UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DIVISION DE ESTUDIOS DE POSTGRADO

SECCION DE ELECTRONICA

SISTEMAS COMPUTACIONALES MULTIPROGRAMABLES Y ALTAMENTE CONFIABLES: UNA REALIZACION CON EQUIPO CASI CONVENCIONAL.

TESIS QUE PRESENTA:

JUAN LUIS NEUMAN VELEZ

PARA OBTENER EL GRADO DE

MAESTRO EN INGENIERIA ELECTRONICA

FACULTAD DE INGENIERIA
DIVISION DE ESTUDIOS DE POSGRADO
SECCION DE ELECTRONICA

SISTEMAS COMPUTACIONALES MULTIPROGRAMABLES Y ALTAMENTE CONFIABLES:

UNA REALIZACION CON EQUIPO CASI CONVENCIONAL

TESIS QUE PRESENTA:

JUAN LUIS NEUMAN VELEZ

PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERIA ELECTRONICA

CREDITOS ASIGNADOS A LA TESIS 12

JURADO:

M. EN C. ALEJANDRO GUARDA AURAS

DR. ROBERTO CAÑALES RUIZ

M. EN C. LUIS HERNANDEZ O.

M. EN I. ROBERTO DAZA-GÓMEZ TORRES

M. EN I. CAUPOLICÁN MUÑOZ GAMBOA

M. EN C. ALEJANDRO GUARDA AURAS
JEFE DE LA SECCIÓN.

DR. UBALDO BONILLA DOMÍNGUEZ
SECRETARIO ACADÉMICO.

Esta tesis es la continuación de una serie de proyectos realizados en el INSTITUTO DE INGENIERIA de la UNAM bajo el patrocinio de la COMISION FEDERAL DE ELECTRICIDAD, GERENCIA GRAL. DE PLANEACION Y PROGRAMA en los años 1975 - 1977.

Por lo tanto, el número de personas que han colaborado directa o indirectamente en este trabajo, es tan grande, que sería difícil mencionarlos sin omisión. No obstante, es necesario destacar los nombres del Dr Adolfo Guzmán Arenas como inspirador y fecundo colaborador en las etapas más tempranas de la empresa, y el Dr Roberto Canales Ruiz quien, además de haber participado activamente en toda su extensión, tuvo la amabilidad de guiar esta tesis. A todos ellos, gracias.

1. INTRODUCCION

Las computadoras digitales actuales, y en particular las microcomputadoras de aparición reciente, son mucho más confiables que las de 10 años atrás. Se descomponen con menos frecuencia y su reparación, en caso de falla, toma menos tiempo.

Sin embargo, estos modelos nuevos también se llegan a descomponer, y el tiempo que requiere un técnico, por rápido que sea, para reparar la falla más insignificante, puede ser excesivo para controlar correctamente un proceso en tiempo real.

Se dice que una computadora opera en tiempo real cuando está programada para tomar decisiones en función del tiempo y ejecutar acciones en instantes precisos con tolerancias esperadas del orden de fracciones de segundo.

Afortunadamente existen técnicas para no interrumpir un proceso mientras se repara una falla en la computadora que lo inició.

Se puede programar una computadora de carácter general para que, con ayuda de dispositivos especiales, detecte las fallas de sus propios componentes y los sustituya automáticamente por otros de repuesto, o simplemente trabaje sin ellos. Cuando una máquina cuenta con elementos de repuesto, se dice que es redundante, y cuando opera en ausencia de alguno de sus componentes, entonces se dice que lo hace en forma degradada.

Es importante hacer notar, sin embargo, que estas opciones elevan el costo de un sistema y que, por lo tanto, no son de uso generalizado; solo se emplean en computadoras que operan en tiempo real, en procesos cuya interrupción puede acarrear consecuencias graves, procesos que llamaremos críticos. La distribución de energía eléctrica y el control de tráfico aéreo [2] son ejemplos de este tipo de procesos.

También se podrían construir computadoras más confiables que las comerciales, empleando técnicas especiales en el diseño y la manufactura de sus partes [4]. Sin embargo, esta alternativa resultaría aún más cara que la que se basa en la programación especializada de aquellas que se pueden adquirir actualmente en el mercado, a las que se calificará como convencionales.

El funcionamiento defectuoso de sistemas no redundantes se detecta mediante pruebas o programas de diagnóstico que producen resultados fácilmente comparables con los que se obtendrían en caso de que dicho sistema se encontrara en perfecto estado. El autodiagnóstico es una práctica generalizada pero insuficiente en algunos casos, en especial para los sistemas que controlan procesos críticos, donde es indispensable contar con una respuesta válida en un tiempo relativamente corto, que se especifica al diseñarlos. En los sistemas redundantes, además del autodiagnóstico, se emplean esquemas de vigilancia mutua entre los componentes duplicados (generalmente entre procesadores). Las dificultades que surgen al duplicar los componentes de una com-

putadora para aumentar su confiabilidad, dependen del papel o función que cada uno de estos componentes desempeñan dentro del sistema; mientras más complejos sean desde el punto de vista funcional, más difícil es sustituir los cuando fallan. Destacan en ese plano los elementos que se encargan de coordinar al resto, es decir, los procesadores junto con los sistemas operativos que les permiten realizar dichas tareas.

El equipo periférico, y en particular los dispositivos de entrada y de salida, suele duplicarse aun en instalaciones convencionales, que no se consideran altamente confiables. El componente más vulnerable (el que se descompone con más frecuencia por sus partes mecánicas) es el más fácil de sustituir por un repuesto cuando sufre una avería. Las lectoras de tarjetas perforadas, las impresoras de línea, etc, se encuentran dentro de la categoría de dispositivos cuya pérdida temporal no se considera catastrófica, aun cuando no se cuenta con repuestos, dado que la información que manejan puede ser recanalizada a otros (discos y cintas magnéticas, por ejemplo) mientras se reestablece el orden. En el otro extremo se encuentran aquellos que son imprescindibles en todo momento ya que sirven como base al sistema operativo (el procesador y reloj de tiempo real, como ejemplos).

Los sistemas que cuentan con varios procesadores se pueden clasificar en dos categorías: los sistemas multiprocesadores jerárquicos, donde uno de los procesadores administra todos los recursos, incluyendo los otros procesadores, y los heterárquicos, donde todos los procesadores colaboran en la administración del equipo periférico que no cuenta con poder de decisión.

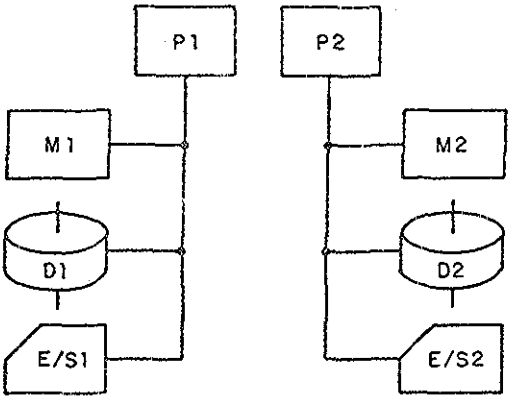
2. SISTEMAS CON DOS PROCESADORES (PROCESADORES DUALES)

La forma más sencilla de introducir redundancia en los sistemas computacionales para aumentar su confiabilidad, consiste en duplicarlos por completo (fig 1). Estos son sistemas formados por dos (o más) computadoras idénticas ejecutando el mismo programa, en donde los resultados se pueden validar por consenso (o por votación mayoritaria cuando cuentan con más de dos computadoras). Pueden seguir entregando resultados útiles a pesar de una descompostura en alguno de sus componentes, pero son muy ineficientes [3].

Los mismos recursos se aprovechan mejor (fig 2) si se toma en cuenta que:

a) No todos los componentes de un sistema tienen la misma probabilidad de fallar; por tanto, no siempre es necesario multiplicar el número de elementos de cada tipo por el mismo factor.

b) Varios sistemas independientes son inherentemente más ineficientes y vulnerables que el mismo conjunto de componentes interconectado, por ejemplo, mediante una matriz de conmutación que permita reconfigurar



P1, P2 Procesadores
D1, D2 Discos
M1, M2 Memorias
E/S1, E/S2 Entrada/salida en general

Fig 1. Sistema redundante formado por computadoras independientes

el sistema de acuerdo con las necesidades y eventualidades que puedan surgir en cualquier momento.

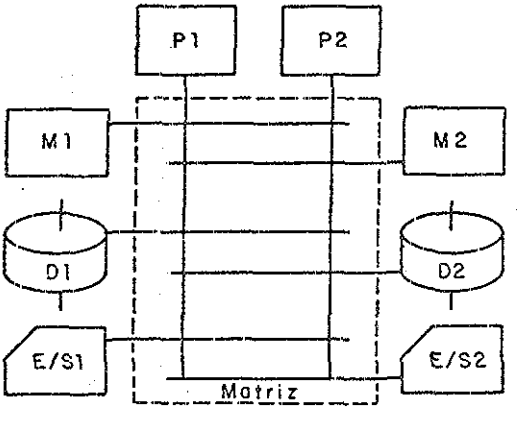


Fig 2. Sistema redundante formado por los mismos componentes de la fig 1, más una matriz de conmutación

La fig. 2 muestra un sistema redundante con los mismos componentes que los de la fig. 1 más una matriz de conmutación que permite el intercambio de periféricos entre procesadores. Este sistema, a pesar de ser más costoso (a causa de la matriz), es más eficiente y puede operar con más elementos descompuestos que el primero, por ejemplo, opera aun en caso de fallar P2, M1, D1 y E/S2.

Este sistema redundante (fig 2) permite, además del autodiagnóstico, la vigilancia mutua entre procesadores, el intercambio automático de dispositivos en caso de falla (y con esto la recuperación automática del sistema), la reconfiguración automática de los subsistemas, y el multiproceso (diferentes programas en los procesadores) sin reducir gravemente el tiempo de recuperación en caso de falla [3]. Sin embargo, requiere de un sistema operativo más complejo que el del sistema con varias computadoras independientes.

Además, aun cuando este tipo de sistemas permite el multiproceso, es importante tener presente que en este caso el objetivo principal es obtener un grado de confiabilidad mayor que el que proporcionan los sistemas no redundantes. Se puede lograr un equilibrio satisfactorio entre ambas funciones, pero no se puede mejorar una sin perjudicar a la otra.

Las matrices de conmutación mencionadas no son componentes convencionales, aunque la mayoría de las empresas fabricantes de equipo de cómputo ofrecen alguna clase de conmutador, ya sea de tipo universal o bien integrado a los diferentes dispositivos (discos magnéticos y memorias con varias vías más de acceso, por ejemplo).

2.1 *Sistemas operativos para procesadores duales.*

En adelante se entenderá que un programa pertenece al "sistema operativo" si forma parte de aquellos programas que sirven para administrar los recursos

de la computadora (equipo, tiempo y ejecución de otros programas) [5].

Se denominará "programa de aplicación o de usuario", o simplemente "tarea", a todo aquel que resuelve problemas que no están relacionados con la administración de los recursos de la computadora, que con base en datos tomados del mundo exterior, entregan resultados significativos también para el mundo exterior.

Por último, se llamarán "programas del sistema" todos aquellos que puedan considerarse como recursos del sistema computacional. Son programas que o bien forman parte del sistema operativo, o bien que, sin tener funciones administrativas, facilitan la operación de la computadora y la elaboración de los programas de aplicación (compiladores, ensambladores, cargadores, etc).

La responsabilidad de superar las descomposturas en el equipo de una computadora redundante recae directamente sobre el sistema operativo. Esta actividad debería ser transparente para los usuarios, y su realización es más compleja que la de los sistemas operativos para computadoras convencionales.

Las computadoras redundantes cuentan con dispositivos especiales para detectar fallas y para intercambiar dispositivos descompuestos por sus repuestos pero normalmente es el sistema operativo y no los dispositivos mismos quien ordena el intercambio o "reconfiguración", quien toma las medidas necesarias para operar en forma degradada cuando no cuenta con un repuesto, quien notifica cualquiera de estas eventualidades al operador, y quien conoce y controla el estado general del sistema.

2.2 *Elaboración de respaldos*

Para que un procesador de repuesto se pueda hacer cargo de un proceso aban-

donado por otro a causa de una descompostura, es necesario que cuente con los programas que realizan el proceso y con información relativa al estado de avance de este. Los programas mencionados pueden ser depositados manualmente en un medio de almacenamiento masivo, accesible al procesador de repuesto, desde el principio de la operación. Pero los registros del estado de avance del proceso, a los que llamaremos "respaldo del proceso" o simplemente "respaldos", tienen que ser actualizados periódicamente y en forma automática.

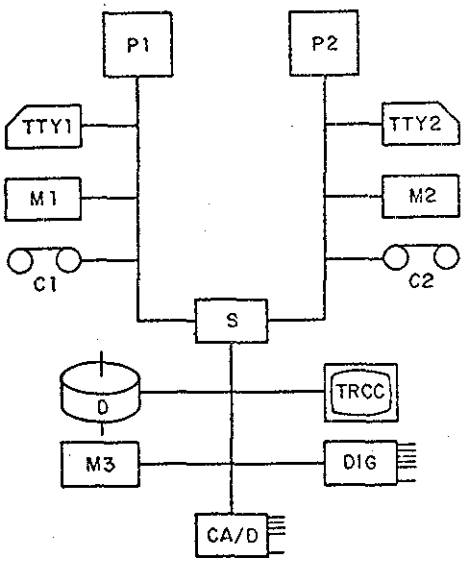
Existen técnicas depuradas [1, 7, 8] para recuperar, después de una falla, la información contenida en una base de datos. Pero en el caso de los procesos en tiempo real, donde la cantidad de información involucrada es menor y la velocidad de recuperación mayor, la forma más directa de respaldar la operación consiste en generar y mantener actualizada una copia de todas las variables del sistema, que pueda ser usada por el procesador de repuesto como punto de partida para continuar el proceso interrumpido.

3. DESARROLLO DE UN SISTEMA EXPERIMENTAL

Con objeto de experimentar en el campo de los sistemas altamente confiables, el Instituto de Ingeniería adquirió un equipo con dos procesadores y un sistema operativo básico que permite ensayar varias técnicas de recuperación automática en caso de falla.

Específicamente, se adquirieron dos procesadores con objeto de ensayar técnicas de vigilancia mutua entre ellos, generación de respaldos en disco magnético (al cual ambos procesadores tienen acceso), y recuperación del proceso, cuando uno de los procesadores falle. Aunque los dos son idénticos, de ahora en adelante se denominará titular al que en un momento dado es el responsable directo del proceso en tiempo real, y se llamará suplente al que vigila al titular y es capaz de suplirlo en caso de que falle.

Cuando el suplente sustituye al titular, asumiendo todas sus funciones, se dice que toma posesión del proceso y se convierte en titular.



