UNIVERSIDAD LA SALLE

ESCUELA DE INGENIERIA

ANALISIS DE LA ESTRUCTURA INTERNA DEL SISTEMA OPERATIVO MVS: UNA SOLUCION AL PROBLEMA DEL APROVECHAMIENTO EFICIENTE DE SISTEMAS COMPUTACIONALES COMPLEJOS

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE

INGENIERO MECANICO ELECTRICISTA

AREA ELECTRONICA

P R E S E N T A

JOSE ANTONIO NUÑEZ POBLETE

DIRECTOR DE TESIS: GUILLERMO ARANDA PEREZ

MEXICO, D. F. 1992

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

Tema	Página

INTRODUCCION

Las empresas han venido experimentando a lo largo de los últimos años un cambio fundamental en la forma en que el cliente o consumidor de bienes y servicios percibe la ventaja competitiva de una empresa con respecto a otra. Este cambio fundamentalmente se refiere a un aumento considerable en las exigencias de los consumidores con respecto a la calidad de los bienes o servicios que las empresas ofrecen, y aún más en la oportunidad y responsividad que estas empresas tienen para ofrecer estos bienes o servicios. Esto quiere decir que el esquema tradicional de lograr competitividad por medio de bienes o servicios de costo razonable y de buena calidad ya no es suficiente: Ahora el consumidor quiere respuestas rápidas y oportunas a sus necesidades. En el ambiente comercial de la década de los 90, el mercado ya no lo controla el producto o el servicio brindado; ahora lo controla el consumidor.

Sin lugar a dudas, uno de los elementos más importantes que permite que una empresa sea más ágil para responder a las necesidades del mercado y mantenerse aventajada con respecto a la competencia es la capacidad de la empresa para manejar su información eficientemente.

La información es para las organizaciones actuales tan crítica como lo fueron la tierra, las máquinas y la mano de obra en la revolución industrial.

La eficiencia del manejo de información en las organizaciones depende de los sistemas de cómputo sobre los cuales esté basado. Obviamente, entre más compleja sea la organización, más complejo será el manejo de información y por lo tanto más exigirá al sistema de cómputo.

La administración de los sistemas de información comprende varias disciplinas. Estas disciplinas abarcan desde la operación, hasta la orientación estratégica a largo plazo de los equipos.

Cada una de estas disciplinas tiene una importancia fundamental; sin embargo, probablemente la faceta más crítica de la administración de los sistemas de cómputo es el manejo de problemas.

Los problemas pueden ser resultado de una mala planeación en la capacidad del sistema, o de una mala administración de cambios (que representan otras disciplinas de la administración de los sistemas de información); sin embargo siempre representan un impacto muy fuerte a la imagen de la organización a la vista de los usuarios finales de los servicios que proporciona el sistema de cómputo, usuarios que pueden ser internos a la organización o, aún más grave, sus clientes.

Es por esto que la resolución pronta de los problemas que se presenten en la instalación y que impidan el proporcionar un servicio puede tener repercusiones económicas muy importantes.

La presente tesis trata acerca de la estructura interna de un sistema operativo llamado "MVS", que por sus siglas significa "Memoria Virtual Múltiple" ("Multiple Virtual Storage"). El MVS fue diseñado para obtener el máximo provecho de los Sistemas/370 y Sistemas/390 de IBM, arquitecturas que por su capacidad, confiabilidad y relación costo/beneficio son de las más usadas por las empresas más grandes e importantes del mundo.

Para poder establecer con claridad el objetivo de esta tesis es necesario cubrir algunos conceptos relacionados al manejo de problemas en las instalaciones que cuentan con una plataforma basada en MVS (y en general, cualquier instalación basada en una "Mainframe", es decir un gran procesador central).

Para cumplir con la parte de manejo de problemas, cada organización cuenta con un "Staff" de soporte técnico, cuya función es generalmente la de diagnosticar y resolver los problemas que pudieran surgir en su instalación, instalar software nuevo, así como dar mantenimiento y hacer modificaciones al ya existente. Este "Staff" lo integran personas con un elevado nivel técnico que les permite hacerse cargo de la mayoría de los problemas que surgen en su instalación.

Sin embargo, en ocasiones se presentan problemas que debido a su complejidad técnica o debido a la falta de información más detallada en cuanto al producto, no pueden resolverse con recursos propios de la instalación. Para estos casos, existe en IBM una fuerte estructura de soporte destinada a auxiliar al cliente en la determinación, diagnóstico y corrección de problemas relacionados a todos los productos de Software que esta compañía ofrece. Esta estructura consta de varios niveles de soporte, niveles que van desde el representante de soporte, que es el punto de contacto único con el cliente, hasta los equipos de desarrollo del producto. En cada nivel de soporte existen diversas herramientas que facilitan de alguna forma el diagnóstico de algún posible defecto en el código del producto.

Las herramientas con las que cuenta el representante de soporte son por ejemplo: una base de datos internacional llamada "RETAIN", en donde se centralizan los reportes de problemas de productos de IBM que han ocurrido en todo el mundo, además de las correcciones a los defectos que ya se han detectado y diagnosticado, correcciones que se llaman "Arreglos Temporales a Programas" ("Program Temporary Fix", PTFs).

Esta base de datos permite identificar problemas ya reportados en otras instalaciones y acelerar el proceso de corrección.

Sin embargo, el número de problemas que se pueden diagnosticar directamente por medio de la base de datos (es decir, sin hacer un análisis más detallado) es pequeño en relación al número de problemas que se presentan en todas las instalaciones. Esto es debido a muchas causas: El diseño de los equipos es tan flexible y versátil que prácticamente ninguna instalación en el mundo entero es igual a otra, además las aplicaciones que se corren en las distintas instalaciones son prácticamente únicas, de tal forma que es difícil (aunque no imposible) que alguna combinación de elementos que de lugar a un problema se de en otra instalación.

En conclusión, se puede decir que en la gran mayoría de los problemas que se presentan en cualquier instalación y que llegan a manos del representante de soporte se tiene que hacer un análisis muy detallado de los síntomas y de las circunstancias en que se produjo este.

Este análisis requiere siempre de un conocimiento muy sólido acerca de la estructura interna del sistema operativo y del producto que se trate, así como una gran familiaridad con las diversas herramientas con las que se cuenta. Estas herramientas son: Manuales de usuario, manuales de "Areas de Datos" ("Data Areas") que es en donde se puede encontrar el formato de las estructuras lógicas que usa el producto (conocidas como "Bloques de control"), manuales de lógica del programa, los cuales contienen una descripción muy a detalles del flujo de operación del mismo y por último las microfichas que contienen el código fuente documentado (en lenguaje ensamblador) de los módulos que forman al producto.

Toda esta información en su conjunto es vastísima y muy completa. Como compañía, IBM siempre se ha caracterizado por producir la documentación más completa del mercado con respecto a sus productos de software.

Sin embargo, esta información tan completa y tan vasta no necesariamente es asimilable ni mucho menos sencilla; y esto es especialmente cierto en el caso del MVS.

El MVS es el sistema operativo "Punta de Lanza" que ofrece IBM. Este sistema operativo es el que más capacidad ofrece, y por lo tanto es el más complejo.

La información que existe con respecto al MVS se puede dividir en cuatro grandes grupos:

1) Manuales de información general, en los cuales se "Platica" de manera muy somera únicamente las funciones y/o el punto de vista del usuario final con respecto a alguno de los componentes del MVS.

2) Manuales referentes al uso del producto y operación del producto, entre los cuales se encuentran los manuales de mensajes o de instalación.

3) Manuales de diagnóstico que contienen información técnica y muy específica necesaria para diagnosticar determinados tipos de problemas; entre estos manuales se encuentran los manuales de lógica, que para el caso del MVS representan la documentación más detallada en cuanto al mismo. Estos manuales de lógica son una excelente referencia para conocer con lujo de detalle el flujo de cualquier parte del proceso del MVS.

4) Y por último las microfichas, que como se mencionó contienen el código fuente, documentado, de todo el producto MVS.

A pesar de esta información tan vasta, en el proceso de diagnóstico de un problema, el paso de un manual de información general (o de un manual de diagnóstico) a un manual de lógica o a una microficha es como verse envuelto de repente en un mar de acrónimos y términos inentendibles, diagramas complejísimos y flujos de lógica exóticos.

El entender un manual de lógica no es una tarea imposible; de hecho, los manuales de IBM resultan textos muy completos y muy bien estructurados; lo que los hace complejos es la naturaleza misma del MVS: compleja y vasta.

Existe, a juicio del autor del presente trabajo, una brecha de información entre los manuales a los que tiene acceso todo usuario (manuales de uso del producto, de diagnóstico y de información general) y los manuales de lógica o las microfichas del producto.

El presente trabajo surgió como idea cuando el autor empezó a trabajar como representante de servicio de Software para IBM. Este trabajo intenta de alguna forma cubrir esta "Brecha" que existe entre los manuales meramente descriptivos y los manuales altamente especializados. Es de alguna forma, el manual que el autor hubiese querido tener a la mano para entender una serie de conceptos muy complejos.

Los objetivos de esta tesis, tal y como se plantea su estructura son fundamentalmente dos: Proporcionar una herramienta de fácil utilización y consulta a todas aquellas

personas involucradas en la detección, diagnóstico y corrección de problemas relacionados con el código del MVS, esto incluye tanto a representantes de servicio como a "System Programmers" (las personas del "Staff" de soporte que se encargan de la solución de problemas) de instalaciones que corran bajo una plataforma MVS. Esto con la idea en mente de proporcionar una solución lo más rápida posible a una falla en el sistema, y sobre todo, en su componente más crítico: El sistema operativo.

Esta tesis representa un documento de consulta en donde el lector encontrará la descripción del funcionamiento de los componentes más importantes del MVS, en particular del llamado "Programa Básico de Control" ("Basic Control Program", BCP), es decir, de la parte medular del MVS. Desde luego que esta tesis no pretende cubrir absolutamente todos los componentes del MVS, ni tampoco ser exhaustiva al abordar cada uno de ellos. De hecho, se limita a aportar un enfoque no encontrado en otro tipo de documentación: Un enfoque que abarca una descripción funcional del componente en cuestión y una vista global de los bloques de control más importantes, su función y su interrelación.

El otro objetivo es proporcionar a cualquier lector con conocimientos básicos de informática un texto en el que esté explicado sencillamente que es un sistema operativo, para que sirve y además lo introduzca a la estructura interna del sistema operativo que probablemente sea el más complejo creado hasta nuestros días: El MVS.

Esta tesis puede resultar especialmente atractiva a cualquier persona que, por ejemplo, programe en algún lenguaje de alto nivel, pero tenga nulo o poco conocimiento acerca del sistema operativo sobre el cual programa, especialmente en ambientes de "Mainframes" IBM.

Para lograr ambos objetivos, el texto está estructuralmente dividido en cuatro capítulos hasta cierto punto independientes, pero ordenados en una secuencia lógica:

El capítulo I revisa algunos conceptos básicos acerca de proceso de datos, que son pre-requisitos para la comprensión de temas más avanzados. En este capítulo se tratan temas como: Qué es un sistema operativo, qué es ensamblador, etc. Para las personas que tengan un conocimiento sólido en estos temas puede servir de repaso, o puede simplemente omitirlo.

El capítulo II trata acerca de la arquitectura del sistema/370. Esta arquitectura es la que explota el sistema operativo que se estudia en esta tesis: El MVS. Este capítulo puede resultar interesante incluso a personas que lleven mucho tiempo programando en ambientes basados en sistema/370. Este capítulo revisa conceptos clave para comprender gran parte de la filosofía de diseño del MVS.

El capítulo III cubre a detalle algunas de las áreas funcionales del MVS/XA. Se hace énfasis especialmente en los bloques de control más importantes y en sus interrelaciones. Esto es debido a que en el proceso de diagnóstico de un problema, usualmente la información más valiosa es un "Vuelco" ("Dump") de la memoria del sistema en el momento de la falla. Este "Dump" contendrá entonces los bloques de control que guardan la información concerniente a lo que el sistema estaba haciendo y como lo estaba haciendo en el momento de la falla. El texto trata de mantener un estilo descriptivo, sin mencionar datos muy específicos, si no más bien el "Cómo" hace tal o cual cosa el sistema operativo. Esto, a juicio del autor, es lo que hace falta la mayoría de las veces en la documentación disponible; los datos específicos y técnicas de diagnóstico se pueden encontrar en los manuales de diagnóstico del producto.

En el capítulo IV se mencionan algunos avances en el desarrollo del MVS, en especial la introducción del sistema/390 con su contraparte en software, llamado el MVS/ESA. Se tratan de manera muy general las nuevas tendencias del MVS, tal como el manejo cada vez más acentuado de los datos en memoria para evitar las operaciones de Entrada y Salida (I/O) excesivas.

Ojalá el lector encuentre en esta tesis un material de referencia valioso, que facilite la solución de los problemas que se llegan a presentar en una instalación que cuente con un sistema operativo MVS; problemas que desgraciadamente, la mayoría de las veces se tornan críticos y en los que no siempre es posible dedicar horas de lectura para comprender el funcionamiento de cierto componente del sistema.

CAPITULO I.- CONCEPTOS PRELIMINARES

1.1 CONCEPTO DE SISTEMA OPERATIVO

Un sistema operativo es una colección organizada de programas y datos especificamente diseñados para manejar los recursos del computador, facilitar la creación de programas de computación y controlar la ejecución de los mismos; en otras palabras, explota a los recursos disponibles eficazmente para realizar alguna tarea productiva.

Considérese una computadora recién salida de la linea de ensamble. Este conjunto de elementos electrónicos no puede hacer absolutamente nada: no puede leer datos de un disco, no responde ordenes de un operador, no puede cargar ni mucho menos ejecutar un programa. Sin embargo, a casi todo el mundo le es familiar el hecho de que es relativamente fácil comunicarse con una computadora y hacer que esta realice algún trabajo productivo. Esto no es totalmente cierto, realmente el usuario no se está comunicándose con la computadora, sino que realmente está comunicándose con un programa o grupo de programas que de alguna manera aisla al usuario del equipo y le presenta una cara más sencilla, mucho más amistosa. Este grupo de programas es el SISTEMA OPERATIVO.

El proporcionar una interfaz con el equipo sencilla de usar es quizá la función más aparente del sistema operativo, sin embargo, la función de un sistema operativo va más allá de simplemente mostrar una faceta "amigable" al usuario. Un sistema de procesamiento de información, basado en la computadora representa una combinación de recursos de equipo ("Hardware"), de soporte lógico y datos ("Software"). Estos recursos son costosos, por lo que resulta esencial que estos recursos se utilicen tan eficientemente como sea posible. Un sistema operativo, especialmente si se trata de un sistema computacional grande, está proyectado para que sea un administrador de recursos, asignando o distribuyendo los recursos de hardware y software eficazmente.

De hecho, un sistema operativo bien proyectado no trata únicamente con el equipo o con el soporte lógico o precisamente con la administración de datos, sino con la optimización de la forma en la cual todos estos recursos funcionan juntos para lograr algún objetivo deseado.

1.2 RECURSOS DEL SISTEMA

Ya se mencionó que una tarea fundamental de los sistemas operativos es administrar eficientemente los recursos con los que cuenta el sistema computacional. De una forma o de otra, el sistema operativo afecta a todos los recursos involucrados en una instalación de computación, esto es, a los recursos de hardware y a los recursos de software.

1.2.1.- Recursos de hardware

Se refieren al sistema físico de computación. Estos recursos, de una forma muy general, se pueden dividir en:

-Unidad Central de Proceso ("Central Processing Unit", CPU) Es la parte del sistema en donde se realizan las operaciones de aritmética y lógica. Es la encargada de leer instrucciones guardadas en la memoria y ejecutarlas, cargando los datos guardados en memoria dentro de "Registros" y una vez ahí manipulándolos de acuerdo a las instrucciones que se ejecuten.

-Memoria principal

También llamada memoria real o núcleo, es la parte del sistema en donde se almacenan las instrucciones que le indican a la CPU qué hacer y además almacena los datos que va a manipular. Solo se pueden ejecutar instrucciones y manipular datos que se encuentren en memoria principal.

-Dispositivos de Entrada/Salida ("Input/Output", I/O)

Son el medio de comunicación al exterior del sistema. Ejemplos de estos dispositivos de I/O son por ejemplo una pantalla o un teclado, (la pantalla sirve de medio de comunicación del sistema hacia el usuario -"OUTPUT"- y el teclado como medio de comunicación del usuario hacia el sistema -"INPUT"-), una impresora, etc.

-Dispositivos de almacenamiento auxiliar

Es el dispositivo de almacenamiento masivo de información (discos, cintas, etc). Los datos contenidos en este almacenamiento auxiliar no son ni ejecutables (si estamos hablando de instrucciones) ni manipulables (si estamos hablando de datos) sino hasta que se transfieren de la memoria auxiliar (dispositivos de almacenamiento auxiliar) hacia la memoria principal.

Cada uno de estos elementos tiene un costo para la instalación distinto. Este costo incluye no solo lo que cuesta económicamente el equipo sino también lo que contribuye al rendimiento global de la instalación. Entre más alto sea este costo para cierto componente, más crítica será la adecuada utilización de este componente. El sistema operativo tiene que estar más al pendiente de la utilización de un componente con un alto costo que de un componente con bajo costo.

Generalmente la CPU y la memoria principal son los componentes con más alto costo, por lo cual el sistema operativo tratará de tener procesando el mayor tiempo posible a la CPU y ocupada la memoria principal lo más eficientemente posible. Debido a que su costo es muy alto, tener sin procesar a la CPU o desocupada a la memoria representa ya un gasto en sí, pues se espera que amorticen su costo realizando trabajo productivo.

1.2.2.- Recursos de software

Estos recursos incluyen a programas y datos. El sistema operativo debe garantizar que los datos y programas sean accesibles a todos los usuarios que requieren acceso a ellos, pero garantizar igualmente tanto la integridad de estos datos (que no los actualicen dos usuarios al mismo tiempo, por ejemplo) como la seguridad de los mismos (que no los accesen aquellos usuarios que no tienen necesidad de ello).

1.3.- PROGRAMAS Y LENGUAJES DE PROGRAMACION

Se mencionó anteriormente que el sistema operativo es una colección de programas que de alguna manera hacen eficiente la operación del computador. Ahora se hará una explicación breve de lo que es un programa, y que es lo que "ve" la computadora cuando ejecuta un programa.

Un programa lo podemos definir como una colección lógica de instrucciones de computadora que se usan para controlar el manejo que esta haga de datos específicos, para obtener un resultado concreto.

En la computadora, la información (lo cual incluye tanto al programa como a los datos) está representada en una forma entendible a la computadora: En forma de un patrón de Bits, que en conjunto, representan a esta información. Estos patrones de bits se representan como números hexadecimales (u octales para algunos sistemas) para manipular los bits en conjunto.

Las instrucciones de computadora (lenguaje máquina) son la serie de instrucciones que la máquina es capaz de entender y ejecutar. Cada instrucción está representada por un número de "n" dígitos hexadecimales, dependiendo de la máquina de la que estemos hablando. Estas instrucciones están diseñadas para ejecutar funciones muy básicas, como por ejemplo sumar dos números.

Resulta poco práctico codificar directamente programas con instrucciones de máquina: Son prácticamente ininteligibles al ojo del ser humano, son muy difíciles de escribir y aún más de depurar. Un programa para sumar dos números, escrito en lenguaje máquina del SISTEMA/370 de IBM se vería así:

```
00000: 05C05850 C01E5860 C0221A56 5050C026
00010: D205C030 C02AD203 C036C026 07FE0000
00020: 00000200 00000100 00000000 E3D6E3C1
00030: D37A4040 40404040 00000000 F1F2F3F4
00040: F5.........
```

Programa para sumer dos números en representación hexadecimal.

Figura 1.1

El número de más a la izquierda representa la localidad de memoria y los números de la derecha representan los datos en sí. Como se puede apreciar, resulta sumamente complejo entender una serie de números así, aún cuando estos números están ya representados en forma de números hexadecimales, más "entendibles" que representados en forma de bits. El mismo programa representado con bits se vería como en la figura 1.2.

```
00000000000000000000000000010111000000000000000000000010010
00000000000000010000110100100000010111000000000110000110...
```

Representación binaria de un programa

Figura 1.2

Debido a esto se han desarrollado formas más convenientes para el ser humano de codificar programas, esta formas son conocidas como "Lenguajes de Programación". Para convertir de estas formas entendibles al ser humano a una forma entendible a la máquina, existen programas "Procesadores de Lenguajes". Entre los programas procesadores de lenguajes tenemos a los "Compiladores" y "Ensambladores".

1.3.1.- Compiladores

Los "Compiladores" al igual que los "Generadores" o "Intérpretes" trabajan con programas escritos en "Lenguajes de alto nivel", los cuales son lenguajes mucho más entendibles al ser humano que el lenguaje máquina o el lenguaje ensamblador. Estos lenguajes permiten programar evitando todos los detalles necesarios para programar en lenguaje ensamblador; se dice que son "Orientados a proceso": El programador especifica una serie de enunciados que serán ejecutados para llevar a cabo alguna tarea en la

computadora. Estos enunciados son, en general, mucho más amplios que un enunciado de ensamblador y por lo general mas parecidos al inglés. Los lenguajes de alto nivel generalmente están orientados a una clase particular de aplicaciones, permitiendo así codificar los enunciados con expresiones usuales en ese campo particular de interés: Un lenguaje de alto nivel para aplicaciones científicas como FORTRAN permite codificar expresiones matemáticas prácticamente igual que como se escribirían en papel, en cambio, un lenguaje de programación orientado a negocios tendrá expresiones similares a las que se usan en ese medio para describir una operación matemática.

No importa en que lenguaje de programación se codifique un programa, finalmente siempre tiene que convertirse en instrucciones de máquina para que la computadora lo entienda y lo pueda ejecutar.

El proceso de convertir de un lenguaje orientado a proceso a instrucciones de máquina (lenguaje máquina) se llama "Compilación", y esto es llevado a cabo a través de un programa "Compilador". El listado original en el que están escritas las instrucciones en el lenguaje que se use se le conoce como "Código Fuente" y al resultado de su conversión a lenguaje de máquina se le conoce como "Código Objeto". Para cada enunciado escrito en lenguaje de alto nivel, se generan muchas instrucciones de lenguaje máquina

Los compiladores son necesariamente dependientes del lenguaje que esté siendo procesado, sin embargo, los etapas de las que se compone el proceso de compilar un programa siempre son las mismas:

1.- El programa fuente se lee y analiza enunciado por enunciado.
2.- Se realiza un análisis "Léxico" para identificar palabras reservadas, variables, símbolos de operador, constantes, etc.
3.- Se realiza un análisis de "Sintaxis" para identificar el tipo de enunciado y verificar que la estructura del mismo es admisible.
4.- Se construyen tablas de símbolos, expresiones, y enunciados para poder construir referencias a los mismos.
5.- Se realiza un análisis del flujo lógico del programa y un análisis global de errores.
6.- Se generan instrucciones de máquina y se optimiza el código máquina generado si es necesario.
7.- Se genera un "Módulo Objeto" y un listado del programa.

El compilador no tiene nada que ver con la ejecución del programa en sí. Una vez que generó el código objeto, no se necesita del programa compilador para que el programa compilado corra.

Algunos compiladores tienen la opción de generar un listado del código objeto en el lenguaje compilado con su traducción al lenguaje ensambaldor (que se verá mas adelante).

1.3.2.- Ensambladores

Los ensambladores trabajan con "Lenguaje Ensamblador". El lenguaje ensamblador es un lenguaje de menor nivel que los lenguajes compilados, es de hecho, una representación simbólica de las instrucciones de lenguaje máquina, en el cual se representa cada instrucción por un nemónico que haga recordar lo que hace una instrucción. Podemos decir que es lo mismo que el lenguaje máquina, representado de otra forma y con algunas ventajas. Por ejemplo, la instrucción "1A" suma el contenido de dos registros (por ejemplo el registro 5 más el registro 6, en cuyo caso se codificaría como "1A56"), en lenguaje ensamblador se codifica como:

AR 5,6

El nemónico AR ("Add Register", suma registros) es más fácil de recordar que el número "1A". Al número (como "1A") que representa alguna instrucción se le conoce como "Código de operación" ("Opcode") y a los datos que lleva después (sobre los que opera) se le conocen como "Operandos". Además, el lenguaje ensamblador permite hacer referencias a ciertas áreas de memoria en donde se encuentran almacenados datos por medio de símbolos, es decir, en lugar de hacer referencia a la localidad de memoria 00028 se hace referencia , por ejemplo, al símbolo "SUM". El mismo programa de la figura 1.1, codificado en lenguaje ensamblador se vería como en la figura 1.3.

Como se ve en la figura 1.3, cada instrucción de lenguaje ensamblador genera solo una instrucción de máquina (excepto las "Macros" que se verán mas adelante) y hay instrucciones de lenguaje ensamblador que no corresponden a ninguna instrucción de máquina (p.e. la instrucción "USING" del ejemplo anterior). El programa "Ensamblador" no es mas que una especie de "Traductor" de nemónicos a instrucciones de máquina.

Hay instrucciones de ensamblador que no tienen una contraparte en instrucciones de máquina (en otras palabras, no se "traducen") y sirven para indicarle ciertas cosas al ensamblador (por ejemplo, en la figura 1.3 la instrucción USING le indica al ensamblador qué registro será usado como base).

El proceso mediante el cual se "traducen" las instrucciones de ensamblador (nemónicos) a instrucciones de máquina ("Opcodes") se conoce como "Ensamble" y a la acción en sí se le conoce como "Ensamblar" un programa.

```
LENGUAJE MÁQUINA          LENGUAJE ENSAMBLADOR

00000: 05C0                1 BEGIN    BALR   12,0
                           2          USING  *,12
00002: 5850 C01E           3          L      5,CONSTANT
00006: 5860 C022           4          L      6,CONSTANT+4
0000A: 1A56                5          AR     5,6
0000C: 5050 C026           6          ST     5,SUM
00010: D205 C030 C02A      7 STARTB   MVC WORKA,MSG
00016: D203 C036 C026      8          MVC WORKA+6(4),SUM
0001C: 07FE                9          BR     14
0001E: 0000
00020: 00000200           10 CONSTANT DC     F'512'
00024: 00000100           11          DC     F'256'
00028:                    12 SUM      DS     F
0002C: E3D6E3C1D37A       13 MSG      DC     C'TOTAL'
00032: 404040404040       14 WORKA    DC     CL6' '
00038:                    15          DS     CL4
0003C: F1F2F3F4F5         16 ZONEDEC  DC     C'12345'
```

Figura 1.3

Programa en lenguaje ensamblador para sumar dos números.

Las "Macro instrucciones" o "Macros" son secuencias bien definidas de instrucciones de lenguaje ensamblador a las cuales se les da un nombre. Por ejemplo, si se usa la misma secuencia de instrucciones para mandar un mensaje a la consola del operador una y otra vez a lo largo de un programa, es mucho mas cómodo para el programador "Llamar" o hacer referencia a una macro que escribir una y otra vez las mismas líneas de código.

Sin embargo, como el código del cual se compone una macro no se codifica directamente sobre el programa fuente, este código debe residir en algún sitio. Generalmente se cuenta con una "Biblioteca de Macros" (una biblioteca es un archivo que reside en un dispositivo de almacenamiento auxiliar, generalmente discos, que contiene código objeto de programas de usuario o del sistema), de tal forma que se puedan recuperar las instrucciones que componen una macro.

Las macros, además sirven como una interfaz entre el programa y el sistema operativo. Cuando un programa desea hacer uso de un servicio que proporciona el sistema operativo, por ejemplo, leer un registro de un archivo se utiliza la macro "READ" . Esta macro, que reside en bibliotecas de macros del sistema, tiene el código necesario para establecer las condiciones necesarias para que una rutina del sistema se encarge de leer un registro de un archivo, además que se entiende que este código está lo suficientemente probado para usarse con confianza.

Cualquier programador podría codificar su propia rutina de lectura de datos o de cualquier otro servicio del sistema operativo, pero esto sería altamente ineficiente en tiempo y recursos, además de que solo después de un uso repetido de la rutina se podría garantizar su confiabilidad. El sistema operativo existe precisamente (como ya se comentó antes) para proveer de este tipo de servicios.

Cuando se hace una llamada a una macro, usualmente se le tienen que dar como entrada una serie de "Parámetros" que son la indicación de lo que tiene que hacer exactamente .Para el ejemplo anterior, a la macro "READ" se le tendría que indicar en que archivo y en que volumen reside el registro que se tiene que leer, además de la longitud del registro a leer y la dirección del inicio del área en memoria donde se va a dejar el registro leído.

1.3.4.- "Linkedición"

Los programas en lenguaje ensamblador se escriben, generalmente, en forma modular. Se evita codificar programas demasiado largos, que serían difíciles de depurar y mantener, en su lugar, se codifican "Módulos" pequeños y con funciones bien definidas. Dentro de cada módulo a su vez se separa lógicamente el flujo del programa en secciones llamadas "Secciones de Control" ("Control Sections" o CSECTS). Estas CSECTS son como pequeños módulos dentro de módulos, también cada una con una función muy particular. Cada módulo, por convención, tiene un nombre único y dentro de cada módulo existen una o mas CSECTS con un nombre único. Un programa complejo, entonces se compone de uno o más módulos interactuando unos con otros. Dentro de un módulo se le puede pasar control a otro módulo o se pueden referenciar valores que residen en otro módulo (lo que se conoce como "Referencias Externas"). Generalmente el programa se compone de un módulo principal (programa principal) que sirve de manejador de una serie de módulos (subprogramas).

Con esta perspectiva en mente, se puede observar que no es sencillo construir un programa completo, que este autocontenido en su totalidad, partiendo de una serie de módulos interconectados entre sí, llamadas a macros, etc. El proceso se puede dividir en dos etapas: Ensamble y "Linkedición".

a) Ensamble

El primer paso es el ensamble de cada módulo en particular que componga al programa. El ensamble se efectúa en dos etapas: En la primera, se le asigna una dirección a cada símbolo y se identifican las macros (localizándolas dentro de alguna biblioteca de macros o dentro del listado fuente mismo) y son "Expandidas", es decir, se generan explícitamente cada una de las instrucciones de las que se

compone la macro. En la segunda etapa, los códigos de operación simbólicos y los operandos son reemplazados por códigos de instrucción de máquina y direcciones, respectivamente. Además el módulo objeto y un listado del programa (con el código fuente en lenguaje ensamblador y el código objeto en instrucciones de máquina).

Para poder combinar una serie de módulos en uno solo se requiere que los módulos sean "Relocalizables". Generalmente los módulos se codifican independientemente unos de otros y cada módulo asume por simplicidad que su dirección inicial (es decir, la dirección en donde está su primera instrucción) es la dirección cero.

Sin embargo, cuando los módulos se combinan y se colocan unos tras otros, la dirección inicial de los módulos no va a ser la dirección cero, salvo tal vez, la dirección del módulo que aparezca antes que todos. Si las referencias a direcciones de memoria dentro de los módulos se hicieran de manera absoluta, cuando se cambiara el origen de las direcciones, esta referencia perdería su significado, pues apuntaría a una dirección en la que seguramente ya habrá otra cosa.

Para evitar esto y permitir que los módulos cambien su localización dentro de memoria sin perder sus apuntadores (es decir, sus referencias) se hace que las direcciones a las que se hace referencia se compongan de una BASE que adquiere el valor de la dirección origen del módulo al momento de la ejecución, sea cual sea esta dirección, y un "Desplazamiento".

Por ejemplo, en la figura 1.4(A) se muestra un programa que hace referencia a dos valores que residen en las localidades de memoria x'1100' y x'1200'. El programa hará lo que se quiere (en este caso, sumar los dos números) siempre y cuando estos números residan en estas localidades. Supóngase ahora que este mismo programa se carga en una localidad diferente (la localidad x'3000'). Si este programa se deja tal y como estaba codificado, seguirá buscando los valores de los números en las localidades x'1100' y x'1200', aunque ahora estas localidades ya no están en el área que le corresponde a este programa, es decir, el programa simplemente dejará de apuntar a los datos que realmente quiere accesar. En la figura 1.4(B) se muestra un programa que usa el esquema Base-Desplazamiento para direccionar a sus datos. Como se observa en la figura, no importa donde se carge el programa, este siempre apuntará a los datos que debe.

PROGRAMA NO RELOCALIZABLE

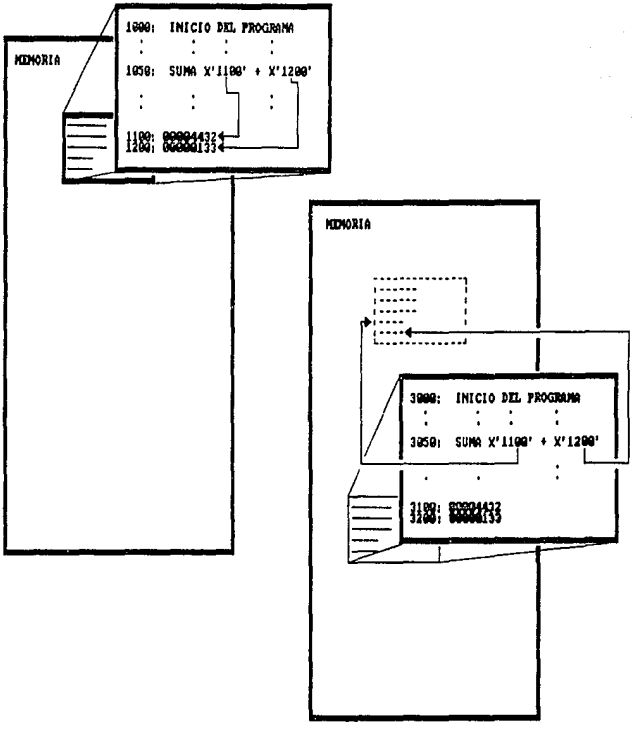


FIGURA 1.4 (A)

EJEMPLO DE PROGRAMA NO RELOCALIZABLE

PROGRAMA RELOCALIZABLE

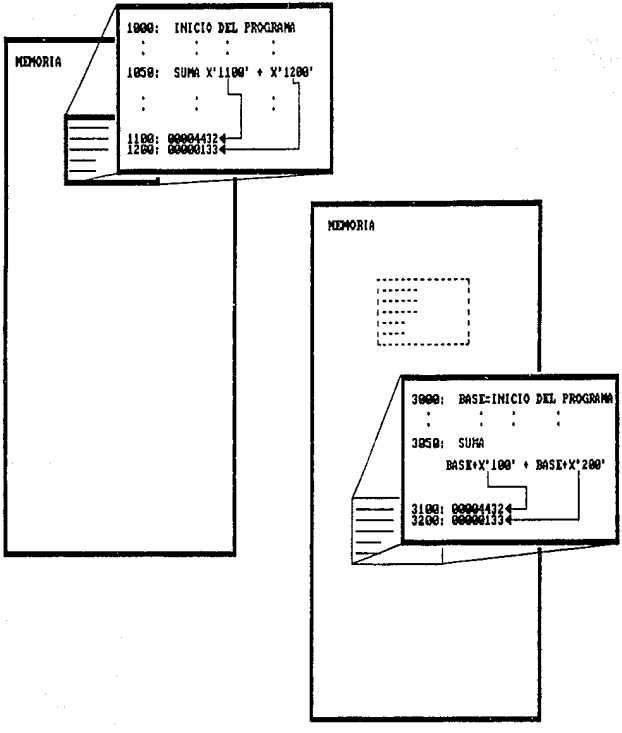


FIGURA 1.4 (B)

EJEMPLO DE PROGRAMA RELOCALIZABLE

b) "Linkedición"

El segundo paso consiste en formar un todo a partir de los módulos, macros, referencias a símbolos, etc. que entren en juego. Para esto se utiliza otro programa , el cual de hecho se utiliza no solo para el caso del lenguaje ensamblador, también para los compiladores, llamado "Editor de enlace" ó "Linkeditor", el cual recopila todo el código necesario para formar un programa completo y funcional a partir de:

1.- Código objeto de los módulos que se necesiten, que residen en bibliotecas de usuario.
2.- Código objeto de funciones del sistema, que residen en bibliotecas del sistema.
3.- Información de control suministrada por el programador para controlar la ejecución y el resultado de la edición de enlace. (información de control, que le informará al editor de enlace qué bibliotecas accesar, por ejemplo). Ver figura 1.5.

El "Linkeditor" asigna una dirección inicial al primer módulo y va asignando direcciones consecutivas a cada uno de los módulos que tengan relación, hasta formar un solo bloque de código.

El "Linkeditor" se encarga de resolver las referencias que se hagan de un módulo a otro módulo ajeno a el (resolver una referencia significa simplemente asignarle a un símbolo una dirección dentro del módulo), es decir, resolver las referencias externas.

El resultado de la linkedición es un "Módulo de carga", el cuál es relocalizable y tiene en él todo el código necesario para operar, excepto servicios del sistema operativo, como el código que maneja las "Llamadas a Supervisor" ("Supervisor Call", SVC) que es código común a todos los programas que corren bajo el control del sistema operativo y que ocupan áreas fijas de memoria, dado que resultaría ineficiente incluir en cada programa todo este código, que a fin de cuentas es común a todos.

De hecho, una Macro que invoque a una función de supervisor, no contiene en sí el código que realizará determinada función del sistema operativo, sino solo las instrucciones que le pasarán de manera correcta los parámetros de funcionamiento de la rutina de supervisor (sistema operativo) y le "Preparará el terreno" a esta rutina para que haga su trabajo correctamente.

Este módulo de carga, es el que se carga cuando se invoca al programa para su ejecución.

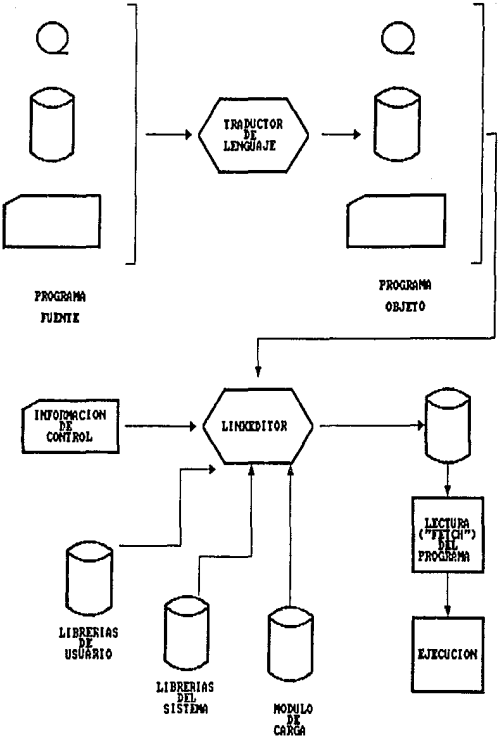


FIGURA 1.5

FLUJO DE DATOS DE UNA "LINKEDICION" DE MODULOS

El módulo de carga será guardado en bibliotecas de carga y los módulos fuente se guardarán en bibliotecas fuente para que, si por alguna razón, es necesario hacer una modificación al programa, esta modificación se haga en los módulos fuente y se lleve a cabo el proceso de ensamble (o compilación) y el de "linkedición".

1.3.5.- Ejecución de programas

Los programas, una vez cargados en memoria, empiezan a ejecutarse partiendo de la dirección inicial del mismo, dirección que suele denominarse punto de entrada ("Entry point") que no tiene porque coincidir con la dirección mas baja del programa, a la cual se le llama punto de carga ("Load point"). Es decir, supongamos que el cuerpo de un programa ocupa desde la dirección 0000 a la dirección 1000, pero la primera instrucción ejecutable del programa se ubica en la dirección 010, pues en las direcciónes 000 a la 00F hay caracteres que representan un identificador del programa, caracteres que definitivamente no son ejecutables.

1.3.6 Ejecución de trabajos

Se le llama "Trabajo" ("JOB") a la serie de programas relacionados que comprenden la totalidad de un procesamiento aplicativo. Por ejemplo, para procesar la nómina de una empresa, se deben correr varios programas que ejecutan tareas específicas en cierto orden, como pueden ser el programa que permite la captura de las percepciones, el programa que calcula los impuestos y deducciones, y el programa que imprime los recibos y cheques de nómina. Es obvio que se requiere que el proceso sea en un orden específico, pues no se pueden imprimir los cheques si antes no se han calculado las deducciones, impuestos y percepción neta de cada empleado.

La ejecución de un solo programa de procesamiento se llama "Paso de trabajo" ("Job Step"). Para que un proceso se lleve a cabo es necesario codificar y "Submitir" (es decir, ordenar que se ejecute) un "Job" compuesto de uno o mas "Job Steps". Este "Job" le indicará a la computadora mediante instrucciones especiales qué programas debe correr, en que orden, que dispositivos de I/O usar, que archivos tomar, etc.

1.4 PROCESAMIENTO CONCURRENTE

1.4.1 Introducción

Ahora se abordará la evolución que han tenido los sistemas operativos, haciendo énfasis en el cambio de la filosofía de diseño de los mismos a través del tiempo.

En esta sección se verá como se ha mantenido un esfuerzo constante en el diseño de los sistemas operativos por mantener lo más eficientemente utilizados los componentes del sistema. Se revisarán en forma muy general diversos métodos de manejo de I/O que buscaban el mantener a la CPU ocupada el mayor tiempo posible y como estos métodos finalmente desembocaron en las formas más comúnmente utilizadas y más ampliamente aceptadas de diseño para procesamiento concurrente: La Multiprogramación y el Multiprocesamiento.

1.4.2.- Evolución de los sistemas operativos

Los primeros sistemas computacionales no poseían un sistema operativo. Los programadores escribían sus programas en lenguaje máquina y los operaban directamente sobre la consola del operador. Primero, el programa era cargado en memoria desde un panel de interruptores, mediante una cinta de papel perforada o mediante tarjetas perforadas. Después se tenían que oprimir los botones indicados para cargar la dirección de inicio del programa. Mientras el programa corría el operador/programador podía supervisar la ejecución a través de luces indicadoras en la consola. Si se descubría algún error en el programa podían parar la ejecución del mismo, examinar los contenidos de memoria y los registros, y corregir el programa directamente en la consola. La salida del proceso era impresa o bien perforada en tarjetas o cinta de papel para su posterior impresión.

Un aspecto importante de este entorno era su naturaleza interactiva. El programador era el operador. Cada programador que deseaba correr su programa debía reservar un tiempo de máquina determinado para hacerlo, lo cual suponía que el tiempo que se reservaba no siempre fuera el más adecuado para lo que se quería hacer.

Conforme pasó el tiempo, se desarrolló nuevo software y hardware que hacían el trabajo más fácil, se inventaron las lectoras de tarjetas, impresoras de líneas, cintas magnéticas, etc. Además se diseñaron ensambladores, "Linkeditores", cargadores, etc. que hicieron la tarea del programador más fácil. Se crearon bibliotecas de funciones comunes, de tal forma que para incorporar una de estas funciones a un programa solo era necesario cargar una copia de una ya escrita, en lugar de tener que codificarla de nuevo.

Las rutinas que realizaban funciones de I/O fueron (y de hecho son) especialmente importantes. Cada nuevo dispositivo de I/O tenía sus propias características, lo cual requería que para usarlos se escribieran rutinas cuidadosamente planeadas y con pleno conocimiento del dispositivo. Se creó así una subrutina especial era escrita para cada dispositivo, estas subrutinas se les denomina

"Manejadoras de dispositivos" ("Device Drivers"). Un driver "conoce" como se deben utilizar las banderas,"Buffers", registros, bits de control, bits de estado, etc.

Más tarde aparecieron los compiladores para FORTRAN, COBOL y otros lenguajes, haciendo la tarea de escribir programas aplicativos más fácil, pero la operación de la computadora en sí más compleja, debido a los pasos que requería desarrollar, depurar y correr un programa.

Era realmente notable la gran cantidad de tiempo de preparación que era necesario para correr un "Job" así, que tenía varios pasos: Un paso para cargar el compilador, otro para cargar el módulo fuente, etc. Si ocurría un error en cualquier parte del proceso, se tenía que reprocesar todo de nuevo.

Poco después se introdujo el concepto de "Secuenciamiento automático de jobs", y con esto, el primer sistema operativo rudimentario era creado. Lo que se deseaba era un procedimiento que automáticamente transfiriera control de un "Job" al siguiente. Se codificó lo que actualmente se conoce como un "Monitor Residente", el cual, como su nombre lo indica, es un pequeño programa de monitoreo de "jobs" que siempre reside en memoria.

Inicialmente, cuando la computadora era prendida el control de la computadora era tomado por el monitor, el cual podía transferir el control a un programa. Cuando el programa terminaba, este regresaba control al monitor y este le daba control al siguiente programa. Una vez concluidos los programas de un "Job", se ejecutaba el siguiente "Job" y así sucesivamente.

Para poder saber que programas debía correr y en que orden, era necesario codificar una descripción del proceso, en la que se le indicaba que programas correr, en que cinta residían, que archivos de entrada iba a usar, que archivos de salida, etc. Estas sentencias de control reciben el nombre de Tarjetas de Control " Control Cards". El término de tarjeta viene del hecho de que se codificaban en tarjetas perforadas.

El monitor poseía un interprete de tarjetas de control, el cuál leía las tarjetas de control de una lectora de tarjetas, e interpretaba lo que estas trataban de decir.

Aún con la implantación del secuenciamiento automático de "jobs", la CPU todavía permanecía inactiva grandes cantidades de tiempo, el problema lo representa la diferencia intrínseca de velocidades entre la CPU y los dispositivos de I/O, los cuales son mucho mas lentos, debido a los elementos mecánicos que los integran.

Esta (relativa) lentitud de los dispositivos de I/O provocan que la CPU frecuentemente estén esperando por I/O.

Aunque con el paso del tiempo los dispositivos son cada vez mas rápidos, también se han incrementado las velocidades de la CPU, y tal parece que en proporción se sigue utilizando más del 80% del tiempo en operaciones de I/O.

Una de las primeras acciones para tratar de balancear esto fue la introducción de unidades de cinta magnética (bastante más rápidas) para sustituir lectoras y perforadoras de tarjetas. En lugar de que la CPU leyera directamente de la lectora de tarjetas, la información de las tarjetas era copiada a una cinta magnética. Cuando la cinta estaba suficientemente llena, era desmontada y llevada a la computadora central para ser leída por esta. De igual forma, la salida del proceso era escrita a cinta, para después ser perforada en tarjetas, o bien impresa si ese era el caso. El proceso de copiar la información de tarjetas a cinta o viceversa se hacía independientemente de la computadora central que procesara la información, a esto se le conoce como proceso "Fuera de Linea" ("Off-Line"), y a menudo se usaba un computador pequeño para tal propósito.

Otra solución para tratar de aliviar la relativa lentitud de los dispositivos de I/O es la técnica de "Buffering". Esta técnica intenta mantener a la CPU y a los dispositivos de I/O ocupados todo el tiempo. La idea es muy simple, después de que los datos se han leído y la CPU está a punto de operar sobre ellos, se hace que el dispositivo de entrada comience a leer la siguiente información de entrada inmediatamente, colocándola en un área de memoria que se conoce como "Buffer", de esta forma se tiene a la CPU y al dispositivo de I/O trabajen simultáneamente; esto permite que, con suerte, cuando la CPU esté lista para procesar los siguientes datos, estos ya estén en memoria. La CPU empezará a procesar estos nuevos datos, mientras el dispositivo de I/O ya estará leyendo los que siguen, y así sucesivamente.

Para el caso de la salida del proceso, los datos generados por el proceso se colocan en los "buffers", para que, a su paso, el dispositivo de salida los imprima,perfore,etc.

En la práctica la técnica de "Buffering" rara vez mantiene a la CPU y a los dispositivos de I/O todo el tiempo ocupados; sin embargo, hace más uniformes las variaciones en el tiempo que se emplea en procesar un registro. Si la velocidad promedio (en registros por segundo) de la CPU y los dispositivos de I/O son la misma, la técnica permite que la CPU esté ligeramente adelante o atrás de los dispositivos, operando ambos a toda velocidad.

Sin embargo, si la CPU, es en promedio mucho más rápida que los dispositivos de I/O (como generalmente es el caso), la técnica es poco eficiente.

Después aparecieron sistemas de disco que mejoraron enormemente la operación "Off-Line". El problema principal en los sistemas de cintas era que , por ejemplo, no se podía escribir leer en un extremo de la cinta mientras se escribía en otro. Es decir, se tenía que escribir toda la cinta antes de ser reembobinada y leída. Los sistemas de disco eliminaron este problema, a través de una cabeza móvil que permite al sistema moverse rápidamente de un área a otra de un disco; por ejemplo de un área donde se están escribiendo datos provenientes de una lectora de tarjetas a otra área donde se requiere leer datos escritos por medio de un teclado.

En un disco, las tarjetas pueden leerse directamente de una lectora de tarjetas y escribirse en el disco. La localidad de estas "imágenes" de tarjetas se guarda en una tabla mantenida por el sistema operativo. Cada "Job" se anota en esta tabla cuando se lee. Cuando el "Job" se ejecuta, sus requerimientos de lectura de tarjetas se satisfacen, no leyendo de una lectora de trajetas, sino del disco. Similarmente, cuando el "Job" requiere que su salida sea impresa, lo que se hace es copiar la información directamente a un "buffer" del sistema, y de ahí se transfiere al disco. Cuando el "Job" acaba de ejecutar, la impresión se realiza.

Este tipo de procesamiento se llama "Operación En Linea Periférica Simultánea" ("Simultaneous Peripheral Operation On-line", SPOOLing). La técnica de "Spooling" lo que hace escencialmente es usar al disco como un "buffer" muy grande, para leer de los dispositivos de I/O cuanto sea posible y para aceptar archivos de salida hasta que los dispositivos de salida estén disponibles.

La técnica de "Buffering" traslapa el I/O de un "Job" con su propio proceso. La ventaja del "Spooling" sobre el "Buffering" es que el "Spooling" traslapa el I/O de un "Job" con el proceso de otros "jobs". Aún en sistemas muy simples, el "Spooler" puede estar leyendo la entrada de un "Job" mientras se está imprimiendo la salida de otro "Jobs". Durante este tiempo, puede ser que otro "Job" (o jobs) estén ejecutando, leyendo sus "tarjetas" del disco e "imprimiendo" sus salidas en el disco. El "Buffering" solo puede traslapar el I/O de un "Job" con su propio proceso, mientras que el "Spooling" puede traslapar el I/O de muchos "Jobs" con el proceso de muchos otros.

Está técnica es ya parte esencial de muchos sistemas operativos (como el MVS, que se estudiará en el capitulo III), sin embargo cuando se introdujo el sistema/360 de IBM, su sistema operativo (el OS/360) no contaba con "Spooling" (a mediados de los 60's). El "Spooling" fue agregado como una característica especial por el centro de computación de Houston de la NASA (era el "Houston Automatic Spooler Program", HASP).

La técnica de "Spooling" tiene un efecto directo sobre el rendimiento del sistema: Por el costo de espacio en disco y algunas tablas en memoria, la CPU puede traslapar el proceso de un "Job" con el I/O de otros "jobs", de tal forma que se mantenga tanto a la CPU como a los dispositivos de I/O trabajando mucho más tiempo.

Adicionalmente a esto, el "Spooling" introduce una estructura de datos importante : Un "Pool de Jobs". Implantar la técnica de "spool" resulta en una serie de "Jobs" leídos y esperando en disco, listos para correr. Un "Pool de Jobs" en disco permite que el sistema operativo seleccione que "Job" correrá a continuación, para incrementar la utilización de la CPU; esto se conoce como "Planificación" (Scheduling) de "Jobs".

1.4.3.- Multiprogramación.

Tal vez el aspecto más importante de la Planificación ("Scheduling") de "Jobs" sea la habilidad de implantar lo que se conoce como "Multiprogramación". La operación "Off-line", el "Buffering", y el "Spooling" para traslapar el I/O de jobs con el proceso de otros, tiene sus limitaciones. Un usuario aislado no puede, en términos generales, mantener a la CPU y a los dispositivos activos simultáneamente todo el tiempo.

El avance de la tecnología permitió la construcción des de CPUps cada vez más rápidas. Las técnicas anteriormente descritas no fueron ya capaces de garantizar nada que se aproximara a un nivel aceptable de utilización de la máquina. La entrada y salida continuaron siendo tan importantes como antes, pero el tiempo necesario para terminar una operación de I/O parecía interminable, cuando se les comparaba con las velocidades de las nuevas máquinas.

La técnica de multiprogramación es una solución al problema de incrementar la utilización de la CPU haciendo que esta siempre tenga algo que ejecutar.

La idea fundamental es como sigue: El sistema operativo escoge uno de los "Jobs" del "Job Pool" y empieza a ejecutarse, eventualmente el "Job" tendrá que esperar por algo, como por ejemplo que se monte una cinta, que se tecleé un comando desde el teclado, o que se complete una operación

de I/O. En un sistema no multiprogramado (uniprogramado) la CPU simplemente estaría inactiva; en un sistema multiprogramado, el sistema operativo simplemente se cambiará a otro trabajo y lo empezará a ejecutar. Cuando este segundo trabajo necesite esperar por algo, la CPU se cambiará a otro trabajo, y así sucesivamente. Eventualmente, el primer trabajo habrá terminado de esperar y retomará el control de la CPU. Esto permite que mientras exista algo que ejecutar, la CPU nunca estará inactiva.

Siempre que se trate de procesos concurrentes (procesos que se ejecutan en memoria al mismo tiempo) se tiene que buscar un compromiso entre dos conceptos: El rendimiento y el tiempo de respuesta; el rendimiento es la cantidad de trabajo que procesa la computadora, el tiempo de respuesta es la medida de tiempo que transcurre desde el comienzo de un trabajo hasta la terminación del mismo. Los objetivos de cualquier sistema se definen en términos de "Maximizar el rendimiento, conservando un tiempo de respuesta razonable", y es importante observar que estos dos objetivos entran en conflicto. En un mismo sistema pueden estar compitiendo programas que atienden a los usuarios en tiempo real (es decir, sistemas a los cuales se les hace un requerimiento desde una terminal y se espera una respuesta en un tiempo de unos cuantos segundos a lo más, sistemas llamados "On-line" o en linea) y programas que están ejecutándose pero no se interactúa con ellos, si no que simplemente se espera de ellos un resultado al final de su ejecución (por ejemplo, un sistema que actualice cuentas de tarjetas de crédito, cuyo resultado final serán los recibos con el estado de cuenta de cada usuario de tarjeta de crédito).

1.4.4 Multiprocesamiento.

Otro método de optimización de uso de recursos que es bastante reciente y es consecuencia de un impresionante avance en la tecnología de procesadores , es el "Multiprocesamiento". El término se refiere a dos o más procesadores compartiendo memoria real y manejados por un solo sistema operativo. Con la multiprogramación se tiene dos o más programas y un solo procesador. Debido a que solo hay un procesador , es imposible ejecutar dos o más instrucciones al mismo tiempo. Con el multiprocesamiento, esto ya es posible, ya que cada uno de los procesadores del "Complejo" pueden estar ejecutando instrucciones distintas a la vez. Esta configuración con varios procesadores incrementa de manera espectacular la potencia de cálculo del sistema; sin embargo, crea otros problemas. Por ejemplo, se debe tener un control estricto de el acceso a las mismas direcciones de memoria al mismo tiempo, pues mientras uno de los procesadores está cambiando el contenido de una localidad de memoria, el otro la puede estar accesando, lo cual puede provocar resultados impredecibles.

Para resolver esto, los sistemas operativos deben contar con mecanismos de "Serialización" de recursos, que permitan que mientras un procesador esté accesando algún recurso, el otro procesador no intente hacerlo al mismo tiempo.

En el siguiente capitulo se introducirá la arquitectura del Sistema/370 de IBM, la cual soporta al sistema operativo MVS, el cual se estudiará en el capitulo III.

CAPITULO II.- LA ARQUITECTURA DEL SISTEMA/370

El presente capitulo cubrirá los conceptos básicos acerca de la arquitectura del Sistema/370 (en general como se programa, se usa y controla el sistema), necesarios para entender la filosofía de diseño del MVS, que se estudiará en el presente trabajo.

2.1 INTRODUCCION

El Sistema/370 es, en concepto, un sistema de gran capacidad, orientado a su uso tanto en aplicaciones comerciales que requieran una alta actividad de I/O, como en aplicaciones numéricas intensivas. Tiene capacidad de manejo de grandes cantidades de información a través de numerosos dispositivos de I/O, así como de manejo en memoria de operaciones numéricas vectoriales y escalares con gran velocidad. Es, en resumen, un sistema de uso general en ambientes que justifiquen la gran capacidad de manejo de datos que proporciona el sistema. Actualmente podemos decir que marca la pauta de sistemas de gran capacidad a nivel mundial; Las principales y más grandes compañías del mundo cuentan al menos con una plataforma basada en el Sistema/370, o bien de su sucesor el Sistema/390.

Un objetivo de diseño del Sistema/370 es su modularidad. Es decir, a los usuarios se les da capacidad de escoger entre una gran cantidad de opciones, tanto en hardware como en software, no sólo en su configuración inicial, sino que esta capacidad continúa a medida que las necesidades del sistema crecen o se modifican.

Aunque se sale de los objetivos del presente trabajo dar una descripción detallada de todas las opciones posibles, se puede mencionar por ejemplo que no sólo hay una gran variedad de CPU's, hay también un rango muy grande de capacidades de memoria principal para cada modelo, una impresionante cantidad de dispositivos de I/O, desde los tradicionales (impresoras, terminales, discos, unidades de cinta) hasta dispositivos "Estado del arte" como impresoras láser, unidades de respuesta de audio, discos ópticos, etc.

