UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
COLEGIO DE CIENCIAS Y HUMANIDADES

INSTITUTO DE INVESTIGACIONES EN
MATEMATICAS APLICADAS Y EN SISTEMAS

Diseño conceptual de la arquitectura de un procesador para desarrollo de programación de sistemas

TESIS que para obtener el grado de
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN
presenta

Armando José Rivero Molina

Abril de 1986

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE.

PROLOGO

El objetivo de este trabajo es presentar la arquitectura de un procesador cuyas características principales son la simplicidad de su diseño y la versatilidad en su funcionamiento.

Entre las características de este procesador se encuentran las siguientes:

- 16 Megabytes de espacio de direccionamiento lógico.
- Mapeo de memoria por paginación.
- Manejo de memoria virtual.
- Manejo de lenguajes de alto nivel.
- 9 modos de direccionamiento para accesar cualquier operando.
- Dos modos de proceso (supervisor y usuario).
- Modo de rastreo.

Como apoyo al diseño de la arquitectura del procesador, se realiza un programa simulador del funcionamiento de éste.

Muchos de los cursos relacionados con sistemas de computación tales como Organización de Computadoras, Programación de Sistemas, Compiladores, Sistemas Operativos, etc., necesitan para su desarrollo contar con alguna computadora para que en ella se puedan basar las descripciones que deben contar con hardware de apoyo.

Dada la simplicidad del diseño de su arquitectura y su relativo poder, no se descarta la posibilidad de que el procesador/simulador, objetivo de este trabajo, pueda servir de base para desarrollos y proyectos en las áreas de Computación antes mencionadas.

Para llevar a cabo el diseño del procesador fue necesario revisar las arquitecturas de algunos de los procesadores que se utilizan actualmente. Se consideró que el describir el resultado de esta revisión podría ser de utilidad en un curso introductorio de Arquitectura de Computadoras o de Ensambladores.

El trabajo consta de tres capítulos. El primero está dividido en tres secciones. En la primera sección se trata de dar una definición, lo más completa posible, de lo que es la

arquitectura de una computadora. En la segunda sección se hace un recorrido histórico por las diferentes etapas de evolución por las que ha atravesado la Arquitectura de Computadoras. En la tercera sección se da una descripción que se podría llamar "Introducción a la Arquitectura de Computadoras", la cual trata de abarcar los temas básicos de esta materia. Para esto, se recurre a la descripción de algunas características de procesadores que están en uso actualmente. Se hace énfasis especial en los microprocesadores, dada su popularidad.

El segundo capítulo es el núcleo de este trabajo. En él se describe la arquitectura del procesador que diseñé. El capítulo está dividido en secciones, cada una de las cuales abarca un aspecto de la arquitectura: conjunto de instrucciones, formato de éstas, modos de direccionamiento, etc.

Por último, en el tercer capítulo se hace una descripción de las estructuras de datos y las funciones que forman el programa simulador. También se muestran algunas de las pruebas que se efectuaron para asegurar su buen funcionamiento.

Deseo manifestar mi agradecimiento a las personas que hicieron posible este trabajo, y especialmente al M. en C. Guillermo Levine Gutiérrez.

1.1. INTRODUCCION.

Los componentes básicos más importantes en las modernas computadoras digitales son las llamadas compuertas lógicas. Para que una computadora pueda cumplir con su finalidad, que es la de procesar información, es necesario analizar y manipular de manera conveniente a estos componentes.

Una manera estructurada de analizar esta organización es visualizando a la computadora como una jerarquía de niveles, en donde cada nivel, para operar correctamente, depende del nivel inmediato inferior y es independiente de los niveles superiores. El nivel más bajo depende de los componentes básicos.

Cada uno de los niveles, a los cuales se les denomina también como máquinas virtuales, requiere de un diseño particular y maneja cierto tipo de datos. También cada nivel, excepto el inferior, posee un conjunto de instrucciones que pueden ser utilizadas por el programador de la computadora.

El programador que trabaja con alguno de los niveles, necesita conocer el lenguaje de esta máquina virtual, pero no necesita conocer los lenguajes de los demás niveles.

La idea de las máquinas virtuales viene del hecho de que diseñar una computadora capaz de ejecutar un lenguaje de alto nivel utilizando directamente compuertas lógicas ha resultado siempre muy costoso y complicado (situación que está cambiando actualmente gracias a los circuitos VLSI, o sea, circuitos con muy alto grado de integración), por lo cual casi siempre se ha hecho necesario, cuando menos, el traducir el conjunto de instrucciones del lenguaje de alto nivel a un conjunto de instrucciones en código de ensamblador, el que a su vez se traduce luego al lenguaje de la máquina. Cada traductor está en un nivel diferente, esto es, cada conjunto de instrucciones define un nivel.

Tanenbaum [1984] define seis niveles o máquinas virtuales:

- nivel de lógica digital
- nivel de microprogramación
- nivel del lenguaje de máquina convencional
- nivel del sistema operativo
- nivel del lenguaje ensamblador

- nivel de los lenguajes de alto nivel.

El nivel de lógica digital es el que trabaja con compuertas lógicas, las cuales ejecutan simples funciones (tales como AND, OR, NAND, NOR, XOR, NOT) sobre señales eléctricas que pueden tener uno de dos valores lógicos (el 0, cuando la señal está entre 0 y 1 volts y el 1 cuando la señal está entre 2 y 5 volts). Los circuitos formados por estas compuertas son de dos tipos: combinacionales y secuenciales, y permiten la construcción de registros, contadores, multiplexores, flip-flops, decodificadores, sumadores, multiplicadores, unidades de control, memorias, etc.

Se podría decir que existe un nivel inferior al de la lógica digital. Este nivel es el del diseño de las compuertas lógicas, las cuales están construidas por medio de transistores, pero este nivel es materia de estudio de la Ingeniería Eléctrica.

En el segundo nivel, el de microprogramación, es donde aparece el concepto de "programa" o sea de secuencia de instrucciones ejecutables. Este es el nivel del verdadero lenguaje máquina de la computadora. Los dispositivos del nivel inferior ejecutan estas instrucciones. Conjuntos de microinstrucciones forman los llamados microprogramas. Los microprogramas se utilizan para interpretar instrucciones del nivel superior (éste es, el de lenguaje máquina convencional) y a veces son llamados emuladores.

El nivel de microprogramación no existe en todas las computadoras; de hecho empezó a aparecer en las máquinas de la Tercera Generación. Las instrucciones en lenguaje máquina convencional son, en las computadoras con este nivel, interpretadas por microprogramas, porque esto permite que las operaciones que se ejecutan cuando el programa lo requiere, sean diseñadas de una manera más fácil y flexible. Es más, la presencia de este nivel permite que una misma computadora pueda ejecutar conjuntos de instrucciones de diferentes lenguajes máquina convencionales con solo cambiar al emulador. Hay que mencionar que esto no se puede lograr en todas las máquinas microprogramables (computadoras que poseen este nivel), ya que algunas veces los emuladores están grabados en ROM (Read Only Memory) y no son intercambiables.

La existencia de este nivel en las computadoras tiene algunas inconveniencias. Una es que se requieren circuitos adicionales en la máquina. Otra es que la ejecución de las instrucciones del lenguaje máquina es más lenta. Esto último debido a la interpretación extra que existe, y a la dificultad de incluir sistemas de tubería (concepto que se explica más adelante).

El siguiente nivel, que es el del lenguaje máquina convencional, es el que la gente que trabaja en Programación de Sistemas (la gente que hace ensambladores, compiladores, simuladores, etc.) debe conocer. En el caso de las computadoras sin nivel de microprogramación, el lenguaje de este nivel viene a ser el verdadero lenguaje de ellas, o sea, el lenguaje que es ejecutado por el hardware (o lo que es lo mismo, por los dispositivos construidos con las compuertas lógicas). Los lenguajes que se interpretan en este nivel están incluidos en el grupo de los llamados lenguajes de bajo nivel.

El nivel siguiente, que es el del sistema operativo, es llamado así porque los programas de esta máquina virtual son soportados por el sistema operativo, el cual es un conjunto de programas cuya función es administrar los recursos de la computadora. Muchas instrucciones en este nivel son las mismas que algunas de las del lenguaje máquina convencional, por lo tanto son interpretadas directamente por el emulador (en caso de existir éste, o bien, por el hardware), pero existen algunas adicionales que permiten que se puedan explotar mejor las facilidades que tiene la computadora y éstas son interpretadas por el sistema operativo.

En el siguiente nivel tenemos a los programas en lenguaje ensamblador. Los programas en este nivel son traducidos para su interpretación por medio de un programa llamado ensamblador. Este nivel provee a los programadores de una manera más cómoda de escribir programas para los niveles inferiores. Un programa ensamblador permite al programador evitar escribir los programas en lenguajes numéricos. También le da algunas otras facilidades, tales como etiquetas, las cuales son utilizadas en lugar de direcciones numéricas, o bien, macroinstrucciones, las cuales permiten al programador no tener que reescribir código.

Por último, en el nivel superior se manejan los programas de aplicación, o sea los que permiten dar a la computadora un uso específico (comercial, científico), de manera relativamente sencilla (con este nivel, hasta los niños las pueden utilizar). El soporte de este nivel son los compiladores. Estos, al igual que los ensambladores, traducen programas de un lenguaje a otro. La diferencia entre ellos es que los ensambladores siempre traducen los programas a algún lenguaje numérico, mientras que los compiladores pueden también (y lo hacen con mucha frecuencia) traducir a algún otro lenguaje simbólico.

De la mayoría de los niveles se podrían considerar algunas características:

El siguiente nivel, que es el del lenguaje máquina convencional, es el que la gente que trabaja en Programación de Sistemas (la gente que hace ensambladores, compiladores, simuladores, etc.) debe conocer. En el caso de las computadoras sin nivel de microprogramación, el lenguaje de este nivel viene a ser el verdadero lenguaje de ellas, o sea, el lenguaje que es ejecutado por el hardware (o lo que es lo mismo, por los dispositivos construidos con las compuertas lógicas). Los lenguajes que se interpretan en este nivel están incluidos en el grupo de los llamados lenguajes de bajo nivel.

El nivel siguiente, que es el del sistema operativo, es llamado así porque los programas de esta máquina virtual son soportados por el sistema operativo, el cual es un conjunto de programas cuya función es administrar los recursos de la computadora. Muchas instrucciones en este nivel son las mismas que algunas de las del lenguaje máquina convencional, por lo tanto son interpretadas directamente por el emulador (en caso de existir éste, o bien, por el hardware), pero existen algunas adicionales que permiten que se puedan explotar mejor las facilidades que tiene la computadora y éstas son interpretadas por el sistema operativo.

En el siguiente nivel tenemos a los programas en lenguaje ensamblador. Los programas en este nivel son traducidos para su interpretación por medio de un programa llamado ensamblador. Este nivel provee a los programadores de una manera más cómoda de escribir programas para los niveles inferiores. Un programa ensamblador permite al programador evitar escribir los programas en lenguajes numéricos. También le da algunas otras facilidades, tales como etiquetas, las cuales son utilizadas en lugar de direcciones numéricas, o bien, macroinstrucciones, las cuales permiten al programador no tener que reescribir código.

Por último, en el nivel superior se manejan los programas de aplicación, o sea los que permiten dar a la computadora un uso específico (comercial, científico), de manera relativamente sencilla (con este nivel, hasta los niños las pueden utilizar). El soporte de este nivel son los compiladores. Estos, al igual que los ensambladores, traducen programas de un lenguaje a otro. La diferencia entre ellos es que los ensambladores siempre traducen los programas a algún lenguaje numérico, mientras que los compiladores pueden también (y lo hacen con mucha frecuencia) traducir a algún otro lenguaje simbólico.

De la mayoría de los niveles se podrían considerar algunas características:

- el diseño del nivel y el método de implementación
- las instrucciones del lenguaje de la máquina virtual y el formato de éstas
- el tipo de datos que se manejan
- la organización de la memoria
- los tipos de direccionamiento
- los registros disponibles .

Al estudio de estas características se le llama algunas veces Arquitectura de Computadoras.

Esto quiere decir que la arquitectura de una computadora describe las características de ésta tal como son vistas por el programador. Más precisamente, la arquitectura está expresada en la especificación detallada de la interconexión entre el hardware y el software.

Se dice que dos computadoras tienen la misma arquitectura si los programas que fueron escritos para una, pueden ejecutarse en la otra sin necesidad de hacer cambios. Esto sin importar el diseño o la tecnología con que haya sido construido el hardware.

Entonces la arquitectura describe que es lo que la computadora hace y no como lo hace.

Cuando Amdahl, Blaauw y Brooks [1964] anunciaron la computadora IBM sistema 360, definieron Arquitectura de Computadoras como el conjunto de atributos de una computadora como son vistos por la persona que hace software para ella en lenguaje máquina, esto es en el nivel de lenguaje máquina convencional. La definición incluye el conjunto de instrucciones, el formato de éstas, los códigos de operación, los modos de direccionamiento y todos los registros y localidades de memoria que pueden ser directamente manipuladas o probadas por un programa en lenguaje máquina.

En este trabajo nos referiremos solamente al nivel de lenguaje máquina convencional, es decir, nos encargaremos de describir la organización del nivel cuyo lenguaje es soportado por la microprogramación, o en su caso, directamente por el hardware.

1.2. UN POCO DE HISTORIA.

Antes de empezar con la historia, hay que aclarar que no existe un acuerdo común entre la gente de Computación acerca de las características exactas de cada generación de computadoras. Tampoco hay acuerdo acerca de las fechas que delimitan cada generación.

1.2.1. LA PRIMERA GENERACION.

La primera computadora de propósito general fue probablemente la ENIAC (Electronic Numerical Integrator And Calculator), la cual fue construida bajo la dirección de J.P. Eckert y de J. Maulchy en la Universidad de Pennsylvania. El proyecto empezó en 1943 y terminó en 1946 (por cierto, esta computadora pesaba treinta toneladas). La ENIAC tenía una memoria de trabajo que constaba de 20 acumuladores, cada uno de los cuales podía acomodar un número decimal con signo de veinte dígitos. Estos acumuladores combinaban las funciones de almacenamiento y adición y substracción. Unidades adicionales tenían que ser incluidas para hacer posibles la multiplicación, la división y la extracción de raíz cuadrada. Esta computadora era programada manipulando conmutadores y conectando y desconectando cables. En la ENIAC los programas y los datos estaban en memorias separadas. El meter programas a la máquina o bien alterarlos era una tarea sumamente tediosa.

Uno de los consultores del proyecto ENIAC, el matemático nacido en Hungría John von Neumann (1903-1957) propuso en 1945 la idea de almacenar programas y datos en la misma memoria. Esto facilitaría el proceso de programación de una computadora. Además haría posible que un programa modificara sus propias instrucciones. A esta idea se le llamó concepto del programa almacenado. Este concepto fue utilizado por primera vez en la computadora EDVAC (Electronic Discrete Variable Computer), la cual llegó a ser operacional en 1951.

EDVAC fue la primera computadora que trabajó con representación numérica binaria, en lugar de decimal. Además su capacidad de almacenamiento era bastante grande: la memoria principal, construida con líneas de retardo de mercurio, era de 1024 palabras, y la memoria secundaria, hecha de cables magnéticos, tenía capacidad de 20K palabras.

En 1946 von Neumann y su grupo de investigadores, en el Princenton Institute of Advanced Studies, empezaron el diseño de una nueva computadora que utilizaría el concepto de programa almacenado, la computadora IAS. Entre las características principales de esta máquina estaban las siguientes: utilizaba una memoria de acceso directo construida a base de tubos de rayos catódicos la cual permitía que una palabra de memoria fuera accesada en una operación. A diferencia de la EDVAC, se emplearon circuitos binarios paralelos para su construcción. Además cada instrucción de programa contenía solamente al operador y a una dirección de memoria.

En la EDVAC cada instrucción contenía un campo en donde se especificaba la siguiente instrucción a ejecutarse. En la IAS, esto no tenía que ser especificado. Las instrucciones se ejecutaban secuencialmente excepto si existía en el programa alguna instrucción de bifurcación.

La Unidad Central de Proceso (CPU) de la IAS contenía un conjunto de registros de alta velocidad que se usaban como almacenamiento temporal de instrucciones, direcciones de memoria y datos. El procesamiento de datos se hacía por medio de los circuitos aritméticos y lógicos. Los circuitos de control decodificaban las instrucciones, enrutaban la información correctamente a través del sistema y proveían señales de reloj para todos los procesos. Un reloj sincronizaba la operación del sistema.

La memoria se utilizaba para almacenar programas y datos. Una palabra de información era de 40 bits y podía almacenar dos instrucciones. En cada instrucción, 8 bits eran para el código de operación y 12 bits eran para la dirección. La transferencia de palabras se podía llevar a cabo entre el registro de datos (DR) de la CPU (era de 40 bits) y el contenido de cualquier localidad de memoria. La dirección de esta localidad se especificaba en el registro de dirección (AR) de 12 bits de tamaño. El registro de datos podía ser utilizado para almacenar operandos durante la ejecución de una instrucción. Habían dos registros de almacenamiento temporal de operandos y resultados. Uno era el acumulador (AC) y el otro era el registro del multiplicador y del cociente (MQ). Dos instrucciones se obtenían simultaneamente de la memoria y se transferían a la unidad de control de programa. La instrucción que no iba a ser ejecutada en ese momento se colocaba en el registro buffer de instrucciones (IBR). El código de operación de la otra instrucción se colocaba en el registro de instrucción (IR) donde se decodificaba. La dirección de la instrucción ejecutándose se transfería al registro de dirección (AR). Otro registro, llamado el registro de dirección de instrucción o contador de programa (PC) se usaba para almacenar la dirección de la siguiente instrucción a ejecutarse.

La IAS permitía cinco tipos diferentes de instrucción:

- De Transferencia de Datos. Por ejemplo: mover el negativo del valor absoluto del contenido de la dirección de memoria X al acumulador.

- De Bifurcación Incondicional. Por ejemplo: ejecutar la instrucción del lado derecho de la que se encuentra en la localidad de memoria con dirección X.

- De Bifurcación Condicional. Por ejemplo: si el número que está en el acumulador es positivo o cero, entonces ejecutar la instrucción que está al lado izquierdo en la localidad de memoria con dirección X.

- Aritméticas. Por ejemplo: dividir el contenido del acumulador entre el contenido de la dirección de memoria X. Poner el resultado en MQ y el residuo en AC.

- De Modificación de Dirección. Por ejemplo: remplazar la dirección de la instrucción que está en la dirección de memoria X por los últimos 12 bits del acumulador.

La computadora IAS puede ser considerada como el prototipo de casi todas las computadoras de propósito general que le han sucedido. Dado que muchas de las características de la computadora IAS se debieron a las aportaciones de John von Neumann, a las máquinas construidas con esa misma estructura se les llaman computadoras con arquitectura de von Neumann.

En 1951, la Eckert Maulchy Computer Corporation (de los directores del proyecto ENIAC) sacaron al mercado la primera computadora comercial, la UNIVAC (UNIversal Automatic Computer).

A las máquinas que se fabricaron después de la ENIAC hasta finales de la década de los años cincuenta se les han llamado computadoras de Primera Generación. No hay acuerdo hasta que año llegó esta generación; para Denning fue 1949, para Hayes fue 1954 y para Bell y Newell fue 1957.

En la ENIAC solo existía un nivel en la arquitectura de la máquina: el nivel de la lógica digital. En subsecuentes computadoras apareció otro nivel: el del lenguaje máquina convencional. De todas maneras aun con este nivel, resultaba de lo menos placentero tener que programar con dígitos binarios. Más difícil aun era el tener que reconocer estos programas para modificarlos o darles mantenimiento.

A principios de los años cincuenta a alguien se le ocurrió fabricar un conjunto de instrucciones en caracteres alfanuméricos que tenían correspondecia uno a uno con el

conjunto de instrucciones en dígitos binarios. Un programa escrito en caracteres binarios podría traducir los caracteres alfanúmericos en dígitos binarios. A este programa se le llamó ensamblador y a este nuevo conjunto de instrucciones, lenguaje simbólico (ahora lenguaje ensamblador). Es con este nuevo lenguaje cuando las computadoras adquieren un nivel más en su arquitectura: el nivel del lenguaje ensamblador.

El componente electrónico principal de las computadoras de Primera Generación eran los tubos de vacío (también conocidos como bulbos electrónicos). La ENIAC contenía alrededor de 18,000, mientras que la EDVAC unos 4,000. Las últimas máquinas de esta generación podían ejecutar unas 10,000 operaciones por segundo.

Computadoras famosas de esta generación fueron, además de las ya mencionadas, la IBM 701 (construida en 1953, fue la primera de la International Business Machines), la Whirlwind I (del Massachusets Institute of Technology), la Bendix-G15, las UNIVAC 60, 90 y 1105, y la Burroughs 220.

1.2.2. LA SEGUNDA GENERACION.

A fines de la década de los cincuentas, los tubos de vacío empezaron a ser remplazados con los transistores. Así nace la Segunda Generación de computadoras. Estos pequeños componentes electrónicos fueron inventados en 1948 por Bell Telephone Company. La primera computadora transistorizada fue construida en el Massachusets Institute of Technology en 1953.

En esta generación de computadoras las memorias de tubos de rayos catódicos y de líneas de retraso de mercurio fueron remplazadas por las celdas de ferrita. Aunque la primera máquina con este tipo de memoria fue la Whirlwind I del M.I.T.

También en esta época se empezaron a construir computadoras que tenían procesadores de entrada y salida, los cuales se encargaban del manejo de los periféricos de entrada y salida. De esta manera el procesador central podría dedicar más tiempo a computar.

Los procesadores de entrada y salida (E/S) supervisan el flujo de información entre la memoria principal y los dispositivos de entrada y salida. Estos procesadores llevan a cabo esta

tarea ejecutando programas especiales compuestos de instrucciones de entrada y salida que se encuentran almacenados también en memoria principal. El procesador de E/S empieza la ejecución de un programa de E/S cuando la CPU le manda una instrucción de iniciación de operación. Usualmente esta instrucción contiene la dirección de la primera instrucción a ejecutarse en el programa de E/S. El procesador de E/S puede entonces ejecutar el programa sin tener algo que ver con la CPU. Sin embargo la CPU puede controlar las operaciones de E/S por medio de instrucciones con información sobre el estado de los procesadores de E/S. Estos pueden también avisar a la CPU si existen condiciones inusuales. Una de estas condiciones puede ser un error en la transmisión de datos. Otra condición es la terminación de la operación de E/S. Los procesadores de E/S llevan a cabo esta comunicación con la CPU por medio de señales de control llamadas interrupciones.

Los datos se transfieren entre la memoria principal y el procesador de E/S por palabras de información. Pero del procesador de E/S al dispositivo de E/S, la transferencia es por caracteres. La CPU y los procesadores de E/S generalmente utilizan un acceso común a la memoria principal. Este medio de acceso es la Unidad de Control de Memoria, la cual está en la CPU. La función de esta unidad de control es hacer la lista y ordenar cual de los procesadores que lo solicitan tendrá acceso a la memoria. Dado que las operaciones de E/S son lentas comparadas con la velocidad de la CPU, se espera que la mayoría de las solicitudes de acceso a memoria vengan de la CPU.

Otra nueva inovación en la arquitectura de esta nueva generación fue el uso de registros de índice los cuales facilitarían el acceso a la memoria de la computadora en un programa cualquiera.

Estas características aparecieron en la IBM-704. Asimismo esta computadora poseía hardware para poder ejecutar operaciones aritméticas de punto flotante, tecnología que también caracteriza a la Segunda Generación. La IBM-704 apareció en 1955 y todavía estaba construida con tubos de vacío.

Los diseñadores de esta época se encuentran con la necesidad de tener un programa de control que maneje el inicio y la terminación de tareas. Este programa era realmente un rudimentario sistema operativo.

Es la computadora Burroughs modelo 5000 la primera de la serie de máquinas de stack de esta compañía. Esta máquina apareció en 1963. Más adelante veremos con detalle el concepto de stack.

La computadora ATLAS, diseñada y construida en la Universidad de Manchester en 1961 fue una de las primeras máquinas con sistema operativo. Además fue la primera computadora que permitía al programador utilizar más memoria principal de la que en realidad tenía disponible. A esta facilidad fue a lo que se le llamo almacenamiento de un nivel (one level storage), y después, memoria virtual. Esta máquina fue también la primera en manejar el concepto de extracódigo, el cual permite que códigos de operación puedan ser utilizados para llamar subrutinas. Funciones complejas de uso común, tales como Seno(X), Coseno(X), Raíz-Cuadrada(X), etc., pueden ser escritas en un sistema operativo de manera que puedan ser accesibles para cualquier usuario.

Otro de los grandes avances de esta época fue la introducción de los lenguajes de alto nivel. Hasta ahora, la forma más utilizada de comunicación entre las computadoras y el hombre, es por medio de los lenguajes de alto nivel.

Fue un grupo de la compañía IBM, dirigido por John Backus entre 1954 y 1957, el que desarrolló el lenguaje Fortran (FORmula TRANslation). Entonces los compiladores empezaron a extenderse por el mundo de la computación. Para aplicaciones científicas (al igual que Fortran) fueron creados los lenguajes ALGOL (ALGOritmic Language) y APL (A Programming Language) poco después. En 1959 el grupo CODASYL (Conference On DAta SYstems Languages) dio las especificaciones de uno de los primeros lenguajes para proceso de datos y aplicaciones comerciales y administrativas: COBOL (COmmon Business Oriented Language).

De esta manera aparecen dos nuevos niveles en la arquitectura de las computadoras, el nivel del Sistema Operativo y el nivel de los lenguajes de alto nivel. Las computadoras de esta época ya podían alcanzar velocidades de ejecución del orden de las 100,000 operaciones por segundo.

Máquinas famosas de esta generación, que no hemos mencionado son: la IBM-7090 y la IBM-7094, la CDC-1604 y la CDC-3600 de la Control Data Corporation, la RCA-501 y la NCR-315.

1.2.3. LA TERCERA GENERACION.

La Tercera Generación de Computadoras hace su aparición entre

1965 y 1968. Una de las principales características de las arquitecturas de las computadoras de esta generación, es que los transistores empezaron a ser remplazados por los circuitos integrados. Estos circuitos, también llamados chips, son pequeñas piezas de Silicio que agrupan transistores para formar compuertas lógicas, las cuales a su vez, forman diversos dispositivos de la computadora. Generalmente, los chips usados en esta época eran de las clases SSI y MSI. Los circuitos SSI (Small Scale Integrated) contienen de una a diez compuertas lógicas y los circuitos MSI (Medium Scale Integrated) contienen de diez a cien compuertas. Hay que mencionar también que los chips empezaron a remplazar también a las memorias de ferrita. Estos chips redujeron considerablemente el costo y el tamaño de las computadoras. Esta es una de las situaciones que permitieron el nacimiento de una nueva clase de máquinas: las minicomputadoras.

Los orígenes de las minicomputadoras se remontan a 1961 cuando la Digital Equipment Corporation introdujo la computadora PDP-1 (Programable Data Procesor), la cual ocupaba un área de menos de diez metros cuadrados y podía ser operada por dos personas.

Luego, en el M.I.T. fue construida la LINC (Laboratory INstrument Computer), la cual, según Bell y McNamara, fue la primera computadora personal. Digital Equipment Corporation, guiada por la arquitectura de esta máquina construyó y empezó a vender en 1963, la PDP-5 y en el año de 1965, la famosa PDP-8.

Probablemente la computadora cuya arquitectura fue más interesante en esta generación, es la de la computadora IBM Sistema 360 con su gran variedad de modelos. Esta máquina fue después sustituida por el Sistema 370 el cual tenía básicamente la misma arquitectura.

La unidad básica de almacenamiento en la IBM 360 es el byte de ocho bits. Cada byte en memoria puede ser accesado directamente. Cuatro bytes forman una palabra. Los datos alfanuméricos pueden ser representados de una forma, mientras que los numéricos, al menos de cuatro. Estas son: formato decimal con zona, formato decimal empaquetado, formato binario de punto fijo y formato binario de punto flotante.

La Unidad Aritmética y Lógica de la CPU del Sistema 360 está dividida en tres partes que ejecutan funciones diferentes. Una parte ejecuta las funciones con datos numéricos de punto fijo, incluyendo aritmética con enteros y computaciones de direcciones efectivas (la dirección efectiva es la que utiliza la CPU para accesar los operandos de una instrucción en memoria). Otra parte se dedica a las operaciones de punto flotante. La última parte es la que se encarga de operaciones

de longitud variable, incluyendo aquellas de aritmética decimal y operaciones con caracteres y cadenas.

La CPU tiene 16 registros de propósito general que pueden ser utilizados para almacenar operandos y resultados y también como registros de índice. Estos registros son de 32 bits cada uno. Además, como ayuda a las operaciones de punto flotante existen cuatro registros de 64 bits cada uno.

Además de los registros de dirección, de instrucción, de datos y el PC que funcionan de igual manera que lo hacen en la computadora IAS, existe en la IBM 360 un registro de 64 bits que contiene cierta información denominada palabra de estado del programa o PSW. La PSW indica el estado del programa que se está ejecutando, las solicitudes de interrupción que le llegan a la CPU y la dirección de la próxima instrucción a ser ejecutada. Cuando llega una interrupción, la CPU almacena la PSW en memoria principal y obtiene de ésta una nueva PSW, la cual especifica cual es el programa a ser ejecutado para procesar la interrupción. Cuando este programa termina, entónces la CPU recupera la PSW del programa que estaba en acción antes de la interrupción.

La memoria en el Sistema 360 está dividida en bloques de 2K cada uno de los cuales tiene una clave de almacenamiento, la cual indica si en el bloque se permite lectura y/o escritura. Cada PSW contiene también una clave de acceso, de manera que si el programa propietario intenta hacer una operación en un determinado bloque, la clave de éste y la clave en la PSW deben ser iguales. De otra manera no se puede ejecutar la operación. A este esquema se le llama de protección de memoria. La PSW contiene también un campo que especifica en que estado de control se encuentra la CPU. El estado en el cual se ejecutan los programas del usuario se llama estado problema o de usuario. El otro estado es el de supervisor. En este estado se encuentra la CPU cuando está ejecutando alguna rutina del sistema operativo. Ciertas instrucciones solo se pueden ejecutar cuando la CPU está en este estado. Se llaman instrucciones privilegiadas y se utilizan por ejemplo, para modificar la PSW o para modificar claves para protección de memoria.

Existen cerca de 200 códigos de operación para la ejecución de instrucciones. El Sistema 360 posee instrucciones aritméticas, de transferencia, de bifurcación, y de entrada y salida, igual que las máquinas de Segunda Generación. Pero además existen instrucciones para mover bloques de información de un lugar a otro de la memoria. Otro tipo de instrucción en esta máquina es el utilizado para cambiar tipos de datos, por ejemplo, de binario con punto fijo a decimal empaquetado. La máquina permite también operaciones lógicas tales como el AND y el OR.

En el Sistema 360 se utilizan dos tipos de procesadores de entrada y salida. Estos son el Selector y el Multiplexor. Son llamados canales. El canal Multiplexor puede transmitir y recibir información de múltiples dispositivos de entrada y salida al mismo tiempo. Por eso se utiliza con dispositivos lentos tales como la impresora, de manera que el canal se use lo más posible. El canal Selector se conecta a un solo dispositivo de entrada y salida. Este dispositivo debe ser de alta velocidad, por ejemplo una unidad de disco magnético.

Como dijimos antes, la microprogramación da facilidad en la implementación del conjunto de instrucciones en lenguaje máquina que un procesador puede ejecutar. Este concepto se empezó a extender entre las computadoras cuando apareció en el modelo 30 del Sistema 360 de la IBM.

Otros modelos del Sistema 360 tenían tambien procesadores microprogramados, estos eran las máquinas pequeñas y no muy rápidas. Cabe mencionar que todos los modelos eran compatibles, esto es, que todos podían ejecutar el mismo conjunto de instrucciones.

Los sistemas 360 y 370 permitían instrucciones aritméticas binarias y decimales, además de varios tipos de datos, para que de esta manera pudieran ser utilizadas por igual para un extenso número de diferentes aplicaciones.

Algunos modelos de estas máquinas eran capaces de manejar multiprogramación y multiproceso. La multiprogramación y el multiproceso permiten que en una misma computadora se estén ejecutando concurrentemente varios programas. En multiprogramación, los varios programas comparten un procesador. Cuando hay multiproceso, pueden ser varios los procesadores que ejecutan los programas, o bien, los que ejecutan un solo programa. Para el caso en que hayan muchos programas, estos deben compartir la misma memoria. Como es difícil que todos quepan completos en ésta, algunos modelos del Sistema 370 utilizaban el concepto de memoria virtual, y una buena cantidad de técnicas de manejo de memoria, las cuales implican hardware adicional. En estas máquinas aparecieron los complicados sistemas operativos, los cuales tenían, entre sus muchas funciones, las de manejar la memoria.

Lo usual de aquella época era que cada máquina tuviera su propio sistema operativo. Multics, un famoso e inmenso sistema operativo que puede manejar multiprogramación, multiproceso, etc., fue el resultado de una investigación de Honeywell, Laboratorios Bell, y el proyecto MAC en el M.I.T. El sistema no tuvo mucho éxito debido a su gran complejidad.

Bell se separó del proyecto en 1969, y algunos de sus investigadores se iniciaron en el desarrollo de otro sistema

operativo llamado UNIX. Este sistema posee un lenguaje de comandos poderoso y simple, y un sistema de archivos independiente de dispositivos. En 1972 se empezaron a escribir los programas de UNIX en lenguaje C (lenguaje que fue diseñado originalmente para UNIX en la PDP-11). Esto hizo al software de UNIX más portable (capacidad del software de ser utilizado en diferentes computadoras) y claro de entender. Hasta la fecha, UNIX se sigue volviendo cada día más popular.

En la sección anterior comentamos la aparición de la B5000 de Burroughs. Tiempo después apareció una modificación de esta máquina, a la cual se le llamó B5500. En 1969 un avance importante se hizo sobre el sistema original y se llamó B6500, la cual fue nombrada después como la B6700.

Estas máquinas con arquitectura muy especial tiene como características intrínsecas en su diseño el manejo de memoria virtual y multiproceso. A las unidades que procesan se les llaman "jobs" (trabajos). Un "job" consiste de un algoritmo invariante en el tiempo , el cual es un conjunto de segmentos de código (o subrutinas), y de una estructura de datos que varía con el tiempo llamada registro de ejecución, la cual define el estado de ejecución del "job", incluyendo los valores para todas sus variables, además del subespacio o subespacios de memoria virtual que el procesador o los procesadores pueden accesar. El registro de ejecución incluye también la historia del flujo de control entre los bloques, los procedimientos y las tareas.