P1 = P2 PD11/40 TTY1= TTY2 Teletipo 30 cps M1 = M2 Memoria 64 kb
C1 = C2 Doble cassette S Conmutador D Disco 2 5 Mb
TRCC Tubo rayos catódicos a color M3 Memoria 64 kb DIG E/S digital
CA/D Convertidor análogo-digital

Fig 3 Configuración del sistema experimental

La clave de este tipo de operación es el conmutador entre procesadores, S (fig 3), el cual permite que ambos tengan acceso a los respaldos en disco, y cuenta con un dispositivo especial, denominado "WAICHDOG IIMER" por el fabricante, que facilita la vigilancia mutua entre procesadores y la toma de posesión, por parte del suplente, cuando el titular falla.

Durante la inicialización del sistema se le asigna el control del conmutador y del equipo asociado (D, M3, CA/D, IRCC y DIG en la figura 3) al procesador titular. Posteriormente se ejecuta un programa en el suplente para solicitar periódicamente al WAICHDOG IIMER que le conceda el control del conmu-

tador. Cada vez que ocurre una petición de estas, el WAICHDOG TIMER se lo informa automáticamente al titular, y este debe negarse a entregarlo. Si este se rehúsa a ceder el conmutador, entonces el WAICHDOG TIMER le avisa al suplente que no se lo puede conceder. Pero si el titular está descompuesto y falla en responder, entonces le transfiere automáticamente el control al suplente, y este debe reaccionar convirtiéndose en nuevo titular y continuar con el proceso en tiempo real (los detalles del conmutador pueden consultarse en el manual correspondiente [10]).

Con base en el equipo anterior se desarrolló un sistema operativo para computadoras duales (SO1) con las siguientes especificaciones externas.

3.1 *Especificaciones externas del sistema SO1*

Descripción

Monitor de tiempo real multitarea para computadoras redundantes.

Equipo que maneja

Computadora PDP 11/40 dual (dos procesadores y 128 kilobytes de memoria) con canal compartido mediante un conmutador programable, memoria secundaria en disco, convertidores análogo-digital y digital-análogo, vía de acceso digital, y unidades de cassette.

Número máximo de tareas

Ilimitado*

Operación de tareas

1. Por evento, insertadas como subrutinas para atender interrupciones.
2. Por hora de entrada, compartiendo los recursos con otras tareas según

---

* Esta opción aún no se ha implantado.

la prioridad especificada.

Prioridad para tareas que se activan por evento

Ocho prioridades diferentes (7-0) según el esquema de interrupciones de la PDP 11.

Prioridad para tareas que se activan por hora

Dieciséis prioridades diferentes (1-16), todas inferiores a las de los programas que corren por interrupción. Los programas que se ejecutan con prioridad 1 cuentan por lo menos con la mitad del tiempo útil del procesador; los de prioridad 2 cuentan cuando menos con la cuarta parte, y así sucesivamente

$$t_p = \frac{1}{2^p} \; t_u$$

donde:

t_p tiempo de procesador para la prioridad p.

t_u tiempo útil del procesador

Puesta de operación de las tareas

Las tareas se ensamblan con ayuda de RT11 y se cargan junto con SO1 mediante el cargador ligador del primero (RT11 es el sistema operativo del fabricante).

Lenguajes disponibles

Macro-ensamblador

Opciones que simplifican la programación.

1. Archivos compatibles con los de RT11.
2. Rutinas de servicio (toda operación de entrada o salida se solicita a SO1 mediante una seudoinstrucción proporcionada por el sistema).

Opciones de seguridad.

1. Procesador suplente que reinicia automáticamente el proceso para garantizar la continuidad de operación en caso de falla del procesador titular.
2. Protección mediante circuitos especiales contra operaciones de entrada o salida no autorizadas*.
3. Protección de memoria contra referencias inválidas*.
4. Protección contra ejecución de instrucciones inválidas.
5. Restablecimiento automático por fallas en el suministro de energía eléctrica.
6. Protección contra monopolios del procesador vía partición del tiempo según prioridades, para las tareas que entran por hora.
7. Verificación periódica (cada 16 mseg típicamente) del estado de las tareas que entran por hora, para evitar desbordes de tiempo inadvertidos.

3.2 *Estructura del sistema SO1*

Las funciones principales del SO1 son distribuir los recursos del sistema entre las diferentes tareas, y evitar el colapso total del sistema debido a mal funcionamiento de ellas o del equipo. La protección contra fallas de programación en las tareas se logra compartiendo el tiempo de computadora (time slice) entre ellas según un esquema de prioridades que asegure su entrada, aunque uno o varias estén perdidos en un círculo vicioso. La integridad del equipo periférico y de memoria se asegura con ayuda de mecanismos que detecten intentos de acceso no autorizados y que notifican al SO1 para que tome las medidas correctivas necesarias.

La redundancia con que se cuenta no es óptima; únicamente persigue la posi-

---

* Esta opción aún no se ha implantado.

bilidad de experimentar en los campos de vigilancia mutua entre procesadores y recuperación automática en caso que falle uno de ellos, áreas consideradas como las más desafiantes.

El SO1 está formado por cinco grandes bloques (fig 4): INICIA, MIRB, LLAMA, SERVES y EMRGEN cuyas funciones son: Inicialización del sistema, administración del procesador con base en las interrupciones de un reloj de tiempo real (RIR), administración del equipo periférico mediante interrupciones programadas, supervisión del equipo periférico (E/S) y atención a emergencias, respectivamente. En particular, desde un punto de vista de alta confiabilidad, MIRB asegura la ejecución de todas las tareas, LLAMA evita el mal uso del equipo periférico y resuelve los conflictos que puedan surgir durante su empleo, y EMRGEN suspende a los programas infractores, vigila el buen funcionamiento del equipo, y si es posible, cuando no puede operar de manera correcta, renuncia ordenadamente para que el procesador suplente tome el control.

A fin de encauzar el esfuerzo hacia áreas poco estudiadas, el SO1 aprovecha muchos programas de apoyo del sistema operativo adquirido, por tanto, no es autónomo. Esto es, no cuenta con ensamblador, ligador, ni cargador propios e inclusive toma prestadas del RT11 algunas subrutinas de atención a periféricos. Sin embargo, una vez iniciada la operación en tiempo real, toma el control absoluto del sistema.

NOTA:

Los programas que se ejecutan normalmente en el procesador suplente no son parte de SO1. Lo único que se requiere del procesador de repuesto es que ejecute periódicamente una subrutina (además de las otras tareas que se le asignen) que vigile la operación del titular y tome posesión en caso necesario (ver el ejemplo del apéndice D).

Dicha subrutina puede operar por interrupción y requiere de muy pocas instrucciones, por lo cual interfiere poco con el buen funcionamiento de los otros programas mientras no falle el titular.

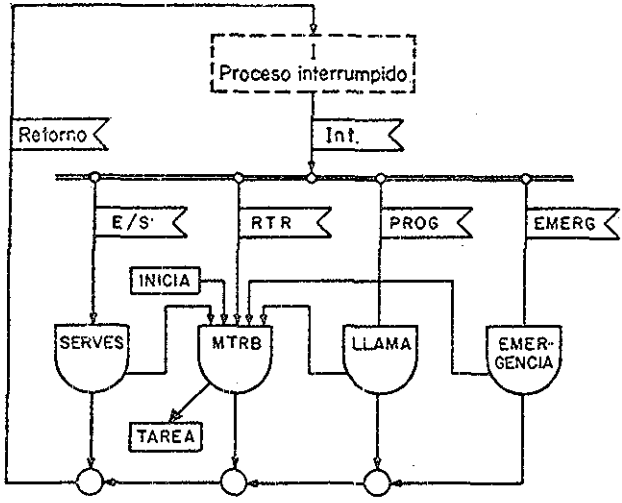


Fig 4 Diagrama global del sistema operativo SO1

3.3 *Teoría de operación de SO1.*

SO1 tiene tres modos de operación a saber:

Un modo de inicialización que sirve exclusivamente para poner el sistema en condiciones normales de operación (después de encender la computadora, por ejemplo).

Un modo normal de operación durante el cual se ejecutan las tareas de aplicación o control asignadas al sistema y las rutinas de diagnóstico y generación de respaldos.

Un modo de alerta durante el cual se analizan las situaciones de emergencia y se toman medidas para corregirlas.

Desde un punto de vista teórico, únicamente los programas asociados a un solo modo necesitan residir en la memoria primaria del procesador, en un momento dado. Si se intercambian los programas (swaping) entre memorias primaria y secundaria cada vez que cambia el modo de operación, entonces se disminuye el tamaño de la primaria requerida, pero se afecta la complejidad, la velocidad y la confiabilidad del sistema. Por esta razón, y considerando que se trata de un sistema experimental en donde el tamaño de la memoria primaria no es una limitación, SO1 no intercambia ninguno de los programas de sistema ni tampoco los programas de aplicación.

3.4 *Inicialización de SO1*

La inicialización de SO1 se ejecuta por medio de la rutina INICIA (que se describe más adelante) con base en la información que recibe de:

-los programas fuente de aplicación,
-el ensamblador empleado para obtener el código de máquina correspondiente, y
-el ligador que conjuga dichos programas en un solo módulo de carga.

La información proveniente de los programas fuente, junto con la proporcionada por el ensamblador, se concentra en una tabla o descriptor de tarea llamado IAi.DES en donde el número natural i identifica a cada programa de aplicación.

Esta tabla incluye las constantes de operación que especifica el programador (la hora de entrada u hora a la que se desea ejecutar por primera vez el programa, la periodicidad con que se desee ejecutar, su prioridad relativa a otros programas, etc), y las que resultan del proceso de ensamblado (direcciones relativas de subrutinas y segmento de datos).

La información proveniente del ligador se localiza en una tabla única (DIRE. ORI) que sirve como directorio para localizar a los descriptores de cada una de las tareas.

De esta forma INICIA puede establecer las condiciones iniciales de operación en una tabla de variables de operación, llamada vector de transacciones (VECI.TRANS), e inicializar los periféricos no manejados por el sistema operativo RT11. Realizadas estas funciones, el programa INICIA entrega el control del sistema al programa MTRB que se hace cargo de la operación normal.

Se ha previsto, mas no implantado ya que no es estrictamente necesario, la inclusión de un programa que interprete comandos introducidos a través del teletipo para establecer o modificar tanto las condiciones iniciales como las normales de operación.

3.5 *Operación de SO1 en condiciones normales.*

Iniciada la operación, y mientras no surja una condición que la comprometa (una emergencia), la responsabilidad de facilitar los recursos del sistema a las tareas de aplicación recae sobre los programas MIRB, LLAMA y SERVES que se describen en detalle más adelante. Esta función se distribuye en la siguiente forma; MIRB distribuye el tiempo de procesador entre tareas de aplicación con base en el estado de cada tarea, representado por el vector de transacciones mencionado (hora de entrada, prioridad, etc), y en un contador de tiempo llamado reloj del sistema (REI.SYS) que el mismo MIRB actualiza cada sesentavo de segundo. Esto permite que las tareas se ejecuten a tiempo con una resolución de un sesentavo de segundo y elimina la posibilidad de que una tarea monopolice el procesador durante un lapso de tiempo mayor que el permitido.

Cuando una tarea necesita de algún servicio o de algún recurso adicional al procesador y a la memoria primaria que tiene asignada, lo solicita a SOI mediante una interrupción programada que es atendida por IIAMA. Este conjunto de programas verifican la validez de la solicitud y le dan curso si es válida y factible, o la rechazan en caso contrario.

Los servicios que presta IIAMA son: manejo de archivos, operaciones de entrada o salida, y algunos misceláneos como son la comunicación entre tareas y MIRB (en el apéndice B se incluye una lista completa y ejemplos de empleo).

La función de IIAMA es la de un intermediario entre las tareas de aplicación y el resto del sistema (programas y equipo). Sus objetivos son facilitar la elaboración de programas de aplicación y aumentar la confiabilidad del sistema evitando que las tareas hagan mal uso del equipo periférico y permitiendo la reconfiguración del sistema cuando falla alguno de sus elementos.

La programación se simplifica porque el programador no tiene que enterarse de los detalles que imponen las limitaciones físicas de los dispositivos de entrada y salida. Estos se le presentan en forma idealizada y los maneja mediante comandos o seudoinstrucciones fáciles de aprender.

Se impide el mal uso de los recursos del sistema porque IIAMA verifica la validez de las solicitudes de servicio y rechaza las inadmisibles. Evita que una tarea destruya información que no le pertenece o disponga de un dispositivo que no le ha sido asignado.

Por último, dado que las tareas no tienen contacto directo con el equipo periférico, el sistema puede ser reconfigurado en caso necesario. Se puede sustituir, por ejemplo, un transporte de disco descompuesto por otro medio masivo de acceso directo, como un tambor, sin que las tareas se vean afectadas.

Una petición a ILAMA puede ser rechazada por una de dos razones, a saber:

a) El comando no es válido (no existe el recurso solicitado, o no está asignado a la tarea que lo intenta utilizar, no está bien especificada la operación deseada, etc). En este caso la tarea puede ser suspendida temporal o permanentemente, dependiendo de la gravedad del error. Después se notifica la causa del rechazo en una palabra de estado, y se le entrega el control a MIRB para que reasigne el procesador en caso necesario.

b) El comando es válido pero el sistema no está en condiciones de ejecutarlo (el recurso existe pero no se encuentra disponible, el sistema está saturado de trabajo, etc). En este caso también se notifica la causa del rechazo, y se le entrega el control a MIRB para que decida a quién le corresponde seguir usando el procesador.

Cuando el comando implica una operación de entrada o de salida, entonces ILAMA solicita ayuda de SERVES para ejecutarlo, y después entrega el control a MIRB.

SERVES es el conjunto de programas (DEVICE HANDLERS) que se encargan de realizar las operaciones de transferencia de datos entre el equipo periférico y la memoria principal o del procesador. En general, las operaciones de entrada y de salida se realizan en tres etapas que son: inicialización, transferencia propiamente dicha y terminación.

La inicialización de una transferencia comienza cuando ILAMA transfiere el control a la subrutina correspondiente de SERVES.

Si dicha subrutina está en condiciones de realizar la transferencia toma nota de los parámetros necesarios (identificación del dispositivo, localización de los datos a transferir, etc), y devuelve el control a ILAMA después de iniciarla o de anotarla en una cola de espera, según las características del dispositivo involucrado y las condiciones o estado que impe-

ran en ese momento.

En la mayoría de los casos, los datos son transferidos por otra subrutina de SERVES como respuesta a las interrupciones de los dispositivos que avisan cuando están listos para aceptar o recibir información, o bien por circuitos electrónicos (acceso directo a memoria) que también interrumpen al procesador cuando terminan de ejecutar la transferencia.

La subrutina de atención a las interrupciones del dispositivo, al enterarse de que la transferencia está completa, entrega nuevamente el control a LLAMA para que este lo notifique a la tarea correspondiente, en caso de ser necesario.