En términos de flexibilidad podemos hablar de la capacidad del sistema para manejar, tanto procesos "batch", como procesos en ambiente de telecomunicaciones (en línea) concurrentemente.

Además, toda la serie de CPU's, memorias principales, y dispositivos de I/O nacidos a partir del Sistema/370 son totalmente compatibles entre ellos, los programas escritos en un sistema, corren sin modificación en otro Sistema/370 , no importa cuán diferentes sean uno del otro en cuanto a velocidad, I/O, memoria.

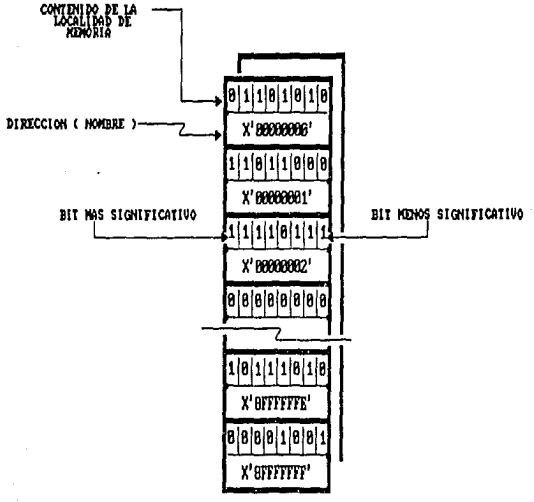


FIGURA 2.1
REPRESENTACION DE LA MEMORIA PRINCIPAL

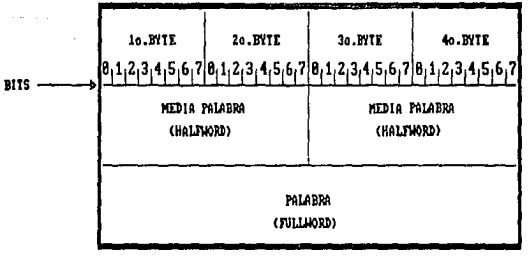


FIGURA 2.2
FRONTERAS DE MEMORIA

Adicionalmente, cada Sistema/370 es capaz de soportar a diferentes sistemas operativos, dependiendo de las necesidades de la instalación, por ejemplo VM, VSE, MVS, etc.

Por último, uno de los principales objetivos de diseño del sistema es tratar de eliminar o al menos disminuir los efectos de una falla del sistema. A través de la interacción del hardware con el sistema operativo, se reintentan operaciones fallidas o bien se ignoran las fallas de la máquina, tratando de aislar estas fallas que resultarían en un mal funcionamiento del sistema en su conjunto para que afecte a la menor cantidad de usuarios posible.

2.2 MEMORIA PRINCIPAL

Cada localidad de memoria del Sistema/370 tiene un nombre, a este nombre se le conoce como "Dirección". En cada localidad de memoria se pueden almacenar 8 bits (un byte) de información (figura 2.1).

La unidad de almacenamiento accesible más pequeña en el Sistema/370 es el byte. La convención que se usa para designar a los bits de un byte se representa en la figura 2.1. Los bits son numerados del 0 al 7, contando de derecha a izquierda, en donde el bit 0 es el más significativo y el bit 7 es el menos significativo.

Por convención y comodidad, el contenido de las localidades de memoria y la dirección de las mismas, no se representan como un patrón de 8 bits, si no como un número "Hexadecimal. A lo largo de este trabajo se representarán los números hexadecimales como:

x' Número Hexadecimal'

En la organización de la memoria del Sistema/370 se conoce como "Palabra" ("Word") a un grupo de 4 bytes, "Media palabra" son dos bytes y una "Doble Palabra" son 8 bytes de memoria. Una palabra está representada por 8 dígitos hexadecimales.

La memoria principal comienza con la dirección cero que corresponde al primer byte de la memoria, y esta dirección se incrementa en uno para cada byte sucesivo. Debido a que cada byte tiene una dirección única y es la unidad direccionable más pequeña dentro de la memoria principal, el byte es considerado el bloque básico de construcción de la memoria del Sistema/370.

El Sistema/370 está diseñado para trabajar con datos de longitud variable y longitud fija: Direccionando el primer byte de un campo y especificando el número de bytes deseados, se puede tener un campo de longitud variable en memoria principal, el cual puede comenzar y acabar en cualquier dirección de memoria.

Existen algunas instrucciones de máquina que necesitan que los datos sobre los cuales va a operar se encuentren alineados en una "Frontera", que puede ser de media palabra, de palabra o de doble palabra. Las "fronteras" son como lineas imaginarias que dividen a la memoria en medias palabras, palabras y dobles palabras. Las direcciones alineadas en fronteras son múltiplos de la longitud de esa frontera. (figura 2.2)

Las direcciones de medias palabras siempre empiezan en direcciones que terminan en número par (x'0,2,4,6,8,A,C,E), las palabras siempre empiezan en direcciones que terminan en x'0,4,8,C' y las dobles palabras siempre terminan en x'0,8'.

2.3 FORMATOS DE DATOS

El Sistema/370 soporta diversos formatos de datos cuyo uso depende de la aplicación que se quiera dar a estos datos, por ejemplo: Caracteres, Números enteros o de punto flotante, Indicadores, etc.

2.3.1 Formato EBCDIC

El primer formato es el EBCDIC ("Extended Binary Coded Decimal Interchange Code"), también conocido como Decimal Zoneado ("Zoned Decimal"). El código EBCDIC usa 8 bits (un byte) para representar un caracter de datos, y normalmente se expresa en términos de dos dígitos hexadecimales, por ejemplo el caracter "A" se representa en EBCDIC como x'C1' (ó 1100 0000). Dado que el código se compone de 8 bits, se pueden representar 256 (2^8) caracteres EBCDIC. Aunque el Sistema/370 usa un gran porcentaje de estos, los más usados son: 26 letras mayúsculas y 10 dígitos numéricos.

El formato EBCDIC es un formato de datos que se usa extensivamente en el Sistema/370. Debido a que aplica principalmente a datos "Alfanuméricos" (es decir, letras, números y caracteres especiales), se usa para representar datos en memoria que se leen desde una terminal, una lectora de tarjetas, etc., o datos que se guardan en un dispositivo de memoria auxiliar (cinta magnética, discos, etc), o datos que se envian a un dispositivo de salida como una impresora.

2.3.2 Formato Empacado Decimal

El código EBCDIC no es usado por el Sistema/370 para realizar operaciones aritméticas. Para esto, existen otros formatos de datos disponibles.

Los números decimales en formato EBCDIC se representan anteponiéndoles una "F" (x'F0' es la representación de un cero, mientras que x'F7' es un siete). Si se necesitan guardar cantidades considerables de datos numéricos en memoria, este formato ocuparía grandes cantidades de memoria ineficientemente. Para evitar esto, se cuenta con un formato "Decimal Empacado", el cual, como su nombre lo indica conserva la orientación decimal, pero reduce la cantidad de memoria requerida. El formato decimal empacado es como sigue: Los datos numéricos sin signo se representan como el número decimal tal como se escribiría pero con el dígito hexadecimal x'F' al final. Sin embargo, si el número tiene signo, el último dígito representa el signo. Los signos válidos para decimales empacados son:

Binario	Hexadecimal	Signo
1010	A	+
1100	C	+
1110	E	+
1011	B	-
1101	D	-

Ejemplos de números decimales empacados sin signo:

Número	Formato	Representación
7502	EBCDIC	F7 F5 F0 F2
	EMPACADO	07 50 2F
1472859	EBCDIC	F1 F4 F7 F2 F8 F5 F9
	EMPACADO	14 72 85 9F

Ejemplos de decimales empacados con signo:

+2378	EBCDIC	F2 F3 F7 C8
	EMPACADO	02 37 8C
-7874	EBCIDIC	F7 F8 F7 D4
	EMPACADO	07 87 4D

Los símbolos "preferidos" para representar el signo positivo y negativo son x'C' (1100) para positivo y x'D' (1101) para negativo.

Como se observa, los datos se guardan de manera más eficiente en forma empacada. La disminución de bytes requeridos para guardar un número en memoria es mayor (en porcentaje) entre más grande sea este número.

Existen instrucciones en el Sistema/370 para realizar operaciones aritméticas en forma decimal empacada, de longitud variable. Sin embargo, no se pueden realizar operaciones matemáticas con datos en formato EBCDIC.

2.3.3 Formato Binario

Este formato puede parecer confuso; los datos en formatos EBCDIC o empacados decimales se representan por una serie de bits "Binarios". Los datos en formato binario no sólo están formados por bits binarios, si no que también representan números binarios, no números decimales. Esto es, cada bit sucesivo (de derecha a izquierda) representa un multiplicador (cero o uno) de 2 elevado a una potencia. En otras palabras, en formato EBCDIC el número x'F2' representa al número 2, en formato binario representa al número 242.

Si este bit de signo vale cero el número es positivo, si este bit vale uno, el número es negativo y el número está representado en la forma de "Complemento a dos".

El formato binario representa una forma muy rápida y eficiente de guardar y procesar datos en el Sistema/370. Una palabra puede representar un valor numérico de más de 2 millones. Esta eficiencia que se menciona viene del hecho de que los datos se procesan en el Sistema/370 dentro de 16 "Registros Generales", con los cuales, debido al diseño del hardware, el tiempo requerido para cargar y recuperar datos de estos registros generales es mínimo. Estos registros generales, que se estudiarán más tarde, guardan exactamente una palabra de información.

Esto es especialmente significativo para crear y accesar direcciones en memoria principal; se hace notar que estas direcciones representan números que están expresados en formato binario y representan una parte significativa de la actividad de procesamiento que se lleva a cabo dentro del sistema.

2.3.4 Formato de Punto Flotante

Este formato resulta irrelevante para la comprensión de temas posteriores por lo que se omitirá de esta discusión. Solo se mencionará que en este formato los datos numéricos se representan por medio de un exponente y una mantissa.

El formato decimal empacado es usado frecuentemente en aplicaciones comerciales. Estas aplicaciones, tipicamente no realizan una gran cantidad de operaciones numéricas.

El formato de punto flotante es usado frecuentemente en aplicaciones científicas, en donde están involucradas cantidades muy grandes o muy pequeñas. Este tipo de aplicaciones usualmente requieren gran cantidad de cálculos matemáticos.

El formato binario es una buena opción para información de control, como contadores, que son usados para tener registro, por ejemplo, de cuantas veces se ha pasado por alguna parte de un programa, o cuantas veces se ha realizado un "Loop". Estos números son generalmente integros (sin parte decimal) y normalmente no son parte de los datos que serán impresos.

2.4 LA UNIDAD CENTRAL DE PROCESO ("Central Processing Unit", CPU)

Ya se ha visto que la memoria principal en el Sistema/370 se usa para mantener (o guardar) datos e instrucciones. Sin embargo, la memoria principal es un dispositivo pasivo. Realmente no "Hace" nada, el trabajo real de la computadora es realizado en la unidad central de proceso (CPU). La CPU es el centro de control y trabajo del sistema.

2.4.1. Estructura lógica de la CPU

La figura 2.3 muestra la relación entre la CPU y la memoria principal. Muestra que las instrucciones a ser ejecutadas y los datos a ser manipulados son obtenidos de la memoria principal.

La figura 2.4 muestra la organización de la CPU. La CPU tiene dos secciones que llevan a cabo cada una de sus funciones principales: Una Unidad Aritmético-Lógica ("Arithmetic Logic Unit", ALU) y una "Sección de Control del Sistema" ("System Control Section").

2.4.2 Unidad Aritmético Lógica ("Arithmetic Logic Unit",ALU)

La función de la unidad aritmético lógica (ALU) es realizar operaciones sobre los campos de datos. En general, la ALU contiene la circuitería necesaria para realizar operaciones aritméticas, como suma, resta, multiplicación y división,así como operaciones lógicas (AND, OR y XOR), comparaciones, movimientos entre campos, y desplazamientos de bits.

Se puede observar de la figura 2.4 que las operaciones con datos de tamaño fijo usualmente son realizadas en registros.

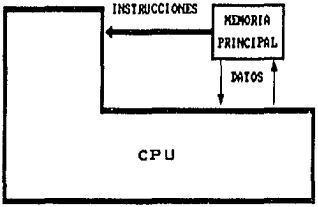


FIGURA 2.3
RELACION ENTRE LA CPU Y LA MEMORIA PRINCIPAL

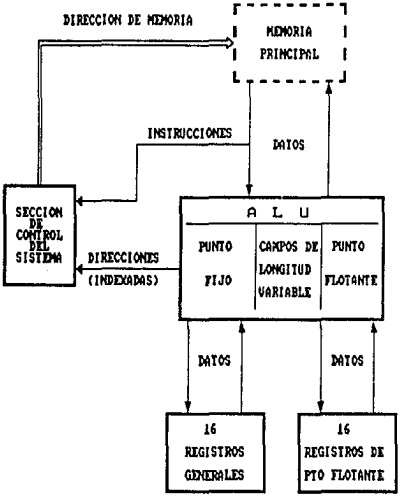


FIGURA 2.4
ESTRUCTURA LOGICA DE LA CPU

El Sistema/370 cuenta con 3 clases de registros, los cuales son:

Registros generales ("General Purpose Registers",GPRs)
Registros punto flotante ("Floating Point Registers",FPRs)
Registros de Control ("Control Registers",CRs)

2.4.3 Registros Generales

Uno de los usos de los registros generales es realizar operaciones en datos binarios de punto fijo. Sin embargo, como se verá más adelante, se usan para una gran cantidad de cosas, de ahí el nombre de "Generales". Existen 16 registros generales, numerados del 0 al 15 (x'0' al x'F'). Cada uno de estos registros mide 32 bits (una palabra). Para el caso de aritmética binaria, el primer bit se usa como signo. Los registros tienen el mismo formato que una palabra en memoria principal, aunque no residen en memoria principal. Las operaciones con medias palabras se expanden automáticamente a una palabra en un registro (es decir, se le agregan ceros a la media palabra más significativa del registro).

2.4.4 Registros de Punto Flotante

Los registros de punto flotante se usan para realizar operaciones en datos de punto flotante. Existen 4 registros de punto flotante, cada uno de los cuales mide dos palabras (64 bits) y están numerados como 0,2,4 y 6. Estos registros no son los mismos que los registros generales 0,2,4 y 6.

2.4.5 Registros de Control

Existen 16 registros de control, cada uno de los cuales tiene una longitud de 32 bits (una palabra). Estos registros se usan para guardar información de control, y el programador del sistema no tiene acceso a ellos.

En la figura 2.4 se muestra que las operaciones con datos de longitud variable no se realizan en registros. En este tipo de instrucciones, una sola instrucción causa que los operandos se lean de memoria, que la operación sobre estos datos se realice, y que el resultado se retorne a memoria. Las operaciones aritméticas en formato decimal empacado y las operaciones de manipulación de caracteres se realizan de esta forma.

Las operaciones que involucran registros son generalmente más rápidas que las operaciones de memoria a memoria debido a la circuitería más rápida usada para los registros. Por esta misma razón, las operaciones de registro a registro son más rápidas que las operaciones de registro y memoria.

2.5 SECCION DE CONTROL DEL SISTEMA ("System Control Section", SCS)

Tanto las instrucciones como los datos se cargan en memoria principal; de aquí se puede deducir que debe existir "Algo" que controle el acceso a esta memoria y la operación de la CPU en general. Ese "Algo" es la "Sección de control del sistema" (SCS) cuyas principales funciones son:

1) Localizar, leer y ejecutar instrucciones del programa que se encuentre en memoria y
2) Localizar y leer datos sobre los que va a operar la ALU.

2.6 INSTRUCCIONES DE MAQUINA Y FORMATOS

Las instrucciones especifican la operación que se va a hacer y la localización de los datos sobre los cuales se va a operar. Los datos pueden estar tanto en memoria principal, registros generales o ambos. La memoria principal se accesa con una dirección que mide 24 bits en el Sistema/370 o 31 bits en el Sistema/370 con Arquitectura Extendida (esto se verá un poco más adelante) mientras que los registros generales se direccionan con 4 bits. Como resultado de esto, las instrucciones pueden ser de diferentes tamaños, dependiendo de donde se localicen los datos. Las instrucciones pueden ser de una, dos o tres medias palabras y siempre están alineadas en frontera de media palabra.

Los formatos de instrucciones muestran donde están los operandos y por lo tanto implican una longitud (figura 2.5). Estos formatos son:

Registro a Registro ("Register to Register") (RR)
Registro y Memoria ("Register and Storage") (RS o RX)
Dato implícito y Memoria ("Implied and Storage") (S)
Memoria y dato inmediato ("Storage and immediate") (SI)
Memoria a Memoria ("Storage to Storage") (SS)

En el método de Registro a Registro (RR), ambos operandos están contenidos en registros y el resultado aparece en uno de los registros.

En el método de Registro y Memoria (RS y RX) uno de los operandos está en memoria principal y el otro está en un Registro. El resultado de la operación puede aparecer, ya sea en un registro o en memoria principal.

En el método de Memoria y dato inmediato (S) uno de los operandos está en memoria y ocupa un byte de la instrucción misma. Este byte dentro de la instrucción es llamado el dato inmediato. El resultado aparece en memoria.

En el método de dato implícito y memoria (SI) uno de los operandos reside en memoria y el otro es un dato implícito

del sistema, como la "Hora del día" ("Time of Day Clock",TOD).

En el método de memoria a memoria (SS) los campos de datos se traen de la memoria principal, se opera y el resultado es colocado de nuevo en memoria principal.

2.6.1 Códigos de operación

Nótese en la figura 2.5 que el primer byte de cada instrucción es un "Código de operación" ("Operation Code", Opcode), excepto en el caso de las instrucciones de formato S, en el que el "opcode" mide dos bytes. El "Opcode" especifica:

La operación a ser realizada.
El formato de los datos.
La longitud de la instrucción.

2.6.2 Direcciones de operandos

El resto de la instrucción se usa para especificar la dirección de los datos (para el caso de las instrucciones con formato S, donde los datos están contenidos en la instrucción misma, el resto de la instrucción especifica a los datos mismos). Los campos en las instrucciones que se usan para referenciar los datos en memoria se llaman "Operandos de dirección". Aquellos que se utilizan para referirse a datos en registros se llaman "Operandos de registro".

Como ya se dijo, los datos en memoria se direccionan por su primer byte. Sin embargo, la computadora necesita saber cuantos bytes debe tomar. Para el caso de instrucciones que tratan con datos de longitud fija, el formato de los datos está implícito en el código de operación de la instrucción, estas instrucciones "saben" de que tamaño es el campo de datos sobre el que van a operar. (por ejemplo, existe una instrucción que suma una palabra completa a un registro "ADD" y una instrucción "ADD HALFWORD" que suma siempre una media palabra a un registro).

Las instrucciones SS siempre tratan con datos de longitud variable y no pueden predecir o "saber" el tamaño de campo de datos que el programador usará. El programador debe proporcionar las longitudes de los campos de datos en la instrucción. La figura 2.5 muestra como las instrucciones en formato SS usan un campo de 8 bits (1 byte) para especificar la longitud (nótese que los campos están etiquetados como L en la figura). En el caso de un sólo campo de 8 bits, se asume que los dos operandos son de la misma longitud (que puede ser tan grande como 256 bytes).

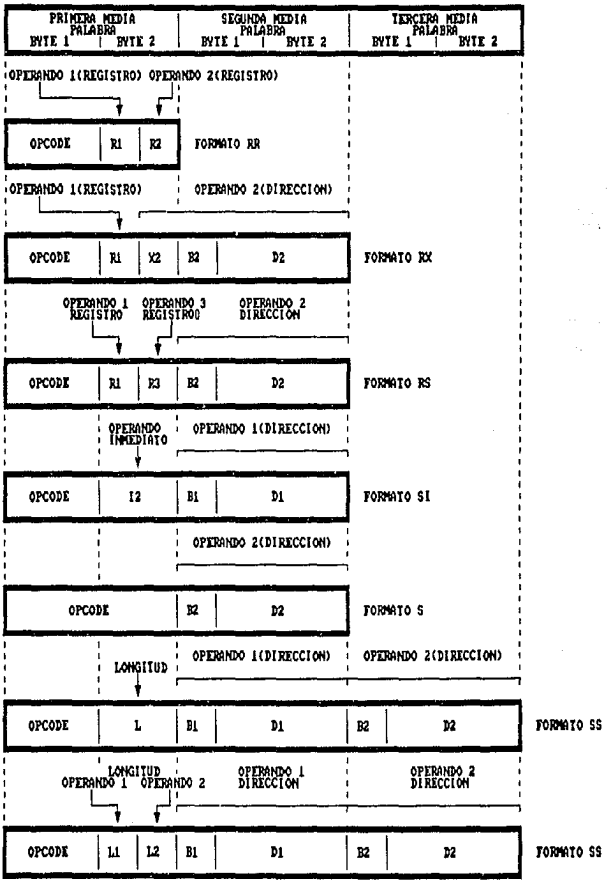


FIGURA 2.5

FORMATOS DE INSTRUCCIONES

Por ejemplo, la instrucción MOVE que se usa para mover datos de un campo a otro, usa una sola longitud de 8 bytes. Para el caso de campos que se especifican con dos longitudes de 4 bits cada una (una longitud para cada operando), los operandos pueden tener longitudes distintas de hasta 16 bytes de longitud. Como estas instrucciones tratan con datos decimales empacados, los campos pueden contener hasta 31 dígitos decimales y un signo.

2.7 LA PALABRA DE ESTADO DE PROGRAMA ("Program Status Word", PSW)

Una instrucción se direcciona también por su primer byte, del mismo modo que se direcciona un campo de datos. La cuestión ahora es la forma en que la CPU localiza la instrucción que ejecutará. Para este efecto se usa la "Dirección de la Instrucción" ("Instruction Address", IA). La IA se encuentra en un registro especial llamado la "Palabra de Estado del Programa" ("Program Status Word", PSW). La PSW es una doble palabra que se usa para guardar información que muestra el estado de la CPU en todo momento. Como se muestra en la figura 2.6, en el Sistema/370 la IA está guardada en los 24 bits menos significativos de la segunda palabra de la PSW.

En el Sistema/370-XA ("Extended Architecture") se extendió este campo de IA a 31 bits, con lo que el Sistema/370-XA es capaz de direccionar 2 GBytes de memoria con 31 bits, en contra de 16 Mbytes que era capaz de direccionar el Sistema/370 con 24 bits.

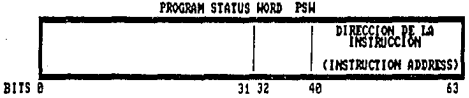
La PSW además de la IA, guarda información adicional del estatus de la CPU que se irá analizando poco a poco (figura 2.8). Entre esta información adicional se encuentra un bit que nos indica si se está direccionando con 24 o con 31 bits (bit 32 de la PSW).

Por el momento sólo se explicará la función del campo de IA en la PSW: La IA normalmente contiene la dirección de la siguiente instrucción a ser ejecutada. Como una dirección "Apunta" hacia donde se encuentra algo, se puede decir que la IA apunta a la dirección de la siguiente instrucción a ser ejecutada.

2.7.1 Ejecución Secuencial

Cuando la CPU está lista para realizar una instrucción, toma la dirección contenida en el campo IA de la PSW, va hacia esa dirección, recupera la instrucción y la coloca en la CPU. Ya en la CPU, la instrucción es interpretada y ejecutada.

SISTEMA/370



SISTEMA/370-XA

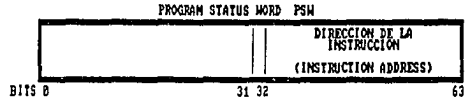
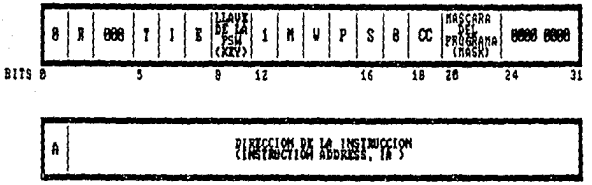


FIGURA 2.6

FORMATOS DE LA "PALABRA DE ESTATUS DE PROGRAMA"



BITS	FUNICION
1 (R)	MASCARA DE GRABACION DE EVENTOS DE PROGRAMA (PROGRAM EVENT RECORDING, PER)
5 (T=1)	MODO DAT
6 (I)	MASCARA DE INTERRUPCIONES DE I/O
7 (E)	MASCARA DE INTERRUPCIONES EXTERNAS
13 (M)	MASCARA DE INTERRUPCIONES DE VERIFICACION DE MAQUINA ("MACHINE CHECK")
14 (W)	ESTADO DE ESPERA
15 (= 1)	ESTADO PROBLEMA
(= 0)	ESTADO SUPERVISOR
16 (= 1)	MODO DE ESPACIO SECUNDARIO
18-19 (CC)	CODIGI DE CONDICION (CC)
20	MASCARA DE SOBREFLUJO PARA OPERACIONES DE PUNTO FIJO
21	MASCARA DE SOBREFLUJO PARA OPERACIONES DECIMALES
22	MASCARA DE SOBREFLUJO PARA OPERACIONES CON EXPONENTES
23	MASCARA DE SIGNIFICANCIA
32 (= 1)	MODO DE DIRECCIONAMIENTO DE 31 BITS
(= 0)	MODO DE DIRECCIONAMIENTO DE 24 BITS

FIGURA 2.7

DEFINICION DE LOS CAMPOS DE LA "PSW"

La CPU sabe cuantos bytes leer a partir de la dirección contenida en la IA gracias al "Opcode", debido a que parte de la información contenida en el "Opcode" es la longitud de la instrucción. Una vez que el "Opcode" se lee, se determina cual va a ser el "Código de Longitud de la Instrucción" ("Instruction Lenght Code", ILC). La localidad de la siguiente instrucción se determina sumando el ILC a la dirección de la IA. A medida que esta operación se repite, se ejecutan las instrucciones secuencialmente.

2.7.2 Ejecución No secuencial

Existen dos formas de romper con la ejecución secuencial de instrucciones: Una de ellas es reemplazar toda la PSW con una nueva PSW cuyo campo de IA contenga una dirección que no corresponda a la siguiente instrucción de la que se acaba de ejecutar (más adelante se estudiará como y por que medios se logra esto). La segunda forma es a través de instrucciones de bifurcación. Estas instrucciones tienen la habilidad de cambiar la IA para que apunte a una instrucción diferente de la siguiente instrucción secuencial. Se dice que el programa "Bifurca" a la dirección de esa instrucción.

2.7.3 Instrucciones de bifurcación

Las instrucciones de bifurcación ("Branch") condicional permiten al programador hacer decisiones; una instrucción de bifurcación condicional permite continuar con la siguiente instrucción secuencial o bien, si se cumple una condición determinada, bifurcar a otra localidad de memoria. El que se bifurque o no, depende de una condición que "Prueba" la instrucción, tal como que:

1) El resultado de una instrucción sea mayor que, menor que o igual a cero, 2) que comparando dos campos, uno de ellos sea mayor que, menor que o igual al otro.

Estas condiciones las guarda el sistema al completarse la ejecución de la instrucción (por ejemplo de suma) en un campo de la PSW (figura 2.7) que se conoce como el "Código de Condición " ("Condition Code", CC). Este código es un campo de dos bytes que puede identificar hasta 4 condiciones (00,01,10,11 o bien 0,1,2,3). Por ejemplo, como resultado de una operación de suma, el CC puede adquirir uno de los siguientes valores:

0 - La suma es cero.
1 - La suma es mayor que cero.
2 - La suma es menor que cero.
3 - Sobreflujo (Overflow).

Esto quiere decir que si se examina el contenido del campo de CC después de una operación de suma y encuentra que este campo tiene el valor de 2, quiere decir que el resultado de la suma es mayor que cero.

Una instrucción de bifurcación condicional prueba el contenido del CC para decidir si realiza una bifurcación o no. Para esto se utiliza un valor en una "Mascara" en donde se hace corresponder a 4 bits cada uno de los valores del CC, de la siguiente forma (para el caso de una suma):

Condition Code	Bits de la máscara	Valor de la máscara	Posible Interpretación
0	1 0 0 0	(8)	Resultado igual a 0
1	0 1 0 0	(4)	Resultado menor a 0
2	0 0 1 0	(2)	Resultado mayor a 0
3	0 0 0 1	(1)	Sobreflujo.

Esto quiere decir que si se asigna el valor de 12 a la mascara y se ejecuta una instrucción de bifuración condicional, se esta probando que el valor del CC sea 8 o bien 4 (8 + 4 = 12, el valor de la mascara), si esto es cierto, el programa hará un salto a la dirección especificada en la instrucción. La instrucción codificada en ensamblador se vería así:

```
BC  12,ADDR
```

El nemónico de la instrucción es BC ("Branch on Condition" ,BC) ó "Bifurcación en una Condición", el valor de la máscara es 12, y si se cumple la condición (en este caso, que el CC sea igual a 0 o igual a 1) se bifurcará a la instrucción que se localiza en la etiqueta ADDR.

Se puede lograr una bifurcación "Incondicional", es decir, que se bifurque a la dirección señalada no importando el valor que tenga en este momento el CC si se hace que la mascara tome un valor de 15 (x'F'), debido a que este valor de máscara incluye todos los valores posibles del CC (15 = 8 + 4 + 2 + 1) , y por lo tanto bifurcará bajo cualquier condición.

2.8 DIRECCIONAMIENTO BASE/DESPLAZAMIENTO

Una vez que la CPU ha leído de la memoria principal la instrucción que va a ejecutar, se analizan las direcciones de los operandos para determinar en que registro o en que localidad de memoria residen los datos.

Para efectos de direccionamiento, un programa está dividido en secciones, cada una de las cuales se llaman "Páginas" y son de 4,096 bytes (4 KBytes) de longitud.

La dirección de inicio de cada sección se llama la dirección "Base" de esa sección.

Considérese el caso en el que un programa requiere 12,000 bytes. Si se divide al programa en páginas de 4,096 bytes, tendremos 3 secciones del programa, cada una con una dirección base. Como se puede ver en la figura 2.8, el programa de ejemplo de 12,000 bytes (xp2EE0' decimal) comienza en la dirección x'800' (2,048 en decimal) y termina en la dirección x'36DF' (14,047 en decimal).

Las primeras dos secciones miden 4,096 bytes cada una (x'1000'), mientras que la tercera sección mide 3,808 bytes (x'EE0').

La dirección base de la sección (página) número 1 es x'800'. Esta sección mide x'1000' bytes, así es que la dirección base para la segunda sección es x'1800' (o sea, x'800' + x'1000'), por último la dirección base para la última sección será x'2800'.

Como cada página es, a lo más de 4,096 (x'1000') bytes de longitud, cualquier byte dentro de una página puede localizarse sumando a la dirección base un número que esté entre 0 y 4,095 (entre x'0' y x'FFF'). Este número (de 12 bits) es llamado el "Desplazamiento". Dicho de otra forma, cada byte dentro de una página está desplazado de la dirección base de 0 a 4095 posiciones. De esto podemos concluir que una dirección se puede formar de la suma de una dirección base más un desplazamiento.

La figura 2.9 muestra un ejemplo. Supongamos que queremos expresar la dirección x'1941' (6,465 en decimal) en términos de una dirección base y un desplazamiento. El dato se localiza en la segunda página del programa, de la cual su dirección base es la x'1800' (6,144 en decimal). El byte que queremos localizar está desplazado x'141' (321 en decimal) bytes de la dirección base. La dirección del dato es, entonces la suma de la "BASE" (x'1800'en este caso) más un "DESPLAZAMIENTO" (x'141' en este caso) lo que resulta en una dirección absoluta (x'1941').

En una instrucción de máquina, un operando de instrucción mide sólo 16 bits. Obviamente no fue diseñado para guardar tanto la dirección base como al desplazamiento, ya que la sola dirección base mide 24 (o 31) bits. La dirección base no se guarda en la instrucción, sino que se guarda en registros generales: un registro general que se usa para guardar una dirección base se llama "REGISTRO BASE".

Una instrucción que hace referencia a datos almacenados en memoria, no especifica a una dirección base, sino que especifica a un registro que contiene la dirección base, es decir a un registro base.

En la figura 2.10 se muestra un operando de dirección de una instrucción; se compone de dos partes: un campo de 4 bits que designa al registro base y un campo de 12 bits que designa un desplazamiento a partir de la dirección base. El campo de base (Designado como B en la figura) puede tener un valor de x'0' a x'F' (en decimal, de 0 a 15) y el campo de desplazamiento puede tener un valor de x'000' a x'FFF' (En decimal de 0 a 4095).

La computadora genera la instrucción que se desea obteniendo primero la dirección base del registro general designado, al contenido de este le suma el desplazamiento designado en la instrucción y como resultado se obtiene una dirección de 24 bits (o 31 bits si es el caso).

En la figura 2.11 se muestra un ejemplo de una instrucción que suma dos datos empacados. Los datos que se suman se encuentran en secciones de datos distintas: El campo "A" se localiza dentro de los 4096 bytes del registro base 7, de manera que para direccionar a este campo se usa como registro base al registro general 7. El campo "B" cae dentro de los 4096 bytes del registro base 8, de manera que el registro 8 se designa como base.

La carga de los registros que serán usados como registro base se hace al inicio de la ejecución del programa. A este hecho se le llama " Establecer direccionamiento" dentro del programa. Normalmente es lo primero que hace un programa.

2.9 RELOCALIZACION

Cuando se codifica un programa, el programador generalmente considera que su programa comienza en determinada dirección de memoria (generalmente la dirección cero), sin embargo en el momento en el que el sistema operativo carga el programa en memoria, este lo cargará en la localidad de memoria disponible en ese momento.

Este hecho implica que todo programa que se escriba para ser ejecutado en el Sistema/370 debe ser escrito en forma "Relocalizable". Esto quiere decir que si en alguna instrucción dentro del programa se hace referencia a una localidad de memoria dentro del mismo, en el momento en que el programa se carge en otra localidad de memoria, esta referencia debe seguir apuntando al dato que realmente se quiere (figura 1.6).

El método Base-Desplazamiento de direccionamiento permite que los programas sean relocalizables.

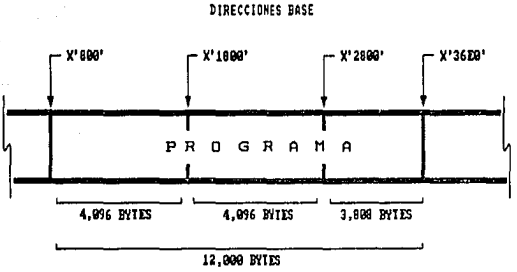


FIGURA 2.8
EJEMPLO DE DIRECCIONES BASE

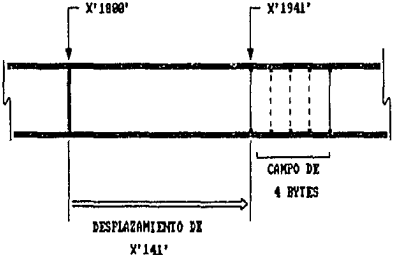


FIGURA 2.9
EJEMPLO DE DIRECCIONAMIENTO BASE-DESPLAZAMIENTO

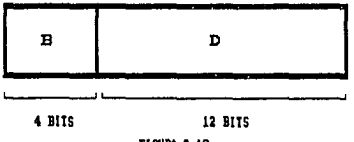


FIGURA 2.10
OPERANDO DE DIRECCION

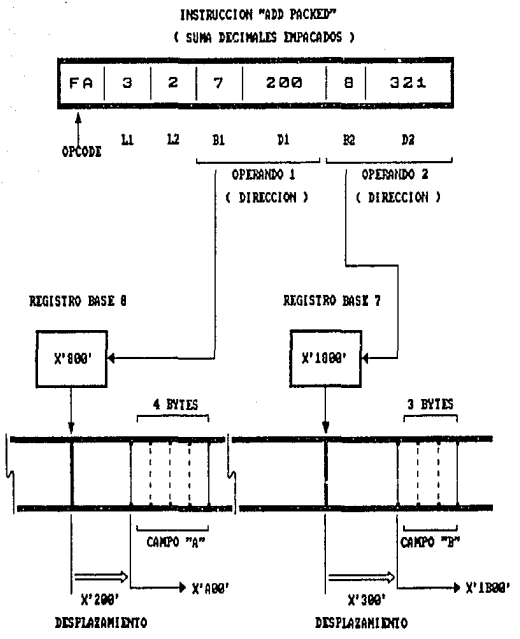


FIGURA 2.11
EJEMPLO DE DIRECCIONAMIENTO BASE DESPLAZAMIENTO
PARA EL CASO DE UNA INSTRUCCION DE SUMA

El procedimiento estandard para establecer direccionamiento en un programa es hacer que una de las primeras instrucciones del programa sea la instrucción "Branch and Link Register" (bifurca y liga al registro) cuyo nemónico en ensamblador es BALR. El formato de la instrucción es como sigue:

$$BALR\ R_1\ ,\ R_2$$

Esta instrucción carga la dirección de la instrucción que siga a la misma en el registro designado por R_1 y hará una bifurcación incondicional a la localidad de memoria contenida en el registro R_2. Si este registro R_2 es cero, entonces no habrá ninguna bifurcación y sólo se guardará en R_1 la dirección de la siguiente instrucción (de hecho se carga el contenido del campo de Instruction Address de la PSW).

Esto se hace para designar al registro R_1 como el registro base de esa sección del programa. No importa en que localidad de memoria se carge el programa, esta instrucción siempre cargará como registro base a una dirección que pertenece al mismo, asegurando que toda referencia relocalizable se haga correctamente.

2.10 INDEXACION

Es frecuente que en programas de aplicación sea necesario accesar cada uno de los elementos de una lista. Por ejemplo, en una forma de búsqueda en una tabla, posiblemente sea necesario comparar cada uno de los elementos de una tabla con un dato dado hasta encontrar uno igual. Normalmente este tipo de problema es manejado codificando una serie de instrucciones llamadas un "Loop" (bucle) que se ejecutan repetidamente. Las instrucciones del "Loop" se codifican como si se hiciera referencia a un sólo elemento de la tabla, pero cada vez que se ejecutan las instrucciones (cada vez que se da una vuelta al "Loop") la dirección que se refiere al elemento de la tabla debe ser modificada para apuntar al siguiente elemento de la tabla. De este modo un sólo juego de instrucciones puede usarse para referenciar a toda la tabla, sin tener que escribir las instrucciones cada vez para cada elemento.

El procedimiento por el cual es posible hacer esto se llama "Indexación" y el Sistema/370 lo proporciona por arquitectura.

En algunas instrucciones (como la instrucción BC que se vio en el ejemplo pasado) además de indicar el registro base y un desplazamiento se le indica a la instrucción qué registro general va a ser usado como índice.

El cálculo de la instrucción efectiva es similar a lo que se había mencionado antes: Al contenido del registro base se le suma un desplazamiento, pero ahora también se le suma el contenido del registro índice. Esto nos permite que sumando un valor al contenido del registro índice en cada vuelta del "loop", se permita accesar todos los elementos de una lista con la misma instrucción.

2.11 LIGAS ENTRE MODULOS

Cuando se "Llama" a una rutina, lo que se hace realmente es hacer un branch o salto a la dirección donde esta rutina comienza (se va a nombrar a la rutina que hace la llamada a otra rutina como la rutina "Que llama", "Caller Routine" y a la rutina que se llamó como la rutina "Llamada","Called Routine"). Cuando la rutina o CSECT llamada recibe el control (es decir, cuando empieza a ejecutarse el código de esta rutina) la misma es responsable de guardar el "ambiente" de la rutina que la llama, este "ambiente" no son otra cosa que el valor que los registros generales tenían en la rutina que llama. Esto se hace "salvando" o guardando estos registros en un área de memoria llamada "Area de Salvado" ("Save Area"), la cual es estandard y sigue ciertas convenciones; estas convenciones son:

1.- El registro general 15 guarda la dirección a la que se va a saltar para darle control, es decir la dirección donde comienza la rutina llamada. La rutina que llama es responsable de cargar el registro 15 con la dirección de la rutina a la que se quiere llamar.

2.- El registro general 14 guarda la dirección dentro de la rutina que llama a la que se va a retornar cuando acabe de ejecutarse la rutina llamada. También es responsabilidad de la rutina que llama establecer esta dirección. Normalmente se retorna el control a una rutina que llama en la instrucción siguiente a la instrucción que hace el branch a la rutina llamada. Esto se puede hacer con la instrucción BALR R14,R15 la cual carga la dirección de la siguiente instrucción en el registro 14 y salta a la dirección que indique el registro 15. Es por esto que esta instrucción es estandard para pasar control a una rutina. Así mismo la instrucción estandard para regresar a la rutina que llama es la instrucción BC 15,R14, es decir, un branch incondicional (valor de máscara igual a 15) a la dirección que indique el registro 14, el cual, como ya se dijo, contiene la dirección de la instrucción de regreso de la rutina que llama.

3.- El registro general 13 contiene la dirección en donde comienza el Area de Salvado en donde se guardan los valores de los registros. Esta Area de Salvado tiene un formato estandard que se muestra en la figura 2.12.

AREA DE SALVADO
(SAVE AREA)

		("BACKWARD")	("FORWARD")
X'0'	SIN USO	APUNTADOR A LA SAVE AREA DE LA CSECT QUE LLAMA	APUNTADOR A LA SIGUIENTE SAVE AREA
X'C'	REGISTRO 14	REGISTRO 15	REGISTRO 16
X'18'	REGISTRO 1	REGISTRO 2	REGISTRO 3
X'24'	REGISTRO 4	REGISTRO 5	REGISTRO 6
X'30'	REGISTRO 7	REGISTRO 8	REGISTRO 9
X'3C'	REGISTRO 10	REGISTRO 11	REGISTRO 12

FIGURA 2.12

FORMATO DE UN AREA DE SALVADO

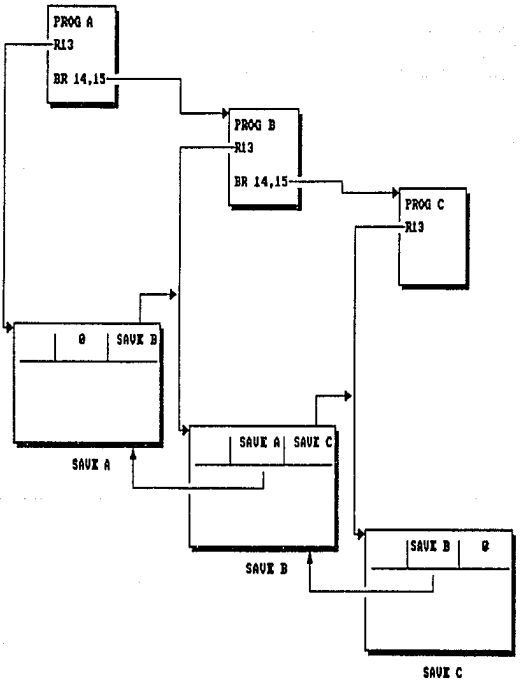


FIGURA 2.13

ENLACES ENTRE LAS AREAS DE SALVADO

Como se observa de la figura 2.12, una Area de Salvado estandard consta de 18 palabras (72 bytes). La primera palabra de la Area de Salvado está reservada para futuro uso, la segunda contiene el apuntador a la Area de Salvado anterior ("Backward pointer") , la siguiente palabra contiene el apuntador al Area de Salvado siguiente ("Forward pointer"), y a partir de la siguiente palabra se guardan los registros del 14 al 12 (Es decir, registro 14, registro 15, registro 0, registro 1, etc. hasta llegar al registro 12).

El objetivo de tener apuntadores a Areas de Salvado anteriores y siguientes se esquematiza en la figura 2.13. En cada traspaso de control de una rutina a otra, la rutina que recibe control es responsable de guardar los registros de la rutina que le dio control.

2.12 MANEJO DE I/O

Todos los datos que procesa el Sistema/370 deben pasar a través de la memoria principal del sistema, las instrucciones de un programa también requieren que los datos sobre los cuales va a procesar están en memoria principal, y los resultados de estas operaciones serán regresados eventualmente a memoria principal.

Sin embargo, no todos los datos que se requieren en un sistema pueden estar al mismo tiempo en memoria principal, sino sólo aquellos que se requieren en determinado, por lo que los datos que no se necesiten deberán guardarse en memoria auxiliar y recuperarse cuando se necesiten. Además eventualmente se tendrá que disponer de estos datos de manera impresa,por ejemplo; o bien tendrán que ser leídos de una terminal para procesarse. Todas estas operaciones se conocen en conjunto como operaciones de "Input/Output" (Entrada/Salida o I/O) y en esta sección se describirá a grandes rasgos como maneja el Sistema/370-XA este tipo de operaciones.

2.12.1 Subsistema de Canal

El Susbsistema de Canal dirige el flujo de información entre los dispositivos de I/O y la memoria principal. Libera a la CPU de tareas tales como comunicarse directamente con los dispositivos de I/O y permite que el proceso de datos se lleve a cabo concurrentemente con el proceso de I/O. El Subsistema de Canal es una "Pequeña" computadora, que una vez que arranca su proceso se hace cargo totalmente de la operación de I/O.

El subsistema de canal se controla por medio de instrucciones, igual que una CPU. Estas instrucciones se denominan "Comandos" para distinguirlos de las instrucciones de CPU; de manera general se denominan Comandos de Canal.

Los comandos de canal residen en memoria, igual que las instrucciones de CPU. Al conjunto de comandos de canal que ejecutan una tarea de I/O se les denomina "Programa de Canal".

Los comandos de canal le dicen al subsistema de Canal cosas como: cuantos datos se van a transmitir, de que localidad de memoria principal va a tomar los datos, o en que localidad los va a poner, que se rebobine una cinta, o que se posicione el mecanismo de acceso de un disco, etc.

Las unidades de control son dispositivos que sirven como intermediarios entre el canal del sistema y los dispositivos de I/O. Proveen la compatibilidad lógica necesaria para operar y controlar un dispositivo de I/O y adapta las características de cada dispositivo a la forma de control estandard que proveé el Subsistema de Canal, dicho de otra forma sirven como una especie de traductores de la señal estandard que reciben los canales a señales que harán al dispositivo en particular hacer el trabajo que se quiere que hagan; Es por esto que existen unidades de control para cada tipo diferente de dispositivo de I/O (figura 2.14).

Al "Camino" por el cual se mueven los datos desde la memoria principal hasta el dispositivo de I/O se conoce como "Camino de Canal" ("Channel Path"). El Sistema/370 permite la definición de más de un "Path" hacia un dispositivo en particular. Esto permite que el sistema operativo sea capaz de balancear la carga de trabajo entre los diferentes "Paths" y además permite que si alguno de ellos presenta una falla, se pueda accesar al dispositivo por otro "Path". En la figura 2.15 se muestra como se pueden definir varios "Paths" hacia un mismo dispositivo.

El subsistema de canal agrupa a conjuntos de hasta 4 "Paths" físicos en una "Unidad de Control Lógica" que usa para fines de balanceo de carga entre los "Paths" que concurren a un dispositivo (figura 2.16(A)). Además asigna a cada dispositivo un identificador único dentro del complejo llamado "Número de Identificación de Subcanal" ("Subchannel ID Number"). Existe un "Subcanal" por cada dispositivo activo dentro del sistema.

Como se muestra en la figura 2.16(B), el Subcanal es simplemente una representación lógica de como ve el subsistema de canales a los dispositivos. Esto es, se tienen varios "Paths" que llegan a una unidad de control y a la vez cada dispositivo puede estar conectado a más de una unidad de control, sin embargo, en la representación por subcanales, no importa cuan complicada pueda estar esta red de intercomunicación: Simplemente cada dispositivo tiene dedicado un "Subcanal" para su uso exclusivo.

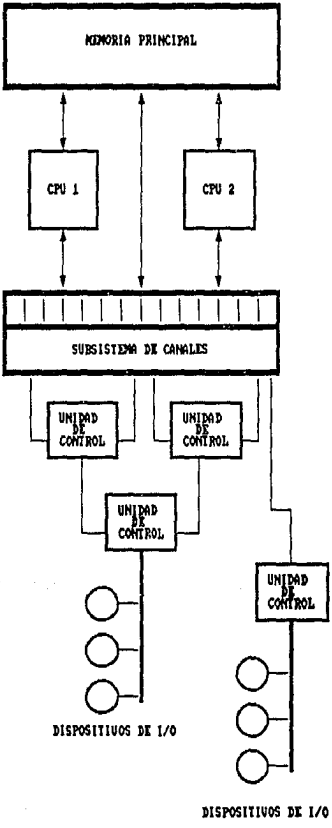


FIGURA 2.14
SUBSISTEMA DE CANALES

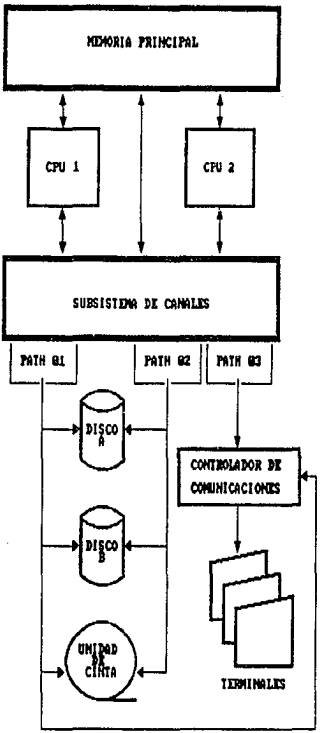


FIGURA 2.15

REPRESENTACION DE DIFERENTES "PATHS" A LOS DIFERENTES DISPOSITIVOS.

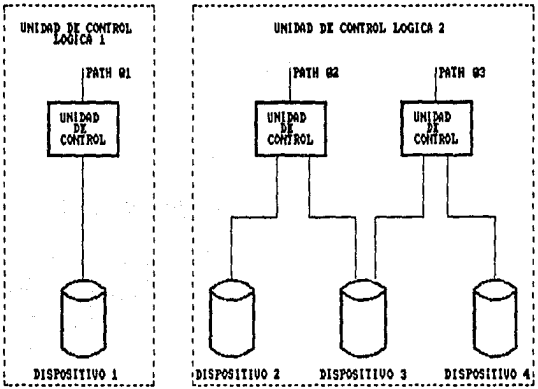


FIGURA 2.16 (A)

EJEMPLO DE ASIGNACION DE UNIDADES DE CONTROL LOGICAS

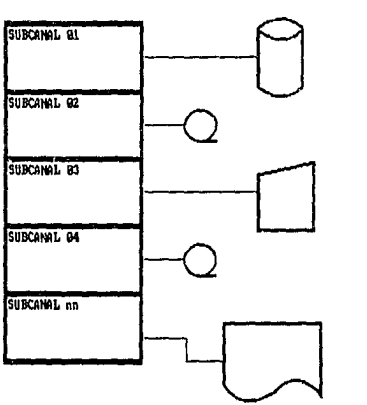


FIGURA 2.16 (B)

EJEMPLO DE ASIGNACION DE SUBCANALES

El número máximo de dispositivos depende del modelo de CPU específico del que se trate, pero se puede decir que son alrededor de 4,000.

Cada comando de canal es una doble palabra llamada la "Palabra de Comando de Canal" ("Channel Command Word",CCW). Una vez que se arranca, el programa de canal realiza la operación de I/O a través de una secuencia de CCW's.

La forma en la que el sistema le indica al Subsistema de Canal que ejecute una operación de I/O es por medio de la instrucción de máquina "Arranca Subcanal" ("Start Subchannel") Nemónico SSCH, Opcode x'B233'. A esta instrucción se le da como operando la dirección de un "Bloque de Requerimiento de Operación" ("Operation Request Block", ORB). En este bloque de información se le comunica al subcanal en donde reside la primera CCW y que formato tienen (existen dos formatos de CCWs).

Una vez que se ejecuta la instrucción SSCH la CPU puede seguir operando mientras que el subcanal comienza a ejecutar en secuencia los comandos de canal (CCWs) de este programa. Los comandos le dicen al canal cuantos datos transferir y de que localidad de memoria tomarlos o en que localidad de memoria poner los datos. El Subcanal es lo suficientemente autónomo para reintentar operaciones fallidas, escoger el "Path" por el que van a pasar los datos, etc. sin notificar de ello a la CPU.

Algunos comandos de canal son "Dependientes del dispositivo", esto es, sólo son significativos para determinado dispositivo en particular, por ejemplo, el comando que provoca que se rebobine una cinta magnética. Estos comandos se pasan a la unidad de control la cual manda al dispositivo la secuencia detallada de señales para que este haga este rebobinado.

Una vez que la operación de I/O se concluye, el canal causa una interrupción de I/O (el concepto de interrupción se explicará más adelante, por ahora sólo se mencionará que una interrupción es el procedimiento por el cual se puede parar un proceso para ser reanudado después) y pasa información acerca del estatus de esta operación. Esta información del estatus de la operación se guarda en una palabra de memoria llamada "Palabra de Estado de Subcanal" ("Subchannel Status Word", SCSW). Esta SCSW proveé información acerca de cómo terminó la operación de I/O.

Cuando la CPU quiere interrogar al subcanal acerca de la operación que acaba de concluir emite la instrucción "Prueba Subcanal" ("Test Subchannel") Nemónico TSCH, Opcode x'B235'.

En el capitulo III en la sección del "Supervisor de I/O" se abundará más en los detalles del proceso de I/O. La ventaja que presenta el proceso concurrente de instrucciones de CPU y operaciones de I/O es que si se cargan varios programas en memoria es que la CPU puede hacer cálculos para un programa mientras que, por ejemplo, el canal esté obteniendo datos para otro programa.

2.13 INTERRUPCIONES

Una interrupción es un evento que altera la secuencia en que el procesador ejecuta las instrucciones; este hecho es provocado por el hardware del sistema. Las interrupciones son además la forma natural por medio de la cual el sistema operativo toma control. Dicho de otra forma, cuando un programador necesita utilizar un servicio que proveé el sistema operativo, no lo hace por medio de una referencia a una rutina especifica del sistema operativo (es decir, una referencia a una etiqueta que corresponda al punto de entrada de una rutina del sistema operativo) sino que lo hace por medio de una instrucción que provoca una interrupción que se denomina "Llamada a supervisor" ("Supervisor Call" o SVC).

Se puede entender fácilmente lo que es una interrupción si se hace una analogía con un evento de la vida cotidiana: Supongase que se está observando una película en videocassette y en determinado momento suena el teléfono. En ese momento es necesario oprimir "Pausa" en la videocassetera para atender la llamada telefónica. Se atiende la llamada telefónica y cuando se sigue viendo la película se asume que esta estará en la misma escena que estaba en el momento en que recibimos la "Interrupción". Este ejemplo servirá para hacer incapié en dos detalles: La interrupción puede ser un evento totalmente ajeno (o asíncrono) al proceso que se está ejecutando en un momento determinado (en este caso, estar viendo la película no causó que se produjera una llamada telefónica), y segundo el hecho de atender (o procesar) la interrupción implica necesariamente que se "Guarde" el estado del proceso que estaba ejecutando para que cuando regrese al él, pueda seguir procesándolo como si no hubiera ocurrido nada.

En el Sistema/370 existen 6 tipos de interrupciones:

-Interrupciones de SVC ("Supervisor Call"): Estas interrupciones son iniciadas por un proceso que ejecuta la instrucción "SVC" (cuyo código de operación es x'0A' y cuyo formato es "SVC xx", en donde xx es el número de "Supervisor Call" que se invoca). Una SVC es una petición generada por el usuario para un servicio particular del sistema, como la obtención de memoria virtual, o la comunicación con el operador del sistema.

El mecanismo de SVC ayuda a mantener al sistema operativo a salvo de los usuarios en general. Los usuarios no pueden accesar directamente el código del sistema operativo, sino que este debe hacer una petición de servicio por medio de una SVC. De esta forma el sistema operativo se puede dar cuenta de los intentos realizados por los usuarios para "Cruzar" sus fronteras y puede negar ciertas peticiones si el usuario no tiene los privilegios adecuados.

- Interrupciones de I/O: Son iniciadas por el hardware de I/O cuando el estado de un subcanal o dispositivo ha cambiado. Por ejemplo, cuando se concluye una operación de I/O previamnete iniciada por la CPU, esta seguramente estará procesando otras tareas; Entonces, para avisar que ya acabó la operación que alguna vez le requirió la CPU al subcanal, este produce una interrupción de I/O.

-Interrupciones externas: Son causadas por varios eventos, incluídos la expiración de un intervalo en el reloj de interrupción, la presión de la tecla de "Interrupción" ("Interruption") en la consola del operador o la recepción de una señal de otra procesador en un complejo de varios procesadores.

-Interrupciones de Reinicio: Ocurren cuando se presiona la tecla de "Reinicio" ("Restart") o cuando llega una instrucción SIGP ("Signal processor") de otro procesador del complejo.

-Interrupciones de "Verificación de programa" ("Program Check"): Son causadas por errores en la ejecución de un programa; por ejemplo tratar de dividir entre cero, intentar ejecutar un código de operación no válido, etc.

-Interrupciones de "Verificación de máquina" ("Machine Check"): Son causadas cuando se detecta un mal funcionamiento del hardware.

Cuando ocurre una interrupción en el sistema se cambia la secuencia de ejecución de las instrucciones, es decir que en lugar de ejecutarse la instrucción que normalmente se ejecutaría después de la que se está ejecutando, se ejecuta otra instrucción diferente. Esto se hace cambiando el contenido de la PSW.

Como se mencionó antes, una parte de la PSW es la llamada "Instruction Address", que es una dirección que apunta a la siguiente instrucción a ejecutar. La PSW que se encuentra "Activa" en cualquier momento de la ejecución se denomina la "PSW actual" ("Current PSW"). Una interrupción implica un cambio en el contenido de la PSW.