El procesador o los procesadores mantienen un par de apuntadores cada uno: el apuntador a instrucciones del algoritmo o IP (instruction pointer) y el apuntador al medio ambiente o EP (environment pointer), este último para referenciar a las porciones accesibles del registro de ejecución.

Debido a que los algoritmos están divididos en segmentos (la arquitectura de la B6700 está orientada a manejar ALGOL, el cual funciona a base de procedimientos, y cada procedimiento puede ser tomado como un segmento), el procesador debe accesar un diccionario de segmentos para obtener las direcciones de éstos. Normalmente solo los segmentos activos se encuentran en memoria principal (los demás están en memoria virtual).

El registro de ejecución de un "job" está formado por unas estructuras llamadas registros de activación. Cada segmento del algoritmo invariante genera un registro de activación. Estos registro son almacenados en un stack. De esta manera, cada vez que un nuevo segmento empieza a ser procesado, su registro de activación es metido al stack, y es ligado a los que ya estaban en él. El stack es una magnifica estructura para almacenar los registros, puesto que los segmentos pueden estar anidados.

Existe también en memoria otra estructura llamada "stack trunk" que contiene entre otras cosas los descriptores de las tablas y de los segmentos de las rutinas del sistema operativo.

De esta manera, el IP está formado por tres elementos. El primero identifica a la tabla que contiene la dirección del segmento en donde se encuentra la instrucción (el diccionario de segmentos o el "stack trunk"), el segundo identifica el segmento en terminos de su desplazamiento dentro de la tabla, el tercero identifica el desplazamiento relativo de la instrucción dentro del segmento.

El EP está formado por varios elementos. El primero es el apuntador al diccionario de segmentos, el segundo es el apuntador al "stack trunk", los demás son los apuntadores a los diferentes registros de activación del "job", los cuales se encuentran en el stack.

Durante el procesamiento de un segmento de un algoritmo, el procesador requiere de espacio para almacenamiento de variables temporales. Para esto utiliza el tope del stack en donde se encuentra el registro de activación del segmento.

La Burroughs ha continuado explotando este tipo de arquitectura que facilita la ejecución de programas de alto nivel similares a ALGOL. La B6800 y la B7800 son modelos más recientes con esta arquitectura.

Otra clase de computadoras apareció en esta generación: las supercomputadoras, capaces de hacer computaciones a velocidades muy altas. Control Data Corporation fue la compañía más aventajada en este campo. Primero la CDC-6600 en 1964, luego la CDC-7600 en 1969 y después las famosas CYBER-170. Estas máquinas se caracterizan por tener varios procesadores de entrada y salida (IOP).

Otra característica que da a estas máquinas gran velocidad, es el concepto de tubería (pipelining). En este diseño, cada uno de los pasos para ejecutar una instrucción tiene una unidad independiente en la Unidad Central de Proceso que lo ejecuta. Cuando la máquina empieza a funcionar, la instrucción inicial es obtenida de la memoria por la primera unidad de ejecución. Entonces la segunda unidad empieza la decodificación de la instrucción inicial mientras la primera unidad está obteniendo de la memoria la segunda instrucción. Después la tercera unidad está examinando la instrucción inicial para ver si necesita datos de la memoria, mientras la segunda unidad está decodificando la segunda instrucción y la primera unidad está obteniendo la tercera instrucción de la memoria. En este ejemplo vemos que la primera unidad independiente está dedicada únicamente a obtener instrucciones de la memoria; la

segunda, a decodificación de instrucciones; y la tercera, a verificar si las instrucciones necesitan datos de la memoria. Máquinas con sofisticados sistemas de tubería son la IBM S/360 modelo 195, la CDC STAR y la MU5 de la Universidad de Manchester.

Otra supercomputadora de la época fue la ILLIAC IV (ILLInois Automatic Computer), la cual tiene 64 unidades aritméticas y lógicas separadas, todas ellas supervisadas por una unidad de control. Las ALU`s son capaces de operar simultáneamente.

Para terminar de hablar de las computadoras de Tercera Generación diremos que en esa época los procesadores ya eran capaces de ejecutar hasta un millón de operaciones por segundo, y que otras máquinas famosas que no hemos mencionado fueron la PDP-10 y la PDP-11 de D.E.C., la UNIVAC-1108, la UNIVAC-1110 y los sistemas 34 y 36 de la IBM.

1.2.4. LA CUARTA GENERACION.

Los principales exponentes de esta generación de computadoras son los microprocesadores.

Desde 1970 se empezaron a producir los circuitos LSI (Large Scale Integration), los cuales tienen de 100 a 100,000 compuertas lógicas y son los dispositivos electrónicos que hicieron posible la existencia de los microprocesadores y por lo tanto, de las microcomputadoras.

En 1971 la Intel Corporation empezó a vender el primer microprocesador comercial, el 4C04. Este procesador estaba completamente contenido en un chip de 2,200 transistores que formaban aproximadamente 750 compuertas lógicas. Podía ejecutar 45 instrucciones. Era lento en ejecución (12.5 microsegundos por instrucción) y solo podía procesar cuatro bits en paralelo. Pero estaba en un cuadrito de silicio de menos de un centímetro cuadrado. La compañía Intel estaba, de esta manera, abriendo un campo que todavía no ha dejado de desarrollarse. Los microprocesadores no son solamente utilizados en computadoras de gran poder, sino también en automóviles, aviones, instrumentos de trabajo, artículos domésticos, juguetes, etc.

Después del 4004, en 1972, la Intel sacó a la venta un

procesador de ocho bits, el 8008, el cual podía procesar instrucciones de una memoria principal de 16K bytes. En 1974 la empresa Motorola produjo un procesador de 8 bits, el 6800, que podía direccionar 64K bytes de memoria. En ese mismo año el Intel 8080 salió al mercado con la misma capacidad de memoria y con un conjunto de 111 instrucciones ejecutables. En 1975, Zilog extendió entre las microcomputadoras el procesador Z-80, un poco más rápido y poderoso que el Intel 8080. En 1978 el procesador Intel 8086, con capacidad de procesar 16 bits en paralelo y un Megabyte de memoria, empezó a producirse. Le siguieron el Zilog Z-8000, en 1979, hasta con ocho Megabytes de memoria, y luego el Motorola 68000, con una velocidad de ejecución de 0.25 microsegundos por instrucción y 16 Megabytes de memoria.

Es importante mencionar que una de las causas de la popularidad que alcanzaron las microcomputadoras se debió a que en 1975 la Digital Research comercializó el sistema operativo CP/M, el cual, bastante confiable, relativamente poderoso, muy barato y portable, se volvió un estándar para los fabricantes de software y de hardware, es decir, se hacían máquinas que pudieran manejar CP/M y programas que pudieran correr bajo CP/M.

Llegó un momento en que las microcomputadoras podían ser configuradas de manera que tuvieran el poder de las minicomputadoras. Pero los fabricantes de estas últimas no se quedaron atrás. La Digital Equipment Corporation sacó a la luz la famosa serie de computadoras VAX-11. En un intento por mejorar a la PDP-11 produjeron esta serie de máquinas muy rápidas y poderosas.

En 1981 la National Semiconductor introdujo el procesador NS16000. Un procesador de 32 bits con soporte para lenguajes de alto nivel y con capacidad de manejo de memoria virtual. Recientemente, Motorola empezó a producir el procesador MC68020, de 32 bits, compatible con los procesadores de 16 bits de la familia (MC68000 y MC68010). Este microprocesador, con soporte para lenguajes de alto nivel es sumamente rápido, con una interfase para coprocesadores para poder satisfacer a diferentes tipos de usuarios. Entre estos coprocesadores están el de punto flotante y la unidad de manejo de memoria paginada. La gran velocidad del procesador central se debe en parte a su mecanismo de tubería, al paralelismo en algunas de sus funciones y al reloj de alta frecuencia que posee.

Intel acaba de anunciar a su nuevo procesador de 32 bits, el 80386, el cual será compatible con el 8086, con el 80186 y con el 80286 (todos ellos de 16 bits), será capaz de ejecutar hasta 4 MIPS (millones de instrucciones por segundo), y podrá ejecutar simultáneamente programas de aplicación escritos para diferentes sistemas operativos (pr ejemplo, Unix y MS-DOS).

Debido al tremendo avance que han tenido las arquitecturas de las minicomputadoras y microcomputadoras, ahora resulta muy difícil decir cuales son las diferencias técnicas entre ellas y las grandes computadoras. Hay que mencionar que mucho de esto se debe al advenimiento de los circuitos VLSI (Very Large Scale Integrated, con más de 100,000 compuertas lógicas). El procesador Hewllet Packard 9000, por ejemplo, contiene 450,000 transistores. Hoy en día las velocidades de ejecución de las computadoras están en el rango de 150,000 IPS (las microcomputadoras Apple II y Atari 400) a 25 MIPS (la supercomputadora Amdahl 580/5880).

1.2.5. LA QUINTA GENERACION.

En Abril de 1982, comenzó en Japón un programa de investigación llamado el Proyecto de Sistemas de Computación de Quinta Generación. Se intenta que las arquitecturas que resulten de estas investigaciones, sean las que se utilicen para el diseño de las computadoras que estarán en uso a partir de 1990.

El procesamiento de datos no numéricos, incluyendo el procesamiento de símbolos y la Inteligencia Artificial jugarán un papel más importante en el futuro en el campo del procesamiento de información. A veces se define a la Inteligencia Artificial como a la ciencia que estudia la manera de que las computadoras hagan las cosas en las que el hombre todavía es mejor.

Las metas principales del proyecto de los japoneses son las siguientes:

1) Implantación de mecanismos básicos para inferencia, asociación y aprendizaje en hardware. Un lenguaje lógico parecido a PROLOG es el más viable hasta ahora para ser utilizado en estas máquinas. Una de las razones principales de esto, es que la ejecución de sus instrucciones da facilidades de paralelismo. Dado que este lenguaje trabaja a base de verificar reglas en una base de datos (una verificación de una regla es más o menos lo mismo que una inferencia lógica), el objetivo es lograr hacer una máquina capaz de hacer inferencias muy rápida. Las máquinas de la Quinta Generación deben ser capaces de ejecutar de cien millones a mil millones de LIPS (Logical Inferences Per

Second). En una máquina con arquitectura de von Neumann, un LIPS es equivalente a la ejecución de cien a mil instrucciones. O sea que una máquina de inferencias con la velocidad que se pretende alcanzar, deberá ejecutar de cien mil millones a un billón de instrucciones por segundo, cosa que, ni siquiera en el futuro, se ve posible siguiendo con el modelo de von Neumann. Es por eso que se hace necesario el desarrollo de nuevas arquitecturas. Paralelismo es uno de los principales puntos de investigación en este proyecto.

2) Preparación de software de Inteligencia Artificial para poder utilizar toda la fuerza del hardware que se pretende construir.

3) Implantación en hardware y en software de mecanismos básicos para almacenar y manejar bases de conocimiento.

4) Desarrollo de interfases hombre-máquina. En la Primera Generación de computadoras las interfases de comunicación entre la máquina y quienes la manejaban, eran las lectoras y perforadoras de tarjetas, así como las cintas de papel. En la Segunda Generación se empezaron a utilizar también las impresoras y las cintas magnéticas. En la Tercera, aparecieron los discos magnéticos y las terminales de video. Por último, en la Cuarta Generación se hicieron populares los equipos de graficación y los lectores ópticos.

En esta Quinta Generación se intenta que las interfases hombre-máquina sean capaces de manejar lenguaje natural (escrito y hablado), gráficas, e imágenes, de tal forma que la comunicación entre el hombre y la computadora sea de una manera más natural para el hombre.

1.3. ESTRUCTURA DE UNA COMPUTADORA.

1.3.1. LA MEMORIA.

La memoria principal es el dispositivo de la computadora en donde se almacena el programa que la unidad central de proceso puede ejecutar.

Cuando la computadora posee el nivel de microprogramación, las instrucciones que ejecuta el procesador provienen de un dispositivo llamado micromemoria. La micromemoria en este caso contiene subrutinas de microinstrucciones que se ejecutan dependiendo de las instrucciones que el programador tenga en la memoria principal. La constitución de la micromemoria es igual a la de la memoria principal.

La memoria principal consiste de un gran número de celdas cada una de las cuales puede almacenar un dígito binario o bit. Las computadoras actuales pueden poseer millones de dichas celdas. Dado que un bit no nos dice mucho y un grupo de bits sí, en la mayoría de las computadoras las celdas están organizadas en grupos de un tamaño fijo, de manera que puedan siempre ser tomadas de la memoria o bien se pueda almacenar en ellas un grupo de N bits en una sola operación. A cada grupo de N bits se le llama palabra de información y a N se le llama el tamaño de la palabra. Los tamaños varían de computadora a computadora. Por ejemplo, las computadoras con procesador Zilog Z80 tiene un tamaño de palabra de 8 bits, mientras que la CYBER 170 trabaja con palabras de 60 bits. Para poder accesar la memoria principal para obtener una palabra, o bien para almacenarla, es necesaro que cada palabra tenga un distintivo. A este distintivo se le llama dirección.

Antes se decía que una computadora era de N bits si su tamaño de palabra era de N bits. Ahora, el decir que una computadora es de N bits es algo muy vago e independiente del tamaño de la palabra. A veces se le llama así a una máquina cuando su bus de datos es de N líneas. O bien si el número de bits que forman una dirección de memoria es N. También se usa esta expresión cuando el tamaño de los registros de CPU es N.

Al conjunto de direcciones que existen realmente en la memoria se le llama espacio de direccionamiento físico. Al conjunto de direcciones que pueden aparecer en un programa, se le llama

espacio de direccionamiento lógico. Este último solo está restringido por el número de bits que puede tener una referencia a dirección de memoria, por ejemplo, el contador de programa (PC). El espacio de direccionamiento físico es un subconjunto del espacio de direccionamiento lógico.

La organización del espacio de direccionamiento lógico define la arquitectura de la memoria de una computadora. Existen dos tipos de arquitectura: la lineal y la segmentada.

En una memoria lineal, las direcciones empiezan en la localidad cero y terminan en la localidad limitada por el número de bits asignados al espacio de direccionamiento lógico. Estas direcciones están en forma estrictamente secuencial. El Intel 8080 y el Z80, por ejemplo, manejan memoria lineal.

En una arquitectura de memoria segmentada, pueden existir varios pequeños espacios de direccionamiento lineal dentro de un espacio de direccionamiento segmentado. En este esquema se le llama segmento al espacio de direccionamiento particular en que un dato está localizado, y se le llama desplazamiento a la distancia en bytes desde el comienzo del segmento hasta la localidad específica en donde se encuentra el dato. Los procesadores Z8000 y 8086 manejan memorias segmentadas.

El 8086 maneja una memoria de 1024K bytes que puede ser considerada como un conjunto de segmentos de 64K bytes cada uno. Cada segmento debe comenzar en una dirección que sea divisible por 16, y cuatro segmentos son accesibles en todo momento. Por esto, existen cuatro registros de 16 bits cada uno, en donde se almacenan las direcciones base (inicio) de cada segmento (no es necesario almacenar los cuatro últimos ceros de la dirección, los cuales siempre existen por la divisibilidad de la dirección entre 16). De manera que el procesador construye las direcciones de 20 bits sumando el contenido de uno de los registros (con sus cuatro ceros adicionales) con una cantidad de 16 bits llamada desplazamiento que da el lugar del dato dentro del segmento. Los segmentos pueden estar adyacentes, disjuntos o traslapados.

Por otro lado, al proceso de traducción de direcciones lógicas a direcciones físicas se le llama mapeo de direcciones. Durante el mapeo, las direcciones lógicas pueden ser asignadas a cualesquiera direcciones físicas. El mapeo de direcciones puede ser llevado a cabo totalmente o en partes, en varios estados de la vida del programa: por el programador, mientras escribe el programa; por el compilador, cuando el programa se está compilando; por el cargador, mientras el programa es cargado a la memoria; y por el sistema operativo y la unidad de manejo de memoria, cuando el programa está siendo ejecutado.

La forma más simple de mapeo, es la que puede hacer el programador, dado que las direcciones lógicas son las mismas que las direcciones físicas. Este esquema solo se utiliza en máquinas en donde solamente un usuario utiliza la memoria. Muchas veces en los sistemas modernos de multiusuarios solo se permite a los programas especificar direcciones relativas dentro del programa, y el mapeo se hace después. Cuando el cargador hace el mapeo asignando valores físicos fijos, al mapeo se le llama de asignación estática. Muchas veces no es posible lograr esto. Por ejemplo cuando existe multiprogramación o algún mecanismo de memoria virtual, en muchas ocasiones es necesario movilizar los programas a diferentes partes de la memoria durante la ejecución. A este tipo de mapeo se le llama de asignación dinámica.

Para llevar a cabo los mapeos de direcciones, se utilizan varios mecanismos de relocalización. Un método muy usual es el que el compilador convierta el programa a cargarse en la memoria en un conjunto de bloques, cada uno de los cuales es una secuencia de palabras de memoria contiguas, en donde cada palabra dentro de un bloque puede ser identificada con una dirección lógica la cual consta de una dirección base y una dirección relativa (los segmentos son tipos especiales de estos bloques).

Entonces el sistema operativo toma el control de las direcciones asignadas a cada bloque. Esto se logra almacenando las direcciones base en un mapa de memoria o tabla de direcciones de memoria que está localizado en memoria principal o en un conjunto de registros de CPU (en cuyo caso, estos son llamados registros base o de relocalización). Los circuitos de generación de direcciones de la CPU calculan las direcciones relativas dentro de un bloque. La dirección relativa es convertida a una dirección física al sumársele el contenido del registro de relocalización correspondiente. Los bloques pueden ser cambiados de lugar (relocalizados) en la memoria con solo alterar su dirección base. Pero esta operación privilegiada solo la ejecuta el Sistema Operativo. Una de las causas por la que es necesario relocalizar determinado bloque, es la siguiente: si hay dos bloques en memoria, y se desea almacenar uno más, pero no hay en memoria un lugar lo suficientemente grande para él, entonces es posible que moviendo los bloques existentes, se pueda abrir un lugar para este nuevo bloque, y eso es lo que hace el Sistema Operativo.

En algunos procesadores, el mapeo de direcciones en memoria es llamado mapeo basado en páginas. El espacio de direccionamiento lógico está dividido, en este caso, en unidades llamadas páginas, de un cierto número fijo de bytes cada una. El espacio de direccionamiento físico está dividido en un número de unidades llamadas marcos de página, cada una

del tamaño de una página. La transferencia de direcciones físicas en direcciones lógicas se hace por medio de una tabla en memoria que contiene las direcciones físicas de las páginas que están en cada marco de página.

En este esquema, las direcciones adyacentes en la misma página lógica serán adyacentes también en la misma página física, pero dos páginas que son contiguas en memoria lógica, no lo son necesariamente en memoria física.

En el NS16000, el cual utiliza el mecanismo de mapeo basado en páginas, el espacio de direccionamiento lógico está dividido en 32,768 páginas, cada una con 512 bytes. Los 24 bits de dirección lógica están divididos en dos campos: el campo selector de página (15 bits más significativos) y el campo de desplazamiento (los otros 9 bits). Solamente el campo selector de página es modificado en el proceso de mapeo. Los 9 bits de desplazamiento especifican una localidad dentro de la página, y son pasados sin alteración a través del proceso. Este proceso se lleva a cabo mediante la unidad de manejo de memoria. El proceso consiste en tratar al campo selector de páginas como a un índice de una tabla de direcciones físicas. Las entradas a esta tabla conservan los 15 bits más significativos de la dirección física de un marco de página. Cuando una dirección lógica es enviada a la unidad de manejo de memoria, sus 9 bits de desplazamiento son adheridos a los 15 bits de la dirección física en la tabla, y los 24 bits resultantes son los que se utilizan para accesar la memoria.

En muchos sistemas de computación, el espacio de direcciones lógico es mucho mayor que el espacio de direcciones físico. Existe un mecanismo llamado sistema de memoria virtual que permite que todo el espacio de direcciones lógico esté disponible para almacenamiento. De manera que la parte que no cabe en memoria principal, se almacena en alguna unidad de disco magnético. Esto quiere decir que en todo momento, solo un cierto número de páginas pueden ser accesadas de manera aleatoria.

Siempre que una computadora con memoria virtual genera una dirección de memoria, cierto proceso se encarga de verificar si esta dirección pertenece a alguna página que se encuentre en memoria. Si se encuentra, entonces la dirección es traducida a la dirección física apropiada; si no, el Sistema Operativo ejecuta una operación de intercambio de página (page swap), en donde la página requerida es cargada a la memoria. La página que utilizaba el espacio que utiliza esta nueva página es almacenada en disco si tuvo alguna modificación. Es usual que se carguen en memoria páginas con localidades cercanas a la requerida. Existen procedimientos para verificar si las páginas están siendo utilizadas frecuentemente, para que, en caso de que no, se saquen de la

memoria y se abra espacio para páginas que se requerirán posteriormente.

Una computadora con sistema de memoria virtual da al programador la sensación de poseer mucho más memoria de la que en realidad tiene. Los procesos de aplicación se alentan un poco por el trabajo extra que el procesador tiene que hacer para ejecutar las rutinas de paginación del Sistema Operativo, pero esto permite que sea posible la ejecución de programas que no cabrían en la memoria principal y que por lo tanto nunca podrían ser ejecutados.

El concepto de multiprogramación hizo muy importante, para una mejor utilización de la memoria, el uso de sistemas de memoria virtual, al igual que el uso de los sistemas de mapeo de direcciones. También hubo otro concepto que se hizo muy necesario: el de protección de memoria. Antes de los sistemas multiusuarios, era necesario proteger al Sistema Operativo del programa de aplicación, el cual, ya fuera por equivocación o adrede, pudiera afectar durante su ejecución, el funcionamiento de los programas que manejaban los recursos del sistema. Pero ahora, con la capacidad de las máquinas de que varios usuarios compartan la misma memoria, se hizo necesario que se crearan esquemas para que los programas se protegieran unos de otros, a la vez de que no pudieran afectar el funcionamiento del Sistema Operativo.

En algunas computadoras, la protección se hace con un sistema de niveles o anillos, en el cual, hay una jerarquía del anillo más privilegiado al menos privilegiado. Un anillo más privilegiado tiene acceso a toda la información en un anillo menos privilegiado. Este sistema tiene el problema de que los procesos en ejecución no siempre tienen una relación jerárquica entre ellos.

Otro esquema de protección es el que utiliza tablas de capacidades. Cada proceso tiene asociado a él una tabla de operaciones que pueden afectar a otros procesos dentro del sistema. De esta manera se elimina el problema de la jerarquía.

Son dos las características de protección más importantes que deben ser tomadas en cuenta. La primera es la que impide que un proceso haga referencia a localidades de memoria que son ocupadas por otros procesos. La segunda es la que impide que sobre un cierto bloque (página o segmento) de memoria se hagan operaciones indebidas, por ejemplo, lectura y/o escritura.

Por otro lado, pasando al tema de representación de objetos en la memoria, tenemos que cada uno de los N bits de una palabra de información pueden ser un uno o un cero independientemente, por lo que una palabra puede ser uno de 2 a la N diferentes

estados. O sea que con una palabra se puede representar uno de 2 a la N objetos diferentes. Usualmente estos objetos son números o caracteres.

La forma usual de representar números es por medio del código binario, es decir, el número X se representa por medio del conjunto de bits BN-...-B2-B1-B0 en donde X = BN*2^N + ... + B2*2^2 + B1*2^1 + B0*2^0. (El signo * indica multiplicación y el signo ^ indica potenciación). El máximo número representable es 2^N - 1, (donde N es el tamaño de palabra), y el mínimo es el cero. Una de las maneras de representar números negativos es utilizando un bit de la palabra para representar el signo; esto hace que el número de rangos representables vaya de -2^(N-1)-1 a +2^(N-1)-1. Este sistema se llama magnitud con signo y tiene un problema: que existen dos representaciones para el cero. Existen otros sistemas tales como el complemento a uno y el complemento a dos. El sistema de complemento a dos da la enorme ventaja de que para sumar y restar números se utiliza solamente la operación de adición.

Para representar caracteres alfanuméricos y símbolos se utilizan varios sistemas. Los más usados son el ASCII y el EBCDIC. En ASCII se utilizan 8 bits para representar 128 caracteres. Esto nos dice que sobra un bit, el cual es utilizado como bit de paridad. El bit de paridad permite verificar cierto tipo de errores durante la transmisión de información. Con el código EBCDIC se pueden representar hasta 256 caracteres con 8 bits también. Naturalmente que en las máquinas de más de ocho bits de palabra se empaquetan varios caracteres en una palabra.

Dado que es importante el manejo de grupos de ocho bits, a un conjunto de estos se le denomina como byte. En algunas máquinas tales como la PDP-11 y la IBM-370, aún cuando sus tamaños de palabra son de 16 y 32 bits respectivamente, ambas permiten que cada byte tenga su propia dirección. Si la memoria permite almacenar N bytes, entonces estos serán direccionados del 0 al N-1.. En la PDP-11 una palabra debe tener una dirección par, y consiste del byte con esa dirección concatenado con el byte de la dirección siguiente.

La IBM-370 permite que las palabras de información comiencen con cualquier byte. La palabra se forma entonces, con el byte direccionado y los tres bytes de las direcciones siguientes. Sin embargo, la palabra es procesada más rápido si el primer byte está en una dirección que sea múltiplo del número cuatro. Esta máquina permite también el manejo de medias palabras (dos bytes) y de dobles palabras (ocho bytes).

Hay que mencionar que en muchas máquinas es posible encontrar instrucciones que permitan hacer referencia a un solo bit dentro de algún byte o dentro de alguna palabra.

Como la memoria es utilizada para almacenar programas y datos y el procesador no distingue entre ambos, a veces ocurren errores de ejecución cuando el procesador trata de ejecutar una palabra que es un dato y no una instrucción. Para resolver este problema, algunas computadoras poseen un bit adicional asociado a cada palabra. Este bit, llamado algunas veces metabit, indica cuando está apagado, que la información en la palabra, es una instrucción. Cuando está prendido indica que la palabra contiene datos.

1.3.2. LA UNIDAD CENTRAL DE PROCESO.

1.3.2.1. ORGANIZACION.

La unidad central de proceso o CPU (Central Processing Unit) de la computadora es la que se encarga de la ejecución de los programas que se encuentran en la memoria principal.

Se puede decir que la CPU es un procesador de un conjunto de instrucciones de propósito general. Se dice de propósito general para diferenciarlo de otros tipos de procesadores, tales como los de entrada y salida, cuyos conjuntos de instrucciones ejecutables son de propósito especial.

Casi toda CPU tiene solamente un procesador. A la que tiene más de uno se le llama multiprocesador.

Una CPU está formada por varios dispositivos. Los principales son: la unidad de control, la unidad aritmética y lógica y los registros.

La unidad de control es capaz de ejecutar dos operaciones sobre la memoria. Una es la de obtener información de ésta y la otra es la de almacenar información en ella.

Para obtener información de la memoria, la CPU tiene que

especificar la dirección de la localidad que quiere accesar. Para esto, transfiere la dirección a un registro de CPU llamado registro de dirección de memoria o MAR (Memory Address Register). El MAR está conectado a las líneas del bus de memoria. El bus de memoria es el conjunto de cables que transporta información desde y hacia la memoria. De esta manera la información que se encuentra en la celda requerida es tranferida a la memoria. Por el mismo bus de memoria, la unidad de control indica a la memoria por medio de una señal de control, que requiere una operación de lectura. La unidad de control espera hasta recibir otra señal de control llamada función-de-memoria-completada o MFC (Memory Function Completed) de la memoria, la cual indica que la información ya ha sido transferida a la CPU. Otro registro de la CPU es a donde la memoria tranfiere la información . El nombre de éste es registro de datos de memoria (Memory Data Register).

La operación de la unidad de control para almacenar información en memoria es similar a la antes descrita. La dirección a donde se almacenarán los datos se coloca en el registro de dirección de memoria, los datos a ser tarnsferidos a memoria se colocan en el registro de datos de memoria, después la unidad de control manda una señal de escritura a memoria. Por último la unidad de control espera que la memoria le envíe la señal MFC.

Los mecanismos de transferencia señalados anteriormente, donde uno de los dispositivos inicia la transferencia (la CPU pide lectura o escritura) y espera hasta que el otro dispositivo responde (la memoria manda la señal MFC) se llaman de transferencia asíncrona. Con este tipo de mecanismo es posible que pueda haber transferencia de información entre dos dispositivos independientes que tengan diferentes velocidades de operación.

Existe otro mecanismmo para transmitir información entre dos dispositivos y se llama de transferencia síncrona. En este esquema, una de las líneas de control transporta pulsos de un reloj funcionando a una frecuencia determinada. Estos pulsos van hacia la memoria y la CPU. Una operación de memoria puede ser completada durante cada periodo de reloj. Los momentos en que la dirección es puesta en las líneas de dirección y en que los datos son cargados en el MDR son relativos a los pulsos del reloj. La transferencia síncrona permite más simplicidad en el diseño del sistema. El problema viene cuando los dispositivos trabajan a diferentes velocidades, pues entonces hay que reducir las velocidades a la misma del dispositivo más lento.

Existe gran variación entre las frecuencias de los relojes de las diferentes computadoras. Generalmente la frecuencia aumenta con el tamaño de la máquina, aunque el reciente

microprocesador MC68020, trabaja con un reloj de 16 MHz (millones de ciclos por segundo), frecuencia que hasta hace muy poco tiempo era solamente encontrada en relojes de grandes computadoras.

Los pasos para la ejecución de una instrucción en la mayoría de las computadoras, son los siguientes:

- Se obtiene la instrucción de la memoria, por cualquiera de las formas antes descritas, y se pasa al registro de instrucción o IR (Instruction Register). Este registro de la CPU siempre contiene la instrucción que se está ejecutando.

- El contador de programa o PC se incrementa para que apunte a la siguiente instrucción en memoria principal. El PC es otro registro de la CPU. Siempre contiene la dirección de la próxima instrucción a ejecutarse.

- Los circuitos de la unidad de control detectan de que tipo de instrucción se trata.

- Si la instrucción necesita datos que están en la memoria, se obtienen de ésta por el mismo método del primer paso. Estos datos se dejan en la CPU en registros especiales de trabajo. El número de estos registros varía de computadora a computadora.

- Se ejecuta la instrucción. Para esto, por ejemplo, si se necesita ejecutar una operación aritmética, se utiliza la unidad aritmética y lógica de la CPU; si la instrucción es de llamada a subrutina, se modifica el PC y se almacenan direcciones en el stack; etc.

Aparte de los registros ya mencionados (IR, PC, etc.) de la CPU, existen otros. Algunos de ellos, manipulables por el programador de sistemas.

En los sistemas de computación existe un área de almacenamiento llamada el stack (pila). El stack generalmente se encuentra en la memoria principal, aunque algunas veces es un conjunto de registros en la CPU. El stack es una estructura de datos del tipo LIFO (Last-In-First-Out o lo que es lo mismo, Ultima-Entrada-Primera-Salida) sumamente útil en algunas aplicaciones que serán descritas después. En estas aplicaciones, la CPU hace uso del stack, pero también el programador puede tener acceso a él. Un registro de la CPU llamado apuntador del stack o SP (Stack Pointer) es el que contiene la dirección del registro o de la memoria que se debe utilizar en el próximo acceso al stack.

Otro registro sumamente útil en la CPU es el llamado registro de estado del programa. Antes dijimos algo de la PSW del

microprocesador MC68020, trabaja con un reloj de 16 MHz (millones de ciclos por segundo), frecuencia que hasta hace muy poco tiempo era solamente encontrada en relojes de grandes computadoras.

Los pasos para la ejecución de una instrucción en la mayoría de las computadoras, son los siguientes:

- Se obtiene la instrucción de la memoria, por cualquiera de las formas antes descritas, y se pasa al registro de instrucción o IR (Instruction Register). Este registro de la CPU siempre contiene la instrucción que se está ejecutando.

- El contador de programa o PC se incrementa para que apunte a la siguiente instrucción en memoria principal. El PC es otro registro de la CPU. Siempre contiene la dirección de la próxima instrucción a ejecutarse.

- Los circuitos de la unidad de control detectan de que tipo de instrucción se trata.

- Si la instrucción necesita datos que están en la memoria, se obtienen de ésta por el mismo método del primer paso. Estos datos se dejan en la CPU en registros especiales de trabajo. El número de estos registros varía de computadora a computadora.

- Se ejecuta la instrucción. Para esto, por ejemplo, si se necesita ejecutar una operación aritmética, se utiliza la unidad aritmética y lógica de la CPU; si la instrucción es de llamada a subrutina, se modifica el PC y se almacenan direcciones en el stack; etc.

Aparte de los registros ya mencionados (IR, PC, etc.) de la CPU, existen otros. Algunos de ellos, manipulables por el programador de sistemas.

En los sistemas de computación existe un área de almacenamiento llamada el stack (pila). El stack generalmente se encuentra en la memoria principal, aunque algunas veces es un conjunto de registros en la CPU. El stack es una estructura de datos del tipo LIFO (Last-In-First-Out o lo que es lo mismo, Ultima-Entrada-Primera-Salida) sumamente útil en algunas aplicaciones que serán descritas después. En estas aplicaciones, la CPU hace uso del stack, pero también el programador puede tener acceso a él. Un registro de la CPU llamado apuntador del stack o SP (Stack Pointer) es el que contiene la dirección del registro o de la memoria que se debe utilizar en el próximo acceso al stack.

Otro registro sumamente útil en la CPU es el llamado registro de estado del programa. Antes dijimos algo de la PSW del

Sistema 360 de IBM, ahora hablaremos del registro del procesador Intel 8086.

En el Intel 8086 el registro de estado es de 16 bits y está dividido en dos secciones: las banderas de status y las banderas de control. Las banderas de status se afectan de manera diferente durante la ejecución de algunas instrucciones. Estas banderas son seis: la de acarreo, la de overflow (sobreflujo o desbordamiento), la de cero, la de acarreo auxiliar, la del signo y la de paridad.

- La bandera o bit de acarreo se prende cuando en una operación de adición existe acarreo en el bit más significativo del resultado. También se prende si en una substracción hay que "prestarle" al bit más significativo. Algunas operaciones de corrimiento y de rotación de bits tambien afectan a esta bandera.

- El bit de overflow se prende cuando hay overflow en alguna operación aritmética, esto es, cuando dígitos significativos se pierden porque el tamaño de la computación excede la capacidad de la localidad destino.

- La bandera de cero se prende cada vez que el resultado de alguna operación es cero.

- La bandera de acarreo auxiliar se prende cuando hay acarreo fuera de los cuatro bits menos significativos dentro de los cuatro bits más significativos en una operación de adición. También se prende cuando hay un préstamo de los cuatro bits más significativos dentro de los cuatro bits menos significativos en una operación de substracción. Esta bandera es útil cuando se hacen operaciones con aritmética decimal.