Si la subrutina no puede aceptar el comando (está llena la cola de espera, por ejemplo) entonces también devuelve el control a LLAMA para que este lo rechace.

Con lo anterior se puede observar que SO1 es un sistema operativo compuesto casi exclusivamente por subrutinas que atienden interrupciones (interrupt driven). Una vez iniciada la ejecución de la primera tarea se considera que opera en el modo normal y la única forma que existe para que el procesador cambie de manos es mediante alguna interrupción. Las tareas son interrumpidas por el reloj de tiempo real cada sesentavo de segundo, y la interrupción es atendida por MTRB para cederle el procesador a otra tarea. Las tareas de aplicación también se pueden interrumpir por su propia voluntad, y son atendidas por LLAMA.

Otra fuente de interrupciones, atendida por SERVES, es el equipo periférico que solicita datos o avisa que ha terminado una transferencia de entrada o de salida.

Por último existe una fuente adicional de interrupciones constituida por dispositivos que vigilan la integridad del sistema e interrumpen al procesador cada vez que se encuentra amenazada. Lo interrumpen, por ejemplo, cuando se trata de ejecutar una instrucción inválida, o cuando se genera una dirección inválida o cuando alguna tarea trata de usar alguna región de memoria que no le ha sido asignada, o cuando falla la energía eléctrica.

3.6 *Condiciones de emergencia*

Se considera como emergencia a toda aquella condición que compromete el buen funcionamiento del sistema que puede provocar el colapso total del proceso.

Todas las emergencias de este tipo son tratadas por un conjunto de programas llamado EMRGEN, que responde a las interrupciones generadas por dispositivos especiales que las detectan. Su origen puede ser interno, como cuando se trata de ejecutar una instrucción inválida, o bien puede ser externo, como cuando falla el suministro de energía eléctrica.

Entre las emergencias de origen interno se distinguen 3 tipos diferentes: las provocadas por un mal funcionamiento del equipo, las provocadas por algún programa del sistema operativo, y las que provienen de errores en los programas de aplicación. En general, las dos primeras causas se consideran fatales y procede la renuncia ordenada para que el sistema suplente continúe la operación e indique la causa de la reconfiguración; la última provoca la suspensión de la tarea infractora y la generación de los mensajes de advertencia al operador.

4. DESCRIPCION DE SO1

Para facilitar la comprensión del funcionamiento de cada una de las rutinas de SO1, conviene describir los datos que maneja por separado cada una de ellas y los datos que son compartidos entre varias.

En general, las variables escalares se explican por sí mismas. No así los vectores y otras estructuras más complejas, como las listas.

Para evitar la confusión a la que conduce un alejamiento repetido del tema principal del texto, se incluye a continuación un subcapítulo que describe las estructuras de datos más complejas que se emplean y en las figuras 5 y 6 se ilustran las relaciones que existen entre ellas antes y después de ejecutar la rutina INICIA.

4.1 Estructuras de Datos

IAi.DES

IAi(rea).DES(criptor) es un arreglo que debe incluirse en el cuerpo de cada

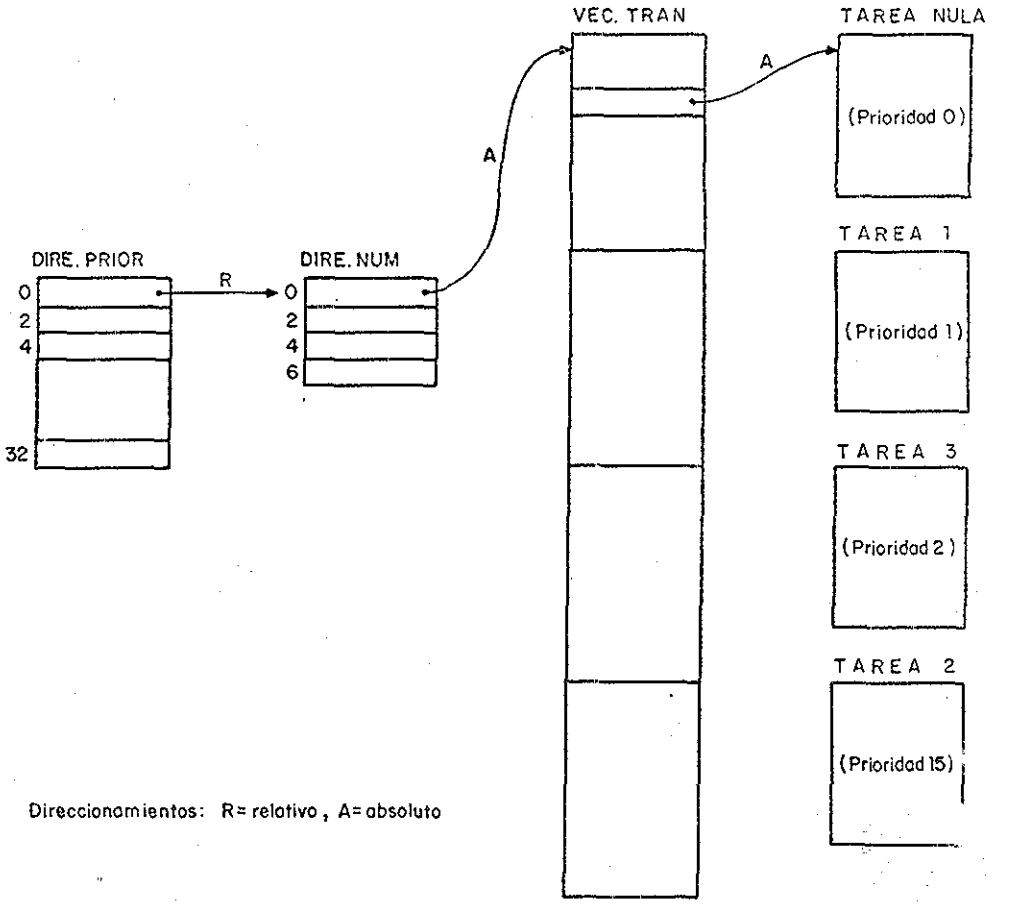


Fig 5 Estado de algunos arreglos de datos antes de ser ejecutado el programa INICIA

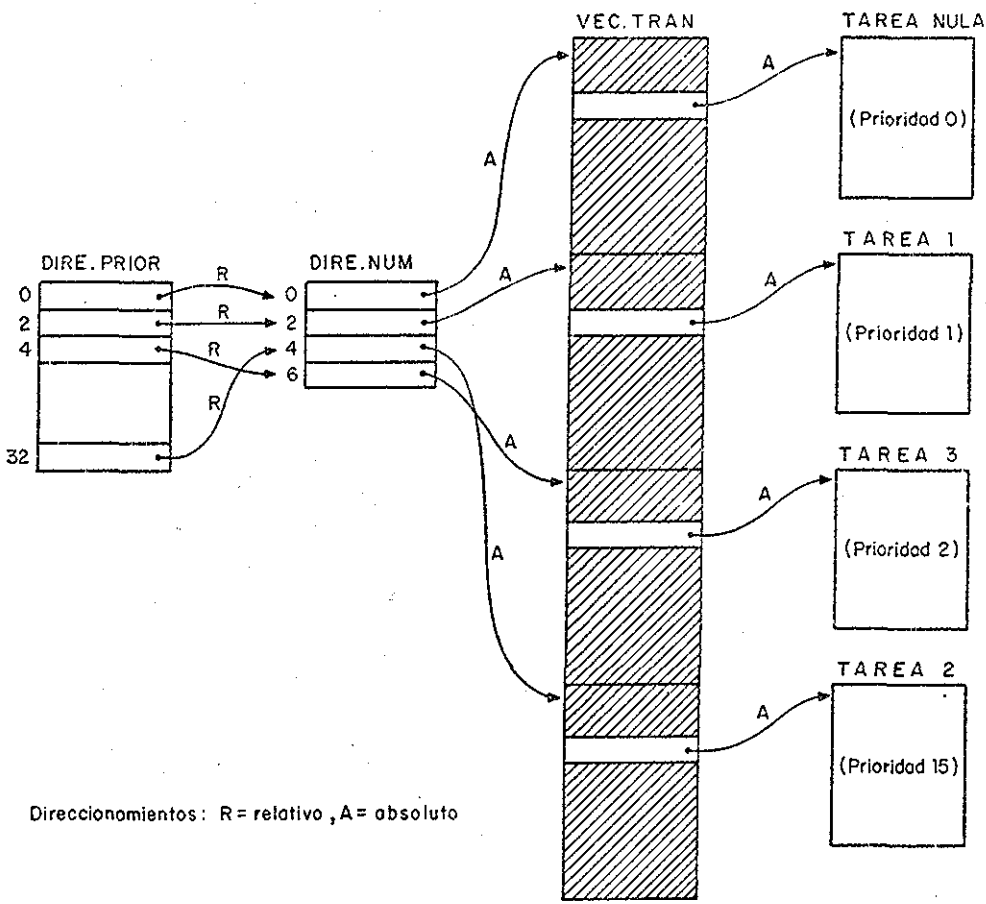


Fig 6. Estado de algunos arreglos de datos después de ser ejecutado el programa INICIA

programa de aplicación o tarea externa, para proporcionar al programa INICIA la información necesaria para iniciar la operación normal. Debe existir un arreglo TAi.DES por cada tarea. El número i es diferente para cada tarea y menor o igual al máximo número de tareas (MAX.PROGS) especificadas durante el ensamblado de SO1.

La información que debe incluir es la siguiente:

TAi.DES:	Nombre (parte menos significativa en R50)
TAi.DES + 2:	Nombre (parte más significativa en R50)
TAi.DES + 4:	Hora de entrada (parte menos significativa en 1/60 seg)
TAi.DES + 6:	Hora de entrada (parte más significativa en 1/60 seg)
TAi.DES + 8.:	Tiempo de interarribo (parte menos significativa en 1/60 seg)
TAi.DES + 10.:	Tiempo de interarribo (parte más significativa en 1/60 seg)
TAi.DES + 12.:	Prioridad numérica de 1-16. Por el momento, solo un programa por prioridad.
TAi.DES + 14.:	Dirección absoluta de entrada al programa
TAi.DES + 16.:	Dirección absoluta de entrada en caso de desborde
IAi.DES + 18.:	Dirección absoluta de la pila (STACK) asignada

En donde la hora de entrada es el tiempo en sesentavos de segundo que debe transcurrir desde la inicialización del sistema, antes de la primera ejecución de la tarea; el tiempo de interarribo es el período con que debe ejecutarse después de la primera entrada. El ligador determina las direcciones absolutas de entrada normal, por desborde y la de la pila. Sin embargo, es necesario solicitar dicho servicio poniendo en el arreglo el nombre o etiqueta correspondiente (ver los ejemplos del apéndice C).

Este arreglo sólo sirve para transferir información entre las tareas y SO1 durante la inicialización; después de esta pueden usarse las posiciones de memoria que ocupa.

DIRE.ORI

DIRE (ctorio).ORI (genes) es un arreglo que se llena, durante el ligado de las tareas y SO1, con las direcciones absolutas de los descriptores de tareas IAi.DES según el orden especificado por el número i de cada tarea. Su función es proporcionar a INICIA las direcciones de los descriptores para que pueda transferir los datos contenidos en aquellos y concentrarlos en un nuevo arreglo VEC.IRAN que se describe más adelante.

LOCALIDAD:	CONTENIDO = DIRECCION ABSOLUTA DE:
DIRE.ORI	descriptor de la TAREA NULA
DIRE.ORI + 2	TA1.DES
DIRE.ORI + 4	TA2.DES

etc, hasta completar un número de entradas o posiciones igual a MAX.PROGS + 1.

La TAREA NULA es una tarea interna, que consume el tiempo de procesador que no usan las tareas externas o programas de aplicación.

DIRE.NUM.

DIRE(ctorio por).NUM(ero) es un arreglo que contiene la dirección del vector de transacciones (que se describe más adelante) para cada tarea de usuario. La primera palabra del arreglo contiene la dirección absoluta del vector de transacciones para la tarea nula (esto es, a la tarea nula se le asigna el número cero), y las siguientes palabras son inicializadas por el programa INICIA con las direcciones del vector de transacciones para los

programas uno, dos, ..etc..

El objeto de este arreglo es proporcionar a MIR, y a las subrutinas asocia das, un acceso rápido al vector de transacciones para cada uno de los usua rios..

LOCALIDAD	CONTENIDO = DIRECCION ABSOLUTA DEL
DIRE..NUM:	vector de transacciones para la tarea nula
DIRE..NUM + 2	vector de transacciones para la tarea 1
DIRE..NUM + 4	vector de transacciones para la tarea 2
DIRE..NUM + 2N:	vector de transacciones para la tarea N

DIRE(ctorio por)..PRIOR(idad)

Es un arreglo de diecisiete palabras; una entrada por cada una de las prioridades posibles.. La primera palabra contiene simpre el número cero, esto es el número de la tarea nula multiplicado por dos.. La segunda palabra contiene el número de la tarea de prioridad uno multiplicado por dos, la tercera palabra contiene el número de la tarea de prioridad dos multiplicado por dos, y así sucesivamente..

Si no existe tarea para alguna de las prioridades, entonces el contenido es un número negativo cualquiera.. Toda esta información es depositada por el programa INICIA para facilitar la ejecución del resto de los programas, y en especial la del programa ELIJE, que selecciona una tarea dada su prio ridad..

Con el esquema descrito se puede usar solo una por prioridad..

LOCALIDAD:	CONTENIDO = 2* NUMERO DEL PROGRAMA DE:
DIRE.PRIOR:	prioridad cero (de la tarea nula)
DIRE.PRIOR + 2	prioridad uno
⋮	
DIRE.PRIOR + 32.:	prioridad dieciséis

VEC.IRAN

El VEC(tor de).IRAN(sacciones) resume el estado de las tareas que operan bajo SO1 en cualquier instante. Prácticamente no existe programa en todo el sistema que no esté relacionado con este arreglo, que no lo modifique o que no esté condicionado por él.

Se le inicializa por medio del programa INICIA con los parámetros que indican el régimen de operación de cada tarea (datos que son proporcionados por cada tarea en particular, a través de su descriptor de tarea (IAi.DES)).

Es analizado por el programa MIR, al menos cada 16 milisegundos, para vigilar el proceso de las tareas de usuario y asegurar el funcionamiento correcto de todas y cada una de ellas. Es consultado por el mismo MIR, y con la misma frecuencia, para distribuir adecuadamente los recursos del procesador central.

El vector de transacciones está formado por N elementos, donde N es el número MAX.PROGS de tareas externas (de usuario) más uno (para la tarea nula). Cada elemento del vector está formado por un número VI.LOW de palabras que sirven para especificar el régimen de operación de la tarea, para resumir su estado, y para condicionar su ejecución. Existen dos formas básicas de tener acceso a VEC.IRAN, a saber:

1. en forma directa

2. en forma indirecta, empleando DIRE.NUM y el número del programa del cual se requiere información.