Se le llama "Nueva PSW" ("New PSW") al valor de la PSW que ocupará el lugar de la Actual cuando se produce una interrupción; este valor de PSW contendrá un IA que apuntará a la dirección de una rutina que manejará el tipo de interrupción que se trate. Además se mencionó que el concepto de interrupción implica que se guarde el estado anterior para poder retornar a él, por lo cual existe otro tipo de PSW llamada la "PSW vieja" ("Old PSW") que contendrá el valor anterior de la PSW (el valor de la PSW que antes de la interrupción era la PSW Actual).

Cabe aclarar que estos tres tipos de PSWs no se refieren a la existencia de 3 PSWs distintas en el sistema. La PSW es un registro especial de memoria y es único en el sistema. El VALOR que se guarde en este registro es a lo que se le dá el nombre de Actual, Vieja ó Nueva PSW.

Existen valores guardados en memoria que corresponden a la PSW nueva de cada tipo de interrupción mencionada. Estas PSW nuevas apuntan a direcciones que contienen rutinas de sistema operativo que atenderán a cada tipo de interrupción. Además existen también direcciones predeterminadas de memoria en las que se guardarán las PSW viejas para cada tipo de interrupción.

Cuando se produce una interrupción, la PSW Actual (es decir, la PSW que está activa en este momento) se copia en el campo reservado para la PSW vieja de la clase de interrupción sufrida (con esto queda guardado el valor de la PSW que se tenía la momento de la interrupción en la PSW vieja). Entonces se recupera el valor de la PSW nueva de esta clase de interrupción y se copia en la PSW Actual. En este momento, la siguiente instrucción a ejecutar será aquella a la que apunte la PSW que se guardó, la cual apuntará a una rutina que se hará cargo de darle servicio a esta interrupción. Parte de este servicio es guardar los registros generales en un Area de Salvado tal y como se explicó anteriormente; esto se hace con el objetivo de tener todo el "ambiente " anterior guardado para regresar a él cuando se acabe de procesar la interrupción.

El intercambio de PSWs se realiza por el hardware, el software no tiene nada que ver con este "Swap" (que es el término como se le conoce comúnmente) de PSWs.

Los valores de las PSW nuevas se cargan al momento de la "Carga Inicial de Programa" ("Initial Program Load", IPL) que es el momento en el cual se carga desde memoria auxiliar el código del sistema operativo. Como parte de este proceso se cargan, entre otras cosas, las rutinas del sistema operativo que sirven para atender cada una de las 6 interrupciones posibles. Estas rutinas (entre otras) forman parte del llamado "Núcleo" del sistema operativo.

Cuando se concluye la atención de la interrupción en cuestión se puede regresar a hacer exactamente lo mismo que se estaba ejecutando regresando el valor de la PSW Actual al guardado en la PSW vieja y restaurando el valor de los registros generales. Una vez hecho esto, el programa puede seguir ejecutando, como si la interrupción nunca hubiera ocurrido.

Las PSW Vieja y la PSW Nueva se guardan en localidades fijas de memoria, en memoria baja (es decir, dentro de la primera página de memoria que comprende las direcciones x'000000' a x'001000') . En la figura 2.17 se muestran las localidad es de memoria asignadas para las Vieja y PSW nueva's de cada tipo de interrupción (excepto de interrupciones de Reinicio que no tienen PSW vieja ni PSW nueva).

LOCALIDAD (Hexadecimal)	Contenido
18-1F	PSW Vieja para Interrupciones Externas
20-27	PSW Vieja para Interrupciones de SVC
28-2F	PSW Vieja para Interrupciones de Verificacion de Programa
30-37	PSW Vieja para Interrupciones de Verificación de Máquina
38-3F	PSW Vieja para Interrupcionde de I/O
58-5F	PSW Nueva para Interrupciones Externas
60-67	PSW Nueva para Interrupciones de SVC
68-6F	PSW Nueva para Interrupciones de Verificacion de Programa
70-77	PSW Nueva para Interrupciones de Verificación de Máquina
78-7F	PSW Nueva para Interrupcionde de I/O

Figura 2.17
Asignación de localidades de memoria fijas para las PSWs nuevas y viejas de cada tipo de interrupción.

En la figura 2.18 se ejemplifica el proceso de una interrupción, en este caso se tomará como ejemplo una interrupción de I/O, aunque de hecho el proceso es muy similar para cualquier clase de interrupción.

Se está ejecutando un programa llamado "MIPROG", específicamente una instrucción ADD . En este momento (figura 2.18(A)) la PSW está apuntando a la siguiente instrucción a ejecutarse; una instrucción CMP ("Compare logical"). Supongase que en este preciso momento se sufre una interrupción de I/O. Como se muestra en la figura 2.17, la PSW nueva y la PSW vieja ocupan localidades predeterminadas de memoria baja.

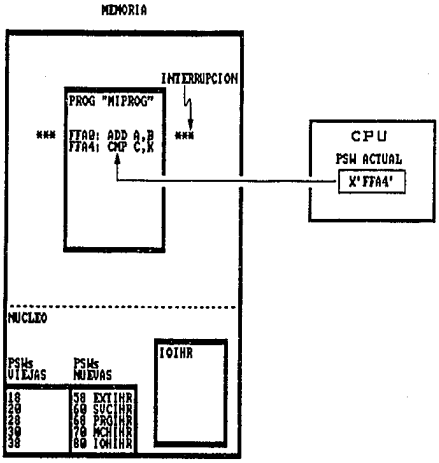


FIGURA 2.18 (A)

EJEMPLO DE UNA INTERRUPCION: LA PSW ACTUAL APUNTA A LA SIGUIENTE INSTRUCCION A EJECUTARSE.

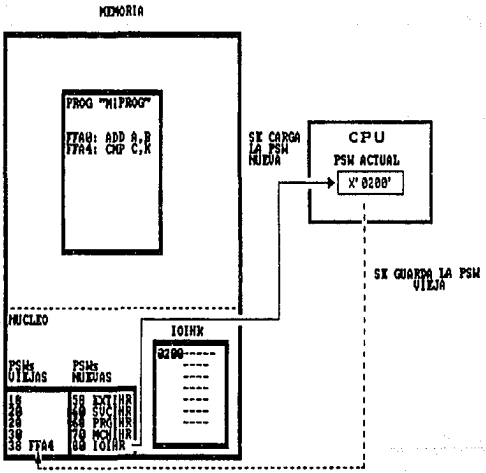


FIGURA 2.18 (B)

EJEMPLO DE UNA INTERRUPCION: EN EL MOMENTO DE LA INTERRUPCION EL HARDWARE REALIZA EL "SWAP" DE PSWs

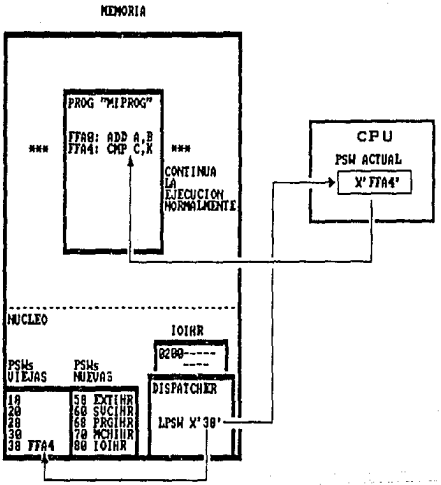


FIGURA 2.18 (C)

EJEMPLO DE UNA INTERRUPCION: CUANDO SE REANUDA LA EJECUCION EN EL PUNTO DONDE SE SUFRIO LA INTERRUPCION, SE CARGA LA PSW VIEJA A LA PSW ACTUAL

Cada PSW nueva apunta a "Rutinas manejadoras de interrupciones" ("Interruption Handler Routine") ó IHR las cuales tienen un nombre particular; por ejemplo, la IHR para interrupciones de I/O recibe el nombre de IOIHR, la de "Machine Check" se llama MCIHR, la de "External" se llama EXTIHR, la de SVC se llama SVCIHR la de "Program Check" se llama ORGIHR, y la de "Restart" se llama RSTIHR.

En el momento de la interrupción (figura 2.18(B)), el hardware guarda el contenido de la PSW Actual en la localidad x'38' (que corresponde a la localidad preasignada para la PSW vieja de I/O) y copia el contenido de la localidad x'78' (que contiene a la PSW nueva de I/O) a la PSW actual. Hecho esto, como ahora el IA de la current PSW apunta a la rutina manejadora de interrupciones IOIHR, la siguiente instrucción ejecutada será precisamente la primera instrucción de esta IOIHR. Seguramente esta rutina lo primero que hará será salvar los registros generales en un Area de Salvado determinada.

En algún momento, cuando esta rutina acabe su ejecución, alguna otra rutina del sistema operativo se encargará de reestablecer la PSW Actual con la PSW vieja y los registros generales (que se guardaron en el momento de la interrupción) para seguir ejecutando lo que estaba haciendo el programa "MIPROG" (en este caso ejecutará la instrucción CMP) como si nada hubiera sucedido (figura 2.18(C)).

El reestablecimiento de la PSW Actual al valor de la PSW vieja se hace por medio de una instrucción llamada "Carga PSW" ("Load PSW", LPSW). El operando de esta instrucción es una localidad de memoria (que en este caso será la localidad x'38' que es donde reside la PSW vieja de la interrupción de I/O) y su función es cargar la PSW Actual con el valor que encuentre en la doble palabra que inicie en la dirección indicada.

Esta instrucción generalmente se ejecuta por el "Despachador" ("Dispatcher") del sistema operativo. Esta rutina siempre recibe control después de manejar una interrupción y es la que decide cuál es el programa al cual se le va a dar control a continuación. Si el dispatcher decide que el programa "MIPROG" es el siguiente, ejecutará la instrucción LPSW con el operando de la PSW vieja de I/O.

Esta instrucción LPSW es una de las instrucciones llamadas "Privilegiadas", debido a que esta y otras instrucciones sólo pueden ser ejecutadas por rutinas del sistema operativo.

2.13.1 Causas de interrupción

Aún cuando sólo existen seis categorias "Mayores" de eventos que causan interrupciones, existen clases y subclases de interrupciones. Por ejemlplo, una interrupción de programa puede ser causada por varias docenas de eventos distintos; el número de interrupciones de I/O depende del número de dispositivos de I/O y pueden ser causadas por varios eventos ocurriendo en cualquier subcanal; de igual forma los eventos que pueden causar una interrupción de verificación de máquina son numerosos, de la misma forma que los eventos clasificados como externos.

Cada evento generador de interrupción produce un patrón de bits específico llamado el "Código de interrupción". Estos códigos se usan en las rutinas de manejo de interrupciones ("Interrupt Handler Routines", IHR) para determinar la naturaleza exacta de la interrupción que se genera y darle el tratamiento adecuado. Este código de interrupción se guarda en la localidad x'84', x'88' ó x'8C' para interrupciones externas, de SVC o de programa respectivamente.

2.13.2 Retardo del efecto de Interrupciones

Supongase que mientras se está procesando una interrupción de I/O, por ejemplo, se genera otra interrupción de I/O diferente. Si a la segunda interrupción se le atendiera, olvidándose de la anterior, nunca se podría acabar de atender las interrupciones si estas se suceden una tras otra. Para evitar esto, existe un mecanismo mediante el cual es posible "Deshabilitar" a las interrupciones; esto es, que mientras se procese una interrupción, se deshabilitan (es decir, se impide que sucedan) interrupciones del mismo tipo de la que se está procesando. Cuando la CPU desconoce algún tipo de interrupciones se dice que está deshabilitada ("Disabled") para este tipo de interrupciones.

Cuando la CPU está deshabilitada para interrupciones de I/O (lo cual significa que está atendiendo una interrupción de I/O), los eventos que causan interrupciones de I/O crean una señal constante. Esta señal es ignorada (permanece pendiente) hasta que la CPU está de nuevo habilitada ("Enabled") para interrupciones de I/O. Cuando la CPU está de nuevo "Enabled" para interrupciones de I/O, la señal pendiente es reconocida y la interrupción se lleva a cabo.

Cuando la CPU está deshabilitada para otro tipo de interrupciones, como verificación de máquina, verificación de programa ó externas, la condición que las causó puede o no permanecer pendiente. Esto depende de la clase y subclase del evento que haya causado la interrupción. Si esta no permanece pendiente, se pierde para siempre.

2.13.3 Enmascaramiento

Para asegurarse de que las interrupciones quedan temporalmente (o en su caso permanentemente) ignoradas tan pronto como sea posible, se utilizan algunos bits de la PSW (los cuales estarán "prendidos" tan pronto como se efectúe el "Swap" de PSW's) y los Registros de Control.

Los bits 6,7 y 13 de la PSW Actual (figura 2.7) son los bits maestros de control de ciertas interrupciones generales: De interrupciones de I/O, Externas, y de Verificación de Máquina respectivamente. Para el caso de las interrupciones de I/O, por ejemplo, existen varias subclases de interrupciones controladas por bits del registro de control 6.

Los patrones de bits en la PSW y los registros de control que permiten o previenen interrupciones son llamadas "Máscaras" ("Masks") y su uso se denomina "Enmascaramiento" ("Masking").

Se deshabilitan las interrupciones poniendo en cero (x'0') los bits apropiados en la PSW y los registros de control. Se habilitan las interrupciones poniendo con valor de uno (x'1') estos bits.

2.13.4 Cambio de máscara

Existen ocasiones en las que durante la ejecución de una rutina de manejo de interrupciones se hace necesario habilitar a la CPU para ciertas interrupciones, es decir, cambiar el patrón de bits de la PSW y de los registros de control. Por ejemplo, si una interrupción de I/O se genera por un error de lectura, se debe habilitar a la CPU para interrupciones de I/O en el reintento de leer el registro dañado.

La instrucción para cambiar la mascara maestra en la PSW es la instrucción "Set System Mask" (Nemónico SSM, Opcode x'80'). La instrucción que se usa para cambiar la mascara de un registro de control se llama " Load Control " (Nemónico LCTL, Opcode x'B7'). Los operandos de estas instrucciones dan el valor de la nueva máscara.

2.14 INSTRUCCIONES PRIVILEGIADAS

Las instrucciones para cargar la PSW, cambiar la mascara de la Actual PSW, cargar los registros generales, y comenzar operaciones de I/O son llamadas instrucciones privilegiadas. Esto significa que sólo ciertas rutinas de supervisor (sistema operativo) pueden usarlas.

2.15 ESTADOS DE LA CPU

Para prevenir que el programador de aplicaciones ejecute instrucciones privilegiadas, se hace que mientras se ejecute el programa de usuario la CPU esté en "Estado Problema" ("Problem State"), lo cual se hace cuando una PSW se convierte en la Actual PSW. El estado problema está determinado por el bit 15 de la PSW: Si este bit tiene un valor de 1, entonces la CPU se dice que está en estado problema. Si el bit 15 de la PSW está en cero se dice que la CPU está en "Estado Supervisor" ("Supervisor State"), estado en el que se pueden ejecutar instrucciones privilegiadas (figura 2.7).

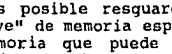
Cuando un programa corriendo en estado problema intenta ejecutar una instrucción privilegiada, se origina una Interrupción de Verificación de Programa no enmascarable. Cuando la rutina manejadora de estas interrupciones detecta que se intentó ejecutar una instrucción privilegiada, es su responsabilidad terminar de inmediato la ejecución de este programa.

2.16 PROTECCION DE MEMORIA

Otra área de interés en ambientes de multiprogramación es la referencia no autorizada o accidental de datos o instrucciones entre los diferentes programas que se estén ejecutando concurrentemente en una CPU. Aún cuando se suponga que todos los programas tengan "Derecho" a accesar toda la memoria, es necesario un mecanismo de protección contra una posible destrucción de datos accidental. Por ejemplo, una mala combinación de instrucciones indexadas puede generar una dirección que ocupe una instrucción de otro programa.

En el Sistema/370 es posible resguardar ciertas partes de memoria con una "Llave" de memoria especial ("Storage Key"). Cada sección de memoria que puede ser protegida contra referencias no autorizadas mide 4096 bytes (es decir, una página). Existen 16 llaves de memoria, cada llave determinada por un patrón de 4 bits. El valor de la llave "Actual" se encuentra en los bits 8 a 11 de la "PSW Actual".

La llave es un campo de 6 bits que se asocia a cada página de 4 KBytes de memoria real y se compone de los siguientes campos:



El campo ACC es el valor de la llave en sí de 4 bits, lo que da un total de 16 valores de llave distintos.

El bit F es el bit de protección contra "Fetch", es decir contra acceso. Si este bit está prendido, sólo que la llave coincida con la de la PSW se puede hacer acceso o guardar, si no coincide no se puede accesar.

El bit R es el bit de referencia, se prende cada vez que se accesa una localidad en la página correspondiente.

El bit C es el bit de cambio, se prende cada vez que se cambia o se guarda un dato en la página correspondiente.

La instrucción que sirve para proteger una sección de memoria se llama "Set Storage Key" (Nemónico SSKE, Opcode B22B). Esta es una instrucción privilegiada, por lo cual sólo se puede ejecutar en estado supervisor.

Cuando una página de memoria se protege por un valor de llave determinado, es imposible guardar o extraer cualquier dato de esta página a menos que se tenga el mismo valor de llave en la PSW.

Una rutina del sistema operativo determina el valor de llave que tiene permitido poseer determinado usuario y se encarga de poner este valor en la que será su PSW Actual cuando el sistema operativo le ceda el control. El usuario que tenga esta llave será capaz de guardar o leer datos de páginas que tengan su misma llave.

2.16.1 Llave Maestra

Muchas de las rutinas de supervisor necesitan procesar datos que se encuentran en las áreas que corresponden a programas problema. Para evitar que el supervisor cambie constantemente de valor de llave, este corre con llave cero, que es el valor de llave maestra.

Con el valor de llave cero, el supervisor puede accesar ("Fetch") o guardar datos en cualquier localidad de memoria.

Existe además otro mecanismo de protección llamado "Protección de memoria baja" que proveé protección contra la destrucción de información residente en memoria real usada por la CPU para procesar interrupciones. Esto se lleva a cabo prohibiendo a las instrucciones grabar en las localidades de la x'00000000' a la x'000001FF'.

La protección de memoria baja se controla con el bit 3 del registro de control 0. Cuando el bit es cero, las direcciones de memoria baja pueden ser accesadas y modificadas, cuando este bit es 1, estas localidades se encuentran protegidas contra escritura.

Si se intenta escribir en estas localidades de memoria con el bit de protección prendido, se produce una interrupción de "Program check", con un código de interrupción x'04' llamado "Excepción de protección".

2.17 MEMORIA VIRTUAL

A lo largo del desarrollo de los sistemas computacionales, los requerimientos de memoria se han hecho cada vez más grandes; mientras más memoria se tiene, se hacen programas cada vez más grandes.

El problema con los programas gigantes no es el hecho de codificarlos (esto consume sólo horas-hombre de trabajo), ni siquiera el hecho de compilarlos; el problema surge cuando un programa de miles de millones de direcciones será ejecutado en memoria como un sólo módulo de carga, para lo cual debe existir un byte direccionable en memoria por cada byte que ocupe el programa. Si no existe un byte en memoria principal para cada byte referenciado en el programa, entonces ocurrirá una "Excepción de Direccionamiento" ("Address Exception").

Una forma común de evitar esto es subdividir el programa en dos o más piezas y ejecutar estas en secuencia. Sin embargo esta implantación produce efectos negativos en el rendimiento y tiempo de respuesta del sistema.

Otra forma de resolver este problema es tener muchos módulos relativamente pequeños y un módulo "Analizador de transacciones". Este analizador de transacciones determina que módulos se necesitan y hace el requerimiento de carga de estos módulos. Aún cuando este método de cargar dinámicamente módulos en memoria justo antes de usarlos es superior en rendimiento a la técnica de proceso secuencial mencionada anteriormente, consume mucho tiempo CPU.

Una forma de solucionar este problema, que es la que ahora se abordará con detalle, requiere de circuitería de Hardware adicional: Se trata del concepto de "Memoria Virtual", mediante la cual es posible asignar direcciones de hasta 2 Gbytes de memoria teniendo un tamaño de memoria real bastanye menor que esta cantidad.

Lo primero que se debe analizar es la necesidad real de tener todo el programa a la vez en memoria: Realmente la CPU sólo puede procesar UNA sola instrucción a la vez, por lo tanto se puede llegar a la conclusión de que no es estrictamente necesario tener todo el código ejecutable en memoria, sino sólo aquella parte del código que se va a procesar en un momento determinado.

Esto implica que en el inicio de la ejecución del programa, no se cargará la totalidad de este a memoria real, sino que se guardará en memoria auxiliar (es decir, en un disco) y sólo se traerá a memoria real la página en donde se encuentre el código que se necesite ejecutar.

Considérese el siguiente ejemplo (figura 2.19): Se comenzará a ejecutar el programa A, para lo cual se copia la primera página del programa A (página A-0) en la memoria real página 0 y se comienza a ejecutar. En un momento determinado se requiere una operación de I/O y se cae en un estado de espera; El código para atender el requerimiento de una operación de I/O se encuentra en el programa B, por lo tanto la primera página del programa B (página B-0) se carga en la página real 1 y comienza. Supongase ahora que una vez arrancado el requerimiento de I/O se sufre una interrupción de I/O de otro requerimiento que ya acabó. El código del manejador de interrupciones se encuentra en el programa C, por lo cual se trae a memoria real la primera página del programa C (página C-0) el cual comienza a ejecutar y termina de atender la interrupción de I/O. Cuando acaba de atenderla se le da el control al programa A en la página A-0 , y eventualmente necesitará ejecutar la siguiente instrucción que reside en la página A-1, la cual se carga en la página real 3. Como se ve, eventualmente se llegará al punto presentado en la figura 2.19(B) , en donde las cuatro páginas reales estarán llenas.

Supóngase que se está ejecutando el código de la página A-1 y en algún momento se hace otro requerimiento de I/O para lo cual tiene que tomar control el programa B, en la página B-1, la cual no está en memoria principal. La CPU no puede tener acceso ni ejecutar instrucciones que no se encuentren en memoria real.

Lo que se hace en este momento es desalojar de memoria real la página que fue referenciada hace más tiempo, esto es, la página A-0. Por lo tanto se mueve la página A-0 a DASD y se carga la página B-1 a la página real desalojada (figura 2.19(D)). Una vez hecho esto se prosigue con las instrucciones de esta página.

Como se observa en las figuras, para el programador, todas las páginas siempre estuvieron en memoria en un lugar fijo de ella. Esta memoria que ve el programador (que en el ejemplo mide 16 páginas) realmente no está disponible en la máquina, esta sólo cuenta con 4 páginas de memoria. Este tipo de memoria se denomina "Memoria Virtual" y se refiere al hecho de contar y "ver" más memoria de la que realmente tiene el sistema. En términos reales, esta memoria no existe como tal (no hay en ningún lado un segmento de memoria contigua que vaya de la dirección x'00000' a la dirección x'10000' que corresponde a 16 páginas) por eso se le da el nombre de memoria "Virtual".

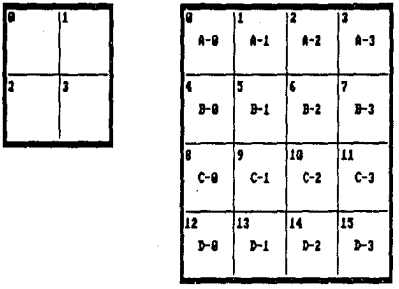


FIGURA 2.19 (A)
ASIGNACION INICIAL DE PAGINAS EN MEMORIA VIRTUAL

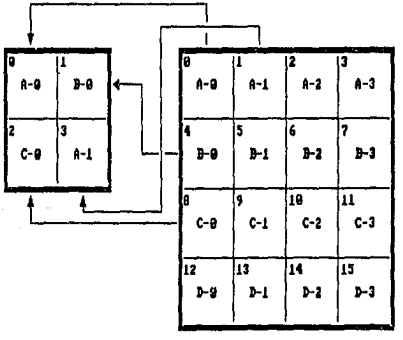


FIGURA 2.19 (B)
ASIGNACION DE "FRAMES" DE MEMORIA REAL A PAGINAS DE MEMORIA VIRTUAL

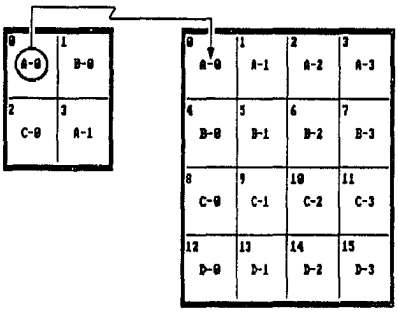


FIGURA 2.19 (C)
SE SACA DE MEMORIA REAL LA PAGINA MENOS ACCESADA

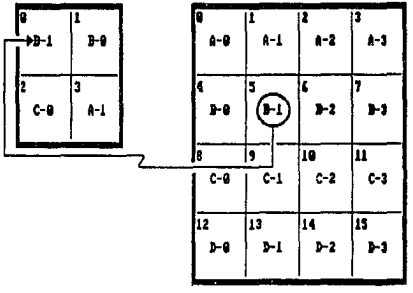


FIGURA 2.19 (D)
SE SUPLANTA ESTA PAGINA POR OTRA QUE SE NECESITE EN ESTE MOMENTO

Sin embargo, el hecho de que esta memoria virtual no exista como memoria física en el sistema no implica que no exista en ningún otro lado; esta memoria sí existe (es decir, los datos están almacenados) tanto en memoria real como en memoria auxiliar. El concepto de memoria virtual es sólo un "Mapeo" entre estos dos tipos de memoria para dar la impresión de que se cuenta con más memoria de la que se tiene físicamente.

Como se pudo constatar en el ejemplo, aún cuando las direcciones asignadas a las páginas de los diferentes programas (A,B,C y D) estaban fijas en todo momento en memoria virtual, cuando se cargaban las respectivas páginas en memoria real, esto no es estrictamente cierto: Cada página que se carga en memoria real se cargará en una página que esté libre y no necesariamente en la misma posición que ocupa en memoria virtual. Esto significa que:

1) Las páginas que están contiguas en memoria virtual no necesariamente están contiguas en memoria real.
2) No existe ninguna relación posicional entre las páginas de memoria virtual con las páginas de memoria real. Es decir, si la página X en memoria virtual tiene direcciones más bajas que la página Y, cuando estén cargadas en memoria real puede ser que la página Y tenga direcciones más bajas que la página X.

Cuando la CPU ejecuta las instrucciones, esta no "Entiende" a las direcciones virtuales, sólo puede ejecutar instrucciones y leer datos de direcciones reales, es decir , direcciones que físicamente existan en su circuitería de memoria. Para poder llevar a cabo el "Mapeo" del que se habló anteriormente entre memoria virtual (que es la que ve el programador) y memoria real (que es la que ve la CPU) el Sistema/370 proporciona un mecanismo de Hardware que nos permite hacer la traducción de una dirección virtual a una dirección real; Este mecanismo se le conoce como "Traducción Dinámica de Direcciones" ("Dynamic Address Translation", DAT).

2.17.1 Traducción Dinámica de Direcciones

El DAT junto con el soporte apropiado de un sistema operativo puede usarse para proveer al usuario de un sistema en donde la memoria aparenta ser mucho más grande que la memoria real que posea. Esta memoria que aparenta ser más grande que la memoria real es la que anteriormente se nombró como "Memoria Virtual" y las direcciones usadas para designar localidades en memoria virtual se llaman "Direcciones Virtuales". La memoria virtual puede exceder por mucho el tamaño de la memoria real disponible en el sistema y normalmente es mantenida en memoria auxiliar, como se explicó anteriormente.

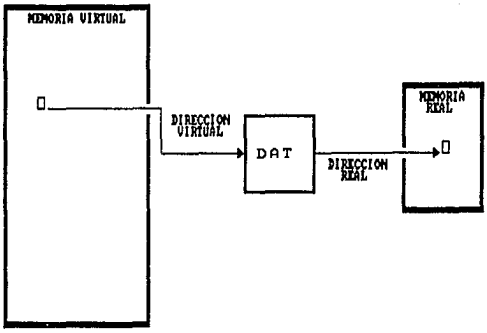
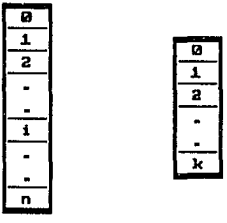


FIGURA 2.20

TRADUCCION DINAMICA DE DIRECCIONES



ESPACIO DE NOMBRES (N)

ESPACIO DE ALMACENAMIENTO (S)

FIGURA 2.21

RELACION ENTRE EL ESPACIO DE NOMBRES Y EL ESPACIO DE DIRECCIONES

La memoria virtual está dividida en bloques llamados "Páginas". Solo las páginas más recientemente referenciadas de memoria virtual son mantenidas en memoria real. Tan pronto como el usuario necesite accesar datos o instrucciones que residan en una página de memoria virtual que no esté en memoria real, esta se trae desde memoria auxiliar y reemplaza a la página que menos se espera que sea necesitada. El intercambio de páginas entre memoria principal y auxiliar lo lleva a cabo el sistema operativo sin el conocimiento del usuario.

Como se dijo anteriormente, la memoria virtual está dividida en "Páginas" de 4K bytes; cuando esta página reside en memoria virtual se le denomina "Frame", y cuando esta reside en memoria auxiliar se le denomina "Slot".

La secuencia de direcciones virtuales asociadas a una memoria virtual es llamada el "Espacio de Direcciones" ("Address Space"). Con el soporte apropiado de un sistema operativo, la facilidad del DAT puede usarse para proveer una cierta cantidad de Espacios de Direcciones. Estos Espacios de Direcciones proporcionan cierto grado de aislamiento entre usuarios. Este soporte puede consistir en un Espacio de Direcciones para cada usuario (como en el caso del MVS) proporcionando así completo aislamiento entre usuarios; y sin embargo permitiendo le existencia de un área compartida, mapeando una porción de cada Espacio de Direcciones a un área común. El Sistema/370 cuenta también con una serie de instrucciones privilegiadas que permiten accesar más de un Espacio de direcciones a la vez, lo cual permite la comunicación entre estos.

La traducción dinámica de direcciones (DAT) es el proceso por el cual se traduce una dirección virtual a su correspondiente dirección real, durante una referencia a memoria. Se puede pensar en el DAT como una "Caja Negra" a la cual le alimentamos una dirección virtual y nos devuelve una dirección real. (figura 2.20).

También podemos pensar en las direcciones virtuales como el "Nombre" que le damos a los bytes de memoria y que el conjunto de direcciones es un "Espacio de nombres", mientras que las direcciones reales en donde se guardan estos bytes a los cuales dimos un "Nombre", representan en conjunto un "Espacio de Almacenamiento" (figura 2.21), y que existe una función (f N -> S) que se aplica a cada argumento en el espacio de nombres N para llegar a la localidad que le corresponde en el espacio S; además debido a que en determinado momento, no todos los elementos de N tienen en memoria real un elemento S, la función es indeterminada para ciertos argumentos de N.

En este contexto, podemos pensar en el DAT como la "Caja" que efectúa la función *f*.

2.17.2 Proceso de Traducción

El DAT no realiza ninguna operación matemática sobre las direcciones virtuales para obtener una dirección real. El proceso que realiza el DAT es una búsqueda en dos tablas: La tabla de Segmentos ("Segment Table") y la tabla de páginas ("Page Table").

Un "Segmento" es un bloque de direcciones virtuales secuenciales que mide 1M byte y que siempre comienza en frontera de 1M. Y una "Página" , como ya se dijo, es un bloque de 4K bytes que siempre comienza en frontera de página.

Una dirección virtual consta de 24 bits o de 31 bits (en el Sistema/370 arquitectura extendida). Con 24 bits se pueden direccionar hasta 16 Mbytes de memoria (es decir, 2**24 = 16777216 bytes = 16 Mbytes), en cambio con 31 bits en arquitectura extendida (XA) se pueden direccionar hasta 2 Gbytes de memoria (2**31 = 2147483648 bytes = 2 Gbytes).

Las direcciones se representan en Sistema/370 como una palabra ("Fullword") de 4 bytes, es decir 32 bits. Si se trabaja en modo 370 sólo se utilizan los 24 bits más a la derecha de la dirección, dejando los 8 bits más significativos sin usar.

En la PSW se indica si se está direccionando con 24 o 31 bits. El bit 32 de la PSW es la bandera que indica el modo de direccionamiento. Si se están usando 31 bits (modo XA) el bit 32 estará prendido, mientras que si se está direccionando con 24 bits (modo 370), este bit estará apagado.

La dirección virtual se divide en tres campos: los bits 1-11 son llamados el "Indice del segmento" ("Segment Index", SX), los bits 12-19 son llamados el "Indice de página" ("Page Index", PX) y los bits 20-31 son llamados el "Indice de byte" ("Byte Index", BX), como se muestra en la figura 2.22.

Por ejemplo, en la dirección:

x'0727ED67'

SX = 072
PX = 7E
BX = D67

La "Tabla de Segmentos" ("Segment Table", SGT) es una tabla que tiene una entrada para cada segmento de memoria virtual, y cada entrada ("Segment Table Entry", SGTE) contiene un apuntador a una Tabla de Páginas. Para direccionar 2 Gbytes

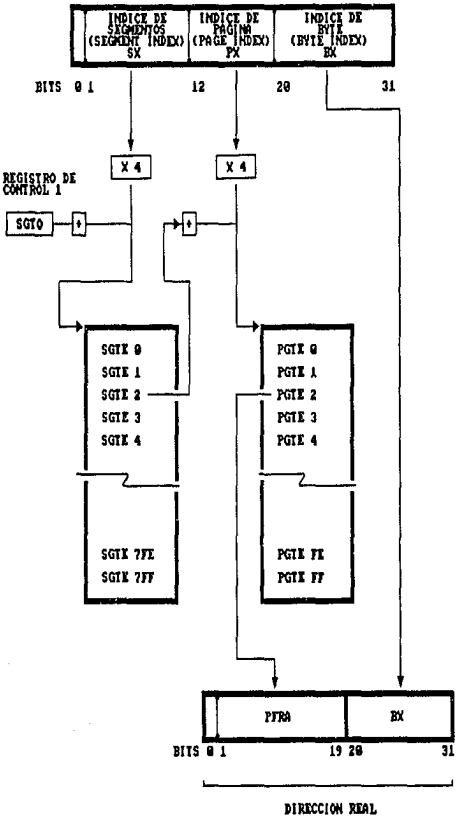


FIGURA 2.22

MECANISMO DE TRADUCCION DINAMICA DE DIRECCIONES

de memoria se necesitan x'7FF'(2,048) entradas , es decir que 2 Gbytes contienen 2048 segmentos. Existe una SGT por cada Espacio de Direcciones que se crea en el sistema.

La "Tabla de Páginas" ("Page Table", PGT) es una tabla que tiene una entrada para cada página del segmento que representa. Contiene x'FF (255) entradas ("Page table Entry", PGTE) cada una de las cuales contiene la dirección en memoria real en donde comienza la página que está representando, si es que esta se encuentra en memoria real.

El registro de control 1 contiene la dirección del origen de la SGT, dirección llamada "Origen de la Tabla de Segmentos" ("Segment Table Origin", SGTO).

Cada entrada en la SGTE tiene una longitud de 4 bytes y contiene un campo que representa la dirección real de comienzo de una Tabla de Páginas, un bit que indica si el segmento asociado con esta entrada está disponible en memoria real. Si este bit es cero, la traducción dinámica continua. Cuando este bit es uno, la SGTE no puede ser usada para la traducción y se produce una interrupción de "Program Check" con código x'10', la cual se llama "Excepción de Traducción de Segmento" ("Segment translation exception"). Además tiene un campo que indica la longitud de la Tabla de Páginas a la que apunta.

Cada entrada de la Tabla de Páginas (PGTE) tiene una longitud de 4 bytes e incluye la "Dirección Real del Frame de Página" ("Page-Frame Real Address", PFRA) que es la dirección en memoria real del inicio de la página que representa esta entrada, un bit que indica si la página asociada a esta entrada está disponible en memoria real. Si este bit es cero, la traducción dinámica continua. Cuando este bit es uno, la PGTE no puede ser usada para la traducción y se produce una interrupción de "Program Check" con código x'11', la cual se llama "Excepción de Traducción de Página" ("Page translation exception").

Además tiene un bit de "Protección de Página" que controla el acceso a la página para guardar datos en la página que representa la entrada. Esta protección es adicional a la protección por llave de memoria y a la protección de memoria baja. Este bit no tiene efecto en los accesos a escritura. Si este bit es cero se permiten los accesos para guardar datos en esta página. Si es uno y se intenta guardar un dato en esta página, se provoca una interrupción de "Program Check" con código x'04' conocida como "Excepción de Protección" ("Protection exception").

Para convertir una dirección virtual a una real se suma al contenido del registro de control 1 (que representa la SGTO) el valor del SX multiplicado por 4, lo cual genera una dirección que corresponde a una entrada dentro de la SGT que

contiene la dirección en memoria real del inicio de una Page table. A esta dirección se le suma el PX multiplicado por 4 lo cual genera una dirección en memoria real que corresponde una entrada dentro de la Page table que contiene la dirección en memoria real del inicio (Byte x'0') de una página. Para encontrar la dirección del byte de memoria virtual que se quiere accesar en memoria real, se concatenan el contenido del campo FPRA de la PGTE y el BX, de tal forma que los bits 1 a 19 de la FPRA corresponden a los bits 1 a 19 de la dirección real y los bits 20 a 31 de la dirección virtual (que corresponden al Byte Index, BX) corresponden a los bits 20 a 31 de la dirección real.(figura 2.22)

Nota: El mecanismo de DAT está interconstruido en el hardware del Sistema/370, no es software, sin embargo utiliza las tablas mencionadas que son mantenidas por software. Es responsabilidad del sistema operativo mantener estas tablas, ya que el DAT sólo hace uso de ellas para la traducción. De igual forma cuando se origina una interrupción de "Program Check" por que el segmento o la página es inválido, la responsabilidad de traer de memoria auxiliar la página o el segmento faltante y de actualizar las tablas es el sistema operativo.

2.17.3 Comunicación entre Espacios de Direcciones

El que determina que tabla de segmentos se va usar y por lo tanto que Espacio de Direcciones se va a accesar es el registro de control 1 que contiene al "Origen de la Tabla de Segmentos Primaria" ("Primary Segment Table Origin", PSGTO). Esto quiere decir que si se cambia el contenido del registro de control 1, se estará direccionando a otro Espacio de Direcciones distinto.

A la PSGTO se le denomina "Primaria", porque existen instrucciones que permiten a a dos Espacios de Direcciones interactuar al mismo tiempo (es decir, en la misma instrucción). Al Espacio de Direcciones en donde se está ejecutando la instrucción se le denomina primario ("Primary") y al otro Espacio de Direcciones se le denomina Secundario ("Secondary"). Estas instrucciones son: "Move Character to Primary" y "Move Character to Secondary". El registro de control 7 es el que guarda el origen de la tabla de segmentos secundaria ("Secondary Segment Table Origin", SSGTO).

La instrucción "Mueve caracteres al primario" ("Move Character to Primary"), Nemónico MVCP, Opcode x'DA', tiene el siguiente formato:

$$\text{MVCP } D_1(R_1,B_1),D_2(B_2),R_3$$

En donde el segundo operando, localizado en la dirección virtual formada por la Base B_1 y el desplazamiento D_1

(dirección traducida utilizando la tabla de segmentos secundaria) se copia a la dirección virtual formada por la base B_2 y el desplazamiento D_2 (dirección virtual traducida utilizando la tabla de segmentos primaria). Los bits 24-27 del registro general indicado en R_3 representan la llave de acceso para el Espacio de Direcciones secundario (los demás bits del registro son ignorados), mientras que el registro R_1 es usado para designar la cantidad (igual o menor a 256) de bytes que se moverán en la instrucción. La instrucción en resumen, mueve caracteres desde el Espacio de Direcciones secundario hacia el Espacio de Direcciones primario.

Para el caso de la instrucción "Mueve caracteres al secundario" ("Move Characters to Secondary"), Nemónico MVCS,Opcode x'DB', el formato es el mismo, es decir:

$$\text{MVCS } D_1(R_1,B_1),D_2(B_2),R_3$$

En donde los operandos y las consideraciones de traducción son las mismas que para el caso de MVCP, pero aquí el movimiento de caracteres es desde el Espacio de Direcciones primario hacia el Espacio de Direcciones secundario.

Estas dos instrucciones permiten una comunicación directa entre dos Espacio de Direcciones, por medio de intercambio de caracteres del Espacio de Direcciones primario al Espacio de Direcciones secundario y viceversa.(figura 2.23)

Existen otras dos instrucciones que permiten que dos Espacios de direcciones interactúen: La instrucción "Llama Programa" ("Program Call") y la instrucción "Transfiere Programa" ("Program Transfer").

La instrucción "Program Call", Nemónico PC, Opcode x'B218', nos permite iniciar la ejecución de un programa en un Espacio de Direcciones distinto al Espacio de Direcciones en el que estamos ejecutando. La instrucción "Program Transfer", Nemónico PT, Opcode x'B228', nos permite regresar al Espacio de Direcciones original, reanudando la ejecución en la instrucción siguiente al "Program Call".

El formato de la instrucción PC es el siguiente:

$$\text{PC } D_2(B_2)$$

En donde la dirección que forma el operando se usa como el "Número de Program Call", no como dirección. Este Número de program Call se usa para hacer una búsqueda en dos tablas: La Tabla de Ligas ("Linkage Table", LT) y la Tabla de Entradas ("Entry Table", ET). Esta búsqueda se lleva a cabo usando partes del segundo operando como índices para cada una de las tablas. Indexando en la Tabla de Ligas llegamos a una Tabla de Entradas, la cual nos da la

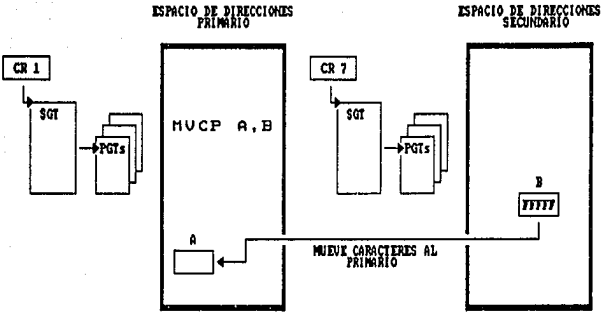


FIGURA 2.23 (A)

MOVIMIENTO DE CARACTERES DEL ESPACIO DE DIRECCIONES SECUNDARIO AL PRIMARIO.

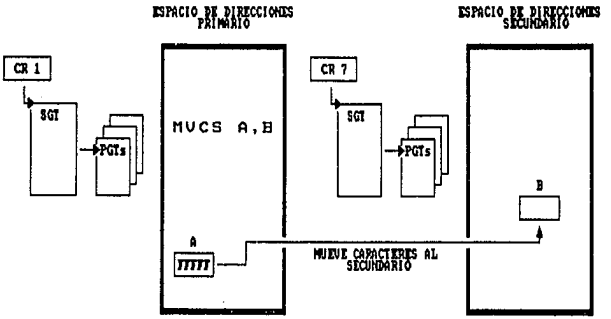


FIGURA 2.23 (B)

MOVIMIENTO DE CARACTERES DEL ESPACIO DE DIRECCIONES PRIMARIO AL SECUNDARIO.

dirección real del programa que queremos ejecutar (en otro address space).

Cada Espacio de Direcciones tiene un "Número de Addres Space" asignado ("Address Space Number", ASN). El sistema proporciona un mecanismo por el cual se puede encontrar el origen de la Tabla de Segmentos conociendo el número de Espacio de Direcciones. El ASN primario está contenido en los bits 16-31 del registro de control 3 y al ASN secundario está contenido en los bits 16-31 del registro de control 4.

La traducción de ASN utiliza dos tablas: La "Primera tabla de traducción de ASN" ("ASN-translation first table") y la "Segunda tabla de traducción de ASN" ("ASN-translation second table").

El origen de la primera tabla (es decir, la dirección en memoria real del inicio de la primera tabla) está contenido en el registro de control 14.

Las entradas de la primera tabla de traducción contienen el "Origen de las Segundas Tablas de Traducción" ("ASN Second Table Origin", ASTO). Las entradas de las segundas tablas de traducción contienen el origen de la tabla de segmentos (STO).

La instrucción "Transfiere Programa" ("Program Transfer"), Nemónico PT, Opcode x'B228' tiene el siguiente formato:

PT R_1,R_2

Esta instrucción regresa control al Espacio de Direcciones primario. El número de Espacio de Direcciones primario, el número de Espacio de Direcciones secundario y la llave de la PSW se especifican en el contenido del primer operando mientras que el contenido del segundo operando especifican el valor de los bits de modo de direccionamiento, estado de la CPU, y la IA en donde se reanudará la ejecución.

La forma en que se establece una Espacio de Direcciones secundario es mediante la instrucción "Set Secondary ASN" Nemónico SSAR, Opcode x'B225', formato:

SSAR R_1

Esta instrucción reemplaza el contenido del registro de control 3 con los bits 16-31 especificados en el registro R_1 (llamado la nueva ASN) y además la STO correspondiente a este ASN es guardado en el registro de control 7 (reemplazando así a la Secondary SGTO, SSTO).

Otras instrucciones que pueden cambiar los contenidos de los registros de control que especifican tanto los Espacio de Direcciones primario, secundario y los ASN primario y

secundario son las instrucciones "Load Address Space Parameters" y "Load Control".

La instrucción "Load Address Space Parameters" (Carga parámetros del Espacio de Direcciones), Nemónico LASP, Opcode x'E500', formato:

$$\text{LASP} \quad D_1(B_1),D_2(B_2)$$

Toma de la doble palabra localizada en la dirección formada por el primer operando, valores que cargará en el registro de control 3 y 4, incluyendo un ASN secundario y un ASN primario. La ejecución de la instrucción consiste en 3 pasos: La traducción del PASN, la traducción del SASN y carga de los registros de control 1,5, y 7 como resultado de las traducciones.

La instrucción "Load Control" (Carga Registros de Control), Nemónico LCTL, Opcode x'B7', formato:

$$\text{LCTL} \ R_1,R_3,D_2(B_2)$$

Carga los registros de control comenzando por el registro indicado en R_1 y finalizando con el registro indicado en R_3, desde las localidades que inician en la dirección virtual formada por $D_2(B_2)$.

En el siguiente capitulo se abordará con detalle la estructura interna del MVS. Se presentará primero una breve introducción acerca de la evolución del MVS, después se revisarán algunos conceptos importantes para la comprensión de temas posteriores y posteriormente abordar cada uno de los componentes del MVS.

CAPITULO III.- ESTRUCTURA INTERNA DEL SISTEMA OPERATIVO MVS

Este Capítulo analizará a detalle la forma en que el sistema operativo "Memoria Virtual Múltiple" ("Multiple Virtual Storage", MVS) explota la arquitectura del sistema/370-XA para solucionar el problema de ejecutar concurrentemente varios programas, es decir la forma en que se implanta el concepto de Multiprogramación.

3.1 EVOLUCION DEL MVS

A mediados de la década de los 60's, aparecieron los primeros sistemas operativos para soportar al Sistema/360 recién diseñado . Estos sistemas operativos sólo permitían que un programa se ejecutara a la vez, con una capacidad de direccionamiento exactamente igual a la memoria principal disponible en la máquina. El ambiente era escencialmente el mismo que se tiene actualmente en muchos sistemas de computadoras personales, en el que el manejo de los trabajos ("Jobs") se deja a un operador humano quiene decide que se va a ejecutar después de que un programa acabe de ejecutar, y en donde todo "Job" debe esperar a que termine el anterior para poder ejecutarse.

En la figura 3.1(A) hay una serie de "Modelos" que esquematizan la evolución del MVS. Como se indica en el modelo del "Programa de control primario" ("Primary Control Program",PCP) en la figura 3.1(A), sólo se cargaban el código del sistema operativo y un programa de aplicación a la vez. Este modo de operar sólo requiere de una distinción de autoridad:o se es un programa privilegiado (sistema operativo) o se es un programa de aplicación. Este modo dedicado a una sola aplicación es una forma ineficiente de manejo de los recursos debido a que deja al procesador sin trabajo durante las operaciones de I/O y entre la terminación de la ejecución de un "Job" y la iniciación de otro.

En la figura 3.1(B) se muestra como se introdujeron cambios en el diseño para permitir que varios programas de aplicación residieran al mismo tiempo en memoria, pudiendo así implantar el concepto de "Multiprogramación". De aquí nacieron los sistemas operativos "Multiprogramación con un número fijo de tareas" ("Multiprogramming with a fixed number of tasks", MFT) y "Multiprogramación con un número variable de tareas"("Multiprogramming with a variable number of tasks", MVT). Varios programas se cargan en memoria concurrentemente para compartir tanto al procesador como la memoria y demás recursos del sistema. De esta forma cuando una tarea requiere hacer alguna operación de I/O, esta tarea se puede suspender hasta la terminación de la operación de I/O y mientras tanto otra tarea puede ejecutarse. Para asegurar la integridad de los datos, este arreglo requiere un mecanismo de protección para restringir el

direccionamiento de cada tarea sólo a la partición de memoria en la que está alojada. En el Sistema/360 esto se hacia con una arquitectura de protección por medio de llave, que permitía tener 16 áreas distintas protegidas y una llave maestra que tenía acceso a toda la memoria.

Aunque la introducción de la multiprogramación hizo que el procesamiento de datos fuera bastante más eficiente que antes además de hacer que el procesador se utilizara en forma eficiente,también creó mucha más demanda de memoria. Como la memoria tenía que ser compartida entre las diferentes tareas que convivían concurrentemente en la memoria, el rango de direccionamiento de estas tareas se vio reducido. Esta restricción en memoria repercutió en una limitación en el tamaño de los programas y la cantidad de datos que podían ser asociados con estos.

Esto dejó a los usuarios con la responsabilidad de buscar un compromiso entre el número de tareas que corrían concurrentemente y el rango de direccionamiento que estas necesitaban para funcionar correctamente. Esto motivó al desarrollo del Sistema/370 y al sistema de memoria virtual mostrado en la figura 3.1(C). En este sistema operativo de "Memoria Virtual Sencilla" ("Single Virtual Storage", SVS), la memoria que estaba disponible para todas las tareas se vio aumentada por el uso de memoria virtual, que en el caso del Sistema/370 permitía explotar todo el límite de direccionamiento impuesto por la arquitectura, es decir un Espacio de direcciones de 16 Mbytes de direccionamiento lineal.En este sistema operativo, sin embargo, para implantar la multiprogramación todavía había que dividir este espacio de direcciones en particiones, una para cada una de las tareas que se ejecutaran concurrentemente en el sistema, y consecuentemente limitando la cantidad disponible para cada uno de los programas.

Este problema se atacó desarrollando un sistema operativo que diera a cada usuario un Espacio de direcciones independiente de los demás. Esta estructura se muestra en la figura 3.1(D); Es la "Memoria Virtual Múltiple" ("Multiple Virtual Storage", MVS). En el MVS el rango de direccionamiento se divide en una área que es común a todos los usuarios y que contiene datos y programas que son relevantes a muchos usuarios, principalmente funciones, servicios y subsistemas propios del sistema operativo; y un área privada que contiene datos y programas importantes para un sólo usuario y que por lo tanto no requiere ser común a todos. Usando direccionamiento virtual, esta estructura de Espacios de direcciones múltiples hizo posible mayor cantidad de memoria virtual por usuario y la posibilidad de extender la protección más allá de los límites de las llaves de memoria.

FIGURA 3.1(A)
PCP

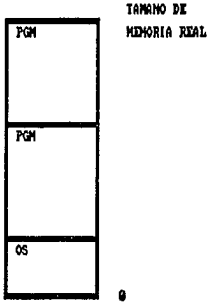


FIGURA 3.1(B)
MVT/MFT

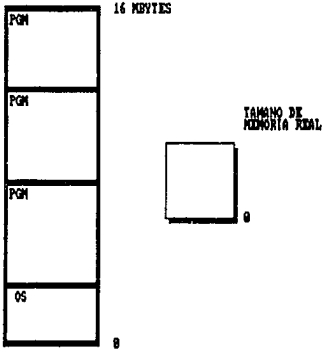


FIGURA 3.1 (C)
SVS

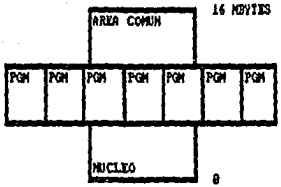


FIGURA 3.1 (D)

MVS

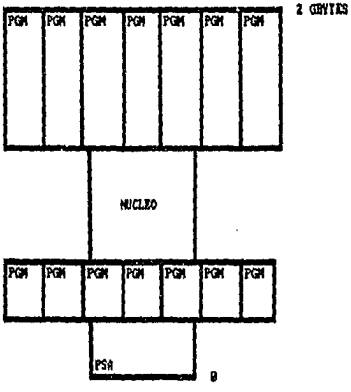


FIGURA 3.1 (E)

MVS/XA

Con las ventajas que tiene la estructura de Espacios de direcciones múltiples , muchos subsistemas que antiguamente residían en el área común del sistema operativo, como el "Subsistema de entrada de trabajos" ("Job Entry Subsystem", JES) comenzaron a explotar esta estructura creándose así un Espacio de direcciones para cada uno de estos, disminuyendo el área común y aumentando el rango de direccionamiento del área privada de cada usuario.

Teniendo cada subsistema su propio espacio de direcciones era posible mantener datos, "buffers" y demás información altamente consumidora de memoria en el área privada de cada subsistema, pero con la parte principal de su código corriendo en el área común. La arquitectura también da facilidades para poder mover datos de un espacio de direcciones a otro, permitiendo la comunicación entre cualquier espacio de direcciones y los subsistemas. Esto se hace permitiendo que el procesador conozca al mismo tiempo a dos espacios de direcciones: uno primario y otro secundario y mediante el uso de instrucciones para mover datos hacia el primario o el secundario o para poder transferir el control a un programa en otro espacio de direcciones. Esta facilidad permite el intercambio directo de información entre espacios de direcciones que de otra forma se tendría que hacer copiando la información de el área privada de un espacio de direcciones al área común y de esta al área privada de otro espacio de direcciones.

Sin embargo, aún cuando estas facilidades mejoraron la utilización de múltiples espacios de direcciones, el espacio de direcciones de 16 Mbytes seguía siendo una restricción importante para muchos ambientes. El Sistema/370-XA se introdujo para incrementar el tamaño del espacio de direcciones, direccionando ahora con 31 bits, lo que extendió el rango de direccionamiento virtual de 16 Mbytes a 2Gbytes. Como se muestra en la figura 3.1(E), la estructura del MVS/XA hace uso de espacios de direcciones más grandes, aprovechando así el cambio de arquitectura.

3.2 CONCEPTOS GENERALES

Como se vio en el capitulo I, Un sistema operativo es un grupo de programas relacionados entre sí que tienen el objetivo de gobernar el sistema computacional. El sistema operativo controla la ejecución de programas y proveé servicios que se necesitan para hacer uso del hardware del computador. El MVS/XA es el sistema operativo que explota el Sistema/370-XA.

La arquitectura de una computadora consiste en aquellas funciones que el sistema proveé. Es distinto del diseño físico, y de hecho, diferentes diseños de máquina pueden conformar la misma arquitectura de computadora.

La arquitectura es, en cierta forma, la computadora tal como la ve el programador del sistema. Por ejemplo, parte de la arquitectura es el "set" de instrucciones de la máquina.

El Sistema/370-XA, como se vio en el punto anterior, es una extensión del Sistema/370. De igual forma, el sistema operativo MVS/XA es una extensión del MVS/370, que explotaba la arquitectura del Sistema/370.

Las diferencias entre el MVS/370 y el MVS/XA se centran en tomar ventaja de las mejoras hechas al hardware en beneficio del rendimiento y la rentabilidad del sistema. Los dos cambios más significativos son:

+ Direccionamiento con 31 bits

El MVS/370 tenía un esquema de direccionamiento de 24 bits. El MVS/XA puede direccionar tanto a 24 bits como a 31 bits. Esto cambia la memoria disponible a cualquier usuario de 16 Mbytes a 2 Gbytes.

+ El Subsistema de canales ("Channel Subsystem")

El subsistema de canales maneja las operaciones de I/O independientemente de la CPU en un sistema MVS/XA. Con el sistema MVS/370 también se podían procesar concurrentemente operaciones de I/O y ejecución de programas, sin embargo con el MVS/XA y el subsistema de canales se incrementa esta capacidad de concurrencia, liberando a la CPU de mucho del trabajo que antes tenía que realizar (como por ejemplo la selección del "Path" que se utiliza en la operación de I/O, función que ahora hace el microcódigo del subsistema de canales).

Sin embargo, ninguno de estos cambios creó la necesidad de cambiar aplicaciones ya existentes: Las aplicaciones que fueron escritas para direccionar con 24 bits pueden correr en un MVS/XA, y no ha habido cambios en la forma en que los programas invocan operaciones de I/O.

3.2.1 El entorno del MVS/XA

Para entender como y porqué el MVS funciona como lo hace es necesario entender el entorno en el que funciona. Las características que hacen único al MVS/XA reflejan las características del ambiente de computación que el MVS/XA maneja.

El Sistema/370-XA, que es el que explota el MVS/XA, cuenta con las herramientas necesarias para implantar la "Multiprogramación", es decir, ejecutar muchos programas concurrentemente.

Por medio de la multiprogramación el sistema puede, por ejemplo, correr cientos de "Jobs" simultáneamente para usuarios que pueden estar en distintos puntos geográficos.