- La bandera de signo se prende siempre que el resultado de una operación aritmética tiene encendido el bit más significativo. Dado que los números binarios negativos se representan con la notación de complemento a dos, la bandera de signo indica el signo del resultado (uno si es negativo y cero si es positivo).

- Si la bandera de paridad está prendida, el resultado de la última operación efectuada tiene paridad par. Esta bandera puede ser utilizada si hubo error en la tranmisión de datos.

Por otro lado tenemos a las banderas de control. Estas son usadas por programas para alterar ciertas operaciones del procesador. Estas banderas son tres: la de dirección, la de interrupciones y la de "trampa".

- El prender la bandera de dirección provoca que las instrucciones de manejo de cadenas se autodecrementen, es

decir, que procesen cadenas desde direcciones altas hacia direcciones bajas, o lo que es lo mismo, de derecha a izquierda. El apagar esta bandera causa que las instrucciones de manejo de cadenas se autoincrementen, o sea, que procesen cadenas de izquierda a derecha.

- La bandera de interrupción prendida permite que la CPU reconozca solicitudes de interrupciones externas enmascarables (incluyendo intento de interrupción de periféricos). Esta bandera apagada deshabilita estas interrupciones. La bandera de interrupción no tiene efecto ni en interrupciones externas no enmascarables ni en interrupciones generadas internamente.

- La bandera de "trampa", al estar prendida pone al procesador en un modo de ejecución tal, que la CPU genera una interrupción interna después de cada instrucción. Esto permite que un programa pueda ser rastreado cuando se está ejecutando, instrucción por instrucción.

Aparte de todos los registros de CPU que hemos mencionado, existen otro grupo de registros que también están allí, y son con los que el programador tiene más contacto. A veces son llamados registros de propósito general y su función es la de permitir al programador algunas localidades de almacenamiento para poder trabajar. La velocidad de acceso a estos registros es mucho mayor que la velocidad de acceso a la memoria. El número de estos registros varía de máquina a máquina. Por ejemplo, el procesador Intel 8080 solo tiene seis registros de ocho bits, mientras que el Sistema Foonly F2 posee 16 registros de 36 bits cada uno.

En algunas máquinas, los registros de propósito general juegan un papel muy importante en la ejecución de algunas de las instrucciones.

Los muy conocidos acumuladores forman parte de este conjunto de registros. En algunos procesadores, como el Z80 por ejemplo, una gran cantidad de las operaciones aritméticas y lógicas toman como uno de los operandos al contenido del acumulador (en el caso del Z80 solo existe uno) y dejan el resultado en éste. Todo eso, implícitamente.

Por otro lado, muchas de las operaciones que ejecutan algún acceso a memoria toman la dirección de la palabra de información implícitamente de algún registro. En el caso del procesador Intel 8080 este registro es el HL. Hay que mencionar que el registro HL se puede utilizar como cualquiera de los otros registros de propósito general, o sea, como ayuda al programador.

1.3.2.2. EL NUMERO DE DIRECCIONES.

Es posible clasificar a las CPU's por la manera en que utilizan la memoria y los registros que poseen. Existen máquinas de tres direcciones, máquinas de dos direcciones, máquinas de una dirección, y por último, máquinas de stack o de cero direcciones.

Supongamos que tenemos que ejecutar la asignación siguiente:

A1=(A2-A3)/A4+A5*A6

donde A1,A2,...,A6 son las direcciones de memoria que contienen las cantidades con las que queremos operar. El procesador solo puede ejecutar una operación aritmética por instrucción. Entonces hay que hacer un pequeño programa para poder llevar a cabo esta asignación. La manera de como están especificados los operandos de las instrucciones que podemos utilizar es variado.

La forma más sencilla que podrían tener las instrucciones es la siguiente:

SUMA X,Y,Z	(Z=X+Y)
RESTA X,Y,Z	(Z=X-Y)
MUL X,Y,Z	(Z=X*Y)
DIV X,Y,Z	(Z=X/Y)

donde X y Y son las direcciones de memoria en donde se almacenarán los operandos, y Z es la dirección en donde se almacenará el resultado.

A las instrucciones con esta forma se les llaman de tres direcciones. A las CPU's que ejecutan predominantemente instrucciones de esta forma se les llama máquinas de tres direcciones. Este tipo de máquinas no requiere de acumuladores.

El programa que necesitaríamos para nuestra asignación es el siguiente:

```
RESTA A2,A3,A1
DIV A1,A4,A1
MUL A5,A6,T1
SUMA A1,T1,A1.
```

Podemos notar que en la asignación hay cuatro operaciones aritméticas y que el programa tiene cuatro instrucciones. Esta relación de uno a uno siempre se cumple. T1 es una localidad de memoria que se utiliza meramente como auxiliar en las computaciones.

Hay que mencionar que este tipo de instrucciones, aunque es muy simple, es muy costoso de construir, puesto que cada instrucción, o bien necesitaría una palabra de memoria con muchos bits, o bien necesitaría ocupar varias palabras, lo cual no sería muy bueno pues la ejecución de una instrucción implicaría muchos accesos a memoria.

Por ejemplo, una máquina con memoria de 1 Mbyte (un millón de bytes) necitaría 60 bits para que una instrucción aritmética pudiera especificar la localización de sus operandos y del resultado. La alternativa en este tipo de máquinas es utilizar registros de propósito general como operandos. Esto reduciría grandemente el número de bits por instrucción. Una máquina que trabaja de esta forma es la CYBER 170.

En últimas fechas se ha estado tratando de orientar el diseño de las arquitecturas para que puedan manejar lenguajes de alto nivel. Debido a que en los lenguajes de alto nivel es usual tener instrucciones con tres operandos, es razonable creer que una arquitectura que maneje a estos lenguajes, debe ser capaz de aceptar instrucciones de tres direcciones.

Sin embargo, D. Knuth [1971] demostró por medio de una investigación, que en 10,000 líneas de programación en lenguaje FORTRAN, el 95 por ciento de todas las instrucciones eran de la forma A=A+B , o de la forma A=B . Estas instrucciones no entran en el esquema de las tres direcciones. Más bien son equivalentes a las instrucciones de dos direcciones.

En las instrucciones de dos direcciones el resultado se almacena en alguna de las direcciones donde hay operandos. Las instrucciones de operaciones aritméticas serían:

```
SUMA X,Y        (Y=X+Y)
RESTA X,Y       (Y=X-Y)
MUL X,Y         (Y=X*Y)
```

```
DIV X,Y          (Y=X/Y)
```

y nuestra asignación podría ser programada de la siguiente manera:

```
RESTA A2,A3
DIV A3,A4
MUL A5,A6
SUM A4,A6
MOVER A6,A1
```

Es necesario introducir una nueva instrucción:

```
MOVER X,Y
```

Esta instrucción indica al procesador que debe poner en la dirección de memoria Y la misma información que existe en la localidad X.

En el programa anterior hay un problema y es que A3, por ejemplo, ha perdido su contenido original. Ahora A3 es igual a A2-A3. Esto no debe suceder pues el contenido de A3 podría ser necesitado en computaciones posteriores. Entonces es conveniente reescribir el programa:

```
MOVER A3,A1
RESTA A2,A1
MOVER A4,T1
DIV A1,T1
MOVER A5,A1
MUL A6,A1
SUMAR T1,A1
```

Algunas computadoras tales como la PDP-11, tienen sus instrucciones de dos direcciones de manera diferente. Por ejemplo, la instrucción RESTAR X,Y indica al procesador que debe restar A a B y debe poner el resultado en B. La PDP-11, al igual que la IBM-370, son máquinas de dos direcciones.

Dado que el hecho de reducir el número de bits necesarios para una instrucción, reduce el costo de las computaciones, existe otro tipo de instrucciones en las cuales el segundo operando es un registro de propósito general. En este caso, si la CPU tiene 8 (o 16) registros entonces la instrucción solo necesita

3 (o 4) bits para especificar el segundo operando. A las instrucciones con este formato se les llama algunas veces instrucciones de una y media dirección. El conjunto de instrucciones de la PDP-10 posee muchas instrucciones de una y media dirección.

Reduciendo aun más el número de bits en una instrucción obtenemos una nueva forma: las instrucciones de una dirección. Las computadoras en donde predominan este tipo de instrucciones se llaman máquinas de una dirección.

Estas máquinas solo poseen un acumulador en su CPU. Entonces este acumulador se utiliza como operando implícito y como lugar en donde se almacenará el resultado de la computación.

Las instrucciones para ejecutar las operaciones aritméticas básicas son las siguientes:

```
SUMA X      (AC=AC+X)

RESTA X     (AC=AC-X)

MUL X       (AC=AC*X)

DIV X       (AC=AC/X)
```

a las cuales tenemos que agregar dos nuevas instrucciones que se hacen necesarias. Estas son:

```
CARGA X

ALMACENA X.
```

La primera es utilizada para cargar el acumulador con X. La segunda permite tomar el contenido del acumulador y almacenarlo en la dirección de memoria X.

Entonces para ejecutar la asignación, necesitaríamos el siguiente programa:

```
CARGA A2
RES A3
DIV A4
ALMACENA A1
CARGA A5
MUL A6
SUM A1
```

ALMACENA A1.

Un procesador típico de una dirección podría ser el Z80, aunque hay que mencionar que las calculadoras de bolsillo son también máquinas de una dirección.

Existe un tipo de instrucciones en donde no se especifican direcciones. Se llaman instrucciones de cero direcciones. Dado que las operaciones aritméticas básicas requieren de dos operandos, estos deben estar en algún lugar conocido antes de la ejecución de la instrucción. Estos lugares podrían ser localidades de memoria fijas o bien un par de registros en la CPU. Dado que el acceso a registros de CPU es más rápido que el acceso a memoria, en muchas computadoras existen dos o más de estos registros. Son usados como acumuladores de manera que hay que especificar direcciones (los números de registro) en las instrucciones.

Sin embargo, existe un tipo de máquina en cuyas instrucciones es posible no especificar direcciones. Este tipo de máquina trabaja con un stack de registros. A este tipo de CPU se le llama máquina de stack o máquina de cero direcciones. El stack contiene a los acumuladores. El acumulador del tope del satck es el único que podemos accesar directamente.

Las operaciones aritméticas básicas, las cuales necesitan dos operandos, actúan siempre sobre el acumulador que está en el tope del stack y sobre el acumulador que está en el nivel inmediato inferior al del tope del stack, o sea, en el segundo nivel. El resultado se carga en el segundo nivel y se elimina el tope del stack, desapareciendo de esta manera, los dos operandos de la instrucción. Así el resultado queda en el tope del stack.

Las operaciones aritméticas básicas son entonces:

SUMA	(Tope del stack= nivel inferior al tope + tope)
RESTA	(Tope del stack= nivel inferior al tope - tope)
MUL	(Tope del stack= nivel inferior al tope * tope)
DIV	(Tope del stack= nivel inferior al tope / tope)

Al manejar el concepto de stack siempre es necesario tener

instrucciones que son de una dirección, esto indica que una máquina de satck no puede solamente trabajar con instrucciones de cero direcciones. Las instrucciones que necesitamos son las siguientes:

```
PUSH X

POP X
```

en donde la primera, toma el contenido de la localidad de memoria X y la pone en el tope del stack. La segunda toma el contenido del tope del stack y lo almacena en la localidad de memoria X.

El programa que se necesita para ejecutar la asignación

```
A1=(A2-A3)/A4+A5*A6
```

es el siguiente:

```
PUSH A2
PUSH A3
RESTA
PUSH A4
DIV
PUSH A5
PUSH A6
MUL
SUMA
POP A1.
```

En cualquier programa de este tipo, siempre habrá una operación PUSH por cada operando de la asignación, el mismo número de operaciones aritméticas que en la asignación, y una operación POP, que es la que completa la asignación.

Típicas máquinas de stack son la B7800 de Burroughs y la HP-3000 de Hewlett Packard.

1.3.2.3. FORMATO DE LAS INSTRUCCIONES.

Las instrucciones que pueden ser ejecutadas por los procesadores de la mayoría de las computadoras, están formadas por dos campos. El primer campo es el código de operación, el cual, muchas veces está en la parte izquierda de la instrucción, y otras está repartido en ésta. El segundo campo es el de los operandos. Este no siempre está presente en la instrucción, pues no todas las instrucciones lo requieren.

En algunas computadoras, una instrucción es del tamaño de una palabra de información de memoria. En otras, varias instrucciones ocupan una palabra. Pero en la mayoría de los microprocesadores, la instrucción ocupa varias palabras. En el procesador Z8000, de 16 bits de palabra, es posible formar una instrucción con 5 palabras. Sin duda, es preferible que la instrucción quepa en una palabra pues esto ahorra accesos a memoria durante la ejecución del programa, reflejándose esto en el tiempo de proceso. Aunque hay que decir que esto es muy difícil por la variedad en la longitud de las instrucciones de un mismo conjunto.

El código de operación es el grupo de bits que especifican al procesador que acción debe tomar. La unidad de control es la parte de la CPU que detecta al código de operación. En una instrucción, el código de operación ocupa, en la mayoría de las computadoras, un número fijo de bits. De esta manera, si se ocupan N bits para esto, es posible tener un conjunto de hasta 2 a la N instrucciones.

Aunque es conveniente aclarar que un mayor número de instrucciones no necesariamente implica una mayor eficiencia de la computadora. De hecho, en la Universidad de California en Berkeley, se inició el proyecto RISC (Reduced Instruction Set Computer), en el cual se investiga una alternativa a las computadoras con conjuntos de instrucciones complejos (CISC). Con un conjunto de instrucciones muy simple y un diseño de arquitectura adecuado, una máquina con una magnífica eficiencia puede llegar a construirse. La simplicidad del conjunto de instrucciones permite que la mayoría de éstas se ejecuten en un ciclo de máquina y la simplicidad de cada instrucción garantizan un tiempo de ciclo bastante corto. Además una máquina de este tipo debe poder ser diseñada en un tiempo muy corto.

El tamaño de las instrucciones es algo que varía con mucha frecuencia dentro de un mismo conjunto de instrucciones. En los sistemas IBM 360 y 370, las instrucciones consisten de 2, 4 o 6 bytes. Se utilizan 5 formatos diferentes, dependiendo de la localización de los operandos requeridos. El código de operación es de una longitud fija de 8 bits para todas las instrucciones. A continuación se describen los formatos.

- Instrucciones RR (Registro-Registro). Los operandos R1 y R2

son registros de propósito general de CPU y el resultado se almacena en R1. La instrucción es de 16 bits.

- Instrucciones RX (Registro-Indice). Un operando está en R1, el cual es un registro de propósito general. El otro está en memoria principal. La dirección de memoria efectiva es R2+X2+D2, en donde R2 y X2 denotan el contenido de registros de propósito general que están siendo usados como registros base e índice respectivamente, y D2 es el desplazamiento al cual también se le llama dirección relativa y es de 12 bits de longitud. El resultado se carga en R1. La longitud de la instrucción es de 32 bits.

- Instrucciones RS (Registro-Memoria). En estas instrucciones dos operandos, R1 y R2 son registros de propósito general, mientras que un tercero está en memoria principal y se forma con B2+D2, donde B2 es otro registro de CPU y D2 es un campo de 12 bits, y es de desplazamiento. Longitud de esta instrucción: 32 bits.

- Instrucciones SI (Memoria-Inmediato). Un operando de esta instrucción está en memoria principal, el otro se encuentra en los bits del 8 al 15 de la instrucción. El operando que está en memoria principal, está otra vez denotado por un registro base y un desplazamiento. Las instrucciones SI son de 32 bits.

- Instrucciones SS (Memoria-Memoria). Ambos operandos se encuentran en memoria principal. La dirección de estos operandos está dada con registros bases y desplazamientos. El detalle importante en instrucciones con este formato es que es posible especificar de cuantos bytes serán los operandos. Existen 8 bits en la instrucción destinados para ello. Estas instrucciones tienen una longitud de 48 bits.

Los dos primeros bits de cada instrucción indican a la unidad de control que formato tienen éstas. Para las instrucciones RR ambos bits deben estar apagados, para las SS ambos deben estar prendidos. En las de formato RX, los bits deben estar en 01, y en las RS y SI, deben estar en 10. Existen en los sistemas 360 y 370 casi 200 instrucciones aunque esta gran cantidad se debe mayormente a la variedad de tipos de datos que los sistemas pueden manejar.

No en todas las computadoras existe un sistema tan elaborado para distinguir los diferentes formatos que pueden tener las instrucciones. Por ejemplo, el conjunto de instrucciones del procesador MC68000 tiene por lo menos 18 diferentes formatos. Esto se debe a que hubo necesidad de incluir las más de 300 instrucciones (en realidad solamente son 56 instrucciones básicas, las demás son variantes de éstas) en palabras de 16 bits, tratando de no desperdiciar ninguna combinación de

estos. El mismo es el caso de las 111 instrucciones del Intel 8080, y el de las 89 instrucciones del Z80, donde el tamaño de sus palabras es de 8 bits. En el 8080 se utilizan de una a tres palabras por instrucción, mientras que el Z80 tiene instrucciones de 1, 2, 3 y 4 bytes. En estos tres procesadores, a veces los bits del código de operación se encuentran distribuidos en la instrución, de manera que en ocasiones, después de un operando se encuentran bits del código de operación. Ejemplo de esto es la instrucción DEC en el Z80, o la instrucción MULS en el MC68000.

1.3.2.4. MODOS DE DIRECCIONAMIENTO.

Ya antes hablamos de la cantidad de direcciones que puede tener una instrucción. Nos referíamos a la cantidad de operandos que pueden estar especificados en una instrucción. Ahora hablaremos de las distintas formas en las que una instrucción puede hacer referencia de sus operandos.

Los operandos no siempre están especificados en la instrucción al momento de la ejecución. En la mayoría de los casos, el formato de la instrucción establece que éstas solo hagan referencia, de algún modo, a los operandos, los cuales, se encuentran generalmente en memoria o en algún registro de la CPU. Al definir la arquitectura de un procesador, a las diferentes maneras de hacer referencia a los operandos de las instrucciones, se les llama modos de direccionamiento.

Existen dos maneras de especificar al procesador con que modo de direccionamiento debe operar. La primera consiste en que cada instrucción tenga un código de operación para cada modo de direccionamiento diferente. La otra consiste en dejar algunos bits en el formato de la instrucción para especificar allí el modo de direccionamiento.

A fin de cuentas lo que el procesador debe hacer al detectar cualquier modo de direccionamiento es encontrar la dirección en donde están los operandos de la instrucción, a la cual se le denomina dirección efectiva.

Los modos de direccionamiento varían de máquina a máquina, aunque hay que aclarar que algunos modos son comunes en casi todas las computadoras. Ejemplos de estos modos son: el inmediato, el de memoria directo, el de registro directo, etc. A continuación daremos una descripción de algunos modos.

- DIRECCIONAMIENTO INMEDIATO.

El direccionamiento inmediato es el modo más sencillo, esto es porque el operando está especificado en la instrucción. De esta manera no hay que hacer referencia a otras localidades de memoria, ni tampoco a los registros de CPU, por lo que es posible ahorrar tiempo de acceso a memoria. Existen dos posibilidades en este modo de direccionamiento. Una es que el operando esté dentro de la palabra que contiene el código de operación. De esta manera no hay accesos extras a memoria para ejecutar la instrucción. La otra posibilidad es que el operando esté en la palabra siguiente a la que contiene el código de operación. En este caso, cuando la unidad de control detecta la instrucción, automáticamente obtiene de la memoria la siguiente palabra.

- DIRECCIONAMIENTO DIRECTO DE MEMORIA.

En esta forma de direccionamiento, el operando se encuentra en memoria, aunque no necesariamente cerca de la instrucción. La instrucción simplemente especifica la dirección de memoria en donde está el operando. A este modo también se le llama absoluto.

En el MC68000 hay dos tipos de direccionamiento directo de memoria llamados direccionamiento absoluto corto y direccionamiento absoluto largo. En el primero, la dirección del operando está en la localidad de memoria cuya dirección es el valor de la extensión con signo de una palabra de 16 bits que se especifica en la instrucción, de manera que es posible hacer referencia a operandos en 64K bytes de memoria. En el segundo tipo, el operando está en la localidad cuya dirección es la concatenación de dos palabras de 16 bits que se especifican después de la instrucción.

- DIRECCIONAMIENTO DIRECTO DE REGISTRO.

En este modo de direccionamiento, el operando se encuentra en un registro de la CPU. El número de registro se especifica en la instrucción.

Debido a que son pocos los registros de CPU manipulables por el programador, las instrucciones en este modo de direccionamiento pueden ser muy cortas. Generalmente existen entre 8 y 16 registros en una computadora, por lo que se necesitan 3 o 4 bits para hacer referencia a ellos. El tamaño de los registros es casi siempre igual al tamaño de palabra de la máquina.

En muchas máquinas es posible manejar el contenido del contador de programa y del apuntador del stack.

El MC68000 tiene dos juegos de 8 registros en la CPU. Un juego se utiliza para cargar datos y el otro se utiliza para cargar direcciones. En el caso de esta máquina no se cumple el hecho de que los registros sean del mismo tamaño que la palabra de memoria (de 16 bits). Los registros son de 32 bits y por eso se ha dicho algunas veces que ésta es una máquina de 32 bits de palabra.

- DIRECCIONAMIENTO INDIRECTO.

Con este modo de direccionamiento, la instrucción contiene un número de registro de CPU o bien una dirección de memoria, en donde se especifica una dirección de memoria cuya localidad contiene al operando que la instrucción requiere. Al número que está contenido en el registro de CPU o en la localidad de memoria es llamado apuntador.

Existen muchos modos de direccionamiento derivados de éste. Por ejemplo, existen los direccionamientos indirectos con postincremento y con predecremento. En el caso de postincremento, la dirección de memoria que contiene al operando es el contenido de un registro especificado en la instrucción, y después de que ésta se ejecuta, el contenido de este registro es incrementado en una cantidad que es igual al tamaño del operando. El caso del direccionamiento con predecremento es similar al antes descrito. La diferencia estriba en que el contenido del registro es decrementado antes de la ejecución de la instrucción.

- DIRECCIONAMIENTO CON REGISTROS INDICES.

En este esquema, la dirección de la instrucción está compuesta por dos elementos. Uno es el número de un registro de CPU, el cual, a veces es llamado registro índice. El otro elemento es una constante. La dirección de la localidad de memoria en donde se encuentra el operando está dada por la suma de la constante y el contenido del registro índice.

El uso de este tipo de direccionamiento es ideal para el manejo de arreglos. Los registros índices hacen el papel de los índices de los vectores cuando se utilizan estos en los lenguajes de alto nivel. Generalmente la constante que se da en la instrucción es igual al valor de la dirección de memoria que contiene al primer elemento del arreglo. Entonces para seleccionar el elemento en posición N del vector, se multiplica N por el número de localidades que ocupa cada elemento (todos los elementos deben ser del mismo tamaño).

Un modo de direccionamiento derivado es el que permite que el contenido de los registros índices se incremente o decremente después o antes de su utilización. Esto es muy útil por ejemplo, para la suma o resta de vectores, en donde es necesario recorrer todo el arreglo.

El MC68000 tiene dos juegos de 8 registros en la CPU. Un juego se utiliza para cargar datos y el otro se utiliza para cargar direcciones. En el caso de esta máquina no se cumple el hecho de que los registros sean del mismo tamaño que la palabra de memoria (de 16 bits). Los registros son de 32 bits y por eso se ha dicho algunas veces que ésta es una máquina de 32 bits de palabra.

- DIRECCIONAMIENTO INDIRECTO.

Con este modo de direccionamiento, la instrucción contiene un número de registro de CPU o bien una dirección de memoria, en donde se especifica una dirección de memoria cuya localidad contiene al operando que la instrucción requiere. Al número que está contenido en el registro de CPU o en la localidad de memoria es llamado apuntador.

Existen muchos modos de direccionamiento derivados de éste. Por ejemplo, existen los direccionamientos indirectos con postincremento y con predecremento. En el caso de postincremento, la dirección de memoria que contiene al operando es el contenido de un registro especificado en la instrucción, y después de que ésta se ejecuta, el contenido de este registro es incrementado en una cantidad que es igual al tamaño del operando. El caso del direccionamiento con predecremento es similar al antes descrito. La diferencia estriba en que el contenido del registro es decrementado antes de la ejecución de la instrucción.

- DIRECCIONAMIENTO CON REGISTROS INDICES.

En este esquema, la dirección de la instrucción está compuesta por dos elementos. Uno es el número de un registro de CPU, el cual, a veces es llamado registro índice. El otro elemento es una constante. La dirección de la localidad de memoria en donde se encuentra el operando está dada por la suma de la constante y el contenido del registro índice.

El uso de este tipo de direccionamiento es ideal para el manejo de arreglos. Los registros índices hacen el papel de los índices de los vectores cuando se utilizan estos en los lenguajes de alto nivel. Generalmente la constante que se da en la instrucción es igual al valor de la dirección de memoria que contiene al primer elemento del arreglo. Entonces para seleccionar el elemento en posición N del vector, se multiplica N por el número de localidades que ocupa cada elemento (todos los elementos deben ser del mismo tamaño).

Un modo de direccionamiento derivado es el que permite que el contenido de los registros índices se incremente o decremente después o antes de su utilización. Esto es muy útil por ejemplo, para la suma o resta de vectores, en donde es necesario recorrer todo el arreglo.

Otro modo de direccionamiento derivado del indexado es el que utiliza el NS16000 para accesar datos de un arreglo en una sola instrucción. Al utilizar este modo de direccionamiento, la dirección del operando se calcula en base al contenido de un registro de propósito general de la CPU y a un segundo método de direccionamiento. El valor del registro se multiplica por una cierta cantidad (que depende del tamaño de los elementos del arreglo). La dirección obtenida con el segundo modo de direccionamiento es sumada a la del contenido del registro ya multiplicada, y el resultado es la dirección del operando.

- DIRECCIONAMIENTO POR MEDIO DE REGISTRO BASE.

Esta forma de direccionamiento es similar a la que funciona con registros índice. El operando se obtiene de la localidad con la dirección que da la suma de una constante llamada desplazamiento con el contenido de un registro llamado base. El desplazamiento y el número de registro base se dan en la instrucción. La diferencia entre estos modos de direccionamiento estriba en que el lugar designado en la instrucción para el desplazamiento, en el modo con registro base, no tiene que tener los suficientes bits como para contener una dirección de memoria. En el modo de direccionamiento con registros índices, la constante tiene que especificar una dirección de memoria. Por otro lado, los registros base deben ser capaces de almacenar una dirección de memoria, mientras que los registros índices, no necesariamente.

Este modo de direccionamiento provee de una forma que facilita direccionar estructuras que pueden estar localizadas en diferentes partes de la memoria. A un registro base se le puede cargar la dirección del principio de la estructura, y los elementos de la estructura pueden ser direccionados por su desplazamiento a partir del principio. Copias diferentes de la misma estructura pueden ser accesadas con solo cambiar el contenido del registro base.

- DIRECCIONAMIENTO CON REGISTROS BASE E INDICE.

En el procesador Z8000, utilizando este modo de direccionamiento, la dirección del operando se da por la suma de los contenidos de dos registros que se especifican en la instrucción. En los procesadores 8086 y MC68000 el operando se encuentra en la dirección que se genera como resultado de la suma de los contenidos de dos registros (base e índice) de CPU y de un desplazamiento.

Este modo de direccionamiento da una manera fácil de que un procedimiento pueda direccionar un arreglo que se encuentra dentro de un stack. El registro base puede contener una

constante que es una referencia en el stack (por ejemplo, la dirección del tope del stack después de que el procedimiento ha salvado registros y separado almacenamiento local). La diferencia entre el inicio del arreglo y el punto de referencia puede ser expresado por medio del desplazamiento, y el registro índice puede ser usado para accesar elementos específicos del arreglo. Este tipo de direccionamiento puede también ser muy útil para accesar de manera fácil vectores que se encuentren dentro de estructuras y de matrices.

- DIRECCIONAMIENTO RELATIVO AL CONTADOR DE PROGRAMA.

Cuando se utiliza este modo de direccionamiento, la dirección del operando se obtiene con la suma de la dirección que posee el PC y un desplazamiento que viene especificado en la instrucción. El Z8000 permite que el desplazamiento pueda ser restado al PC.

El MC68000 posee un modo derivado de este que se llama direccionamiento relativo al PC con índice y desplazamiento. En este caso la dirección es la suma del PC con un desplazamiento y con el contenido de un registro índice. Estos dos últimos se especifican en la instrucción.

- DIRECCIONAMIENTO IMPLICITO.

Algunas instrucciones hacen referencia implícitamente a sus operandos. Estos operandos generalmente son registros tales como el PC, el SP, o el registro de banderas. En el Z8000, en las instrucciones de manejo de cadenas y las funciones de autoincremento y autodecremento, el modo de direccionamiento está implícito. Cuando una instrucción usa un registro que contiene la dirección del operando, el registro es incrementado o decrementado después de la instrucción.

En el 8086 las instrucciones de manejo de cadenas no utilizan modos de direccionamiento de memoria para accesar sus operandos. En lugar de eso, los registros índices se utilizan de manera implícita. Cuando una instrucción de éstas es ejecutada, un registro especial apunta al primer elemento de la cadena fuente y otro registro apunta al primer elemento de la cadena destino. Cuando la instrucción es de repetición, estos registros son automáticamente ajustados para obtener los elementos subsecuentes de la cadena.

En el 8086 muchas instrucciones que operan sobre memoria asumen que la dirección a ser accesada se encuentra en el registro BL de la CPU.

Las operaciones que se utilizan para accesar un stack también tienen direccionamiento implícito. Por ejemplo, al ejecutar una operación aritmética básica sobre el stack, se asume que

uno de los operandos es el elemento que está en el tope del stack y el otro operando es el elemento que está en el nivel inmediato inferior al tope. Estos elementos son extraídos del stack y el resultado de la operación aritmética se coloca en el tope de éste.

- DIRECCIONAMIENTO RELATIVO A MEMORIA.

Este modo de direccionamiento se utiliza especificando en la instrucción dos desplazamientos. Al primero de ellos se le suma el contenido de un cierto registro. La dirección obtenida de esta suma se utiliza para tomar de memoria una palabra de información, la cual es sumada al segundo desplazamiento especificado en la instrucción. Esta instrucción es útil para manipular los campos de una estructura de datos llamada registro (record). Esta estructura contiene un cierto número de elementos, a los cuales se les llama campos, que pueden ser de diferente tipo (enteros, caracteres, etc.) y de diferente tamaño. Son muy usuales en algunos lenguajes de alto nivel.

Cuando este modo de direccionamiento se utiliza para accesar este tipo de estructura de datos, el segundo desplazamiento especifica la localidad del campo en el registro apuntado por la palabra de información que se obtiene de la memoria. El tamaño del campo se especifica en la instrucción.

- DIRECCIONAMIENTO EXTERNO.

Este modo de direccionamiento, que puede ser encontrado en el NS16000, se utiliza para accesar operandos que están en módulos diferentes al que se está ejecutando.

En algunos nuevos lenguajes de alto nivel tales como Ada, se ha empezado a manejar el concepto de módulo. Cada módulo puede ser desarrollado independientemente de otros módulos, los cuales, a fin de cuentas, pueden ser combinados para una ejecución conjunta.

Asociada con cada módulo, existe una tabla que contiene las direcciones absolutas de variables externas, y las direcciones relativas de procedimientos externos. Entonces con este modo de direccionamiento se especifican dos desplazamientos. Uno de ellos es el número de la variable externa y el otro es el desplazamiento a un subcampo de esta variable. Esto último podría ser, por ejemplo, el campo de un registro (record).

- DIRECCIONAMIENTO DE TOPE DEL STACK.

Cuando se utiliza este modo, el apuntador al stack (SP) en uso (en muchas máquinas existen dos modos de operación: de usuario y de supervisor, y cada modo posee su propio stack) especifica

la dirección en donde se encuentra el operando. Dependiendo de la instrucción, el SP será incrementado o decrementado permitiendo que las operaciones PUSH y POP se ejecuten normalmente. Este modo de direccionamiento permite que los operandos de muchas instrucciones sean obtenidos del stack.

1.3.2.5. TIPOS DE INSTRUCCIONES.

A las instrucciones se les puede clasificar por el tipo de función que el procesador ejecuta cuando las detecta. En la mayoría de las computadoras, cuando se describe su arquitectura, se agrupan a las instrucciones por tipos. Casi siempre es posible encontrar cinco tipos:

Instrucciones de transferencia de datos

Instrucciones aritméticas

Instrucciones lógicas

Instrucciones de control de programa

Instrucciones de entrada y salida

Recientemente ha aparecido un nuevo tipo de instrucciones. Estas son las instrucciones de apoyo a los lenguajes de alto nivel.

- INSTRUCCIONES DE TRANSFERENCIA DE DATOS.

La operación de transferencia de datos es la más usual entre las instrucciones que se le dan a una computadora. Es tan usual, que en la mayoría de los lenguajes de alto nivel, no existe una palabra reservada para la instrucción, de manera que se le ahorra al programador el tener que escribir demasiado. Normalmente se escribe esta instrucción, en estos lenguajes, como una operación con operador infijo. Se le llama asignación y el operador es un signo de igual, o bien dos puntos seguidos del signo de la igualdad (= o :=).

En el nivel de lenguaje máquina convencional existen varias instrucciones de transferencia de datos. Esto es porque existen diferentes medios de almacenamiento manipulables a

este nivel. Estos medios de almacenamiento son: la memoria principal, los registros y el stack. Es conveniente aclarar que aunque el stack se encuentra físicamente en la memoria o en los registros, las operaciones de acceso a él son completamente diferentes a las operaciones de acceso a la memoria o a los registros.

A la más general de las instrucciones de transferencia se le llama MOVER. Aunque realmente lo que hace esta instrucción no es mover, sino que copiar información. Cuando movemos una cosa, lo que en realidad hacemos es cambiarla de lugar, y eso no es lo que sucede con la información en una computadora cuando se utiliza la instrución MOVER. Lo que sucede es que la información permanece en su lugar de origen, mientras que una copia idéntica es transmitida y permanece en el lugar destino. O sea que en realidad copiamos información, por lo que la instrucción debería llamarse COPIAR. Pero el término MOVER ya está totalmente generalizado.

En las instrucciones de transferencia de datos es importante especificar tres cosas. Si no especificarlas en la instrucción, pues a veces están implícitas, si hay que tenerlas en cuenta. La primera es el objetivo a copiar o movilizar. Hay que saber cual es su dirección, si es que está en memoria, o bien en que número de registro se encuentra. La otra posibilidad es que lo incluyamos como parte de la instrucción. El segundo punto importante es el destino, esto es, a donde se pasará la información. Otra vez, puede ser a alguna localidad de memoria, o a algún registro de la CPU. El tercer punto importante es cuanta información se va a copiar. Generalmente en una instrucción se moviliza implícitamente toda una palabra de información, o todo un registro de CPU (por cierto casi siempre son estos del mismo tamaño en una misma computadora), pero es posible copiar un solo bit de algún lugar, o también es posible copiar una buena porción de información de la memoria principal.