Cada uno de los N elementos de este vector contiene la siguiente información.

LOCALIDAD	CONTENIDO
VEC.IRAN	NOMBRE de la tarea
VEC.IRAN + 2	dos palabras en código RADIX 50 (R50)
VEC.IRAN + 4	HORA de entrada, en sesentavos de segundo;
VEC.IRAN + 6	dos palabras
VEC.IRAN + 8.	TIEMPO de interarribo en sesentavos de segundo; dos pala-
VEC.IRAN + 10.	bras
VEC.IRAN + 12.	PRIORIDAD. La posición de un bit único en la palabra determina la prioridad si (VEC.IRAN + 12) = $\Sigma_{i=1}^{16} 2^{i-1} b_{i-1}$, entonces prioridad = i para el único $b_{i-1} \neq 0$
VEC.IRAN + 14.	DIRECCION absoluta de entrada a la tarea
VEC.IRAN + 16.	DIRECCION absoluta de entrada por desborde. Si una tarea necesita ser reiniciada antes de completarse la ejecución anterior, entonces se dice que hubo un desborde y se le activa por esta entrada.
VEC.IRAN + 18.	DIRECCION absoluta de la pila de la tarea.
VEC.IRAN + 20.	CALIFICADOR del estado de la tarea en un instante dado. Si (VEC.IRAN + 20) = $\Sigma_{i=1}^{16} 2^{i-1} b_{i-1}$, entonces las variables b_{i-1} representan el estado de la tarea de acuerdo a la convención siguiente: $b_0 = 1$ la tarea está activa $b_1 = 1$ la tarea está interrumpida

b_2 = 1 la tarea está suspendida

b_4 = 1 ha ocurrido un desborde

b_5 = 1 la tarea puede ser activada (es elegible)

b_7 = 1 es hora de que sea activada

b_{10} = 1 la tarea fue inscrita durante la inicialización (está en la mezcla).

VEC.IRAN + 28. CALIFICADOR de interrupciones

VEC.IRAN + 30. CALIFICADOR de suspensiones

VEC.IRAN + 32. a VEC.IRAN + 50. REGISTROS generales (RØ - R7 y PSW) al momento de ser suspendida o interrumpida

VEC.IRAN + 52. PC = R7 y PSW al momento de detectarse un

VEC.IRAN + 54. desborde

VEC.IRAN + 56. a VEC.IRAN + 66. AREA para argumentos de seudoinstrucciones (PROGRAMMED REQUESTS).

En las figuras 5 y 6 se muestran las interrelaciones entre los diversos arreglos descritos, antes y después de la iniciación del sistema.

DIRE.CAN

DIRE.CAN, o directorio de canales de entrada y salida, es un arreglo que tiene 15 palabras.

Cada vez que SO1 asigna un canal de entrada y salida a una tarea de aplicación, anota la identificación de la tarea (USUARIO) en la palabra de DIRE.CAN que le corresponde a ese canal según su número. Dado que solamente 15 de los 16 canales disponibles pueden ser usados por las tareas de aplicación (el décimosexto está reservado para uso exclusivo del sistema operativo SO1). el arreglo solo cuenta con 15 palabras.

Cada vez que una tarea solicita un canal de entrada y salida mediante una seudoinstrucción, el programa IIAMA consulta esta tabla en busca de un canal desocupado para asignárselo.

Y cada vez que un dispositivo de entrada o de salida interrumpe el proceso, para dar aviso de la terminación de una transferencia que involucra algún canal, IIAMA emplea esta tabla para identificar a la tarea que solicitó la transferencia y notificarle los resultados.

4.2 *Descripción de los programas*

A continuación se describen los programas más importantes del sistema y algunas de sus subrutinas. Los detalles menos relevantes pueden ser consultados en el listado general.

4.2.1 INICIA

Este programa prepara todas las tablas del sistema e inicializa variables para que pueda empezar a funcionar el programa MIR. Las variables más importantes, involucradas en el proceso, son: el contador para elegir usuarios (ELI.CONJA), el reloj del sistema (REL.SYS) y el usuario en operación (USUARIO). Las tablas que requieren inicializarse son: el vector de interrupciones, el vector de transacciones (VI.ESTADO), el directorio de usuario por número (DIRE.NUM) y el directorio de usuario por prioridad (DIRE.PRIOR).

Lo primero que hace INICIA, después de identificarse por la impresora del teletipo, es calcular las dimensiones, en palabras y en bloques de 256 palabras, que tendría un respaldo de las tareas. Posteriormente busca un archivo en disco con el nombre CHECK.NEU, y si lo encuentra, entonces lo lee y toma la información contenida como punto de partida para continuar la ope

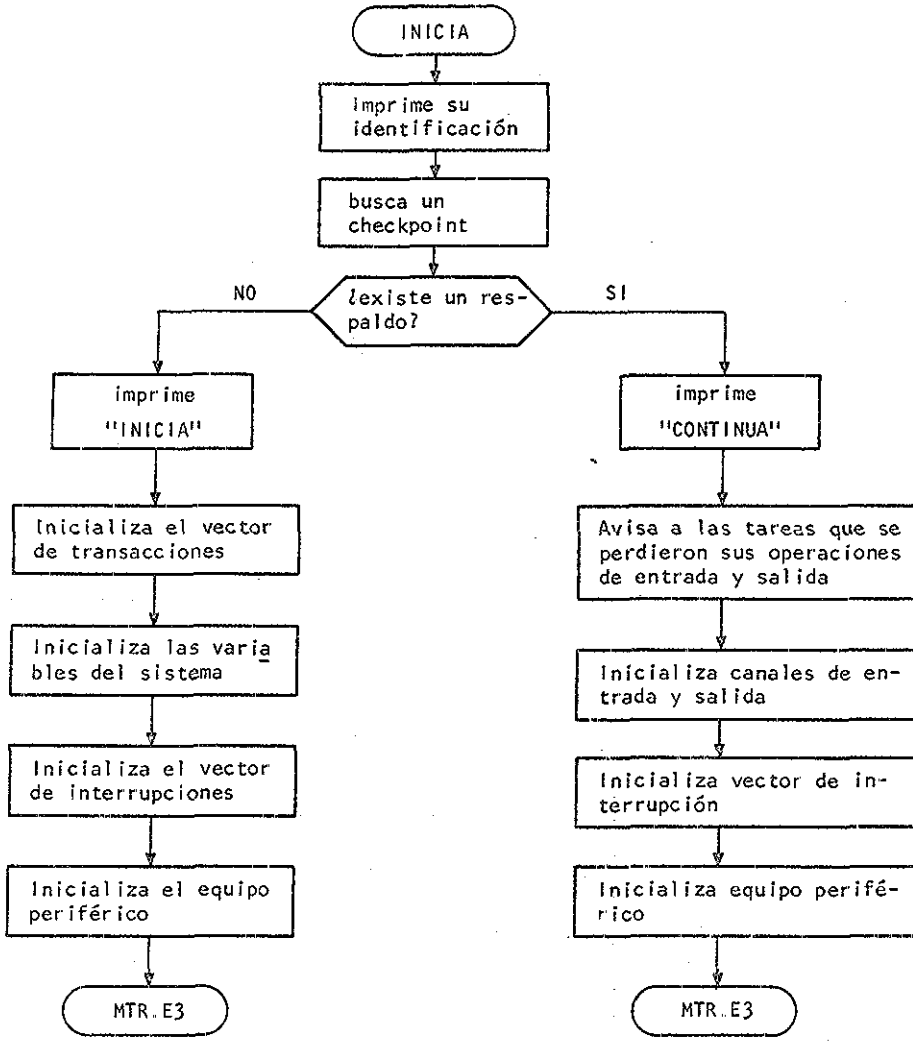


DIAGRAMA DE BLOQUES DEL PROGRAMA INICIA

ción. Si no lo encuentra inicializa las variables del sistema y de las tareas con base en la información contenida en los descriptores de tarea mencionados anteriormente. Por esta razón, es necesario que SO1 se ensamble especificando el número máximo de tareas que se van a ejecutar (MAX.PROGS), y que se incluya una entrada por programa usuario o tarea en el arreglo DIRE.ORI.

Por parte de los usuarios, es necesario que se incluya un arreglo descriptor de la tarea (TAi.DES) con los datos descritos en la sección 4.1.

espués de la inicialización, INICIA transfiere el control a MIR por la entrada especial MIR.E3.

4.2.2 MTR

Ia función principal de MTR es distribuir el tiempo de procesador disponible entre las tareas o programas de aplicación, según su prioridad. Para lograrlo, se activa sesenta veces por segundo. Cada vez que entra en acción verifica que las tareas también han tenido oportunidad de trabajar y revisa el estado de cada una de ellas para elegir, con ayuda de una subrutina (EII.US), al usuario que utilizará el procesador durante el siguiente sesentavo de segundo (ver el diagrama de bloques de MIR y el listado general). Otras de sus funciones, involucradas en este proceso, son las de guardar el estado del sistema cuando una tarea sale de operación, restablecerlo cuando vuelve a entrar en acción y, en general, entregarlo a las tareas tal y como lo requieren según estén empezando a operar, a medio proceso, o en desborde.

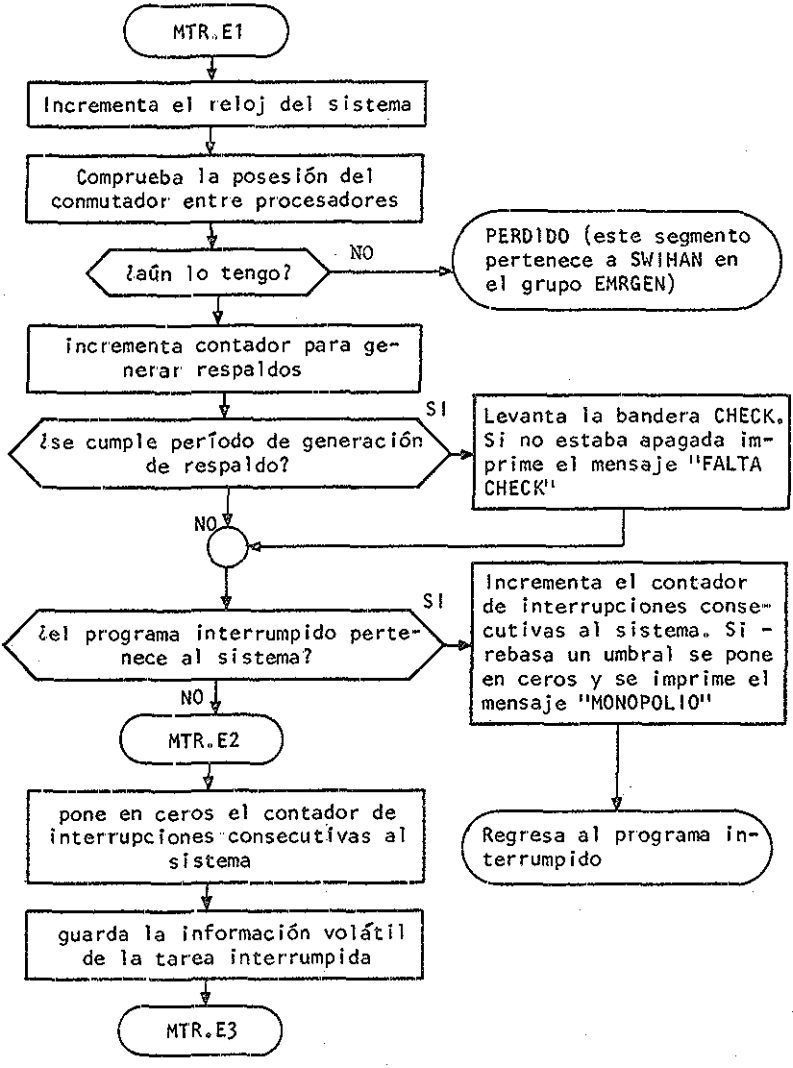


DIAGRAMA DE BLOQUES DE MTR

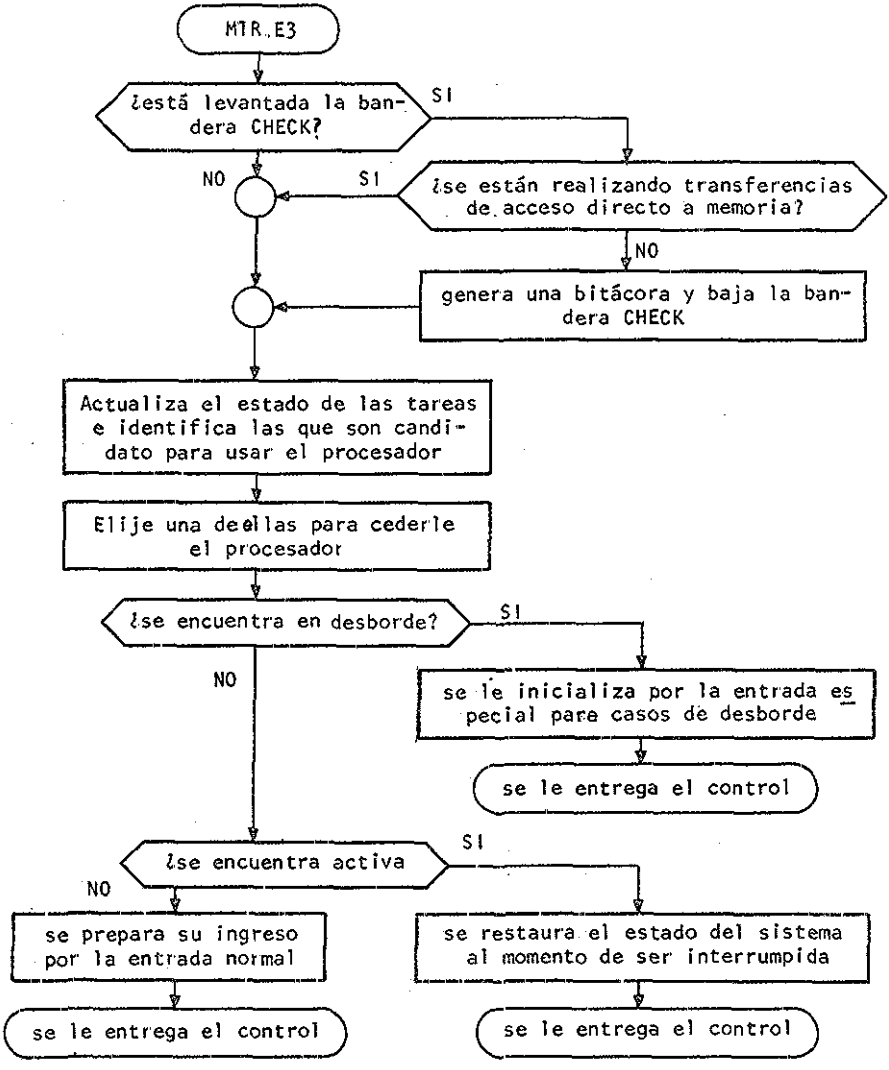


DIAGRAMA DE BLOQUES DE MTR (Continuación)

Por otro lado, genera los respaldos (mediante la subrutina CHEGEN) que emplea el programa INICIA para reiniciar el sistema en un estado conocido, después de una interrupción no prevista. Los respaldos se generan con una periodicidad determinada por el contador CONT_MIR+2 (aproximadamente cada minuto), siempre y cuando el contador de transferencias de acceso directo a memoria (DMAPEN), indique que no se está realizando ninguna transferencia a memoria que pueda cambiar la información contenida en esta durante su generación.