El MVS/XA como sistema operativo, puede manejar también el "Multiprocesamiento", que es la operación simultánea de dos o más procesadores que comparten los dispositivos de hardware del sistema, como se muestra en la figura 3.2.

El hecho de que muchos usuarios tengan capacidad de correr muchos programas simultáneamente usando a la vez varios procesadores implica que se necesitan grandes cantidades de memoria para asegurar un rendimiento óptimo del sistema. Estas aplicaciones necesitan también que el sistema operativo les proporcione rutinas para proteger su privacía, en un compromiso con rutinas que les permitan compartir bases de datos y servicios comunes de software.

Así es que la capacidad de brindar Multiprogramación, Multiprocesamiento, y la necesidad de grandes cantidades de memoria significa que el MVS/XA debe dar a los usuarios mucho más de lo que se esperaría de un simple manejador de trabajos. Los siguientes conceptos explican de manera introductoria los atributos que permiten al MVS/XA manejar configuraciones computacionales complejas.

- Memoria Virtual

Las siglas de MVS quieren decir "Memoria Virtual Múltiple" ("Multiple Virtual Storage"), lo que indica que cada usuario tiene acceso a memoria virtual, no real (física). La memoria virtual (como se explicó en el capítulo II) implica que cada programa puede asumir que tiene acceso a toda la memoria que el esquema de direccionamiento permite, el único límite es el número de bits en una dirección de memoria.

El esquema de direccionamiento de 31 bits soportado por MVS/XA permite a un programa direccionar hasta 2 Gbytes de memoria; en contraste, el sistema tiene mucho menos memoria real; el tamaño de esta memoria real depende del modelo y la configuración de la instalación.

Para permitir que cada usuario crea que tiene mucha más memoria de la que existe realmente, MVS/XA hace uso de las facilidades de traducción del sistema (DAT) para llevar un control de cuales son las partes activas de cada programa; estas partes activas son las que mantiene en memoria, mientras que el resto se encarga de guardarlos en archivos especiales llamados "Archivos de paginación" ("Page Data Set") que residen en "Dispositivos de Acceso Directo" ("Direct Access Storage Devices", DASD) es decir, discos magnéticos.

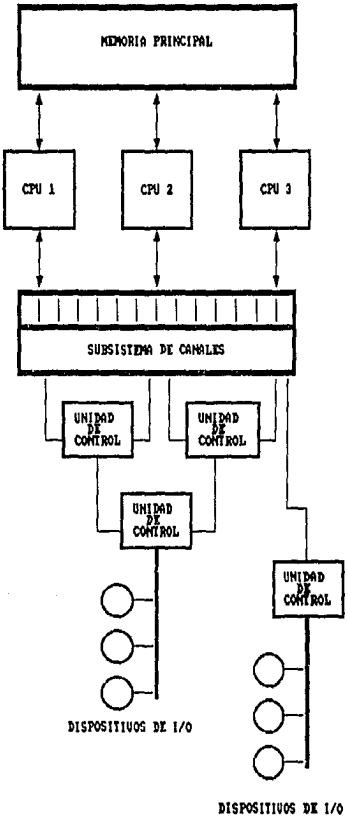


FIGURA 3.2

"MULTIPROCESAMIENTO": COMPLEJO DE 3 PROCESADORES COMPARTIENDO SUBSISTEMA DE CANAL Y MEMORIA PRINCIPAL

El concepto de memoria virtual es entonces la combinación de memoria real y auxiliar (DASD) y el manejo adecuado del sistema operativo. Desde luego esto implica que el MVS/XA requiere de una gran cantidad de capacidad de almacenamiento en DASD para hacer posible esto.

- Espacios de Direcciones

Como se mencionó en el capitulo II, el rango completo de direcciones virtuales de 0 a 2 Gbytes se conoce como "Espacio de Direcciones" ("Address Space"). El MVS/XA proveé a CADA USUARIO de un espacio de direcciones de 2 Gbytes. También proveé servicios de "Memoria Cruzada" ("Cross Memory Services") que permiten que cualquier usuario se comunique con otro espacio de direcciones en cualquier momento.

Sin embargo, la habilidad de compartir recursos entre todos los usuarios implica también la necesidad de mecanismos de protección para poder garantizar la integridad de datos y programas. El MVS/XA utiliza, además de el uso de llaves de memoria, mecanismos de "Cerradura" ("Locking") y mecanismos de serialización de requerimientos para asegurar esta integridad, mismos que se tratarán posteriormente.

3.2.2 Manejo de tareas

El MVS/XA separa cada "Job" en diferentes unidades de trabajo conocidas como tareas ("Tasks") e intenta procesar cada una tan eficientemente como sea posible. Las tareas de un "Job" compiten unas con otras, y con tareas de otros "Jobs", por el uso de los recursos del sistema. La responsabilidad de controlar el progreso de las tareas cae en un componente del MVS que se llama el "Supervisor". El supervisor aloja recursos (Excepto los dispositivos de I/O) y mantiene información actualizada de cada tarea de manera que hace posible que se reanude la ejecución en el punto indicado en caso de que se sufra una interrupción.

El MVS/XA cuenta con algunos mecanismos que permiten al supervisor y a otros componentes del sistema mantener el control. En esta sección se describirán brevemente 4 de ellos : Bloques de Control, "Program Status Word", Interrupciones y las macro instrucciones (algunos de estos conceptos ya se abordaron con algún detalle en el capítulo anterior). Posteriormente se describirán otras características especiales del manejo de tareas del MVS.

1) Bloques de Control

Los módulos del MVS/XA normalmente guardan la información necesaria para controlar una unidad de trabajo particular o para manejar un recurso en áreas de memoria que se llaman "Bloques de Control". Estos bloques de control son

simplemente localidades de memoria que contienen información acomodada de manera predeterminada. Se habla de "Desplazamientos" dentro de un bloque de control cuando nos referimos al número de bytes que hay desde el inicio del bloque de control hasta un byte en particular de este bloque de control. Cada pieza de información reside en un desplazamiento específico dentro de este bloque de control.

Hablando en términos generales, existen tres tipos de bloques de control en MVS/XA:

-Bloques de control del sistema

Son bloques de control que contienen información que afecta al sistema en su totalidad, información tal como cuantos procesadores están funcionando en el sistema.

-Bloques de control de recursos

Cada bloque de control de recursos representan un recurso, como un procesador o un dispositivo de almacenamiento auxiliar.

-Bloques de control de Tareas

Cada bloque de control relacionado con una tarea representa una unidad de trabajo.

Los bloques de control son el vehículo de comunicación interna del MVS/XA. Esta comunicación es posible debido a que la estructura de los bloques de control es conocida por todos los usuarios: Está documentada en manuales que reciben el nombre genérico de "Areas de datos" ("Data Areas").

Los bloques de control que representan muchas unidades del mismo tipo pueden ser encadenados en "Colas" ("Queues"), con cada bloque de control apuntando al siguiente o bien con cada bloque de control apuntando al siguiente y al anterior. Un programa puede hacer una búsqueda a lo largo de la cola para encontrar una unidad de trabajo en particular o un recurso. Estos recursos pueden ser:

- Una dirección (de un bloque de control o de una rutina).
- Datos, como una cantidad, un parámetro o un nombre.
- "Banderas" de estatus (usualmente bits particulares que indican si una condición está presente).

Todos los campos dentro de un bloque de control están definidos e identificados en la estructura documentada en los "Data Areas" del bloque de control específico. (En la figura 3.3 se muestra un extracto de una descripción de un bloque de control, tal y como está documentado en un manual "Data Areas").

Bloque de Control : CVT
Nombre Común: Tabla de vectores de comunicación. (Communications vector table)
Creado por: IEAVCVT
Tamaño: 1264 bytes
Apuntada por: El campo PLCCVT de la PSA (Localidad x'10')
Función: Proveé el medio por el cual las rutinas no residentes se pueden referir a la información contenida en el núcleo del programa de control. Contiene direcciones de otros bloques de control y tablas usadas por rutinas del programa de control.

OFFSETS DEC	HEX	TIPO	LONG.	NOMBRE	DESCRIPCION
0	(0)	Dirección	4	CVTTCBP	Dirección de la siguiente TCB a ser despachada.
4	(4)	Dirección	4	CVT0EF00	Dirección de la rutina de "Exits" asíncronas.
.	.	.	.	.	.
116	(74)	Bitstring	1	CVTDCB	
		1...		CVTMVSE	x'80'-Modo S/370
		.1..		CVT1SSS	x'40'-Opción 1 (PCP)
		..1.		CVT2SPS	x'20'-Opción 2 (MFT)
		...1		CVT4MS1	x'10'-Opción 4 (MVT)
		.... .1..		CVT4MPS	x'04'- Multiprocesamiento mod 65
		.... ..1.		CVT6DAT	x'02'- DAT por CPU
		1		CVTMVS2	x'01'- OS/VS2 presente.

Figura 3.3
Ejemplo de Especificación de un bloque de control

Los bloques de control pueden ser de distintos tamaños y tener formatos distintos. Usualmente, los bloques de control consisten en una serie de campos de palabras ("Fullwords"), como el campo CVTTCBP de la figura 3.3, pero también pueden ser más grandes (Como el nombre de un archivo de datos) o más cortos como el byte de banderas llamado CVTDCB de la figura 3.3, en donde cada bit tiene un significado.

Los puntos importantes a recordar en cuanto a bloques de control son el hecho de que los bloques de control están

estructurados de manera única e invariable, que esta estructura está documentada y que usualmente los bloques de control están encadenados a otros bloques de control. En la figura 3.4 se ilustra una cadena de bloques de control llamados TCB's.

Durante el resto del presente trabajo se hará especial énfasis en los bloques de control más importantes del MVS y en las relaciones que existen entre ellos.

2) "Program Status Word"

La Palabra de estatus del programa o PSW es una localidad de memoria de 64 bits que, como ya se analizó, nos da detalles cruciales tanto del hardware como del software. Incluye la dirección de la siguiente instrucción a ejecutarse e información de control del programa que está corriendo, tal como si se direcciona en 24 o 31 bits o si está corriendo en estado supervisor o en estado de problema.

El MVS/XA, como programa, es un usuario privilegiado y la mayor parte del tiempo corre en estado supervisor. Cada procesador tiene sólo una PSW actual, así es que sólo se puede ejecutar una tarea a la vez en un procesador. Sin embargo, la multiprogramación es posible debido a que las interrupciones cambian el contenido de la PSW y por lo tanto cambian la tarea que se estaba ejecutando.

3) Interrupciones

Una interrupción es un requerimiento de atención hacia algún procesador. Indica que ha sucedido un evento, como la terminación de una operación de I/O, la expiración de un intervalo de tiempo o un error en un programa.

La filosofía de diseño del MVS/XA hace que cada vez que se sufre una interrupción, el sistema operativo tome control y realize alguna acción. Se dice que el MVS/XA es un sistema operativo "Manejado por interrupciones" ("Interrupt Driven").

Cuando se sufre una interrupción corriendo bajo el sistema operativo MVS/XA, una rutina manejadora de interrupciones toma control. En MVS/XA existen dos niveles de manejo de interrupciones. Para cada tipo de interrupción existe una rutina "Manejadora de Interrupciones de Primer Nivel" ("First Level Interrupt Handler", FLIH) que guarda información crucial acerca del estatus de la tarea interrumpida y que le da control a una rutina "Manejadora de Interrupciones de Segundo Nivel" ("Second Level Interrupt Handler", SLIH) que es la que realmente atiende a la interrupción.

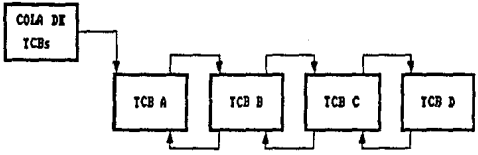


FIGURA 3.4
COLA DE BLOQUES DE CONTROL (TCBs)

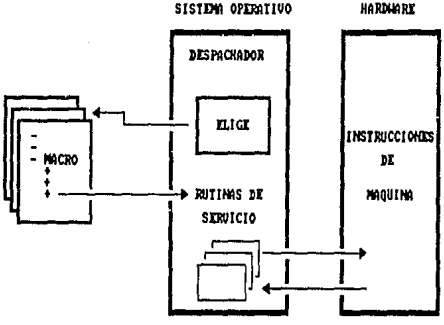


FIGURA 3.5
FUNCION DE LAS MACRO INSTRUCCIONES

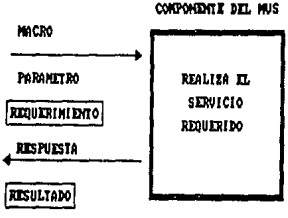


FIGURA 3.6
PARAMETROS DE ENTRADA DE LAS MACRO INSTRUCCIONES

Una vez que se atendió la interrupción, un componente del sistema llamado el "Despachador" ("Dispatcher") se encarga de seleccionar la unidad de trabajo de más alta prioridad que esté lista para procesarse (No necesariamente la que fue interrumpida) y le da control hasta que termina o hasta que otra interrupción ocurre. Una interrupción, por lo tanto, le da la oportunidad al despachador de que reevalúe y reasigne la prioridad de cada tarea para hacer más eficiente el trabajo.

4) Macro Instrucciones

La comunicación entre los programas que corren en MVS/XA es posible debido a que los programadores del sistema siguen convenciones de programación establecidas y usan macro instrucciones comunes.Estas macro instrucciones (o macros) invocan segmentos de código frecuentemente usados que "mapean" bloques de control o realizan alguna función muy común. Existen macros de MVS/XA disponibles para los programadores del sistema que realizan funciones tales como abrir y cerrar archivos, cargar y borrar programas de memoria, o enviar mensajes al operador del sistema.

3.2.3 Manejo de Recursos

La implantación del multiprocesamiento y la multiprogramación crean la necesidad de medir la actividad del sistema y ajustar la carga de trabajo para ajustarse a condiciones cambiantes. El MVS/XA monitorea cual es el consumo que hace cada espacio de direcciones de recursos del sistema como: procesadores, dispositivos de I/O, localidades de memoria real, etc. Un componente del MVS/XA llamado el "Manejador de recursos del sistema" ("System Resource Manager", SRM) interpreta esta información y determina si un espacio de direcciones debe permanecer residente en memoria real, o si se puede, en determinado momento, crear un espacio de direcciones nuevo.

Como ya se explicó, el concepto de memoria virtual implica el hecho de mover páginas de memoria hacia o desde un archivo ("Data Set") , con el objetivo de tener en memoria real sólo a las páginas activas o más recientemente referenciadas en determinado momento; este movimiento de paginas entre memoria auxiliar y memoria real se llama "Paginación"; se dice que una página esta "Paginada dentro" ("Paged-in") cuando está en memoria real y "Paginada Fuera" ("Paged-out") cuando reside en memoria auxiliar.

El SRM va más allá de la paginación y se encarga de analizar qué espacios de direcciones tienen actividad en un momento dado y cuales están esperando por algún recurso. El SRM pagina afuera ("Page-out") todas las páginas pertenecientes a aquellos Espacios de Direcciones que estén esperando por algo o a aquellos que el SRM determine que por alguna razón

no deban estar en memoria real; esta operación se conoce como "Swappear Fuera" ("Swapp-out") a un espacio de direcciones y a la operación inversa se le conoce como "Swappear dentro" ("Swapp-in") a un espacio de direcciones. En conclusión, todos los espacios de direcciones que están activos en determinado momento se encuentran "Swappeados dentro" y todos aquellos que estén inactivos se encuentran "Swappeados fuera".

El SRM también se encarga de estar al tanto de los objetivos de carga de trabajo y de las prioridades de cada usuario, especificados en la "Especificación del rendimiento de la instalación" ("Installation Performance Specifications", IPS). El SRM es en conclusión el principal instrumento por medio del cual la instalación (entendida como el complejo de los sistemas computacionales) maneja los recursos del sistema.

El MVS/XA proporciona al usuario herramientas adicionales para el control del uso de los recursos. Estas incluyen Parámetros del sistema, Rutinas de Salida (Exit Routines, ó "Exits") y la consola del operador.

1) Parámetros del sistema

Los parámetros del sistema especificados por IBM o por la instalación están guardados en una "Biblioteca" del sistema llamada la SYS1.PARMLIB (Todas las bibliotecas del sistema se identifican por el primer calificador "SYS1"). Cada uno de los miembros de esta biblioteca contiene parámetros que el operador selecciona para controlar el proceso. Por ejemplo, el miembro IEASYS00 contiene los valores que el sistema toma por omisión para la configuración del MVS/XA en su arranque. El sistema usa estos parámetros y otros valores durante el proceso de inicialización del sistema (IPL).

2) Rutinas de Salida ("Exits")

Una "Exit" es un punto definido de un módulo del sistema operativo en donde se llama a otro programa. Los programas que son llamados por omisión son proporcionados por IBM; sin embargo su intención es que sean reemplazados por otros programas que el usuario codifique de acuerdo a sus propias necesidades. Estas "exits" permiten que el usuario, en determinadas partes del código, pueda modificar el procesamiento, sin necesidad de alterar el código del MVS/XA en sí.

3) Consola del Operador

La instalación controla al MVS/XA por medio de "Comandos del sistema" tecleados desde uno o más dispositivos definidos al MVS/XA como consolas de operador. Existe además una consola que permite a los Ingenieros de hardware diagnosticar y corregir problemas de hardware, esta consola se conoce como "Consola Nativa".

A través de comandos del sistema tecleados desde una consola, el operador o el programador del sistema pueden controlar al MVS/XA o responder a una condición que el MVS/XA detecte. Mediante comandos de sistema se puede, por ejemplo:

- Cambiar el estatus de los dispositivos de Hardware, es decir habilitarlos (ponerlos "Online" ó en linea) o deshabilitarlos (ponerlos "Offline" o fuera de linea).

- Monitorear la actividad de varias unidades de trabajo en el sistema.

- Cambiar algunos parámetros después de la inicialización del sistema.

- Arrancar y parar funciones del sistema.

- Habilitar una "trampa" para diagnosticar problemas recurrentes.

3.2.4 Manejo de datos y I/O

Prácticamente todas las tareas involucran cierta cantidad de operaciones de entrada y salida de datos (operaciones de I/O). El subsistema de canales maneja el uso de los dispositivos de I/O, como discos, unidades de cintas, impresoras,etc; mientras que el MVS/XA, por parte del software, asocia los datos de una tarea al dispositivo en donde residen o residirán.

El MVS/XA maneja los datos por medio de "Data Sets". Los "Data Sets" pueden guardar información usualmente conocida como "Archivos" o bien, los "Data Sets" pueden guardar información que la computadora necesita, como parámetros del sistema o programas.

Los registros de un "data set" pueden estar organizados de varias formas, dependiendo de la manera en la que la información va a ser accesada; con esta base, los "Data Sets" se pueden agrupar en forma general en dos tipos de acceso: Acceso secuencial o Acceso directo.

En un "Data Set" organizado para acceso secuencial, los registros son piezas de información (registros) guardadas consecutivamente. Esto implica que para recuperar el décimo elemento (o registro) de un "Data Set" secuencial, es necesario leer por orden desde el primer registro hasta el décimo. Este tipo de organización se usa para guardar datos que por su naturaleza siempre necesitan recuperarse uno tras otro, como una lista de nombres, por ejemplo.

En un "Data Set" organizado para acceso directo, también llamado "Acceso aleatorio" ("Random Access"), los registros son grabados con información de control de manera que el sistema pueda recuperar un elemento sin necesidad de pasar por los elementos que le preceden.

Otro tipo de "Data Sets", llamados "Data sets particionados" combinan las características de acceso secuencial y directo. Los "Data Sets" consisten de un "Directorio" y "Miembros". El directorio guarda la dirección de cada miembro y hace posible el acceso directo a cada miembro. Sin embargo, los miembros consisten en registros guardados secuencialmente.

Los "Data Sets" particionados se llaman "Bibliotecas". Los programas son guardados como miembros de "Data Sets" particionados de manera que, aunque el código de estos programas se guarda en forma secuencial, el sistema operativo puede accesarlos directamente en el momento en que se seleccione uno para ejecución.

El MVS/XA soporta diferentes dispositivos para guardar datos. Los discos (DASD) y las cintas son los más usados para guardar datos a largo plazo. Los dispositivos DASD se usan generalmente para guardar datos de acceso directo (de hecho, de ahí las siglas de "Direct Access Storage Devive" ó Dispositivo de Memoria de Acceso Directo) aunque esto no excluye que puedan guardar "Data Sets" secuenciales. Las unidades cintas, por el contrario, son dispositivos de acceso secuencial únicamente.

Para permitir al sistema localizar un "Data Set" específico rápidamente, el MVS/XA cuenta con un "Data Set" conocido como el "Catálogo Maestro" ("Master Catalog") que permite el acceso a cualquier "Data Set" del sistema o bien a otros catálogos de "Data Sets". El MVS/XA requiere que el "Master Catalog" resida en un DASD que siempre esté habilitado (Online) en el sistema. Otros "Data sets" que son vitales para el sistema operativo requieren residir en un DASD particular conocido como SYSRES, o "Volumen Residente del Sistema" (SYStem RESidence Volume), que también debe estar siempre "Online".

3.2.5 Manejo de "Jobs"

Para indicarle al MVS/XA lo que se quiere que haga, es necesario codificar un "JOB". Un "Job" no es más que una serie de enunciados de " Lenguaje de Control de Jobs" ("Job Control Language", JCL). Estos enunciados, que son como un lenguaje de programación, permiten identificar el programa que se quiere ejecutar, los "Data Sets" que va a accesar, la forma en que los va a accesar y los dispositivos que va a usar. Además, si se van a ejecutar varios programas, permite hacer decisiones tales como si se va a ejecutar o no el siguiente programa dependiendo de como haya acabado la ejecución del anterior.

Los "Jobs" se procesan mediante un subsistema del MVS/XA llamado el "Subsistema de entrada de Jobs" ("Job Entry Subsystem", JES) que interpreta los enunciados del "Job", organiza los programas necesarios, datos y recursos y le presenta al MVS/XA un "Job" que está listo para ejecutarse. Después de que el "Job" concluye su ejecución, el JES libera a los recursos usados y se encarga de manejar la salida del "Job" en forma impresa.

El MVS/XA permite muchas formas de introducir "Jobs" al sistema. Con procesamiento "Batch" (en lotes), un usuario introduce un "Job" por medio de una terminal o por medio de una "Entrada de Job Remota" ("Remote Job Entry"), a través de una terminal remota, o desde cinta o lectora de tarjetas (actualmente estos dispositivos son totalmente obsoletos, aunque muchos dispositivos modernos "emulan" a una perforadora o lectora de tarjetas), y el sistema procesa este "Job" en algún momento posterior. En el proceso de asignar tiempo y recursos al "Job", el sistema sigue lineamientos específicos para la instalación establecidos en los parámetros del sistema.

Con la introducción "Interactiva" de "Jobs", como la llamada "Opción de Tiempo Compartido" ("Time Sharing Option", TSO), el sistema responde a los usuarios de las terminales mientras estos están usando el sistema. El MVS/XA permite también que el operador inicie la ejecución de "Jobs" por medio del comando "START" (Arranca); los "Jobs" iniciados de esta forma son llamados "Tareas Iniciadas" ("Started tasks").

3.2.6 Manejo de Recuperación

El sistema de procesamiento de datos debe estar disponible cuando se le necesite. Para el caso de un sistema grande, esto significa que el sistema debe funcionar aún cuando uno de los componentes falle y que el sistema pueda, si es posible, diagnosticar la falla y corregir o compensar la función de este componente. El MVS/XA incluye mecanismos de

recuperación que pueden: prevenir que un error de usuario repercuta en una falla de todo el sistema, aislar y recuperarse de errores del sistema operativo y proteger al sistema de errores de hardware. También cuenta con programas que "Rastrean" ("Trace") la actividad del sistema y despliegan el estatus y contenidos de varios recursos del sistema.

El resto de este capítulo estudiará con detalle la forma en que el MVS/XA implanta las características mencionadas en esta visión global del mismo, y la forma en que el MVS/XA aborda el problema de manejar un sistema con la capacidad de procesar cientos de programas concurrentemente. Se usará el termino MVS en lugar de MVS/XA para referirse al sistema operativo en el resto del trabajo, sólo se hará referencia específica al MVS/XA cuando así lo amerite.

3.3 ESTRUCTURA INTERNA DEL MVS

3.3.1 Interacción entre el MVS y programas de aplicación

El MVS es la conexión entre los programas de aplicación y el hardware. El MVS no solamente controla la ejecución de las aplicaciones, sino que también proveé interfaces entre el usuario y el hardware y soporta el uso del hardware por los programas de aplicación. Las principales interfaces son entre:

- el MVS y el hardware
- el hardware y el MVS
- el usuario y el MVS
- el MVS y el usuario

Interfaz MVS-Hardware: Existe sólo una forma en que el MVS puede comunicarse con el hardware, y es vía las instrucciones. La instrucción expresa que clase de servicio se requiere del hardware. Por ejemplo, para comenzar una operación de I/O se emite la instrucción SSCH ("Start subchannel"). El hardware puede indicar para ciertas instrucciones algunas condiciones particulares (p.e. si el resultado de una operación es cero) regresando un código de condición (CC).

Interfaz Hardware-MVS: Al igual que la interfaz anterior, sólo existe una forma posible en que el hardware puede requerir interacción con el MVS, y es por medio del concepto de interrupción. Cabe notar que no hay interacción directa entre el usuario y el hardware, siempre esta el MVS de por medio. De hecho esta es una razón por la cual es necesario un sistema operativo.

Interfaz Usuario-MVS: Hay varias formas en que el usuario puede requerir servicios del MVS:

- Llamada a Supervisor ("Supervisor Call SVC")
- Llamada a programa ("Program Call PC")
- Instrucciones de bifurcación ("Branch")

En principio es posible que cada usuario codifique las instrucciones que ejecutarían el servicio requerido en su propio programa. Sin embargo esto requiere gran cantidad de codificación porque todos los servicios del MVS necesitan una descripción detallada del requerimiento. Para estandarizar y simplificar la interfaz entre el usuario y el MVS, se dispone de MACROS (Macro-instrucciones) predefinidas. Las macros se usan como vía de comunicación entre el usuario y el MVS. Una macro inicia el requerimiento para un servicio específico.Por ejemplo la macro "GETMAIN" se usa para obtener memoria virtual.

Interfaz MVS-Usuario: La mayoría de los servicios del MVS, cuando acaban su trabajo no regresan directo al que llamó al servicio, sino que van al "Despachador" ("Dispatcher"). Esta es la parte del MVS que se encarga de que la tarea más importante (la de más alta prioridad) tome control inmediatamente. El Despachador cede control a esta tarea (sea cual sea) mediante la instrucción LPSW ("Load PSW") con lo cual cambia el flujo de ejecución, es decir, la CPU empezará a procesar la instrucción que se encuentre en la localidad de memoria a la cual apunta la PSW. (figura 3.5)

3.3.2 Componentes básicos del MVS

El MVS proveé diferentes servicios y estructuralmente está organizado en "Componentes", cada uno de los cuales es responsable de un cierto servicio. Cada componente tiene su propia estructura, estructura que se tratará con detalle en secciones separadas.

Los principales componentes del MVS son:

+ Manejador de tareas

+ Manejador de memoria
  - Manejador de memoria real
  - Manejador de memoria virtual
  - Manejador de memoria auxiliar

+ Manejador de programas

+ Supervisor de Entrada/Salida

+ Manejador de recuperación y terminación

+ Manejador de recursos del sistema

+ Subsistema de entrada de trabajos (JES)

Estos componentes tienen ciertas características en común: Cada componente es responsable de realizar requerimientos específicos y cada componente establece una interfaz para establecer comunicación entre el que requiere el servicio que este proporciona (que puede ser el usuario u otro componente del MVS), y el componente mismo.

Las interfaces básicamente pueden ser:

- Macros: Que piden el servicio.
- Bloques de control: Que se usan para pasar información al componente del servicio requerido y para que el mismo pase información de regreso al que lo solicitó. (figura 3.6)

Como se mencionó, cada componente en sí se representa por un bloque de control dentro del MVS. Este bloque de control es el punto de "anclaje" (punto de partida) principal de toda la información relacionada con este componente. Como ejemplo de esta información es el trabajo que se va a realizar, también representado por bloques de control.

Si existen múltiples requerimientos representados por bloques de control, estos pueden ser encolados para procesarlos uno a uno, dependiendo de la estructura del componente.

El componente en sí está formado de muchos "Módulos" simples, cada uno responsable de un trabajo específico dentro del contexto del componente.

En las subsecuentes secciones se estudiarán las estructuras internas de cada componente del MVS, haciendo referencia, cuando sea pertinente, a la interacción entre los diferentes componentes en el contexto global del sistema.

3.4 ESTRUCTURA DE UN ESPACIO DE DIRECCIONES

Conceptualmente, el espacio de direcciones del MVS consiste de 2 Gbytes de memoria virtual disponible para cada usuario. El espacio de direcciones está dividido en general en:

-El área de salvado prefijada ("Prefixed Save Area", PSA)
-Areas privadas
-Areas comunes

Cada usuario tiene su espacio de direcciones y por lo tanto acceso a las tres clases de áreas. El MVS/XA aisla efectivamente un espacio de direcciones de otros por medio de sus tabla de segmentos y tablas de páginas. Además por medio de las áreas comunes los usuarios pueden compartir datos y programas. De esta manera el MVS/XA balancea la necesidad de compartir recursos y de mantener la privacía de los usuarios.

Los módulos de programas, datos y bloques de control se ubican en el espacio de direcciones de acuerdo a sus características, tales como :

-Si pueden ser compartidos entre todos los espacios de direcciones.
-Si pueden ser paginados o deben estar siempre en memoria real (Fijos).
-Si deben residir por debajo de la línea de 16 Mbytes o donde sea.

El mapa de la figura 3.7 muestra las diferentes áreas de un espacio de direcciones. El espacio de direcciones del MVS/XA aparece como se muestra debido a que se buscó mantener la compatibilidad con el MVS/370. Casi todas las áreas existen debajo de la línea y tienen su área homóloga extendida arriba de la línea. Sin embargo, el MVS/XA trata a cada área con su correspondiente área extendida como a una sola. A continuación se describirán a grandes rasgos las características de cada área.

3.4.1 Area Prefijada de Salvado ("Prefixed Save Area", PSA)

La PSA ocupa de la primer página de memoria real (Direcciones x'000000' a x'00001000'); contiene información que le concierne tanto al hardware como al software, como las PSWs viejas y PSWs nuevas para cada tipo de interrupción, el código de interrupción, área de salvado de registros (Todas las localidades preasignadas de las que se habló en el capitulo 2) y apuntadores a los más importantes bloques de control, como la CVT.

La CVT es el punto de anclaje más importante del MVS, es siempre el punto de salida para encontrar cualquier bloque de control del MVS. Está apuntada por la PSA, debido a que siempre tiene que ser localizable: Si no se encuentra a la CVT, no se puede encontrar ningún otro bloque de control del sistema.

Para un sistema con un sólo procesador, la PSA ocupa la primer página de memoria real (Direcciones x'000000' a x'00001000'), es decir los primeros 4 Kbytes de memoria. En un sistema de multiproceso (con varios procesadores) cada procesador tiene su PSA y la accesa como si residiera en la primera página de memoria.

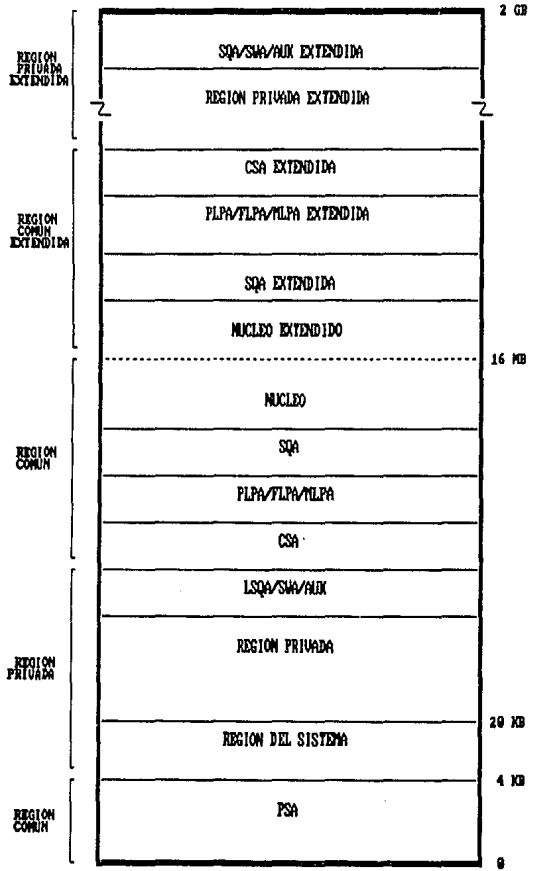


FIGURA 3.7
ESTRUCTURA DE UN ESPACIO DE DIRECCIONES EN MVS/XA

Esto se logra mediante el uso de un registro de ajuste que contiene una dirección de memoria real en frontera de página.

Cada procesador tiene un registro de ajuste con un valor distinto, de manera que cuando accesa la primera página de memoria, el hardware le suma automáticamente el valor de su registro de ajuste, con lo que realmente accesa una página de memoria real distinta de la primera, aunque para el software, se siguen accesando las direcciones 0-4096 (figura 3.8).

3.4.2 Areas Privadas

El área privada contiene módulos y datos no compartidos por otras espacios de direcciones. Consta de 5 secciones:

1.- Región del sistema: Es la única región del área privada que no tiene una contraparte arriba de la línea. Es usada por funciones del sistema cuando realizan trabajo para el espacio de direcciones. Estos trabajos corren bajo la TCB del RCT (que se verá más adelante).

2.- Región de usuario: Región de usuario extendida: Es la región en que los programas de usuario corren. Existen dos clases de región de usuario: la Virtual=Virtual (V=V) y la Virtual=Real (V=R). Las dos son mutuamente excluyentes.

Una región V=V puede ser desde 1 byte hasta todo el tamaño del área privada menos la LSQA,SWA,AUK y la región del sistema. Su tamaño puede ser limitado por el parámetro "REGION" en el JCL del "Job" que esté corriendo o por medio de "exits" que la instalación codifique.

Las Regiones reales (V=R) sólo pueden existir por debajo de la línea de 16 Mbytes. Cada dirección virtual corresponde a la misma dirección real. La memoria real para toda la región se aloja y fija cuando la región es creada, de manera que una región V=R es no paginable y no "swappeable".

Este tipo de regiones debe usarse sólo para "Jobs" que no puedan esperar el proceso de traer a memoria real una página (proceso de "page-fault") para ejecutar, es decir para "Jobs" muy críticos que necesiten respuesta muy rápida.

3.- Región de Usuario Autorizado por LLave ("Authorized User Key", AUK) y la región extendida ("Extended AUK"):

El área AUK de la región privada contiene datos que usa el sistema que se refieren a un usuario específico. Los recursos protegidos del usuario residen en esta área, como los bloques de control que representan "Data Sets" del usuario.

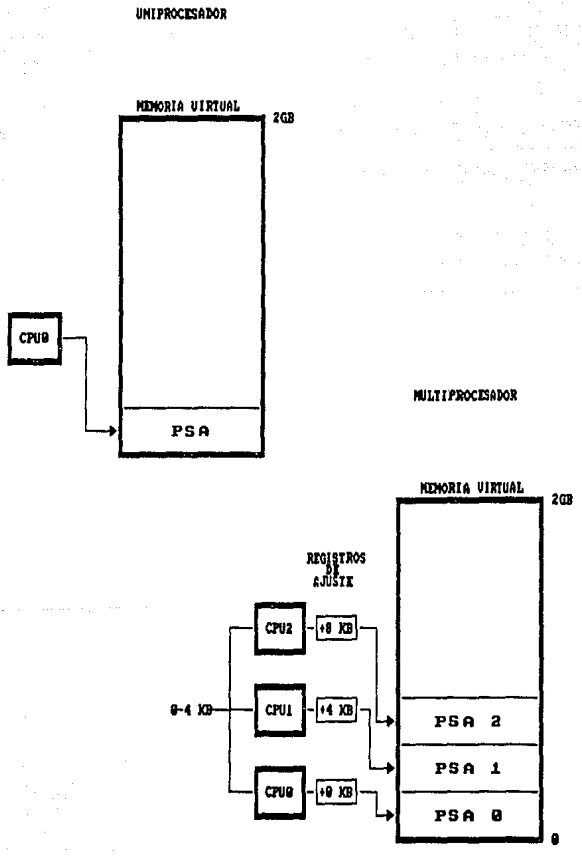


FIGURA 3.8

USO DE LOS REGISTROS DE AJUSTE EN UN MULTIPROCESADOR

Esta área también se conoce como los "subpools" 229 y 230 (como se verá más adelante, la memoria virtual está estructuralmente dividida en "subpools"). Esta área de memoria está protegida por la llave del usuario, lo cual significa que a menos que la llave de la PSW coincida, no se puede grabar en esta área. Además, el "subpool" 229 está protegido contra "Fetch", así es que si la llave no coincide, tampoco se puede leer.

4.- Area de trabajo del planificador ("Scheduler Work Area", SWA) y su área extendida ("Extended SWA")

La SWA contiene bloques de control que existen desde el inicio de un "Job" hasta el final de este. Contienen la forma interna de representación del JCL que acompaña al "Job". La información en SWA se crea cuando un "Job" se interpreta y se usa durante la iniciación y ejecución del mismo. La SWA es paginable y "Swappeable".

5.- Area local de colas del sistema ("Local System Queue Area", LSQA) y su área extendida ("Extended LSQA").

La LSQA contiene tablas y colas de bloques de control que son únicas a este espacio de direcciones en particular. Por ejemplo, aquí reside la Tabla de Segmentos del usuario y las Tablas de Páginas que "mapean" el área privada del espacio de direcciones. La LSQA también contiene todos los bloques de control que necesita la RCT. La LSQA es "swappeable" pero no paginable.

3.4.3 Areas Comunes

Las áreas comunes contienen información del sistema, como código de programas, bloques de control y áreas de datos. Es común a todos los espacios de direcciones, en el sentido de que esta área tiene las mismas direcciones virtuales en todos los espacios de direcciones. Las áreas comunes incluyen:

1.-Area de servicio común ("Common Service Area, CSA") y su área extendida ("Extended CSA").

La CSA es direccionable por todos los programas activos y es compartida por todos los usuarios "swappeados" dentro (Es decir activos). El uso más importante de la CSA es permitir la comunicación entre espacios de direcciones. La CSA contiene algunas páginas fijas y otras paginables. La cantidad total de bytes en la CSA y la CSA extendida se especifica como un parámetro en la biblioteca SYS1.PARMLIB.

2.-Area Empacada Paginable de Enlaces ("Pageable Link Pack Area", PLPA) y su área extendida ("Extended PLPA").

La PLPA contiene programas de control del MVS/XA como rutinas de SVC, métodos de acceso, algunos otros programas del sistema de "Solo Lectura" y programas de usuario seleccionados.

Como estos módulos son muy usados y el cargar la PLPA completa es un proceso lento, el MVS/XA salva su contenido de un IPL a otro. De hecho, esta área se carga durante el IPL sólo si se le indica que se haga, si no se le indica, toma el área guardada.

Como su nombre lo indica, la PLPA es paginable, sin embargo no se realiza ningún "page-out" físico (ninguna operación de I/O) debido a que su contenido no cambia. Las páginas de la PLPA que no se han usado recientemente pueden ser "robadas" (esto es, como no se modifica su contenido, en lugar de paginarse simplemente se marcan como "frames" disponibles cuando se requieren).

El espacio de la PLPA es alojado en páginas completas. El tamaño de la PLPA está determinado por el número de módulos incluidos, y una vez que se establece, no se expande dinámicamente.

3.- Area Empacada Fija de Enlaces ("Fixed Link Pack Area", FLPA) y su área extendida ("Extended FLPA").

Las páginas de la FLPA están fijas en memoria real. Contienen módulos que pudieran estar en la PLPA pero que requieren una respuesta extremadamente rápida (rapidez que viene debido al hecho de que están fijas en memoria).

Debido a que la FLPA está fija, reduce la cantidad de memoria real disponible para otros usos, así es que la selección de los módulos que residen en la FLPA debe hacerse con mucho cuidado, para poner en esta sólo los módulos que realmente requieren una respuesta tan rápida. El MVS/XA normalmente mantiene en memoria real los módulos de la PLPA más usados, debido a esto los candidatos más loables de incluirse en la FLPA son aquellos módulos que no son muy usados pero si requieren una respuesta rápida cuando se usen. La instalación determina los módulos que estarán en la FLPA en parámetros del sistema.

4.-Area Empacada Modificable de Enlaces ("Modified Link Pack Area", MLPA) y su area extendida ("Extended MLPA").

La MLPA se puede usar por módulos reentrantes de bibliotecas selectas de usuario o de sistema; actúa como una extensión de la PLPA, pero existe sólo durante una sesión del MVS (es decir, sólo de un IPL al otro).

En el momento en que se da IPL, hay que volver a especificar los módulos que se incluirán en la MLPA.

Los módulos de la MLPA son normalmente sólo para lectura. Debido a que el MVS busca en la MLPA antes que en la PLPA, normalmente se usa para probar los módulos candidatos para la PLPA antes de incluirlos definitivamente.

5.-Area de Colas del Sistema ("System Queue Area", SQA) y su área extendida ("Extended SQA")

Contiene Tablas y colas que se relacionan con todo el sistema. Por ejemplo, las tablas de páginas que definen a el área del sistema y al área común residen en SQA. Los contenidos de la SQA dependen de la configuración de la instalación y de los requerimientos de los "Jobs".

La cantidad de memoria asignada a la SQA se especifica en parámetros de la SYS1.PARMLIB. Si el MVS/XA necesita más memoria para la SQA toma memoria de la asignada para la CSA. Si al sistema se le acaba la memoria de CSA deja de crear espacios de direcciones. La SQA siempre está fija en memoria real.

6.-Núcleo ("Nucleus")/Núcleo Extendido ("Extended Nucleus").

El núcleo y el núcleo extendido guardan la parte residente del código del "Programa de Control del Sistema" ("System Control Program", SCP) del MVS/XA. Además de módulos del SCP el núcleo contiene PFTEs,Bloques de control para bibliotecas del sistema, "Bloques de Control de Unidades" ("Unit Control Blocks", UCBs) que representan dispositivos de I/O y rutinas de recuperación de errores. El núcleo y el núcleo extendido rodean la línea de 16 Mbytes del espacio de direcciones. Esto comprende el núcleo que utiliza al DAT para traducir sus direcciones ("DAT-ON Nucleus").

Existe también otra parte del núcleo que no utiza al DAT para convertir direcciones ("DAT-OFF Nucleus"), en este núcleo residen rutinas de recuperación de errores de hardware, que entran en funcionamiento cuando el DAT no puede operar. Este código reside siempre en memoria real y ocupa las localidades más altas posibles de esta. Este código no lo puede ver la memoria virtual.

3.4.4 Espacios de Direcciones de Componentes del MVS/XA

El "Planificador Maestro" ("Master Scheduler") es un componente del MVS/XA que interactúa con comandos del operador y con parámetros del sistema para iniciar ciertas funciones, funciones como crear Espacios de direcciones.

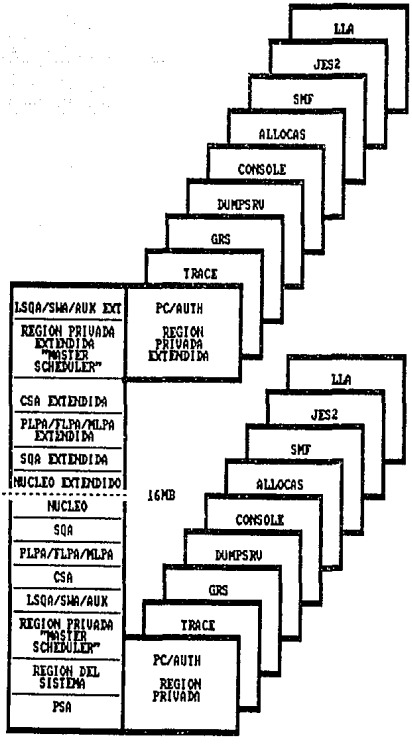


FIGURA 3.9

ESPACIOS DE DIRECCIONES DE COMPONENTES DEL MVS

Cuando el MVS/XA se inicializa en el IPL, el Espacio de direcciones del Planificador Maestro es el primer Espacio de direcciones que se crea. Después de él, otros espacios de direcciones de otros componentes clave del sistema se crean. Debido a que el Planificador Maestro es el primer espacio de direcciones creado, al inicializar sus áreas comunes, también se inicializan las de todos los espacios de direcciones.

Un componente del sistema puede ejecutar en su propio espacio de direcciones y dar servicio a otros usuarios. Teniendo cada componente su propio espacio de direcciones se logra reducir el consumo de áreas comunes. Si un componente tiene su propio espacio de direcciones, este se debe crear e inicializar de tal forma que sea capaz de manejar los requerimientos de otros espacios de direcciones.

Los siguientes componentes tienen su propio espacio de direcciones. Se listan en el orden en que se crean:

1.-Llamado de Programas/Autorización ("Program Call / Authorization"), Espacio de direcciones de PC/AUTH. Es el primer Espacio de Direcciones que se inicializa después del Planificador Maestro. Contiene las rutinas y tablas necesarias para establecer comunicación entre Espacios de Direcciones. Conforme se van inicializando otros Espacios de Direcciones, las rutinas de inicialización usan servicios de PC/AUTH para crear e inicializar sus propias tablas de "Memoria Cruzada".

2.-Rastreo del Sistema ("System Trace"), Espacio de direcciones de TRACE. Se encarga de rastrear (es decir de dejar un registro en una tabla) de eventos relevantes del sistema, tales como interrupciones.

3.-Serialización Global de Recursos ("Global Resource Serialization"), Espacio de Direcciones del GRS. Se encarga de serializar los recursos del sistema entre los procesadores.

4.-Servicios de Vuelco de Memoria ("Dumping Services"), Espacio de Direcciones de DUMPSRV, se encarga de proporcionar servicios necesarios para volcar el contenido de memoria a un archivo o a papel.

5.-Tarea de Comunicación ("Communication Task"), Espacio de Direcciones de CONSOLE. Se encarga de la comunicación con el operador, mandar mensajes y recibir respuestas de este.

6.-Alojamiento de Recursos ("Allocation"), Espacio de Direcciones de ALLOCAS. Se encarga de llevar control de los alojamientos de recursos a usuarios.

7.-Facilidades de Manejo del Sistema ("System Management Facilities"), Espacio de Direcciones del SMF. Se encarga de proporcionar servicios de monitoreo y control del sistema.

8.-Subsistema de entrada de "Jobs" ("Job Entry Subsystem"), Espacio de Direcciones del JES2. Se encarga de manejar la entrada, proceso y salida de "Jobs".

9.-Mira-al-lado de Listas de ligas ("Link List Lookaside"), Espacio de Direcciones del LLA, se encarga de mantener las listas de la LNKLST para eficientar las busquedas de módulos.

En la figura 3.9 se muestran los espacios de direcciones mencionados.

Existen 2 tipos diferentes de espacios de direcciones en MVS:

-Espacios de direcciones de funciones totales: Son todos los espacios de direcciones de usuarios y el espacio de direcciones del Planificador Maestro. Pueden ser creados como:

+ "Jobs" en lote ("Batch"),
+ Tareas arrancadas y
+ Usuarios de TSO.

Estos espacios de direcciones de funciones completas pueden utilizar cualquier servicio del sistema operativo.

-Espacios de direcciones de funciones limitadas: La mayoría de los espacios de direcciones del sistema son de este tipo, debido a que no pueden requerir cierta clase de servicios, como: Alojar "Data Sets" ó leer un JCL desde la biblioteca de procedimientos del sistema (SYS1.PROCLIB).

3.4.5 Bloques de Control de un Espacio de direcciones

Los dos bloques de control principales que representan un Espacio de direcciones son:

1.-"Bloque de Control del Espacio de Direcciones" ("Address Space Control Block", ASCB)

Existe una ASCB para cada Espacio de direcciones en el sistema. La ASCB es el punto de anclaje principal para toda la información relacionada con el Espacio de direcciones. La ASCB reside en la SQA, todas las ASCB's listas están encadenadas unas con otras. Una ASCB está lista si existe una unidad de trabajo en ese espacio de direcciones que se pueda ejecutar. Cada ASCB tiene una extensión, llamada ASXB

2.-"Bloque de Extensión del Espacio de Direcciones" ("Address Space Extension Block", ASXB)

La ASXB se localiza en la LSQA de cada Espacio de direcciones. Contiene apuntadores e información adicional que se usa para controlar al espacio de direcciones (figura 3.10).

A continuación se presentará la estructura interna de cada uno de los componentes del MVS.

3.5 MANEJADOR DE TAREAS

3.5.1 Conceptos generales

Unidad de Trabajo:

El MVS necesita alguna forma de identificar y manejar el trabajo en el sistema. Esto se lleva a cabo clasificando el trabajo en "Unidades de trabajo". Las unidades de trabajo son:

- Tareas ("Tasks")
- Requerimientos de Servicio ("Service Requests")
- Salidas Especiales ("Special Exits")

3.5.2 Tareas

Las tareas internamente se representan por medio de un bloque de control llamado "Bloque de Control de Tareas" ("Task control block", TCB). Una TCB representa una tarea ejecutándose dentro de un espacio de direcciones. Estas tareas pueden ser trabajos del usuario, o bien tareas del sistema. Las tareas del sistema son necesarias para crear y manejar al espacio de direcciones, a la vez que proporcionan soporte para el usuario. Todas las TCBs dentro de un espacio de direcciones se encadenan unas a otras en una cadena que recibe el nombre de "Cadena de Despacho", debido a que esta cola es la que recorre el Despachador para decidir a que TCB le cede el control a continuación.

Relacionada a cada TCB, existe siempre uno o más bloques de requerimiento ("Request Blocks", RBs). Las RBs representan la rutina que está activa en un momento determinado. La RB identifica al programa que toma control; en este bloque de control se guardan los registros del programa cuando este pierde control, es decir sirve como un área de salvado. La RB siempre contiene la PSW usada para dar control a la rutina que ejecuta bajo este par TCB/RB. Las tareas normalmente son interrumpibles y pueden perder control en cualquier momento. En este caso el sistema debe salvar el estatus (PSW y registros) de la tarea interrumpida, información que será usada posteriormente para regresar control a esta (figura 3.11).

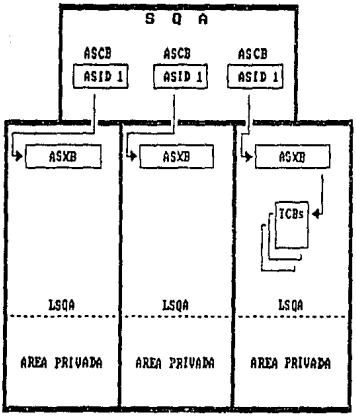


FIGURA 3.10

BLOQUES DE CONTROL MAS IMPORTANTES PARA CONTROL DE ESPACIOS DE DIRECCIONES

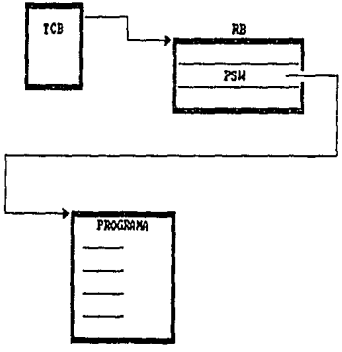


FIGURA 3.11

FUNCION DEL BLOQUE DE CONTROL "REQUEST BLOCK" (RB)

Se pueden construir dos tipos de RBs, dependiendo del requerimiento que se haga:

Bloque de Requerimiento de Programa ("Program Request Block", PRB): Se construye para representar el programa que está siendo ejecutado, como un programa de aplicación.

Bloque de Requerimiento de Supervisor ("Supervisor Request Block", SVRB): Se construye para representar una rutina de supervisor, como una rutina de SVC ejecutándose para la tarea.

3.5.2.1 Creación de Tareas

Las TCBs se crean como respuesta a la macro "ATTACH" (Conecta). Ejecutando esta macro, una rutina de usuario (corriendo en estado problema) o una rutina del sistema (corriendo en estado supervisor) causa que el supervisor comience la ejecución del programa especificado en la macro "ATTACH" como una "Subtarea" o una tarea "Hija" de la tarea que creó la TCB. La subtarea puede usar ciertos recursos ya alojados a la tarea "Madre".

La macro "ATTACH" causa una interrupción de supervisor (es decir, una interrupción de SVC). La rutina de interrupciones de primer nivel (FLIH) le da control a la rutina de segundo nivel (SLIH) de esta SVC para realizar el servicio requerido. La rutina de "ATTACH" hace lo siguiente:

-Obtiene memoria para una nueva TCB
-Coloca en la nueva TCB información necesaria para controlar la subtarea.
-Encadena la nueva TCB en la cadena de TCBs para ese espacio de direcciones.
-Bifurca a rutinas del Manejador de programas para localizar el primer programa a ser ejecutado por la nueva tarea y si es necesario transfiere el programa de disco a memoria.

Con la macro "DETTACH" (Desconecta) se borra una TCB creada por una TCB que ejecutó la instrucción "ATTACH".

La macro "CHAP" (Cambia prioridad, "CHAnge Priority") sirve para cambiar la prioridad de despacho de las tareas dentro de un espacio de direcciones.

3.5.2.2 Tareas Comunes de un espacio de direcciones

Existen varias TCBs que representan tareas que tienen todos los espacios de direcciones. Estas tareas son las siguientes:

La "Tarea de control de la región" ("Region Control Task", RCT) es la tarea de más alta prioridad dentro de un espacio de direcciones, es la primer tarea que se crea y su función es:

-Inicializar al espacio de direcciones.
-Encadenar abajo de ella a las tareas de Dump y "Control de Tarea Arrancada" ("Started Task Control", STC).
-Preparar al espacio de direcciones para "swap-in" ó "swap-out".
-Terminar al espacio de direcciones.

Todas las tareas de un espacio de direcciones son subtareas de la RCT. La TCB de la RCT se apunta desde la ASXB y esta TCB apunta a la siguiente TCB dentro del espacio de direcciones.

La tarea de "Dump" (Vuelco de memoria) siempre es la que sigue de la RCT en la cola de despacho del espacio de direcciones. Su función es procesar requerimientos de "Dump" asíncronamente a la tarea que falla. Un "dump", lo obtiene automáticamente el sistema o lo pide el operador cuando se detecta un error en el sistema. El "dump" consiste en un vuelco hexadecimal de la memoria asignada a un espacio de direcciones. Este vuelco se guarda en los "Data Sets" llamados SYS1.DUMPxx en donde xx es un número del 00 al 99.

La tarea de "Control de Tarea Arrancada" ("Started Task Control", STC) se encarga de inicializar el Espacio de direcciones del sistema y de procesar los comandos de "START" (Arranca) y "LOGON" (Firmarse). Estos comandos son la forma normal de crear espacios de direcciones. Solo para el caso de espacios de direcciones del sistema, el Planificador Maestro lo pide directamente.

El STC revisa el tipo de espacio de direcciones que se debe crear y dependiendo de esto, hace lo siguiente:

- Para espacios de direcciones limitados, la STC pasa control a la rutina de inicialización del espacio de direcciones del sistema. Esta rutina de inicialización hace procesos que dependen del componente que se trate y entra en un "Wait" que no termina. Esto quiere decir que el STC nunca toma el control de nuevo.

-Para cualquier otro tipo de espacio de direcciones, la STC usa un iniciador/terminador como una subrutina para completar el proceso del "START", "MOUNT" ó "LOGON". Durante este proceso, la tarea especificada en el comando será creada (vía la macro "ATTACH"). Cuando la tarea creada termina y regresa el control a la STC, la STC hace limpieza de las áreas de memoria ocupadas y libera al espacio de direcciones.

Los iniciadores no son otra cosa que espacios de direcciones pre-creados que siempre existen en el sistema y que el JES asigna a los "Jobs" en lote (Batch) para ejecutar.

En la figura 3.12 se muestra la estructura de diferentes tipos de espacios de direcciones (espacios de direcciones de "Job" Batch, de Tarea Arrancada y de usuario de TSO).

Como se puede observar en la figura 3.13 la cadena de TCBs está anclada a la ASXB la cual tiene apuntadores a la primera TCB y a la última TCB y cada TCB apunta a la siguiente y a la anterior TCB. En la ASCB está un apuntador a la primera TCB lista a ejecutarse en este espacio de direcciones.

Adicionalmente a esto las TCBs tienen apuntadores a una "Tarea hermana" ("Sister Task", NTC), a una "Tarea Madre" ("Mother Task", OTC) y a una "Tarea Hija" ("Daughter Task", LTC) como se muestra en la figura 3.14. Una tarea corriendo bajo una TCB que efectúa una macro "ATTACH" se convierte en TCB "Madre" de la que creó y recíprocamente esta última es la TCB "Hija" de la que la creó. Varias TCBs creadas por la misma TCB madre son TCBs "Hermanas". Por ejemplo, en la figura 3.14 se puede ver que la "STC" y la tarea de "DUMP" son hermanas, por que ambas son creadas por la "RTC".

Existe otro bloque de control llamado "Entrada de Contenidos del Directorio" ("Contents Directory Entry", CDE) que contiene el nombre del programa en control, su dirección de entrada y los atributos del módulo (como por ejemplo el AMODE y el RMODE que se verán más adelante).