Normalmente a las operaciones que se hacen para poner datos en la memoria se les llama ALMACENAR (STORE). Y a las operaciones que se utilizan para transferir información a los registros se les llama CARGAR (LOAD).

Otras instrucciones de transferencia de datos son el SET, cuya función es la de prender todos los bits del operando (ponerlos en uno), y el RESET, el cual apaga los bits del operando (los pone en cero).

En muchos procesadores existe la instrucción INTERCAMBIA (EXCHANGE) con dos operandos, la cual permite que se intercambien los contenidos de los operandos. En el procesador Z80, esta instrucción actúa sobre registros especiales. Aparte de los conocidos registros de propósito

general, existe en su CPU un juego de registros auxiliares idéntico a estos, a los cuales solo se tiene acceso con la instrucción EXCHANGE. Esta instrucción pasa el contenido de todos los registros de propósito general a los registros auxiliares y la información que está en estos, pasa a los registros de propósito general. De manera que el programador solo puede tener acceso a un juego de registros.

En el procesador MC68000 existe la instrucción SWAP que intercambia el contenido de las dos mitades de 16 bits de cualquiera de los registros programables de CPU.

También a las operaciones sobre el stack se les podría considerar como intrucciones de transferencia de datos. Estas son el PUSH y el POP. La instrucción PUSH almacena el operando (o bien el contenido de la localidad cuya dirección se especifica en el operando) en el stack. La instrucción POP saca una palabra del stack y la almacena en la dirección especificada por el operando. Dado que un stack es una estructura de datos del tipo Ultima-Entrada-Primera_salida, o LIFO (Last-In-First-Out), la información que se almacenó en el stack con el último PUSH que se utilizó, será la que se obtendrá con el próximo POP.

En algunos procesadores tales como el Intel 80186, es posible utilizar las instrucciones PUSHA y POPA. PUSHA almacena todos los registros de CPU dentro del stack. POPA los restaura en sus lugares correspondientes. El procesador 8080 posee las instrucciones PUSH PSW y POP PSW, donde PSW es la palabra de estado de programa.

- INSTRUCCIONES ARITMETICAS.

Todas las computadoras de tamaño regular poseen en su conjunto de instrucciones las cuatro operaciones aritméticas básicas: suma, resta, multiplicación y división. En los microprocesadores pequeños no se inplementaron la multiplicación y la división por razones de costo. Las microcomputadoras de 16 bits ya poseen estas instrucciones.

Las operaciones de punto flotante siguen siendo un tanto caras aún para los microprocesadores nuevos. Aunque el procesador NS16000 de 32 bits ya las puede ejecutar. Y el MC68020, también de 32 bits, tiene una interfase para un coprocesador de punto flotante.

Existen dos tipos de operaciones aritméticas. Las de un operando y las de dos. Las operaciones básicas son de dos.

Las operaciones de sumar y de restar tienen una modalidad que es la de ACARREO y de PRESTAMO respectivamente. La suma con ACARREO toma en cuenta que en la suma efectuada anteriormente,

haya habido acarreo (verificando la bandera de acarreo del registro de estado del programa). De esta manera se pueden hacer sumas de números con más dígitos de lo que normalmente acepta la unidad aritmética y lógica de la CPU. La resta con PRESTAMO funciona con la misma idea y también se puede utilizar para hacer rutinas para restar números grandes.

Instrucciones aritméticas que podamos encontrar de un operando son INCREMENTAR y DECREMENTAR, donde el operando es un acumulador o bien el contenido de una localidad de memoria. Estas instrucciones lo único que hacen es sumar o restar un uno del operando.

Otra operación de un operando es ABSOLUTO, instrucción cuya especialidad es remplazar el operando por su valor absoluto. Hay que mencionar también a NEGACION, la cual cambia el signo del operando.

Existe otra clase de instrucciones aritméticas. Estas, por no ser de resultados aritméticos, se podría decir que son de utilería. Ejemplos de estas instrucciones son las que utiliza Intel en el 8086, o Motorola en el MC68000, para cambiar dígitos de 8 bits a 16 bits, o de 16 a 32 bits, de manera que se puedan utilizar como dividendos para la división. Esto lo hacen cambiando de lugar el signo. Otras instrucciones de utilería podrían ser las que sirven en el 8086 para hacer ajustes en la suma, la resta, la multiplicación y la división, esto es, cambiar el contenido de un registro a un número decimal válido. También es posible que un registro de 8 bits pueda ser convertido en dos números en código BCD (Binary Coded Decimal).

- INSTRUCCIONES LOGICAS.

Aún cuando son muchas las operaciones lógicas booleanas que podría ejecutar una computadora sin mayores complicaciones en su diseño, solo cuatro instrucciones de este tipo existen en la mayoría de los conjuntos de instrucciones. Estas son: NOT, AND, OR INCLUSIVO y OR EXCLUSIVO. El NOT, a diferencia de la negación aritmética, lo que hace es invertir los bits del operando, esto es, forma el complemento a uno de éste. El NOT es la única instrucción de éstas que solo tiene un operando.

La instrucción AND hace una operación bit a bit entre sus dos operandos. Pone un cero en el bit correspondiente del resultado si cualquiera de los bits correspondientes de los operandos es cero, y pone un uno, si los dos bits en los operandos están prendidos. El AND se puede utilizar para aislar bits en un byte o en una palabra.

La instrucción OR INCLUSIVO es también bit a bit, y da un cero en el resultado solo si los dos bits correspondientes en los

operandos son cero. De otra manera da unos. Esta instrucción es utilizada a veces para unir grupos de bits sin afectar la información. El OR EXCLUSIVO da un cero dentro del resultado si los bits correspondientes en los operandos son iguales y da un uno si son diferentes. El OR EXCLUSIVO puede ser utilizado para obtener la paridad de un grupo de bits.

A veces se ha incluido a la operación de EQUIVALENCIA en algunos procesadores. El esultado de esta operación es igual a la negación del OR EXCLUSIVO.

Otra instrucción de tipo lógico es la comparación. Esta instrucción tiene también dos operandos con los cuales la unidad aritmética y lógica hace una resta pero sin almacenar el resultado en ningún lugar accesible para el programador. Simplemente dependiendo de este resultado, afecta al registro de banderas, el cual puede ser verificado después para tomar alguna acción.

El corrimiento y la rotación son consideradas también operaciones de tipo lógico. La instrucción de corrimiento funciona en dos versiones: corrimiento a la izquierda y corrimiento a la derecha, y lo que hace es lo siguiente: si el corrimiento es a la derecha, el bit más significativo del operando se pasa una posición a la derecha y lo mismo sucede con todos los demás. Esto quiere decir que se pierde el bit menos significativo. El bit más significativo se vuelve cero o uno dependiendo de la versión de corrimiento que se halla ejecutado. Hay dos posibilidades. Una es que se haya hecho corrimiento con extensión de signo, el cual es a veces llamado también corrimiento aritmético. En este caso el bit más significativo queda en el mismo estado que se encontraba el bit de signo del operando original. La otra posibilidad es que el corrimiento haya sido sin extensión de signo (se le llama a veces corrimiento lógico), en cuyo caso el bit más significativo siempre será cero. Un corrimiento a la derecha es equivalente a una división entre dos.

En el corrimiento a la izquierda siempre el bit más significativo se corre a la izquierda quedando un cero en su lugar. Todos los demás bits se corren a la izquierda también. De esta manera se pierde el bit más significativo. El corrimiento a la izquierda puede ser equivalente a la multiplicación por dos.

Es posible que en algunos procesadores la instrucción de corrimiento tenga dos operandos, donde el primer operando es la información a correrse y el segundo, es la cantidad de veces que se hará el corrimiento. Esto sería equivalente a la multiplicación o a la división por 2 a la N, donde N es el segundo operando.

Las instrucciones de rotación son similares a las de corrimiento. La diferencia estriba en que en el corrimiento a la derecha el bit más significativo quedaría vacante (dependiendo del caso), y el menos significativo se perdería. En la rotación a la derecha, el bit menos significativo pasa a ocupar el lugar del más significativo. La situación es análoga en la rotación a la izquierda. En instrucciones de rotación también es posible especificar como operando el número de rotaciones que se deseen. Algunos procesadores permiten que la bandera de acarreo (de la palabra de estado) sea tratada como parte de la información a rotar, esto es, en la rotación a la izquierda, que su valor es rotado al lugar del bit menos significativo y su lugar es ocupado por el bit más significativo del operando. En la rotación a la derecha, el valor de la bandera de acarreo pasa a donde el bit más significativo y su lugar adquiere el valor del menos significativo del operando. Las instrucciones de rotación pueden ser utilizadas para empacar o desempacar secuencias de bits que se encuentren en bytes, palabras, etc.

- INSTRUCCIONES DE CONTROL DE PROGRAMA.

A este tipo pertenecen todas aquellas instrucciones que son capaces de hacer que un programa desvíe su curso normal, esto es, lo hagan detenerse, o impidan que continúe la ejecución secuencial de las instrucciones, etc.

Este tipo de instrucciones podría ser dividido en cuatro grupos: las de bifurcación o salto, las de llamada a subrutinas, las de interrupciones, y las de control del procesador.

Las de bifurcación o salto vienen en dos grupos: las condicionales y las incondicionales.

Las incondicionales lo único que hacen es modificar el PC (contador de programa) sin verificar ninguna condición antes. Este tipo de instrucciones puede no tener operandos, lo que indica que la dirección que tomará el PC está en algún lugar inplícitamente especificado. Por ejemplo, en el Intel 8080, la instrucción PCHL indica al procesador que mueva el contenido del registro HL al PC.

Las instrucciones de salto condicional cambian el contenido del contador de programa solo si se cumple la condición que verifican. Existen dos instrucciones por cada bandera de condición de la PSW (aunque no siempre todas las banderas son de interés). Una instrucción ejecuta el salto si la bandera está apagada en el momento de la ejecución de la instrucción. Otra instrucción ejecuta el salto si la bandera está prendida.

Existen algunas instrucciones que evalúan funciones de verdad

cuyas variables son banderas de la PSW. Dependiendo del resultado (verdadero o falso) se ejecuta o no el salto.

Las banderas que generalmente se verifican son la de acarreo, la de overflow, la de cero, la de signo y la de paridad.

El salto se hace a la dirección indicada por el operando de la instrucción. Si no existe operando, entonces la dirección en donde está la instrucción a ejecutarse en caso de que se cumpla la condición, está en algún lugar fijo de la memoria o en algún registro de CPU.

La siguiente tabla contiene la descripción de las instrucciones de salto condicional del Intel 8086. La mayoría de los procesadores entienden un conjunto similar.

INSTRUCCION	CONDICION QUE SE VERIFICA	EJECUTA SALTO SI...
JA	(AF or CF)=0	más alto
JAE	AF=0	arriba o igual
JB	AF=1	abajo
JBE	(AF or CF)=1	abajo o igual
JC	AF=1	acarreo
JE	CF=1	igual
JG	((SF xor OF) or CF)=0	mayor
JGE	(SF xor OF)=0	mayor o igual
JL	(SF xor OF)=1	menor
JLE	((SF xor OF) or CF)=1	menor o igual
JNC	AF=0	no acarreo
JNE	CF=0	no igual
JNO	OF=0	no overflow
JNP	PF=0	no paridad
JNS	SF=0	no signo
JO	OF=1	overflow
JP	PF=1	paridad

JS	SF=1	signo

AF es la bandera de acarreo, CF la de cero, OF la de overflow, SF la de signo, y PF la de paridad. Arriba y abajo se refieren a la relación de dos valores sin signo, mientras que mayor y menor se refieren a la relación de dos valores con signo. Estas instrucciones generalmente están después de una instrucción de comparación (la cual actualiza banderas) que contiene dos operandos, de modo que el salto se ejecuta o no dependiendo de estos dos operandos.

Otras instrucciones de salto condicional no verifican banderas. Simplemente analizan un cierto operando, de manera que si éste es menor, igual o mayor que cero, se ejecuta el salto.

Existen otras dos instrucciones de salto en algunos procesadores que debemos mencionar. Ambas pueden ser condicionales o incondicionales. Una permite la ejecución de alguna instrucción que se encuentra en memoria. El operando indica en que localidad se encuentra la instrucción. Después de ejecutarse ésta, el programa continúa su proceso normal. El PC realmente no se altera con esta instrucción. El nemónico para esta instrucción en el Sistema IBM 370 es EX.

La otra ejecución a la que nos referíamos se llama SKIP. Esta instrucción hace que al contenido del PC se le sume alguna cantidad (que a veces puede ser especificada por el programador a manera de operando), de manera que el procesador deje de ejecutar una o más instrucciones.

El DECsystem 10 (PDP-10) tiene una gran cantidad de instrucciones que verifican diversas condiciones de modo que si se cumplen, causan bifurcaciones. Son tantas, que es difícil de creer que todas se han necesitado utilizar alguna vez.

El siguiente grupo de instrucciones del tipo de control de programa es el de llamadas a subrutinas.

Una subrutina, a la cual también se le llama procedimiento (sobre todo en los lenguajes de alto nivel), es un grupo de instrucciones capaces de ejecutar algún trabajo, de manera que puedan ser llamadas desde muchas partes del programa. A principios de la década de los setentas se empezó a hacer muy popular una técnica de programación introducida por Hoare y Dijkstra que hasta la fecha es muy importante por las facilidades que da al programador para la creación y mantenimiento de programas. Se llama Programación Estructurada. El concepto más importante en esta técnica es el de procedimiento.

Las instrucciones de este tipo, que se encuentran en todos los juegos que manejan subrutinas son CALL y RETURN. La instrucción CALL tiene un operando que es la dirección de la primera instrucción a ejecutarse en la subrutina. El control del programa se transfiere a esta dirección. Pero una vez terminada la ejecución de la subrutina, el control del programa debe regresar a la instrucción que está después que la de CALL. El regreso del control a esta instrucción lo hace el procesador cuando detecta la instrucción RETURN. En este proceso existe un detalle que es importante considerar. Este es, el lugar en donde se almacena la dirección de retorno. Probablemente el lugar más adecuado sea el stack. Esto permite el anidamiento de subrutinas, esto es, que dentro de una subrutina se pueda llamar a ejecución a otra, y dentro de esta otra, a otra, y así sucesivamente. Esto se logra gracias a que la última dirección en guardarse es la primera en recuperarse (por definición de stack).

Probablemente la ventaja mayor del almacenamiento en stack, es que permite la recursión, esto es, que un procedimiento se pueda llamar a sí mismo. El concepto de recursión se ha vuelto sumamente importante en la Computación. Además es la base de los lenguajes de alto nivel descriptivos, los cuales son los que se utilizan en Inteligencia Artificial. Y la Inteligencia Artificial es ahora uno de los principales puntos de investigación ahora en la Computación.

Tanto el CALL como el RETURN, pueden ser condicionales. En el procesador Z80, por ejemplo, están de esta manera.

El siguiente grupo a considerar dentro de las instrucciones de control de programa es el de las interrupciones. En realidad son bastante parecidas a las del grupo de llamadas a subrutinas. Permiten que las rutinas de servicio a interrupciones sean activadas desde los programas. El procesador MC68000 posee la instrucción TRAPV, la cual hace que se ejecute una rutina de error en caso de que la bandera de overflow esté encedida. En el procesador 8086 existe una instrucción similar llamada INTO.

También el 8086 tiene una instrucción con un operando llamada INT. Esta instrucción activa el procedimiento de interrupción especificado por el operando. Mientras que otra instrucción, IRET transfiere el control del programa, de regreso al punto de interrupción. En el procesador 8080 existe una instrucción llamada RST que tiene un operando. Este operando es un entero mayor o igual a 0, y menor o igual a 7. La instrucción permite la utilización de los primeros 64 bytes de la memoria principal a manera de subrutinas de 8 bytes cada una. Se hablará más sobre interrupciones en una sección posterior.

El último grupo entre las instrucciones de control de programa están las de control del procesador.

Entre estas instrucciones podemos citar a la única que debe estar en todo programa; la que sirve para detener al procesador. En muchos conjuntos de instrucciones se llama HALT.

Una instrucción que también detiene al procesador y que generalmente se llama WAIT, causa que la CPU se quede en estado de espera mientras cierta condición se verifica contínuamente en el exterior. Cuando la codición se cumple (si es que llega a cumplirse) debe llegar un aviso a la CPU por alguna línea. En este momento el programa continúa su ejecución. Si la condición nunca se cumple el procesador tampoco continuará trabajando.

La instrucción NOP no detiene al procesador, pero tampoco le ordena ejecutar acción alguna. Se utiliza a veces para rellenar espacios dentro de un programa. Estos espacios son utilizados para almacenar información durante la ejecución del programa.

Hay sistemas en los que es posible deshabilitar interrupciones. Para esto existe la instrucción DI (deshabilita interrupciones) y su contraparte para habilitarlas de nuevo.

En el grupo de instrucciones de control del procesador podríamos incluir a aquellas que dan al programador la facilidad de manipular las banderas de condición. Estas instrucciones simplemente ordenan prender, apagar, o complementar (si el bit esta en uno, ponerlo en cero, y viceversa) cualquiera de los bits que forman el registro de banderas.

En los procesadores en donde existen relojes o contadores manipulables por el programador, existen instrucciones para controlarlos, aunque hay que aclarar que en algunas máquinas esto se maneja por medio de interrupciones.

- INSTRUCCIONES DE ENTRADA Y SALIDA.

Los dispositivos de entrada y salida son los que hacen posible que la computadora se comunique con el ser humano. Además permiten que la capacidad de almacenamiento de la máquina sea prácticamente ilimitada. El manejo de los dispositivos lo hacen los procesadores mediante instrucciones de entrada y salida.

Hay procesadores cuyo conjunto de instrucciones de entrada y salida es sumamente sencillo. Un ejemplo es el Intel 8086 en donde solo existen dos instrucciones: IN y OUT. IN transfiere un byte o una palabra de información desde un puerto de entrada y salida a el acumulador. El número de puerto puede

ser especificado con una constante (lo cual permite acceso a los puertos numerados del 0 al 255), o con un número previamente colocado en otro registro de CPU (lo cual permite acceso variable a puertos numerados del 0 al 65535). La instrucción OUT funciona con las mismas opciones, pero de manera inversa. Esto es, transfiere un byte o una palabra del acumulador a un puerto de salida especificado por el programador. En este esquema solamente es posible mandar un byte o una palabra a o del dispositivo de entrada y salida.

Existen otros mecanismos en los que es posible transferir de memoria a los dispositivos de entrada y salida, y viceversa, un grupo de bytes o palabras por medio de una sola instrucción. Uno de estos mecanismos funciona a base de los procesadores de entrada y salida, de los cuales ya hemos hablado. La CPU indica al procesador de entrada y salida en donde se encuentra el programa que debe ejecutar para llevar a cabo la operación. Las instrucciones de estos programas consisten de un campo con el código de operación, el cual indica al procesador de entrada y salida, que acción ejecutar, por ejemplo, LEER o ESCRIBIR, y un campo que indica en donde se encuentra el bloque de información a ser transmitido y cual es el tamaño de éste.

Otro mecanismo con el que es posible transferir bloques, es el que trabaja a base de dispositivos con acceso directo a la memoria o DMA (Direct Memory Access). La CPU les indica en donde se encuentra el bloque de información en la memoria (si la operación es de salida), o bien, en que lugar de ésta se colocarán los datos (si la operación es de entrada). Entonces el dispositivo ejecuta la transferencia sin la intervención de la CPU. La CPU y el dispositivo solo interactúan cuando la CPU debe cederle el control del bus de memoria al dispositivo, cuando éste lo solicita.

Existen algunos procesadores que no tienen instrucciones explícitas de entrada y salida. Estos procesadores generalmente utilizan el esquema de mapeo de entrada y salida en memoria (memory mapped input output). En este esquema ciertas localidades fijas de memoria son asignadas para cada dispositivo de E/S. Algunas de las localidades son para control del dispositivo, y otras son para almacenar la información que se está transmitiendo. De manera que cuando hay necesidad de ejecutar alguna operación de entrada o de salida, la CPU almacena la información necesaria en estas localidades (datos y órdenes). El dispositivo constantemente está analizando las localidades que tiene asignadas, de modo que cuando detecta órdenes, las lleva a cabo. El procesador MC68000 maneja este esquema.

- INSTRUCCIONES PARA MANEJO DE LENGUAJES DE ALTO NIVEL.

Las instrucciones para facilitar la vida a los compiladores de lenguajes de alto nivel podrían encajar en tipos de instrucciones de los cuales ya hemos hablado.

En la mayoría de los lenguajes de alto nivel, excepto probablemente en el lenguaje C (aunque a veces no se le considera completamente de alto nivel), los subíndices de los arreglos son verificados para ver si no exceden el rango para el que fueron definidos. En el procesador NS16000 de National Semiconductors existe una instrucción para llevar a cabo esto. La función se llama CHECK.

En el NS16000 también existe la instrucción INDEX que sirve para facilitar el cálculo de las direcciones en memoria que ocupan los arreglos multidimensionales. Es claro que la memoria es un medio natural para almacenar vectores. Pero resulta un poco difícil seguir la pista de una matriz o de un arreglo de tres dimensiones almacenado en ella. Y las matrices son estructuras de datos que se manejan en la mayoría de los lenguajes de alto nivel.

El procesador 80186 de Intel puede ejecutar un par de instrucciones que permiten minimizar el trabajo extra que existe al traducir una llamada a procedimiento de un lenguaje de alto nivel a código de máquina. Estas instrucciones son ENTER y LEAVE y lo que hacen es manejar los recursos que deben ser asignados al comienzo de un procedimiento y reclamados al final de éste.

El manejar cadenas es un trabajo que siempre se había dejado a los lenguajes de alto nivel. Esta situación ha ido cambiando en últimas fechas. Ahora los lenguajes de bajo nivel, entre ellos los lenguajes máquina, tienen incluídas entre sus instrucciones, algunas que manipulan cadenas.

En el procesador 8086 por ejemplo, ciertas instrucciones llamadas primitivas, operan en cadenas de bytes o de palabras, un elemento a la vez. Hay instrucciones para mover y comparar cadenas, y también para buscar valores enteros en ellas. Estas instrucciones pueden trabajar con cadenas de hasta 128K bytes. Aunque hay que aclarar que las instrucciones deben ir precedidas por un prefijo especial que hace que las instrucciones sean repetidas por el hardware (pues trabajan sobre un solo elemento de la cadena, ya sea un byte o una palabra), procesando a las cadenas mucho más rápido de lo que se haría por medio de software. Las repeticiones terminan con alguna de las múltiples condicionales que existen. Estas instrucciones pueden ser interrumpidas o abortadas. Las abortadas pueden continuar después en donde se quedaron procesando a la cadena.

Dijimos antes que el concepto de procedimiento es muy

importante en la técnica de Programación Estructurada. El concepto de iteración también es importante. Tres instrucciones son utilizadas para el control de iteraciones en los lenguajes de alto nivel: REPEAT...UNTIL "condición" , DO WHILE "condición" , y FOR.

La instrucción REPEAT...UNTIL "condición" ejecuta cierto conjunto de instrucciones repetidamente hasta que se da cierta condición. Debido a que la condición está al final de las instrucciones que se están iterando, éstas se ejecutarán cuando menos en una ocasión. Y esto aún cuando la condición se cumpla antes de la ejecución de las instrucciones, lo cual es incorrecto. Esto indica que esta instrucción solo se debe utilizar si hay seguridad de que el conjunto de instrucciones a iterarse debe ser ejecutado al menos una vez.

La instrucción DO WHILE "condición" permite las iteraciones mientras se cumple la condición. Además ésta se verifica antes de la ejecución de cada iteración. Por esto, si la condición no se cumple desde el principio, entonces la iteración no será ejecutada ninguna sola vez.

La instrucción FOR se utiliza cuando se sabe exactamente cuantas veces hay que iterar. O sea que el proceso puede ser controlado simplemente con un contador.

En los lenguajes del segundo nivel (lenguaje máquina), las iteraciones se controlan con contadores o con el estado de las banderas de condición. Es decir, las instrucciones incrementarán o decrementarán el valor en cierto registro designado para el caso, de manera que verifican este registro en cada iteración (si su valor ha llegado a donde se deseaba), o verifican alguna bandera (si ha cambiado de estado despúes de alguna operación reciente), para ver si se debe ejecutar de nuevo el conjunto de instrucciones que se está iterando.

Las instrucciones tienen como operando una dirección de memoria, la cual es la localidad en donde se encuentra la primera instrucción del conjunto a iterar. Estas instrucciones se colocan al final del conjunto. Esta estructura se puede hacer equivalente a la estructura de la instrucción REPEAT...UNTIL "condición" , en el sentido de que se ejecuta al menos una vez.

Una estructura que sería equivalente a la de DO WHILE "condición" tiene código menos eficiente dado que la primera instrucción del conjunto a iterarse debe verificar la condición. Además de que debe haber una instrucción de bifurcación al final del conjunto. El operando de esta instrucción debe ser la dirección de memoria de la instrucción del conjunto.

1.3.2.6. INTERRUPCIONES.

Cuando la CPU se encuentra ejecutando las instrucciones de un programa, siempre existe la posibilidad de que surja alguna situación anormal. Cuando sucede esto, la CPU deja de ejecutar el programa en curso y se va a ejecutar otro programa.

Cada tipo de situación anormal tiene un diferente programa a ejecutarse en memoria, y la CPU posee un mecanismo para detectar cual fue el caso anormal que se dio, de manera que pueda ejecutar el programa correspondiente.

Este mecanismo funciona de manera semejante a la instrucción "case" de algunos lenguajes de programación; al ejecutarse cada instrucción del programa de aplicación, se verifica si ocurrió alguna de las situaciones inusuales pero previstas, y en caso afirmativo, el control se transfiere a donde se requiera, dependiendo del caso.

El programa que se ejecuta después de la condición anormal sirve para que la computadora pueda hacer algo respecto a la situación, y luego regrese a la ejecución del programa. Esto último, si la situación no fue fatal.

A este mecanismo se le llama "trampa".

Para que el programa de la "trampa" pueda ser ejecutado sin que afecte a la ejecución del programa de aplicación, todos los contenidos de los registros de trabajo de la CPU y la palabra de estado del programa se guardan en memoria principal, de manera que después se pueda proseguir con la ejecución del programa lo más normalmente posible.

En algunos lenguajes de máquina convencionales existen instrucciones que verifican por situaciones similares a las que ocasionan "trampas". Por ejemplo, el overflow es una causa de "trampa", pero en el procesador Intel 8086 puede verificarse por medio de la instrucción JO (Jump if overflow).

En algunos lenguajes de alto nivel existen instrucciones que funcionan de manera similar a las "trampas". Por ejemplo, existen versiones del intérprete del lenguaje BASIC que poseen la instrucción ON ERROR GOTO X. Esta instrucción permite que en el proceso de ejecución del programa, en las líneas de código posterior a esta instrucción, si se detecta cierto tipo de errores, el control del programa pase a la línea X.

Las "trampas" están sincronizadas perfectamente con el programa en ejecución, esto es porque éstas son efecto directo del programa. El problema es que algunas veces se dan situaciones no esperadas, externas al programa. Esto quiere decir que debe existir otro mecanismo para manejar los casos no sincronizados, puesto que también la CPU debe tomar acciones inmediatas para responder a la situación. A esta acción se le llama interrupción. Y a las situaciones que las causan se les llama condiciones de interrupción.

En algunas máquinas no existe diferencia entre las "trampas" y las interrupciones y los mecanismos para manejar a ambas son idénticos. En estos casos al mecanismo se le llama de interrupción. Es por eso que en algunas máquinas no se maneja el concepto de "trampa".

Algunas de las condiciones que producen "trampas" son las siguientes:

- Condiciones aritméticas. Este tipo de condición es el que se provoca debido a fallas en la ejecución de algunas instrucciones de tipo aritmético. El intento de dividir entre cero, o bien el overflow (que el producto de por ejemplo, una multiplicación, no quepa en la localidad designada para almacenar el resultado) son condiciones de este tipo. Como decíamos antes, algunos lenguajes poseen instrucciones para verificar algunas de estas condiciones. Pero al hacerlo a mano se consume más tiempo de procesador y se ocupa más memoria principal. La ventaja de hacerlo así es que el usuario puede escribir sus rutinas de acciones a tomarse como respuestas a las condiciones. Aunque hay que aclarar que muchas máquinas permiten que en la "trampa", el usuario especifique que acción tomar dependiendo de la condición.

- Condiciones de stack. Si el stack está completo por haberse efectuado demasiadas operaciones de PUSH en él, y existe un nuevo PUSH, se generará una "trampa" de este tipo. De igual manera, una operación POP en un stack vacío tiene resultados similares.

- Instrucciones ilegales. Cierto número de bits del grupo que conforman las instrucciones, corresponden al código de operación. El número de códigos que se pueden obtener de la combinación de estos bits, casi siempre es mayor al número de instrucciones que el procesador de la máquina puede ejecutar. Esto indica que es posible que al procesador le pueda llegar una palabra cuyo campo del código de operación no corresponda a ninguna instrucción. O sea, que se esté tratando de ejecutar una instrucción ilegal. El mecanismo de "trampa" abortará casi siempre la ejecución del programa si se da un caso de estos.

- Violación de protección de memoria. Dado que los sistemas de computación grandes tienen varios programas en la memoria principal que comparten la misma CPU, y los procesadores de entrada y salida para su ejecución, es necesario que existan esquemas de protección de memoria. Estos esquemas impiden que un programa invada áreas de memoria asignadas a otros programas. De esta manera, si un programa intenta semejante acción, generará una "trampa", que generalmente abortará la ejecución del programa.

- Condiciones de dispositivos de entrada y salida. En algunos sistemas es necesario asignar dispositivos de entrada y salida a cada programa en ejecución. De manera que si hay un intento de asignar un dispositivo que no existe o no está conectado, se generará una "trampa". De igual manera, si un programa intenta accesar un dispositivo que no le fue asignado, generará otra "trampa".

- Errores de transmisión. Algunas veces existen errores en la transmisión de información de la memoria al procesador. Existen diversos métodos para detectar estos errores. El más conocido es, probablemente, el del bit de paridad. De hecho el popular código ASCII, puede manejar el concepto de paridad. Este método consiste en contar el número de bits prendidos o apagados, contenidos en el grupo de bits que se está transmitiendo. Si este número de bits es par o impar, se especifica en un bit adicional que se envía con el grupo de bits. De esta manera, al llegar a su destino el grupo, se cuenta de nuevo el número de bits prendidos o apagados y se determina si el número es par o impar y se compara con el que se había calculado antes y que venía con el grupo. Este esquema asegura que si cambió algún bit en el camino, por algún error, se detectará. Y de esta manera, por medio de una "trampa" se puede pedir la retransmisión de la información. No es posible detectar de esta forma si hubo cambio en más de un bit.

Existen métodos muy complicados que permiten la detección de dos o más errores en la transmisión de bits. Estos métodos incluso permiten la corrección automática de errores sin necesidad de retransmisión. El problema es que requieren de más tiempo de chequeo y de muchos bits adicionales. El código de Hamming es un ejemplo de estos métodos.

Existen también varias condiciones de interrupción. Aquí describiremos algunas.

- Condiciones de entrada y salida. Cuando un dispositivo de entrada y salida fue activado por un programa en ejecución, empieza a funcionar. Una vez completada su función, avisa a la CPU que está listo para la siguiente misión. Dado que la CPU puede en ese momento estar ejecutando alguna otra tarea, y

no está esperando que el dispositivo de entrada y salida le avise de la terminación de su trabajo, esta señal es asincrona y por lo tanto es una interrupción a CPU. De idéntica manera, un error en un dispositivo de entrada y salida, genera una interrupción, de modo que si es posible, se repita la operación que falló.

- Relojes. Cuando existen múltiples programas compartiendo un procesador para su ejecución, es necesario que tengan cada uno, un intervalo de tiempo de ejecución, de manera que ninguno se adueñe del procesador. Para esto se utilizan los contadores, los cuales pueden ser programados para generar una interrupción cada determinado tiempo (Este tiempo se mide en micro o milisegundos). Estos contadores se utilizan también para generar interrupciones de otro tipo. A veces, por error de programación, es posible que un grupo de instrucciones iterando no cumpla nunca con una condición de terminación, de manera que el ciclo se ejecute infinito número de veces. Esto no es deseable, por lo que al iniciarse las iteraciones, se enciende un contador, de modo que, si se rebasa un tiempo determinado, se genera una interrupción para acabar con la ejecución del programa.

Existe otro tipo de reloj en las computadoras. Es el reloj de tiempo real, el cual lleva un registro de la fecha y la hora en todo momento. Existen instrucciones en algunas máquinas que permiten generar una interrupción cuando el reloj de tiempo real detecta alguna fecha y hora determinada.

- Sincronización. Cuando existen varios procesadores en una misma computadora, es necesario que algunas veces estos se sincronicen, es decir, que ejecuten tareas no independientemente. Entonces es posible que un procesador genere una interrupción para que sea recibida por otro o por otros procesadores.

Las computadoras tienen diferentes maneras de responder a las instrucciones de entrada y salida. Por ejemplo, la IBM 370 maneja una PSW (de la cual hemos hablado antes, pues es igual a la del Sistema 360). Cuando un procesador de entrada y salida genera una interrupción, en el código de interrupción de la PSW, se coloca el número de dispositivo que la causó. Entonces la PSW se almacena en cierta localidad de memoria y una nueva PSW se carga desde otra localidad. Entonces la CPU empieza la ejecución de una rutina de servicio a interrupciones general, la cual averigua que dispositivo causó la interrupción (verificando la PSW que salió) y luego llama a la rutina especializada para manejarla. Por último un bit de máscara (existe uno por cada procesador de entrada y salida) deshabilita interrupciones de ese mismo procesador hasta que se termine de procesar la anterior.

En la PDP-11, la PSW y el contador de programa (PC) se almacenan en un stack cuando un dispositivo de entrada y salida genera una interrupción. Entonces otra PSW y otro PC se cargan a la CPU desde cierta localidad de memoria asociada con el dispositivo que causó la interrupción. A estas direcciones se les llaman vectores de interrupción y cada dispositvo tiene uno propio. De igual manera, cada dispositivo tiene un número de prioridad al igual que la CPU, de forma tal que si la prioridad del dispositivo es mayor que la de la CPU, entonces la interrupción requerida por el dispositivo puede llevarse a efecto. De otra forma, el dispositivo tiene que esperar que la prioridad de la CPU disminuya. Este mecanismo permite que solo interrupciones de más alta prioridad puedan afectar el proceso de la que se está llevando a cabo en un momento dado. Los dispositivos que tiene más alta prioridad son aquellos que deben ser atendidos de manera inmediata. Las prioridades varían de 0 a 7, siendo la número 7 la más alta.