Otras variables de entrada para MIR son:

1) REL_SYS o reloj del sistema, de doble precisión (dos palabras de 16 bits) que se incrementa cada sesentavo de segundo, cuando interrumpe un dispositivo llamado reloj de tiempo real (RTR).

2) USUARIO que determina cual de todas las tareas se encuentra en operación en un momento dado

3) VEC_IRAN o vector de transacciones, está formado por un arreglo de N = MAX_PROGS elementos (ver descripción de VEC_IRAN en ESTRUCTURA DE DATOS), y que resume el estado de cada una de las tareas que compiten por usar el sistema.

4) CONT_MIR que se inicializa con el valor uno, cada vez que RTR interrumpe a una tarea y se incrementa cada vez que RTR interrumpe a un programa del sistema. Si la cuenta alcanza un umbral prefijado, entonces SOI imprime un mensaje advirtiendo al operador que el procesador está siendo monopolizado por los programas del sistema ope-

rativo

5) DIRE.PRIOR (ver su descripción en ESTRUCTURA DE DATOS) que sirve para localizar una tarea dada su prioridad

6) ELI.CONTA que se inicializa al principio de la operación del sistema y se usa para distribuir el tiempo de procesador entre las tareas, según un algoritmo que se describe junto con la subrutina ELI.US. A excepción del directorio por prioridad (DIRE.PRIOR), todas estas variables también son variables de salida. Es decir, pueden ser modificadas durante la ejecución MIR.

4.2.3 CHEGEN

Esta subrutina escribe en disco magnético, en el archivo CHECK.NEU, toda la información contenida en las primeras 256 (400 octal) palabras de memoria, seguida del contenido del segmento COMUN donde se encuentran las variables del sistema.

Para lograr que las variables de las tareas se incluyan en el segmento COMUN y, por lo tanto, en los respaldos, es necesario que se sigan las siguientes reglas (ver el ejemplo del apéndice C):

1) Todas las tareas se deben ensamblar juntas y con el archivo ESO1C.NEU

2) Antes de declarar una variable o un grupo de variables consecutivas es necesario usar la seudoinstrucción ".CSECT COMUN" para so -

licitar que se incluyan en dicho segmento

3) Antes de declarar una instrucción o grupo de instrucciones consecutivas se debe emplear la seudoinstrucción ".CSECI IAREAS" para solicitar que no se incluyan en el segmento COMUN

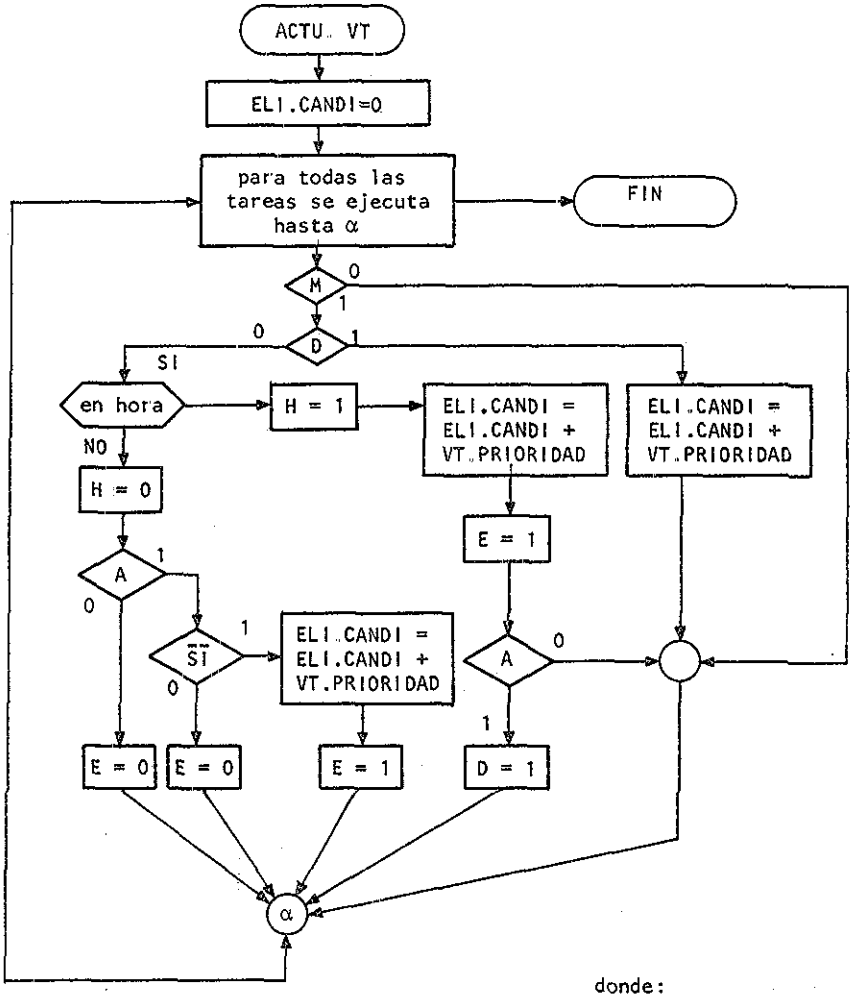
4) Después de declarar todas las variables de todas las tareas, pero todavía dentro del segmento COMUN, se debe declarar la etiqueta "FINDAT:", que le indica a SOl el fin del segmento COMUN y, con ello, el fin de la información que se debe respaldar

4.2.4 ACTU.VT

ACIU.VI es una subrutina de MIR que actualiza, para cada tarea, las palabras de estado que se encuentran en el vector de transacciones. En particular busca las condiciones de cada tarea en hora de entrar (EDO.HOR), desborde (EDO.DES), y elegibilidad (EDO.ELE); anota los hallazgos en las palabras de estado correspondientes.

MIR calcula la siguiente hora de entrada (VI.HE) cada vez que activa a una tarea por su dirección inicial (VI.START), y hace cero el dígito EDO.HOR de su palabra de estado VI.ESIADO.

Se dice que una tarea está en hora de entrar cuando el reloj del sistema REL.SYS se hace mayor o igual a VI.HE. ACIU.VI hace EDO.HOR = 1 en el momento en que detecta dicha condición.



donde:

M = EDO.MEZ
D = EDO.DES
H = EDO.HOR
E = EDO.ELE
A = EDO.ACT
SI= $\overline{(EDO.SUS)}\,\overline{(EDO.INT)}$

DIAGRAMA DE BLOQUES DE ACTU.VT

MTR también hace EDO.ACI = 1 cada vez que activa una tarea, y no lo apaga (EDO.ACI = 0) hasta que la tarea termina la ejecución. Se dice que una tarea está en dosborde (EDO.DES =1) a partir del momento en que EDO. HOR = 1 y EDO.ACI = 1.

Por último, en ocasiones se desea que alguna tarea específica no se ejecute. Existe un dígito binario (EDO.MEZ) en la palabra de estado que sirve para diferenciar las tareas que se encuentran en la mezcla (EDO.MEZ = 1) de aquellas que no deben ser ejecutadas (EDO.MEZ = 0), y también existen otros dígitos (EDO.INI y EDO.SUS) que sirven para suspender temporalmente la ejecución de las tareas. Se dice que una tarea es candidata para ser ejecutada (es elegible, EDO.ELE = 1) si está la mezcla y además está en hora de entrar o si está en mezcla y también está activa pero no interrumpida ni suspendida temporalmente.

Resumiendo:

$$\text{EDO.HOR} = 1 \text{ si REL.SYS} > \text{VT.HE}$$

$$\text{EDO.DES} = 1 \text{ si (EDO.HOR)(EDO.ACI)} = 1$$

$$\text{EDO.ELE} = 1 \text{ si (EDO.MEZ) (EDO.HOR)} = 1$$

$$\text{o si (EDO.MEZ) (EDO.ACI)} \overline{(\text{EDO.INT})}\ \overline{(\text{EDO.SUS})} = 1$$

Además de buscar y marcar estas condiciones para cada tarea, ACIU.VI inscribe en una lista de candidatos (ELI.CANDI) a las que resultan ser elegibles. Específicamente inscribe la prioridad de las tareas que compiten en ese momento por usar el procesador. Esta preinscripción facilita el proceso de elección del nuevo usuario, función que le co -

rresponde a la subrutina ELI.US.

4.2.5 *ELI.US*

Esta subrutina elige,a partir de la lista de candidatos ELI.CANDI y del estado actual de un contador especial (ELI.CONIA), una tarea para que MIR le otorgue el próximo sesentavo de segundo.

Para garantizar que la tarea de prioridad p cuente con un tiempo de procesador t_p mayor o igual a $(1/2^p)t_u$, en donde t_u es el tiempo útil del procesador, se emplea un algoritmo inspirado en el comportamiento de los dígitos de un contador binario puro (ELI.CONIA), en donde el bit menos significativo cambia de estado con cada incremento (60 veces por segundo), el que le sigue en significancia cambia la mitad de las veces (30 por segundo), etc.

Previendo cambios en los algoritmos de asignación de recursos ELI.US realiza su función en dos etapas. En la primera etapa elige, con ayuda de la subrutina PRIORIDAD que se basa en el algoritmo mencionado,la prioridad de un candidato a utilizar el procesador y, en la segunda etapa elige,a través de la subrutina ELIGE, a un candidato que tenga esa prioridad.Actualmente esto es simple porque solo existe un candidato por prioridad,pero puede complicarse en el futuro cuando se elimine la restricción.

En todo caso, independientemente de los cambios que pueda sufrir ELI.US entrega la identificación de la tarea electa en la variable USUARIO.

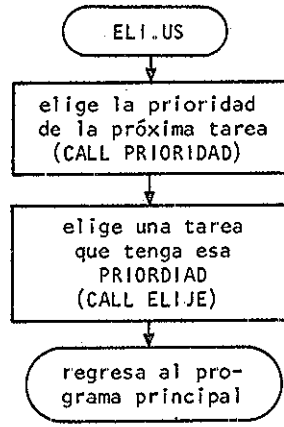


DIAGRAMA DE BLOQUES DE LA SUBRUTINA
ELI.US

4.2.6 PRIORIDAD

Esta subrutina emplea una variable llamada ELI.PRIOR para transferir el valor de la prioridad elegida a la subrutina ELIGE.

Lor primero que hace es verificar la existencia de candidatos y entrega ELI.PRIOR = 0 en caso de que no halla ninguno inscrito en ELI.CANDI.

Una vez verificada la existencia de candidatos; encuentra la prioridad nominal o sea la prioridad que tendría que tener el próximo usuario para que el tiempo de procesador se distribuyera según la fórmula $t_p = (1/2^p)t_u$.

Por último, como la tarea que tiene derecho al procesador (según la distribución mencionada) no siempre lo requiere, se verifica la existencia en ELI.CANDI de un candidato con prioridad igual a la nominal calculada. En caso negativo se elige la prioridad del candidato de menor prioridad, lo cual favorece a las tareas de menos prioridad sin afectar los derechos de las otras.

El algoritmo para calcular la prioridad nominal es el siguiente:

1) se guarda el valor de ELI.CONIA (RO= ELI.CONIA)

2) se incrementa dicho contador (ELI.CONIA = ELI.CONIA + 1)

3) se encuentran los dígitos binarios que cambiaron de valor con el incremento (ELI.IM = RO $\oplus$ ELI.CONIA).

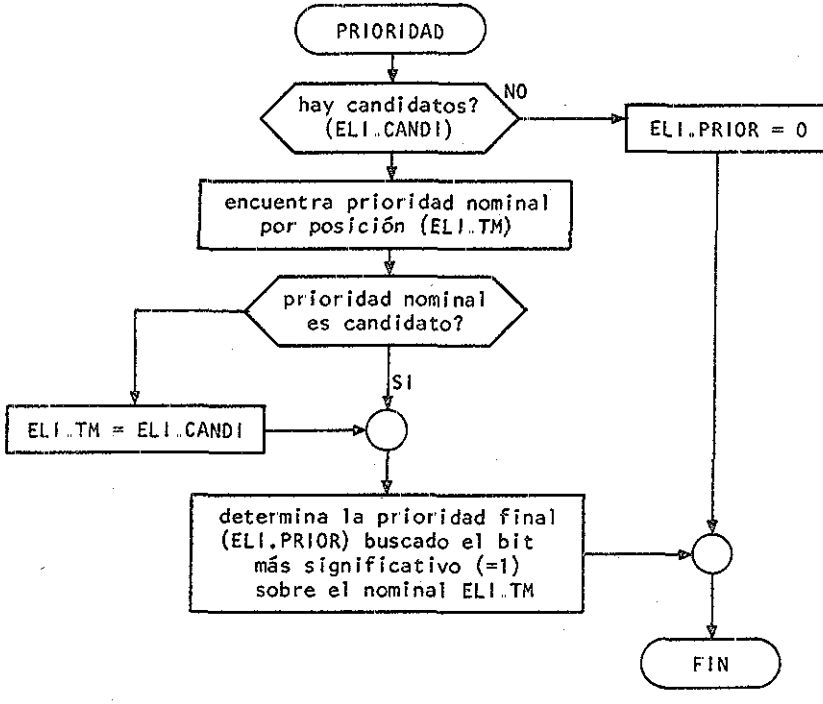


DIAGRAMA DE BLOQUES DE LA SUBRUTINA PRIORIDAD

Los dígitos binarios de ELI.IM, vistos de izquierda a derecha, siempre forman una secuencia de ceros seguida de una secuencia de unos (o,0...1,1, por ejemplo)

4) se encuentra el de mayor peso entre todos los que cambiaron (ELI.IM = ELI.TM ⊕ ELI.IM/2)

5) la posición de este dígito, contando de derecha a izquierda, representa el valor de la prioridad nominal (ver los diagramas de flujo y el listado general)

4.2.7 LLAMA

LLAMA es el intermediario entre las tareas de aplicación y el resto del sistema, incluyendo al equipo y al sistema operativo. Luego es un programa de servicio, y su función es dual. Por otro lado atiende a las tareas, facilitándoles los recursos del sistema (equipo periférico u otros programas), y por otro, sirve al sistema, filtrando las solicitudes de atención emitidas por las primeras. Analiza, y rechaza aquellas que no se pueden satisfacer o que no son válidas. Ejecuta el resto, o lo canaliza a otros programas cuando no son de su comptencia.