3.5.2.3 Terminación de Tareas

El objetivo del proceso de terminación de una tarea es liberar los recursos asociados a una tarea cuando termina su ejecución. El proceso de terminación se efectúa por alguna de las siguientes causas:

- Terminación Normal de la Tarea: Cuando el último programa de una tarea acaba de ejecutar, emite una macro "RETURN" (Regresa) que eventualmente le cede control a un componente del MVS llamado "Manejador de terminación de Recursos" ("Resource Termination Manager", RTM).

- Terminación Anormal de la Tarea: Cuando un programa decide que hubo un error, emite la macro de "ABEND", ó terminación anormal ("Abnormal End", ABEND).

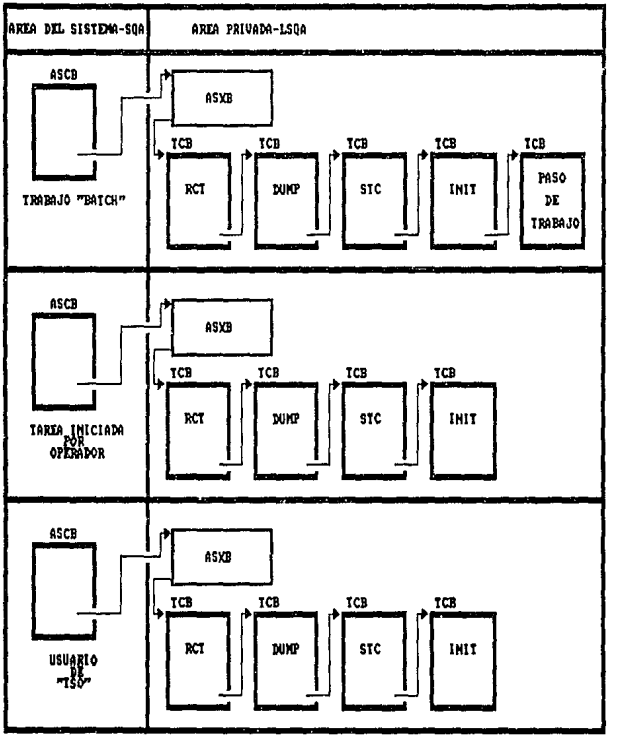


FIGURA 3.12

TAREAS COMUNES DE TODOS LOS ESPACIOS DE DIRECCIONES

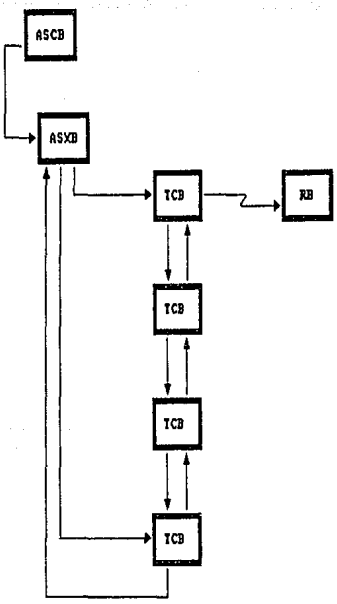


FIGURA 3.13
COLA DE DESPACHO DE TAREAS

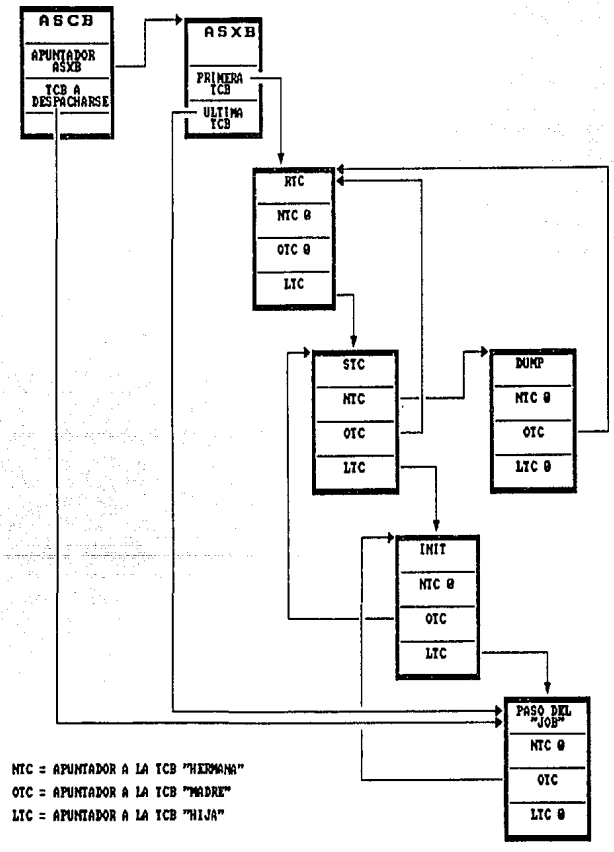


FIGURA 3.14
RELACIONES ENTRE LAS TCBs

- Ejecución de la macro "DETTACH"

Para la terminación normal o anormal, el proceso de terminación de tarea es llamado vía una SVC 13 (macro de "Abend"). Para terminación normal, se prende un bit en el registro 1 antes de llamar a esta SVC.

3.5.3 Requerimientos de servicio

Representan un requerimiento para ejecutar una rutina de servicio en un espacio de direcciones generalmente diferente al que se está ejecutando . Internamente se representan por un bloque de control llamado "Bloque de Requerimiento de Servicio" ("Service Request Block", SRB). Las SRBs se crean cuando un espacio de direcciones detecta un evento que afecta a un espacio de direcciones diferente; Las SRBs son de hecho un mecanismo de comunicación entre Espacios de direcciones. Las SRBs tiene varias características especiales:

- Suspendibles, Una SRB es suspendida si por ejemplo tiene que esperar a que se complete una operación de I/O para paginar. A una SRB suspendida se le conoce como SSRB ("Suspended SRB").

- Limitadas en funciones, debido a que no pueden emitir instrucciones de SVC.

- Corren habilitadas para interrupciones, en modo supervisor y con llave cero.

Sólo las rutinas corriendo en estado supervisor y con llave 0 pueden crear una SRB. Estas rutinas obtienen memoria para inicializar el bloque de control con información tal como la identidad del espacio de direcciones en el cual reside la rutina que se quiere ejecutar y la dirección de entrada de esta rutina.

Una vez hecho esto, la rutina encargada de crear la SRB ejecuta la macro "SCHEDULE" (Planifica) y le indica, como parámetros, si la SRB tiene una prioridad "Global" (es decir, que afecte a todo el sistema) o una prioridad "Local". Es responsabilidad de la rutina que emite la macro "SCHEDULE" el liberar la SRB si ya no se necesita

La SRB contiene una dirección de entrada a la rutina que se quiere correr ("Entry Point Address"). Se le cede control a la rutina cargando la PSW con esta dirección. (con la instrucción LPSW). De hecho la SRB es usada sólo para darle control a la rutina, después de esto ya no existe para el sistema; tiene una vida muy corta.

Las SRBs con prioridad global tienen más alta prioridad que cualquier otra unidad de trabajo , sin importar en que espacio de direcciones se ejecutarán realmente. Las SRBs con prioridad local tienen una prioridad igual a la del espacio de direcciones en el que serán ejecutadas, pero más alta que cualquier otra TCB dentro de ese espacio de direcciones. La asignación de prioridad global o local depende de la "Importancia " del requerimiento. Por ejemplo, las SRBs para interrupciones de I/O se emiten con prioridad Global, para minimizar retrasos en el I/O.

Las SRBs tienen una característica muy especial, si una rutina representada por una SRB se interrumpe recibirá control después de que la interrupción se procesa. En contraste, si una rutina representada por una TCB se interrumpe, el control se le regresa al Despachador, el cual decide cuál de las tareas listas para ejecutarse se despacha a continuación.

Una SRB puede ejecutarse concurrentemente en un espacio de direcciones diferente al de la tarea que creo la SRB (es decir, la tarea que emitió la macro de "SCHEDULE"). Esto significa, entre otras cosas, que una SRB sirve como medio de comunicación asíncrona entre espacios de direcciones. Esta comunicación mejora la disponibilidad de recursos en un ambiente de multiprocesamiento.

Como un ejemplo, considérese que cuando un espacio de direcciones A está ejecutando, ocurre una interrupción de I/O requerida por una operación que se completó y que se inició por el espacio de direcciones B. El FLIH de interrupciones de I/O colecta la información necesaria acerca de la interrupción, crea y emite una SRB para controlar el procesamiento final de la operación de I/O concluida. Entonces, el FLIH arranca cualquier otro requerimiento pendiente para el "Path" de I/O que usó la operación que acaba de completar y puede, entonces, aceptar cualquier interrupción pendiente adicional. El construir una SRB permite un reuso mucho más rápido del "path" de I/O y menos tiempo de dashabilitación del procesador a interrupciones de I/O.

La SRB identifica la rutina que procesa el requerimiento de I/O concluido y el espacio de direcciones en el que la rutina se debe ejecutar. En el ejemplo anterior, la SRB se debe ejecutar en el espacio de direcciones B, debido a que este espacio de direcciones requirió la operación de I/O.

3.5.4 Sincronización de Eventos

La sincronización de eventos es el medio por el cual una tarea puede esperar a que cierta actividad termine para reanudar su ejecución. Un ejemplo de esto es cuando se espera a que termine una actividad de I/O, en el caso en que

un programa requiere esperar que se lean los datos que necesitan para reanudar su proceso. El programa hace esto emitiendo una macro "WAIT" (Espera) lo que provoca que se regrese control al Despachador. Tan pronto como los datos requeridos estén disponibles, el Supervisor de I/O notificará a la tarea de esto.

La sincronización es una técnica de apoyo a la multiprogramación que se practica entre tareas de el mismo espacio de direcciones, entre diferentes espacios de direcciones y entre tareas y servicios de I/O o de supervisor.

El proceso de sincronización consiste en:

- Poner a una tarea en estado de espera ("Wait") hasta que se complete cierto evento.

- Informar a la tarea que está esperando que se completó el evento.

La forma más simple y común de sincronización de eventos se conoce como el proceso "WAIT/POST" (Espera/Avisa). Este proceso se lleva a cabo a través de las macros: "EVENTS", "WAIT" y "POST".

Las macros "WAIT" y "EVENTS" se usan para indicar que una tarea no puede seguir procesando hasta que uno (Para el caso de "WAIT") o más eventos (Para el caso de "EVENTS") se completen. La macro "POST" se usa para avisar que un evento se completó.

Se utiliza un bloque de control llamado "Bloque de Control de Evento" ("Event Control Block", ECB) como área de comunicación. Este bloque de control lo proveé la tarea que esperará en el momento de ejecutar la macro "WAIT" o "EVENTS", y posteriormente la usa la macro "POST" para localizar a la tarea que espera.

La tarea que espera guarda la dirección de la ECB o la lista de ECBs en el registro 1. Cuando se ejecuta la macro "WAIT" se ejecuta una SVC (SVC 1), la cual incrementa un "Contador de Espera" localizado en la RB de la tarea que emitió la macro ("Wait Count"), este contador es el número de eventos por el cual espera la tarea. Además se prenden en la TCB unos bits que indican que la tarea no es despachable, esto es, que no es elegible por el Despachador para darle control (figura 3.15).

La rutina que se encarga de realizar el evento por el cual está esperando la tarea es la responsable de avisarle que este evento está terminado vía la macro "POST" a la cual hay que indicarle la dirección de la ECB para que la "Posteé". La macro de "POST" se expande en una SVC que se encarga de

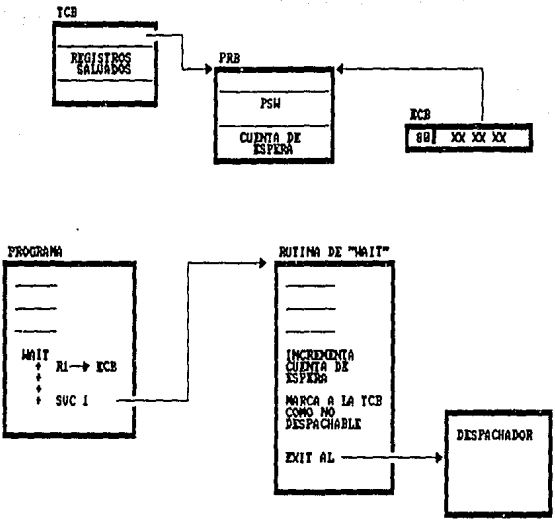


FIGURA 3.15

MECANISMO DE ESPERA ("WAIT") DE TAREAS

decrementar en uno el contador de espera de la RB. La tarea se hace despachable cuando este contador se hace cero. Si la tarea que espera no pertenece al mismo espacio de direcciones de la rutina que emite el "POST" se emite una SRB, y efectúa el "Posteo" en el espacio de direcciones de la tarea que espera. Este tipo de "posteo" es restringido a programas autorizados (el "postear" a la ECB no es más que cambiar el contenido del primer byte de x'80' a x'40', es decir una ECB está posteada si contiene un x'40'). En la figura 3.16 se esquematiza este proceso.

3.6 MANEJO DE PROGRAMAS

3.6.1 Atributos de Módulos de Carga

Las características de ejecución de un módulo de carga son controladas por sus atributos. Estos atributos son especificados generalmente en la etapa de "linkedición". Estos atributos son:

3.6.1.1 Atributos de Reutilización

Los atributos de reutilización se refieren al hecho de que la misma copia de un módulo pueda usarse por más de una tarea concurrentemente o bien, por una sola tarea a la vez. Un módulo puede ser "Reentrante" ,"Reusable Serialmente" o "Refrescable". Si no se especifica ninguna de estas características se toma por omisión que el módulo no es reusable y que se tiene que traer de disco una copia nueva cada vez que se ejecuta el módulo.

-Módulo Reentrante:

Un módulo que es reentrante puede ser ejecutado por más de una tarea a la vez; esto es, una tarea puede empezar a ejecutar el módulo antes de que otra acabe de ejecutarlo. La condición para que esto sea cierto es que el módulo no se puede modificar a sí mismo, es decir no puede guardar ni cambiar valores dentro de su código. Si necesita guardar algo debe hacerlo en un área de salvado o en un bloque de control ajeno al módulo (figura 3.17(A)).

-Módulo Reusable Serialmente :

Un módulo reusable serialmente puede ejecutarse por una sola tarea a la vez, es decir que cualquier tarea que quiera ejecutarlo tiene que esperar a que la tarea anterior termine de ejecutarlo. Este tipo de módulos debe restaurar cualquier parte del mismo que haya cambiado, antes de terminar su ejecución. Ver figura 3.17(B).

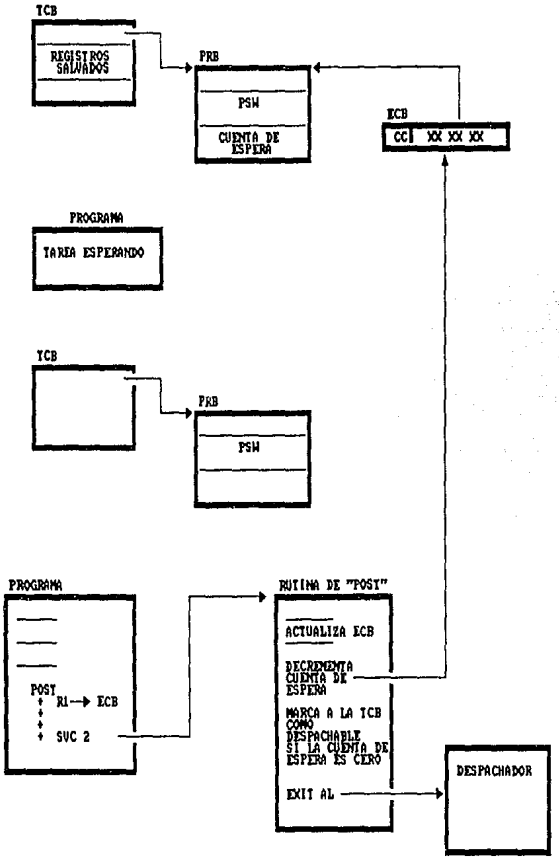


FIGURA 3.16

MECANISMO DE "POSTEO" DE TAREAS

-Módulo Refrescable:

Un módulo refrescable es aquel que puede ser reemplazado por una copia nueva del mismo módulo durante su ejecución sin afectar a esta. Esto implica también que este tipo de módulos no se pueden modificar a ellos mismos. Ver figura 3.17(C).

3.6.1.2 Atributo de Direccionamiento (A-MODE)

Este atributo indica si el módulo direcciona con 24 o 31 bits.

3.6.1.3 Atributo de Residencia (R-MODE)

Indica en que parte de la memoria virtual se puede cargar el módulo. Es decir, abajo de la línea de 16 Mbytes o donde sea.

3.6.2 Facilidad de Programas Autorizados("Authorized Program Facility", APF)

La facilidad de APF permite limitar el uso de servicios de sistema delicados o bien de rutinas de usuario. Los programas autorizados son aquellos que ejecutan en:

- Estado supervisor o
- Llave de sistema 0 a 7 o
- Programas que sean parte de una biblioteca autorizada por APF.

Las bibliotecas se autorizan incluyendo su nombre en el miembro IEAAPF00 de la SYS1.PARMLIB.

3.6.3 Localización de Módulos de Carga

Los módulos siempre residen en bibliotecas, sin embargo para ejecutarse deben estar en memoria virtual.

El control de cualquier módulo cargado en memoria virtual se lleva mediante el bloque de control "Elemento Descriptor de Contenido" ("Contents Descriptor Element", CDE), el cual tiene información concerniente al módulo, tal como su dirección de entrada y los atributos del mismo. Estos CDEs se enlazan en cadenas dependiendo de el área en que residan. Las diferentes áreas en las que puede residir un módulo son:

- La LPA ("Link Pack Area"): La LPA como se mencionó consiste de dos partes: la PLPA y la FLPA. La PLPA contiene rutinas de SVC, todos los módulos de métodos de acceso y los módulos más frecuentemente usados.

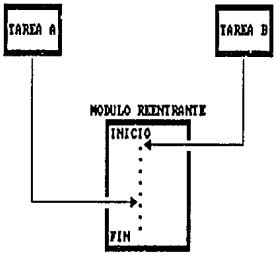


FIGURA 3.17(A)
EJEMPLO DE MODULO REENTRANTE

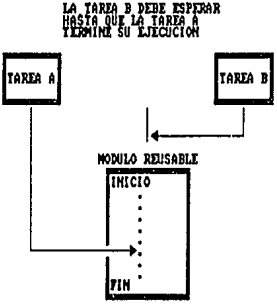


FIGURA 3.17(B)
EJEMPLO DE MODULO REUSABLE

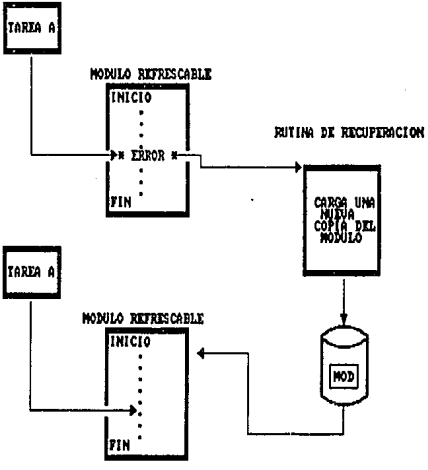


FIGURA 3.17 (C)
EJEMPLO DE MODULO REFRESCABLE

Todos estos módulos deben ser reentrantes. La FLPA contiene a módulos que no van a ser paginados y que van a ser frecuentemente usados.

Cuando se efectúa un IPL con opción de Borrar la LPA ("Clear LPA", CLPA) todos los módulos de la biblioteca SYS1.LPALIB son cargados en memoria virtual y se forza a que hagan un "page-out". Si se da IPL sin esta opción la PLPA se forma a través de los "data sets" de paginación. La FLPA siempre se carga en el IPL, los módulos que forman parte de esta se lista en el miembro IEAFIX00 de la SYS1.PARMLIB.

Los módulos de la MLPA se cargan siempre en el IPL pero pasan a formar parte de la PLPA. Estos módulos se listan en el miembro IEALPA00 de la SYS1.PARMLIB.

- Las "Areas de Paquetes de Job" ("Job Pack Area", JPA). Existe una JPA por cada paso de un "Job" (es decir, por cada programa distinto dentro de un "Job"). La JPA es la parte de el área privada de cada espacio de direcciones en donde los programas se cargan cuando se requieren. Si un módulo no está en la LPA, se carga de una biblioteca al área privada cuando se requiere. Esta copia del módulo está disponible a cualquier tarea de este paso del "Job". Si el módulo es reentrante o reusable esta copia satisfacerá cualquier requerimiento posterior por este módulo. Si no es así, se traerá una nueva copia del módulo cada vez que este sea requerido.

- Las bibliotecas, que son "Data Sets" particionados. La biblioteca en la que reside un programa se especifica en el JCL del "Job" que se ejecuta, sin embargo la biblioteca que se toma por omisión para buscar un módulo es la SYS1.LINKLIB. A esta biblioteca se le "Concatenan" (es decir se le encadenan o agregan) otras bibliotecas de usuario o del sistema. Todas estas bibliotecas concatenadas se accesan con el nombre genérico de SYS1.LINKLIB. Estas bibliotecas se listan en el miembro LNKLST00 de la SYS1.PARMLIB.

Las bibliotecas privadas que se usan durante un paso de trabajo se pueden especificar de varias formas:

-Una biblioteca para todo el "JOB" ("JOBLIB") o una biblioteca para un paso del "Job" en particular ("STEPLIB").

-Mediante el parámetro "TASKLIB" de la macro "ATTACH".

-Mediante el "Bloque de Control de Data Set" ("Data Set Control Block", DSCB) que se especifica en las macros "ATTACH","LINK", "LOAD", "XCTL" (que se verán más adelante).

3.6.4 Macro Instrucciones de Manejo de Programas

La macro "Conecta" ("ATTACH") como se vio, causa que el supervisor construya una nueva tarea y le indica la dirección de entrada a la que se le dará control cuando la tarea sea despachada, crea una TCB y el programa continúa ejecutando después de ejecutada la macro. El que ejecuta la macro debe especificar la dirección de entrada del módulo. Si esta dirección de entrada no se puede encontrar, la subtarea es terminada anormalmente.

El parámetro "TASKLIB" permite al usuario indicar la biblioteca en la que el manejador de programas debe buscar primero para cualquier requerimiento de manejo de programas que haga la subtarea.

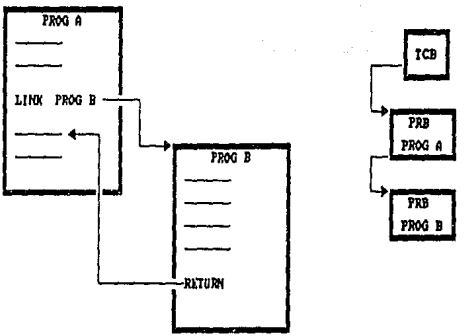
La macro "Liga" ("LINK") se usa para pasar control a un punto de entrada (una dirección particular a la que se le da un nombre) en otro módulo de carga. Al pasar control, la tarea que ejecutó la macro pierde automáticamente control. El módulo llamado toma control a través del supervisor (es decir, la instrucción LPSW con la que se transfiere control no se ejecuta bajo la TCB de la tarea que está ejecutándose sino bajo una SVRB que corresponde a una rutina de supervisor invocada a través de una SVC). Cuando el programa que se llamó acaba de ejecutarse, se regresa control al programa que lo llamó, también vía el supervisor, a la siguiente instrucción de la instrucción "LINK". Una nueva PRB es creada debajo de la PRB del programa que emite la macro "LINK". No se crea una TCB debido a que no se crea una nueva tarea.

La Macro "Carga" ("LOAD") sirve para cargar un módulo de carga en memoria virtual, si es que no se tiene ya una copia usable del él en memoria. Cuando se ejecuta esta macro, no se pasa control al módulo, si no que se regresa la dirección de entrada en el registro 0 y la longitud del módulo en el registro 1.

La macro "Transfiere Control" ("XCTL") sirve para dar control a un módulo cargado en memoria virtual y provoca que el módulo que ejecuta la macro sea lógicamente removido de la tarea activa. Cuando el programa al que se transfirió control acaba su ejecución, no se regresa el control al módulo que lo llamó. Una nueva PRB se crea para este módulo sustituyendo a la PRB del módulo que lo llamó (es por esto que no se regresa control al módulo que llama). Ver figura 3.18.

La macro "Identifica" ("IDENTIFY") sirve para agregar un punto de entrada a un módulo que se encuentre en memoria virtual. Este punto de entrada será identificado con un nombre hasta que este módulo se borre de memoria virtual.

LINK



XCTL

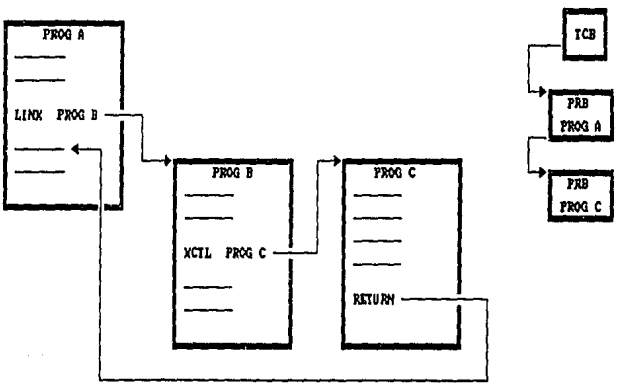


FIGURA 3.18
DIFERENCIA ENTRE "LINK" Y "XCTL"

La macro "Borra" ("DELETE") cancela el efecto de una macro "LOAD" previa para un módulo de carga específico. La memoria virtual que ocupa el módulo de carga se libera.

La macro "Sincroniza" ("SYNCH") hace posible que un programa corriendo en estado problema tome una salida síncrona a otro programa. Se crea una PRB para el programa que se quiere correr, se encola como la RB actual, y se le da control. La macro "SYNCH" no carga el módulo, este debe estar ya presente en memoria virtual.

3.7 SERIALIZACION DE RECURSOS

En un sistema de multiprogramación uno de los mayores problemas que se presentan es el hecho de que es posible que diferentes usuarios puedan accesar los mismos datos simultáneamente. Realmente no hay problema si todos ellos accesan los datos sólo para leerlos, sin embargo , si uno de ellos modifica los datos, los resultados pueden ser impredecibles debido a que los datos leídos por un usuario están siendo modificados por otro, lo que hace inválidos los datos para el primero. Para resolver esto el MVS/XA cuenta con dos mecanismos de "Serialización" llamados:

-Encola/Desencola/Reserva ("ENQUEUE/DEQUEUE/RESERVE")
-Cerraduras ("LOCKING")

Una característica importante del MVS/XA es la serialización de recursos "Reusables Serialmente" entre múltiples usuarios; usuarios que pueden ser programas de aplicación o funciones del sistema.

La serialización de recursos puede requerirse entre:

-Rutinas en el mismo espacio de direcciones
-Rutinas en diferentes espacios de direcciones
-Procesadores diferentes

En pocas palabras se puede decir que la filosofía de serialización del MVS/XA consiste en avisar al que requiere cierto recurso si el mismo esta en uso o no.

Los recursos que se quieren serializar pueden ser cualquier cosa dentro del sistema: una TCB, una parte del código, un área de salvado, etc.

La serialización puede ser "Compartida" ó "Exclusiva":

-Exclusiva: El recurso se da a un sólo usuario, no se permite el uso de este recurso por otros usuarios. Este tipo de serialización es necesaria cuando se va a cambiar el contenido del recurso.

-Compartida: El recurso se da a un sólo usuario, pero otros usuarios también pueden accesar el recursos al mismo tiempo. Este tipo de serialización se usa principalmente cuando el recurso no va a cambiarse. Si el recurso está en cierto momento como compartido, un usuario que lo requiera como exclusivo tendrá que esperar a que todos los demás dejen de usar el recurso para tomarlo.

Una forma en la que algunos sistemas operativos anteriores implantaban la serialización era deshabilitar la PSW a interrupciones durante todo el lapso de tiempo que se requiriera el recurso, lo cual permitía que ninguna otra rutina tomara control y pudiera usar el recurso. Este tipo de serialización tenía sus inconvenientes como el hecho de que interrupciones importantes (como de "Machine Check") no pudieran ser atendidas, además de que teniendo varios procesadores esto realmente no resuelve el problema de serialización.

Los mecanismos con los que cuenta el MVS/XA son:

-Encola/Desencola/Reserva ("ENQUEUE/DEQUEUE/RESERVE").

Este mecanismo se usa para proteger recursos que son compartidos entre los usuarios. El mecanismo de Encola/Desencola ("ENQUEUE/DEQUEUE") previene que un recurso se altere concurrentemente por dos usuarios. Para recursos que pueden ser usados por programas ejecutando en el mismo sistema se usan las macros de "ENQUEUE" y "DEQUEUE" y para recursos que están compartidos entre diferentes sistemas se usan las macros "RESERVE" y "DEQUEUE".

El usuario emite una macro "ENQUEUE" dando como parámetro el nombre del recurso al cual se quiere encolar. Si otro usuario quiere usar el recusro, también emite una macro "ENQUEUE" contra el mismo recurso. El requerimiento que se encoló primero accesa primero el recurso y si la macro se emitió con carácter de "Exclusivo" el segundo requerimiento se detiene hasta que el primero deja de usar al recurso. Cuando se acaba de usar el recurso se emite la macro "DEQUEUE" y se quita el requerimiento de la cola contra este recurso.

-Cerraduras ("Locks")

El uso de cerraduras serializa el uso de recursos del sistema entre las rutinas de supervisor y entre los diferentes procesadores de un sistema. Una cerradura es simplemente un campo en memoria que indica si un recurso está siendo usado y quién lo está usando.

Los mecanismos de "LOCKING" y "ENQUEUE/DEQUEUE/RESERVE" no son efectivos automáticamente, si no que tienen que usarse como una convención de programación.

Para que el mecanismo de encolamiento funcione, se debe emitir una macro "ENQUEUE" contra el nombre del recursos, y cuando ya no se vaya a usar este se debe emitir una macro "DEQUEUE".

Para que el mecanismo de "LOCKING" funcione, se debe pedir un "LOCK" y mantenerlo al entrar a cualquier rutina protegida por "LOCKS". Después de regresar de esta rutina se debe liberar el "LOCK".

La codificación de las rutinas de "ENQ" y "DEQ" es transparente para el programador de aplicaciones debido a que esto se hace como parte de la inicialización de los "Jobs". Los requerimientos del programa como "Data Sets" se especifican en el JCL. Usando esta información el iniciador aloja los "Data Sets" y los dispositivos emitiendo "ENQUEUEs" a los "Data Sets" o a los dispositivos antes de que se le pase control a la nueva tarea. Cuando se acaba la ejecución de un paso del trabajo, el terminador se encarga de emitir los "DEQUEUEs" para liberar los recursos.

3.8 SERIALIZACION GLOBAL DE RECURSOS

Las macros de "ENQ/DEQ" Y "RESERVE" son usadas en MVS/XA para proteger a los recursos reusables contra alteraciones simultáneas, sin embargo cuando los datos se comparten entre diferentes sistemas este proceso se hace más complejo. Para hacerse cargo de esto, existe un espacio de direcciones del sistema llamado "Serialización Global de Recursos" ("Global Resource Serialization", GRS) que extiende esta capacidad de serialización a los sistemas de un complejo.

3.8.1 Funciones de las Macros "ENQ/DEQ"

La macro "ENQ" pide al sistema operativo que se asigne un recurso a la tarea activa. El sistema operativo determina el estado del recurso y hace lo siguiente:

-Si el recurso está disponible, el MVS atiende el requerimiento y regresa el control a la tarea activa.

-Si el recurso ya ha sido asignado a otra tarea, el MVS retrasa el requerimiento colocando a la tarea activa en "WAIT" hasta que el recurso se hace disponible.

-Regresa un código de retorno indicando el estatus del recurso.

-Provoca un "ABEND" (Terminación anormal) de la rutina que pidió el recurso si esta pidió el recurso incondicionalmente y este no está disponible.

Cuando el estatus del recurso cambia de tal forma que la tarea qu espera pueda tomar control, esta es sacada de la condición de "WAIT" y se pone en condición de "Lista" ("READY") para ejecución .

La macro "DEQ" es usada para liberar al recurso que fué obtenido por medio de la macro "ENQ"; Si una tarea termina sin haber liberado al recurso, el MVS libera este automáticamente.

Las reglas para usar debidamente a las macros "ENQ/DEQ" son:

-Todos deben usar "ENQ" y "DEQ".
-Todos deben usar el mismos nombre de recurso para el mismo recurso.
-Todos deben ser consistentes con el protocolo de "ENQ/DEQ".

3.8.2 Tipos de Recursos

Desde el punto de vista del GRS existen dos tipos de recursos:

- Recursos Locales:

Son recursos accesibles por un sólo sistema. Se identifican en las macros "ENQ/DEQ" por medio del parámetro "SCOPE=SYSTEM" (Alcance=Sistema). Un recurso local sólo puede ser reconocido y serializado dentro de un sólo sistema operativo.

- Recursos Globales:

Los recursos Globales residen en DASD y son accesibles por más de un sistema operativo.

Anteriormente, sin los servicios provistos por el GRS el único medio de serializar un recurso gobal era por medio de la instrucción "RESERVE" de hardware, la cual se genera por una macro "RESERVE". Esta instrucción "RESERVE" reserva a un volúmen completo (un disco completo) que contiene al recurso requerido hasta que se liberaba el mismo por medio de la macro "DEQ". Esto tiene la desventaja de que se reserva a todo un volúmen por reservar sólo un recurso que reside en ese volúmen, dejando en espera a otras tareas que puedan estar requiriendo otro recurso que nada tenga que ver con el que se está alterando.

El GRS serializa recursos globales sin necesidad de usar la instrucción "RESERVE". Esto lo hace comunicando los requerimientos globales a todos los sistemas definidos en el complejo, lo que permite serializar a los recursos individuales y no a todo el volúmen.

Para serializar el uso de recursos globales entre diferentes sistemas del complejo, el programa emite una macro "ENQ" con SCOPE=SYSTEMS (Alcance=sistemas).

3.8.3 Bloques de Control de serialización

El GRS es la parte del sistema operativo que se encarga de manejar los "ENQUEUEs", "DEQUEUEs" y "RESERVEs". Tiene su propio espacio de direcciones, en donde residen todos los bloques de control.

Las macros "ENQ" y "RESERVE" se expanden en una SVC 56 y la macro de "DEQUEUE" en una SVC 48. Estas rutinas de SVC generan un "Program Call" (PC) al espacio de direcciones del GRS en donde se procesa el requerimiento.

Cuando el GRS recibe un requerimiento, construye un "Bloque de trabajo de cola" ("Queue Work Block", QWB) en la SQA para representar el requerimiento de "ENQ","DEQ" o "RESERVE". Si el requerimiento es para un recurso global, el GRS copia este QWB de la SQA al área privada del espacio de direcciones del GRS.

De la información de los QWBs, el GRS crea otros bloques de control (QCBs, QELs y QXBs) que usa para satisfacer el requerimiento. El GRS pasa a cada sistema del complejo las QWBs de los requerimientos de recursos globales de cada sistema. Como resultado de ello, cada sistema crea y encadena QCBs,QELs y QXBs que representan requerimientos globales y sus propias QCBs,QELs y QXBs para los requerimientos locales.

El "Bloque de Control de Cola" ("Queue Control Block", QCB) describe a los recursos requeridos, el "Elemento de Cola" ("Queue Element", QEL) describe al espacio de direcciones que hace el requerimiento (contiene la identificación del espacio de direcciones) y si el requerimiento es para control compartido o exclusivo del recurso.

El "Bloque de Extensión de Cola" ("Queue Extension Block", QXB) describe el requerimiento de "ENQ", contiene la dirección de la TCB y la dirección de la ECB o la SVRB.

3.9 CERRADURAS ("LOCKING")

El MVS cuenta con múltiples "Palabras Cerradura" ("Lockwords") para serializar el uso de recursos del sistema entre rutinas de supervisor y entre los diferentes procesadores de un complejo.

Una cerradura ("Lock"), como ya se mencionó, es simplemente un campo en memoria ("Lockword") que indica si un recurso se está utilizando y quien lo está utilizando.

Las "Lockwords" no son necesariamante parte de un recurso, pero siempre están relacionadas a un recurso. El recurso no está protegido por la Lockword "Físicamente", es decir que sea imposible accesar el recurso si el "Lock" está cerrado, si no que funciona como un acuerdo entre todos los usuarios del sistema, los cuales deben checar el estado de la "Lockword" antes de intentar usar el recurso.

Para usar un recurso protegido por un "Lock", la rutina debe requirir el "Lock" por medio de la macro "SETLOCK"(Adquiere "Lock"). Esta macro es la interfaz con el "Manejador de Locks", una rutina de supervisor que se encarga de adquirir y mantener todos los "Locks".

Si el "Lock" no está disponible (es decir que se encuentra adquirido por otro procesador) se toma un acción dependiendo del tipo de "Lock" requerido. Los tipos de "Lock" son:

- Spin "Lock": En este tipo de "Locks" el procesador que requiere el "Lock" seguirá intentándo obtenerlo hasta que lo logra, lo cual se denomina "Spin Loop" (figura 3.19).

- "Lock" de suspensión: En este tipo de "Locks", la unidad de trabajo que requiere el "Lock" se suspende hasta que el "Lock" se hace disponible, y el procesador ejecuta código del Despachador para encontrar otra tarea para que la CPU ejecute (figura 3.20).

Dependiendo del alcance de la protección, los "Locks" pueden ser:

-"Locks" globales: Son para recursos relacionados con más de un espacio de direcciones. Por ejemplo, los bloques de control usados por el Despachador o por el supervisor de I/O.

-"Locks" locales: Son usados para serializar recursos de un espacio de direcciones en particular.

3.9.1 Proceso de "SETLOCK"

La macro de "SETLOCK" es usada para obtener, liberar o probar el estado de un "Lock".

El proceso de obtener un "Lock" depende de la disponibilidad y el tipo de "Lock" requerido. Si el "Lock" está disponible, se coloca la dirección lógica de la CPU que obtiene el "Lock" en un campo de este. Después de esto se actualiza un campo de la PSA que corresponde al procesador en donde se contabilizan los "Locks" obtenidos.

Si el "Lock" no está disponible , el manejador de "Locks" regresa al que lo requiere con un código de retorno indicando que no está disponible.

Si el requerimiento es incondicional, se toman diferentes acciones dependiendo del tipo de "Lock":

- "Spin Lock":

Se prende un bit que indica que esta en proceso un "Spin loop". Este "loop" consiste en una instrucción que prueba la disponibilidad del "Lock", incrementa un contador y se repite el proceso.

- "Lock" local:

La tarea que requiere el "Lock" se supende. Su estatus es salvado en la TCB o la SRB dependiendo de el modo en el que estaba ejecutando. Si está ejecutando bajo una TCB , los registros y una PSW que apunta a la rutina que se encarga de obtener el "Lock" se salvan la TCB. Entonces la rutina de "setlock" bifurca al Despachador para tratar de encontrar otra cosa que hacer.

Si se está ejecutando bajo una SRB, se crea una SSRB (es decir, se suspende), se salvan los registros y la PSW en la SSRB además de la PSW que apunta a la rutina que obtiene el "Lock". Entonces la SSRB es encolada a una cola de SRBs suspendidas para "Locks" locales. Después se bifurca al despachador para encontrar otro trabajo que ejecutar.

- "Lock" de Servicios de Memoria Cruzada ("Cross Memory Services", CMS):

La unidad de trabajo que requiere el "Lock" se supende. Si se está ejecutando bajo una TCB se guardan los registros y la PSW en el "Area de Salvado del Manejador de Interrupciones" ("Interrupt handler Save Area", IHSA). La "Instruction Address" de la PSW que se salvó apunta a la rutina que obtiene el "Lock" local. Se prende un bit de la ASCB para evitar que el despachador despache a alguna TCB de este espacio de direcciones. La ASCB se encola a la cola de suspendidas de CMS. Por último, se bifurca al Despachador. Si se está ejecutando bajo una SRB, se obtiene una SSRB, los registros y la PSW (apuntando a la rutina que obtiene el "Lock") se salvan en la SSRB. Se prende el mismo bit de la ASCB que evita que se despache cualquier TCB del espacio de direcciones y se encola la SSRB a la cola de suspendidas por CMS. En esta cola se pueden encontrar ASCBs y SSRBs esperando por el "Lock" de CMS. Por último se bifurca al Despachador.

El proceso de liberar un "Lock" se hace también por medio de la macro "SETLOCK" y la forma en que esto se ejecuta depende de si se va a liberar un "Lock" de suspensión o un spin "Lock":

-"Spin Lock": Regresa al que emite la macro si el que la emite no poseé el "Lock". Si lo poseé guarda ceros en el "lockword", y actualiza la PSA indicando que ya no poseé el "Lock".

-"Lock" de suspensión: Checa si el que emite la macro está tratando de liberar el "Lock" local mientras poseé el "Lock" de CMS, lo que provoca un "ABEND". Entonces, si se está liberando un "Lock" local se desencola la primera SSRB de la cola local de suspensión del espacio de direcciones. Si se está liberando un "Lock" de CMS se desencola la ASCB de la cola de suspensión, se actualiza la PSA y se regresa al que emitió la macro.

3.9.2 "Locks" Especiales

"Lock" de CPU.- Este tipo de "Lock" no serializa ningún recurso, sino que después de obtenerse el "Lock" el procesador se deshabilita a interrupciones externas y de I/O. La "Lockword" sólo contiene un contador, el cual se incrementa cuando se obtiene el "Lock" y se decrementa cuando se libera. Solo hasta que este contador es cero se puede habilitar al procesador.

"Locks" Exclusivos o Compartidos.- El "Lock" del "Manejador de Memoria Real" ("Real Storage Manager", RSM) puede obtenerse exclusivo o compartido, esto permite a más de un usuario leer datos obteniendo el "Lock" compartido y a la vez tener serialización para escribir datos. Si el "Lock" se obtiene compartido para uno o más usuarios no se puede obtener exclusivo, a la vez si un "Lock" se obtiene como exclusivo, no se puede atender ningún otro requerimiento.

3.9.3 Reglas para obtener "Locks"

Para evitar inter-bloqueos entre CPUs, los "Locks" deben pedirse en un orden específico, por esta razón se definió una jerarquía de "Locks". La forma en que trabaja el esquema de jerarquias es como sigue:

- Los "Locks" pueden obtenerse condicionalmente o incondicionalmente. Sin embargo, sólo "los Locks" con más alta prioridad que el "Lock" que actualmente se tenga pueden obtenerse incondicionalmente.

- El que pide el "Lock" sólo puede requerir un sólo "Lock" del mismo tipo.

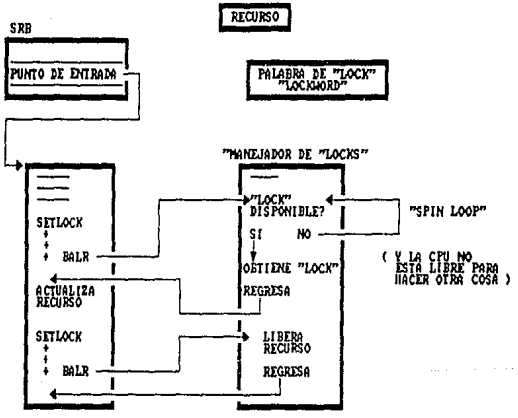


FIGURA 3.19
PROCESO DE UN "SPIN LOOP"

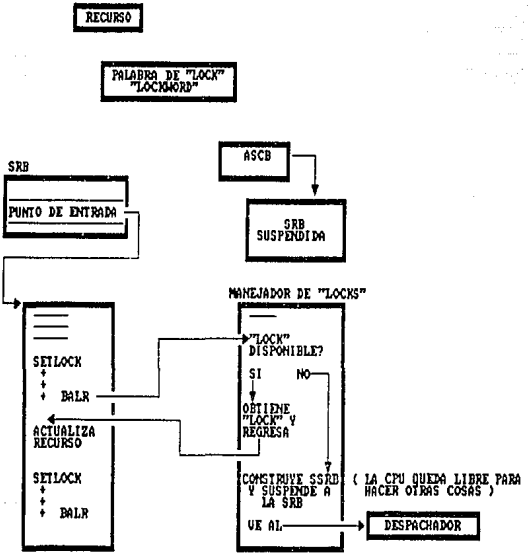


FIGURA 3.20

PROCESO DE UN "LOCK" DE SUSPENSION

-Cuando se pide un "Lock" de CMS, el que pide el "Lock" ya debe tener un "Lock" local. Para otro tipo de "Locks" no es necesario tener ningun otro "Lock" de menor jerarquía.

3.10 DESPACHADOR

En un sistema de multiprogramación como el MVS, muchas tareas activas compiten por procesarse, sin embargo una CPU sólo puede procesar una tarea a la vez. El que decide quien es la tarea que se va a procesar a continuación es el componente del MVS llamado "Despachador" ("Dispatcher")

El despachador tiene la responsabilidad de encontrar a la unidad de trabajo de más alta prioridad en el sistema, y pasarle control para que ejecute. Como se vió anteriormente, las unidades de trabajo pueden ser:

-Salidas especiales
-Requerimientos de servicio
-Tareas

El proceso del despachador es como sigue:

Antes que nada el despachador se protege a el mismo con una "Rutina de Recuperación Funcional" ("Functional Recovery Routine", FRR) llamada SUPERFRR y prende unos bits en la PSA para indicar que el despachador está en control.

Después el despachador checa la cola de despacho en busca de una unidad de trabajo lista para ejecutar en el siguiente orden:

1.-Salidas especiales

Una salida especial es la unidad de trabajo de más alta prioridad del sistema. Por ejemplo, si un procesador avisa a los demás procesadores que tiene una falla (lo cual lo hace a través de la instrucción SIGP -"Signal Processor"-) se encola en la cola de despacho una salida especial para atender esto.

2.-SRBs Globales

Estas son las unidades de trabajo con la siguiente más alta prioridad en el sistema. Estas SRBs se encadenan a una tabla llamada "Tabla de Vectores del supervisor" ("Supervisor Vector Table", SVT) que es apuntada por la PSA. (figura 3.21).

3.- Espacio de direcciones de más alta prioridad

A continuación el despachador intenta despachar a un espacio de direcciones. En la PSA se indica que espacio de direcciones está actualmente despachado.

El despachador comienza con el primer ASCB de la cola de despacho de espacios de direcciones, la cual se encadena a la CVT (figura 3.22).

Para determinar si un espacio de direcciones es despachable el despachador checa ciertos campos en la ASCB que le indican el número de TCBs listas, el número de SRBs suspendidas, el número de CPUs que están usando este espacio de direcciones, etc.

Si el espacio de direcciones no es despachable, hace lo mismo con el siguiente (Todas las ASCBs están encadenadas).

Si es despachable, se actualiza la PSA, se carga el registro de control 1 con el valor del STO (orígen de la Tabla de Segmentos, el cual lo obtiene de la ASCB) e incrementa en uno el contador de CPUs.

4.- SRBs locales

Las SRBs locales son las unidades de trabajo de más alta prioridad del espacio de direcciones.

5.- Tarea de más alta prioridad del espacio de direcciones.

Si la unidad de trabajo está lista para ejecutar, el despachador le pasa control a esta.

Si no se encuentra absolutamente nada que procesar en todo el sistema, el despachador carga una PSW Falsa ("Dummy") que tiene el bit de "Wait" prendido (bit 14). Esta PSW siempre se vé así:

070E0000 00000000

3.11 MANEJO DE INTERRUPCIONES

Com se vio en el capitulo II existen 6 clases de interrupciones:

1.-Interrupciones de "RESTART" : Iniciadas por el operador cuando este oprime la tecla de "RESTART" en la consola.

2.-Interrupciones EXTERNAS: Iniciadas por eventos tales como una alerta de mal funcionamiento, un evento de tiempo que acaba, etc.

3.-Interrupciones de "SUPERVISOR CALL": Causadas por software al ejecutarse la instrucción SVC.

4.-Interrupciones de "PROGRAM CHECK": Causadas por un error de programación, como un código de operación no válido.

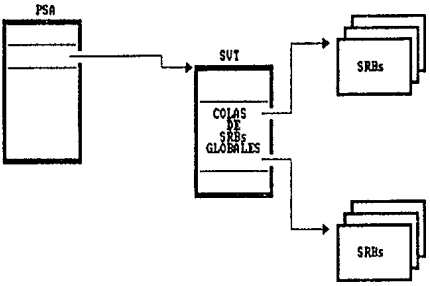


FIGURA 3.21
COLAS DE SRBs GLOBALES

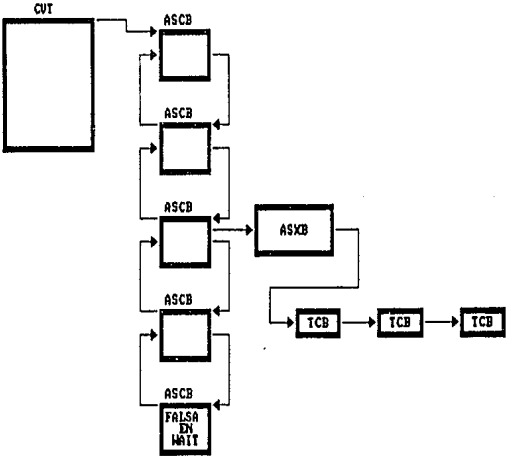


FIGURA 3.22
COLA DE ASCBs EN UN ESPACIO DE DIRECCIONES

5.-Interrupciones de "MACHINE CHECK": Que indican una falla en la CPU, Memoria ó Subsistema de Canal.

6.-Interrupciones de I/O: Causadas por un cambio de estatus en el subcanal.

Las rutinas del MVS que reciben control primero cuando ocurre una interrupción se denominan "Manejadores de Interrupciones de Primer Nivel" ("First Level Interrupt Handlers", FLIH). Existe una FLIH para cada tipo de interrupción, sin embargo su función es idéntica:

- Salvar el estatus de la rutina interrumpida.
- Prender un bit de la PSA (un bit de una serie de bits que se denominan SUPER BITS, y que indican que alguna rutina de supervisor está en control).
- Analizar el código de interrupción y dar control a la rutina "Manejadora de Interrupciones de Segundo Nivel" ("Second Level Interrupt Handler", SLIH), la cual es la rutina que realmente atiende a la interrupción.
- Al regreso de la SLIH se le dá control a la rutina interrumpida o bién al Despachador. (figura 3.23).

La información que guarda el FLIH cuando toma control es:

- La PSW
- Los Registros Generales
- Información de "Cross Memory"
- Información de Recuperación (por ejemplo "Pilas" de FRRS, que se verán más adelante)

Esta información es guardada en diferentes bloques de control, dependiendo del estado de la tarea que haya sido interrumpida (es decir si la tarea estaba en modo normal, deshabilitada para interrupciones, con un "Lock" adquirido, si es una SRB, etc.).

Por ejemplo, para una tarea en modo normal, los registros se guardan en la TCB y la PSW en la RB.

3.11.1 Interrupciones de SVC

Una interrupción de SVC ocurre cuando se ejecuta una instrucción SVC que se usa para requerir servicios de supervisor. Estos servicios los realizan rutinas que:

-Son reentrantes y reusables
-Se localizan en el núcleo o en la LPA
-Corren en llave cero y estado supervisor
-Corren habilitadas para interrupciones

El concepto de interrupción es una forma fácil de pasar control a estas rutinas y establecer la autoridad necesaria

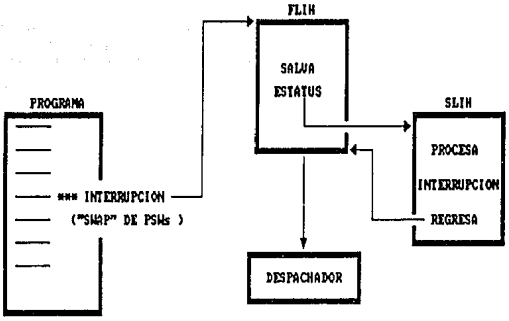


FIGURA 3.23
MANEJADORES DE INTERRUPCIONES DE PRIMER NIVEL (FLIH) Y
DE SEGUNDO NIVEL (SLIH)

para estas. La ejecución de la instrucción SVC provoca una interrupción lo cual provoca que el FLIH de interrupciones de SVC tome control. En este caso una de las funciones del FLIH además de las arriba descritas es preparar el ambiente necesario para que la rutina de servicio tome control.

La arquitectura cuenta con 256 SVCs diferentes. El MVS usa de la 0 a la 139 y el resto son para usuario. Existen además 5 tipos de SVCs numeradas del 1 al 6 (la SVC tipo 5 no existe) que dependen de sus características (como si residen en la LPA, núcleo, corren habilitadas para interrupciones, etc). Solo las SVCs tipo 2, 3 y 4 construyen una SVRB cuando se ejecutan. Si estas rutinas de SVC son interrumpidas sus registros son guardados en la TCB y la PSW en la SVRB.

Las SVCs tipo 1 y 6 no crean SVRBs debido a que la función de este bloque de control es la de servir como area de salvado en caso de interrupción, y como estas SVCs corren deshabilitadas a interrupciones, entonces no necesitan construir este bloque de control. En la figura 3.24 se muestra como ocurre una interrupción de supervisor de tipos 2,3 o 4. En la figura 3.25 se muestra el caso de una interrupción tipo 1 o 6.

La interrupción de I/O se tratará en la sección dedicada al IOS. La discusión detallada de los demás tipos de interrupciones resulta irrelevantes para la comprensión del resto del trabajo.

3.12 MANEJO DE RECUPERACION DE ERRORES

Un sistema está disponible sólo cuando el hardware y el software a la vez son capaces de procesar trabajos. El manejo de errores en MVS/XA está diseñado para incrementar la disponibilidad del sistema y reducir el impacto que provoca en los usuarios los ocurran errores en componentes críticos de software o de hardware. Si la recuperación no es posible, entonces la acción que toma el sistema es continuar trabajando sin el componente dañado. En general, los procedimientos de recuperación se llevan a cabo de tal manera que resultan transparentes para el usuario.

Las rutinas de recuperación tienen tres objetivos:

-Aislar al error. Es decir evitar que un error en una tarea afecte a todo el sistema.

-Indicar las acciones a tomar, como volcar la memoria ("Dump") o terminar anormalmente la tarea ("ABEND").

-Reparar el daño y realizar "Limpieza" de manera que la función pueda ser reiniciada.

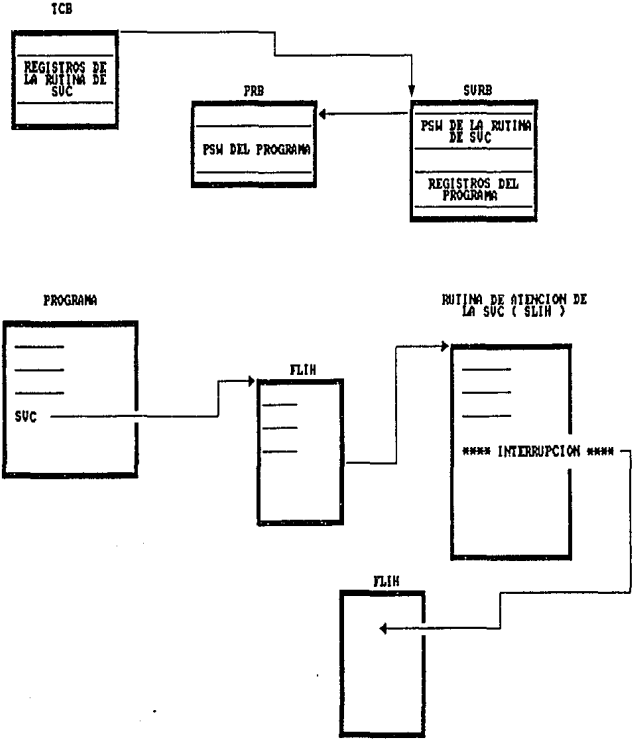


FIGURA 3.24

PROCESO DE UNA INTERRUPCION DE UNA SVC TIPO 2,3 o 4

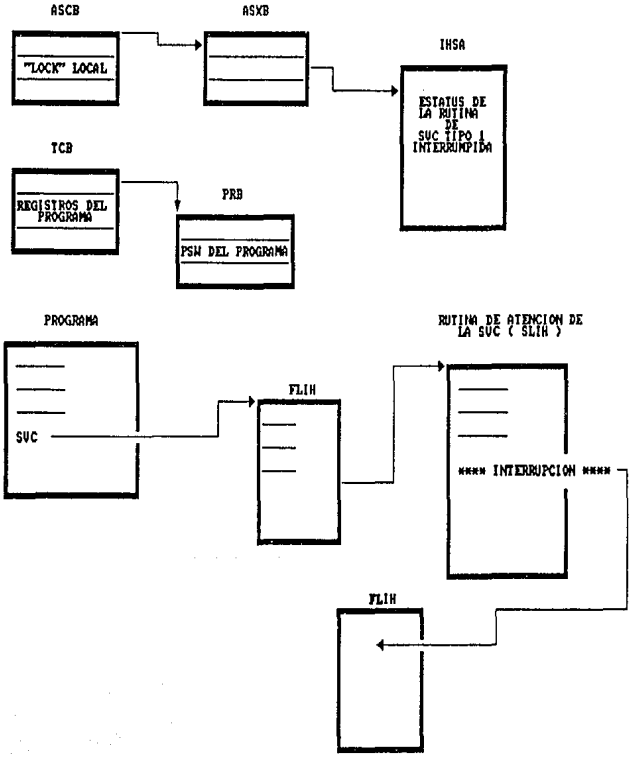


FIGURA 3.25

PROCESO DE UNA INTERRUPCION DE UNA SVC TIPO 1

Para realizar estas funciones existe un componente del MVS/XA llamado el "Manejador de Terminación y Recuperación" ("Recovery Termination Manager", RTM) el cual monitorea el flujo del proceso de recuperación de software, manejando todas las terminaciones normales y anormales de tareas y espacios de direcciones, y pasando control a rutinas de recuperación. El RTM permite que los usuarios establezcan sus propias rutinas de recuperación para aumentar la disponibilidad del sistema.