En el procesador 360 existen dos niveles de interrupción: interrupciones enmascarables e interrupciones no enmascarables. Las enmascarables pueden ser deshabilitadas con la instrucción PI, mientras que las no enmascarables no pueden ser deshabilitadas. En el nivel enmascarable existen tres modos de funcionamiento. En el modo 0, el dispositivo que genera la interrupción manda a la CPU la dirección de la siguiente instrucción a ser ejecutada. En el modo 1, la siguiente instrucción a ser ejecutada se encuentra en una dirección fija de memoria. Por último, en el modo 2, que es el más poderoso, el programador mantiene una tabla de direcciones iniciales de cualquier rutina de servicio de interrupción. Los 8 bits más significativos de cada dirección provienen del registro I, el cual debe ser cargado con anterioridad por el programador, con el valor deseado. Los 8 bits menos significativos de la dirección provienen del dispositivo que generó la interrupción.

1.3.3. BUSES.

Para que las diferentes partes de una computadora (CPU, memoria, etc.) puedan formar un sistema capaz de ejecutar tareas, es necesario que estén conectadas entre si. Esta conexión se lleva a cabo por medio de conjuntos de cables o lineas llamadas buses.

Los buses están diseñados de forma tal que puedan transmitir entre los diferentes dispositivos del sistema, señales digitales agrupadas para un propósito común, por ejemplo, para la transmisión de una palabra de información entre la memoria y la CPU. Esto se lleva a cabo agrupando en paralelo una cierta cantidad de cables, de manera que cada cable transmita un bit, y el conjunto de cables transmitan una palabra de información.

Hay que aclarar que los buses no solo transmiten datos, sino también transfieren direcciones y señales de control. Es por eso que a las líneas que forman un bus se les clasifican como líneas de datos, de direcciones y de control.

Existen dos tipos de buses: los unidireccionales y los bidireccionales. Los unidireccionales transfieren señales en una dirección, mientras que los bidireccionales lo pueden hacer en ambas direcciones, aunque esto no simultáneamente.

Probablemente la manera más sencilla de conectar los dispositivos en un sistema es por medio de líneas que comuniquen a cada par de dispositivos de manera que cada línea solo tenga un lugar de donde salen las señales y un lugar a donde van. A este tipo de líneas se les llaman buses dedicados.

Cuando las líneas permiten la comunicación entre varios dispositivos, de modo que solo dos dispositivos se puedan comunicar en un momento dado, y los demás tengan que esperar a que se desocupe la línea, al sistema se le llama de buses compartidos.

Existen diferentes arquitecturas dependiendo de la manera en que los buses están distribuidos en una computadora. Una forma es la llamada estructuctura de dos buses, la cual tiene al menos dos versiones. En ambas versiones existe un bus de memoria y un bus de entrada y salida.

En una versión, la memoria y la CPU están conectadas por el bus de memoria, mientras que los controladores de entrada y salida y la CPU se comunican por el bus de entrada y salida. Los datos pasan a través de la CPU en su paso entre la memoria y los dispositivos de entrada y salida, de manera que es la CPU quien controla las operaciones de E/S, iniciándolas y siguiéndoles la pista durante su proceso.

Otra versión de arquitectura de doble bus es la que tiene al bus de memoria entre CPU y memoria igual que antes, pero el bus de E/S está entre la memoria y los controladores de E/S, de manera que no es la CPU quien se encarga de las operaciones de E/S. Esta función queda en manos de los conocidos procesadores de E/S.

La arquitectura más utilizada en las mini y microcomputadoras es la de un solo bus. En este esquema todos los dispositivos están conectados a un mismo bus. Esto implica reducción en los costos de arquitectura y flexibilidad para incluir dispositivos periféricos. La desventaja es que la velocidad de operación también se reduce.

Existen dos maneras en que los dispositivos transmiten señales por medio de los buses. Una es la comunicación síncrona y la otra es la asíncrona. La comunicación síncrona implica que cada grupo de señales se transmite en un periodo de reloj el cual es conocido de antemano por el dispositivo transmisor y por el dispositivo receptor. La sincronización se puede llevar a cabo o bien conectando los dispositivos a un mismo reloj, o bien conectándolos a relojes con la misma frecuencia. Este esquema es relativamente sencillo, pero la velocidad de transmisión está determinada por el dispositivo más lento.

En la otra forma de comunicación, llamada asíncrona, cada grupo de datos transmitiéndose es acompañado de una señal de control para indicar su presencia al dispositivo receptor, quien debe responder al dispositivo transmisor con otra señal de control.

Un dispositivo puede ser seleccionado para conectarse al bus compartido si la unidad de control de bus lo requiere en respuesta a alguna instrucción en un programa o a una condición que ocurra en el sistema, que requiera el servicio del dispositivo. Otra conexión funciona de modo que el mismo dispositivo manda una señal a la unidad de control de bus avisándole que requiere hacer uso del bus. Si son muchos los dispositivos que pueden generar solicitud de acceso al bus simultáneamente, la unidad de control de bus debe poseer un método para seleccionar al dispositivo que hará uso de las líneas.

A continuación revisaremos rápidamente tres de estos métodos: el de daisy chaining, el de polling y el de solicitud independiente.

El método de daisy chaining trabaja de la siguiente forma: la señal que cede el bus a un dispositivo, es transmitida de dispositivo a dispositivo de manera que cuando un dispositivo recibe esta señal y no necesita hacer uso del bus, la transfiere al siguiente dispositivo, pero si lo necesita bloquea la propagación de ésta. De esta manera el dispositivo con más prioridad es el que se encuentra alambrado físicamente más directamente a la unidad de control de bus. Este sistema es muy simple y no requiere de muchas líneas de control, pero si un dispositivo falla, los que están después de él, nunca tendrán acceso al bus. Además las prioridades de acceso solo pueden ser cambiadas de manera física (no con programas).

Cuando un dispositivo utiliza el método de polling, cada dispositivo tiene conectado a la unidad de control de bus un grupo de líneas, de modo que es ésta la que va cediendo el bus a los dispositivos de acuerdo a las prioridades que tengan. Esta prioridad normalmente puede ser programada o alterada en cualquier momento.

El otro método de selección de prioridad es el llamado de solicitud independiente. En este esquema cada dispositivo puede solicitar a la unidad de control de bus que se le permita el acceso a las líneas, de modo que esta decidirá en un momento dado, por medio de una tabla de prioridades, a quien le corresponderá hacer uso de ellas. La tabla es programable.

Para comprender más a detalle el funcionamiento de los buses, a continuación daremos una descripción del funcionamiento del sistema de comunicación de la mini PDP-11, el cual es llamado UNIBUS. Este sistema trabaja con un bus compartido de comunicación asíncrona, al cual están conectados la CPU, la memoria y los dispositivos de entrada y salida. Utiliza los métodos de daisy chaining y solicitud independiente para la selección de prioridades. El UNIBUS contiene 56 líneas (casi todas bidireccionales) de las cuales 16 son de datos, 18 de dirección y 22 de control.

A continuación se da la lista de líneas que componen al UNIBUS:

bus de datos ---- 16 líneas ----

bus de direcciones ---- 18 líneas ----

bus de control

---- 2 líneas ------ CONTROL

---- 1 línea ------- SINC MAESTRO

---- 1 línea ------- SINC ESCLAVO

---- 2 líneas ------ PARIDAD

---- 5 líneas ------ SOLICITUD DE BUS

---- 5 líneas ------ PERMISO DE ACCESO

---- 1 línea ------- CONOCIMIENTO DE SELECCION

---- 1 línea ------- INTERRUPCION

---- 1 linea ------ BUS OCUPADO

---- 1 linea ------ INICIALIZACION

---- 1 linea ------ LINEA AC BAJA

---- 1 linea ------ LINEA DC BAJA

En cualquier momento solo dos dispositivos pueden estar conectados entre si por medio del UNIBUS. A uno de estos se le llamará maestro pues tendrá el control del bus (aunque solo la CPU tiene el control total sobre él), mientras que al otro se le llamará esclavo. En una operación de lectura o escritura en memoria requerida por la CPU, es esta última el dispositivo maestro, mientras que la memoria es el dispositivo esclavo. En una operación de E/S entre memoria y una unidad de disco, la CPU cede el control del bus a la unidad de disco, quien se convierte en el dispositivo maestro.

Una operación de transferencia de datos de un esclavo a un maestro se llevaría a cabo de la siguiente manera: El dispositivo maestro coloca la dirección del esclavo en el bus de dirección y coloca también señales en las lineas de CONTROL para indicar que la transferencia de datos de un esclavo a un maestro se requiere. Todas las unidades que pueden ser esclavas tratan de decodificar el contenido del bus de direcciones. De éstas solamente una detectará su dirección, de manera que sabrá que es ella la esclava. Después de un tiempo el maestro activa la señal de SINC MAESTRO (sincronización de maestro) que indica que la transmisión de los datos debe a comenzar. Entonces el esclavo, al recibir esta señal, pone los datos a transmitirse en el bus de datos y activa la señal de SINC ESCLAVO (sincronización de esclavo). Cuando el maestro recibe esta señal, toma los datos del bus y desactiva la señal de SINC MAESTRO y después de un tiempo limpia el bus de dirección y las lineas de CONTROL. El esclavo responde a la desactivación de SINC MAESTRO limpiando el bus de datos y desactivando la señal de SINC ESCLAVO. Esto completa la transferencia.

Un dispositivo de E/S puede requerir del UNIBUS generalmente por una de dos situaciones. Cuando necesita transferir datos desde o hacia la memoria y cuando quiere mandar un comando de interrupción a CPU para que ésta ejecute alguna rutina de servicio.

Cada dispositivo que puede ser maestro está conectado a una linea de SOLICITUD DE BUS y a una de PERMISO DE ACCESO, de manera que cuando el dispositivo requiere del bus, activa la primera señal y la CPU activa la segunda. Si dos dispositivos

solicitan acceso al mismo tiempo, se le da permiso al que tenga mayor prioridad. Cuando el dispositivo solicitante detecta la señal de PERMISO DE ACCESO, responde desactivando la señal de SOLICITUD DE BUS y activando la de CONOCIMIENTO DE SELECCION, por la que la CPU al detectarla, desactiva la señal de PERMISO DE ACCESO. Si había una transmisión de datos en proceso, al terminar, la unidad seleccionada toma control del bus activando la señal de BUS OCUPADO. Esta unidad es ahora maestra y puede iniciar cualquier transferencia. También puede iniciar una secuencia de interrupción activando la señal de INTERRUPCION, la cual retorna el control a la CPU. El dispositivo deja el control del UNIBUS después de una transmisión de datos apagando las señales de BUS OCUPADO y de CONOCIMIENTO DE SELECCION. Si la CPU ha designado a otro dispositivo como maestro, entonces éste toma el control del UNIBUS y de esta manera el control es transferido de dispositivo a dispositivo, aunque hay que tener en cuenta que solo la CPU activa la señal de PERMISO DE ACCESO para designar quien puede ser el dispositivo maestro.

Cuando la CPU se encuentra ejecutando algún programa y además hay información que se está transmitiendo de memória a algún dispositivo de E/S o viceversa, la transmisión de un bloque de datos entre la memoria y el periférico se lleva a cabo cada vez por el UNIBUS después de la ejecución de un determinado número de instrucciones. El procesador debe dejar el UNIBUS y esperar a que se le devuelva después de la transmisión del bloque de información. A este proceso se le llama robo de ciclo.

Para terminar de hablar de los buses hay que aclarar que desde el punto de vista conceptual de la operación de las computadoras, la estructura de estos no tiene mucha importancia. Los buses son muy importantes si lo que nos interesa es el comportamiento (performance) del sistema (el cual se define vagamente como la medida de velocidad del sistema y uso de recursos). Veamos esto con ejemplos. El bus de datos del MC6800 es de 16 bits, mientras que el del MC68000 es de 8 bits. En todo lo demás los procesadores son idénticos. Esto quiere decir que la transmisión debe ser multiplexada en este último, lo que da como resultado que el MC68000 tenga 1.66 veces mejor "performance" que el MC6800, lo cual hace que éste no sea muy interesante. Un caso parecido se da en los procesadores de Intel 8086 y 8088. El primero tiene un bus de datos con 16 líneas, mientras que el bus del segundo, solo tiene 8 líneas.

2. DISEÑO DE LA ARQUITECTURA DEL PROCESADOR.

2.1. INTRODUCCION.

Se podría decir que éste es el capítulo más importante de este trabajo. Aquí se describe la arquitectura del procesador con propósito académico que se obtuvo como resultado de la revisión de las arquitecturas de los procesadores que han sido descritas en el capítulo anterior.

No se intenta que el procesador sea llevado a los circuitos lógicos ni a la microprogramación, puesto que no estaba considerado que el procesador fuera eficiente en el ahorro de memoria y en la velocidad. Esto se debió a que se puso más énfasis en la claridad del diseño, de manera que el estudiante de Programación de Sistemas entienda de una manera más estructurada, que es lo que puede hacer con el procesador.

Construir este procesador en hardware o en firmware es, sin embargo, posible. Esto quedará demostrado por medio de la construcción de un simulador del procesador.

Para que el procesador pueda ser llamado de alguna forma, escogimos un nombre: AD-24. Entre sus características más importantes, se encuentran las siguientes:

- 16 Megabytes de espacio de direccionamiento lógico.
- 18 registros de propósito general de 24 bits cada uno.
- Mapeo de memoria por paginación.
- Manejo de memoria virtual y multiprogramación.
- Manejo de lenguajes de alto nivel.
- 80 instrucciones básicas.
- 9 modos de direccionamiento.
- 2 modos de proceso (Supervisor y Usuario).

2.2. REGISTROS.

El procesador AR-24 posee 16 registros en la unidad central de proceso que pueden ser manipulados por el programador. Los registros están numerados del 0 al 15 y cada uno de ellos es de 24 bits.

Entre estos registros se encuentran el contador de programa (PC) y el apuntador de stack (SP). El PC esta en el registro 14 y el SP en el 15. Los otros 14 registros, que son los de propósito general, también pueden ser utilizados para las operaciones de punto flotante.

El PC contiene siempre la dirección de la siguiente instrucción a ejecutarse. El PC direcciona los bytes de la memoria, del byte 0 al byte 16,777,215. El SP puede apuntar a cualquiera de los dos stacks que se encuentran en memoria. Existe un stack para cada uno de los dos modos de operación del procesador. Estos modos son el de supervisor y el de usuario. Cuando hay un cambio de modo de operación, el procesador almacena el contenido del registro 15 (el SP del modo en curso) en un lugar de la memoria y recupera de otro lugar de ésta el SP del modo de operación a que está cambiando. El SP siempre contiene la dirección de la última localidad del stack ocupada en memoria. El SP es decrementado cuando los datos son metidos al stack (con la operación PUSH), e incrementado cuando los datos son sacados de éste (con la operación POP). Los stacks pueden estar en cualquier parte de la memoria y la única restricción en su tamaño es el tamaño de ésta.

Existe otro registro llamado de estado del programa o PSW que contiene las banderas de condición y algunos bits que indican el estado del procesador.

Las banderas de condición que tenemos son las siguientes: de acarreo, de overflow (desbordamiento), de signo y de cero.

La bandera de acarreo se prende cuando en las operaciones aritméticas de suma y de resta, existe un acarreo o un préstamo al bit más significativo del operando (que es el bit de signo). También se utilizada en las operaciones de corrimiento y de rotación.

La bandera de overflow se prende cuando el resultado de una operación aritmética no cabe en el espacio que debe utilizar como destino. En las operaciones normales el destino es de 23 bits de tamaño; en las de punto flotante, el destino del exponente es de 7 y el de la mantisa es de 15; en las operaciones con cadenas el destino es de 8 bits.

La bandera de signo se prende cuando el operando es negativo, es decir, cuando el bit más significativo está prendido.

La bandera de cero se prende cuando los 23 bits menos significativos del operando estén apagados.

Además tenemos a las banderas de control las cuales son tres: de rastreo, de modo de operación del procesador y de modo de operación de la memoria.

Cuando la bandera de rastreo está prendida, después de la ejecución de cada instrucción se genera un interrupción tipo 1. Esto implica que el PC y la PSW son almacenadas en el stack, la bandera de rastreo es apagada y el PC es cargado con el contenido de las localidades 2 a 5 de memoria. La siguiente instrucción del programa que se estaba ejecutando se procesará hasta que ocurra la instrucción RETTI.

La bandera de modo de operación del procesador indica, si está prendida, que el procesador se encuentra en modo de supervisor, y si está apagada, en modo de usuario. El modo de supervisor sirve para que sea posible la ejecución de las instrucciones privilegiadas entre las que tenemos a PARA, LEE, ESC, etc.

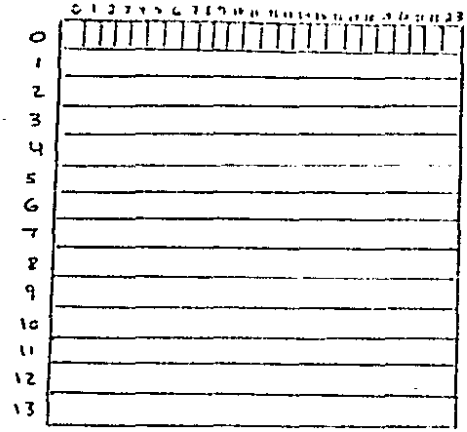
La bandera de modo de operación de la memoria indica, cuando está prendida, que el procesador debe mapear las direcciones lógicas que aparecen en un programa a sus respectivas direcciones físicas. Esto es necesario cada vez que se utilicen mecanismos de relocalización o de memoria virtual. Si esta bandera está apagada, entonces se evitarán los pasos que debe ejecutar la unidad de manejo de memoria para efectuar el mapeo de direcciones, de manera que la ejecución del programa se hace más veloz.

La PSW se encuentra en un registro de 12 bits en el cual las banderas se encuentran distribuidas de la siguiente manera: (el bit 0 es el más significativo)

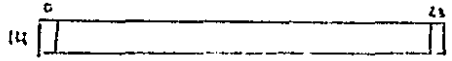
- bit 0 - modo de operación de memoria
  1 - modo de operación del procesador
  2 - modo de rastreo
  8 - acarreo
  9 - overflow
  10 - signo
  11 - cero.

Se dejaron 5 bits sin uso para la posible inclusión de otras banderas.

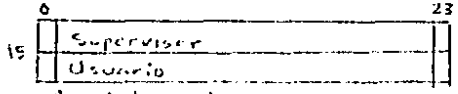
Estructura de los registros.



Registros de Propósito General



Contador de Programa



Apuntador de Stack

Registro de Estado del Programa

2.3. FORMATO DE LAS INSTRUCCIONES.

El primer punto a considerar en el diseño del procesador es el formato de las instrucciones.

La mayoría de los procesadores no tienen un solo formato de instrucción sino que tienen varios. Esto se debe a que sus diseñadores deben tratar de aprovechar todas las combinaciones de bits que el procesador puede entender en un momento dado, de manera que si una instrucción es de dos direcciones, tendrá un campo para el código de operación, un campo para el primer operando, y otro campo para el segundo operando, definiendo así, las instrucciones de dos direcciones, un formato. En el mismo procesador, las instrucciones de una dirección podrían utilizar el campo del segundo operando, dado que no es necesario un segundo operando, para otros fines, por ejemplo, el del código de operación, de forma que puedan haber más instrucciones de una dirección que de dos direcciones en el conjunto de instrucciones ejecutables del procesador. Entonces las instrucciones de una dirección definen otro formato. En este caso el procesador tendría al menos, instrucciones con dos formatos diferentes.

En el AR-24 existen instrucciones de 0,1 y 2 operandos, en donde cada operando de más, define un formato básico de instrucción.

Existen algunos criterios que deben ser tomados en cuenta en el diseño del formato de las instrucciones. Unos se refieren al aprovechamiento de la memoria para que no se desperdicie, y también a la manera de que se puede obtener la mayor velocidad de proceso.

Uno de estos criterios de diseño es el que dice que las instrucciones cortas son mejores que las largas. Otro criterio habla del tamaño de las unidades direccionables en memoria. En el AR-24 esta unidad es el byte de 8 bits. Esto facilita el trabajar con caracteres, a los cuales es posible representar con el código ASCII. El AR-24 permite direccionar hasta 2 a la 24 bytes o aproximadamente 16 Mbytes.

Es conveniente que una instrucción ocupe un número entero de palabras de información, o bien que un número entero de instrucciones ocupen una palabra de información. Esta relación debe existir en toda máquina puesto que permite cierto grado de eficiencia en la transferencia y proceso de las instrucciones, puesto que los buses y los registros son casi siempre del tamaño de la palabra de información de la computadora. En el AR-24 la palabra es de 24 bits y las instrucciones básicas son de 8, 16 o 24 bits.

Es conveniente también que la palabra de información sea un

número múltiplo del byte. Esto para que se almacenen los caracteres eficientemente en la memoria. En el AR-24 es posible almacenar 3 caracteres por palabra.

El formato de las instrucciones en el AR-24 es bastante sencillo. El código de operación requiere de 8 bits para todas las instrucciones. De esta manera las instrucciones sin operandos solo miden 8 bits.

Si la instrucción es de un operando, entonces ésta tendrá el byte del código de operación, más un byte que indica el modo de direccionamiento del operando y un número de registro (en un caso especial este byte contiene el operando de la instrucción). Los 4 primeros bits de este byte indican con cual de los 9 modos de direccionamiento se obtendrá el operando, ya sea de memoria o de registro de CPU. Los otros 4 bits del byte dan la posibilidad al programador de especificar un número de los 16 que se encuentran en CPU en caso de que el modo de direccionamiento lo requiera. Si el modo de direccionamiento requiere de un dato (o dirección) este puede ser especificado en los tres bytes subsecuentes. De esta manera una instrucción de una dirección puede ser de 2 o de 5 bytes, dependiendo esto del modo de direccionamiento que se utilice para accesar al operando.

En las instrucciones de dos operandos existe un byte que es el que especifica el código de operación de la instrucción, más dos bytes que tienen cada uno el formato descrito antes para las instrucciones con una dirección. De esta manera cada operando puede ser obtenido con un modo de direccionamiento diferente. Estas instrucciones pueden tener una palabra (3 bytes) de extensión por cada operando y si éste es el caso, esta palabra va colocada después de el byte especificador y antes del segundo operando (si pertenece al primer operando y la instrucción es de dos operandos) o al final de la instrucción (si pertenece al segundo operando). Así las instrucciones de dos operandos pueden ser de 3, 6 o 9 bytes cada una.

Son los dos primeros bits del código de operación los que indicarán cuantos operandos tiene la instrucción:

- 00 - para instrucciones sin operando
- 01 - para instrucciones con un operando
- 10 - para instrucciones con dos operandos.

De esta manera quedan 6 bits para la instrucción en sí. Y entonces pueden existir 64 instrucciones de cero direcciones,

64 de una dirección y 64 de dos direcciones. Siguiendo este esquema hubiese sido fácil definir a las instrucciones de 3 direcciones para este procesador, aunque ya antes se ha dicho (sección 1.3.2.2.) que éstas no son del todo necesarias, y en cambio si alargan demasiado el tamaño de la instrucción.

Los formatos básicos de instrucción y sus variaciones en BNF.

```
<INSTRUCCION DE CERO DIRECCIONES>::=
    <CODIGO DE OPERACION>

<INSTRUCCION DE UNA DIRECCION>::=
    <CODIGO DE OPERACION><OPERANDO>

<INSTRUCCION DE DOS DIRECCIONES>::=
    <CODIGO DE OPERACION><OPERANDO><OPERANDO>


<OPERANDO>::=
    <BYTE ESPECIFICADOR DEL OPERANDO>
   |<BYTE ESPECIFICADOR DEL OPERANDO><PALABRA DE EXTENSION>

<BYTE ESPECIFICADOR DEL OPERANDO>::=
    Bit-apagado<MODO DE DIRECCIONAMIENTO><REGISTRO DE CPU>
   |Bit-prendido<DATO DE 7 BITS>

<PALABRA DE EXTENSION>::=
    <DIRECCION>
   |<DATO DE 24 BITS>
```

2.4. MODOS DE DIRECCIONAMIENTO.

Para que un procesador pueda ejecutar una instrucción con operandos, estos deben estar disponibles en algún lugar de la máquina. Ya antes (sección 1.3.2.4.) se había dicho que los operandos pueden encontrarse en los registros de CPU o en la memoria, y que es posible hacer referencia a ellos de muchas diferentes maneras. A estas diferentes maneras se les llama modos de direccionamiento.

Algunos procesadores permiten que algunas de sus instrucciones hagan referencia a estos operandos por un solo modo de direccionamiento; mientras que permiten que otras de sus instrucciones tengan acceso a sus operandos de múltiples maneras, esto es, dan la opción de que el programador, en la instrucción, especifique el modo de direccionamiento.

El hecho de que en algunos procesadores, algunas de sus instrucciones solo permitan un modo de direccionamiento, se debe a que los diseñadores del procesador consideraron que dichas instrucciones no necesitaban operar con otros modos y por lo tanto se debía utilizar el espacio disponible en la instrucción para otras cosas. Mientras más opciones de modos de direccionamiento se den al programador, más espacio se utiliza del formato de la instrucción. En el AR-24, el espacio no es problema, por lo tanto todas las instrucciones pueden ser utilizadas con todos los modos de direccionamiento disponibles, cuando esto sea coherente. Además las instrucciones de dos operandos pueden ser utilizadas de forma tal que el primer operando sea referenciado con un modo de direccionamiento, mientras que el segundo sea referenciado con otro modo diferente al del primero. Esto hace al procesador muy general y flexible.

Como dijimos antes, existen 4 bits por cada operando en cada instrucción para especificar el modo de direccionamiento. A continuación se da una lista de los modos con la cantidad (en dígitos binarios) que los identifica en las instrucciones. Además se da una de las formas en que cada modo podría ser referenciado en un programa escrito en ensamblador (r es un número de registro y d es un dato o una dirección).

- 0000 - inmediato — #Ld
- 0001 - relativo al contador de programa — Pd
- 0010 - directo de registro — Rr
- 0011 - directo de memoria — Md
- 0100 - indirecto de registro — @Rr

- 0101 - indirecto de memoria &Rd
- 0110 - registro índice o registro base BrRd
- 0111 - autoincremento +Rr
- 1XXX - inmediato corto. #d

X indica que esos bits pueden estar prendidos o apagados. A continuación se da una descripción de cada uno de estos modos.

- DIRECCIONAMIENTO INMEDIATO.

Cuando se utiliza este modo, la instrucción debe llevar consigo una palabra de información de extensión. La palabra de 3 bytes es el o uno de los operandos de la instrucción. En este caso los bits que corresponden al número de registro en la instrucción, no son utilizados. Este modo de direccionamiento puede ser obtenido también, utilizando el modo de autoincremento junto con el registro 14 (PC).

- DIRECCIONAMIENTO RELATIVO AL CONTADOR DE PROGRAMA.

Este modo es el que permite especificar una palabra de extensión cuyo valor se suma al contador del programa y da como resultado una dirección de memoria que contiene al operando de la instrucción. Cuando este modo es utilizado en un programa en lugar de los modos que hacen referencia a direcciones específicas de memoria, permite, si no hay direcciones de retorno de subrutina en el stack, que el programa pueda ser cargado en cualquier parte de la memoria y se ejecute correctamente.

- DIRECCIONAMIENTO DIRECTO DE REGISTRO.

Cuando se utiliza este modo, el operando de la instrucción se encuentra en el registro cuyo número se especifica en los 4 bits reservados para el caso. Dado que este modo no utiliza palabra de extensión, las instrucciones que lo utilizan son más cortas y necesitan menos accesos a memoria y por lo tanto son más rápidas de ejecutarse.

- DIRECCIONAMIENTO DIRECTO DE MEMORIA.

En este modo, una palabra de 3 bytes de extensión en la instrucción, es la dirección de la localidad de memoria en donde se encuentra el operando. También se le llama direccionamiento absoluto.

- DIRECCIONAMIENTO INDIRECTO DE REGISTRO.

Cuando se utiliza este modo, el operando está contenido en una localidad de memoria cuya dirección está en un registro de CPU. El número del registro se especifica en la instrucción. Este modo no requiere de palabra de extensión.

- DIRECCIONAMIENTO INDIRECTO DE MEMORIA.

Cuando se utiliza este modo, el operando está contenido en una localidad de memoria cuya dirección se encuentra dada en otra dirección de memoria cuya dirección está especificada en una palabra de extensión de la instrucción.

- DIRECCIONAMIENTO CON REGISTRO BASE O REGISTRO INDICE.

Cuando hablamos de modos de direccionamiento en la sección 1.3.2.4., dijimos que la diferencia entre el direccionamiento con registro índice y el direccionamiento con registro base estriba en el tamaño del registro y del desplazamiento que se especifican en la instrucción. Dado que que en el AR-24 los registros son de 24 bits, en ellos caben direcciones de memoria, por lo que pueden ser tomados como registros base (y desde luego como registros índice). Y como el desplazamiento, el cual es una palabra de información que se coloca como extensión de la instrucción y tiene 3 bytes, también puede alojar una dirección de memoria, se satisface así el requerimiento de la indexación. Independientemente de como se le llame al modo de direccionamiento, la dirección en memoria del operando se obtiene sumando el contenido del registro especificado en la instrucción y el contenido de la palabra de extensión.

- DIRECCIONAMIENTO CON AUTOINCREMENTO.

Cuando se utiliza este modo de direccionamiento, se especifica en la instrucción un número de registro, de manera que la dirección del operando es el contenido de este registro. Después de obtenido el operando, el contenido de ese registro se incrementa en tres. El valor del incremento es tres debido a que las palabras de memoria son de tres bytes.

- DIRECCIONAMIENTO INMEDIATO CORTO.

A este modo de direccionamiento se debe que haya otra versión en el formato de los bytes especificadores de operandos. Cuando se utiliza este modo, es posible que los bits 1 a 7 (donde el bit 7 es el menos significativo) de este byte tengan cualquier valor, el cual es tomado como operando. El bit 0 de este byte es el que indica al procesador, cuando está prendido, que se está utilizando este modo. Esto es posible puesto que los demás modos operan con este bit apagado.

La idea de utilizar este modo viene del hecho de que es usual

trabajar con operandos inmediatos de poca magnitud que no justificarían el uso del modo inmediato normal, el cual da 3 bytes para el almacenamiento del operando.

Es conveniente aclarar que las dos versiones de modo inmediato son los únicos tipos de direccionamiento que no pueden ser utilizados en cualquier instrucción. Por ejemplo, en una operación de suma, si el segundo operando es inmediato, no hay manera de que el procesador se entere de cual es el lugar en donde debe almacenar el resultado.

2.5. EL CONJUNTO DE LAS INSTRUCCIONES.

Una computación puede ser vista como la evaluación de una función Y=f(X) donde X son los datos de entrada, Y son los datos de salida y f es una secuencia de pasos (instrucciones) que procesan los datos de entrada para obtener los de salida. Al conjunto de pasos se le conoce como programa. Existen computaciones que no es posible llevar a cabo.

El matemático Alan Turing introdujo en 1936 un modelo abstracto con cuatro operaciones (escribir, avanzar, retroceder y parar) llamado Máquina de Turing, el cual es capaz de describir lo que conocemos como algoritmo.

Ahora bien, todo conjunto de instrucciones de cualquier procesador debe ser completo, esto es, cualquier algoritmo debe poder ser expresado mediante una secuencia de instrucciones obtenidas del conjunto, usando el espacio de memoria disponible. Esto quiere decir que el conjunto de instrucciones de un procesador podría ser tan simple como el que tiene una máquina de Turing. Como dato curioso, Van der Poel diseño una máquina cuyo conjunto de instrucciones tiene una sola instrucción.

Pero hay que tener en cuenta otro aspecto importante, que es el de la eficiencia del conjunto de instrucciones. Mientras menos instrucciones tenga el conjunto, más largos y complejos son los programas. De manera que es conveniente que existan instrucciones que ejecuten las funciones que son requeridas comúnmente en los programas.

Otro aspecto que puede ser considerado en el diseño de las instrucciones, es el de la similitud con las instrucciones ya existentes en otros procesadores. Sería difícil que tuviera buena acogida una máquina con instrucciones que no se parecieran a las de las computadoras ya existentes, puesto que, en este caso, el programador tendría que aprender a programar con otros tipos de instrucciones.

A continuación se da una descripción de las instrucciones del AR-24, agrupándolas por sus diferentes tipos.

2.5.1. INSTRUCCIONES ARITMETICAS.

SUM X,Y - suma.

RES X,Y - resta.

MUL X,Y - multiplicación.

DIV X,Y - división.

Las operaciones aritméticas básicas son instrucciones de dos operandos en donde el resultado se almacena en el segundo operando. En la resta, el primer operando se sustrae del segundo. En la división, el segundo operando se divide por el primero. Todas las operaciones toman a sus operandos como magnitudes con signo. La suma y la resta afectan a las cuatro banderas de condición. La multiplicación afecta a la de overflow, a la de signo y a la de cero. La división afecta a la de signo y a la de cero.

SUMAC X,Y - suma con acarreo.

Esta operación suma los dos operandos y almacena el resultado en el lugar que ocupaba el segundo de ellos. Al resultado se le suma el contenido de la bandera de acarreo. Esta operación afecta a todas las banderas de condición.

RESAC X,Y - resta con préstamo.

Esta operación sustrae X de Y y almacena el resultado en Y. Si la bandera de acarreo esta prendida, resta uno al resultado. RESAC afecta a todas las banderas de condición.

SUMF X,Y - suma con punto flotante.

RESF X,Y - resta con punto flotante.

MULF X,Y - multiplicación con punto flotante.

DIVF X,Y - división con punto flotante.

Estas son las instrucciones que toman a sus operadores como números de punto flotante. El signo del número está en el bit 0 (0 indica positivo; uno, negativo), el exponente (que está expresado en el sistema de magnitud con signo) ocupa los bits del 1 al 8 (el signo del exponente está en el bit 1) y la mantisa está en los bits 9 a 23. El resultado de las operaciones se almacena en el lugar del segundo operando. Las cuatro operaciones actualizan a las banderas de overflow, de signo y de cero.

INC X - incrementa.

DEC X - decrementa.

INC incrementa el operando en una unidad, DEC lo decrementa. Todas las banderas de condición excepto la de acarreo, se afectan con estas instrucciones. El operando es tomado como una magnitud con signo.

NEG X - cambia de signo.

Esta instrucción complementa el bit 0 del operando, el cual es el signo. De manera que si el bit está prendido, lo apaga, y si está apagado lo prende. Esta operación afecta a las banderas de signo y de cero.

ABS X - valor absoluto.

Esta instrucción pone el signo del operando en positivo. Es decir pone un cero en el bit 0. La operación afecta a las banderas de signo y de cero.

MOD X,Y - módulo.