Las facilidades que este programa proporciona a los de aplicación se pueden clasificar en: a) funciones de control del proceso, b) comunicación con otras tareas, y c) trasnferencias de información entre tareas y equipo periférico. Entre las transferencias que involucran al equipo periférico se distinguen las transferencias hacia o desde memo -

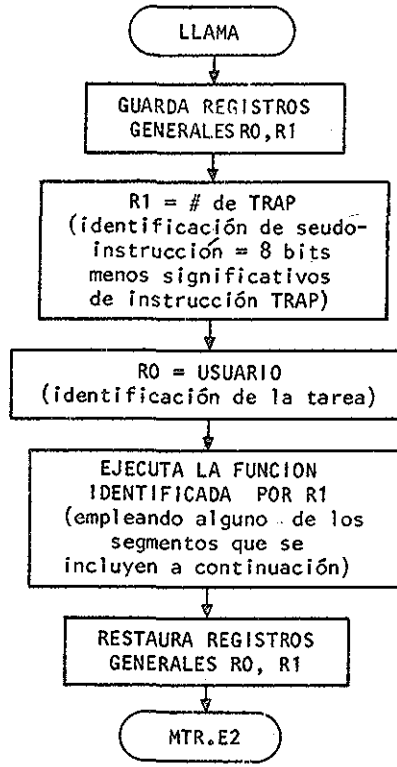


DIAGRAMA DE BLOQUES DE LLAMA

ria secundaria (disco magnético), y las de entrada y de salida (convertidores analógicos, por ejemplo).

Desde el punto de vista programático, se puede solicitar cualquiera de esos servicios empleando una seudoinstrucción ("programmed request"; ver apéndice B) en el programa de aplicación, cuya ejecución provoca interrupciones que son atendidas por IIAMA. Cada vez que el programa ensamblador encuentra una seudoinstrucción de este tipo, expande un macro que incluye el código de máquina necesario para transferir sus parámetros a IIAMA vía el registro general R0 y, por supuesto, una instrucción (en este caso, TRAP) que provoca la interrupción y además identifica, en sus 8 dígitos binarios menos significativos, el servicio que se solicita (ver definición de macros tipo "$XXX" en el listado general).

La identificación de los servicios diferentes (ver diagrama de bloques) se realiza empleando, a manera de índice, el número especificado en esos 8 dígitos binarios. El índice indica la posición, dentro de una tabla (LLA.IAB), de la dirección del segmento de programa que realiza la función.

Algunas de estas funciones se ejecutan por completo dentro del programa IIAMA; otras requieren de la intervención de rutinas de transferencia entre memoria y periféricos, que más bien forman parte del bloque de programas SERVES y del sistema RIII. Pero eventualmente IIAMA entrega el control, después de procesar la interrupción, al monitor MIR, vía la entrada especial MIR.E2.

Dada la extensión de este programa, y lo simple de algunas funciones, se discuten solo los segmentos más interesantes. (IRANSMIIE, RECIBE, BUSCA, CREA, DESECHA, GUARDA, SESCRIBE, SIEE). Los detalles pueden consultarse en el listado general.

4.2.8 TRANSMITE Y RECIBE

La comunicación entre tareas se realiza mediante los subprogramas (segmentos de IIAMA) SIRANSMIIE y SRECIBE.

Dado que para establecer el flujo de información entre tareas de aplicación es necesario el consentimiento de ambos: transmisor y receptor, los programas SIRANSMIIE y SRECIBE, y sus variantes IRANSMIIE y RECIBE (sin el prefijo "S"), definen 4 formas diferentes de llevar a cabo el proceso, a saber (ver seudoinstrucciones $(S)IRANSMIIE y $(S)RECIBE en el apéndice B):

1) Una tarea solicita transmitir información a otra con la seudoinstrucción $IRANSMIIE. La transmisión se lleva a cabo sólamente si la otra ya la está esperando, lo que se logra emitiendo un $SRECIBE con anticipación. (El prefijo "S" implica espera hasta que el otro programa emita el $IRANSMIIE que inicie la transmisión.) Si el receptor no se encuentra esperando, entonces se le notifica al transmisor

2) Una tarea solicita recibir información de otra, emitiendo una $RECIBE. Es necesario, para que tenga éxito la transmisión, que el transmisor esté esperando transmitirla, lo cual se logra sólamente si emi

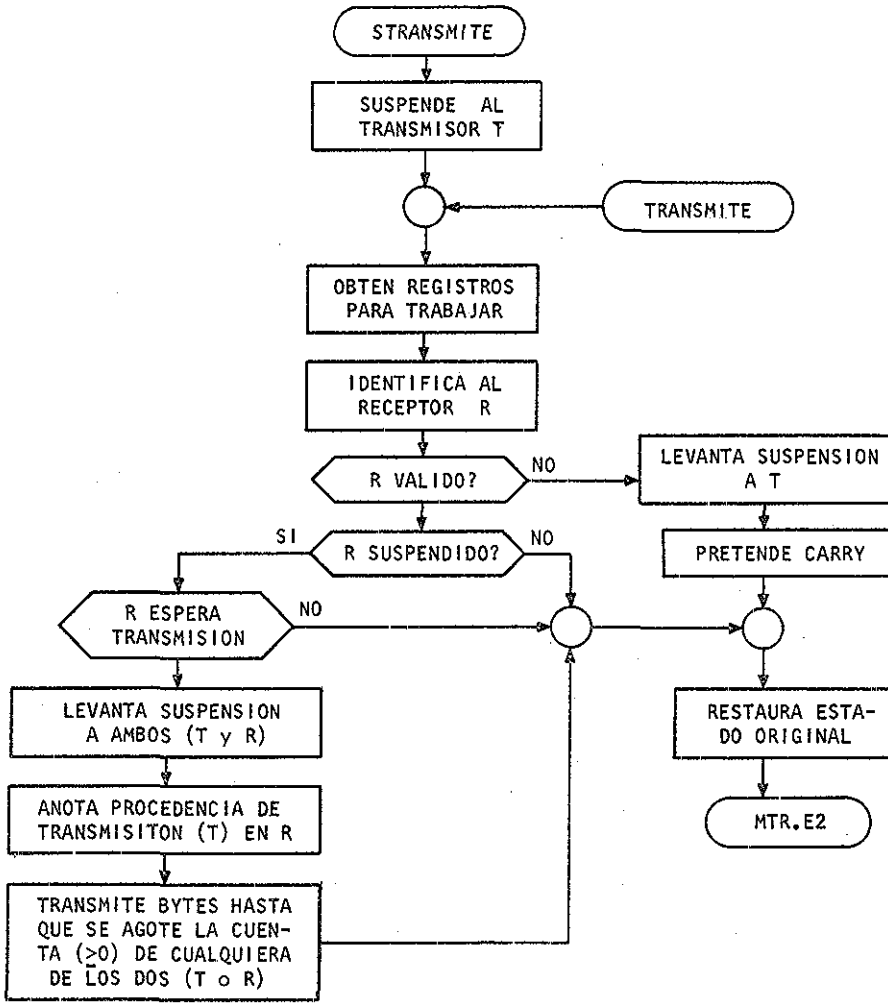


DIAGRAMA DE BLOQUES DE (S)TRANSMITE

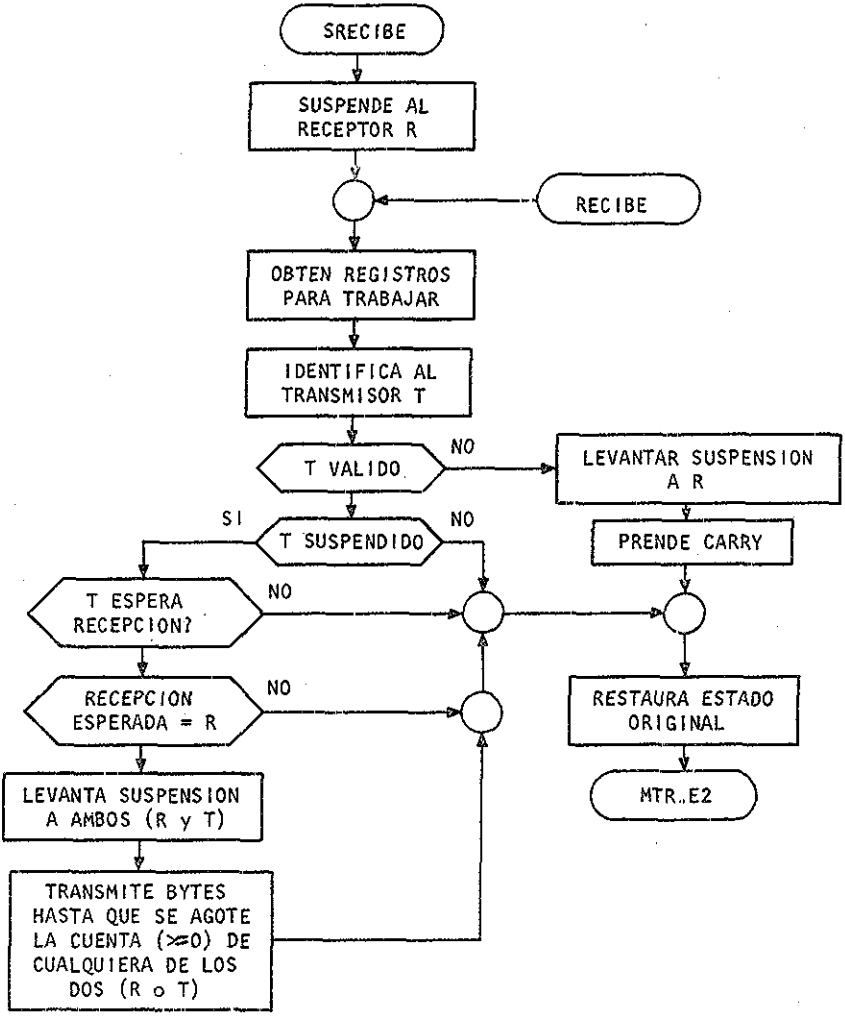


DIAGRAMA DE BLOQUES DE (S)RECIBE

te un $TRANSMITE con anticipación. En caso negativo, se notifica al receptor.

3) Una tarea solicita transmitir mediante un $TRANSMITE. Si la otra tarea se encuentra esperando, entonces se realiza la transmisión. En caso contrario, se suspende al transmisor hasta que el receptor lo libere aceptando al menos una porción de la información.

4) Una tarea solicita recibir información mediante un $RECIBE. Si la otra se encuentra esperando transmitir, entonces se establece la comunicación. En caso de no ser así, se suspende al receptor hasta que algún transmisor lo libere transmitiéndolo al menos un dato.

SO1 está diseñado, por razones de confiabilidad, para evitar que una tarea interfiera con el resto. Por principio, una tarea no debería ser capaz de modificar las variables o el curso de ejecución de otra sin su consentimiento. Con el esquema de comunicación descrito, y el equipo de protección de memoria, se logra esta meta sin impedir las relaciones de comunicación o dependencia entre tareas (excepto, claro está, la relación dictatorial). Es posible, por ejemplo, que una tarea se someta voluntariamente a otra, poniéndose en estado de espera (con $RECIBE), hasta recibir un comando que la libere.

Para hacer aún más flexible la comunicación, un programa que espera recibir información o comandos puede ser liberado o activado por cualquier otro que esté dispuesto a transmitirle al menos un dato. Es decir, un programa receptor se puede someter a varios transmisores (no a uno solo), y la longitud o tamaño del mensaje puede ser desconocido pa

ra el receptor y diferentes para cada uno de los transmisores.

En resumen, se permite que cualquier tarea transmita información a otra en porciones de longitud variable, y que reciba mensajes cuyo origen y tamaño se determina en el momento de la recepción.

Por último, las restricciones que existen en la transmisión, son de tipo físico (no de orden lógico), y obedecen nuevamente a la necesidad de preservar la confiabilidad del sistema. Dado que LLAMA (no las tareas) es el que realiza la transferencia de información, y que no es admisible que una tarea lo induzca a hacer transferencias inválidas o demasiado largas, y que tampoco es deseable hacer demasiadas verificaciones en cuanto a la validez de éstas, la longitud de las transferencias se restringe a un máximo, por seudoinstrucción, de 256 datos de 8 dígitos binarios (256 bytes). Si se desea transmitir más información, entonces es necesario hacerlo por partes.

En el diagrama de bloques de ambos subprogramas, SRECIBE y SIRANSMIIE, se puede observar que cuentan con dos entradas adicionales, RECIBE y IRANSMIIE, respectivamente, que lo único que no hacen es suspender al programa que provoca la interrupción; el resto del código es común para las opciones con suspensión (o espera al otro) y sin suspensión. Iambién se observa que los programas que atienden al transmisor y al receptor son completamente análogos, y la secuencia general que siguen es esta:

1) suspenden, en caso de entrar por la entrada con prefijo "S", al programa que solicita el servicio

2) guardan el valor de los registros de carácter general que necesitan para el proceso

3) verifican la validez del comando, comprobando la existencia de otro programa que se necesita para consumar la comunicación.

   En caso de que dicho argumento no sea válido, se levanta la suspensión, se notifica el hecho encendiendo el "CARRY" de la palabra de estado que corresponde a la tarea infractora, no al "CARRY" actual, (ver descripción de palabras de estado "PSW" y "CARRY" en el manual de la PDP11[9]), se restaura el estado de los registros de carácter general empleados, y se entrega el control al monitor vía la entrada MIR E2

4) verifican la disposición de la otra tarea para establecer la comunicación, o sea, investigan que la otra esté en espera. En caso afirmativo, se levanta la suspensión a ambas tareas y se transmiten tantos datos (bytes) como sea posible, con base en los argumentos entregados por cada una de las tareas. Esto es, cada tarea específica, mediante un contador, el número máximo de datos que está dispuesto a transmitir a recibir (según sea el caso), y IIAMA actualiza ambos contadores (y también las direcciones de transferencia) cada vez que transfiere un dato, hasta que la cuenta de transferencias faltantes se haga cero para algunos de ellos.

De esta forma, una tarea puede conocer el número de datos que le faltó transmitir o recibir, consultando el contador mencionado, y también puede conocer la dirección de los datos que no fueron aceptados por la otra tarea

5) finalmente, restauran el estado original de los registros generales empleados en el proceso y devuelven el control a MIR.

La única diferencia significativa entre ambos programas es el hecho de que TRANSMIIE sólo verifica que la tarea receptora esté en estado de espera, sin importar el origen esperado de la transmisión; en cambio, RECIBE, además de verificar que el transmisor se encuentra en estado de espera, también verifica que el destino del mensaje que espera transmitir sea el correcto.