El RTM es invocado cuando se presentan distintos eventos: Un error de I/O en una operación de paginación, Un error en un programa que no es manejado por un manejador de interrupciones, un error de máquina que no es manejado por hardware, etc.

3.12.1 Rutinas de Recuperación

Existen dos tipos de rutinas de recuperación : las "Rutinas de recuperación de tareas" y las "Rutinas de recuperación Funcionales".

3.12.1.1 Rutinas de recuperación de Tareas

Las rutinas de recuperación de tareas se denominan "Salidas Extendidas de Especificación Anormal de Tareas" ("Extended Specific Task Abnormal Exit", ESTAE) y proveén protección a tareas que corren habilitadas para interrupciones, sin "Locks" y en estado problema. Estas rutinas se establecen emitiendo la macro "ESTAE".

Mediante el uso de estas rutinas, los programas pueden interceptar un ABEND previsto. Con el uso adecuado de esta facilidad, el usuario puede diagnosticar la causa de un ABEND, tratar de corregirlo o bien reintentar la operación sin que la tarea sea terminada. Esto permite que subsistemas como el "Sistema de Control de Información a Clientes" ("Customer Information Control System", CICS) maneje el mismo sus errores de usuario, tenga sus propios "ABENDs" y evite que debido a un error de usuario el sistema operativo falle.

Si una tarea se va a terminar anormalmente, la rutina de recuperación especificada en la macro de ESTAE más reciente toma control. Si esta ESTAE no puede recuperar el error, la siguiente ESTAE de mayor nivel (si existe) toma control y así sucesivamente. Este proceso de pasar control a una rutina de recuperación de nivel más alto se conoce como "Percolación".

A cada ESTAE le es cedido el control en orden LIFO (el último en entrar es el último en salir, "Last IN First OUT"), hasta que alguna de ellas pide que se reintente la operación o se acaban las rutinas de recuperación, lo cual siempre termina con una terminación anormal (ABEND).

3.12.1.2 Rutinas de Recuperación Funcionales

Las rutinas de recuperación funcionales ("Functional Recovery Routines", FRRs) son un medio de recuperación para aquellas rutinas que corren deshabilitadas, con "Locks" o bajo una SRB, o bién para rutinas que corren en llave cero y estado supervisor. Estas rutinas establecen una FRR por medio de la macro "SETFRR" (Establece FRR).

La macro "SETFRR" da a los programas del sistema la habilidad para definir su entorno de recuperación único. Cada FRR establecida por el programa del sistema es colocada en una pila ("Stack") LIFO. La macro "SETFRR" se usa para adicionar, borrar o reemplazar FRRs de esta pila.

El proceso mediante el cual se pasa control a rutinas de recuperación de más alto nivel es igual al de las ESTAEs.

3.12.2 Estructura del RTM

Funcionalmente el RTM está dividido en dos componentes principales: El RTM1 y el RTM2.

El RTM1 es la interfaz primaria entre las rutinas de supervisor que detectan errores y las rutinas de recuperación que protegen al supervisor de errores. El RTM1 también es el encargado de emitir la SVC 13 (x'D') que es la SVC de "ABEND" la cual invoca al otro componente, el RTM2.

El RTM1 se invoca a través de la macro "CALLRTM" (Llama RTM), lo cual se hace, como ya se mencionó si se detecta un error de I/O en paginación, cuando se quiere forzar la terminación de un espacio de direcciones, cuando se detecta un "Page Fault", para intentar recuperación por software en un error de máquina, cuando se emite una SVC corriendo bajo una SRB, etc.

El RTM1 puede funcionar en tres modos de operación:
-Como un SLIH cuando se le pasa control de una FLIH que detectó un error.
-Como un "Prestador de servicios" cuando se le llama para terminar un espacio de direcciones.
-En modo de reparación de hardware, cuando intenta recuperar un error de hardware por software.

El RTM2 es el componente que se encarga de terminar normalmente y anormalmente las tareas y los pasos de un "Job", obtener un "Dump" si es necesario, y controlar la "Limpieza" de los recursos que tenía la tarea.

3.12.3 Bloques de Control de Recuperación

El "Area de Diagnóstico del Sistema" ("System Diagnostic Work Area", SDWA) es el medio de comunicación entre el RTM y las rutinas de recuperación. Esta área es de especial interés cuando se está tratando de diagnosticar una falla en el sistema. La SDWA se usa para crear registros en un "Data set" llamado SYS1. LOGREC el cual sirve como "Bitácora" de los errores de Software y Hardware que haya sufrido el sistema.

La SDWA se compone de dos áreas: Un área estandar que contiene información general acerca de la falla y un área variable cuyo contenido y tamaño depende del tipo de falla que esta SDWA este representando.

Existe otro bloque de control llamado RT1W que se encuentra como un encabezado a cada entrada del "Stack" de FRRs y que contiene información que sirve para comunicar a las FRRs.

Otros bloques de control llamados "Descriptores de Errores Extendidos" ("Extended Error Descriptor", EED) sirven como interfaz de comunicación entre el RTM1 y el RTM2. Existen 7 tipos de EEDs que pueden contener desde registros y PSW al momento de un "ABEND" hasta opciones de "Dump" por ejemplo.

El bloque de control RTM2WA es el área de trabajo del RTM2 y el bloque más importante de este. Aquí reside una copia de la SDWA además de información adicional. Este es uno de los primeros lugares a los que hay que acudir para recabar información acerca de un error en el sistema.

3.12.4 Salida Especificada de Programa

Las "Salidas Especificadas de Programa" ("Specified Program Exit", SPIE) y las SPIEs extendidas (ESPIEs) permiten que un programa especifique su propia rutina de salida a la que se dará control si ocurren interrupciones de verificación de programa ("Program Checks"). Las SPIEs y las ESPIEs cumplen con la misma función sólo que las SPIEs sólo permiten direccionamiento con 24 bits y las ESPIEs con 31 bits.

Una ESPIE se especifica emitiendo la macro SPIE o ESPIE, en esta macro se pasan como parámetros el código de "Program Check" que se quiere atrapar, así como la dirección de la rutina a la que se va a dar control cuando se produzca el mismo.

Este tipo de macros permiten atrapar "Program Checks" de tipos x'1' hasta x'F' y x'11'. Los "Program Checks" tipo x'10' (Falta de Segmento) no se pueden manejar por medio de SPIE/ESPIE.

Cuando ocurre uno de los códigos de "Program Check" especificados, la rutina que lo maneja recibe control con la misma llave de memoria que la TCB bajo la cual ejecutaba el programa y con el modo de direccionamiento con el que se emitió la SPIE/ESPIE.

3.13 MANEJADOR DE MEMORIA

Como se mencionó anteriormente, los dos gigabytes de memoria de un espacio de direcciones de MVS/XA están compartidos entre programas de usuario y programas del sistema MVS/XA. Las áreas del sistema incluyen:

-El "Area de Salvado Prefijada" ("Prefixed Save Area", PSA), la cual contiene información crítica y única para cada procesador en el sistema.

-El área del "Núcleo" ("Núcleus") del Programa de control del sistema ("System Control Program", SCP) que debe estar siempre en memoria.

-Programas y rutina comunes a todos los usuarios y subsistemas.

En la figura 3.26 se muestra un mapa de un espacio de direcciones mostrando que las direcciones alojadas para estas áreas es la misma para todos los espacios de direcciones.

La organización del espacio de direcciones de MVS/XA surgió de la necesidad de mantener compatibilidad con el MVS/370. Este sistema operativo explotaba la arquitectura del Sistema/370 y su esquema de direccionamiento de 24 bits, lo que le permitía direccionar como máximo 16 Mbytes. El espacio de direcciones del MVS/370 era exactamente igual a la parte inferior del espacio de direcciones de la figura 3.26, es decir, la parte que está por debajo de "La línea" de 16 Mbytes. Como se puede observar, el MVS/XA agregó áreas homólogas a las que existían en MVS/370, arriba de la línea, lo que permitió mantener compatibilidad y explotar el direccionamiento extendido introducido con el Sistema/370-XA.

3.13.1 Modo de Direccionamiento y Modo de Residencia

Como ya se ha mencionado, para mantener compatibilidad con el MVS/370, el MVS/XA reconoce direcciones de 24 bits.

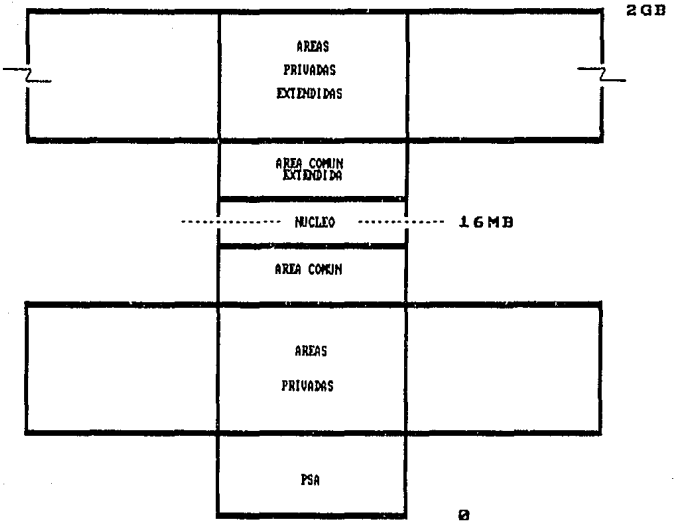


FIGURA 3.26

ESTRUCTURA GENERAL DE UN ESPACIO DE DIRECCIONES EN MVS/XA

El que el MVS/XA reconozca a una dirección como de 24 o 31 bits depende del bit 32 de la PSW (ver figura 2.7). Si este bit está prendido, las direcciones se interpretan como de 31 bits. Los programas que corren en modo de 31 bits pueden accesar desde la dirección 0 hasta 2 Gbytes de memoria virtual.

El MVS/XA permite cambiar el modo de direccionamiento durante la ejecución de un programa para accesar datos o módulos corriendo en otro modo, de manera que los nuevos programas que corran en 31 bits pueden tomar ventaja del direccionamiento de 31 bits y ser compatibles con los que corran en 24 bits.

Todos los módulos de programas bajo MVS/XA tienen un atributo de "Modo de direccionamiento" ("Addressing Mode", AMODE) que indica qué modo de direccionamiento tomará efecto cuando el módulo tome control. Este AMODE se debe dar como parámetro durante la "linkedición". El valor por omisión ("Default") es un AMODE de 24 bits. El parámetro se indica como AMODE=24, AMODE=31 ó AMODE=ANY que implica cualquier modo de direccionamiento.

Los módulos también tienen un atributo de "Modo de Residencia" ("Residence Mode", RMODE) que indica si deben ser cargados abajo de la linea de 16 Mb (RMODE=24) ó pueden ser cargados en cualquier lugar del espacio de direcciones (RMODE=ANY).

Un programa que debe ser directamente direccionable por programas corriendo en modo 24 bits debe residir debajo de la línea. Un programa que no es referenciado por programas de 24 bits o que son referenciados por estos indirectamente pueden residir en cualquier parte.

La figura 3.27 muestra esquemáticamente que significan los atributos AMODE y RMODE.

3.13.2 Memoria Virtual

Como se ha mencionado, la memoria virtual es el mecanismo por el cual es posible que un usuario accese toda la memoria que por diseño sea posible direccionar, aunque físicamente esta memoria no esté disponible en el sistema. Esto se logra con la ayuda del mecanismo de hardware conocido como DAT (que ya se explicó) y con la ayuda de los componentes del MVS llamados "Manejador de Memoria Virtual","Manejador de Memoria Real" y el "Manejador de Memoria Auxiliar".

Las tareas se comunican con estos manejadores a través de servicios que se piden por medio de macros de memoria.

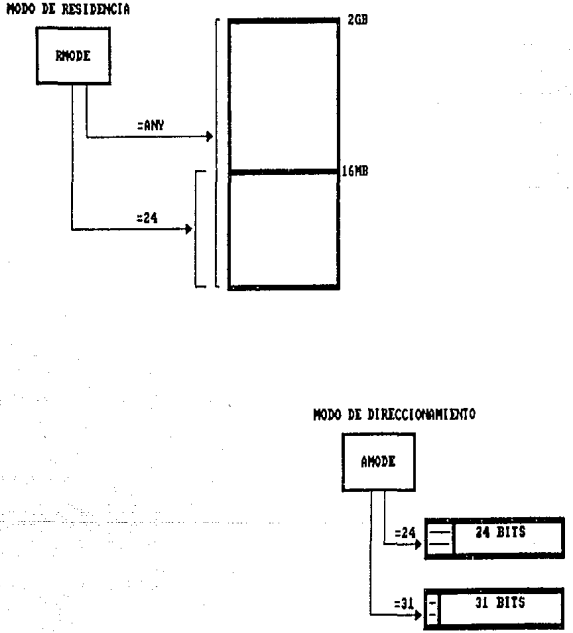


FIGURA 3.27

ATRIBUTOS DE PROGRAMA: MODO DE RESIDENCIA (RMODE) Y MODO DE DIRECCIONAMIENTO (AMODE)

Las macros más importantes son la macro "GETMAIN" (obtener Memoria Principal), la cual sirve para pedir memoria virtual y la macro "FREEMAIN" (liberar Memoria Principal) la cual sirve para liberar memoria virtual cuando ya no se va a utilizar.

3.13.2.1 Paginación

Adicionalmente a las tablas de segmentos y tablas de páginas que el DAT necesita para operar, el proceso de "Paginación" involucra a todo el código de los componentes mencionados anteriormente.

Por ejemplo, supóngase que el DAT encuentra una PGTE inválida durante el proceso de traducción, indicando que la página requerida no está en un "Frame" de memoria real. Para resolver esta condición (conocida como "Falta de página", "Page Fault") , el sistema debe traer la página que reside en un "slot" de memoria auxiliar. Para esto, primero debe encontrar un "frame" de memoria real, si no existe un "frame" libre, se debe escoger uno para liberarlo, sin embargo para hacer esto, primero hay que salvar este "frame" en memoria auxiliar y marcar su PGTE correspondiente como inválida. Esta operación es llamada "Page-out" (realmente el sistema sólo hace Page-out cuando el contenido del "frame" fué modificado desde que el "frame" se copió desde memoria auxiliar; si no fué modificado, el "frame" simplemente se libera marcando su PGTE como inválida).

Después de que se localiza un "frame" libre, el contenido de la página se copia de memoria auxiliar a memoria real, marcando su PGTE como válida. Esta operación se llama "Page-in" (figura 3.28). Realmente, para evitar operaciones de I/O innecesarias, se checa antes de hacer un "page-in" si el "frame" que previamente contenía a la página tiene la misma información que el "slot" que se piensa traer a memoria real; si es así, entonces quiere decir que la información no ha sido cambiada y que no se requiere el "page-in", y se hace lo que se conoce como un "Reclamo" de la página, esto es, se marca como válida la PGTE de esta página, sin hacer ninguna operación de I/O.

La paginación también puede suceder cuando el cargador de programas carga un programa en memoria virtual. El MVS obtiene memoria virtual para el programa de usuario, y aloja un "frame" de memoria real para cada página de programa. Desde ese momento cada página es una página activa y está sujeta a la actividad normal de paginación; esto es, las páginas más activas serán retenidas en memoria real mientras que las páginas no activas serán paginadas a memoria auxiliar.

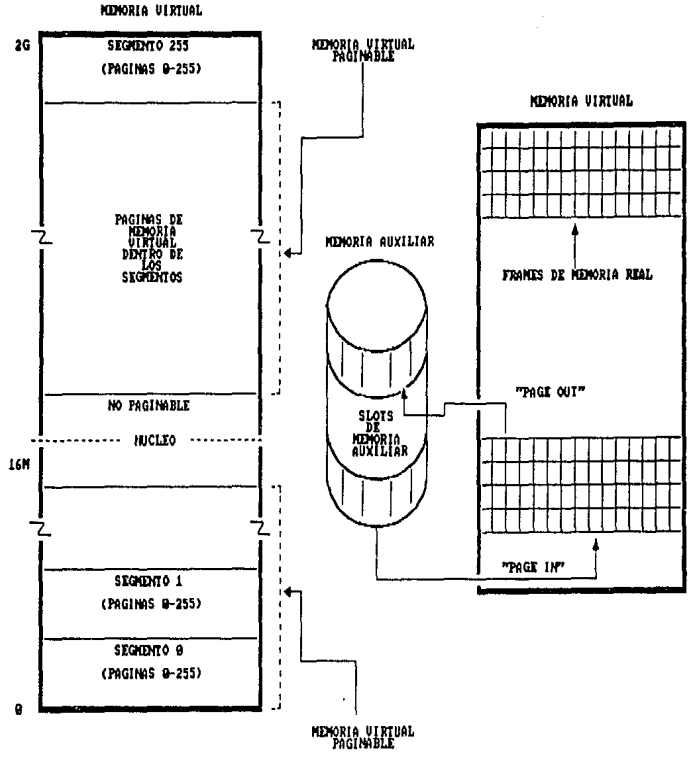


FIGURA 3.28

OPERACIONES DE "PAGINACION" DENTRO Y FUERA

El MVS trata de mantener un suministro adecuado de "frames" de memoria real a la mano para satisfacer requerimientos ocasionados por "page-faults". Cuando este suministro se hace insuficiente, el sistema usa el procedimiento conocido como "Robo de Páginas" para reabastecerlo.

3.13.2.2 Robo de Páginas

El robo de páginas ocurre cuando el sistema toma "frames" asignados a un usuario activo y los hace disponibles para otro trabajo. La decisión de robar un determinado "frame" se basa en la historia de la actividad de cada página que reside en determinado momento en memoria real. Las páginas que no han sido accesadas por un tiempo relativamente largo de tiempo son buenas candidatas para el robo de páginas.

Para determinar qué páginas pueden ser robadas, el MVS/XA examina la historia de la actividad de las páginas que se encuentran actualmente en el sistema. Esta información la guarda en una tabla llamada la "Tabla de Frames de Páginas" ("Page Frame Table", PFT). Existe sólo una PFT para todo el sistema, y tiene una entrada (PFTE) para cada "frame" disponible en memoria real.

Cada PFTE incluye: El propietario de este "frame" (el Identificador del Espacio de direcciones que lo poseé), la dirección virtual dentro del espacio de direcciones que poseé esta página, banderas que indican su estatus y contadores que describen la historia de la actividad de la página.

Adicionalmente a esta tabla, el hardware lleva el control de qué páginas han sido cambiadas y/o referenciadas en el mismo campo que se usa para guardar la llave de acceso de la página de memoria (Capitulo II). Esto lo hace a través de los bits de "Cambio" (C) y "Referencia" (R).

El hardware pone el valor de 1 en el bit de referencia cuando se accesa cualquiera de las localidades de memoria de la página asociada, y pone el valor de 1 en el bit de cambio cuando se modifica cualquiera de estas localidades .

A intervalos regulares, el sistema checa si el bit de referencia está prendido. Si este bit no está prendido, entonces incrementa un contador que se guarda en la PFTE de la página correspondiente, llamado el "Contador de Intervalo de No-referencia" ("Unreferenced Interval Counter", UIC). Si el bit de referencia es 1, quiere decir que la página ya se sido accesó, entonces el sistema pone en ceros el UIC.

Cuando se seleccionan las páginas para Robo, se seleccionan aquellas que tengan el más alto valor de UIC.

El bit de cambio es puesto a cero cuando se trae inicialmente la página a memoria real. Cuando se cambia el contenido de alguna localidad de la página, este bit se pone a 1. Si el bit de cambio vale 1 cuando se hace un "page-out", entonces implica que se debe realizar una operación de I/O para copiar toda la página a memoria auxiliar. Si este bit es cero, entonces en un "page-out" sólo se marca como inválida a la PGTE si hacer ninguna operación de I/O y se marca como disponible la página en la PFTE.

3.13.2.3 "Swapping"

El proceso de "swapping" es el proceso mediante el cual se transfieren todas las páginas activas de un espacio de direcciones (que estén en un momento dado en memoria real) hacia memoria auxiliar o viceversa. Esto es equivalente a hacer un "page-in" ó un "page-out" pero de un espacio de direcciones completo. Este es uno de los métodos que el MVS/XA emplea para balancear la carga total del sistema, además permite asegurar que se mantenga un suministro adecuado de "frames" de memoria real en todo momento. Los espacios de direcciones que son "Swapped-in" están activos, teniendo páginas en "frames" de memoria real y páginas en "slots" de memoria auxiliar. Los espacios de direcciones que son ""swapped-out"" están inactivos, no tienen ninguna página en memoria real y necesitan ser "Swapped-in" para empezar a ejecutar. La decisión de qué espacios de direcciones deben ser "swapped-in" o "swapped-out" se toma de acuerdo a recomendaciones hechas por el "Manejador de Recursos del Sistema".

3.13.2.4 Tablas de memoria

Para convertir una dirección virtual en una dirección real, el DAT sólo necesita de la SGT y la PGT. Sin embargo, cuando una tarea sufre un "page-fault" debido a que la página que intentaba referenciar no estaba en memoria, el MVS intenta recuperar esta página de memoria auxiliar.

Para mantener el control de donde residen las páginas cuando no están en memoria real el MVS se vale de otra tabla llamada la "Tabla de Páginas Externas" ("External Page Table", XPT). Cada Page Table tiene asociada una XPT. Cada entrada de la XPT (llamadas XPTE's) tiene su correspondiente entrada en la PGT en la misma posición (es decir, a la 3a. entrada de la PGT le corresponde la 3a. entrada de la XPGT) y contiene la posición de la página correspondiente en memoria auxiliar (figura 3.29).

La forma en que se identifica el "slot" en que residen las páginas es mediante un "Identificador de Slot" ("Logical Slot ID", LSID) único para cada "slot" de un "Data Set" de paginación.

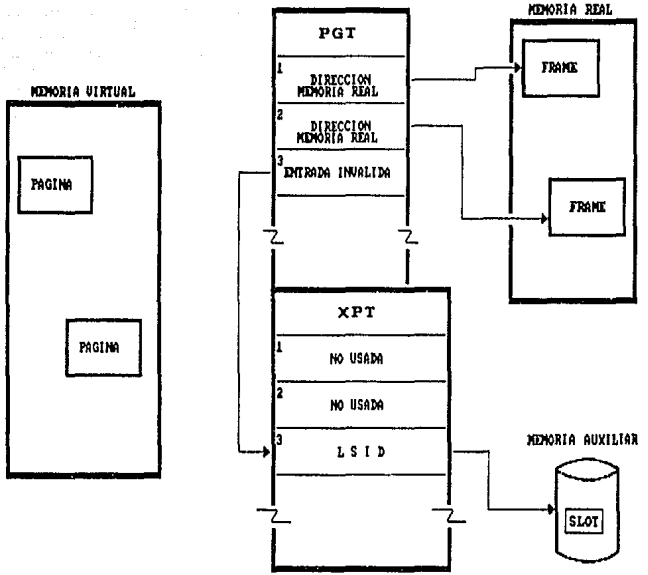


FIGURA 3.29
TABLAS DE CONTROL DE MEMORIA REAL: PGT Y XPT

3.13.3 Manejador de Memoria Real ("Real Storage Manager", RSM)

La tarea del RSM es controlar el uso de la memoria real. El RSM actúa en cooperación con el "Manejador de Memoria Virtual" ("Virtual Storage Manager", VSM) y el "Manejador de Memoria Auxiliar" ("Auxiliary Storage Manager", ASM) para soportar el concepto de memoria virtual. Además el RSM proveé servicios a muchos otros componentes del SCP y programas de aplicación para manipular el estatus de páginas y "frames".

El RSM controla el uso de "frames" de memoria real, manipula el estatus de páginas y "frames" y se encarga de atender la interrupción que se genera en "Segment Faults" y en "Page Faults".

3.13.3.1 Proceso de "Page-Fault"

Como se vio en un capitulo anterior, la Traducción Dinámica de Direcciones (DAT) es posible gracias a la existencia de Tablas de Segmentos (SGT's) y Tablas de Páginas (PGT's) debido a que esta traducción no es más que una búsqueda en estas dos tablas.

Cuando una página no reside en memoria real, el bit de "Página Inválida" (I) de la PGTE estará prendido y ocurrirá un "Page-Fault" si se accesa esta PGTE. Si la PGT que corresponde a la dirección virtual que se quiere accesar no está presente en memoria real, el bit de "Segmento Inválido" de la SGTE estará prendido y se producirá un "Segment-fault". Tanto el "Page-fault" como el "Segment-fault" son en realidad una interrupción de "Program Check" con código de interrupción x'11'y x'10' respectivamente.

La tarea de asignar un "Frame" en memoria real a la página virtual que está experimentando el "Page-fault" es responsabilidad del RSM. Este "Frame" puede ser uno nuevo que será llenado con la página que resida en un "Slot" de memoria auxiliar o que se deje en blanco si es la primera referencia a este "frame". También es posible "Reclamar" el "Frame" si ya había sido usado por la misma página.

Cuando el RSM toma control desde el FLIH de interrupciones de "Program-Check" (debido al código de interrupción x'11), El RSM verifica lo siguiente:

1.- Si la SGTE es inválida (lo cual pudo suceder en el tiempo entre que ocurrió el "page-fault" y este código tomó control) se regresa al FLIH, el cual retornará control a la tarea que sufrió el "page-fault" para que ahora sufra un "segment-fault".

2.- Si la PGTE es válida, debido a que otro procesador pudo haber ya validado la PGTE mientras que este procesador estaba corriendo el código de su FLIH. Si esto es cierto, se regresa control a la tarea que sufrió el "page-fault".

3.- Si la página ya se obtuvo; la obtención de memoria lo hacen las tareas por medio de la macro "GETMAIN". En esta macro se le pasan al RSM como parámetros la cantidad de memoria que desea obtener y regresa como resultado la dirección de la memoria que acaba de obtener. Si la memoria no ha sido obtenida (indicado como un bit prendido en la PGTE) se produce un "ABEND 0C4", y el código especifica un error en particular por el cual el programa terminó anormalmente.

Después de estas pruebas iniciales, el proceso continua como sigue:

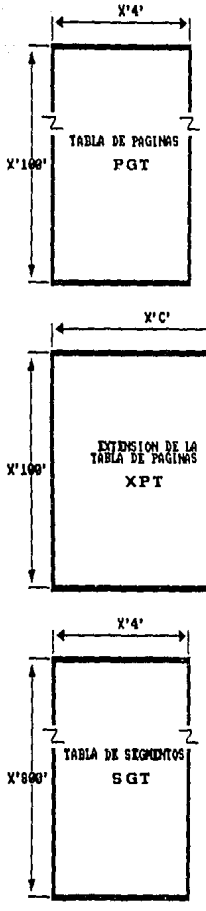
-Si es la primera referencia, significa que la dirección virtual nunca ha sido accesada hasta ahora. Entonces, el RSM obtiene un "frame", pone su llave de protección y reinicia a la tarea. Si no hay un "frame" disponible se construye un bloque de control llamado Bloque de control de página ("Page Control Block", PCB) y es encadenado a una cola de "Postergación" ("Defer Queue"). Este requerimiento pendiente se atenderá cuando existan "frames" disponibles

-Si la página ya ha sido "Postergada", es decir que un "page-fault" anterior para la misma página no ha sido aún completado, se encadena una "PCB relacionada" a la PCB original, de manera que cuando el "page-fault" anterior sea completado, se remueva a la PCB original de la cadena de postergación y se encuentre a la PCB relacionada, la cual servirá para removerla de la cadena y reanudar la ejecución de esta tarea.

-Si es posible la reclamación, la reclamación es la función opuesta al robo de páginas. Ambas funciones trabajan juntas como sigue:

Cuando el RSM detecta que el umbral inferior de páginas disponibles se alcanzó (este umbral se le especifica en parámetros de inicialización), el RSM informa al SRM acerca de esto. El SRM entonces llamará al RSM para robar páginas, para lo cual buscará las páginas con mayor UIC, invalidando su PGTE, marcando como disponible su PFTE y encadenándola a la cola de "frames" disponibles ("Available Frames Queue", AFQ).

Si la página se modificó (bit de cambio prendido) esta sufrirá un "Page-out". Sin embargo, la página está encadenada a la cola de "frames" disponibles (AFQ) pero todavía tiene los datos anteriores. Si este "frame" se usa



CADA PGT MAPEA 1MB DE MEMORIA

HAY X'100' PAGINAS EN 1MB

CADA PGT OCUPA X'400' BYTES

X'1'	SEGMENT TABLE	X'2000'
X'800'	PAGE TABLES DE X'400' C/U	X'200000'
X'800'	XPTs DE X'C00' C/U	X'600000'
		X'802000'

8 MBYTES

CADA XPT OCUPA X'C00'

EN CONJUNTO CADA PAR PGT-XPT OCUPAN

```
  X'400'
+ X'C00'
--------
 X'1000'
```

1 KBYTE = X'400' BYTES
1 MBYTE = X'100000' BYTES
1 PAGINA = X'1000' BYTES

CADA SGT OCUPA X'2000'
UNA SOLA SGT POR ESPACIO DE DIRECCIONES

FIGURA 3.30
CONSUMO DE MEMORIA DE LAS TABLAS DE CONTROL DE LA MISMA

en un requerimiento de "page-fault", estos datos serán reescritos por la nueva página.

Si esto no sucede, y el propietario original de la página "Reclama" esta página (es decir, que sufra un "page-fault" sobre esta página), esta simplemente se saca de la cola de AFQ y su PFTE es hecha válida para este usuario. Esto es posible debido a que en el proceso de robo de página, la información de la página no se destruye.

-Si existe en proceso una operación de I/O para esta página, si otro usuario ya tuvo un "page fault" para esta página y ya está manejando esta situación, el nuevo usuario que sufre un "page-fault" se representa por medio de una PCB en la cola relacionada. Cuando el "page-fault" original se maneja después de completar el I/O, se reanuda la tarea original.

Si ninguna de las condiciones anteriores es cierta, el RSM continúa con el proceso normal:

El RSM construye una PCB representando el "page-fault" y trata de obtener un "frame" libre de la cola de AFQ. Si no existe un "frame" disponible, la PCB se encadena a la cola de postergación ("Deferred Queue") y se maneja más tarde, cuando un "frame" se haga disponible.

La PCB y su bloque de control asociado, el "Area de Requerimiento de I/O para el ASM"("ASM I/O Requirement Area", AIA) se actualiza con los datos necesarios, como el número de "frame", el número de página, etc. y la AIA se pasa al ASM para que haga el proceso real de I/O para traer la página. Después de regresar del ASM, el RSM regresa al FLIH. La tarea que sufrió el "page-fault" se reanuda cuando se completa la operación de I/O.

3.13.3.2 Mapeo del espacio de direcciones

Para mapear una dirección virtual en una dirección real, el espacio de direcciones debe estar descrito por una Tabla de Segmentos (SGT) y varias tablas de páginas (PGTs). La tabla de segmentos y algunas de las tablas de páginas se crean en el momento de la creación del espacio de direcciones.

Como se vio, cada SGTE tiene un apuntador al inicio de una PGT. La PGT y su correspondiente XPT necesarias para mapear 1 Mbyte de memoria ocupan 4 Kbytes de memoria (figura 3.30), esto significa que la tabla de segmento (SGT), y las 2048 tablas de páginas (PGT) y tablas de páginas externas (XPT) necesarias para mapear un espacio de direcciones de 2 Gbytes ocupan aproximadamente 2048 x 4K = 8 Mbytes de memoria. Obviamente no es práctico tener en memoria real esta cantidad de bytes ocupada para mapear la memoria virtual de un sólo espacio de direcciones, por lo tanto se hace lo siguiente:

- La SGT si se mantiene en memoria real, siempre que el espacio de direcciones esté "swapped-in".
-Las PGT's que describen a las áreas comunes también deben quedar en memoria real siempre, debido a que esta área siempre se accesa.
-Las PGT's que representan al área privada debajo de la línea pueden estar en memoria real siempre, debido a que requieren sólo una cantidad pequeña de memoria.
-El área privada de encima de la línea es la que representa la mayor parte, de manera que sólo se crean las PGT's se vayan o necesitando, es decir:

Las PGT's no se crean en el momento del "GETMAIN" si no hasta la primera referencia que se haga a esta página. La primera referencia a una localidad de memoria, no descrita por una "page table", es manejada por el procesamiento de "segment-fault".

Las PGT's que representen área privada de arriba de la línea son "Paginables". Esto es, las tablas necesarias no siempre ocupan memoria real, estas se traen a memoria ("Paged-in") si realmente se van a usar. El procesamiento de "segment-fault" maneja esta situación también.

- Las tablas de segmentos y páginas están contiguas en los 9 Mbytes más altos de la memoria virtual; en memoria real, desde luego, pueden estar donde sea. (figura 3.31)

Debido a que una tabla de páginas junto con su tabla de páginas externas ocupan una página de memoria, estas están representadas por una PGTE en una tabla de páginas. Esto también es cierto para la "segment table", existen una o dos PGTE's que mapean la memoria en donde se sitúa la SGT.

Teniendo esto en cuenta, lo mínimo que se necesita para poder direccionar cualquier localidad de memoria es un segmento que contenga a la SGT que a su vez apunta a la PGT que describa este segmento. Ambas tablas están en el mismo segmento y tienen que estar fijas en memoria real.

-La PGT que describe el segmento más alto se llama "PGT Tope" ("TOP PGT"), y es la primera de todas las PGT's.

-Las siguientes 8 PGT's dentro de este segmento, que describen los siguientes 8 segmentos son llamadas las "PGT's Altas" ("HIGHER PGTs"), y son más altas que las restantes PGTs.

-Las restantes 2039 PGTs son llamadas "PGTs Bajas" (figura 3.32).

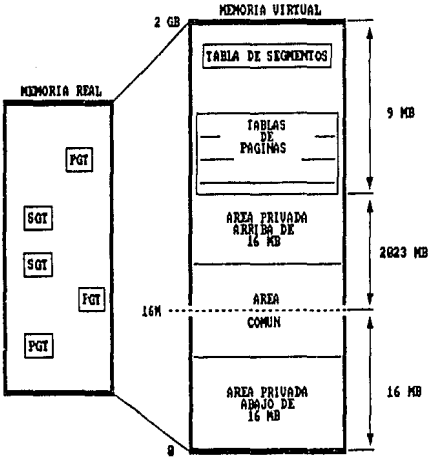


FIGURA 3.31

UBICACION EN MEMORIA DE LA SGT Y LAS PGTs

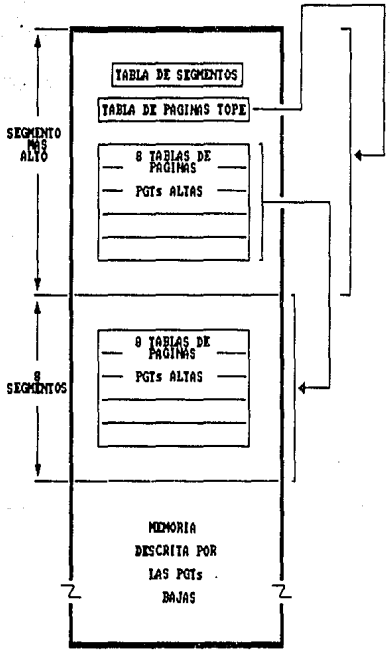


FIGURA 3.32

UBICACION DE LAS TABLAS DE PAGINAS ALTAS Y BAJAS

Algunas caracteristicas adicionales son:

+ Solo las PGTs Bajas son paginables.

+ No todas las PGTs bajas son realmente usadas. Las PGTs para el área privada de debajo de la línea están fijas en memoria, al igual que las PGTs de el área Común.

3.13.3.3 Procesamiento de "Segment-Fault"

Cuando una página se accesa y esta es descrita por una PGT que aún no se construye o que fué paginada hacia afuera, el resultado será un "segment-fault". Esto se detecta cuando el bit de segmento inválido de la SGTE está prendido.

Cuando el RSM toma control del FLIH debido a una interrupción de "program check" (código x'10'), checa si la página que se accesó ya fué adquirida (a través de la macro GETMAIN), por medio de un bit de la SGTE. Si no fué adquirida, se produce un "ABEND" código 0C4.

Después el RSM checa si la PGT baja está disponible. Esta PGT baja puede no estar disponible debido a lo siguiente:

1.- La PGT baja nunca se construyó debido a que el segmento que mapea esta PGT nunca ha sido referenciado, lo cual se conoce como "Primera Referencia". Si esto es cierto, se adquiere un "frame" de memoria real y se construye un esqueleto de PGT. La página que contiene esta página baja se conecta a la PGT alta que describe el espacio de la PGT baja. La PGT baja se marca como adquirida ("GETMAINED") e inválida.

2.- La PGT baja fué creada hace algún tiempo pero fué paginada debido a que no se usó. En este caso, la localidad en memoria auxiliar de la PGT paginada fuera se salva en la XPT de la PGT alta. Usando esta información, comienza una operación de paginación dentro ("Page-in") después de obtener un "frame" para guardar a la PGT baja que se traerá. Después de completarse el I/O, el RSM conectará la PGT baja a la PGT alta. La correspondiente PGTE en la PGT baja no será validada.

Después de completar el proceso de cualquiera de estos casos, la tarea que obtuvo el "segment fault" se reanuda. Debido a que el proceso de "segment-fault" no hace valida a la PGTE de la dirección que falló, ocurrirá un "page-fault" cuando la misma instrucción se ejecute de nuevo.

Esto es porque el objetivo del proceso del "segment fault" no es proporcionar a la tarea que falla con la memoria accesada, sino suministrar al proceso de "page fault" con todas las tablas necesarias para cargar, construir y reclamar a la página accesada.

3.13.3.4 Contabilización de memoria real

Como se ha mencionado, el RSM es responsable de mantener el control de todos los "frames" del sistema. Cada "frame", como se vio, se representa por una entrada en la Tabla de "frames" (PFT), figura 3.33. Todas las PFTEs (Es decir las entradas de la PFT) están guardadas secuencialmente por su número de "frame" en la PFT.

Las PFTEs se encolan a diferentes colas ("Queues") dependiendo de sus características. Por ejemplo, existen varias colas para encadenar todas las PFTEs de "frames" que se encuentran disponibles para su uso y otra cola de "frames" que representan áreas comunes . Estas colas se anclan todas en la "Tabla interna del RSM" ("RSM internal Table", RIT). La RIT representa el punto de anclaje común a todos los bloques de control del RSM (es decir, contiene apuntadores a todos los bloques de control del RSM). A la RIT la apunta un bloque de control llamado la "Tabla de vectores de páginas" ("Page Vector Table", PVT) que contiene todos los puntos de entrada de rutinas comunes del RSM y a su vez a la PVT la apunta la CVT. (Ver figura 3.34). Existe sólo una RIT y una PVT por procesador.

A la RIT se encadenan TODAS las PFTEs, las PCBs, las FCBs y las ESTEs de TODOS los espacios de direcciones. Para el caso de las PFTEs hay varias colas, que en general, pueden ser de "frames" disponibles y de "frames" ocupados.

Las colas de "frames" disponibles a las que se pueden encadenar las PFTEs desde la RIT (colas comunes a todos los espacios de direcciones) pueden ser la:

-PAFQ ("Preferred Above available Frame Queue")- representan "frames" de memoria que son preferidas para arriba de 16Mb.

-PBFQ ("Preferred Below available Frame Queue")- representan "frames" de memoria que son preferidos para abajo de 16Mb.

-NAFQ ("Non-preferred Above available Frame Queue")- representan "frames" de memoria que no son preferidos para arriba de 16Mb.

-NBFQ ("Non-preferred Below available Frame Queue")- representan "frames" de memoria que no son preferidos para abajo de 16Mb.

Loas colas de "frames" comúnes (ocupados) a las que pueden encadenarse los PFTEs son:

-SFQ ("SQA Frame Queue")- representan "frames" de memoria que actualmente contienen información de la SQA.

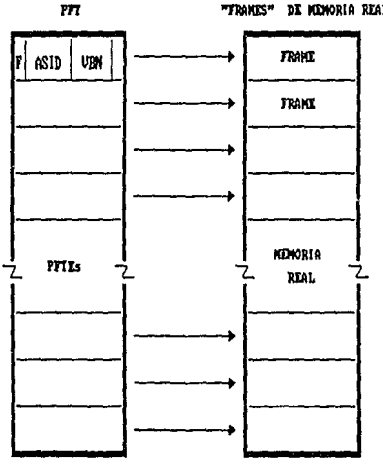


FIGURA 3.33

TABLA DE "FRAMES" DE MEMORIA REAL: PFT

-DFFQ ("Deferred Freemain Frame Queue")- representan "frames" de memoria que estaban fijos en memoria y que fueron liberados de memoria virtual por medio de un FREEMAIN.

-FFQ ("Fixed Frame Queue")- representan "frames" de memoria que actualmente están fijos en memoria, es decir que no se pueden paginar.

-PFQ ("Pageable Frame Queue")- representan "frames" de memoria que respaldan páginas de memoria virtual paginables.

Existe otro bloque de control llamado "Bloque de Espacio de direcciones del RSM" ("RSM Addres Space control Block", RAB). Hay dos tipos de RABs, existe un sólo RAB común que apunta a una cadena de RABs (uno por cada espacio de direcciones). Cada RAB de espacio de direcciones o RAB local contiene apuntadores a las colas de PFTEs, de PCBs, de FCBs y de ESTEs de ese espacio de direcciones en particular.

A partir de la RAB comun se puede localizar la SGT. A la RAB común la apunta la RIT. Las colas de PFTEs que se anclan a las RABs locales son colas de "frames" fijos, "frames" paginables, etc. (figura 3.35).

Los PCBs, como se explicó, representan un trabajo de I/O pendiente para un requerimiento de "page-in" ó "page-out" (y siempre están asociados a un AIA), mientras que los FCBs son bloques de control de funciones("Function Control Block", FCB) que se crean cuando alguna de las funciones del RSM se suspende ó interrumpe y guardan información como la PSW y losregistros; sirven para retornar el ambiente que existía antes de la interrupción ó la suspensión.

3.13.3.5 Memoria Expandida

Los procesadores más nuevos ahora cuentan con una "Memoria Expandida" que se usa como una área de paginación de alta velocidad. La memoria expandida no puede ser direccionada byte por byte sino solamente página por página (en bloques de 4 Kbytes) y sólo puede ser accesada por código del sistema.

Durante la operación del sistema, el RSM está continuamente checando que tan frecuentemente se usan los "frames" de memoria real asignados. Cuando una página virtual no ha sido accesada durante algún tiempo , el RSM robará el "frame" asignado a esta, sin embargo, si fué cambiada deberá ser salvada en memoria auxiliar. El RSM tomará la decisión de salvar esta página en memoria expandida o salvarla en memoria auxiliar, dependiendo de que tan frecuentemente se haya usado esta página.

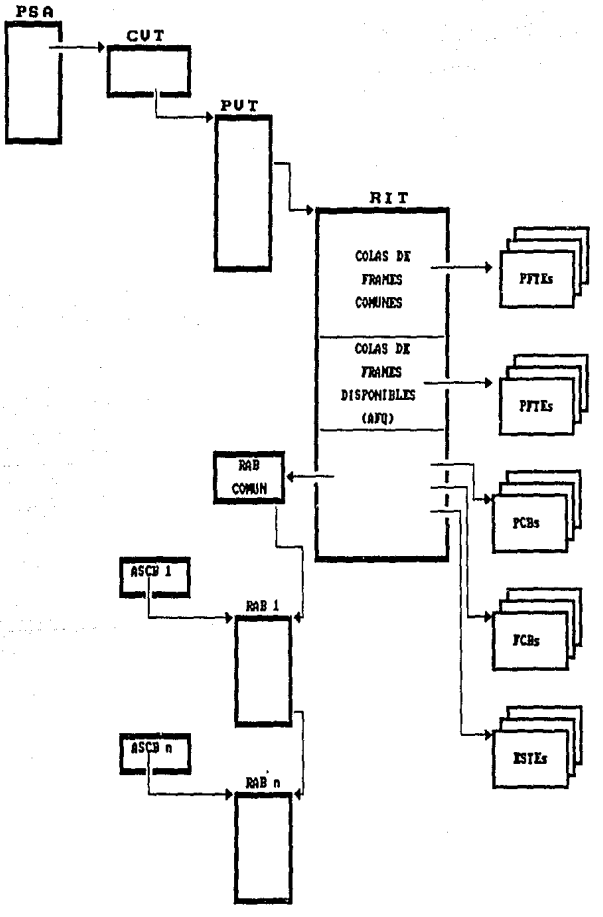


FIGURA 3.34

BLOQUES DE CONTROL DEL MANEJADOR DE MEMORIA REAL

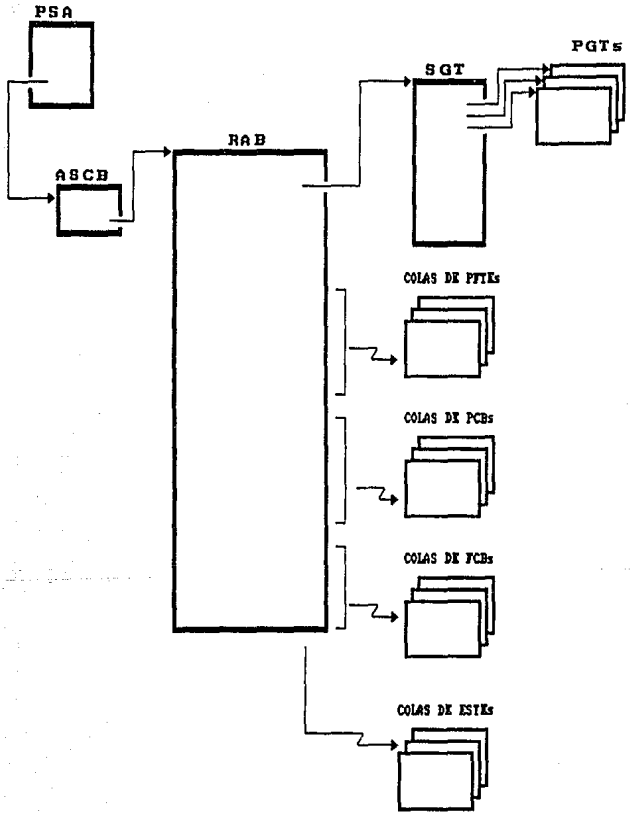


FIGURA 3.35
BLOQUES DE CONTROL DEL MANEJADOR DE MEMORIA REAL

Las páginas que se usan más frecuentemente tienen una alta probabilidad de salvarse en memoria expandida. Para salvarlas en memoria expandida no hay necesidad de hacer ninguna operación de I/O, por lo que esta operación es muy rápida. El RSM hace esto seleccionando un "frame" de memoria expandida y transfiriendo el contenido del "frame" de memoria real a este.

Cuando una página que reside en memoria expandida se accesa, se selecciona un "frame" en memoria real y se transfiere el contenido del "frame" expandido al "frame" real.

Los "frames" de memoria expandida se representan en el sistema por una tabla llamada "Tabla de Memoria Expandida" ("Expanded Storage Table", EST) compuesta de entradas ESTEs ("Expanded Storage Table Entry") que describen al "frame" y a la página de memoria virtual que reside en este. Cuando una página se mueve a memoria expandida, la dirección de la ESTE se coloca en la XPTE que le corresponde a esta página.

La memoria expandida es un área de paginación inmediata para páginas que es posible que sean accesadas pronto. Si la página reside en memoria expandida por demasiado tiempo, y se necesitan "frames" de memoria expandida, esta página se migra a memoria auxiliar. El RSM hace esto moviendo las páginas de memoria expandida a memoria real y de ahí a memoria auxiliar. No pueden moverse páginas directamente de memoria expandida a memoria auxiliar (figura 3.36).

El RSM lleva cuenta de los "frames" disponibles y ocupados de memoria expandida por medio de las ESTEs. La ESTE contiene el número de bloque virtual y el "Identificador del Espacio de Direcciones" ("Address Space ID", ASID) de la página virtual ocupando el "frame". Todo los ESTEs se guardan secuencialmente en la EST.

3.13.3.6 Servicios del RSM

La función principal del RSM es soportar la memoria virtual, sin embargo existen otros servicios que el RSM ofrece a los componentes del SCP y a los programas de aplicación, como:

Servicios de Paginación: Para modificar las características de una página y/o el "frame" que contiene a la página, como por ejemplo:

- Fijar páginas ("Page-fix"): Para asegurar que una página no será paginada ni cambiada de "frame"

- Liberar páginas("Page-free"): Operación contraria a la anterior.

- Sacar páginas("Page-out"): Ya explicado.

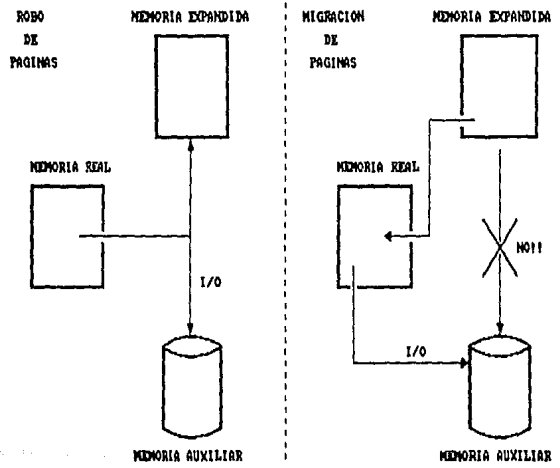


FIGURA 3.36

DIFERENCIA ENTRE PAGINACION A MEMORIA AUXILIAR Y EXPANDIDA

- Relevar Páginas ("Page-Release"): Destruye toda liga entre una página y el "slot" que conserva su copia.

- Cargar página ("Page-load"): Trae una copia de la página de memoria auxiliar a memoria real. Usando esta función, las aplicaciones dependientes del tiempo pueden ahorrar tiempo evitando un "page-fault".

(Todos estos servicios los hace a través de la macro "PGSER").

- Alojamiento V=R: Servicio mediante el cual aloja páginas con una dirección real igual a la virtual que se le asigne.

- Creación de Espacios de direcciones: En la creación de Espacios de direcciones, el RSM se invoca para construir el ambiente del nuevo espacio de direcciones. El RSM construye la Tabla de Segmentos y las tablas de páginas necesarias para establecer direccionamiento, así como sus propios bloques de control para poder servir al espacio de direcciones posteriormente.

-Verificación de llave de memoria: Como se vio en el capitulo II, la información en memoria real se protege por medio de llaves de protección de memoria. Un campo de control llamado "Llave" ("Key") se asocia a cada 4 Kbytes de memoria real. El campo de llave contiene tanto al valor de la llave como el bit de referencia, el bit de cambio y el bit de protección contra lectura.

El VSM obtiene la llave del "Job" que está corriendo de la PSW y lo pasa al RSM para asociar esta llave con la memoria real que es asignada a la página virtual. La asignación de llaves de protección para los programas del sistema se muestra en la figura 3.37.

Llave	Asignada a
0	Supervisor y otras funciones del sistema que requieren acceso a áreas privadas de la memoria
1	Planificador ("Scheduler") de Jobs y JES.
2	VSPC
3-4	Reservada
5	Manejo de datos
6	Métodos de acceso de telecomunicaciones.
7	IMS ("Information Management Subsystem")
8	Usuarios de V=V en memoria virtual
9-15	Programas de usuario V=R que requieran memoria real

Figura 3.37
Asignación de llaves de protección

3.13.4 Manejador de Memoria Auxiliar ("Auxiliary Storage Manager", ASM)

El manejador de memoria auxiliar es un componente crítico del manejador de memoria. Se encarga del control de los datos que fluyen entre memoria real y memoria auxiliar, ya sea una página a la vez (paginación) o todo un espacio de direcciones a la vez ("Swapping").

El ASM mantiene el control de los contenidos de los "Data sets" de paginación y de los "Data sets" de "Swapp" (es decir, aquellos "Data sets" que guardan las páginas de una espacio de direcciones "swappeado").

Se requieren mínimo 3 "Data sets" de paginación, un "Data Set" para la PLPA, uno o dos "Data sets" para áreas comunes y uno o más "Data sets" para áreas privadas. Los "Data sets" de "swapping" son opcionales, sin embargo la falta de ellos puede impactar fuertemente el rendimiento y tiempo de respuesta del sistema. Cada uno de estos "Data sets" debe ser un "Data Set" accesado por el "Método de acceso de memoria virtual" ("Virtual Storage Access Method", VSAM), o como se les conoce comúnmente, un archivo VSAM, formateado en bloques de 4 Kbytes.

El ASM es la capa intermedia entre el RSM y el "Supervisor de I/O" ("I/O supervisor", IOS) que es el que se encarga de la operación física de I/O (figura 3.38). El ASM se encarga de los preparativos para el requerimiento que se hace al IOS, como seleccionar el "slot" en un "Data Set" de paginación, construir el programa de canal (es decir, la cadena de CCWs), etc.

3.13.4.1 Flujo de operación

El RSM pasa el requerimiento de I/O para paginación al ASM a través del bloque de control "Area de I/O del ASM" ("ASM I/O Area", AIA), usualmente asociado a un PCB (generalmente se le conoce como el par PCB/AIA). Se construye un PCB/AIA por cada requerimiento de paginación de parte del RSM. A el PCB lo usa el RSM para tener control de el requerimiento que se recibe mientras que a el AIA lo usa con el mismo el ASM. El AIA contiene información como la identificación del "Slot" y si se trata de una operación de lectura (para un "page-in") ó escritura (para un "page-out").

El RSM cede control al ASM en un componente del mismo que se llama el "Manejador de I/O" ("I/O Driver"). Este componente, a partir de la información de la AIA escoge el "Data Set" de paginación adecuado y encola la AIA a una tabla llamada "Tabla de Referencia de actividad de páginas" ("Page Activity Reference Table", PART); esta tabla tiene una

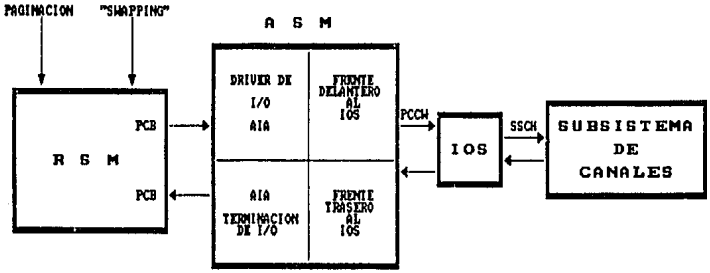


FIGURA 3.38

ESTRUCTURA GENERAL DEL MANEJADOR DE MEMORIA AUXILIAR

entrada llamada PARTE por cada "Data Set" de paginación del sistema. Cada PARTE entonces tiene una cola de AIAs que representan el trabajo pendiente para el "Data Set" que representa. Además de esto, el "Manejador de I/O" traduce el identificador de "slot" a una dirección de búsqueda física en el DASD.

Hasta este punto, todos los módulos se ejecutan bajo la tarea (TCB) que sufrió el "page-fault". Ahora se le pasa control al IOS que corre en el espacio de direcciones del "Planificador Maestro" ("Master Scheduler"), en un componente llamado "Frente delantero del subsistema de I/O" ("IOS Front End") lo cual se hace emitiendo una SRB hacia el espacio de direcciones del "Master", después de poner en estado de espera ("Wait") a la TCB de la tarea que sufrió la falta. Esta espera será hasta que la operación de I/O termine.

El frente delantero del IOS ejecuta la macro de "STARTIO" (inicia I/O) que se expande en una instrucción BALR hacia el código del IOS que reside en el núcleo. Después, el código del IOS emite la instrucción SSCH (arranca al Subcanal, "Start Subchannel") para comenzar la operación física de I/O, con lo que termina la rutina bajo la SRB.

El proceso de I/O es hecho siempre con cadenas estandarizadas de CCWs, que son guardadas en el "Area de trabajo de CCWs de paginación" ("Page Channel Command Work Area", PCCW).

Cuando la operación de I/O termina, se presenta una interrupción de I/O. Como parte del manejo de la interrupción, se invoca al componente del IOS conocido como "Frente trasero del IOS" ("IOS back-end"). Este frente trasero libera a la PCCW y le cede control al componente del ASM llamado "Completador de I/O", el cual remueve a la AIA de la PARTE y señaliza ("Post") a la tarea que estaba esperando, para que pueda reanudar su ejecución (figura 3.38).

3.13.4.2 Contabilización de Memoria Auxiliar

Como se mencionó, el ASM es responsable de mantener el control de la memoria auxiliar y el estatus de cada "slot" en los "Data sets" de "swapp" y paginación. A continuación se describen las tablas y bloques de control más importantes que el ASM usa para hacer esto:

El punto de anclaje principal del ASM es la "Tabla de Vectores del ASM" ("ASM Vector Table", ASMVT). Contiene todas las variables y constantes importantes para todo el sistema, además aquí se encadenan todos los bloques de control de interés general. A la ASMVT la apuntada la CVT y reside en el núcleo.

La "Tabla de Referencia de Actividad de Páginas" ("Page Activity Reference Table", PART) es el bloque de control que se usa para controlar a los "Data sets" de paginación. Se compone de un "Encabezado" con información como el número total de entradas en la tabla y las que se están usando. Cada entrada PARTE corresponde a un "Data Set" de paginación. Esta tabla reside en la SQA y está apuntada por la ASMVT.