Esta operación pone en Y el residuo de la división de X entre Y. MOD toma a X y a Y como magnitudes sin signo. No se afectan banderas de condición.

2.5.2. INSTRUCCIONES LOGICAS.

Y X,Y - y lógico.

Esta instrucción obtiene el Y lógico entre X y Y poniendo el resultado en Y. La instrucción pone un bit prendido en el lugar correspondiente del resultado si los bits correspondientes de los operandos están prendidos. De otra manera pone el bit apagado. La instrucción afecta a las banderas de signo y de cero.

O X,Y - o lógico.

Esta instrucción obtiene el O lógico de X y Y poniendo el resultado en Y. La instrucción pone un bit apagado en el lugar correspondiente del resultado solo si los bits correspondientes están apagados. De otra manera pone un bit prendido. La instrucción afecta a las banderas de cero y de signo.

OEX X,Y - o exclusivo.

Esta instrucción obtiene el O exclusivo de X y de Y poniendo el resultado en Y. La instrucción pone un bit apagado en el lugar correspondiente del resultado si los bits correspondientes en los operandos tienen valores iguales. Si no son iguales los valores, entonces pone un bit prendido. La operación afecta a las banderas de signo y de cero.

CODER X,Y - corrimiento lógico a la derecha.

Esta operación efectúa el corrimiento lógico a la derecha del operando X una cantidad Y de veces, poniendo ceros en los lugares vacantes de la izquierda. La bandera de acarreo toma el valor del último bit corrido del operando. Las banderas de cero y de signo se afectan normalmente.

COIZQ X,Y - corrimiento lógico a la izquierda.

Esta instrucción corre Y bits a la izquierda en el operando X. El número de bits corridos es ocupado por ceros. La operación afecta a las banderas de cero y de signo. La bandera de acarreo toma el valor del último bit sacado de X.

CODEA X,Y - corrimiento aritmetico a la derecha.

Esta instrucción corre Y bits a la derecha en el operando X. El valor del bit de la izquierda se queda con el valor que había allá antes del corrimiento, conservándose así el signo del operando. Los demás bits vacantes se ocupan con ceros. Esta instrucción afecta a las banderas de cero y signo normalmente, pero la de acarreo toma el valor del último bit corrido del operando.

RODER X,Y - rotación a la derecha.

Esta instrucción rota el contenido del operando X un número de bits especificado por Y. Esta operación afecta a las banderas de signo, de cero y de acarreo, la cual toma el valor del bit menos significativo del resultado.

ROIZQ X,Y - rotación a la izquierda.

Esta instrucción rota al operando X, el número de bits especificado por Y. ROIZQ afecta a las banderas de signo, de cero y de acarreo, la cual toma el valor del bit más significativo del resultado.

RODEA X,Y - rotación a la derecha con acarreo.

Esta operación rota X a la derecha el número de veces especificado por Y, pero tomando a la bandera de acarreo como

parte del operando, o sea que su valor es rotado al bit más significativo del operando cada vez, y su valor es remplazado por el bit menos significativo de éste. Las banderas de cero y de signo también se afectan.

ROIZA X,Y - rotación a la izquierda con acarreo.

Esta operación rota X a la izquierda un número de veces especificado por Y, pero tomando a la bandera de acarreo como parte del operando, o sea que su valor es rotado al bit menos significativo del operando y su valor es remplazado por el bit más significativo de éste cada una de las veces. Las banderas de cero y de signo también se afectan.

COMP X,Y - comparación.

Esta operación sustrae el operando Y al operando X sin almacenar el resultado en ninguna parte. Esta instrucción afecta a todas las banderas de condición.

INV X - complementación.

Esta instrucción remplaza al operando por su complemento a uno. La instrucción afecta a las banderas de cero y de signo.

2.5.3. INSTRUCCIONES DE TRANSFERENCIA.

MOV X,Y - transferencia.

Esta instrucción copia la información contenida en el operando X al lugar indicado por el operando Y. MOV no afecta banderas.

MOVC X - mueve ceros.

La instrucción MOVC mueve ceros al operando. No se afectan banderas.

MOVU X - mueve unos.

Esta instrucción mueve unos al operando especificado por X. La operación no afecta a ninguna bandera.

PBIT X,Y - prende bit.

ABIT X,Y - apaga bit.

CBIT X,Y - complementa bit.

Estas instrucciones prenden, apagan o complemetan (si está prendido lo apaga; si está apagado lo prende), respectivamente, el bit número Y del operando X. Los bits están numerados de izquierda a derecha del 0 al 23. No se afecta ninguna bandera con esta instrucción.

DBIT X,Y - deja bit.

Esta instrucción aisla el bit número Y del operando X. Este bit quedará ocupando el bit menos significativo (el número 23) del operando X, quedando todos los demás bits apagados. No se afectan banderas con esta operación.

PUSH X - mete información al stack.

Esta instrucción mete los datos que están especificados por X al stack que está en funciones (dependiendo del modo de operación del procesador: supervisor o usuario). El SP (registro 15) siempre contiene la dirección del último byte ocupado del stack y es decrementado en tres unidades cada vez que se ejecuta esta instrucción. Si el operando es de tres bytes, el menos significativo es el que entra primero al stack. El más significativo entra de último. PUSH no afecta banderas.

POP X - saca información del stack.

Esta instrucción obtiene datos del stack que está en funciones (dependiendo del modo de operación: supervisor o usuario) y lo coloca en X. El apuntador del stack se incrementa en tres unidades al ejecutarse la instrucción y queda apuntando al último byte ocupado del stack. Siempre se obtienen tres bytes del stack, y el primero de ellos es el que será el más significativo en el operando. No se afectan banderas.

PUSHR - mete los registros al stack.

PUSHR mete al stack los registros 0 al 13 de CPU. El primero en entrar es el 0 y el último es el 13. No se afectan banderas.

POPR - saca los registros del stack.

Esta instrucción obtiene del stack los registros 0 a 13 y los restaura en sus posiciones de CPU. El primer registro en salir es el 13 y el último es el 0. POPR no afecta banderas.

PUSHB - mete las banderas al stack.

Esta instrucción mete al stack el registro de banderas o PSW. El SP queda decrementado en tres unidades. No se afectan las banderas durante la operación.

POPB - saca las banderas del stack.

Esta instrucción obtiene del stack el registro de banderas y lo restaura en su lugar en la CPU. El SP queda incrementado en tres unidades. Todas las banderas de condición y de control se afectan con esta instrucción.

MPMTP X - actualiza el apuntador a tablas de mapeo de páginas.

El registro apuntador a las tablas de mapeo de páginas en memoria o PMTP (page map table pointer) de la unidad de manejo de memoria puede ser actualizado por medio de esta instrucción. El contenido de X pasa al PMTP. No se afectan banderas.

2.5.4. INSTRUCCIONES DEL CONTROL DEL PROGRAMA Y DEL PROCESADOR.

BRI X - brinca.

Esta instrucción transfiere incondicionalmente el control del programa ejecutándose a la instrucción que se encuentra en la localidad de memoria indicada por X. No se afectan banderas.

BMAC X,Y - bifurca a Y si X es mayor que cero.

BMEC X,Y - bifurca a Y si X es menor que cero.

BIC X,Y - bifurca a Y si X es igual a cero.

Estas instrucciones comparan al operando X con el valor cero. Si X resulta mayor, menor o igual (dependiendo de la instrucción) se transfiere el control del programa a la localidad de memoria indicada por Y. De otra manera el programa sigue [illegible]

[illegible] fórmulas, cuyas variables son banderas de condición, se cumplen. A continuación damos una lista de los nemónicos, sus significados y las fórmulas que se evalúan para cada instrucción. La utilización de mayor y menor supone que los operandos son cantidades con signo, mientras que la utilización de arriba y abajo los suponen caracteres de 8 bits. Estas instrucciones no afectan a las banderas. (a es la bandera de acarreo, o es la de overflow, c la de cero y s la de signo).

INSTRUCCION	SE VERIFICA QUE...	SALTA SI...
BMA X	(s O c)=0	mayor.
BME X	s=1	menor.
BIGU X	c=1	iguales.
BMAI X	s=0	mayor o igual.
BMEI X	(s O c)=1	menor o igual.
BAR X	(o O c)=0	arriba.
BAB X	o=1	abajo.
BDIF X	c=0	diferentes.
BARI X	o=0	arriba o igual.
BABI X	(o O c)=1	abajo o igual.
BAC X	a=1	acarreo.
BNAC X	a=0	no acarreo.

Continuamos con las instrucciones de control de programa.

BSUB X - brinca a subrutina.

Esta instrucción es la llamada a subrutina. Primero el PC es metido al stack con la dirección de la siguiente instrucción. El byte menos significativo del PC es el que entra primero. Luego el control del programa es transferido a la instrucción que se encuentra en la localidad de memoria especificada por X. No se afectan banderas.

RET - retorno de subrutina.

Esta instrucción toma tres bytes del stack (por medio de una operación POP) y forma con ellos el contador del programa que es el que indica a que lugar de la memoria se transfiere el control del programa. La instrucción no afecta banderas.

PBA X - prende bandera.

ABA X - apaga bandera.

PBA prende la bandera de la PSW especificada por X. ABA la apaga. Las banderas están numeradas en el siguiente orden: 0 es la bandera de acarreo, 1 es la de overflow, 2 es la de signo y 3 es la de cero. La bandera a la que se hace referencia es la única que se afecta.

CMME - cambia modo de operación de memoria.

Esta instrucción cambia el modo de operar de la memoria, de manera que si está trabajando en modo de paginación, deje de hacerlo, y si no está en modo de paginación empiece a estarlo. En la sección 2.7. se dan detalles del funcionamiento de la memoria. Esta instrucción solo afecta a la bandera de control de modo de memoria en la PSW. Esta es una instrucción privilegiada.

CMPR - cambia modo de operación del procesador.

El procesador tiene dos modos de funcionamiento: de supervisor y de usuario. Esta instrucción intercambia los modos de manera que se complementa la bandera de control de la PSW y además el contenido del registro 15 se intercambia con el contenido de la memoria en los bytes del 0 al 2, para que de esta manera, el SP del modo en curso quede accesible al programador. Esta es una instrucción privilegiada.

CMRA - cambia modo de rastreo.

Esta instrucción invierte el contenido de la bandera de rastreo. Cuando esta bandera está prendida, el procesador se detiene después de la ejecución de cada instrucción. La instrucción es privilegiada.

INT X - genera una interrupción.

Esta instrucción mete al stack el PC y la PSW (en ese orden), luego apaga la bandera de rastreo de la PSW, después toma los siete bits menos significativos del operando (que indican de que tipo de interrupción se trata) y los multiplica por 3. Por último obtiene de la localidad de memoria apuntada por esta cantidad, una dirección que asigna al PC. La única bandera afectada es la de rastreo.

RETIN - retorno de interrupción.

Esta instrucción obtiene del stack la PSW y el PC, en ese orden, y los restaura en la CPU. De esta manera todas las banderas resultan afectadas.

PARA - detiene al procesador.

Esta instrucción privilegiada obliga al procesador a detenerse. No afecta a ninguna bandera.

NOP - no operación.

Cuando esta instrucción es detectada, el procesador no ejecuta ninguna operación, simplemente incrementa el contador de programa. No se afectan banderas.

2.5.5. INSTRUCCIONES DE ENTRADA Y SALIDA.

LEE X - lee byte.

Esta instrucción obtiene un byte del puerto de entrada y lo almacena en la localidad de memoria especificada por X. No se afectan banderas. Esta instrucción es privilegiada.

ESC X - escribe byte.

Esta instrucción manda un byte que se encuentra almacenado en la localidad de memoria especificada por X, al puerto de salida. No se afectan banderas. La instrucción es privilegiada.

LEEBL X,Y - lee un bloque de información.

Esta instrucción obtiene un bloque de información del puerto de entrada y lo almacena en memoria a partir de la localidad especificada por X. El número de bytes (tamaño del bloque) está especificado en Y. No se afectan banderas. La instrucción es privilegiada.

ESCBL X,Y - escribe un bloque de información.

Esta instrucción manda un bloque de bytes almacenado en memoria a partir de la localidad indicada por X al puerto de

salida. El número de bytes a transmitirse se especifica en el operando Y. No se afectan banderas. Esta instrucción es privilegiada.

ESCRE - escribe el contenido de los registros.

Esta instrucción manda al puerto de salida el contenido de los registros de CPU (0 al 15), el contenido del SP y del PC al igual que la PSW y el PMTP. No se afectan banderas. La instrucción es privilegiada.

2.5.6. INSTRUCCIONES DE ALTO NIVEL.

MOVST X,Y - transfiere una cadena.

Antes de la ejecución de esta instrucción, el registro 0 de la CPU debe ser cargado con un valor que indica el número de bytes a transmitirse. Entonces al efectuarse MOVBL, se copia el número de bytes indicado en el registro 0 a partir del byte en memoria especificado por X a la porción de memoria que comienza en el byte especificado por Y.

COMST X,Y - compara cadenas.

Antes de la ejecución de esta instrucción es necesario que el registro 0 de CPU contenga el tamaño de las cadenas a compararse. La instrucción compara las cadenas que comienzan en las localidades de memoria especificadas por X y Y. No se afectan las cadenas, pero se afectan las banderas de overflow y de cero. La instrucción compara cada byte de la cadena y se detiene cuando han sido recorridas completamente las cadenas o bien cuando se detectan bytes correspondientes diferentes en ellas. El registro 0 contendrá su valor original si las cadenas son iguales o bien contendrá una cantidad que será igual al número de bytes recorridos en las cadenas, si éstas son diferentes.

ANAST X,Y - analiza una cadena.

Antes de la ejecución de esta instrucción es necesario que el registro 0 de CPU contenga el tamaño de la cadena a analizarse. La instrucción compara una cadena que comienza en la localidad de memoria indicada por X, con un byte que está especificado por Y. La instrucción no afecta a ninguno de los

operandos, pero afecta a las banderas de overflow y de cero. La instrucción va comparando el byte especificado en Y con cada uno de los bytes que componen la cadena y se detiene cuando encuentra un byte en la cadena que es igual al especificado, o bien cuando termina de recorrer a la cadena. Cuando termina la ejecución de la instrucción, el registro 0 contendrá su valor original si no encontró algún byte igual al del operando, o bien contendrá una cantidad que será igual al número de bytes recorridos en la cadena, si se encontró el byte buscado.

ITERA X,Y - itera un bloque de instrucciones.

Esta instrucción ejecuta una o más veces a un bloque de instrucciones en un programa. El operando X indica la dirección de la primera instrucción de este bloque y el operando Y especifica el número de registro que contiene el número de veces que se ejecutarán las instrucciones del bloque. La instrucción va al final del bloque y no afecta a las banderas.

LIGA X,Y - entrada a un procedimiento.

Esta instrucción almacena en el stack el contenido del registro especificado por X, luego carga a este registro con el apuntador del stack (ya decrementado en tres unidades por el PUSH), por último decrementa el SP, restándole la cantidad indicada por Y para que apunte más allá del área del stack destinada para almacenamiento temporal. Ninguna bandera es afectada por esta instrucción.

DESLI X - salida de un procedimiento.

Esta instrucción almacena el contenido del registro especificado por X en el apuntador de stack (registro 15) y carga a este registro con el contenido del stack en la localidad apuntada por el nuevo SP. Después el SP se incrementa en tres (por el POP). No se afectan banderas.

2.6. INTERRUPCIONES.

Las interrupciones generadas por software ("trampas") que detecta el AR-24, pueden ser de 127 tipos (de la 1 a la 127). El tipo de interrupción 1 es la que se genera cuando el modo de rastreo del procesador está habilitado. La interrupción 2 se genera cuando ocurre una división entre cero en el programa que se está ejecutando. Por último, las interrupciones 3 a 127 se dan cuando el usuario las solicita en su programa por medio de la instrucción INT. Esta instrucción posee un operando que es un número que puede estar en el rango de 1 a 127.

Cuando una interrupción es detectada, el PC y la PSW del programa en ejecución son almacenados en el stack. La bandera de rastreo es apagada, y la dirección de la rutina de servicio de interrupción es cargada a la CPU. La bandera de rastreo se apaga para que esta rutina pueda ser ejecutada sin ser interrumpida.

La dirección de la rutina de servicio es tomada de una tabla que se encuentra en las localidades más bajas de la memoria. Esta tabla ocupa 381 bytes (de la localidad 3 a la 383. Las localidades 0 a 2 son ocupadas por el SP del modo usuario o supervisor.), esto es, 127*3, donde 127 es el número de tipos de interrupción y 3 es el número de bytes que ocupa cada dirección.

La localidad de la tabla donde se encuentra la dirección de la rutina de servicio a la interrupción se obtiene multiplicando el tipo de la interrupción por 3.

De esta manera, la dirección de la rutina que se ejecuta después de cada instrucción cuando el modo de rastreo está habilitado, se encuentra en las localidades 3 a 5 de memoria, y la dirección de la rutina que se ejecuta cuando ocurre una división entre cero se encuentra en las localidades 6 a 8.

Las otras posiciones de la tabla pueden ser utilizadas para almacenar las localidades de inicio de rutinas de servicio a interrupciones que el sistema operativo o el usuario tengan definidas.

Para que el procesador siga procesando el programa que se estaba ejecutando antes de que se generara la interrupción, se requiere que detecte la instrucción RETIN. Esta instrucción toma el PC y la PSW que se habían almacenado en el stack al momento de la interrupción, y los carga en sus lugares respectivos en la CPU.

2.7. ORGANIZACION DE LA MEMORIA.

Como vimos en la sección 1.3.1., las memorias de las computadoras pueden estar organizadas de diferentes maneras. La memoria del AR-24 tiene una arquitectura lineal. Esto indica que la memoria puede ser direccionada desde la unidad 0 hasta la unidad N, donde N es el número total de unidades que tiene, y las direcciones están en forma secuencial. En el AR-24 las unidades direccionables son los bytes de 8 bits. Tres bytes forman una palabra de información de memoria. Esto proporciona facilidades para que puedan ser direccionados 2 a la 24 bytes, o aproximadamente 16 Mbytes. De manera que éste es el espacio de direccionamiento lógico de este procesador.

Se escogió al byte como unidad direccionable en lugar de la palabra (24 bits), dado que el direccionamiento de bytes ofrece facilidades en el manejo de cadenas.

Las palabras pueden empezar en cualquier byte. Esto es así pues, aunque las direcciones de memoria son de tamaño fijo, no así las instrucciones, las cuales varían de 1 a 3 bytes. Como habíamos dicho antes, el espacio de direccionamiento lógico de una computadora puede ser más grande que su espacio de direccionamiento físico. Para esto, es necesario que la computadora esté provista de algún mecanismo de relocalización, de manera que las direcciones lógicas puedan ser mapeadas en el espacio de direccionamiento físico de manera automática.

En el AR-24 el mapeo de direcciones es llevado a cabo mediante un sistema basado en páginas. El espacio de direccionamiento lógico está dividido en 4096 (4 K) páginas de 4096 bytes cada una. Los 24 bits utilizados para almacenar direcciones están divididos en dos campos: uno es el campo selector de página y el otro es el campo de desplazamiento, cada uno de 12 bits de tamaño. De esta manera, una dirección lógica indica en sus primeros doce bits en que número de página se encuentra, y con los últimos doce bits, en que byte de esa página está.

El espacio de direccionamiento lógico está dividido en páginas, cada una de las cuales empieza en una dirección múltiplo de 4096 (o lo que es lo mismo, en cada dirección que termina con doce ceros binarios). El espacio de direccionamiento físico está dividido en marcos de página, cada uno de los cuales empieza también en una dirección múltiplo de 4096.

Una tabla en memoria llamada tabla de mapeo de páginas o PMT (page map table) se encarga de mantener el control de las páginas de un programa. Si se está trabajando en un ambiente de multiprogramación, es posible tener en memoria una tabla por cada uno de los programas que estén compartiendo el

procesador. Un registro llamado apuntador a la tabla de mapeo de páginas o PMTP (page map table pointer) contendrá la dirección de la PMT del programa que esté utilizando el procesador en un momento dado.

Cada tabla ocupa 8K bytes de espacio de memoria (2 páginas). La primera de estas dos páginas siempre deberá estar en un marco de página par (terminado en cero binario), y la otra, a continuación (de número impar). En cada tabla hay un mapa de las páginas del espacio de direccionamiento lógico y cada página (de las 4096) representada, es una posición de la tabla y tiene un valor asociado que es el de la dirección del marco de página en donde se encuentra físicamente (si se encuentra) la página en la memoria. Si no se encuentra, un bit situado en la tabla, lo indica. Debido a que los marcos de página terminan con doce ceros, no es necesario almacenar toda la dirección. Los primeros doce bits son suficientes. Pero como no caben en un byte, cada posición de la tabla ocupa dos bytes consecutivos. Y esa es la razón por la que son necesarias dos páginas de memoria por cada PMT.

Los cuatro bits que sobran en cada posición de la tabla son utilizados para lo siguiente: el bit número 4 (de izquierda a derecha) del segundo byte de la posición en la tabla indica, cuando está prendido, que la página ha sido modificada por el programa. De esta manera, cuando deja de ser utilizada en la memoria física para que el espacio que está ocupando lo utilice otra página, sea copiada de vuelta a la memoria secundaria (disco). Cuando el bit está apagado no es necesario hacer esta operación.

Los siguientes bits (el 5 y el 6) sirven para la protección de la página. Cuando ambos están apagados, es posible tener acceso a la página (leerla o modificarla); cuando el bit 5 está apagado y el 6 está prendido, solo es posible leerla, estando el procesador en modo de usuario, mientras que en modo de supervisor, puede ser leída o modificada; cuando el bit [illegible] está prendido y el 6 [illegible] hay posibilidad de acceso a la página, mientras que en el modo de supervisor es posible leerla o modificarla; cuando los dos bits están prendidos, la página solo puede ser leída en el modo de supervisor, mientras que en el modo de usuario no puede ser accesada.

El último bit (el 7) del byte indica, cuando está prendido, que la página correspondiente se encuentra en la memoria física. Cuando el bit está apagado, el sistema operativo debe traer la página del disco a la memoria para que pueda ser utilizada.

El proceso de mapeo de direcciones se lleva a cabo cuando la bandera de control del modo de memoria está prendida. Si lo

está, el proceso empieza cuando una dirección lógica es pasada a la unidad de manejo de memoria. Entonces el PHTP es accesado, pues contiene la dirección de la tabla que está en ese momento en ejecución. Esta dirección es la de una página par y por lo tanto, sus últimos trece bits están apagados. Entonces se toman los primeros 11 bits de ese registro y se concatenan con el selector de página (12 primeros bits) de la dirección lógica. Entonces estos 23 bits se concatenan con un cero al final.

Con estos 24 bits se accesa a la memoria de la que se obtiene en esa posición un número que es concatenado con los 4 primeros bits del byte siguiente. Estos 12 bits dan un marco de página, los cuales concatenados con el desplazamiento (últimos 12 bits) de la dirección lógica, dan los 24 bits de la dirección física buscada.

Este proceso puede parecer en extremo largo y complicado. En realidad lo es, por eso, en los procesadores que utilizan esquemas similares a éste, utilizan memorias cache y asociativas que mantienen los mapeos de las direcciones lógicas más utilizadas, de manera que el proceso no se hace siempre que haya que accesar la memoria.

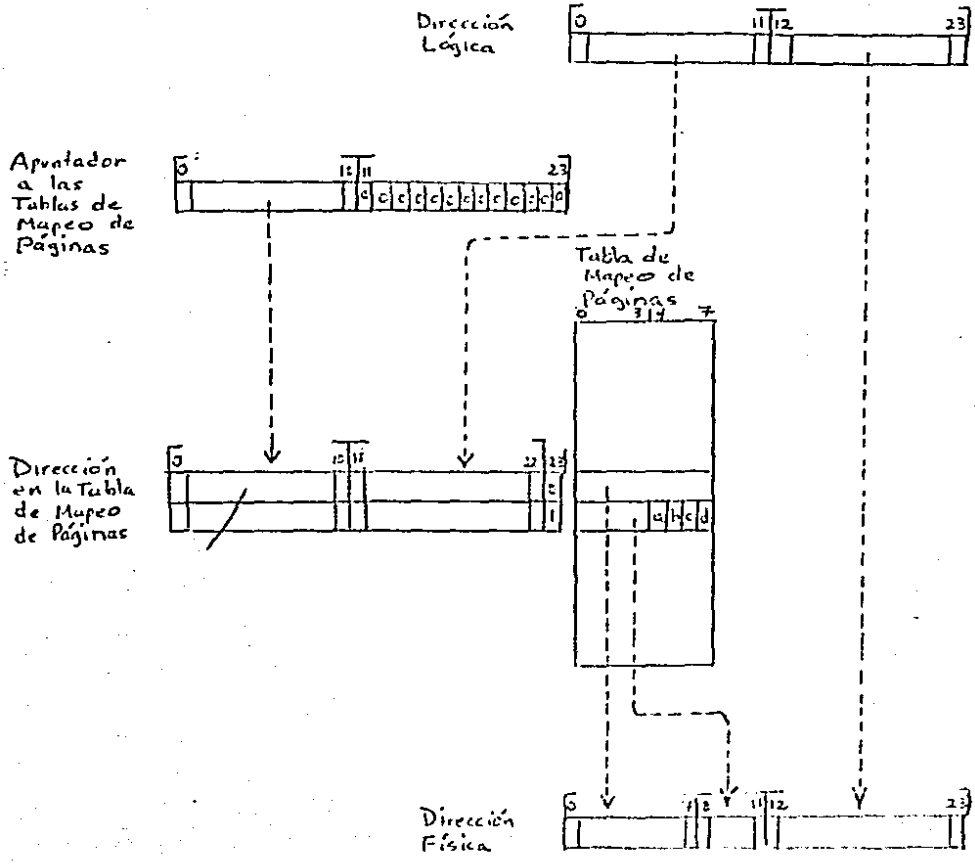
Este esquema proporciona muchas facilidades para el manejo de multiprogramación y de memoria virtual, además de un sistema de protección de memoria que impide que los programas puedan accesar localidades de memoria que no les pertenecen.

Pasando al tema de representación de objetos en la memoria, tenemos que los caracteres alfanuméricos en el AR-24 son representados en un byte cada uno, por lo que se puede utilizar el código ASCII o el EBCDIC, mientras que los enteros son representados en 24 bits con la notación de magnitud con signo.

Con esta notación los números positivos tienen un cero en el bit más significativo de la palabra (bit 0). Los números negativos tienen un uno en esta posición. El problema de esta notación es que da dos representaciones para el número cero. La notación fue escogida entre las varias que existen, pues es la más clara para propósitos académicos.

Los números de punto flotante se representan de la siguiente forma: el bit 0 es el signo del número, el bit 1 es el signo del exponente, los bits 2 al 8 representan al exponente, y los bits 9 al 23, a la mantisa. Los números de punto flotante siempre deben estar normalizados antes de toda operación, las cuales los dejan normalizados.

Las tablas de mapeo de páginas.



a - bit que cuando está prendido indica que la página ha sido modificada.

b.c - bits que indican el tipo de protección de la página.

d - bit que cuando está prendido indica que la página está en memoria.

3. EL SIMULADOR DEL PROCESADOR.

3.1. INTRODUCCION.

Un simulador es un programa de computación cuyo propósito es imitar el comportamiento de algún sistema. El simulador de un procesador debe ser capaz de dar exactamente los mismos resultados que daría éste si estuviese hecho en hardware o en firmware.

Algunas veces los simuladores se construyen para evaluar el comportamiento (performance) de un sistema, pues al mismo tiempo que van ejecutando las tareas del sistema simulándose, van recopilando datos estadísticos que son de gran importancia para la evaluación.

Otras veces, los simuladores se utilizan durante el diseño de una nueva computadora. Esto para poder comprobar que los detalles de la arquitectura sean coherentes. Es muy fácil comprobar los efectos de pequeñas modificaciones en el diseño de un sistema por medio de un simulador.

Para que el procesador AR-24 no se quedara completamente en la teoría, se hizo necesaria la construcción de un programa que simulara su funcionamiento. El programa fue escrito en lenguaje C y fue corrido en una computadora ONIX C5000 con procesador Z8001, trabajando bajo el sistema operativo UNIX III.

El programa simulador fue dividido en dos partes. La primera abarca a la función principal (main) y a todas las que se podrían llamar de utilería, y la segunda contiene a la función que propiamente simula la ejecución de las instrucciones. Dichas partes fueron compiladas por separado.

3.2. MANUAL DEL USUARIO.

Para que el simulador funcione apropiadamente es necesario darle 3 argumentos. El primero es el nombre del programa cuya ejecución va a ser simulada. Este programa debe estar escrito en el lenguaje máquina del AR-24. El código debe estar escrito en dígitos hexadecimales (esto por dar facilidad de escritura al programador), donde los dígitos A a F pueden estar escritos en mayúsculas o en minúsculas. Se permiten los espacios y los retornos de carro. El archivo debe estar en código ASCII.

El segundo argumento es un número que indicará al simulador en que lugar de su memoria deberá cargar el programa cuya ejecución se simulará.

El tercer argumento es un número que será asignado al contador de programa, de manera que la ejecución del programa empezará a partir de la instrucción que se encuentre en la localidad de memoria con ese número.

Si el segundo y tercer argumento son omitidos, el programa sería cargado a partir de la localidad cero de su memoria, y su ejecución empezaría en esa dirección.

3.3. ESTRUCTURAS DE DATOS.

Las estructuras de datos externas que maneja el simulador son las siguientes:

- Un arreglo de enteros de 16 bits que simula a la memoria en donde se carga el programa a ejecutarse. En cada posición del arreglo se almacenan dos bytes, de manera que si el arreglo está declarado con 16 K de tamaño, puede almacenar 32 Kbytes. Cada acceso para lectura o escritura de memoria se hace por medio de un simple algoritmo de conversión.

- Un entero que siempre contendrá el código de operación de la instrucción en proceso de ejecución. Esta variable, al igual que el arreglo que simula a la memoria, pudieron haber sido declarados como caracteres. Esto hubiese dificultado un poco la programación del simulador debido a que bajo la conversión de caracteres a enteros en la ONIX (lo cual se hubiese necesitado en algunas ocasiones), se conserva el signo. Esto es, si un caracter tiene un uno en el bit más significativo, se convertiría en un entero negativo y tendría prendidos los bits más significativos. Esto no sucede en todas las computadoras (en la Columbia PC no sucede), y la diferencia se debe a que el lenguaje C no especifica si los caracteres poseen signo (aunque garantiza que ningún caracter del juego de caracteres de la instalación será negativo), de manera que esto depende de las características de la arquitectura de cada máquina.

- Un arreglo de 16 enteros largos (de 32 bits), los cuales almacenan los contenidos de los registros de propósito general, del contador de programa y del apuntador de stack. De estos 32 bits de cada entero, solo se utilizan los 24 menos significativos.

- Un entero de 32 bits que contendrá la dirección de la tabla de mapeo de páginas en memoria del programa en ejecución, cuando se utiliza el sistema de relocalización. Solo se utilizan los 24 bits menos significativos.

- Un entero de 16 bits, de los cuales los últimos 12 son utilizados para almacenar a las banderas de condición y de control.

- Un arreglo de estructuras que almacena, antes de la ejecución de cada instrucción, el operando (en caso de que solo exista uno, se utiliza una de las estructuras), o los operandos (en cuyo caso se utilizan las dos estructuras) si la instrucción los requiere. Cada instrucción consiste de un entero largo para almacenar al operando, un entero largo para almacenar el lugar en donde se encuentra (ya sea en memoria o en registros) y un caracter que indica con el número uno que

el operando es inmediato, con el dos que éste está en memoria, y con el tres, que se encuentra en algunos de los registros.

3.4. FUNCIONES.

El simulador consta de una función principal, de 19 funciones de utilería y de una función que es la que se encarga de la simulación de la ejecución de cada una de las instrucciones del conjunto del AR-24.

La función principal abre el archivo de datos (el programa cuya ejecución se simulará), inicializa el contador de programa y el apuntador de stack y llama a la función para cargar la memoria y a la función para la decodificación.

La función para cargar la memoria convierte los caracteres ASCII del programa a ejecutarse, en dígitos binarios, acomodándolos en el arreglo que simula la memoria.

La función de decodificación se ejecuta hasta que sea detectada la instrucción PARA, la cual detiene al procesador y por lo tanto al simulador. Esta función toma de la memoria, por medio de la función de obtención de 8 bits, una instrucción.

Después se procede a la obtención de los operandos. Los dos primeros bits de la instrucción indican si ésta requiere de 0, 1 o 2 operandos. La función que obtiene los operandos es llamada una o dos veces según el caso, y almacena los operandos con su localización y su tipo, en el arreglo de estructuras definido para esto.

Entonces se hace una llamada a la función de ejecución para que haga su trabajo. Después se verifica si el procesador está en modo de rastreo, para que en este caso, se detenga el simulador cándole al programador algunas opciones de verificación.

Debido a que en algunas ocasiones los operandos se encuentran en palabras de extensión de las instrucciones (pues algunos modos de direccionamiento así lo requieren), se hace necesario definir a la función que obtiene 24 bits de la memoria, la cual, en realidad es un trío de llamadas a la función que obtiene un byte. Por cierto, esta última función, para su eficaz funcionamiento, verifica primero el modo de funcionamiento de la memoria. Si ésta está en modo de paginación, habrá que accesar al PMTP (apuntador a las tablas de mapeo de páginas) y por lo tanto, a las tablas que dan la distribución de las páginas de un programa en los marcos de página en memoria, las cuales indican si la página está o no en memoria, y si está, el número de marco en donde se encuentra y cual es su clave de protección. En caso de que la página no esté, se produce un "page fault" que debe ser manejado por el sistema operativo. El simulador no lleva a cabo ninguna acción en este caso, pero lo lógico sería

transferir el control del sistema operativo a la rutina diseñada para el caso. Algo similar debe ocurrir en caso de que la clave de protección no permita el acceso a la página.

La función de ejecución está compuesta de un conjunto de bloques, cada uno de los cuales agrupa de entre 8 a 16 pequeños procedimientos. Cada procedimiento es la simulación de la ejecución de una instrucción del conjunto del AR-24.

Las instrucciones están en orden secuencial ascendente, de la 00 al FF hexadecimal. Las instrucciones con código de operación inexistente detienen la ejecución del programa y del simulador.

Se hubiesen podido utilizar otros métodos para agrupar las rutinas de ejecución dentro de esta función, de manera que el código en C con que se escribió el simulador, fuera un poco más eficiente, pero la forma secuencial que se utilizó es, probablemente, la más clara.

Los pequeños procedimientos son, en general, bastante sencillos. Hacen uso de unas pocas funciones del exterior. Entre éstas están la que escribe un byte a memoria, la cual, similar a la que obtiene un byte de ésta, verifica si el procesador está trabajando en modo de paginación, para que, si éste es el caso, cheque si la página que contiene el byte está en memoria, en que dirección está, cual es la clave de protección de la página, y además actualice el bit que indica que la página ha sido modificada.