4.2.9 *BUSCA, CREA, DESECHA, GUARDA, ESCRIBE y SLEE*

Todas las operaciones con memoria secundaria (en nuestro caso particular, con disco magnético), que pueden realizar las tareas de aplicación bajo SO1, se llevan a cabo por rutinas del sistema operativo RIIl de la PDP11. Sin embargo, dado que RIIl no es un sistema multitarea, las tareas no podrían solicitar el servicio a este en forma directa, sin causar conflictos y sin comprometer la confiabilidad de todo el proceso. No se podría evitar, por ejemplo, que una tarea destruyera la información contenida en disco, perteneciente a otra tarea, ya que RIIl las vería como una sola. Tampoco sería aceptable que una tarea solicitara a RIIl una transferencia con indicación de que no devuelva el control del procesador hasta

terminar la transferencia, lo que puede tomar demasiado tiempo.

Luego, para evitar que diferentes programas interfieran entre sí, o que provoquen la pérdida (aunque sea temporal) del procesador, es necesario que LLAMA supervise y regule las solicitudes de operaciones con memoria secundaria.

En este caso, LLAMA presenta una nueva faceta, sirviendo como intermediario entre tareas y RI11. Esto no se menciona entre las generalidades del programa LLAMA porque se espera prescindir de RI11 en un futuro agregando a SO1 las rutinas necesarias como parte de SERVES.

No obstante, la filosofía general de LLAMA, no se modifica independientemente de la identidad de los programas del sistema que se vean involucrados en la operación de servicio. Analiza las solicitudes de las tareas, verifica su validez y viabilidad, rechaza las que no son aceptables, y le da curso o recanaliza (según sea el caso) al resto.

Las seudoinstrucciones para usar el disco magnetico que SO1 ofrece a los programas de aplicación son: $BUSCA, $CREA, $DESECHA, $GUARDA, $SESCRIBE y $SLEE. Su descripción detallada puede ser consultada en el apéndice B, y corresponden a los "programed requests" LOOKUP, ENTER, DELETE, CLOSE, WRITW y READW de RI11, respectivamente, cuya descripción se encuentra en el manual correspondiente.

Como existe discrepancia entre los fabricantes de equipo de cómputo y los autores de textos respecto a los conceptos de archivo y de canal de

acceso a un periférico, por claridad y para la exposición de este escrito se definen a continuación:

Un archivo es una porción física de un periférico, identificable mediante un nombre arbitrario (nombre de archivo) fijado por una tarea, que sirve como memoria secundaria para guardar información o consultarla. Por medio de los archivos se puede hacer referencia, por su nombre, al conjunto de datos que contiene, independientemente de las características físicas del periférico empleado.

En cuanto al concepto del canal algunos autores {5 } asocian el nombre de canal de entrada o salida a un procesador especial (el equipo) que ejecuta instrucciones de entrada y salida. Otros {11} lo asocian a un conjunto de programas y variables cuya función es realizar este tipo de operaciones.

Dado que SO1 no es el que realiza las operaciones con disco magnético, basta definirlo en forma abstracta como sigue:

Un canal de acceso, o simplemente un canal, es una vía de comunicación que se asigna, por un tiempo determinado, a un programa para transferir información hacia o desde un archivo fijo cualquiera.

SO1 usa actualmente 16 canales de RI11, numerados de cero a 15, pero sólamente los 15 primeros (los numerados del cero al 14) pueden ser usados para las tareas de aplicación. El restante se reserva para uso exclusivo de los programas del sistema.

Desde un punto de vista programático, lo que se necesita saber acerca de estos canales, es:

1) que antes de poder consultar o modificar la información contenida en un archivo, es necesario solicitar un canal (usando la seudoinstrucción $BUSCA) que le sirva como medio para cumplir su objetivo

2) que si se desea generar un archivo nuevo en disco, entonces también debe solicitar un canal, empleando la seusoinstrucción $CREA

3) que si todos los canales están ocupados, el servicio será negado, y tendrá que solicitarlo de nuevo

4) que todas las lecturas y escrituras en disco se realizan mediante las seudoinstrucciones $SIEE y $SESCRIBE, haciendo referencia al canal asociado al archivo correspondiente, no al nombre del archivo

5) que LLAMA no le permite emplear un canal que no le ha sido asignado previamente

6) que debe liberar un canal después de usarlo mediante la seudoinstrucción $GUARDA, para que otros lo puedan utilizar, y

7) que para destruir un archivo, mediante un $DESECHA, también se le asigna temporalmente un canal (si lo hay disponible), pero que en este caso único no necesita solicitarlo ni tampoco liberarlo explícitamente.

Las funcionres $BUSCA, $CREA y $DESECHA son atendidas, después de su identificación, por los subprogramas BUSCA, CREA y DESECHA (vea y compare sus diagramas de bloques), los cuales son muy similares entre sí y siguen la secuencia que se describe a continuación.

1) Buscan, con ayuda de la subrutina GET.CANAL, un canal disponible para realizar la operación correspondiente. Si fracasan en el intento, entonces dan parte de esto a la tarea, encendiendo su CARRY a la vez que guardan el valor cero en el registro general número cero

2) Despositan en una área especial (VI.UTILERIA) la información que requiere RI11 para realizar la operación solicitada

3) Apuntan a dicha área con el registro general número cero para que RI11 la localice

4) Turnan la solicitud a RI11 mediante otra interrupción programada, y cuando este les devuelve el control, entonces notifican a la tarea del éxito o del fracaso de la operación, según la información proporcionada por RI11 en el CARRY y en la posición número 52 de memoria

5) Finalmente devuelven a MTR el control del procesador.

Además, cuando los segmentos BUSCA y CREA logran su objetivo principal, entonces también anotan, en la tabla DIRE,CAN, y en la posición que le corresponde al canal utilizado, la identificación (USUARIO) del

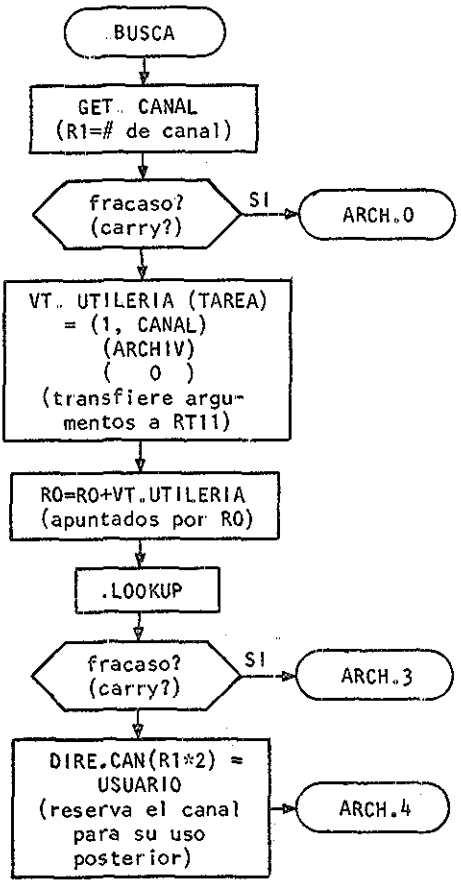


DIAGRAMA DE BLOQUES DE BUSCA

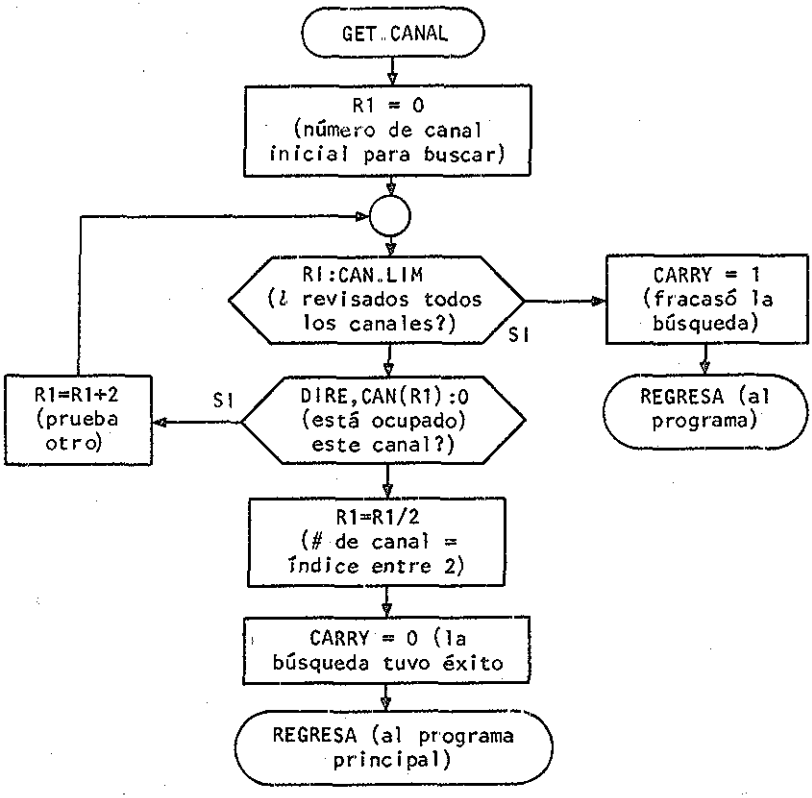


DIAGRAMA DE BLOQUES DE GET.CANAL

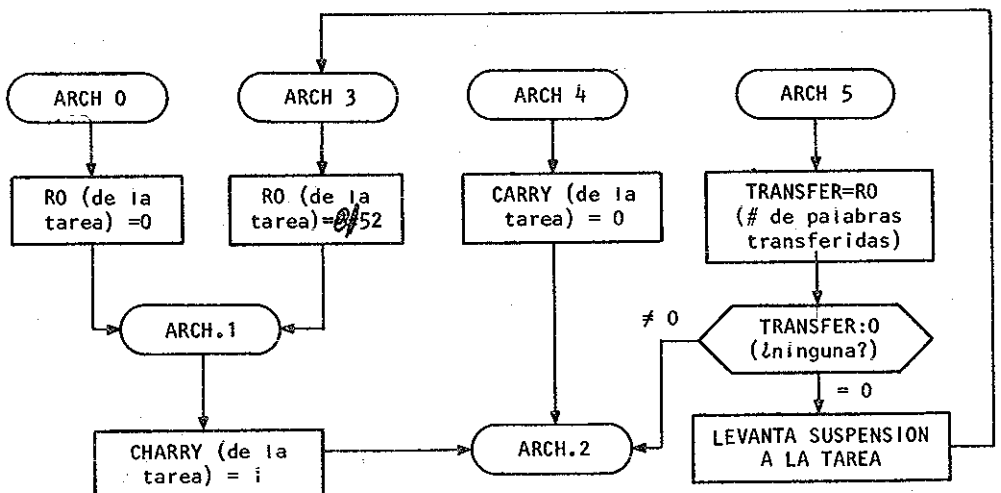


DIAGRAMA DE BLOQUES COMPLEMENTARIO PARA VARIAS SEUDO-INSTRUCCIONES

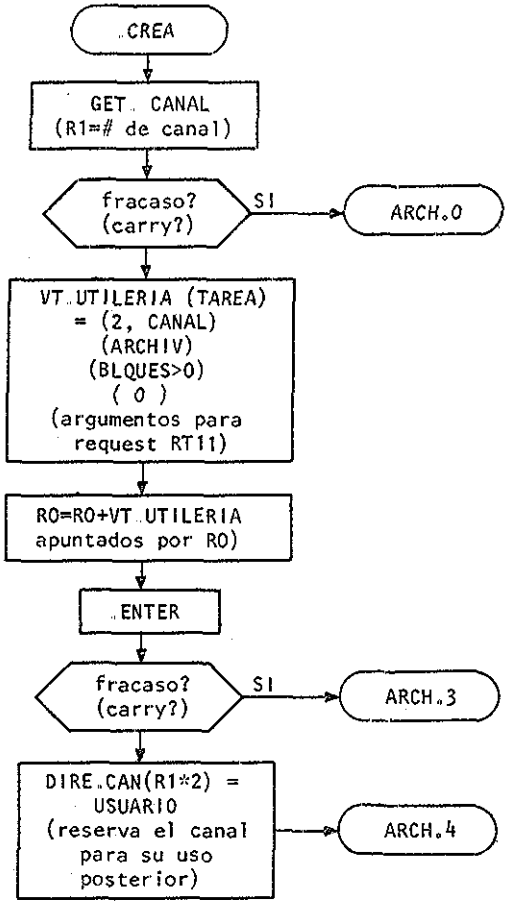


DIAGRAMA DE BLOQUES DE CREA

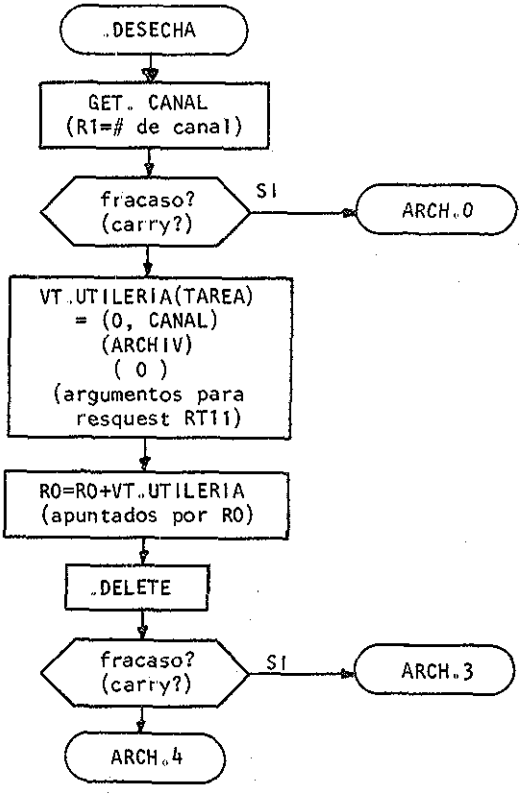


DIAGRAMA DE BLOQUES DE DESECHA

que solicitó el servicio. De esta forma, el archivo correspondiente queda asociado al canal y el canal queda asignado a la tarea.

SLEE (ver el diagrama de bloques) investiga primero si el canal a que hace referencia la seudoinstrucción $SLEE que lo activó está asignado efectivamente a la tarea que trata de utilizarlo. En caso contrario, notifica que el número de datos transferidos resultó ser cero (TRANSFER = 0), debido a que el canal especificado no le pertenece (R0 = 2 implica canal inválido), y devuelve el control a MIR. Si la referencia es válida, entonces suspende a la tarea para que no trate de utilizar la información solicitada antes de que se encuentre en su destino. Todas las transferencias hacia o desde el disco realizan con circuitos electrónicos de acceso directo a memoria, no por programa, y el tiempo que demoran es aprovechado por SO1 para ejecutar otras tareas. Cuando dichos ciruitos terminan su labor, entonces RI11 interrumpe a SO1 para que este levante la suspensión y le indique el resultado a la tarea que solicitó el servicio.

