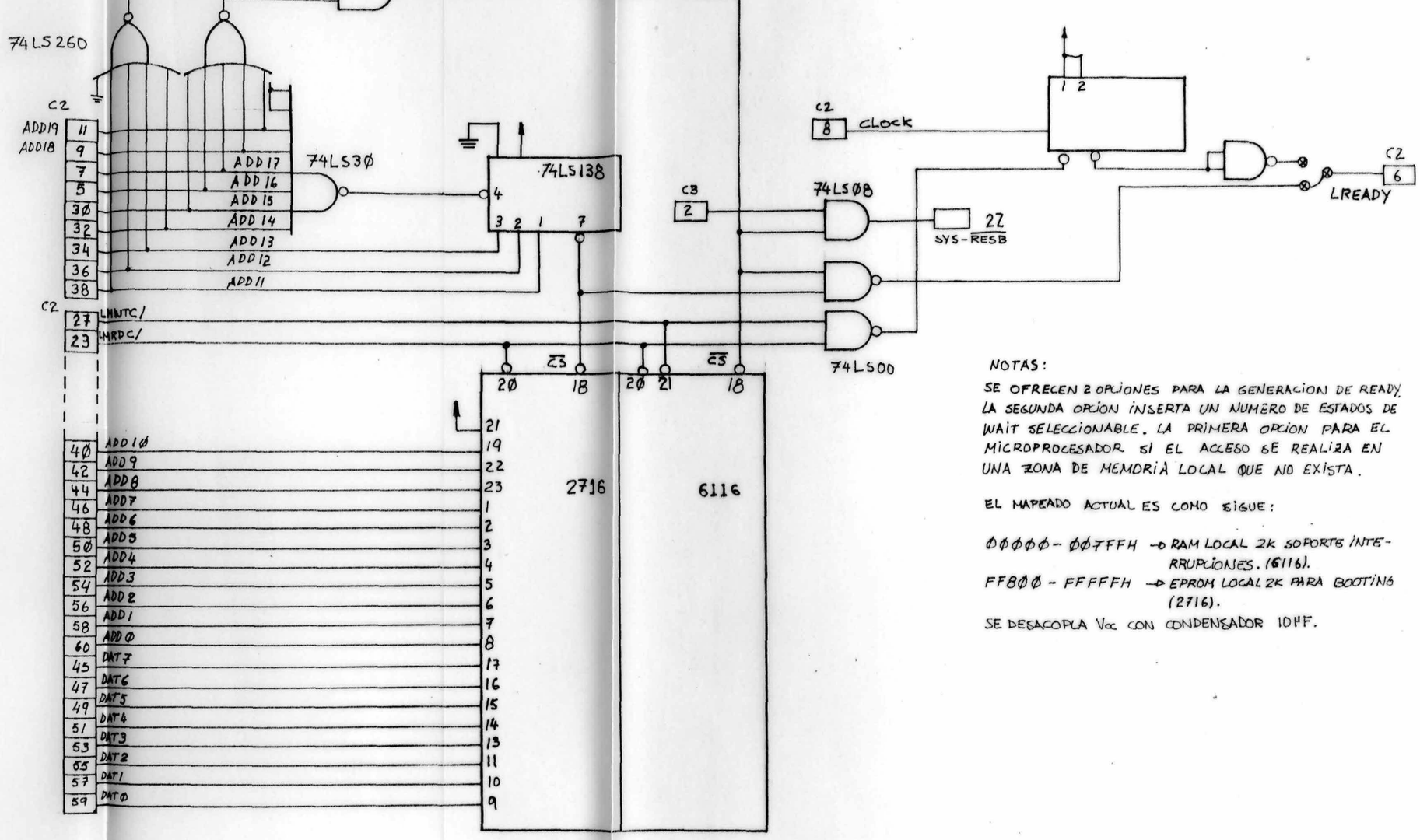
LAEN/ A MASA (ACTIVO).

NOMBRE	JAVIER A. MORAN CARRERA	PETICIONARIO:
FECHA	JULIO-86	
FIRMA		E.U. POLITECNICA. TELECOMUNICACION. U.P.L.P.
S/E	PROYECTO SISTEMA MULTIMICROPROCESADOR 88 PLACA DE ARBITRAJE Y CONTROL DE BUS. I	Nº 2 DE 4



NOMBRE	JAVIER A. MORAN CARRERA	PETICIONARIO:
FECHA	JULIO - 86	
FIRMA		E.U. POLITECNICA. TELECOMUNICACION. U.P.L.P.

S/E	PROYECTO SISTEMA MULTIMICROPROCESADOR88 PLACA DE ARBITRAJE Y CONTROL DE BUS II	Nº 3 DE 4



NOTAS:

SE OFRECEN 2 OPCIONES PARA LA GENERACION DE READY. LA SEGUNDA OPCION INSERTA UN NUMERO DE ESTADOS DE WAIT SELECCIONABLE. LA PRIMERA OPCION PARA EL MICROPROCESADOR SI EL ACCESO SE REALIZA EN UNA ZONA DE MEMORIA LOCAL QUE NO EXISTA.

EL MAPEADO ACTUAL ES COMO SIGUE:

00000 - 007FFH → RAM LOCAL 2K SOPORTE INTERRUPCIONES. (6116).

FF800 - FFFFFH → EPROM LOCAL 2K PARA BOOTING (2716).

SE DESACOPLA Vcc CON CONDENSADOR 10nF.

NOMBRE	JAVIER A. MORAN CARRERA	PETICIONARIO:
FECHA	JULIO - 86	
FIRMA		E.U. POLITECNICA. TELECOMUNICACION. U.P.L.P.

S/E	PROYECTO SISTEMA MULTIMICROPROCESADOR 88 PLACA DE MEMORIA LOCAL	Nº 4 DE 4