La "Tabla de Referencia de Actividad de Swapp" ("Swapp Activity Reference Table", SART) es la contraparte de la PART para "Data sets" de "swapping". Las entradas de la SART se llaman SARTEs, reside en SQA y también se apuntada desde la ASMVT.

Los "Bloques de Control de Páginas" ("Page Control Block", PCB) son bloques de control que representan una operación de I/O que el RSM desea que el ASM efectúe, y siempre van asociadas a un "Area de I/O del ASM" ("ASM I/O Request Area", AIA).

El "Elemento de I/O" ("I/O Element") se usa para identificar un requerimiento de I/O del ASM que está listo para ser procesado. Contiene un apuntador a la AIA que tenga asociada.

El encabezado de la PART contiene 4 puntos de anclaje para 4 colas de IOEs, en los cuales se van encadenando los IOEs en el orden en que llegan y de acuerdo a las características de la página que se va a paginar. Estos mismos IOEs se acomodan finalmente en colas partiendo de las PARTEs de cada "Data Set" en una forma tal que se logren optimizar operaciones de lectura y escritura, combinando operaciones a "slots" que se encuentren físicamente cercanos en los "Data sets" y tratando de obtener el menor movimiento físico de la cabeza del disco.

El "Encabezado de ASM" ("ASM Header", ASMHD) se usa para manejar las operaciones de I/O relacionadas a un espacio de direcciones específico. Aquí se encadenan las PCB/AIAs en colas correspondientes a cada "Data Set" de paginación o "swapping".

El "Area de Trabajo de CCWs de paginación" ("Paging Channel Command Word Area", PCCW) contiene el programa de canal que se pasa al IOS para una operación de I/O. Cada PCCW apunta a su AIA asociada.

El "Area de Trabajo de CCWs de swapping" ("Swapp Channel Command Word Area", SCCW) es la contraparte de la PCCW para "swapping".

La "Tabla de Alojamiento de páginas" ("Page Allocation Table", PAT) se usa para traducir la identificación del "slot" en su posición física dentro del disco. Reside en la SQA y la apunta la PARTE.

La "Tabla de Características de rendimiento" ("Performance Characteristics Table", PCT) tiene información dependiente del dispositivo en el cual reside el "Data Set" de "swapp" ó paginación y sirve para obtener el máximo rendimiento del dispositivo en particular del que se trate. Reside en SQA y la apunta la PARTE.

La "Tabla de Alojamiento de Swapp" ("Swapp Allocation Table", SAT) es la contraparte de la PAT para "swapping".

La "Tabla de Características de Dispositivo para Swapp" ("Swapp Device Characteristics Table", SDCT) es la contraparte para "swapping" de la PCT.

El "Bloque de requerimiento de I/O" ("I/O request block", IORB) se usa para monitorear el progreso de un requerimiento de I/O. Tiene apuntadores al IOSB (que se verá cuando se aborde el IOS) y a la SRB que se usó para comunicarse con el IOS. Está encedenada a la PARTE o SARTE.

En la figura 3.39 se sumariza la relación entre los bloques de control del ASM.

3.13.5 Manejador de Memoria Virtual ("Virtual Storage Manager", VSM)

El VSM proveé funciones para absolutamente todos los usuarios del sistema. El VSM conjuntamente con otros componentes construye la estructura de los espacios de direcciones en donde trabajan los usuarios. Una vez hecho esto, el VSM se invoca para manejar las áreas dinámicas de el espacio de direcciones.

3.13.5.1 Creación de Espacios de direcciones

Los espacios de direcciones generalmente se crean de tres formas distintas: Como una "Tarea Arrancada" por medio del comando "START" (Arranca), Por medio de un "Iniciador" (que se verán en la sección de JES), o firmándose a TSO ("LOGON TSO"). Durante la creación del espacio de direcciones, los manejadores de memoria (RSM y VSM) se invocan para prestar ciertos servicios. Su función principal en este punto es construir la tabla de segmentos (SGT), Tablas de páginas (PGTs), y los bloques de control del VSM para llevar control de la memoria virtual del nuevo espacio de direcciones. Para construir estos bloques de control se usan los bloques de control del Planificador Maestro como esqueleto.

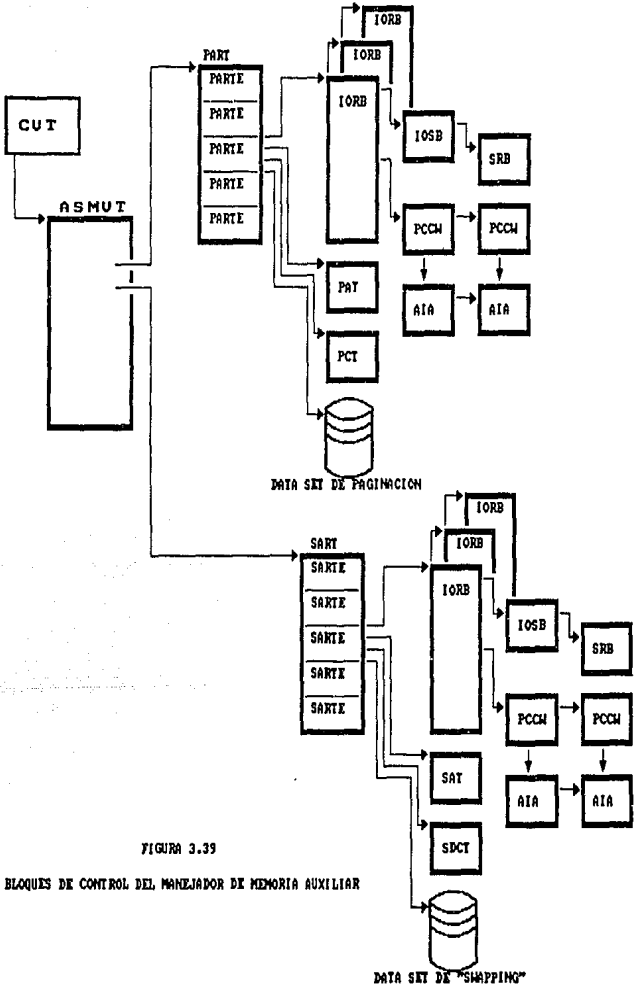


FIGURA 3.39

BLOQUES DE CONTROL DEL MANEJADOR DE MEMORIA AUXILIAR

El VSM se llama de nuevo al inicio de un paso del "job" para establecer el tamaño de memoria virtual del área privada.

Después de que el espacio de direcciones se crea, el VSM se llama para manejar el espacio libre en el. Esto generalmente se hace por medio de las macros "FREEMAIN" y "GETMAIN".

3.13.5.2 Macros "GETMAIN" y "FREEMAIN"

"GETMAIN" es la macro que se usa para requerir el alojamiento de una o más áreas de memoria virtual. Es la función del VMS que con más frecuencia se utiliza. El VSM lleva control de todo el espacio libre dentro de un espacio de direcciones.

La memoria virtual se divide en 4 rangos generales: SQA, CSA, LSQA y área privada.

Las páginas de memoria virtual pueden tener diferentes características, como por ejemplo:

-Localización en memoria virtual
-Fijas en memoria o paginables
-Residente debajo o arriba de la linea
-Protegidas contra lectura y/o escritura, etc.

Para simplificar el manejo de memoria virtual, esta se divide en "Subpools". Los "Subpools" están numerados del 0 al 255, cada uno de los cuales define un área de memoria virtual con características específicas.

Cuando se recibe un requerimiento de "GETMAIN" para un "Subpool" específico, este se satisface con un área de memoria dentro del rango que le corresponde a ese "Subpool". Los requerimientos de memoria pueden ser requerimientos de:

-Area Sencilla
-Lista de requerimientos
-Area variable (el que requiere memoria proporciona las longitudes mínima y máxima aceptables)
-Incondicional (el que requiere memoria terminará anormalmente su ejecución si no se cumple el requerimiento)
-Condicional (el que requiere memoria será notificado si no se puede satisfacer el requerimiento)

El tipo de requerimiento se pasa como parámetro en la macro y en todos los casos se puede especificar el número de "Subpool" y otros parámetros para especificar con más detalle la localidad de memoria que se quiere obtener. Por ejemplo, si se requiere memoria del "Subpool" 0, significa que se desea memoria del área privada con la llave del usuario que la requiere. También se puede especificar si el área de memoria se necesita arriba o abajo de la linea.

3.13.5.3 Servicios informativos del VSM

Existen Macros que permiten a los usuarios obtener información acerca del estatus de la memoria virtual. Estas macros son:

-VSMLIST
Regresa información acerca de memoria ya alojada. Suministrándole parámetros como "CSA", "SQA", "SP=0", ó "TCB=xxxxx" regresa un rango de direcciones correspondiente al área que se requirió.

-VSMLOC
Regresa información, vía un código de retorno, acerca de alguna dirección o rango de direcciones. Por ejemplo, para saber si cierto rango de direcciones ya fue adquirido.

-VSMREGN
Regresa la dirección de inicio y longitud de la región privada arriba y abajo de la linea del espacio de direcciones en donde se está corriendo.

3.13.5.4 Servicios de "Lote de Celdas"

Cuando un bloque de control se crea, se ejecuta una macro de "GETMAIN" para obtener un área de memoria virtual. Cuando el bloque de control ya no se va a usar, se ejecuta un "FREEMAIN" para hacer disponible de nuevo el área de memoria donde residía el bloque de control. Existen muchos bloques de control, como las SRBs que son creadas y liberadas muy frecuentemente, además de que tienen un período de vida muy corto. La sobrecarga de trabajo que se crea en este proceso de "GETMAIN"/"FREEMAIN" es tremendo.

Para evitar esto, se cuenta con "Servicios de Lote de Celdas". En lugar de hacer un "FREEMAIN" cada vez que se necesita crear un bloque de control y después liberarlo cuando ya no se usa, se adquiere (a través de "GETMAIN") un área mucho mayor de memoria, la cuál es lógicamente "Cortada" en trozos del tamaño del bloque de control formando "Celdas", que en su conjunto forman un "Lote de Celdas". Cuando un bloque de control se crea, se toma una celda del lote y cuando se libera, la celda se regresa como disponible. La cantidad de código ejecutado y el tiempo para hacer este proceso es mucho más corto que si se hiciera un "GETMAIN" y un "FREEMAIN".

Para requerir un servicio de lote de celdas se usa la macro "CPOOL" con los siguientes parámetros:

-"Construye" ("BUILD") : Construye un nuevo lote de celdas, indicándole el tamaño y el número de celdas a crear.

-"Obtiene" ("GET") : Obtiene una celda de un pool ya creado.

-"Libera" (FREE") : Libera una celda adquirida.

-"Borra" ("DELETE"): Libera todo el lote de celdas.

3.13.5.5 Funciones del sistema soportadas por el VSM

Ciertas funciones del sistema requieren soporte del VSM para inicializar áreas de memoria virtual. Estas funciones son:

-Inicialización del Sistema: Durante el IPL, El "Programa de Inicialización del Núcleo" ("Nucleus Initialization Program", NIP) le pasa control al VSM para alojar la memoria de SQA, CSA y LPA que se especifica en parámetros de la SYS1.PARMLIB.

-Inicialización de Espacios de direcciones: El VSM recibe control durante la inicialización del espacio de direcciones para inicializar las colas locales del VSM.

-Inicialización y Terminación de la Región Virtual: Las rutinas de "GETMAIN" y "FREEMAIN" se usan para alojar y liberar la memoria asociada a los bloques de control del sistema.

-Iniciación y terminación de tareas: El VSM recibe control durante el proceso de la macro "ATTACH" para inicializar las colas de VSM relacionadas a tareas. También recibe control durante la terminación de tareas para liberar toda la memoria relacionada con la tarea.

3.13.5.6 Contabilización de Memoria Virtual

El VSM usa diferentes esquemas de control para llevar la cuenta de las páginas libres y ocupadas para cada área; por ejemplo, usa un esquema distinto para llevar el control de las páginas de CSA que para las páginas de SQA. A continuación se describen los esquemas de contabilización de memoria de las áreas más importantes del espacio de direcciones.

-Páginas no asignadas del Area Privada

Las páginas libres del área privada, extendida y no extendida, se representan por una cadena de bloques llamados "Elemento de Cola de Bloque Libre" ("Free Block Queue Element" , FBQE). Un FBQE representa todas o una parte de las páginas dentro de un rango que no se han asignado a un "Subpool". La cadena de FBQEs se ancla a un "Descriptor de Región" ("Region Descriptor", RD).

El RD contiene apuntadores a la cadena de FBQEs así como a la dirección inicial del rango que representa, y su tamaño.

Existen 4 RDs para el área privada de cada espacio de direcciones. Están incluidos en el "Area Local de Datos" ("Local Data Area", LDA) de ese espacio de direcciones. (figura 3.40). Estos 4 RDs representan:

-Area V=V debajo de la línea
-Area V=R debajo de la línea
-Area extendida arriba de la línea
-Area de la región del sistema

-Páginas asignadas en el área privada

Después de que una página se asigna a un "Subpool", cualquier espacio libre en ella será contabilizado en la cadena de espacio libre del "Subpool" al que fue asignado. Cada "Subpool" dentro del área privada se representado por un "Elemento de Cola de Subpool" ("Subpool Queue Element", SPQE). Puede haber hasta 3 cadenas de SPQEs por cada TCB en un espacio de direcciones. Ver figura 3.41.

El SPQE se usa para identificar y describir un "Subpool". También contiene un apuntador al "Area de Anclaje de la Cola del Subpool" ("Subpool Queue Anchor Block", SPQA).

La SPQA se usa para anclar a los "Elementos Descriptores de Cola" ("Descriptor Queue Element", DQE) para ese "Subpool". La memoria en cada "Subpool" está dividida en 3 áreas:

+ Memoria abajo de la línea que debe ser respaldada abajo de la línea en memoria real.
+ Memoria abajo de la línea que puede ser respaldada en cualquier lugar en memoria real.
+ Memoria arriba de la línea que puede ser respaldada en cualquier lugar en memoria real.

Hay una cadena de DQEs diferente para cada uno de estos casos. Las tres cadenas están encadenadas a la SPQA.

El DQE representa una o más páginas que están asignadas a un "Subpool". Cada DQE contiene un apuntador a una cadena de "Elementos de Cola libres" ("Free Queue Elements", FQEs). Las FQEs de la cadena representan la memoria libre en las páginas asignadas a ese "Subpool".

El hecho de que la memoria ocupada esté dividida en 3 cadenas separadas de DQE/FQEs reducen el la longitud de la búsqueda cuando se hace un "GETMAIN".

-Páginas no asignadas de CSA

Las páginas libres en la CSA se contabilizan de la misma forma que las páginas de el área privada.

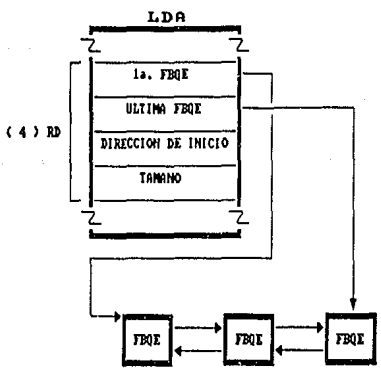


FIGURA 3.40
CADENAS DE FBQEs DE LA LDA

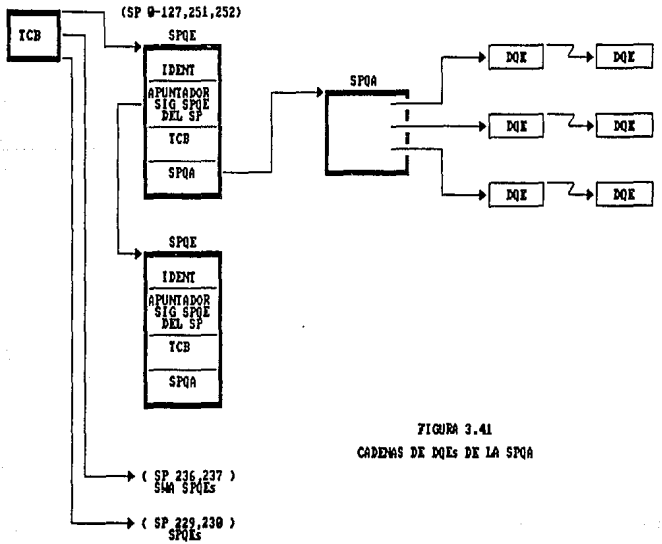


FIGURA 3.41
CADENAS DE DQEs DE LA SPQA

Las páginas libres se representan por FBQEs. Existen dos cadenas de FBQEs: Una para la CSA debajo de la línea y otra para la CSA arriba de la línea. Estas cadenas se anclan en 2 RDs, los cuales forman parte de el "Area de datos generales" ("General Data Area", GDA). Ver figura 3.42.

-Páginas asignadas de CSA

El espacio libre dentro de la páginas se contabiliza de una forma similar al del área privada. La estructura de bloques de control: SPQA, DQE y FQE es la misma que para el área privada, pero los "Subpools" deben estar anclados a un bloque de control común.

El VSM cuenta con una tabla de "Subpools" (CSASPT) para anclar los "Subpools" de la CSA. La CSASPT es una matriz de entradas llamadas SPTs que se ven y sirven para lo mismo que las SPQAs. La CSASPT tiene una SPT para cada "Subpool" y para cada llave de protección que exista en la CSA. Cada SPT tiene tres cadenas de DQE/FQEs ancladas a él. (figura 3.43).

- Memoria de SQA y LSQA

Los rangos de SQA y LSQA son áreas de alta utilización. El tamaño de estas áreas es grande, al igual que sus cadenas de memoria libre. Se usa el mismo esquema de bloques de control para llevar control del espacio libre en ellas, sin embargo para reducir el tiempo de búsqueda a lo largo de las cadenas para hacer un "Freemain" o un "Getmain", se cuenta con bloques de control adicionales:

- Páginas Asignadas de SQA

Cada pieza de memoria libre en una página asignada se representa por un "Elemento de Doble Liberación" ("Double Free Element", DFE). Las DFEs para cada "Subpool" de SQA están encadenadas en orden ascendente de acuerdo a la cantidad de memoria que representan (figura 3.44).

La cadena de DFEs para cada "Subpool" se ancla a la "Tabla de Anclaje de Colas de Tamaño" ("Size Queue Anchor Table", SQAT) para ese "Subpool".

La SQAT para cada "Subpool" contiene 24 apuntadores dentro de la cadena de DFEs. Estos apuntadores llegan a 24 DFEs "Falsos" que sirven como puntos de entrada en la cadena para tamaños dados de memoria libre.

Existe un DFE falso para representar tamaños de memoria desde x'8' hasta x'C0' bytes. Los tamaños representados crecen en incrementos de 8 bytes.

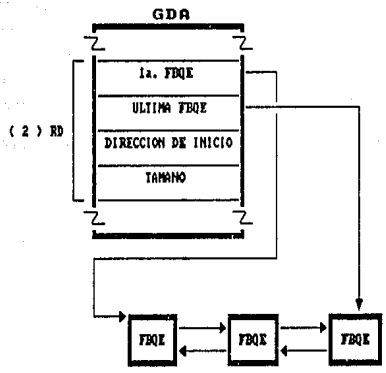


FIGURA 3.42
CADENA DE FBQEs DE LA GDA

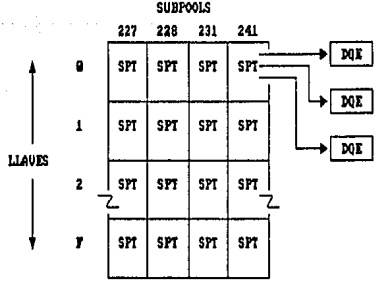


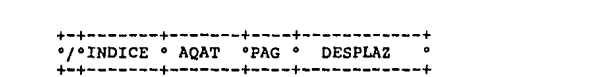
FIGURA 3.43
DISPOSICION DE LOS SPTs PARA CADA SUBPOOL

Los DFEs falsos son colocados antes de cualquier DFE que represente un área de memoria libre del tamaño equivalente o más grande al tamaño que representa el DFE falso.

Por ejemplo:

Se va a hacer un "GETMAIN" de x'62' bytes para el "Subpool" 226. El "Subpool" 226 es memoria de la SQA y sólo existe abajo de la línea. El VSM obtendrá la dirección de la SQAT del "Subpool" 226 de la GDA, e indexará por medio del apuntador al DFE falso de x'60' bytes. El VSM empezará entonces la búsqueda ahí y usará el primer DFE no falso que se ajuste al requerimiento. Existe una DFE para cada exceso de memoria y este se inserta en la cadena en el lugar adecuado.

Existen ocasiones en las que el VSM necesita buscar en la cadena de DFEs para buscar una dirección específica, o bien en el proceso de un "FREEMAIN", el VSM debe checar si la memoria recién liberada es adyacente a otra área de memoria libre, para combinarlas si es posible; para hacer esto el VSM mantiene una "Tabla de Anclaje de Colas de Direcciones" ("Address Queue Anchor Table", AQAT) y un "Indice de AQAT" (AQATINDX) para cada "Subpool". La dirección virtual se separa en campos de la siguiente forma:



Por ejemplo, se efectúa un "Freemain" para x'100' bytes en la dirección x'00EE2010'. (figura 3.44).

Se usa el INDICE para indexar a la AQATINDX, la entrada de la AQATINDX tiene un apuntador a la AQAT que representa a esta dirección.

El campo AQAT se usa como un índice dentro de la AQAT para encontrar una entrada de AQAT representa 16 páginas (16 Kbytes) de memoria. La entrada de la AQAT consistirá en un apuntador a la primera DFE para memoria libre en ese rango y un mapa de bits que representan las 16 páginas. El mapa de bits es usado para verificar que la página está en este "Subpool". En la figura 3.44 se representa el bit 2 del mapa como que la página 2 está asignada a este "Subpool".

Si la página está en el "Subpool", el VSM irá a la primer DFE y checará si es adyacente al área a la que estamos dando "FREEMAIN".

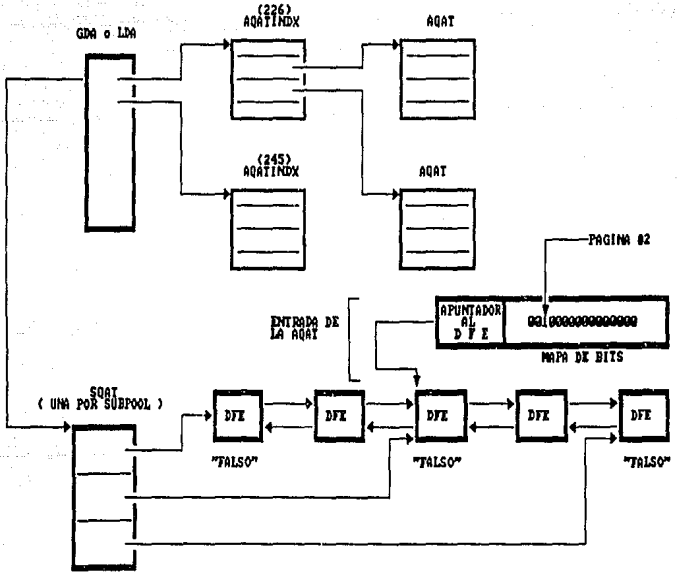


FIGURA 3.44

BLOQUES DE CONTROL RELACIONADOS A LAS AQATs

La DFE contiene un segundo juego de apuntadores, que están encadenados a direcciones dentro del rango de 16 páginas. Este mecanismo permite que el VSM inserte y combine las DFEs en la cadena de memoria libre.

La combinación AQATINDX/AQAT se usa para saber qué páginas están asignadas a un "Subpool" específico. También se usa para mantener la cuenta de todas las páginas libres dentro del rango de SQA.

- Páginas no asignadas de SQA

El VSM utiliza también el esquema AQATINDX/AQAT para mantener control de las páginas libres de SQA. Todas las páginas libres de SQA se asignan al "Subpool" 245. El "Subpool" 245 tiene muchas DFEs que representan todas las áreas libres en ese rango.

Cuando se necesita una página para satisfacer un "GETMAIN" en algún "Subpool", el VSM saca esta página del área libre del "Subpool" 245 y lo asigna al "Subpool" correspondiente.

Cuando esto sucede, el VSM actualiza la cadena de DFEs y el mapa de bits para el "subpool" 245 y para el "Subpool" que obtiene la página.

- Consideraciones de LSQA

El VSM usa el mismo mecanismo para mantener control del espacio de la LSQA que de la SQA. Todas las páginas libres de la LSQA se asignan al "Subpool" 255 y el VSM sólo maneja una AQAT para la LSQA. Además el VSM maneja 2 SQATs para la LSQA, una para la LSQA debajo de la línea y otra para la extendida. La SQAT para la LSQA tiene sólo 3 cadenas de DFEs, en lugar de 24.

3.14 SUPERVISOR DE I/O

En el capítulo II se describió brevemente la configuración básica del subsistema de I/O del Sistema/370-XA. En la presente sección se explicará la forma en que el Software (el MVS) ve a los dispositivos de I/O, como los controla y como interactúa con el hardware.

El objetivo del componente llamado "Supervisor de I/O" ("I/O Supervisor", IOS) es permitir al usuario accesar datos que residan en dispositivos de I/O.

Los programas deben especificar en el JCL que "Data Sets" quieren accesar y en que dispositivos residen dichos "Data Sets". Esto se hace por medio de una tarjeta de "Definición de Data Set" ("Data Set Definition", DD) en la cual se especifica el nombre del "Data Set", en qué volumen (Disco) reside, que longitud de registro tiene, etc. Estos datos son usados por el IOS para construir bloques de control para manejar estos "Data Sets".

Pero no solamente se pueden accesar datos contenidos en "Data sets" que residan en disco, también es posible que se quiera accesar a una terminal remota o a una impresora; la operación básica es la misma, sin embargo se tienen que usar "Métodos de Acceso" distintos para cada caso: Si por ejemplo se quieren accesar datos de un disco probablemente usemos el "Método de Acceso de Memoria Virtual" ("Virtual Storage Access Method", VSAM) o si se intenta establecer I/O hacia una terminal remota probablemente se use el "Método de Acceso Virtual para Telecomunicaciones" ("Virtual Telecommunications Access Method", VTAM),

3.14.1 Configuración de I/O

Cada dispositivo debe estar previamente definido para el sistema. Esto se hace como parte de un proceso conocido en su conjunto como "Generación del Sistema" ("System Generation", Sysgen). Para esto se utiliza un programa llamado "Programa de configuración de I/O" ("I/O Configuration Program", IOCP) al cual se le codifican una serie de macros que representan las características de cada uno de los dispositivos que se le definen. Esta definición se le hace al Hardware, de manera que cuando se da IPL no importa cual sistema operativo corra en este sistema, el sistema reconocerá los dispositivos que se le definieron por medio del IOCP.

También al Software se le tienen que definir los dispositivos que reconocerá el sistema. En MVS esto se hace a través de un programa llamado "Programa de Configuración del MVS" ("MVS Configuration Program", MVSCP). El IOS se encarga de hacer la "Conexión" entre las definiciones del software y las definiciones de hardware.

En el IOCP se declaran tres tipos de macros para definir a los dispositivos de I/O, a las unidades de control y a los "Caminos de Canal" ("Channel Paths"); estas macros son: IODEVICE, CNTLUNIT y CHPID respectivamente.

Como se vio en el capitulo II, cada dispositivo puede ser accesado a través de diferentes "Paths" que van del subsistema de canales a las unidades de control, y además cada dispositivo puede estar conectado a diferentes unidades de control.

Sin embargo, para el sistema, la vista lógica de estos dispositivos se representa por medio de un "Subcanal"; de tal manera que cada dispositivo tiene asociado un subcanal, sin importar por cuantos "Paths" distintos pueda accesarse este dispositivo ni a cuantas unidades de control esté conectado (figura 2.16(B)).

La representación interna de un subcanal es una "Palabra de Control de Unidad" ("Unit Control Word", UCW) la cual es construida a partir de la definición de una macro IODEVICE en el IOCP. En esta macro se define el "Número de dispositivo" ("Device Number", DEVNUM) con el que se conoce al dispositivo en el sistema. En la UCW se encuentra información como: La dirección de la CCW, el número de subcanal, el DEVNUM, la subclase de interrupción que ocasiona, e información acerca de los "Paths" que llegan a el dispositivo.

En la macro CHPID se codifican las definiciones de los "Paths" del subsistema de canales a los dispositivos de I/O; en esta macro se le da un "Nombre" a cada "Path" que llega a las unidades de control de los dispositivos. Se permiten hasta cuatro "Paths" distintos para accesar cada dispositivo.

En la macro de CNTLUNIT se define la dirección de la unidad de control que representa esta macro, y por que "Path" se puede accesar esta unidad de control.

Además existe una "Imagen" lógica de las unidades de control que se construye a partir de las definiciones de CNTLUNIT y CHPID. Esta imagen se conoce como una "Unidad de control lógica" y representa a una combinación de hasta 4 "Paths" que van hacia un dispositivo o grupo de dispositivos. Esta "Unidad de control lógica" no representa a una unidad de control específica, si no que sirve para encolar los requerimientos que haya sobre alguno de los "Paths". Esta "Unidad de control lógica" se representa por medio de una "Palabra de control de la unidad de control" ("Channel Unit Control Word" , CUCW). Un ejemplo de unidades de control lógicas se representa en la figura 2.16(A). Las unidades lógicas son "Arboles" independientes de "Paths", unidades de control y dispositivos, de manera que cada unidad de control lógica es independiente de todas las demás. Si existiera un sistema en el que todos los dispositivos estuvieran conectados a todas las unidades de control y estas a todos los "Paths", este sistema sólo tendría una unidad de control lógica.

Finalmente el subsistema de canal usa la "Dirección de la Unidad" ("Unit Address", UA) para comunicarse con el dispositivo: El primer dígito de la UA representa a la unidad de control y el segundo dígito representa al dispositivo.

Estos bloques de control los usan el Hardware y el Software para controlar a los dispositivos y residen en un área de memoria conocida como "Area de salvado de Hardware" ("Hardware Save Area", HSA).

Estos bloques de control "nacen" a partir de la definición que se le hace al Hardware a partir del IOCP y existen independientemente del sistema operativo que corra en el sistema. Estas definiciones que se hacen al IOCP se guardan en un "Archivo de Configuración de I/O" ("I/O Configuration Data Set", IOCDS) que reside en el controlador del procesador. Esta información se lleva a la HSA al momento del IPL.

A su vez, el MVS por medio del MVSCP necesita conocer a los dispositivos a los que tiene acceso y representa a los mismos con otros bloques de control. El IOS, como se mencionó, se encarga de hacer la conexión lógica entre los dispositivos de hardware y los dispositivos lógicos.

En MVS los dispositivos se representan por medio de un "Bloque de control de unidad" ("Unit Control Block", UCB). Esta UCB se construye a partir de las definiciones que se le hagan al MVSCP y contiene información como el DEVNUM, el nombre de la unidad y el "Número de Subcanal". Estas UCBs residen en el núcleo del MVS (figura 3.45).

Durante el IPL ("Initial Program Load") se carga el código del sistema operativo y entre otras cosas se construyen los bloques de control que representarán a los dispositivos de I/O. Este proceso se lleva a cabo por medio de un programa que se llama "Programa de Inicialización del Núcleo" ("Nucleus Initialization Program", NIP) el cual en el momento de la inicialización de I/O le cede controla al "Módulo de Inicialización de Recursos" ("Resource Initialization Module", RIM).

El RIM emplea dos instrucciones de máquina para guardar y extraer la información del subcanal que son: "Guarda Subcanal" ("Store Subchannel"), Nemónico STSCH, y "Modifica Subcanal" ("Modify Subchannel"), Nemónico MSCH. Estas dos instrucciones tienen el siguiente formato:

$$\text{STSCH } D_2(B_2)$$

$$\text{MSCH } D_2(B_2)$$

En ambos casos el segundo operando especifica la localidad de un bloque de control llamado el "Bloque de Información del Subcanal" (" Subchannel Information Block", SCHIB).

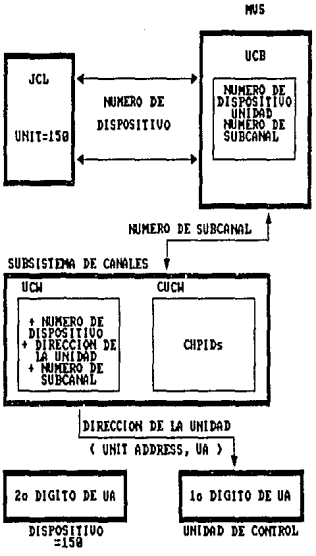


FIGURA 3.45

RELACION ENTRE LAS UCWs Y LAS UCBs

SCHIB

PARAMETROS DE INTERRUPCION			
LPM	PNOM	LPUM	PIM
MBI		POM	PAM
CHPID0	CHPID1	CHPID2	CHPID3
CHPID4	CHPID5	CHPID6	CHPID7
SCSW			

FIGURA 3.46

ESTRUCTURA DE UNA SCHIB

Este bloque de control (figura 3.46) guarda toda la información concerniente a un subcanal y se compone de 3 partes: Una campo de parámetros de interrupción, Información de los "Paths" y la "Palabra de Estatus de subcanal" ("Subchannel Status Word", SCSW). Realmente la SCHIB es la parte de la UCW a la que el programador tiene acceso.

La instrucción STSCH sirve para obtener la información del SCHIB y ponerla en la localidad de memoria del segundo operando, mientras que la instrucción MSCH sirve para obtener de la localidad de memoria del segundo operando la información que guardará en la SCHIB. En cualquiera de los dos casos se tiene que proporcionar en el registro 1 el número de subcanal al que se quiere accesar.

El RIM mapea la información contenida en las UCWs (a través de la información de la SCHIB que es a la que tiene acceso) con la información de las UCBs por medio de las instrucciones anteriores.

3.14.2 Flujo de una operación de I/O

Un programa de usuario comienza una operación de I/O emitiendo la macro "OPEN" por medio de la cual hace accesible los datos sobre los cuales quiere trabajar. Como resultado de la ejecución de esta macro se obtiene información concerniente al "Data set" que se quiere accesar. Esta información la obtiene de varios bloques de control:

- "Bloque de Control del Archivo del JOB" ("Job File Control Block", JFCB) el cual contiene información que se obtuvo del proceso de traducción del JCL y que contiene información acerca del "Data set" que se declaró en el JCL que se quiere accesar.

-"Bloque de Control de Data Set" ("Data Set Control Block", DSCB) el cual contiene información que describe la organización interna del archivo, es decir sus características lógicas (como por ejemplo si es secuencial o indexado).

-"Bloque de Control de Datos" ("Data Control Block",DCB) el cual debe ser construido por el programa de aplicación. Este programa de aplicación pone en la DCB información acerca de la organización del archivo y la localización de los datos que quiere accesar, es decir sus características físicas. Cuando concluye la ejecución de la macro "OPEN" en este bloque de control se pone toda la información necesaria para pasarle el requerimiento al método de acceso. Este bloque de control se llama "Bloque de Control del Método de Acceso" ("Access Method Control Block", ACB) en otros métodos de acceso como VSAM o VTAM.

La rutina de "OPEN" (figura 3.47) además construye un bloque de control llamado "Bloque de Extensión de Datos" ("Data Extent Block", DEB) que contiene información acerca de la localización física del "Data set" (es decir en que disco reside) y lo que mide este archivo. La DEB es en realidad una extensión de la DCB.

Por medio de otra macro indica que tipo de operación quiere efectuar (por ejemplo si quiere hacer una operación de lectura emitirá una macro "READ" o una macro "GET" o bien si quiere hacer una operación de escritura emitirá una macro "WRITE" o una macro "PUT"). Estas macros son macros del "Método de Acceso". Este Método de Acceso interpreta el requerimiento de I/O y determina qué recursos se necesitan para satisfacer el requerimiento. Las macros se expanden en una instrucción BALR ("Branch an Link Register") al código del método de acceso

El método de acceso construye otros dos bloques de control: el "Bloque de Control de I/O" ("I/O Block", IOB) en el que se guarda la información concerniente a la operación que se va a realizar y un "Bloque de control de Evento" ("Event Control Block", ECB) el cual se "Posteará" cuando la ejecución del I/O se complete. Esta ECB sirve como una bandera que le indica al despachador que el requerimiento por el cual espera la tarea se cumplió y que esta puede ser despachada de nuevo.

Además el Método de acceso se encarga de construir el programa de canal que llevará a cabo la operación de I/O, es decir la cadena de CCWs que le indicarán al subsistema de canal lo que tiene que hacer. El método de acceso coloca un apuntador a la 1a. CCW en el IOB. Por último el Método de acceso invocará a un "Manejador" ("Driver") del IOS, el cual, para el caso de programas de aplicación, se conoce como "Procesador EXCP" o simplemente EXCP. Entonces el código del método de acceso se queda en "WAIT" esperando la terminación de la operación. Esta "WAIT" terminará cuando el EXCP indique que la operación de I/O está completa "Posteando" a la ECB.

La llamada al EXCP se efectúa mediante una SVC (SVC 00) que invoca finalmente al código del mismo. El EXCP traduce la información que se le suministró por medio de las macros del método de acceso a una forma que entienda el subsistema de canales, construyendo los bloques de control que representan a la operación de I/O, como el "Bloque de Control del IOS" ("IOS Block", IOSB) el cual sirve para indicarle al IOS que operación quiere realizar y con que dispositivo. Tiene información tal como: Un apuntador a la UCB del dispositivo, un apuntador a la 1a. CCW del programa de canal, un apuntador a la SRB (que también construye el EXCP), etc.

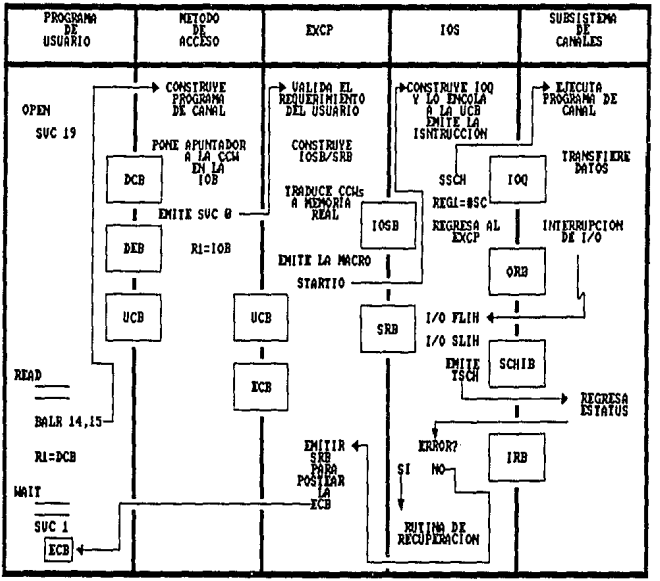


FIGURA 3.47

FLUJO DE UNA OPERACION DE I/O

3.14.3 Encolamiento de requerimientos

Cuando el IOS recibe el requerimiento de I/O del EXCP a través del IOSB, este encola el requerimiento a la UCB del dispositivo que se quiere accesar por medio de un "Bloque de Cola de I/O" ("I/O Queue", IOQ). Este IOQ es una representación interna de una operación de I/O. El IOS construye un IOQ por cada requerimiento de I/O que recibe. Este IOQ se encola a la UCB del dispositivo como se muestra en la figura 3.48. El IOQ contiene información tal como: un apuntador al siguiente IOQ, un apuntador al IOSB que representa la operación, etc.

Estos IOQ están ordenados por prioridades. La cola de las IOQs es de hecho una cola de despacho para procesar los requerimientos de I/O contra una UCB en particular; Los IOQs que aparecen primero en la cola son los de más alta prioridad.

Existe otro bloque de control llamado el bloque de "Cola de Clase de Dispositivo" ("Device Class Queue", DCQ) que sirve para los casos en que se necesita encontrar un dispositivo de determinada clase (por ejemplo una unidad de cinta) de entre todas las UCBs. Las DCQs se apuntan unas a otras en una cadena que se ancla en la CVT (como se muestra en la figura 3.49) en donde cada DCQ representa un tipo de dispositivo particular (es decir existe una DCQ para DASD, para cintas, para impresoras, etc) y estas DCQs apuntan a las UCBs las cuales se encadenan con las UCBs de los dispositivos del mismo tipo. Como se puede ver todas las UCBs se encadenan también de la CVT.

Las IOQs se procesan en orden cuando la UCB no está ocupada (es decir cuando no se está procesando I/O contra esta UCB).

3.14.4 Arranque de la operación de I/O

El IOS arranca la operación de I/O (es decir, al subsistema de canales en sí) por medio de la instrucción de máquina "Start Subchannel", Nemónico SSCH. El formato de la instrucción es el siguiente:

$$\text{SSCH } D_2(B_2)$$

En donde el segundo operando apunta a un "Bloque de Requerimiento de Operación" ("Operation Request Block", ORB) el cual contiene 3 campos principales: La dirección de la UCB, un campo de banderas (que contienen por ejemplo a la PUM, la llave del subcanal, etc.) y la dirección donde empieza el programa de canal. El dispositivo a arrancar se apunta por medio de el registro general 1, el cual debe contener el número de subcanal (figura 3.50).

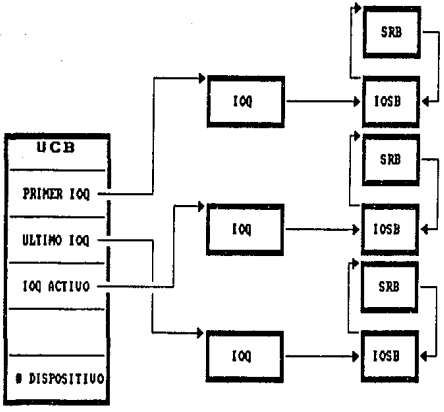


FIGURA 3.48
ENCOLAMIENTO DE REQUERIMIENTOS DE I/O

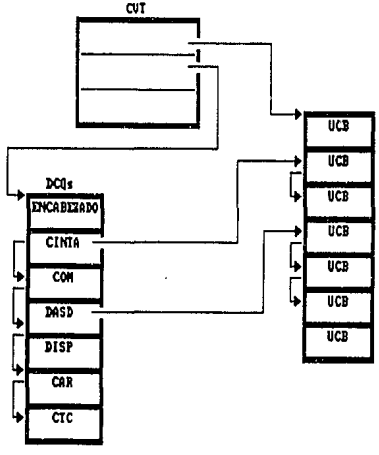
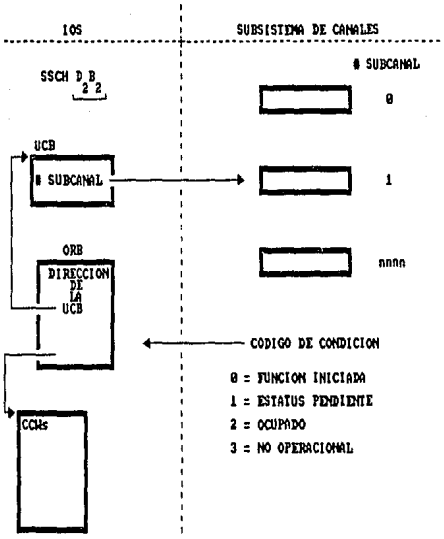


FIGURA 3.49
APUNTADORES DE LAS DCQs PARA CADA TIPO DE DISPOSITIVO



ORB

DIRECCION DE LA UCB		
LLAVE	BANDERAS	LPUM
DIRECCION DEL PROGRAMA DE CANAL		

FIGURA 3.50

EJECUCION DE LA INSTRUCCION "START SUBCHANNEL"

Una vez que se ejecuta la instrucción de SSCH se regresa controla al EXCP.

3.14.5 Selección de "Paths"

La selección del "Path" que se usará cuando un requerimiento llega al subsistema de canales originalmente era responsabilidad del IOS, sin embargo esta función se incluyó en el microcódigo (Hardware) del subsistema de canales. Esta selección se basa en información que está contenida en la SCHIB del subcanal que se quiera accesar. Esta información está contenida en forma de patrones de bits ("Máscaras") que indican la disponibilidad física y lógica de los "Paths" que van hacia las unidades de control así como de los "Paths" que fueron usados más recientemente. Estas máscaras son 8 bits que representan los paths (CHPID) que van al dispositivo, correspondiendo cada bit a la identificación de los "Paths" que están definidos en la SCHIB (figura 3.46) en el mismo orden. Estas máscaras son las siguientes:

-LPM ("Logical Path Mask"): Indica la disponibilidad lógica de cada "Path" al subcanal. Si un bit de la máscara vale 1 indica que el CHPID que le corresponde está lógicamente disponible.

-PNOM ("Path Non-operational Mask"): Indican si un "Path" es inoperativo desde el punto de vista del subcanal. Si un bit de la máscara vale 1 indica que el CHPID que le corresponde es inoperante.

-LPUM ("Last Path Used Mask"): Indica cual fué el último "Path" que se usó para accesar este subcanal. Solo uno de los bits vale uno, el cual corresponde al último path usado.

-PIM ("Path Installed Mask"): Indica con bits prendidos los "Paths" que están físicamente instalados.

-POM ("Path Operational Mask"): Indica el último estatus operativo conocido del "path".

-PAM ("Physical Availability Mask"): Indica que "Paths" están físicamente disponibles (lo cual no necesariamente tiene que coincidir con lo que está instalado).

La forma en que se asigna el "Path" que se va a usar es la siguiente: Si se especificó en el IOCP un "Path" particular para la unidad de control, se intentará usar este. Si este "Path" está o no operacional ocupado se usa un algoritmo de rotación entre los "Paths" que pertenecen a la misma unidad de control lógica (CUCW).

3.14.6 Interrupciones de I/O

Cuando el sistema de subcanal determina que el estado de un subcanal cambió provoca una interrupción de I/O. Cuando esta interrupción ocurre se guarda información en la PSA, información como el número de subcanal y parámetros de la interrupción que en su momento se pasaron al subsistema en la ORB (este parámetro no es más que la dirección de la UCB). Sin embargo en la interrupción no se presenta ningún estado de la misma, es decir, en ningún lugar aún se ha presentado la forma en que concluyó la operación de I/O.

Cuando ocurre el "Swapp" de PSWs toma control el FLIH de interrupciones de I/O, el cual determina que subclase de interrupción se produjo y le pasa controla al SLIH de I/O (ambas rutinas, es decir la FLIH y la SLIH son parte del código del IOS). El SLIH emite una instrucción "Test Subchannel", Nemónico TSCH para conocer como acabó la operación de I/O. El formato de la instrucción es:

$$\text{TSCH } D_2(B_2)$$

En donde el segundo operando es la dirección en donde se copiará el contenido de un bloque de control llamado "Bloque de Respuesta de Interrupción" ("Interruption Response Block", IRB) que contiene a la "Palabra de Estatus del Subcanal" ("Subchannel Status Word", SCSW) que se mencionó en el capitulo II y a una "Palabra de Estatus Extendida" ("Extended Status Word", ESW). Estos campos de la IRB proporcionan al IOS toda la información concerniente a como terminó la operación de I/O. (figura 3.51)

Cuando el IOS evaluó la terminación de la operación emite una SRB que posteará la ECB que representa el "WAIT" del método de acceso: esto se hace en forma asíncrona al resto del proceso del IOS. El IOS mientras tanto sigue trabajando y procesa otra IOQ si la hay contra alguna de las UCBs que están libres. Si no encuentra otra IOQ para procesar el IOS regresa control al EXCP.

3.14.6.1 Manejador de Interrupciones Faltantes

El IOS tiene un componente que se encarga de detectar condiciones anormales en el proceso de las interrupciones. Los problemas más frecuentes son la falta de interrupciones y el I/O "Caliente" ("Hot I/O"). El componente que se encarga de las interrupciones faltantes se llama el "Manejador de Interrupciones Faltantes" ("Missing Interrupt Handler", MIH).

Algunas operaciones de I/O pueden estar encoladas en el subsistema de canales por períodos extensos de tiempo mientras que la UCB refleja que está ocupada y el dispositivo físico no esta activo.

IRB

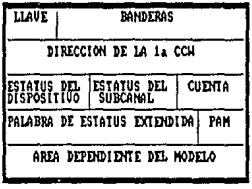


FIGURA 3.51

FORMATO DEL BLOQUE DE RESPUESTA DE INTERRUPCION (IRB)

Esto implica que el dispositivo se arrancó en algún momento y para el subcanal está activo todavía, sin embargo por alguna razón este no detecta un cambio en el estado del subcanal y no se produce una interrupción de I/O. Este tipo de situaciones sólo pueden ser detectadas por el MIH.

El MIH recorre las UCBs cada determinado tiempo buscando operaciones de I/O activas; si encuentra una, prende un bit en la UCB, bit que será apagado sólo cuando se presente una interrupción de I/O contra este dispositivo. Si el bit sigue prendido la siguiente vez que el MIH recorra las UCBs es posible que exista una condición de Falta de interrupción, así es que se hace un chequeo posterior.

Si el dispositivo está esperando a que se monte (como por ejemplo una cinta) el MIH no hace nada. Si la UCB se encuentra inactiva pero tiene requerimientos encolados, emite una instrucción de "Borra Subcanal" ("Clear Subchannel"), Nemónico CSCH y reintenta la operación. Si tiene un requerimiento de I/O de alta prioridad pendiente emite una instrucción de "Para Subcanal" ("Halt Subchannel"), Nemónico HSCH y reintenta la operación.

Existen dos clases de interrupciones faltantes: Aquellas que se detectan para instrucciones de SSCH y RSCH ("Reset Subchannel") y aquellas que se detectan para instrucciones de HSCH y CSCH.

Los intervalos de tiempo que usa el MIH para recorrer las UCBs en busca de interrupciones faltantes se codifican en el miembro IECIOSxx de la SYS1.PARMLIB.

Otro problema relativo a interrupciones de I/O que es posible detectar en MVS es el "Hot I/O". Esta situación consiste en recibir un número de interrupciones no solicitadas de parte de un dispositivo en particular, es decir, es exactamente lo contrario que la falta de interrupciones.

Cuando se detecta esta situación el código de "Hot I/O" toma control y se empiezan a contar las interrupciones. Cuando esta cuenta alcanza un valor de umbral predeterminado (que se codifica también en el miembro IECIOSxx) se le notifica al operador por medio de un mensaje en la consola.

Cuando esto sucede la primera vez, se emite una instrucción CSCH. Si vuelve a suceder se puede tomar alguna acción posterior (dependiendo de lo que el operador conteste al mensaje de "Hot I/O") acciones que pueden ser: Borrar el estatus del controlador y reintentar, remover la unidad de control, poner fuera de línea el "Path", etc.

3.15 SUBSISTEMA DE ENTRADA DE TRABAJOS

El MVS hace una clara distinción entre poner a punto los "Jobs" para ejecutarse, ejecutar el trabajo y remover la salida de este "Job" del sistema. El poner a punto los "Jobs" para ejecutarse y remover la salida que estos generen es responsabilidad de un componente del MVS llamado "Subsistema de Entrada de Jobs" ("Job Entry Subsystem",JES). La ejecución del trabajo en sí es responsabilidad del MVS.

3.15.1 Conceptos básicos del JES

El MVS procesa el trabajo de una instalación como "Jobs". Un "Job", como ya se ha mencionado, no es más que una serie de sentencias de JCL que le indican al sistema que programas procesar, que archivos accesar, la ubicación física de los mismos (cinta o disco), que dispositivos de salida va a usar (impresoras), etc.

Es función del JES interpretar estas sentencias del JCL, traducirlas, alojar los dispositivos que van a usar los programas, y ordenar por orden de prioridad los "Jobs" que tiene listos a ejecutar, para poder procesar primero los "Jobs" que tengan más prioridad; para lo cual hace uso del concepto de "Spool" que se mencionó en el capitulo I (esta técnica permite tener a un grupo de "Jobs" en disco y despacharlos según su prioridad y "Clase de Job").

El concepto de "Clases de Job" permite agrupar a "Jobs" que tengan requerimientos de dispositivos y características de proceso semejantes, por ejemplo se pueden asignar "jobs" que tarden mucho en procesarse a una misma clase de job; esto permite que la ejecución de los "Jobs" se pueda planificar y evitar que estos compitan demasiado por los dispositivos.

Además, una vez finalizado el proceso del "Job"", se encarga de manejar la salida que este "Job" haya producido (por ejemplo, una impresión) y mandarla a determinadas "Clases de Salida", mismas que representarán distintos formatos de impresión, por ejemplo. El uso de "Clases de Jobs" y "Clases de Salida" permite que el uso de los dispositivos sea más continuo y más eficiente.

Hablando en términos generales, el JES maneja los "Jobs" antes y después de su ejecución. El MVS maneja a los "Jobs" durante su ejecución.

Existen dos clases de JES llamados JES2 y JES3. Los dos hacen básicamente las mismas funciones descritas anteriormente, sin embargo la diferencia fundamental entre ambos es que el JES3 hace esto de manera centralizada, es decir todos los "Jobs" de un complejo confluyen a un sólo "Spool" y un procesador está dedicado únicamente de planificar y despachar a estos.

Existe un sólo JES3 que controla a todos los "Jobs" de todos los procesadores del complejo. El JES2 en cambio trabaja de manera distribuida, cada sistema tiene su propio JES2 y cada JES2 es responsable de manejar a estos, aun cuando el "Spool" puede ser compartido. En esta sección se hablará en general del JES2, haciendo en su momento incapié en alguna diferencia entre el JES2 y el JES3.

3.15.2 Funciones del JES2

El JES2 lleva a cabo su trabajo en 6 etapas (figura 3.52):

1) Fase de Entrada de "Jobs"

La entrada de los "Jobs" al sistema se puede hacer por diversos medios:
-Desde un terminal local (conectada directamente a canal a través de un controlador de terminales).
-Desde una terminal remota por medio de la facilidad que se conoce como "Entrada Remota de Jobs" ("Remote Job Entry", RJE).
-Desde una lectora de tarjetas local o remota.
-Desde otro nodo de una red de teleproceso por medio de la facilidad "Entrada de Jobs por Red" ("Network Job Entry", NJE).

Los "Jobs" que son leidos son colocados en la "Cola de Lectura" y el JCL así como una parte del mismo que se conoce como SYSIN (que representa los datos de entrada del Job) son colocados en el "Spool", y JES2 construye los bloques de control iniciales para identificar al "Job" y encontrar sus datos asociados en el "Spool". Si el "Job" se va a ejecutar en este nodo, se pone en la "Cola de Conversión" listo para la siguiente fase del proceso; si no es así se coloca en la "Cola de Transmisión" para ser enviado al sistema que ejecutará al "Job"".

2) Fase de Conversión

La función de la fase de conversión es convertir al JCL del "Job" a una forma conocida como "Texto Interno". La entrada de la conversión es el JCL que se guardó en el "Spool" en la fase de entrada, y la salida de este es el"Texto interno" que se guarda de nuevo en el "spool". Si no se encuentran errores en el JCL entonces se encola el "Job" a la "Cola de Ejecución" para la siguiente fase. Si se encuentran errores se encola directamente a la "Cola de Salida" junto con los mensajes de diagnóstico para procesarse por la fase de salida.

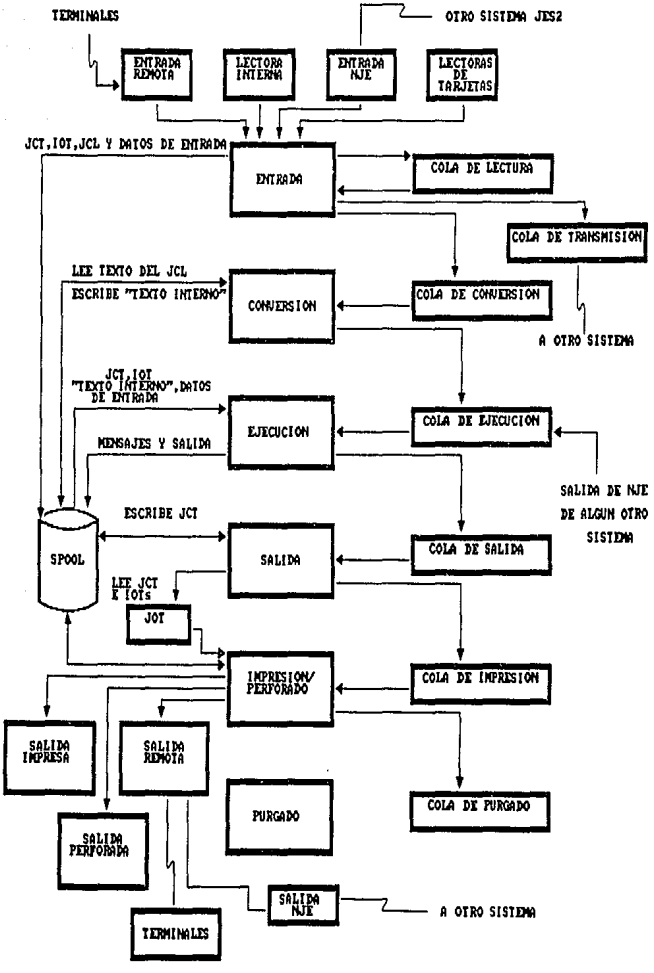


FIGURA 3.52
FASES DE EJECUCION DEL JES2

3) Fase de ejecución

Un iniciador de MVS (que como se explicó anteriormente es un Espacio de Direcciones ya activo pero "vacio", listo para usarse por un "Job") pedirá al JES2 un trabajo que ejecutar a través de servicios de la "Interfaz de Subsistemas" ("Subsystem Interface", SSI). El JES2 seleccionará de la cola de ejecución al "Job" de más alta prioridad dentro de la clase de trabajos del iniciador. El JES2 además proveerá al iniciador con todos los datos necesarios, el "Texto interno" y se crearán bloques de control como los JFCBs y SIOTs en la SWA (los bloques de control se verán más adelante). Una vez arrancada la ejecución del "Job" el JES2 proveerá servicios tales como: Lectura de los datos del SYSIN que reside en el "Spool", y escritura de datos de SYSOUT (que representa la salida del Job) al "Spool".