Otra función utilizada es la que escribe una palabra (3 bytes) en memoria, la cual es una triple llamada a la función descrita anteriormente. Rutinas como POP, POPR y RET utilizan a esta función.

Algunas rutinas que resultaron un poco tediosas de hacer fueron aquellas que incluyen operaciones aritméticas. Esto se debió a que la representación de los números en la ONYX (igual que en la mayoría de las máquinas) se hace con la notación de complemento a dos, mientras que la notación utilizada por el AR-24 es la de magnitud con signo.

Las instrucciones INC y DEC utilizan para su simulación a una misma función cuyo parámetro indica si el operando debe ser incrementado o decrementado en una unidad.

Las instrucciones de suma, resta, multiplicación, división, suma con acarreo, resta con préstamo, y de comparación, hacen uso de una misma función que también, con un parámetro, ejecuta la operación aritmética que la invocó. La comparación es una simple resta cuyo resultado no es almacenado.

Las cuatro instrucciones de aritmética de punto flotante también hacen uso de una misma función de manera similar a las antes descritas. Los resultados de éstas son siempre normalizados después de la operación. Estas instrucciones no hacen uso de las funciones de actualización de banderas al igual que casi todas las demás operaciones que requieren de esto. Las otras excepciones son las instrucciones de rotación y de corrimiento, las cuales actualizan a la bandera de acarreo por su cuenta.

Las funciones de actualización de banderas son cuatro. La de overflow, la de acarreo, la de signo y la de cero. La de overflow tiene un parámetro de más que indica a ésta si el operando es un caracter de 8 bits o una cantidad de 24.

A continuación se muestran los nombres que se le dieron a cada una de las funciones junto con los argumentos con que son llamadas en el simulador:

acarreo(x). Se actualiza la bandera de acarreo. El argumento es el valor a verificarse.

cargamem(x). Carga la memoria. El argumento indica desde que localidad se cargará ésta.

cero(x). Se actualiza la bandera de cero. El argumento es el valor a verificarse.

decodifi(). Decodifica instrucciones.

detener(). Da opciones de rastreo al programador.

ejecuta(). Ejecuta instrucciones.

esc(x,y). Escribe un byte en memoria. El argumento "x" contiene el byte a escribirse, y el argumento "y" indica en que lugar de la memoria se almacenará éste.

escpal(x,y). Escribe una palabra en memoria o en un registro de CPU. El argumento "x" contiene la palabra que se escribirá, y el operando "y" contiene un número de estructura (de alguna de las dos que contienen los operandos y los lugares donde están estos).

fetch(x). Obtiene un byte de la memoria. El argumento indica la dirección de memoria de donde se obtendrá este byte.

instil(). Despliega mensaje por instrucción ilegal.

leehex(). Lee un caracter hexadecimal del archivo que contiene al programa que va a ser ejecutado.

leelong(). Lee seis dígitos hexadecimales de pantalla.

leepal(x). Lee una palabra de memoria. El argumento indica a partir de que localidad de memoria se obtendrán estos tres bytes.

main(x,y). Rutina principal. El argumento "x" indica el número de parámetros con que es llamado el simulador al tiempo de ejecución. El argumento "y" es un arreglo que contiene el valor de cada uno de estos parámetros.

noaccpag(x). Imprime mensaje por imposibilidad de acceso a la página requerida. El argumento indica si esto se debió a un "page fault" o bien a la clave de protección de la página.

obtoper(x). Obtiene el operando de la instrucción. El argumento indica si éste se trata del primero o del segundo operando.

overflow(x,y). Actualiza la bandera de overflow. El argumento "x" es el valor que se verificará, y el argumento "y" indica si este valor es de 8 o de 24 bits.

signo(x). Actualiza la bandera de signo. El argumento es el valor a verificarse.

util1(x). Ejecuta operaciones de incremento y decremento. El argumento indica de que operación se trata.

util2(x). Ejecuta operaciones de suma, resta, multiplicación, división, suma con acarreo, resta con préstamo, y comparación. El argumento indica de que operación se trata.

util3(x). Ejecuta operaciones de punto flotante. El argumento indica de que operación se trata.

En la siguiente página se muestra el diagrama estructural de estas funciones.

```
main    cargamem    leehex
        decodifi    fetch       noaccpag
                    obtoper     fetch       noaccpag
                                leepal      fetch       noaccpag

                    ejecuta     overflow
                                signo
                                cero

                                util1       overflow
                                            signo
                                            cero
                                            escpal      esc   noaccpag

                                util2       acarreo
                                            overflow
                                            signo
                                            cero
                                            escpal      esc   noaccpag

                                util3       escpal      esc   noaccpag
                                esc         noaccpag
                                escpal      esc         noaccpag
                                fetch       noaccpag
                                leepal      fetch       noaccpag
                                instil

                    detener     fetch       noaccpag
```

3.5. EL RASTREADOR.

Una característica que tiene el simulador que vale la pena recalcar es la facilidad que da para el rastreo de programas a tiempo de ejecución.

Normalmente, cuando la bandera de rastreo está prendida, una interrupción debe ser generada y el registro 14 (PC) debe ser cargado con la dirección especificada en las localidades 2 a 5 de la memoria, de manera que debe existir una subrutina en esta dirección que defina la acción a tomar en esta situación. Dado que en la mayoría de los casos lo que se desea es desplegar el contenido de los registros o de la memoria, el simulador tiene una función que evita al usuario escribir dicha rutina.

Entonces si la bandera de rastreo está prendida, el simulador permite al programador escoger alguna opción que le permitirá verificar el funcionamiento del programa que se está ejecutando.

Para prender la bandera de rastreo, se utiliza la instrucción CMRA.

Una de estas opciones permite desplegar los registros de propósito general, el PC, el SP, la PSW, y el PHTP.

Otra opción hace que se despliegue en pantalla alguna región de la memoria. El programador debe indicar al simulador cual es la región que requiere.

Hay una opción que permite abortar el programa.

Otra opción permite deshabilitar el modo de rastreo. Esta indica al simulador que debe apagar la bandera de rastreo en la PSW.

Por último, existe una opción que hace que la siguiente instrucción del programa sea ejecutada.

Siempre que el modo de rastreo esté habilitado, el código de operación de la instrucción en ejecución, será desplegado en pantalla.

3.6. PRUEBAS.

Para garantizar el buen funcionamiento del simulador, éste fue probado en dos fases diferentes.

La primera fase fue la que tomó más tiempo. Consistió en probar cada una de las instrucciones del AR-24 por separado. Fue fácil hacer los programas de prueba pues eran muy pequeños. Algunos de ellos consistían de una sola instrucción: la que se estaba probando. Otros probaban una sola instrucción, pero en la mayoría de los casos diferentes que podía tener. Lo tardado de esto fue que muchas veces era tedioso comprobar los resultados.

La segunda fase fue la más importante y tuvo un doble propósito. Consistió en simular la ejecución de programas con un cierto grado de complejidad. Esto demostró el correcto funcionamiento del simulador, y más importante, la flexibilidad y facilidad de programar con las instrucciones y modos de direccionamiento del AR-24.

Comprobar los resultados de la segunda fase fue muy sencillo. Escribir los programas resultó una tarea no tan tediosa como generalmente lo es escribir programas en lenguaje máquina. Hay que decir que la opción de rastreo fue de gran utilidad para la depuración de estos.

A continuación se dan tres de los programas que se utilizaron en la segunda fase. Se dan en lenguaje máquina con breves comentarios en cada instrucción. Antes de algunas instrucciones fueron incluidas las direcciones de éstas para hacer más fácil la lectura de los programas.

En estos programas se utilizan varias instrucciones de cada uno de los seis tipos de instrucciones, además se utilizan todos los modos de direccionamiento, aunque hay que admitir que el modo indirecto de memoria se utilizó a la fuerza pues no se necesitaba.

Hubo en el simulador nada más un detalle que no fue probado. Esto fue el algoritmo de conversión de direcciones lógicas a direcciones físicas cuando la memoria se está accesando por paginación. No fue probado, pues se hubiese necesitado un pequeño monitor que por no estar entre los objetivos de éste trabajo, no se fabricó. Pero hay que aclarar que el algoritmo es, en términos de programación, bastante sencillo, y no debe presentar mayor problema.

3.6.1. PROGRAMA QUE OBTIENE EL PRODUCTO ESCALAR.

Este programa calcula el producto escalar de dos vectores. Estos se encuentran antes del programa y su dimensión es de seis palabras. El programa despliega después el resultado en la pantalla, en dígitos octales. El programa escrito en C sería de la siguiente manera:

```
main()
{
   int ind;                  /* Indice */
   int acum;                 /* Acumulador */
   int aux;                  /* Variable auxiliar */
   int arr1[6],arr2[6];      /* Arreglos */

   /* Inicialización de los arreglos */
   acum=0;                   /* Se inicializa el acumulador */

   for(i=0;i<6;i++)
      acum=acum+arr1[ind]*arr2[ind];

   for(i=5;i>=0;i--)  /* Se imprime el resultado en octal */
   {                  /* con ceros en donde los haya */
      aux=acum>>(4*i);
      aux=aux&0xf;
      printf("%d",aux);
   }

   printf("\n");
}
```

El programa en lenguaje máquina del AR-24 es el siguiente:

```
00   6c af            /* BRI (brinca) a la localidad 2fH

02   000000           /* localidad auxiliar
05   000004           /* Dirección de la localidad auxiliar

08   000006

0B   000009           /* datos del primer vector
     000003
     00000a
     000012
```

```
      000005
      000004

1D    000001            /* datos del segundo vector
      000015
      000003
      000007
      000008
      00000c

2F    a4 88 20          /* MOV valor 8 al registro 0
      a4 8b 22          /* MOV valor b al registro 2
      a4 40 23          /* MOV indirección del R0 al R3
      84 83 23          /* MUL valor 3 por reg. 3
      80 22 23          /* SUM reg. 2 a reg. 3
      a4 80 24          /* MOV valor 0 a reg. 4
      a4 40 21          /* MOV indirección de R0 a R1

44    a4 72 25          /* MOV R2 con autoincremento a R5
      84 73 25          /* MUL R3 con autoincremento por R5
      80 25 24          /* SUM reg. 5 a reg. 4
      a6 c4 81          /* ITERA a dir. 44H con reg. 1

50    a4 88 26          /* MOV val. 8 a reg. 6
53    a4 26 27          /* MOV reg. 6 a reg. 7
      41 27             /* DEC reg. 7
      84 83 27          /* MUL val. 3 por reg. 7
      a4 24 28          /* MOV reg. 4 a reg. 8
      90 28 27          /* CODER a reg. 8, reg. 7 veces
      8d 87 28          /* Y de val. 7 y reg. 8
      80 b0 28          /* SUM va. 30 con reg. 8
      a4 28 30 000002   /* MOV reg. 8 a dir. 000002H
      4d 50 000005      /* ESC indirección de dir. 000005H
      a6 d3 86          /* ITERA a dir. 000053H con reg. 6
      4d 10 000001      /* ESC val. en dir. PC+1
      04                /* PARA
      0a                /* código del avance de linea
```

3.6.2. PROGRAMA DE BUSQUEDA BINARIA.

El segundo programa intenta encontrar un cierto valor en un arreglo ordenado con dimensión 13, por medio del método de búsqueda binaria. El programa despliega como resultado una V si encuentra el valor, y una F si no. El programa en lenguaje C sería de la siguiente forma:

```
main()
{

   int valor,          /* Valor a buscarse en el arreglo */
       indinf,         /* Indice inferior */
       indsup,         /* Indice superior */
       aux;            /* Variable auxiliar */
   int arr[13];        /* Arreglo ordenado */

   /* Inicialización del arreglo */
   /* Asignación a x del valor a buscarse */
   indinf=1;
   indsup=13;

   do {
      aux=(indinf+indsup)/2;
      if(valor>arr[aux])
         indinf=aux+1;
      else
         indsup=aux-1;
      } while(arr[aux]!=valor && indinf<=indsup)

   if(arr[aux]==valor)        /* Si el valor fue encontrado */
      printf("V\n");

   if(indinf>indsup)          /* Si no fue encontrado */
      printf("F\n");
}
```

El programa en lenguaje máquina:

```
00   6c ae              /* goto dir. 00002eH

02   56                 /* val. de V
03   46                 /* val. de F

04   000001             /* val. a buscar en el arreglo

07   000005             /* vector ordenado
     000001
     000010
     000022
     000041
     000045
     000052
     000066
     0000a3
     000100
     000151
```

```
      0001cd
      000209

2E    a4 87 24          /* a=7
      a4 30 000004 23   /* x=-1
      a4 81 20          /* i=1
      a4 8d 21          /* j=13

3D    a4 20 22          /* k=i
      80 21 22          /* k=k+j
      85 82 22          /* k=k/2=(i+j)/2
      a4 22 25          /* z=k
      41 25             /* z=z-1
      84 83 25          /* z=z*3
      80 24 25          /* z=z+a=(k-1)*3+a
      93 23 45          /* mem(z)=a[k]. Compara x con a[k]
      64 00 000063      /* if x<=a[k] goto dir. 63H
      a4 22 20          /* i=k
      40 20             /* i=i+1=k+1
      6c 00 000068      /* goto dir. 68H

63    a4 22 21          /* j=k
      41 21             /* j=j-1=k-1

68    93 45 23          /* Compara a[k] con x
      62 00 00007a      /* if a[k]=x goto dir. 7AH
      93 20 21          /* Compara i con j
      60 00 000084      /* if i>j goto dir. 84H
      6c bd             /* goto dir. 3DH

7A    4d 30 000002      /* printf("V")
      6c 00 000089      /* goto dir. 89H

84    4d 30 000003      /* printf("F")
89    4d 10 000001      /* printf("\n")
      04                /* PARA
      Ca
```

3.6.3. PROGRAMA DE LAS TORRES DE HANOI.

Por último, el tercer programa obtiene la solución al problema de las torres de Hanoi con tres discos. Este programa aunque pequeño, es muy ilustrativo, pues es recursivo y su subrutina es llamada con parámetros.

Los parámetros de la subrutina Torres son los siguientes: el

primero es la cantidad de discos que se quieren mover; el segundo es el número de varilla desde donde se quieren mover los discos; el tercero es el número de la varilla a donde se quieren mover los discos.

En lenguaje C, el programa se vería de la siguiente forma:

```
main()
{
   torres(3,1,3);
}


torres(num,varini,varifin)
int num,varini,varifin;
{
   int otravari;

   if(num==1)
      printf("\n MUEVE DISCO DE %d A %d",varini,varifin);
   else
   {
      otravari=6-varini-varifin;
      torres(num-1,varini,otravari);
      torres(1,varini,varifin);
      torres(num-1,otravari,varifin);
   }

}
```

Este es el programa escrito en lenguaje máquina del AR-24:

```
00    6c 94                     /* goto dir. 14H

02    0a4d5545564520444953      /* val. "MUEVE DISCO DE A "
      434f204445204120

14    4a 83                     /* push 3 (n)
      4a 81                     /* push 1 (i)
      4a 83                     /* push 3 (j)
      6d a0                     /* torres(3,1,3)
      80 89 2f                  /* SP=SP+9
      04                        /* PARA

20    a7 80 83                  /* push SPA (R0); SPA=SP; SP=SP-3
      a4 81 21                  /* aux=1
```

```
      93 60 00000c 21        /* Compara n con 1
      67 c5                  /* if n!=1 goto dir. 45H
      4a 60 000009           /* push i
      4a 60 000006           /* push j
      6d 00 0000a0           /* imprimemov(i,j)
      80 86 2f               /* SP=SP+6
      6c 00 00009d           /* goto dir. 9DH

45    a4 86 21               /* k=6
      82 60 000009 21        /* k=k-i
      82 60 000006 21        /* k=k-j=6-i-j
      a4 21 60 800003        /* push k (respecto a SPA)
      a4 60 00000c 21
      41 21                  /* n=n-1
      4a 21                  /* push n
      4a 60 000009           /* push i
      4a 60 800003           /* push k
      6d a0                  /* torres(n-1,i,k)
      80 89 2f               /* SP=SP+9
      4a 81                  /* push 1
      4a 60 000009           /* push i
      4a 60 000006           /* push j
      6d a0                  /* torres(1,i,j)
      80 89 2f               /* SP=SP+9
      a4 60 0000cc 21
      41 21                  /* n=n-1
      4a 21                  /* push n
      4a 60 800003           /* push k
      4a 60 000006           /* push j
      6d a0                  /* torres(n-1,k,j)
      80 89 2f               /* SP=SP+9

9D    4e 80                  /* SP=SPA; pop SPA
      01                     /* RET

A0    ad 82 90               /* printf("MUEVE DISCO DE ")
      80 b0 6f 000006        /* i=i+30H
      4d 6f 000008           /* printf("%d",i)
      ad 91 83               /* printf(" A ")
      80 b0 6f 000003        /* j=j+30H
      4d 6f 000005           /* printf("%d",j)
      01                     /* RET
```

CONCLUSIONES.

Generalmente, las arquitecturas de los procesadores que están al alcance de los estudiantes del área de Computación no están diseñados de manera que les puedan ofrecer facilidades didácticas.

Los puntos que se podrían decir más importantes de la arquitectura de un procesador, son los siguientes:

- Registros disponibles.
- Formato de las instrucciones.
- Modos de direccionamiento.
- Conjunto de instrucciones.
- Organización de la memoria.

La importancia de estos puntos es para las personas que hacen software en lenguaje máquina o ensamblador, esto es, para los programadores de Sistemas.

En el diseño del procesador AR-24 se tomaron en cuenta cada uno de estos puntos, de manera que pudieran ofrecer simplicidad en la forma en que se manejan, y al mismo tiempo tuvieran características que hicieran al procesador suficientemente poderoso.

El procesador tiene 14 registros de propósito general, un apuntador de stack y un contador de programa que son manipulables por el programador. Los programas de aplicación comunes bien estructurados, con poca frecuencia utilizan más de 14 variables locales (sin incluir arreglos y estructuras), por lo que con la existencia de los 14 registros, se puede evitar el uso de la memoria para almacenar variables. Los registros se pueden utilizar en cualquiera de las instrucciones de una manera muy sencilla.

Los formatos de las instrucciones son muy simples en este procesador. La ortogonalidad de códigos de operación y operandos es la característica principal en el diseño de los formatos. La ortogonalidad indica que todas las instrucciones constan de un código de operación y cero, uno o dos campos de operando, en donde todos los campos de operando se especifican de la misma manera. En la mayoría de los microprocesadores, los formatos de las instrucciones son variados y confusos. Esto es así porque los diseñadores se tuvieron que enfrentar a las restricciones que implican el espacio y en general, la

eficiencia. Por ejemplo, el MC68000 tiene al menos 18 formatos básicos diferentes en sus instrucciones, el Z80 tiene 26.

En cambio el AR-24 tiene solo 3 formatos básicos de instrucción. Cada instrucción posee un byte para el código de operación, más un byte por cada operando que requiera. A veces, cada operando puede requerir de una palabra (3 bytes) de extensión. Los dos primeros bits del código de operación indican si la instrucción es de cero, uno o dos operandos.

Los modos de direccionamiento en el AR-24 son también muy fáciles de manejar y bastante poderosos. Habíamos dicho que por cada operando que tiene la instrucción, se incluye un byte en ésta. A éste se le llama byte especificador del operando. La mitad de este byte se utiliza para especificar cual de los nueve modos de direccionamiento se utilizará para accesar al operando. La otra mitad se utiliza para especificar un número de registro (de los 16 que hay en la CPU, el SP y el PC inclusive), cuando el modo de direccionamiento así lo requiere. A veces el modo de direccionamiento requerirá una palabra de extensión, la cual se colocará inmediatamente después de este byte. Existe un modo especial en donde un operando corto se incluye en el byte especificador.

La ventaja del sistema de modos de direccionamiento de este procesador estriba en que cada instrucción puede, siempre que esto sea coherente, utilizar cualquier modo de direccionamiento para el acceso de sus operandos. Además, si la instrucción es de dos operandos, estos podrán, si se desea, ser accesados por modos distintos. En la mayoría de los procesadores esto no es posible.

Además la variedad de modos del AR-24 son de fácil manejo. Basta indicar el número del modo en la mitad del byte especificador. Esto tampoco se da en todas las máquinas. En la PDP-11 por ejemplo, para accesar un operando con el modo absoluto, hay que invocar al modo de autoincremento indirecto en conjunción con el registro que contiene al contador del programa.

En cuanto a tipos de instrucciones se refiere, el AR-24 puede ejecutar la mayoría de las instrucciones que podría ejecutar cualquier otro procesador. Posee instrucciones aritméticas, lógicas, de transferencia, de control de programa, de control de procesador, de entrada y salida, y de alto nivel.

Por otro lado, algunas de sus instrucciones se encuentran solo en un reducido grupo de procesadores.

Es conocida la importancia de los procedimientos en los programas bien estructurados. Para que un procedimiento sea

recursivo, es necesario almacenar sus variables en algún lugar. El lugar ideal es el stack. Entonces cada vez que un procedimiento es llamado, es necesario reservar en el stack un espacio para los parámetros, la dirección de retorno y las variables locales de él. Debido a que es posible utilizar el stack al entrar a otro procedimiento sin haber terminado con el primero, necesitamos una dirección de referencia para poder accesar los parámetros y las variables, al regresar a éste. Esta dirección no puede ser el apuntador del stack, pues puede variar. Entonces necesitamos un lugar para conservarla. A veces a esta dirección se le llama apuntador de base local (local base) y se puede almacenar en algún registro de CPU.

Al entrar a un procedimiento es necesario almacenar el apuntador de base local anterior en el stack (para que se pueda recuperar a la salida de otro procedimiento posterior), copiar el apuntador del stack al apuntador de base local, y decrementar o incrementar el SP (dependiendo de la manera en que crezca el stack) para reservar espacio para las variables locales. Al salir del procedimiento es necesario copiar el apuntador de base local al SP y restaurar (sacando del stack) el antiguo apuntador de base local.

Algo muy importante en una computadora es la rapidez con la que puede ejecutar los pasos para entrar y salir de un procedimiento. En el AR-24 una instrucción ejecuta los pasos de entrada (LIGA) y otra instrucción los de salida (DESLI).

Por otro lado las computadoras no solo se utilizan para procesar cantidades numéricas, también ejecutan operaciones sobre cadenas. No siempre se ha tomado esto en cuenta al diseñar un procesador. En el AR-24 existen 5 instrucciones que operan sobre cadenas almacenadas en memoria.

Existe una instrucción que mueve (copia) cadenas de un lado a otro de la memoria. Existe otra instrucción que las compara. También hay una que analiza una cadena en busca de un byte determinado. Por último hay dos instrucciones, una para manejar la entrada y otra para manejar la salida de cadenas desde y hacia el puerto de salida.

Otras ventajas que se pueden mencionar acerca de las instrucciones de este procesador se refieren a las operaciones aritméticas. Todas ellas operan sobre números con la notación de magnitud con signo. Esto facilita el trabajar con números a la hora de la teoría.

Además el procesador posee 4 instrucciones cuyos operandos son tomados como números de punto flotante. Estas instrucciones son las operaciones aritméticas básicas: suma, resta, multiplicación y división.

Ahora pasemos al último punto que es el de la organización de la memoria. Posiblemente la arquitectura de memoria más fácil de manejar sea la lineal. Es por eso que la del AR-24 fue diseñada de esta manera. Gracias a que la palabra del AR-24 es de 24 bits, éste es capaz de manejar un espacio de direccionamiento lógico de aproximadamente 16 Mbytes. El procesador 8086 por ejemplo, tiene que utilizar el menos simple esquema de memoria segmentada, probablemente, para solucionar el problema del almacenamiento de direcciones de memoria (pues es un procesador de 16 bits que linealmente solo podría tener un espacio de direccionamiento de 64 Kbytes).

La tecnología en el campo de la electrónica ha avanzado a pasos agigantados en los últimos años, por lo que las memorias de acceso directo cada vez se pueden construir a precios más bajos. Pero no ha llegado aún el día en que una memoria de 16 Mbytes sea accesible para cualquier persona. Por eso, es necesario que las computadoras con grandes espacios de direccionamiento lógico diseñadas para aprovechar al máximo sus características, posean un sistema de memoria virtual.

En este procesador, el sistema de memoria virtual está apoyado en el esquema de mapeo basado en páginas. Este esquema facilita también, combinado con algunos detalles técnicos, la multiprogramación y la protección de memoria y es, hasta cierto punto, sencillo de comprender y manejar.

Ahora bien, alguien que se convenciera de las ventajas de este procesador sobre la máquina a que tiene acceso, podría decir que sería difícil tener acceso a éste. Pero es por ese punto por donde podría comenzar un curso de Programación de Sistemas.

Un buen primer proyecto de este curso sería la construcción de un programa que simule el funcionamiento del procesador. Esta tarea no implicaría muchas dificultades debido a que el funcionamiento del procesador puede verse desde una manera muy estructurada. Hasta sería posible hablar de ventajas al hacer el simulador de este procesador en lugar de hacer el simulador de otro procesador, digamos el Z80.

Antes decíamos que en la mayoría de los microprocesadores, en particular el Z80, el formato de las instrucciones es muy variado debido a restricciones de espacio. Puede haber mucha pérdida de tiempo de la persona que esté diseñando un simulador en la, desde mi punto de vista, intrascendental tarea de descifrar como se las ingeniaron los diseñadores del Z80 para acomodar todas las instrucciones en tan pocos bits. Esto desde luego, si se quiere que el programa simulador no sea solamente un conjunto de rutinas en donde cada una de ellas interpreta el valor de un conjunto de bits.

Regresando al curso de Programación de Sistemas, después del primer proyecto ya todos los estudiantes tendrían acceso al AR-24, sin importar con que computadora estuvieran trabajando. Entonces se podría tener como un segundo proyecto, la construcción de un programa ensamblador para este procesador.

Una de las ventajas que podría tener el hacer este ensamblador en lugar de hacer un ensamblador para el Z80, estriba en que los modos de direccionamiento del AR-24 son explícitos, mientras que los del Z80 están implícitos en las instrucciones, por lo que el estudiante tendría que construir un tipo de ensamblador más general. Además del ya mencionado hecho de lo estructurado del formato de las instrucciones, lo cual permite centrar el problema del diseño en el analizador lexicográfico (scanner) más que en el procedimiento de traducción.

Si pudiéramos seguir hablando de proyectos del o de los cursos de Programación de Sistemas, sería posible hablar de la construcción de un compilador que genere código en ensamblador o en lenguaje máquina del AR-24.

En un curso de Sistemas Operativos, la organización de la memoria de este procesador sería de gran ayuda para explicar y comprender el problema del manejo de memoria, tema al que muchas veces se le pone mucho énfasis durante este curso. Es más, sería interesante la construcción de un monitor o pequeño sistema operativo, que fuese capaz de manejar memoria virtual, así como multiprogramación, y el sistema de protección de memoria.

Hay que admitir que diseñar el monitor no sería una tarea fácil, puesto que cuando menos, para manejar multiprogramación, es necesario tener que interrumpir al procesador cada quántum de tiempo para que todos los procesos puedan tener acceso a él. Y puesto que estamos hablando de un simulador en software, las interrupciones externas (por ejemplo de un reloj) tendrían que ser simuladas de distintas maneras, dependiendo del caso. La simulación de interrupciones podría dividirse en dos grupos: síncronas y asíncronas.

En un curso de Arquitectura de Computadoras, un repaso del Capítulo Uno de este trabajo sería una buena introducción. En este capítulo se da una definición amplia de la materia, luego se habla un poco de la evolución de las computadoras (a partir de la EDVAC) y por último se dan los principios básicos de la arquitectura de las computadoras.

Un buen proyecto final de esta materia, sería la construcción de un microprograma emulador del AR-24. No es fácil tener acceso a una máquina microprogramable, pero sí es relativamente fácil construir el simulador de una de ellas.

Por último podemos hablar del curso de Lenguaje Ensamblador. Este curso, que se da a nivel licenciatura, es necesario para los estudiantes que tratan de conocer cómo funciona una computadora.

La tercera sección del primer capítulo de este trabajo podría ser de utilidad para ese curso. Y con la ayuda de un simulador y un ensamblador, sería posible hacer del AR-24 el centro de atención del curso. Para esto, sería necesario el Capítulo Dos de este trabajo, el cual describe la arquitectura del procesador y da los nemónicos que se podrían utilizar para su ensamblador.

En este trabajo también se proporciona un simulador del procesador. El programa fue escrito en lenguaje C, pues desde mi punto de vista, es el mejor lenguaje para la Programación de Sistemas. No se hizo el intento de optimizar el código, sino que se trató de presentarlo con claridad, de manera que hasta cierto punto pueda remplazar la descripción del procesador que se da en el Capítulo Dos. Los detalles de cómo fue diseñado el simulador pueden ser encontrados en el Capítulo Tres.

Por último hay que mencionar que el sistema de rastreo incluido en el simulador puede ser de gran ayuda para la depuración de programas escritos para este procesador.

BIBLIOGRAFIA.

- G. Amdahl, G. Blaauw, F. Brooks. "Architecture of the IBM System/360" IBM J. Res. and Dev.. April, 1964.

- J. Backus. "Can Programming be Liberated from the vonNeumann Style? A Functional Style and its Algebra of Programs". Comm. of ACM. August, 1978.

- Y. Chu. High-Level Language Computer Architecture. Academic Press. New York. 1975.

- H. Deitel. An Introduction to Operating Systems, Revised First Ed. Addison-Wesley. Reading, Mass. 1984.

- J. Donovan. Systems Programming. McGraw-Hill. New York. 1972.

- Electronics. "Intel's 80386 Runs Multiple Operating Systems". Electronics. October. 1985.

- W. Gear. Computer Organization and Programming, Third Ed.. McGraw-Hill. New York. 1980.

- V. Hamacher, Z. Vranesic, S. Zaky. Computer Organization. McGraw-Hill. New York. 1978.

- J. Hayes. Computer Architecture and Organization. McGraw-Hill. New York. 1978.

- C. Hunter, E. Farquhar. "Introduction to the NS16000 Architecture". EDN Magazine. April, 1984.

- IBM. IBM System/370, Principles of Operation, Systems. GA22-7000-6. 1980.

- Intel. iAPX 86, 88, 186 and 188 User's Manual, Programmer's Reference. Intel, Santa Clara, CA. 1983.

- G. Kane. 68000 Microprocessor Handbook. Osborne/McGraw-Hill. Berkeley, CA. 1981.

- B. Kernighan, D. Ritchie. The C Programming Language. Prentice-Hall. Englewood Cliffs, N.J. 1978.

- D. Knuth. The Art of Computer Programming, Vol. I, Fundamental Algorithms. Addison Wesley. Reading, Mass. 1968.

- D. Knuth. "An Empirical Study of FORTRAN Programs". Software, Practice and Experience. March, 1971.

- G. Levine. Introducción a la Computación y a la Programación Estructurada. McGraw-Hill. México. 1984.

- D. MacGregor, D. Mothersole, B. Moyer. "The Motorola MC68020". IEEE Micro. August, 1984.

- S. Madnick, J. Donovan. Operating Systems. McGraw-Hill. New York. 1974.

- T. Moto-Oka, K. Fuchi. "The Architectures in the Fifth Generation Computers". Proc. IFIP. North Holland. New York. 1983.

- E. Organick. Computer System Organization, The B5700/B6700 Series. Academic Press. New York. 1973.

- D. Patterson, C. Sequin. "RISC I: A Reduced Instruction Set VLSI Computer". IEEE. 1981.

- M. Ruggiero, S. Zaky. "A Microprocessor-based Virtual Memory System". The 7th Annual Symposium on Computer Architecture, Conference Proc. IEEE, ACM. 1980.

- R. Russell. "The PDP-11; A Case Study of How Not to Design Condition Codes". The 5th Annual Symposium on Computer Architecture, Conference Proc. IEEE, ACM. 1978.

- D. Siewiorek, G. Bell, A. Newell. Computer Structures: Principles and Examples. McGraw-Hill. New York. 1982.

- P. Stanley. "Address Size Independence in a 16-bit Minicomputer". The 5th Annual Symposium on Computer Architecture, Conference Proc. IEEE, ACM. 1978.

- A. Tanenbaum. Structured Computer Organization, Second Ed.. Prentice Hall. Englewood Cliffs, N.J. 1984.

- J. Ullman. Fundamental Concepts of Programming Systems. Addison-Wesley. Reading, Mass. 1976.

APENDICE A. GLOSARIO DE LAS INSTRUCCIONES DEL PROCESADOR AR-24.

A continuación se dan las instrucciones del procesador AR-24. Se dan el nemónico de la instrucción, su codificación en dígitos hexadecimales, su codificación en dígitos binarios, y la relación de las banderas que afectan.

Las banderas están referenciadas por sus iniciales. (a=acarreo, o=overflow, s=signo, c=cero, m=modo de memoria, p=modo del procesador, r=rastreo).

INTRUCCIONES ARITMETICAS.

INTRUCCION	CODIGO HEX	CODIGO BINARIO	BANDERAS AOSC	BANDERAS HPR
SUM X,Y	80	10000000	XXXX	---
RES X,Y	82	10000010	XXXX	---
MUL X,Y	84	10000100	-XXX	---
DIV X,Y	85	10000101	--XX	---
SUMAC X,Y	81	10000001	XXXX	---
RESAC X,Y	83	10000011	XXXX	---
SUMF X,Y	86	10000110	-XXX	---
RESF X,Y	87	10000111	-XXX	---
MULF X,Y	88	10001000	-XXX	---
DIVF X,Y	89	10001001	-XXX	---
INC X	40	01000000	-XXX	---
DEC X	41	01000001	-XXX	---
NEG X	42	01000010	--XX	---
ABS X	43	01000011	--XX	---
MOD X,Y	8C	10001100	----	---

INSTRUCCIONES LOGICAS.

Y X,Y	8D	10001101	--XX	---
O X,Y	8E	10001110	--XX	---
OEX X,Y	8F	10001111	--XX	---
CODER X,Y	90	10010000	X-XX	---
COIZQ X,Y	91	10010001	X-XX	---
CODEA X,Y	92	10010010	X-XX	---
RODER X,Y	94	10010100	X-XX	---
ROIZQ X,Y	95	10010101	X-XX	---
RODEA X,Y	96	10010110	X-XX	---
ROIZA X,Y	97	10010111	X-XX	---
COMP X,Y	93	10010011	XXXX	---
INV X	44	01000100	--XX	---

INSTRUCCIONES DE TRANSFERENCIA.

MOV X,Y	A4	10100100	----	---
INTER X,Y	A3	10100011	----	---
MOVC X	A6	10100110	----	---
MOVU X	A7	10100111	----	---
PBIT X,Y	A8	10101000	----	---
ABIT X,Y	A9	10101001	----	---
CBIT X,Y	AA	10101010	----	---

BBIT X,Y	AB	10101011	---- ---
PUSH X	4A	01001010	---- ---
POP X	4B	01001011	---- ---
PUSHR	02	00000010	---- ---
POPR	03	00000011	---- ---
PUSHB	09	00001001	---- ---
POPB	0A	00001010	XXXX XXX
HPNTP X	45	01000101	---- ---

INSTRUCCIONES DE CONTROL DEL PROGRAMA Y DEL PROCESADOR.

BNA X	60	01100000	---- ---
BNE X	61	01100001	---- ---
BIGU X	62	01100010	---- ---
BNAI X	63	01100011	---- ---
BNEI X	64	01100100	---- ---
BAR X	65	01100101	---- ---
BAB X	66	01100110	---- ---
BDIF X	67	01100111	---- ---
BARI X	68	01101000	---- ---
BABI X	69	01101001	---- ---
BAC X	6A	01101010	---- ---
BNAC X	6B	01101011	---- ---
BRI X	6C	01101100	---- ---
BSUB X	6D	01101101	---- ---
INT X	6E	01101110	---- --X
BNAC X,Y	A0	10100000	---- ---
BNEC X,Y	A1	10100001	---- ---
BIC X,Y	A2	10100010	---- ---
PEA X	48	01001000	XXXX ---
ABA X	49	01001001	XXXX ---
CMPE	05	00000101	---- X--
CMPR	06	00000110	---- -X-
CMBA	07	00000111	---- --X
RET	01	00000001	---- ---
RETIN	0B	00001011	XXXX XXX
PARA	04	00000100	---- ---
NOP	00	00000000	---- ---

INSTRUCCIONES DE ENTRADA Y SALIDA.

LEE X	4C	01001100	---- ---
ESC X	4D	01001101	---- ---
LEEBL X,Y	AC	10101100	---- ---
ESCBL X,Y	AD	10101101	---- ---
ESCRE	08	00001000	---- ---

INSTRUCCIONES DE ALTO NIVEL.

MOVST X,Y	A5	10100101	---- ---
COMST X,Y	8A	10001010	-X-X ---
ANAST X,Y	8B	10001011	-X-X ---
ITERA X,Y	A6	10100110	---- ---
LIGA X,Y	A7	10100111	---- ---
DESLI X	4E	01001110	---- ---

APENDICE B. LISTADO DEL SIMULADOR DEL PROCESADOR AR-24.

A continuación se muestra el listado del programa que simula el funcionamiento del procesador AR-24. El programa fue escrito en lenguaje C.

```
/*  PROGRAMA QUE SIMULA EL FUNCIONAMIENTO DEL
    PROCESADOR AR-24.

    ARMANDO JOSE RIVERO MOLINA.

    24 DE FEBRERO DE 1986        */

/* El programa se debe ejecutar con tres argumentos:

   el primero es el nombre del programa en lenguaje
   maquina cuya ejecucion se simulara;

   el segundo indica al simulador la localidad de memoria
   desde la cual debe empezar a cargarce el programa cuya
   ejecucion se va a simular;

   el tercero indica al simulador con que cantidad debe
   inicializar al program counter.

   En caso de que no se den los dos ultimos argumentos,
   el simulador cargara el programa desde la localidad
   cero, e inicializara al program counter con cero.     */

#include <stdio.h>
#define TAMEM 0x4000
#define STACK TAMEM-2

int inst;                          /* Instruccion */
int mem[TAMEM];                    /* Memoria */
int psw;                           /* Registro de banderas */
long atap;                         /* Apuntador a las tablas de mapeo de paginas */
long reg[16];                      /* Registros de CPU */
FILE *fe;                          /* Apuntador al archivo de entrada */
struct ope { long op,lug;          /* Estructura para almacenar operandos */
             char tip; } oper[2];

main(argc,argv)
int argc;
char *argv[];
{
    long x,y;
    if((fe=fopen(argv[1],"r"))==0)
    {
        printf("\n No se pudo abrir el archivo\n");
        exit();
    }
    y=atoi(argv[2]);
    x=atoi(argv[3]);
    reg[14]=x;                     /* Inicializacion del PC */
    reg[15]=STACK;                 /* Inicializacion del SP */
    psw=0x400;
    cargamem(y);
    decodifi();
}
```

```
/* Rutina que lee las instrucciones de la memoria y las
   examina para determinar su numero de operandos */

decodifi()
{                       /* Se ejecuta hasta que se encuentra la instruccion PARA */
   while((inst=fetch(regE14)))!=4 || (psw&0x400)!=0x400)
   {
      inst=inst&0xff;
      if((inst>>6)==1)                  /* Si la instruccion tiene un operando */
         obtoper(0);
      else
         if((inst>>6)==2)               /* Si tiene dos operandos */
         {
            obtoper(0);
            obtoper(1);
         }
      ejecuta();
      if((psw&0x200)==0x200)            /* Si hay modo de rastreo */
         detener();
   }
}

/* Rutina que toma un byte de la memoria */

fetch(x)
long x;
{
   long w,y;
   if((psw&0x800)==0x800)               /* Para el caso de relocalizacion */
   {
      y=stmpl(x)>>1130x1ffe);
      y=y/2;
      w=memEy]&0xff00;
      w=(w<<8)|(y&0xff);
      if(((memEy]&6)==4) || ((memEy]&6)==6)) && ((psw&0x400)==0))
         pagefpag(0);                   /* Se puede acceder la pagina? */
      if((memEy]&1)==0)                 /* Esta la pagina en memoria? */
         pagefpag(1);
      return(memEw]);
   }
   else                                 /* Si no hay relocalizacion */
   {
      y=x/2;
      if(x%2==0)
         return((memEy]>>8)&0xff);
      else
         return(memEy]&0xff);
   }
}

/* Rutina que lee una palabra de memoria (3 bytes) */

long lecpal(x)
```

```
long x;
{
    long a,b,c,d;
    d=x;
    a=fetch(d++);
    b=fetch(d++);
    c=fetch(d);
    d=(a<<16)|(b<<8)|c;
    return(d);
}


/* Rutina que escribe un byte en memoria */

esc(x,y)
long x,y;
{
    long b,c;
    if((pcw&0x0800)==0x0800)                /* Si hay relocalizacion */
    {
        b=getmp((y)>>11&0x1ffe);
        b=b/2;
        c=mem[b]&0xfff0;
        c=(c<<8)|(y&0xfff);
        if( ((mem[b]&6)==6) ||              /* Es posible escribir en la pagina? */
            ((((mem[b]&6)==2) || ((mem[b]&6)==4)) && (!(pcw&0x400)==0)) )
            mmucerr(0);
        if((mem[b]&1)==0)                   /* Esta la pagina en memoria? */
            mmucerr(1);
        mem[b]=mem[b]|8;                    /* Se modifica la pagina */
        b=c/2;
        if(c%2==0)
            mem[b]=(mem[b]&0xff)|(x<<8);
        else
            mem[b]=(mem[b]&0xff00)|(x&0xff);
    }
    else                                    /* Si no hay relocalizacion */
    {
        b=c/2;
        if(c%2==0)
            mem[b]=(mem[b]&0x00)|(x<<8);
        else
            mem[b]=(mem[b]&0xff00)|(x&0xff);
    }
}


/* Rutina que escribe una palabra (2 bytes) en memoria o en registro */

escpal(x,y)
long x;
char y;
{
    if(oper[y].tipo==2 || oper[y].tipo==1)          /* En memoria */
    {
        esc(x,(oper[y].lugar));
```

```
        esc(x,0x880+ff,oper[y].lug+');
        esc(x,0+ff,oper[y].lug);
    }
    else                                        /* En registro */
        if(oper[y].tip==3)
            reg[oper[y].lug]=x;
}


/*    Rutina que se encarga de los errores al intentar accesar paginas
      en memoria */

hsaccpag(x)
char x;
{
    if(x==0)
        printf("\n ERROR. No es posible accesar la pagina");
    else
        printf("\n ERROR. La pagina no esta en memoria");
}


/* Rutina que almacena en memoria el programa a ejecutarse */

cargamem(x)
long x;
{
    int c,y;
    long s,t,i,z;
    y=0;
    if(x%2==0)
        t=1;
    else
        t=3;
    s=x/2;
    for(i=t;(c=leeblox())!=EOF;i++)
    {
        x=i%4;
        --x;
        if(x==-1)
            x=0;
        y=(c<<(x*4))|y;
        if(i%4==0)
        {
            mem[s++]=y;
            y=0;
        }
    }
    mem[s]=y;
    if(i%2==0)
        printf("hubo error");
    fclose(fe);
}


/* Rutina que lee un digito hexadecimal del archivo (programa) */
```

```
leehex()
{
    int c;
    c=getc(fe);
    if(c==EOF)
        return(EOF);
    else
        if(c>='0' && c<='9')
            return(c-'0');
        else
            if(c>='a' && c<='z')
                return(c-'a'+10);
            else
                if(c>='A' && c<='Z')
                    return(c-'A'+10);
                else
                    error();
}

/* Rutina que obtiene un operando ya sea de memoria o de registro */

obtoper(x)
char x;
{
    int a;
    long w,y;
    a=fetch(++reg[5142]);
    if((a&0x4)<0)
        switch(a&0x3)
        {
            case 0:                 /* Modo inmediato */
                oper[x].tipo=reg[5143];
                oper[x].op=lvapal(reg[5142]);
                reg[5142]+=2;
                oper[x].tip=1;
                break;
            case 1:                 /* Modo relativo al PC */
                w=lvapal(reg[5142]);
                reg[5142]+=2;
                y=(w&0x8000)?(w|0xffff0000):w;
                w=reg[5142]+y;
                oper[x].tipo=4;
                oper[x].op=lvapal(w);
                reg[5143]=0;
                break;
            case 3:                 /* Modo de salto de registro */
                error();
                oper[x].tipo=reg[5143];
                oper[x].tip=0;
                break;
            case 2:                 /* Modo directo de registro */
                oper[x].op=lvapal(reg[5142]);
```

```
            reg[1+3]=3;
            oper[x].lugar=y;
            oper[x].op=leeopal(y);
            oper[x].tipo=0;
            break;
        case 4:                 /* Modo indirecto de registro */
            y=x+0xf;
            w=oper[y-1].op+reg[y];
            oper[x].tipo=2;
            oper[x].op=leeopal(w);
            break;
        case 5:                 /* Modo indirecto de memoria */
            y=leeopal(reg[1+3]);
            reg[1+3]+=3;
            w=leeopal(y);
            oper[x].lugar=w;
            oper[x].tipo=2;
            oper[x].op=leeopal(w);
            break;
        case 6:                 /* Modo con registro indice */
            y=x+0xf;
            w=leeopal(reg[1+3]);
            if((w&0x2361)==w)
              w=(w<0x7fff)?w+1:0;
            reg[1+3]=w;
            w=w+reg[y];
            oper[x].lugar=w;
            oper[x].tipo=2;
            oper[x].op=leeopal(w);
            break;
        case 7:                 /* Modo con autoincremento */
            y=x+0xf;
            oper[x].tipo=0;
            y=reg[y];
            reg[1+3]+=3;
            oper[x].op=leeopal(y);
            oper[x].lugar=y;
            break;
      }
    else                        /* Modo inmediato corto */
      {
        oper[x].tipo=1;
        oper[x].op=(y<0x7f);
        oper[x].lugar=reg[1+3]++;
      }
}

/* Rutina que lee de pantalla un numero hexadecimal de seis digitos */

long leelong()
{
    char c;
    long y;
    int i,j;
    y=0;
```

```
    gets(c);
    for(i=6,j=0;i>0;i--,j++)
    {
        if(c[j]>='0' && c[j]<='9')
            c[j]=c[j]-'0';
        else
            if(c[j]>='a' && c[j]<='z')
                c[j]=c[j]-'a'+10;
            else
                if(c[j]>='A' && c[j]<='Z')
                    c[j]=c[j]-'A'+10;
        y=y<<4|c[j];
    }
    return(y);
}

/* Rutina que da opciones cuando el procesador esta en modo de rastreo */

detener()
{
    long x,y,i;
    char *c;
    printf("\nSe abortara\nn no rastreara\n registros\nm memoria\nc continua");
    printf("\n CUAL ES SU OPCION? ");
    gets(c);
    while(c[0]!='c')
    {
        switch(c[0])
        {
            case 'a':               /* Abortar */
                printf("\n EJECUCION ABORTADA \n");
                exit();
                break;
            case 'n':               /* Cambiar a modo de no rastreo */
                psw=psw&0xdff;
                break;
            case 'r':               /* Desplegar registros */
                for(i=0;i<14;i++)
                    printf("\n reg[%d]=%lx",i,reg[i]);
                printf("\n PC=%lx \n PSW=%x \n ATMP=%lx",
                    reg[14],reg[15],psw,atmp);
                break;
            case 'm':               /* Desplegar region de la memoria */
                printf("\n CUAL ES EL LIMITE INFERIOR? (6 digitos hex.) ");
                x=hex2long();
                printf("\n CUAL ES EL LIMITE SUPERIOR? (6 digitos hex.) ");
                y=hex2long();
                for(i=x;i<=y;i++)
                    printf("\n memoria[%6x]=%x",i,getch(i));
                break;
        }
        printf("\nSe abortara\nn no rastreara\n registros\nm memoria\nc continua");
        printf("\n CUAL ES SU OPCION? ");
        gets(c);
    }
```

```
}

/* Rutina que prende o apaga la bandera de acarreo */

acarreo(x,y)
long x;
char y;
{
    if(y==0)
        if((x>>23&1)==1)
            psw=psw|8;
        else
            psw=psw&0xff7;
}

/* Rutina que prende o apaga la bandera de overflow */

overflow(x,y)
long x;
char y;
{
    if(y==0)
        if(x>>23==0)
            psw=psw&0xffb;
        else
            psw=psw|4;
    else
        if(x>>0==0)
            psw=psw&0xffb;
        else
            psw=psw|4;
}

/* Rutina que prende o apaga la bandera de signo */

signo(x)
long x;
{
    if((x>>2331)==0)
        psw=psw&0xffd;
    else
        psw=psw|2;
}

/* Rutina que prende o apaga la bandera de cero */

cero(x)
long x;
{
    if((x&0xffffff)==0)
        psw=psw|1;
    else
```

```
        psw=psw&0x7fe;
}

/* Rutina utilizada por las instrucciones INC y DEC */

util(x)
char x;
{
    long a;
    if(oper[0].op>>23==1)
        oper[0].op=(oper[0].op&0x7fffff)*-1;
    if(x==0)                                        /* INC X */
        a=oper[0].op+1;
    else                                            /* DEC X */
        a=oper[0].op-1;
    if(a<0)
    {
        a=a*-1;
        overflow(a,0);
        a=a|0x800000;
    }
    else
    {
        overflow(a,0);
        a=a&0x7fffff;
    }
    cero(a);
    signo(a);
    escpal(a&0xffffff,0);
}

/* Rutina que es utilizada para la simulacion de las instrucciones
   SUM, RES, SUMAC, RESAC, MUL, DIV y COMP */

util(x)
char x;
{
    long a;
    if(oper[0].op>>23==1)
        oper[0].op=(oper[0].op&0x7fffff)*-1;
    if(oper[1].op>>23==1)
        oper[1].op=(oper[1].op&0x7fffff)*-1;
    switch(x)
    {
        case 0: a=oper[0].op+oper[1].op;            /* SUM X,Y */
                break;
        case 1: a=oper[0].op+(oper[1].op|(psw&0x1)<<23);  /* SUMAC Y,Y */
                break;
        case 2: a=oper[1].op-oper[0].op;            /* RES X,Y */
                break;
        case 3: a=oper[1].op-oper[0].op-(psw>>23&1); /* RESAC Y,Y */
                break;
        case 4: a=oper[0].op*oper[1].op;            /* MUL X,Y */
                break;
```

```
        case 5: if(oper[0].op!=0)                                   /* DIV X,Y */
                    a=oper[1].op/oper[0].op;
                break;
        case 6: a=oper[0].op-oper[1].op;                            /* COMP X,Y */
                break;
    }
    if(a<0)
    {
        a=a*-1;
        if((x>=0 && x<=4) || x==6)
            overflow(a,0);
        if((x>=0 && x<=3) || x==6)
            acarreo(a,0);
        a|=0x800000;
    }
    else
    {
        if((x>=0 && x<=4) || x==6)
            overflow(a,0);
        if((x>=0 && x<=3) || x==6)
            acarreo(a,0);
        a=a&0x7fffff;
    }
    cero(a);
    signo(a);
    if(x!=6)
        a=cpal(a&0xffffff,1);
}

/* Rutina utilizada para la simulacion de las instrucciones
   de aritmetica de punto flotante */

util3(x)
char x;
{
    long a,b,c,m1,m2,s1,s2;
    if(x==1)
        if(oper[0].op>>23==1)
            oper[0].op=oper[0].op&0x7fffff;
        else
            oper[0].op=oper[0].op|0x800000;
    if((oper[0].op>>22&1)==1)
        s1=(oper[0].op>>15&0x7f)*-1;
    else
        s1=oper[0].op>>15&0x7f;
    if((oper[1].op>>22&1)==1)
        s2=(oper[1].op>>15&0x7f)*-1;
    else
        s2=oper[1].op>>15&0x7f;
    if((oper[0].op>>22&1)==1)
        m1=(oper[0].op&0x7fff)*-1;
    else
        m1=oper[0].op&0x7fff;
    if((oper[1].op>>22&1)==1)
        m2=(oper[1].op&0x7fff)*-1;
```

```
    else
        m2=oper[i].op&0x7fff;
    if((m==0 || m==1) && m2!=1)
    {
        m=m2;
        m2=m1;
        m1=m;
        b=m2;
        m2=m1;
        m1=b;
    }
    switch(m)
    {
        case 0:                                 /* SUMF X,Y */
        case 1:                                 /* MESF X,Y */
            m2=m2%(m1-m2);
            m=m1;
            b=m1-m2;
            break;
        case 2:                                 /* MULF X,Y */
            m=m1*m2;
            b=m1*m2;
            break;
        case 3:                                 /* DIVF X,Y */
            m=m2 m1;
            if(m1!=0)
               b=m2/m1;
            break;
    }
    m=0;
    if(b<0)
    {
        b=-b;
        while((b&0x8000)!=0 && m<16)
        {
            b=b<<1;
            m--;
            m++;
        }
        b=b>>1;
    }
    else
    {
        while((b&0x4000)==0 && m<16)
        {
            b=b<<1;
            m++;
        }
    }
    if(m<0)
    {
        m=-m;
        b=(b+(1<<(m-1)))>>m;
        m=0;
    }
    else
        if(m>15)
            b=0x7fff;
```

```
    if(b<0)
    {
       b=b*-1;
       c=c|(b&0x7fff);
       c=c|0x800000;
    }
    else
       c=c|(b&0x7fff);
    if(a>>7==0 || b>>15==0)
       psw&=0xffb;
    else
       psw|=4;
    if((b&0x7fff)==0)
       psw|=1;
    else
       psw&=0xffe;
    if((c>>23&1)==1)
       psw&=0xffd;
    else
       psw|=2;
    escpal(c,i);
}

/* Rutina que es llamada cuando se intenta la ejecucion de una
   instruccion no implementada */

instil()
{
    printf("\nERROR. Instruccion ilegal\n");
    exit();
}
```

```
extern int inst;
extern int psw;
extern  long stmp,reg[16];
extern struct ope { long op,lug;
                    char tip; } oper[3];

ejecuta()
{
   char *ch;
   long a,b,c;
   long i;


   if((psw&0x300)==0x300)                          /* Si hay rastreo */
      printf("\nInstruction #2x%n%n",inst);


   if((inst>=0x00 && inst<=0x00)
      switch(inst)
      {

         case 0x00:                                /* NOP */
            break;
         case 0x01:                                /* RET */
            reg[14]=tempal(reg[15]);
            reg[15]+=3;
            break;
         case 0x02:                                /* PUSHR */
            for(i=0;i!=14;i++)
            {
               esc(reg[15]-3,reg[i]);
               esc(reg[15]-2,0);
               esc(reg[15]-1,0);
            }
            a=reg;
            esc(a,reg[15]);
            esc(a+2,reg[15]+2);
            esc(0,reg[15]+4);
            break;
         case 0x03:                                /* POPR */
            psw|=esc(reg[15],&i,2);
            reg[15]+=2;
            for(i=13;i>=0;i--)
            {
               reg[i]=lesc(reg[15]);
               reg[15]+=4;
            }
            break;
         case 0x04:                                /* PUSA */
            break;
         case 0x05:                                /* CHME */
            if((psw&0x400)!= 0x400)
            {
               if((psw&0x800)==0x800)
                  psw=psw|0x700;
               else
```

```
                psw=psw&0x800;
            }
            break;
        case 0x06:                                  /* CMPR */
            if((psw&0x400)==0x400)
            {
                if((psw&0x400)==0x400)
                    psw=psw&0x3ff;
                else
                    psw=psw|0x400;
                a=leapal(01);
                oper[0].lng=01;
                oper[0].tip=2;
                escpal(reg[15],0);
                reg[15]=a;
            }
            break;
        case 0x07:                                  /* CMRA */
            if((psw&0x400)==0x400)
            {
                if((psw&0x200)==0x200)
                    psw=psw&0xdff;
                else
                    psw=psw|0x200;
            }
            break;
        case 0x08:                                  /* ESCRE */
            if((psw&0x400)==0x400)
            {
                for(i=0;i<14;i++)
                    printf("\n reg[%d]=%lx",i,reg[i]);
                printf("\n PC=%lx \n SP=%lx \n PSW=%x \n ATMP=%lx",
                        reg[14],reg[15],psw,atmp);
            }
            break;
    }


if(inst>=0x00 && inst<=0x3f)
    instil();


if(inst>=0x40 && inst<=0x4e)
    switch(inst)
    {

        case 0x40:                                  /* INC X */
            util1(0);
            break;
        case 0x41:                                  /* DEC X */
            util1(1);
            break;
        case 0x42:                                  /* NEG X */
            if(oper[0].op>>23==1)
                a=oper[0].op|0x7fffff;
            else
```

```
            a=oper[0].op|0x800000;
        cero(a);
        signo(a);
        escpal(a,0);
        break;
    case 0x43:                                  /* ABS X */
        a=oper[0].op&0x7fffff;
        cero(a);
        signo(a);
        escpal(a,0);
        break;
    case 0x44:                                  /* INV X */
        a=(~oper[0].op)&0xffffff;
        cero(a);
        signo(a);
        escpal(a,0);
        break;
    case 0x45:                                  /* MATMF X */
        atmp=oper[0].op;
        break;
    case 0x46:                                  /* MOVC X */
        escpal(01,0);
        break;
    case 0x47:                                  /* MOVU X */
        escpal(0xffffff,0);
        break;
    case 0x48:                                  /* FBA X */
        switch(oper[0].op)
        {
            case 0: psw=psw|8;
                    break;
            case 1: psw|=4;
                    break;
            case 2: psw|=2;
                    break;
            case 3: psw|=1;
                    break;
        }
        break;
    case 0x49:                                  /* ABA X */
        switch(oper[0].op)
        {
            case 0: psw&=0xff7;
                    break;
            case 1: psw&=0xffb;
                    break;
            case 2: psw&=0xffd;
                    break;
            case 3: psw&=0xffe;
                    break;
        }
        break;
    case 0x4a:                                  /* PUSH X */
        esc(oper[0].op&0xff,--reg[15]);
        esc(oper[0].op>>8&0xff,--reg[15]);
        esc(oper[0].op>>16&0xff,--reg[15]);
```

```
            break;
        case 0x4b:                                  /* POP X */
            escpal(leepal(reg[15]),0);
            reg[15]+=3;
            break;
        case 0x4c:                                  /* LEE X */
            if((psw&0x400)==0x400)
            {
                gets(ch);
                a=ch[0]&0xff;
                esc(a,oper[0].lug);
            }
            break;
        case 0x4d:                                  /* ESC X */
            if((psw&0x400)==0x400)
                printf("%c",fetch(oper[0].lug));
            break;
        case 0x4e:                                  /* DESLI X */
            reg[15]=reg[oper[0].op];
            reg[oper[0].op]=leepal(reg[15]);
            reg[15]+=3;
            break;
    }


if(inst>=0x4f && inst<=0x5f)
    instil();


if(inst>=0x60 && inst<=0x6d)
    switch(inst)
    {

        case 0x60:                                  /* BMA X*/
            if(((psw>>1&1) || (psw&1))==0)
                reg[14]=oper[0].op;
            break;
        case 0x61:                                  /* BME X */
            if((psw>>1&1)==1)
                reg[14]=oper[0].op;
            break;
        case 0x62:                                  /* BIGU X */
            if((psw&1)==1)
                reg[14]=oper[0].op;
            break;
        case 0x63:                                  /* BMAI X */
            if((psw>>1&1)==0)
                reg[14]=oper[0].op;
            break;
        case 0x64:                                  /* BMEI X */
            if(((psw>>1&1) || (psw&1))==1)
                reg[14]=oper[0].op;
            break;
        case 0x65:                                  /* BAC X */
            if(((psw>>2&1) || (psw&1))==0)
                reg[14]=oper[0].op;
```

```
            break;
         case 0x66:                                    /* BAB X */
            if((psw>>2&1)==1)
               reg[14]=oper[0].op;
            break;
         case 0x67:                                    /* BDIF X */
            if((psw&1)==0)
               reg[14]=oper[0].op;
            break;
         case 0x68:                                    /* BARI X */
            if((psw>>2&1)==0)
               reg[14]=oper[0].op;
            break;
         case 0x69:                                    /* BABI X */
            if(((psw>>2&1) || (psw&1))==1)
               reg[14]=oper[0].op;
            break;
         case 0x6a:                                    /* BAC X */
            if((psw>>3&1)==1)
               reg[14]=oper[0].op;
            break;
         case 0x6b:                                    /* BNAC X */
            if((psw>>3&1)==0)
               reg[14]=oper[0].op;
            break;
         case 0x6c:                                    /* BRI X */
            reg[14]=oper[0].op;
            break;
         case 0x6d:                                    /* BSUB X */
            esc(reg[14]&0xff,--reg[15]);
            esc(reg[14]>>8&0xff,--reg[15]);
            esc(reg[14]>>16&0xff,--reg[15]);
            reg[14]=oper[0].op;
            break;

      }


   if(inst>=0x6e && inst<=0x7f)
      instil();


   if(inst>=0x80 && inst<=0x8f)
      switch(inst)
      {

         case 0x80:                                    /* SUM X,Y */
            util2(0);
            break;
         case 0x81:                                    /* SUMAC X,Y */
            util2(1);
            break;
         case 0x82:                                    /* RES X,Y */
            util2(2);
            break;
         case 0x83:                                    /* RESAC X,Y */
```

```
            util2(3);
            break;
        case 0x84:                              /* MUL X,Y */
            util2(4);
            break;
        case 0x85:                              /* DIV X,Y */
            util2(5);
            break;
        case 0x86:                              /* SUMF X,Y */
            util3(0);
            break;
        case 0x87:                              /* RESF X,Y */
            util3(1);
            break;
        case 0x88:                              /* MULF X,Y */
            util3(2);
            break;
        case 0x89:                              /* DIVF X,Y */
            util3(3);
            break;
        case 0x8a:                              /* COMST X,Y */
            a=fetch(oper[0].op++);
            b=fetch(oper[1].op++);
            cero(a-b);
            overflow(a-b,1);
            i=reg[0];
            while(a==b && i>0)
            {
                i--;
                if(i!=0)
                {
                    a=fetch(oper[0].op++);
                    b=fetch(oper[1].op++);
                }
                cero(a-b);
                overflow(a-b,1);
            }
            reg[0]=reg[0]-i;
            break;
        case 0x8b:                              /* ANAST X,Y */
            a=fetch(oper[0].op++);
            b=fetch(oper[1].op);
            overflow(a-b,1);
            cero(a-b);
            i=reg[0];
            while(a!=b && i>0)
            {
                i--;
                if(i!=0)
                    a=fetch(oper[0].op++);
                cero(a-b);
                overflow(a-b,1);
            }
            reg[0]=reg[0]-i;
            break;
        case 0x8c:                              /* MOD X,Y */
```

```
            a=oper[0].op^oper[1].op;
            escpal(a,1);
            break;
         case 0x8d:                                    /* Y X,Y */
            a=oper[0].op&oper[1].op;
            signo(a);
            cero(a);
            escpal(a,1);
            break;
         case 0x8e:                                    /* O X,Y */
            a=oper[0].op|oper[1].op;
            signo(a);
            cero(a);
            escpal(a,1);
            break;
         case 0x8f:                                    /* OEX X,Y */
            a=oper[0].op^oper[1].op;
            signo(a);
            cero(a);
            escpal(a,1);
            break;
      }

   if(inst>=0x90 && inst<=0x97)
      switch(inst)
      {

         case 0x90:                                    /* CODER X,Y */
            a=oper[0].op<<1;
            a=a>>oper[1].op;
            if((a&1)==1)
               psw=psw|8;
            else
               psw=psw&0xff7;
            a=a>>1;
            signo(a);
            cero(a);
            escpal(a,0);
            break;
         case 0x91:                                    /* COIZQ X,Y */
            a=oper[0].op<<oper[1].op;
            if((a>>24&1)==1)
               psw=psw|8;
            else
               psw=psw&0xff7;
            a=a&0xffffff;
            signo(a);
            cero(a);
            escpal(a,0);
            break;
         case 0x92:                                    /* CODEA X,Y */
            a=(oper[0].op&0x7fffff)<<1;
            a=a>>oper[1].op;
            if((a&1)==1)
               psw=psw|8;
```

```
            else
                psw=psw&0xff7;
            a=a>>1;
            if((oper[0].op>>23&1)==1)
                a=a|0x800000;
            signo(a);
            cero(a);
            escpal(a,0);
            break;
        case 0x93:                                  /* COMP X,Y */
            util2(6);
            break;
        case 0x94:                                  /* RODER X,Y */
            for(i=0;i<oper[1].op;i++)
            {
                a=oper[0].op&1;
                b=oper[0].op>>1;
                oper[0].op=a<<23|b;
                if(a==1)
                    psw|=8;
                else
                    psw&=0xff7;
            }
            signo(oper[0].op);
            cero(oper[0].op);
            escpal(oper[0].op,0);
            break;
        case 0x95:                                  /* ROIZQ X,Y */
            for(i=0;i<oper[1].op;i++)
            {
                a=oper[0].op<<1;
                b=a>>24&1;
                oper[0].op=(a|b)&0xffffff;
                if(b==1)
                    psw|=8;
                else
                    psw&=0xff7;
            }
            signo(oper[0].op);
            cero(oper[0].op);
            escpal(oper[0].op,0);
            break;
        case 0x96:                                  /* RODEA X,Y */
            for(i=0;i<oper[1].op;i++)
            {
                c=psw>>3&1;
                a=(c<<24)|oper[0].op;
                b=a&1;
                oper[0].op=a>>1;
                if(b==1)
                    psw|=8;
                else
                    psw&=0xff7;
            }
            signo(oper[0].op);
            cero(oper[0].op);
```

```
            escpal(oper[0].op,0);
            break;
         case 0x97:                                    /* ROIZA X,Y */
            for(i=0;i<oper[1].op;i++)
            {
               a=oper[0].op<<1;
               a=a|(psw>>3&1);
               b=a>>24&1;
               oper[0].op=a&0xffffff;
               if(b==1)
                  psw|=8;
               else
                  psw&=0xff7;
            }
            signo(oper[0].op);
            cero(oper[0].op);
            escpal(oper[0].op,0);
            break;
      }


   if(inst>=0x98 && inst<=0x9f)
      instil();


   if(inst>=0xa0 && inst<=0xad)
      switch(inst)
      {

         case 0xa0:                                    /* BMAC X,Y */
            if((oper[0].op&0x800000)==0 && (oper[0].op&0x7fffff)!=0)
               reg[14]=oper[1].op;
            break;
         case 0xa1:                                    /* BMEC X,Y */
            if((oper[0].op&0x800000)==0x800000 && (oper[0].op&0x7fffff)!=0)
               reg[14]=oper[1].op;
            break;
         case 0xa2:                                    /* BIC X,Y */
            if((oper[0].op&0x7fffff)==0)
               reg[14]=oper[1].op;
            break;
         case 0xa3:                                    /* INTER X,Y */
            escpal(oper[0].op,1);
            escpal(oper[1].op,0);
            break;
         case 0xa4:                                    /* MOV X,Y */
            escpal(oper[0].op,1);
            break;
         case 0xa5:                                    /* MOVBL X,Y */
            a=oper[0].op+reg[0];
            b=oper[1].op;
            for(i=oper[0].op;i<a;i++,b++)
            {
               c=fetch(i);
               esc(c,b);
            }
```

```
            break;
         case 0xa6:                                    /* ITERA X,Y */
            reg[oper[1].op]--;
            if(reg[oper[1].op]>0)
               reg[14]=oper[0].op;
            break;
         case 0xa7:                                    /* LIGA X,Y */
            a=reg[oper[0].op];
            esc(a&0xff,--reg[15]);
            esc(a>>8&0xff,--reg[15]);
            esc(a>>16&0xff,--reg[15]);
            reg[oper[0].op]=reg[15];
            reg[15]=reg[15]-oper[1].op;
            break;
         case 0xa8:                                    /* PBIT X,Y */
            a=1;
            for(i=0;i<(24-oper[1].op-1);i++)
               a=a*2;
            escpal(oper[0].op|a,0);
            break;
         case 0xa9:                                    /* ABIT X,Y */
            a=1;
            for(i=0;i<(24-oper[1].op-1);i++)
               a=a*2;
            a=(~a)&0xffffff;
            escpal(oper[0].op&a,0);
            break;
         case 0xaa:                                    /* CBIT X,Y */
            a=1;
            for(i=0;i<(24-oper[1].op-1);i++)
               a=a*2;
            if((oper[0].op&a)==a)
               b=oper[0].op&((~a)&0xffffff);
            else
               b=oper[0].op|a;
            escpal(b,0);
            break;
         case 0xab:                                    /* BBIT X,Y */
            a=24-oper[1].op-1;
            a=oper[0].op>>a&1;
            escpal(a,0);
            break;
         case 0xac:                                    /* LEEBL X,Y */
            if((psw&0x400)==0x400)
            {
               gets(ch);
               b=oper[0].op;
               for(i=0;i<oper[1].op;i++,b++)
               {
                  a=ch[i];
                  esc(a,b);
               }
            }
            break;
         case 0xad:                                    /* ESCBL X,Y */
            if((psw&0x400)==0x400)
```

```
            {
                for(i=0;i<oper[1].op;i++,oper[0].op++)
                    printf("%c",fetch(oper[0].op));
            }
            break;
    }


if(inst>=0xae && inst<=0xff)
    instil();
}
```