Cuando el "Job" acaba su ejecución, el código de terminación del MVS avisa al JES2, el cual mueve al "Job" hacia la "Cola de Salida" para procesarse por la fase de salida.

4) Fase de Salida

El objetivo de la fase de salida es procesar los "Data sets" de SYSOUT de los "jobs" que han completado su ejecución y construir "Elementos de Salida de Jobs" ("Job Output Elements", JOEs) que representan estos "Data sets" de SYSOUT. Estos JOEs se colocan en una "Tabla de Salidas de Job" ("Job Output Table", JOT). Los JOEs se construyen de acuerdo a las características de los "Data sets" de salida (destino, clase, formato, etc.). Cuando todos los "Data sets" de SYSOUT se procesaron y todos los JOEs se añadieron a la JOT, entonces el "Job" se encuentra listo para la fase de impresión ("Hardcopy"). Los JOEs se encolan de acuerdo a una prioridad basada en el tamaño de los "Data Sets" que es definida por el usuario.

5) Fase de Impresión

Como su nombre lo indica, el propósito de esta fase es enviar al SYSOUT a su destino final, el cual puede ser: Una impresora local, una impresora remota (RJE), otro nodo de la red (NJE) o una perforadora de tarjetas.

Las salidas de "Jobs" destinadas a otro nodo residen en la "Cola de transmisión". El JES2 que recibe la salida reencola al "Job" a su propia cola de salida o la reencola a su cola de transmisión para retransmitirla a otro nodo.

Si la salida pertenece a este nodo, el JES2 construye el programa de canal (CCWs) necesario para escribir el SYSOUT al dispositivo o usa macros de VTAM para transmitirla a un dispositivo remoto.

Una vez que la salida ha sido enviada a su destino, el "Job" es movido a la "Cola de Purgado".

6) Fase de Purgado

El proceso de purga del JES2 libera todo el espacio de "Spool" asociado con el "Job", todo la información relacionada al "Job" y cualquier otro recurso que haya tenido el "Job".

3.15.3 Organización interna del JES2

El JES2 se compone de 3 módulos principales:

-HASJES20: Es el módulo principal del JES2. Corre en su propio Espacio de Direcciones y está compuesto de aproximadamente 30 Secciones de Control (CSECTs).

-HASPSSSM: Es el módulo de soporte al subsistema. Proveé las rutinas de soporte para comunicarse con un componente del MVS llamado la "Interfaz de Subsistemas" ("Subsystem Interface", SSI) el cual es una interfaz que sirve como frente común para comunicar al MVS con sus subsistemas. Este módulo además contiene el método de acceso para usar al "Data Set" del "Spool".

-HASPINIT: Es el módulo de inicialización del JES2. Normalmente se carga en el Espacio de Direcciones del JES2 durante la inicialización del mismo y se borra una vez que dicha inicialización se completa.

Una consideración importante en el diseño del JES2 es que la tarea (TCB) principal del JES2 no debe emitir "WAITs" de MVS; es decir el JES2 no le cede control al despachador del MVS a menos que no tenga trabajo que hacer. Esto implica que ciertas rutinas del JES2 que necesariamente deben emitir "WAITs" deben correr como subtareas de la TCB principal del JES2 (es decir como TCBs hijas). Existen aproximadamente 8 subtareas del JES2 que realizan operaciones tan diversas como alojar dinámicamente dispositivos del JES o servir de interfaz con VTAM.

3.15.3.1 La interfaz de susbistema

Como se explicó, la SSI permite la comunicación entre el MVS y los subsistemas que residen en su propio Espacio de Direcciones (IMS, CICS, JES2, etc).

Las rutinas y bloques de control de esta interfaz residen en áreas comunes. Para cualquier requerimiento al SSI se necesita que el que requiere el servicio identifique que servicio quiere y quien quiere que se lo de.

4.2.2 Datos en Virtual ("Data in Virtual")

La facilidad de "Datos en Virtual" ("Data in Virtual", DIV) es un servicio del sistema que permite que el contenido de un archivo que reside en disco (DASD) permanentemente, sea accesible al usuario como si este estuviera cargado en su totalidad en memoria virtual.

Esta facilidad ofrece una nueva dimensión al significado de la memoria virtual, porque permite relacionar un rango de direcciones virtuales al contenido de un archivo residente en DASD. El archivo debe ser un tipo especial de "Dataset" accesado por VSAM, llamado "VSAM lineal" (figura 4.6).

Normalmente, una página en memoria virtual se relaciona a un "slot" de memoria auxiliar sólo cuando la página virtual se pagina fuera ("Page-Out"). Cuando esta página de memoria virtual se libera, también se libera el "Slot" de memoria auxiliar que la respalda.

En cambio, un "Objeto" que se accesa por medio de DIV siempre reside en DASD, no importando el estado de la memoria virtual que lo mapea.

Existen macros del sistema que establecen la relación entre el objeto de DIV y el rango de memoria virtual con el que se va a accesar. Cuando se accesan objetos de esta forma, no se hace ningún I/O físico hasta que se intenta accesar alguna localidad de memoria por primera vez. Los cambios que se hagan al objeto cuando está en memoria virtual son guardados al objeto permanente sólo hasta que se emite una macro para salvarlos. De hecho, sólo aquellas porciones que realmente fueron modificadas, se salvan físicamente al objeto residente en DASD.

Esta facilidad significa un enorme ahorro de esfuerzo para el programador y de sobrecarga para el sistema en aplicaciones que necesiten modificar y hacer cambios a objetos residentes en DASD. Si se accesara en la forma tradicional, todo el archivo tendría que ser cargado de DASD a memoria virtual (lo cual no sucede con DIV, pues sólo se cargan aquellas porciones que sean accesadas), además para hacer válidos los cambios al objeto, tendría que guardarse de nuevo todo el archivo de memoria virtual a DASD, a menos que el código del programa llevara un control de los cambios que se hicieron y sólo se salvaran aquellas porciones de datos que fueron cambiadas (lo cual se hace automáticamente con DIV).

Los objetos manejados por DIV no contienen registros de control insertados por el método de acceso, de hecho, estos pueden tener cualquier estructura.

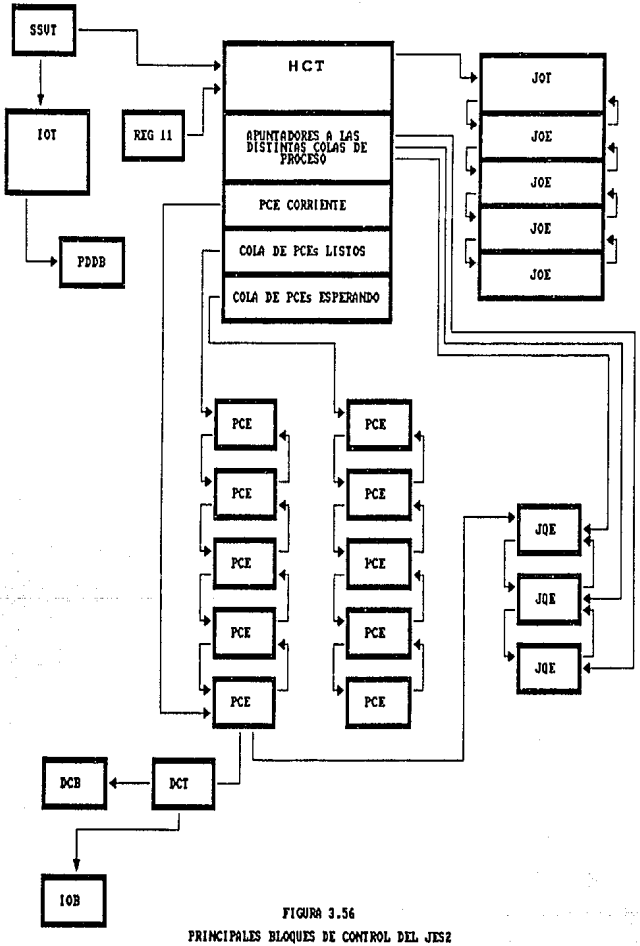


FIGURA 3.56
PRINCIPALES BLOQUES DE CONTROL DEL JES2

JQE es el bloque de control que se mueve de cola en cola (la cola de entrada, conversión, ejecución, etc.) durante la vida del "Job". Los JQEs son escritos colectivamente al "Data set" de "Punto de chequeo" ("Check Point Data Set") que sirve para mantener un registro del trabajo que el JES2 está efectuando en determinado momento y poder rearrancarlo si alguna falla sucede, exactamente con el mismo trabajo que estaba haciendo.

La "Tabla de Control De Jobs" ("Job Control Table", JCT) es otra representación de los "Jobs" que reside en el "Spool". Sirve como punto de anclaje a otros bloques que también residen en el "Spool". Contiene características del Job.

La "Tabla de I/O" ("I/O Table", IOT) representa la utilización que hacen los "Jobs" de el espacio de "Spool" y los de "Data Sets" que accesa el Job en su ejecución.

El "Bloque descriptor de Datos Periféricos" ("Peripheral Data Descriptor Block", PDDB) representan "Data sets" de SYSIN y SYSOUT individuales. Es análogo al par DCB/DEB del MVS. Contiene información del "Data Set".

El "Elemento de Salida de Job" ("Job Output Element", JOE) representa trabajo para el procesador de "Hardcopy" (impresoras y perforadoras). Es construido por el procesador de salidas y representa un conjunto de "Data Sets" de SYSOUT con similares características. El total de las JOEs constituyen la "Tabla de Salida de Jobs" ("Job Output Table", JOT).

3.15.3.3 El Despachador del JES2

El Despachador del JES2 es el elemento vital que asegura que el JES2 procese mientras tenga trabajo que hacer. El Despachador del JES2 se encarga de los PCEs tal y como el despachador del MVS se encarga de las TCB/RB. Un PCE puede ser despahable o no despachable. Si el PCE es despachable se encola a la "Cola de PCEs Listas" la cual se ancla en la HCT. Si el PCE es no despachable entonces es que está esperando por un recurso y estará encolado a la "Cola de PCEs en espera" que también se ancla a la HCT.

El Despachador selecciona al primer PCE de la "Cola de PCEs listos" y lo despacha, convirtiéndose así en el PCE actual (apuntado por un campo de la HCT). El PCE tiene un área de salvado estandard de MVS, y el despacho consiste en cargar los registros generales ahí guardados y hacer un BALR a la dirección contenida en el registro 15. El registro 13 siempre apunta al PCE.

Cuando el proceso debe entregar el control, emite una macro "$WAIT" (todas las macros de JES2 comienzan con el signo "$"). Esto provoca que el PCE sea encolado a la cola de espera y que la siguiente PCE sea despachada. Solo cuando ya no hay PCEs listas para ejecución, el JES2 emite un "WAIT" de MVS para regresar control al despachador de este.

Cuando un recurso por los que espera alguna PCE se hace disponible, se emite una macro "$POST" para indicar que el recurso está libre y que el PCE que está esperando por este recurso puede ser despachado. Esto se hace como parte del proceso del despachador: Cuando la cola de despacho está vacia, el despachador busca si se emitió un "$POST" contra algún recurso por el que estaba esperando un PCE, si lo hay se mueve el PCE de la cola de espera a la cola de PCEs listos.

CAPITULO IV.- EL SISTEMA/390 Y EL SISTEMA OPERATIVO MVS/ESA

4.1 LA ARQUITECTURA DEL SISTEMA/390

La arquitectura del Sistema/370-XA proporciona una capacidad de direccionamiento de hasta 2 GBytes derivada de direcciones de 31 bits que mapean un espacio de direcciones en MVS. Aunque pueden existir una gran cantidad de espacios de direcciones, en general solamente uno de ellos es direccionable en un determinado momento. La capacidad de "Doble Espacio de Direcciones" ("Dual Address Space", DAS) inherente a la arquitectura permite una limitada capacidad de accesar dos espacios de direcciones concurrentemente. Esta capacidad permite solamente el movimiento de caracteres entre un espacio de direcciones primario y uno secundario; el juego completo de instrucciones del sistema solamente es aplicable a un sólo espacio de direcciones, lo que limita la capacidad efectiva de direccionamiento de un programa a 2 Gbytes.

En respuesta a esta limitante (y algunas otras que se mencionaran más adelante) se diseñó el sistema "Arquitectura de Sistemas empresariales/370" ("Enterprise System Arquitecture", ESA/370) y posteriormente el Sistema/390.

La tendencia a lo largo del tiempo de las arquitecturas 360,370 y ahora la 390 es sin duda alguna el constante crecimiento de la memoria disponible.

Esta memoria fue expandida primero del Sistema/360 al Sistema/370 introduciendo el concepto de memoria virtual, a través de la "Traducción Dinámica de Direcciones" del Sistema/370, y después por medio del incremento del tamaño de las direcciones accesibles de 24 a 31 bits en el Sistema/370-XA.

Asociado con el incremento aparente de la memoria principal (que se mencionará en el resto del capítulo simplemente como "Memoria"), siempre ha estado presente el deseo de proveer por medio de la arquitectura el concepto de "Dominios", en donde un "Dominio" se puede definir como "Un conjunto de información y de autorizaciones para manipular esta información dentro de un sistema de computación", en otras palabras, "Celdas" aisladas de información en las que, mediante mecanismos de protección, los diversos usuarios de un sistema pueden o no tener acceso a la información contenida en estas.

Los dominios inicialmente fueron implantados en el Sistema/360 a través del uso de las llaves de memoria, llaves que se asignan a cada página de memoria y en donde para hacer una referencia o una modificación a esta página, la llave de la PSW tiene que coincidir. Mas tarde, en el Sistema/370, estos dominios fueron mejorados dando a cada usuario distinto un espacio de direcciones para su uso exclusivo, con la ayuda de la traducción dinámica de direcciones provista por hardware y el soporte apropiado del sistema operativo, creando el concepto de "Memoria virtual". En el Sistema/370-XA se agregó una facilidad de protección por segmento y por página.

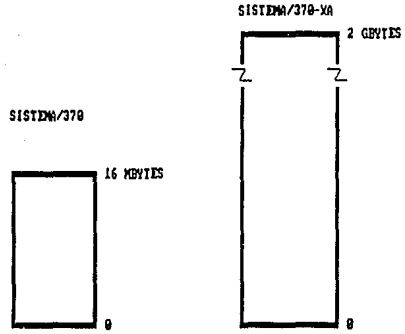
El Sistema/370-XA además introdujo el concepto de "Espacio de Direcciones Dual" ("Dual Address Space") que permite que, independientemente de que cada usuario tenga su propio espacio de direcciones totalmente "aislado" de los demás, estos espacios de direcciones puedan comunicarse unos con otros, sin poner en riesgo la integridad de los datos. El DAS lo conforman las instrucciones de máquina para mover datos de un espacio de direcciones a otro (MVCP y MVCS) y las instrucciones que permiten transferir control de un espacio de direcciones a otro (PC y PR).

Las nuevas facilidades del Sistema/390 son una evolución del DAS y pretenden solucionar problemas que se encontraron en esta facilidad.

Usando las instrucciones "Mueve Caracteres al Primario" ("Move Characters to Primary") y "Mueve Caracteres al Secundario" ("Move Characters to secondary") un programa puede mover datos de un espacio de direcciones a otro, pero existe la restricción de que esto sólo es posible cuando uno de estos espacios de direcciones sea en el que se están ejecutando las direcciones. Es decir, no se pueden mover datos entre dos espacios arbitrarios mientras se esté ejecutando en un tercero.

El Sistema/390 permite por arquitectura utilizar todo el juego de instrucciones de máquina sobre datos localizados en espacios de direcciones arbitrarios, ajenos al espacio de direcciones en donde se está ejecutando. Este hecho implica que cualquier programa corriendo en cualquier espacio de direcciones tiene la capacidad de direccionar y manejar los datos de cualquier otro espacio de direcciones, lo cual incrementa de manera impresionante la capacidad de direccionamiento del usuario; capacidad que era de 2 Gbytes en el Sistema/370-XA y que en el Sistema/390 es virtualmente infinita. En la figura 4.1 se esquematiza el crecimiento "Vertical" que tuvo el direccionamiento del Sistema/370 al Sistema/370-XA, y como el Sistema/390 representa un crecimiento de direccionamiento "Horizontal", al poder accesar virtualmente todos los espacios de direcciones a la vez.

CRECIMIENTO VERTICAL EN DIRECCIONAMIENTO



CRECIMIENTO HORIZONTAL EN DIRECCIONAMIENTO

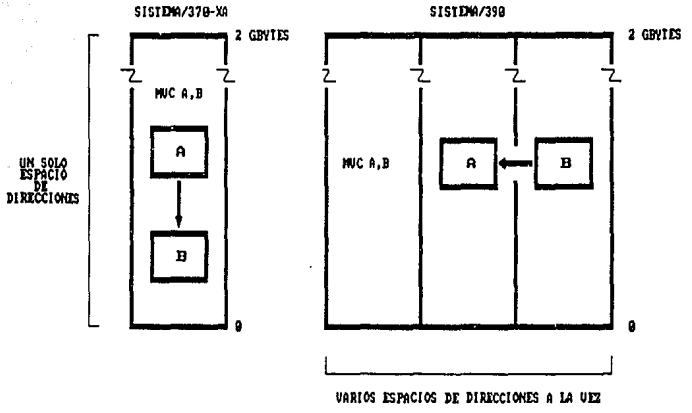


FIGURA 4.1

DIRECCIONES DE CRECIMIENTO EN EL DIRECCIONAMIENTO
DEL S/370 AL S/370-XA Y AL S/390

4.1.1 Registros de Acceso

Para poder lograr esto se hicieron cambios significativos en el hardware y en la mecánica de direccionamiento del sistema. El cambio más notable es la adición de 16 "Registros de Acceso" ("Access Registers") que van aparejados uno a uno con los registros generales que se usan como base para accesar cualquier dirección y que sirven para identificar a espacios de direcciones distintos.

Estos registros de acceso permiten que cualquier instrucción de máquina accese al mismo tiempo dos espacios de direcciones arbitrarios para obtener o guardar los datos sobre los que opera.

Por ejemplo, la instrucción "Mueve Caracteres" ("Move Character"), Nemónico MVC, y cuyo formato es:

$$MVC\ D_1(L,B_1),D_2(B_2)$$

copia "L" bytes que obtiene de la dirección del segundo operando (obtenida a través del registro base B_2 y el desplazamiento D_2) a la dirección del primer operando. Dando valores a los operandos (figura 4.2) tenemos:

$$MVC\ 10(5,3),20(6)$$

en donde R3=x'00001000'
y R6=x'00003000'

Si esta operación se ejecutara en el Sistema/370, se moverían 6 bytes (siempre se codifica uno menos del número que se va a mover) de la dirección formada por el contenido del registro 6 más un desplazamiento de x'20' (o sea la dirección x'00003020') a la dirección formada por el contenido del registro 3 más un desplazamiento de x'10' (o sea la dirección x'00001010'), en donde ambos operandos se encuentran en el mismo espacio de direcciones.

Si esta misma instrucción se ejecutara en el Sistema/390, habría 16 registros de acceso que contendrían cada uno un valor que correspondería cada uno a la identificación de un espacio de direcciones, de tal forma que cuando se formara la dirección del segundo operando (x'00003020') este se recuperaría del espacio de direcciones (el "Y" de la figura) que corresponda al registro de acceso 6 (el mismo número que el registro base del operando) y estos datos se copiarían a la dirección x'00001010' pero del espacio de direcciones (el "X" de la figura) designado en el registro de acceso 3 (de nuevo, el mismo número que el registro base).

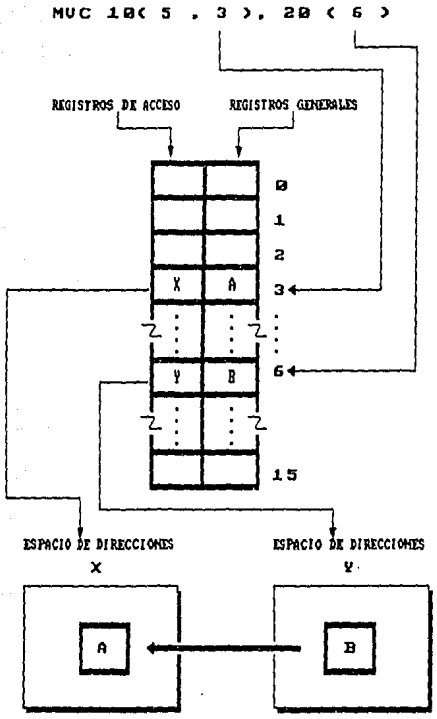


FIGURA 4.2
DIRECCIONAMIENTO POR MEDIO DE REGISTROS DE ACCESO

De esta forma, se pueden accesar al mismo tiempo, sin hacer nada adicional, dos espacios de direcciones diferentes al espacio de direcciones en donde se está ejecutando, teniendo la posibilidad de accesar simultáneamente hasta 16 espacios de direcciones sin cambiar el contenido de los registros de acceso.

Para que este mecanismo se ponga en funcionamiento es necesario que esté "prendido" un bit en la PSW que especifica este nuevo modo de direccionamiento llamado "Modo de Registros de Acceso".

Esto permite que exista total compatibilidad entre el modo de direccionamiento tradicional y el modo de direccionamiento de varios espacios de direcciones.

Los espacios de direcciones se relacionan a un registro de acceso mediante un mecanismo de traducción. El contenido del registro de acceso es una entrada ("Token") de una lista denominada "Lista de Acceso" ("Access List"), esta lista de acceso se usa para encontrar el "Origen de la Tabla de Segmentos" del espacio de direcciones correspondiente.

La arquitectura del Sistema/390 ofrece con este mecanismo la capacidad de compartir los datos de una manera controlada por el sistema, proporcionando los medios necesarios para restringir y otorgar la autoridad para accesar a los espacios de direcciones: El acceso se puede otorgar a un programa corriendo bajo una unidad despachable (TCB, SRB,etc) particular o se puede dar acceso a un programa sin importar bajo que unidad despachable esté ejecutando. Se puede también dar diferentes autoridades de acceso a diferentes rutinas en un mismo espacio de direcciones. La lista de acceso de hecho especifica que unidades despachables o que espacios de direcciones tienen acceso a que espacios de direcciones.

4.1.2 Manejo de Ligas entre Módulos

Otra mejora fundamental introducida con el Sistema/390 es el manejo de las áreas de salvado de registros generales por hardware.

Como se vio en el capitulo II, cada traspaso de control de un módulo a otro en la ejecución de un programa involucra la creación de un área de salvado (apuntada por el registro 13) que contiene a los registros generales del 14 al 12 (figura 2.12) además de apuntadores a las áreas de salvado anterior y siguiente (figura 2.13).

El Sistema/390 introduce una facilidad que permite que cuando se hace una llamada a un programa en otro espacio de direcciones o en el mismo en que se está ejecutando, los

registros generales, los registros de acceso y el estado del programa (PSW, registros de control, etc) se guardan en una "Pila" de Ligas entre módulos ("Linkage Stack"). A su vez, al regresar al programa original, estos registros se restauran automáticamente, así como el estado del programa, sin necesidad de codificar explícitamente instrucciones de máquina para ello, reduciendo considerablemente la cantidad de código y sobrecarga para el sistema.

Este nuevo mecanismo lo usan automáticamente las instrucciones "Llama Programa" ("Program Call", PC) y una nueva instrucción de máquina llamada "Regresa al Programa" ("Program Return", PR) que sustituye a la instrucción "Transfiere Programa" ("Program Transfer", PT). La instrucción PR permite al programa que fué llamado regresar control al programa que lo llamó sin necesidad de explícitamente recuperar el entorno de aquel (figura 4.3).

Además se introdujeron otras instrucciones que permiten el manejo de esta pila de ligas. La primera de ellas es la instrucción "Bifurca y Apila Registros" ("Branch and Stack Registers", BAKR) que permite (al igual que la instrucción BALR) bifurcar a la dirección indicada en el segundo operando y ligar la dirección de la siguiente instrucción al primer operando, con la ventaja extra de que los registros generales, de acceso y la PSW se guardan automáticamente en la pila (figura 4.4).

Además se crearon otras instrucciones para extraer registros de la pila, extraer la PSW de la pila o modificar los contenidos de esta pila.

4.2 EL SISTEMA OPERATIVO MVS/ESA

Con la introducción de la arquitectura ESA/370 y posteriormente del Sistema/390 se introdujo un nuevo MVS que explota las nuevas facilidades que el sistema tiene: el sistema operativo MVS/ESA o MVS de "Arquitectura de Sistemas Empresariales" ("Enterprise System Arquitecture", ESA).

El nuevo MVS/ESA (el más actual de la Familia del MVS) incluye nuevos conceptos y nuevas facilidades que surgen para aprovechar las mejoras en la arquitectura.

Los facilidades más importantes del MVS/ESA son:

- Los "Espacios de Datos" ("Data Spaces").
- Los "Datos en Virtual" ("Data in Virtual", DIV).
- Los "Hiper-espacios" ("Hiperspaces").
- Y el "Mira al lado Virtual" ("Virtual Lookaside Facility", VLF)

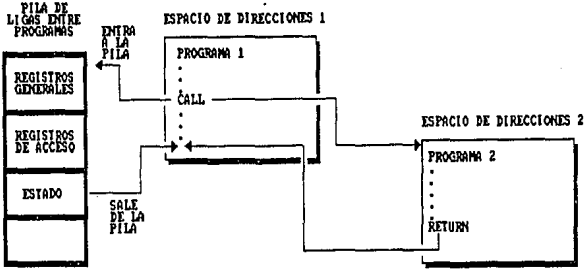


FIGURA 4.3

GUARDADO AUTOMATICO DE REGISTROS Y PSW EN LA PILA DE LIGAS

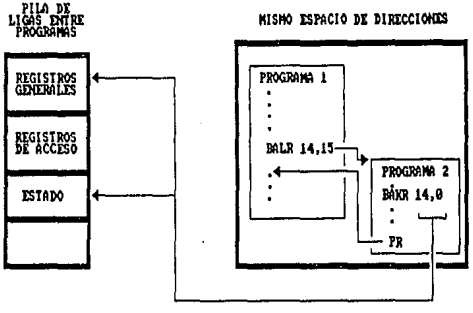


FIGURA 4.4

USO DE LA INSTRUCCION BAKR PARA PASAR CONTROL EN EL MISMO ESPACIO DE DIRECCIONES

4.2.1 Espacios de Datos ("Data Spaces")

Un espacio de datos ("Data Space") es un nuevo tipo de espacio de direcciones del MVS/ESA que contiene únicamente datos; esto es, no contiene áreas del sistema, ni tampoco se puede ejecutar código en estos espacios de datos. Solo contiene los datos que el usuario pone en el.

La implantación de este concepto se basa en la nueva capacidad del Sistema/390 de poder accesar con todo el juego de instrucciones de máquina a otros espacios arbitrarios distintos al espacio de direcciones en el que se están ejecutando las instrucciones.

El espacio de datos representa un mapeo lineal de memoria virtual que comienza en la dirección virtual cero (x'0') y puede ser hasta de 2 Gbytes, como cualquier otro espacio de direcciones (figura 4.5).

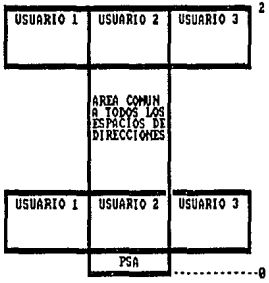
Un Espacio de datos recién creado aparece al usuario como inicializado a ceros (x'00'). Este espacio de datos no requiere de memoria real hasta que el usuario accesa la primera vez el espacio de datos. Al espacio de datos se le asigna una llave de protección de memoria igual al del usuario que pidió la creación del mismo. También se puede requerir que el espacio de datos sea protegido contra accesos de lectura de determinados espacios de direcciones o que sea compartido entre algunos o todos los espacios de direcciones.

4.2.1.1 El manejador de Espacios de Datos

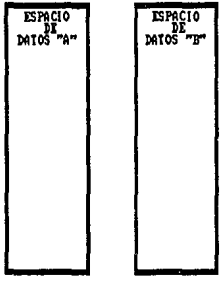
Para Manejar a los recién creados espacios de datos, el MVS/ESA cuenta ahora con otro componente llamado "Manejador de Espacios de Datos" ("Data Space Manager", DSM). Este manejador reside en un espacio de direcciones exclusivo para el, y además cuenta con varios espacios de datos para manejar la información que necesita para la estructura de control de todos los espacios de datos. Esto evita que se use memoria virtual del usuario del espacio de datos para contener información de control del mismo.

La macro DSPSERV ("Data Space Services") es la interfaz entre todo aquel que requiera un servicio relacionado con un espacio de datos (crear, borrar, etc.) y el DSM. Esta macro se expande en un "Program Call" (PC) al punto de entrada respectivo en el espacio de direcciones del DSM que proporciona el servicio requerido.

ESPACIOS DE DIRECCIONES



ESPACIOS DE DATOS



CARACTERISTICAS:

1.- ORIGEN EN LA DIRECCION X'0'

2.- NO TIENEN AREA COMUN

3.- NO TIENEN PSA

4.- NO PUEDEN CONTENER CODIGO EJECUTABLE

FIGURA 4.5

DIFERENCIAS ENTRE ESPACIOS DE DIRECCIONES Y ESPACIOS DE DATOS

4.2.2 Datos en Virtual ("Data in Virtual")

La facilidad de "Datos en Virtual" ("Data in Virtual", DIV) es un servicio del sistema que permite que el contenido de un archivo que reside en disco (DASD) permanentemente, sea accesible al usuario como si este estuviera cargado en su totalidad en memoria virtual.

Esta facilidad ofrece una nueva dimensión al significado de la memoria virtual, porque permite relacionar un rango de direcciones virtuales al contenido de un archivo residente en DASD. El archivo debe ser un tipo especial de "Dataset" accesado por VSAM, llamado "VSAM lineal" (figura 4.6).

Normalmente, una página en memoria virtual se relaciona a un "slot" de memoria auxiliar sólo cuando la página virtual se pagina fuera ("Page-Out"). Cuando esta página de memoria virtual se libera, también se libera el "Slot" de memoria auxiliar que la respalda.

En cambio, un "Objeto" que se accesa por medio de DIV siempre reside en DASD, no importando el estado de la memoria virtual que lo mapea.

Existen macros del sistema que establecen la relación entre el objeto de DIV y el rango de memoria virtual con el que se va a accesar. Cuando se accesan objetos de esta forma, no se hace ningún I/O físico hasta que se intenta accesar alguna localidad de memoria por primera vez. Los cambios que se hagan al objeto cuando está en memoria virtual son guardados al objeto permanente sólo hasta que se emite una macro para salvarlos. De hecho, sólo aquellas porciones que realmente fueron modificadas, se salvan físicamente al objeto residente en DASD.

Esta facilidad significa un enorme ahorro de esfuerzo para el programador y de sobrecarga para el sistema en aplicaciones que necesiten modificar y hacer cambios a objetos residentes en DASD. Si se accesara en la forma tradicional, todo el archivo tendría que ser cargado de DASD a memoria virtual (lo cual no sucede con DIV, pues sólo se cargan aquellas porciones que sean accesadas), además para hacer válidos los cambios al objeto, tendría que guardarse de nuevo todo el archivo de memoria virtual a DASD, a menos que el código del programa llevara un control de los cambios que se hicieron y sólo se salvaran aquellas porciones de datos que fueron cambiadas (lo cual se hace automáticamente con DIV).

Los objetos manejados por DIV no contienen registros de control insertados por el método de acceso, de hecho, estos pueden tener cualquier estructura.

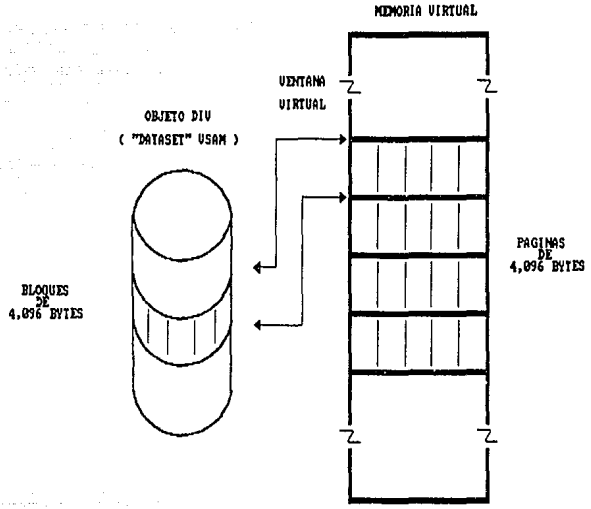


FIGURA 4.6

EJEMPLO DE CORRESPONDENCIA DE 1 BYTE A 1 BYTE ENTRE UN OBJETO DIV Y UN RANGO DE DIRECCIONES VIRTUALES.

La técnica DIV puede considerarse como un nuevo método de acceso para "Data Sets" residentes en DASD, sin embargo presenta ventajas con respecto a los métodos de acceso convencionales: No se necesita el uso de "Buffering", ni requiere de manejo de registros.

Para accesar un objeto manejado por DIV se usa normalmente un espacio de datos, que servirá como el rango de memoria virtual sobre el cual se va a operar el objeto (figura 4.7).

Además el I/O se optimiza al no ser necesario el cargar y guardar todo el objeto a la vez; esto lo hace automáticamente el sistema y sólo de las partes que se requieran: Sólo se leen las páginas que realmente se accesan y sólo se guardan las páginas que fueron cambiadas.

Los servicios que proporciona el DIV se llaman:

- IDENTIFY y ACCESS: Preparan al "Dataset" para su acceso.

- MAP: Define un rango de memoria virtual y un desplazamiento dentro del objeto para relacionarlos byte a byte. En este momento no se lee ningún dato, los datos se leen en el momento en que la aplicación accesa la memoria virtual. El rango de direcciones asignado puede ser desde una página hasta 2 Gbytes de un espacio de datos o un espacio de direcciones.

- SAVE: Guarda las páginas cambiadas.

- RESET: Desecha todos los cambios que se pudieron haber hecho a las páginas.

- UNMAP: Termina la relación entre la memoria virtual y el objeto DIV.

- UNACCESS y UNIDENTIFY: Cierra el objeto.

4.2.3 Hiperespacios ("Hyperspaces")

Los hiperespacios ("Hyperspaces") son un tipo diferente de espacios de datos que se respaldan en memoria expandida y eventualmente se paginan a memoria auxiliar. Un hiperespacio nunca se respalda en memoria real.

Estos hiperespacios permiten que las aplicaciones manejen grandes cantidades de datos sin hacer uso explícito de los registros de acceso.

Se puede pensar en un hiperespacio como en un espacio de datos que va a guardar datos, pero no los va a manipular.

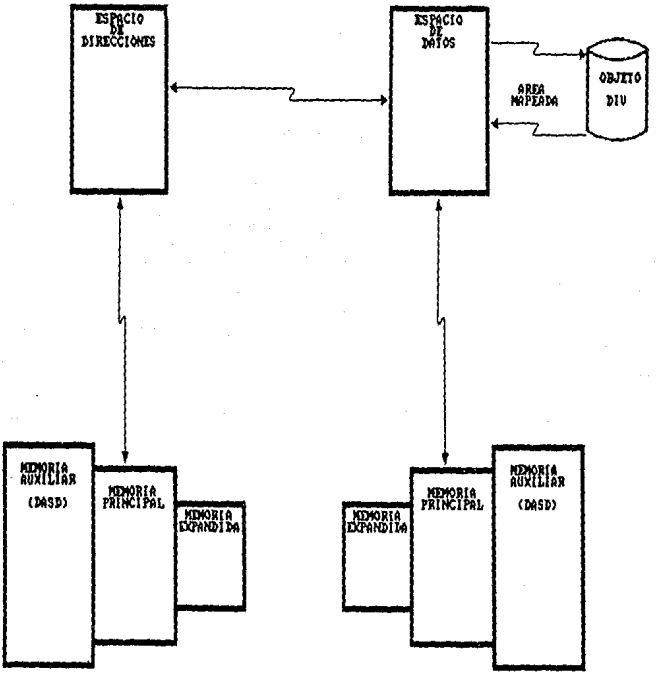


FIGURA 4.7

OBJETO DIU MAPEADO EN UN ESPACIO DE DATOS

El nombre de hiperespacios viene del hecho de que son un "Espacio de Datos de Alto Rendimiento" ("HIgh PERformance Data Space").

El propósito de los hiperespacios es que las aplicaciones usen a la memoria expandida como un substituto a las operaciones de I/O tradicionales, con lo cual se ganan importantes ventajas en rendimiento, pues es mucho más rápido hacer accesos a la memoria expandida que a un dispositivo DASD.

Como se mencionó, un hiperespacio difiere de un espacio de datos normal en el hecho de que nunca se usa memoria real para respaldar las páginas virtuales del hiperespacio. Otra diferencia es que los datos se guardan y recuperan de un hiperespacio en bloques de 4 Kbytes (esto debido al diseño físico de la memoria expandida, que se explicó en capítulos anteriores).

Existen dos tipos de hiperespacios: Aquellos que sólo se respaldan en memoria expandida y que nunca se paginan a memoria auxiliar (TYPE=CACHE) y aquellos que si pueden paginar a memoria auxiliar (TYPE=SCROLL). Los hiperespacios CACHE sólo los pueden usar usuarios autorizados y los hiperespacios SCROLL los pueden usar programas en "estado problema".

Para el manejo de estos existen macros de lectura (READ) y escritura (WRITE) diferentes para cada tipo de hiperespacio (Las macros para los hiperespacios tipo CACHE se llaman CREAD y CWRITE y para los tipo SCROLL se llaman SREAD y SWRITE).

4.2.4 Mira-al-lado Virtual

La facilidad de "Mira-al-lado Virtual" ("Virtual Lookaside Facility", VLF) es un nuevo servicio del MVS/ESA que reside en su propio espacio de direcciones y utiliza espacios de datos para guardar y recuperar objetos a los que se les da un nombre.

Los datos se guardan en espacios de datos y se pueden recuperar a nivel de bytes (a diferencia de un hiperespacio que sólo puede recuperar bloques de 4 Kbytes).

Los espacios de datos de VLF se manejan con la misma jerarquía de paginación que cualquier otro espacio de direcciones: En memoria real, memoria auxiliar y memoria expandida.

El concepto de "Mira-al-lado" es simplemente una forma muy eficiente de recuperar datos, la cual los mantiene en "memoria" mientras estos se necesiten.

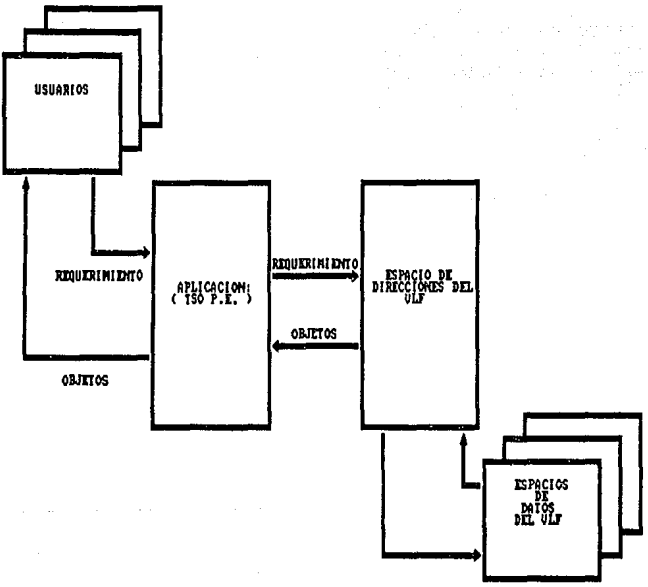


FIGURA 4.8

EJEMPLO DE COMO UNA APLICACION ACCESA OBJETOS MANTENIDOS EN VLF

El hecho de que los datos están en memoria no significa que vayan a estar siempre en memoria virtual (lo cual resultaría poco práctico) sino que siguen una estructura de migración de memoria real a memoria expandida y de ahí a memoria auxiliar dependiendo de la frecuencia con que se accesan estos datos.

El objetivo primario del VLF es permitir a otros componentes del MVS recuperar repetidamente objetos de alta utilización a los cuales se les da un nombre propio. Estos objetos pueden ser "Data sets" particionados, o catálogos por ejemplo. El uso de VLF elimina operaciones de I/O innecesarias manteniendo en memoria virtual a los objetos que más frecuentemente se utilicen (figura 4.9).

Sin embargo, al VLF lo puede utilizar cualquier usuario y no necesariamente un componente del MVS. El VLF puede usarse para guardar y recuperar una variedad muy grande de objetos.

Las aplicaciones que se pueden beneficiar del VLF son aquellas en las cuales existen múltiples usuarios de los datos o bien en las que los datos tiene una frecuencia de uso muy alta.

Los datos guardados en VLF están sujetos al robo de páginas, así es que estos datos deben tener una frecuencia de uso suficientemente alta para garantizar su estancia en memoria real o memoria expandida. Además, el VLF funciona mejor con objetos de tamaño pequeño, debido a que se usa menos memoria virtual para conservarlos y ahorrarse el I/O. Los objetos muy grandes, cuando no se usan frecuentemente, puede ser que tarden más en recuperarse usando VLF que usando I/O tradicional.

Los objetos manejados por VLF estructuralmente divididos en "Clases" y cada objeto se reconoce por un único nombre compuesto de un calificador "Mayor" y un calificador "Menor".

En resumen se puede decir que las mejoras y nuevas facilidades introducidas en el Sistema/390 y el MVS/ESA se dirigen a un mayor uso de la memoria virtual, evitando en lo posible el uso de operaciones de I/O. Esto tiene una razón: La memoria real o principal ha venido decrementando su precio a lo largo de los años, por lo que ahora resulta más barata y es posible tener mayor número de Bytes de memoria real, lo que antes era verdaderamente un lujo. Además, el hecho de manejar los datos en memoria virtual en lugar de traerlos de memoria auxiliar cuando se necesitan hace que el rendimiento de los programas que usan estas facilidades se incremente de manera notable.

CONCLUSIONES

El MVS, tal y como se ha mencionado a lo largo de esta tesis, es el sistema operativo que va a la vanguardia y que ofrece más capacidad de los sistemas operativos que existen para explotar la arquitectura de los Sistemas/370 y Sistema/390 de IBM.

Su éxito es tal, que aunque existen un gran número de fabricantes de hardware que ofrecen equipos compatibles con la arquitectura 370 o 390, todavía no existe un fabricante de software que ofrezca un sistema operativo que intente competir con el MVS.

Esto es debido a que no existe otro sistema operativo que permita el manejo de cargas de trabajo tan grandes, que proporcione una integridad y seguridad de datos tan buena y que sea tan confiable como el MVS. Como resultado de esto, se puede afirmar que la mayoría de las 500 compañías más importantes del mundo cuentan con al menos una plataforma basada en arquitectura 370 o 390 corriendo MVS (o bien, corriendo varios MVS bajo VM).

El MVS proporciona al usuario una disponibilidad tan grande que no es raro encontrar instalaciones que trabajan 24 horas al día durante los 7 días de la semana y que no realizan un IPL en varios meses.

Esta disponibilidad tan grande es un resultado obvio de la filosofía de diseño del MVS; Aproximadamente el 80% del tiempo de proceso de cualquier "Job" corriendo bajo MVS, se procesa en estado supervisor, es decir corriendo código del sistema operativo. Esto significa que 4/5 partes del tiempo de proceso, el sistema operativo asegura la integridad, la disponibilidad y la corrección de errores de las aplicaciones que corren bajo él.

Se ha demostrado durante todo la evolución del MVS, que su código es sólido y que raras veces se encuentran problemas de defecto en la parte medular del producto. La mayoría de las veces los problemas son debidos a errores de usuario y como tales, son difíciles de diagnosticar, porque es muy difícil encontrar un problema exactamente igual (que no sea de defecto) en la base de datos de IBM. Es en estas ocasiones en que se hace tan necesario un documento de referencia como esta tesis.

La importancia actual del sistema operativo MVS y de la arquitectura 370 y 390 en general es indiscutible. Sin embargo, ¿ Cuál será su importancia en el futuro?, ¿ Serán los sistemas abiertos la tendencia durante la próximas década ?, ¿ Tiende a desaparecer la tecnología de "Mainframes" ?.

La importancia del MVS en el futuro mediato es prácticamente la misma con la que cuenta hasta ahora: El MVS y la arquitectura 390 se caracteriza por ser una decisión estratégica en cuanto a los sistemas de información de las empresas se refiere, esto es debido a la continuidad que le proporciona el proveedor (IBM) al soporte de los productos existentes. Además (como se vio cuando se habló de la evolución del MVS) IBM siempre ha tenido un compromiso con sus clientes por mantener la compatibilidad entre sus nuevos productos y los que ya existían anteriormente. Esto significa para las empresas que invierten muy fuertes cantidades de dinero en estos sistemas, que su inversión estará segura conforma pasa el tiempo. Este hecho aunado a la importancia y solidez de la compañía aseguran una estancia todavía muy larga del MVS en el mercado de la informática.

Aún considerando los rivales más grandes de la tecnología de "Mainframes" (y por consiguiente del MVS) se pueden hacer predicciones favorables a esta tecnología.

Por ejemplo, es un hecho que los "Sistemas Abiertos" prácticamente están dominando el mercado. En el sentido estricto de la definición de un sistema abierto (según la IEEE), un sistema abierto se define como "Un conjunto de estándares comprensibles y consistentes relacionados a la tecnología de la información y relacionados a perfiles que especifican interfaces, servicios, y formatos de soporte para implantar la interoperabilidad y portabilidad de aplicaciones, datos y gente".

Sin embargo, realmente la concepción de lo que es un sistema abierto y hacia donde se dirige el mercado de sistemas abiertos es muy subjetiva.

Una visión de un ambiente de sistemas abiertos sugiere un sistema operativo común a muchas plataformas de hardware, que permita que los datos y aplicaciones fluyan libremente de sistema a sistema y que todos en conjunto sean capaces de operar en conjunto. Este tipo de enfoque de un sistema abierto generalmente se enfoca alrededor del sistema operativo "UNIX" (Propiedad de UNIX Systems Laboratories, Inc. que es un consorcio de fabricantes de equipo de computación formado por AT&T, SUN y NCR entre otras). De hecho, una forma de implantar el concepto de sistema abierto es a través del sistema operativo UNIX, sin embargo no es la única forma ni la más eficiente.

La visión más amplia y realista de un sistema abierto surge de la necesidad de los usuarios de resolver los problemas de manejo de información a los que se enfrentan, usando todas las fuerzas de los fabricantes de hardware y software sinergísticamente. Esto implica:

1) La habilidad de adquirir productos de hardware y software de cualquier proveedor, protegiendo su inversión en el desarrollo de nuevas aplicaciones, a la vez que sean capaces de operar con los sistemas ya instalados

2) La habilidad de implantar una infraestructura de sistemas de información que permita el crecimiento sin afectar el servicio.

3) La habilidad de manejar y accesar información en un ambiente de muchos proveedores de equipo y software.

4) Y la habilidad de proteger la inversión existente en hardware, software y conocimientos de la gente involucrada al mismo tiempo que se explote nueva tecnología.

Como se ve, el concepto de sistemas abiertos va mucho más allá de la concepción simplista de un sistema basado en UNIX; el concepto de sistemas abiertos tiene como fin último poner orden en el aparente caos de proveedores y consumidores de sistemas de información.

Es por esto que el papel de las arquitecturas 390 y 370 dentro de el contexto de sistemas abiertos es decisiva, tanto como lo será cualquier otro tipo de arquitectura que proporcione al usuario una ventaja competitiva. El Sistema/370 y 390 tienen mucho que ofrecer en cuanto a manejo de I/O se refiere: Todavía no existe una arquitectura que se le supere en cuanto al manejo de cargas de trabajo y I/O, y el sistema operativo MVS es el que mejor explota esta capacidad.

Ahora bien, el que una arquitectura o un sistema operativo sea capaz de convivir en un sistema abierto, implica que este debe cumplir con una serie de estándares. Estos estándares permiten que arquitecturas y sistemas disímbolos puedan comunicarse unos con otros, al mismo tiempo que den la sensación al usuario de que está tratando con un ambiente homogéneo.

Existe un comité en la IEEE, llamada el "Comité Técnico en Sistemas Abiertos" ("Technical Commitee on Open Systems", TCOS) que se encarga de evaluar, probar y publicar los estándares y especificaciones para un ambiente abierto de sistemas operativos. Estos estándares y especificaciones se conocen como "POSIX" ("Portable Operating Systems Interface for Computer Environments") o Interfaz Portable de Sistemas Operativos para Ambientes de Computación.

Estas especificaciones son una evolución que tiene sus inicios en el sistema UNIX, pero no piden como requisito obligatorio para implantar sus recomendaciones que se cuente con una plataforma UNIX.

Las versiones más recientes de los sistemas operativos para casi todas las plataformas que produce IBM acatan ya los estándares propuestos en POSIX. Entre estos, desde luego, el MVS. Las últimas versiones del MVS cuentan con interfaces que acatan estos estándares y que lo convierten en un candidato para convivir en sistemas abiertos.

Pero, ¿ Que tiene de malo el concepto de un sistema abierto basado en UNIX ?. Realmente nada, sin embargo tampoco es lo bueno que podría ser. Si consideramos que cada fabricante de equipo tiene su propia filosofía de diseño de equipos y que cada uno busca proporcionar a sus clientes alguna ventaja en particular, se puede intuir que es más eficiente permitir que cada equipo cuente con su propio sistema operativo, que fue diseñado ad-hoc para sacar el máximo provecho de esa arquitectura en particular, y que juntos convivan en un sistema cooperativo, más que competitivo.

Otro tema muy de moda en el medio de la informática y en las publicaciones especializadas es la idea de que las computadoras personales (PCs) ya pueden competir en velocidad con una "Mainframe" (factor que hasta hace unos años era la principal diferencia entre una tecnología y otra) a una fracción pequeñísima de su costo. Esto aunado al hecho de que ofrecen un ambiente mucho más amigable al usuario, ofrecen tiempos de desarrollo de aplicaciones muy cortos y funciones más ricas en variedad.

Esto hace dudar a algunas personas en el futuro de los ambientes basados en "Mainframe". Algunas personas vislumbran el futuro de los sistemas de información de las organizaciones basados en redes locales de PCs.

Uno de los factores que ha influido en el repudio a los sistemas centralizados es el "satff" de soporte, que usualmente aparece como intransigente y burocrático, más que orientado a satisfacer a sus usuarios. Además de que, es común la creencia de que una red local no implica trabajo de administración ni soporte.

Otro de los factores que ha hecho popular esta creencia es el pobre entendimiento de las diferencias de arquitectura entre una plataforma y otra.

Por ejemplo, si se hace una prueba de velocidad de una "Mainframe" pequeña contra una PC con procesador 80386, que conste de procesar digamos 1000 instrucciones en FORTRAN, seguramente la PC sobrepasará en velocidad a la "Mainframe" o quedarán casi iguales. Si además de esto consideramos el precio de la PC que será algo así como 3000 veces más barata que una "Mainframe" parece obvio que las "Mainframes" no tienen ya nada que hacer en el mercado.

Esto desde luego es una concepción muy simplista, que está descartando la capacidad más importante de una "Mainframe" y es el manejo de grandes volumenes de I/O concurrentemente con el proceso que pudiera necesitar. En la misma prueba del ejemplo, la "Mainframe" podría haber estado atendiendo a 2000 terminales al mismo tiempo que haciendo trabajar a 100 impresoras al tope de su capacidad y haberse tomado unos segundos más en completar el proceso al que se le vio sometida. Una PC, difícilmente podría hacer trabajar a la mitad de su capacidad a una sola impresora, dedicándose de completamente a esto.

El diseño de una "Mainframe" es manejar un ambiente de computación en el que los procesadores, le memoria principal, los datos, dispositivos de comunicación, impresores, etc. se comparten entre un grupo de usuarios que usan al sistema simultáneamente. Una característica fundamental de las "Mainframes" es que son ricas en capacidad de I/O, pero su relación de MIPs a cantidad de datos es pequeña.

El diseño de una PC en cambio es el de maximizar le productividad y el rendimiento de un usuario, proporcionándole acceso rápido, directo, exclusivo, y no compartido al procesador. Además cuenta con una excelente interfaz para el usuario que está dedicada completamente a este. En esta plataforma, al contrario de las "Mainframes", se tiene una capacidad de MIPS en relación a los datos que maneja muy alta, porque la PC por arquitectura no puede manejar gran cantidad de información al mismo tiempo.

Otro factor importante es considerar el costo real de una plataforma y otra. Aparentemente no hay discusión en cuanto a que la PC es mucho más barata que una "Mainframe". Sin embargo, el punto a considerar aquí es la adecuada selección del tamaño del equipo para satisfacer las necesidades del usuario.

El procesador de la PC se encuentra inactivo la mayor parte del tiempo, usando su exceso de poder de procesamiento para atender con una responsividad casi instantánea al usuario y para crear la interfaz gráfica con la que se sienten tan a gusto la mayoría de los usuarios. El procesador de estas PCs se encuentra 100% activo sólo durante breves instantes de tiempo, refrescando una presentación gráfica o reevaluando una hoja de cálculo. La utilización de las PCs fluctúa en promedio alrededor del 1% de su capacidad. De esto se concluye que la mayoría de estos "MIPs" baratos se desperdician esperando a que el usuario tecleé "Enter".

En cambio, una "Mainframe" típicamente se encuentra (Tomando como base un año) aproximadamente el 70% del tiempo haciendo trabajo productivo.

Además, considerando el costo como elemento de decisión fundamental, se deben tomar en cuenta otros factores como instalación, entrenamiento, soporte, software, mantenimiento, etc. y no sólo el costo de un componente en particular. El hardware sólo representa el 25% de los costos de un sistema de información.

Bajo esta perspectiva, se han hecho estudios que demuestran que un sistema basado en PCs resulta más caro si los usuarios a los cuales está dedicado el equipo realizan menos de 300 transacciones al día.

En conclusión se puede decir, que si bien los sistemas basados en PC se han hecho muy populares y poderosos, y se harán aún más, existe una diferencia fundamental en el tipo de aplicaciones que son susceptibles a implantarse bajo una plataforma de PC y una plataforma de "Mainframe". Las transacciones basadas en "Mainframe" generalmente toman unos cuantos millones de instrucciones en completar, porque el proceso que se hace sobre los datos es sencillo generalmente, lo fuerte aquí es el volumen de información que se maneja. Una aplicación en PC seguramente usará cientos de Kbytes para hacer su trabajo, pero la cantidad de datos que puede manejar es muy modesta. Otra característica que tienen las aplicaciones basadas en "Mainframe" es su alta auditabilidad, seguridad y confiabilidad.

En el extremo de las aplicaciones, encontramos a las aplicaciones numéricas intensivas que involucra el manejo de imágenes, manejo de cálculos de elemento finito o de CAD/CAM (diseño de máquinas asistido por computadora). En estos ambientes, aparte de una gran capacidad de proceso, se necesitan manejar inconmensurables cantidades de datos al mismo tiempo. Para este tipo de aplicaciones las "Mainframes" todavía son la única opción.

Las aplicaciones que son aptas para implantarse en PC son muy distintas. Las aplicaciones que requieren una fuerte interfaz gráfica con el usuario, que tienen una función fija, como hojas de cálculo, o procesadores de palabras, y además que su expectativa de vida sea de mediana a pequeña son candidatos ideales para desarrollarse en PC.

Se debe tener mucho cuidado al seleccionar la plataforma considerando la vida útil que tendrá la aplicación. Generalmente las aplicaciones basadas en "Mainframe" se proyecta que den servicio durante muchos años, pues se espera que protejan la inversión inicial tan fuerte. En cambio, las aplicaciones basadas en PC son volátiles por naturaleza; de hecho la tecnología de PCs es la más volátil del mercado.

Lo que sí es un hecho es que en el futuro el proceso distribuido hará cada vez mayor uso de PCs y en general de "Workstations" inteligentes para hacer parte del trabajo, pero manteniendo, por ejemplo, la base de datos central en una "Mainframe".

En resumen se puede asegurar que, lejos de desaparecer o hacerse menos importante, tanto la tecnología de "Mainframes" como su sistemas operativos tomarán un auge importante durante los próximos años. Aún considerando las tecnologías alternas y tendencias que se vienen dando, se ve que siempre habrá procesos que por sus características y exigencias propias necesitarán de una "Mainframe" y un buen sistema operativo, como lo es el MVS.

BIBLIOGRAFIA

1.- Deitel, Harvey M.
Introducción a los Sistemas Operativos
Addison-Wesley Iberaomericana
México D.F.,1987

2.- Peterson, James L.
Operating System Concepts
Addison-Wseley Publishing Company
Austin Texas, 1985

3.- Davis, William S.
Sistemas Operativos de la Computación
Representaciones y Servicios de Ingeniería
México D.F.,1985

4.- Katzan, Harry Jr.
Operating Systems
Van Nostrand Reinhold Company
New York, 1973

Publicaciones Internas IBM:

1.- IBM System/370 Extended Arquitecture
Principles of Operation

2.- IBM System/390 Enterprise System Arquitecture
Principles of Operation

3.- MVS/XA Debbugging Handbook Volume 1-6 (Data Areas)

4.- MVS/XA Diagnostic Techniques

5.- MVS/XA Initialization and Tunning

6.- MVS/XA System Logic Library

7.- MVS/XA Overview

8.- IBM Systems Journal, Vol. 28 No.1 1989
"Enterprise Systems"