Después de suspender a la tarea, SLEE vacía en una tabla especial (VI. UTILERIA) toda la información que necesita transferir a RI11 para turnarle la solicitud, la apunta con el registro R0, y provoca la interrupción que lo activa. RI11 inicia la transferencia, si es factible, y devuelve el control a SLEE, indicando el resultado de la iniciación. Si no es posible transferir un solo dato (TRANSFER = 0), entonces SLEE lo notifica a la tarea, junto con la causa explicada mediante la palabra 52 de memoria, levantándole la suspensión. Pero si la trans-

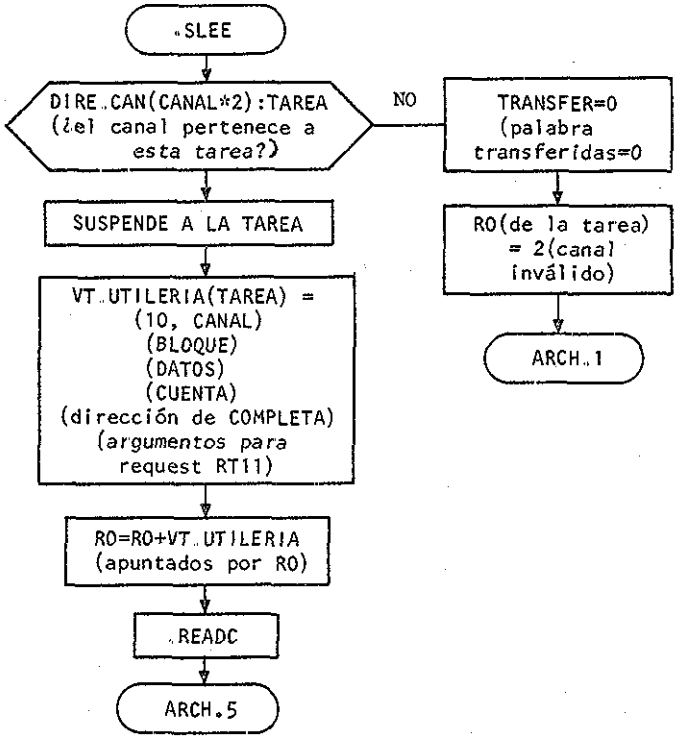


DIAGRAMA DE BLOQUES DE SLEE

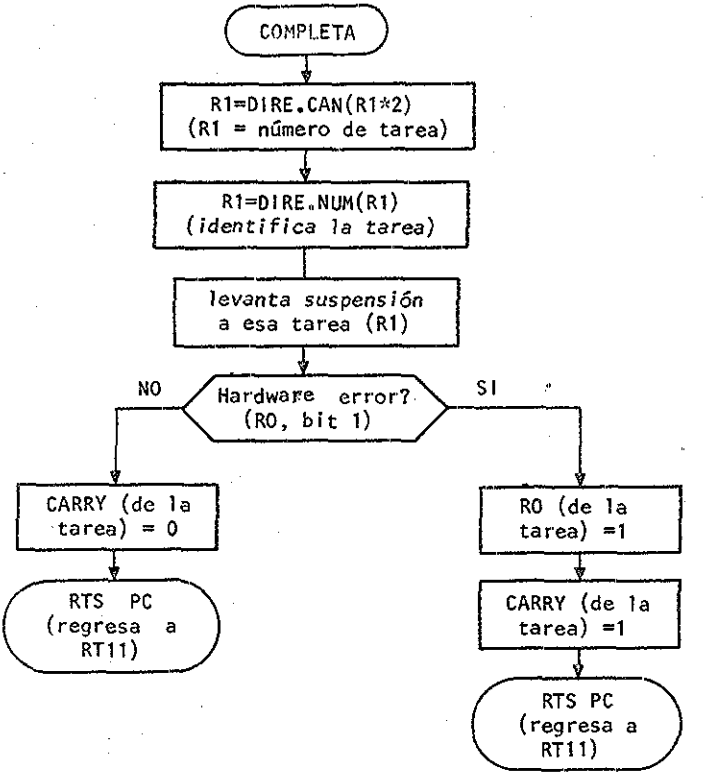


DIAGRAMA DE BLOQUES DE COMPLETA

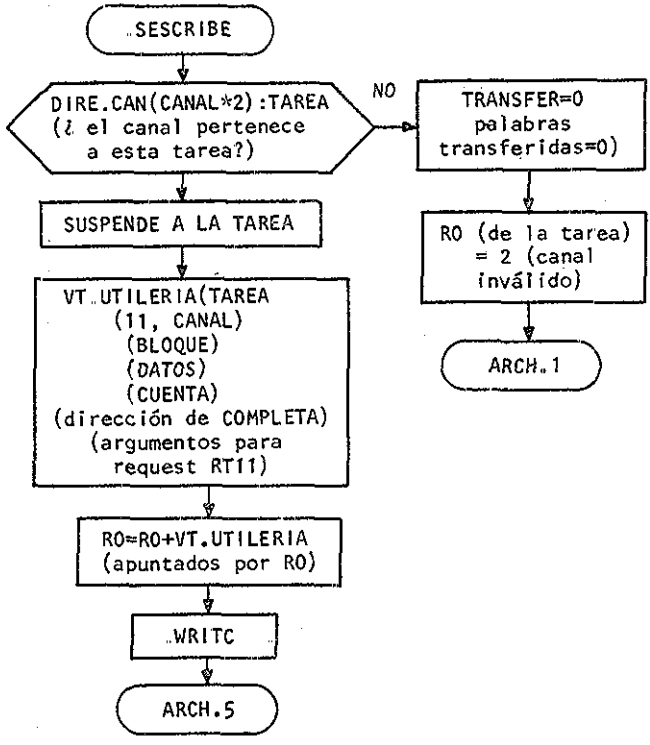


DIAGRAMA DE BLOQUES DE SESCRIBE

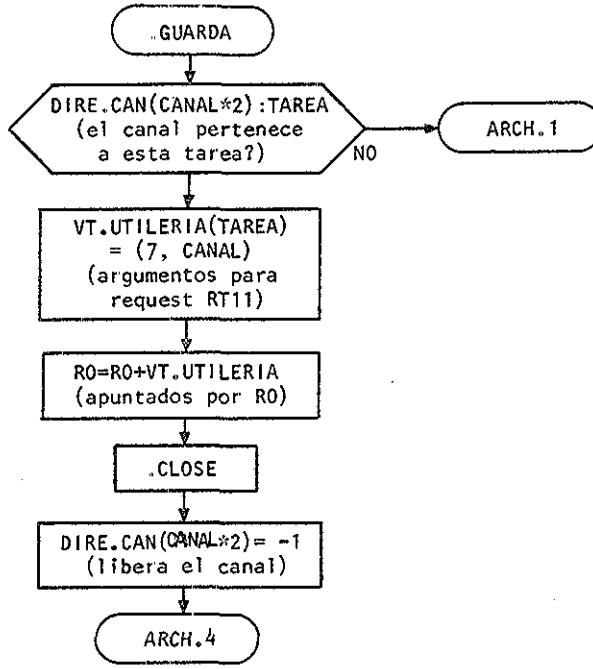


DIAGRAMA DE BLOQUES DE GUARDA

ferencia se inicia con éxito, entonces simplemente devuelve el control a MIR, para que aproveche el tiempo que dura.

Como se dijo anteriormente, RIll interrumpe a LLAMA cuando la transfe - rencia termina. Este lo atiende por la entrada COMPLEIA (ver diagrama de bloques) en la forma siguiente: con ayuda de la tabla o directorio de canales DIRE.CAN y el número del canal indicado en el registro R1, identifica a la tarea cuya transferencia ha terminado, le levanta la suspensión, le avisa de los resultados logrados vía el CARRY y el registro general RO, y entrega el control a MIR para que la active en el momento que le corresponda según su prioridad y la ocupación del procesador.

El segmento SESCRIBE, el que atiende a la seudoinstrucción $SESCRIBE, es completamente análogo al anterior, salvo que solicita a RIll una escritura en lugar de una lectura (compare los diagramas de bloques co - rrespondientes). GUARDA también es muy similar, pero, dado que no ne - cesita mucho tiempo, no suspende a la tarea que lo activa, y en lugar de asociar esta con el canal involucrado, disocia la relación de ocupación, anotando el número menos uno en la entrada correspondiente del directorio de canales. Este número negativo le indica a la subrutina GEI.CAN que el canal está libre y que puede ser asignado a cualquier programa que lo necesite (vea el diagrama de bloques de GEI.CAN).

El algoritmo que se emplea para encontrar un canal desocupado es una búsqueda lineal simple a lo largo de la tabla DIRE.CAN. El número de orden de la primera entrada cuyo valor sea negativo es anotado en el re

gistro general R1 para que sirva como índice en la operación de asignación, y el éxito o fracaso de la búsqueda se anota, como es costumbre, en el CARRY.

4.2.10 SERVES

Actualmente, SO1 maneja por interrupción solamente el convertidor análogo digital y el reloj que lo activa 1600 veces por segundo, esto es, 100 veces por cada uno de los 16 canales con que cuenta.

El convertidor digital análogo es operado completamente por LLAMA (ver segmentos DAX y DAY en el listado general, bajo el subtítulo de LLAMADAS), porque no provoca interrupciones.

El teletipo, los cassettes y el disco todavía se maneja a través de RT11, y la entrada y salida digital no han sido incluidas por falta de tiempo.

SERVES cuenta con una tabla de 16 entradas, llamada AD.AREA, en donde mantiene actualizados los valores de los 16 canales de conversión análoga digital, de manera que cuando una tarea le pide alguno de ellos a LLAMA, este lo entrega copiando el valor de la entrada correspondiente en esa tabla.

La adquisición de valores es continua desde la inicialización del sistema, etapa en la cual INICIA ajusta el reloj programable para que active al convertidor analógico digital, en forma automática, 1600 veces

por segundo. Cada vez que este es actividado, inicia una conversión por el canal especificado, e interrumpe al procesador cuando tiene lista la muestra. La subrutina AD.HAN, que forma parte del grupo SERVES, toma el dato, lo deposita en la entrada correspondiente de AD.AREA, especifica un nuevo canal de conversión para la próxima muestra, y devuelve el control al programa interrumpido (los detalles se pueden consultar en el listado feneral).

4.2.11 EMRGEN

Bajo el nombre genérico de EMRGEN están agrupados todos los programas que por interrupción atienden las condiciones detectadas que ponen en peligro la integridad del sistema.

Actualmente se contemplan 3 casos diferentes, a saber:

Las interrupciones provocadas por programas que intentan ejecutar una instrucción inválida, o examinar o modificar una posición inválida de memoria, y que son atendidas por la subrutina I4.10HAN .

Las interrupciones provocadas por interrupción o reestablecimiento de energía eléctrica, que son atendidas por A.C.HAN y, finalmente,

Las producidas por el conmutador entre procesadores, cuando el procesador suplente comprueba el buen funcionamiento del titular, que son atendidas por SWIHAN.

4.2.1 2T4.10HAN

La filosofía general de I4.10HAN es expulsar a las tareas de aplicación que tratan de usar instrucciones o posiciones de memoria inválidas, y suspender la operación de SO1 cuando este es el infractor (evento que se considera como prueba de que el equipo o el programa no funciona correctamente).

Sin embargo, existe un caso especial: Durante la inicialización del sistema, INICIA investiga la existencia del equipo periférico con que cuenta SO1. Para lograrlo, hace referencia (mediante una instrucción ISI) a posiciones de memoria que podrían no existir y, para evitar que esto sea tomado como señal de mal comportamiento, avisa a I4.10HAN (mediante la bandera IOERR) que se trata de una prueba de equipo periférico. Por su parte, I4.10HAN le contesta, con la misma bandera, que la referencia provocó una interrupción, de lo cual se concluye que la posición no responde y que, por lo tanto, el periférico correspondiente no existe o no funciona correctamente.

En caso de que la interrupción provocada por SO1 no sea una prueba de equipo periférico, I4.10HAN salta al segmento RENUNCIA, en donde prepara un posible retorno a RI11, restaurando los valores de algunas variables afectadas, y suena una alarma localizada en el teletipo (ver listado general).

La expulsión de una tarea se reduce a: sacarla de la mezcla (apagando el dígito binario EDO.MEZ en su palabra de estado VI.ESTADO, con lo

cual nunca más es activida), liberar los canales que haya estado empleando, e imprimir un mensaje de advertencia al operador.

4.2.13 A.C.HAN

Emplea la variable "A.C." para determinar si se trata de una interrupción o del reestablecimiento de la energía eléctrica. En el primer caso actualiza la variable "A.C." y copia el valor de todos los registros volátiles del procesador, escribiéndolos en la memoria de núcleos magnéticos. En el segundo, el segmento REESTABLECE (ver listado) también actualiza "A.C.", protege el conmutador entre procesadores, espera a que el teletipo esté en condiciones de imprimir un mensaje de advertencia, inicializa los periféricos mediante la subrutina INIPER, notifica a las tareas que se perdieron las transferencias que se estaban realizando con el equipo periférico (INIERR), y repone el valor original de los registros volátiles para devolver el control al programa interrumpido.

4.2.14 SWIHAN

La vigilancia entre procesadores se lleva a cabo mediante un dispositivo especial incluido en el conmutador S de la figura 3, el cual permite compartir cualquier tipo de periférico, incluso memoria, como se dijo anteriormente. Este dispositivo adicional, llamado "WAICHDOG-TIMER" por el fabricante permite que el procesador suplente vigile al titular de la siguiente manera:

Durante la inicialización se le asigna el control del conmutador, y de

todos los periféricos incluidos, al procesador titular. Posteriormente se ejecuta en el suplente un programa que solicita periódicamente el control del conmutador. Cada vez que lo hace, el titular es interrumpido, y si este no ejecuta una subrutina (SWIHAN) que rehace la solicitud en un lapso de tiempo razonable, entonces el conmutador pasa automáticamente al dominio del suplente. Aunque SO1 podría funcionar perfectamente sin el conmutador, la pérdida se ha definido arbitrariamente como incapacidad para retenerlo y, por ende, prueba de un mal funcionamiento. SWIHAN rechaza la solicitud y posteriormente comprueba si todavía tiene control sobre el conmutador (ver listado general). En caso afirmativo, devuelve el control al programa interrumpido. En caso contrario, realiza una secuencia (PERDIDO) de renuncia similar a la de I4. IOHAN; preparando un posible retorno a RI11, repone el valor de algunas variables afectadas y suena una alarma en el teletipo, después de imprimir un mensaje de advertencia.

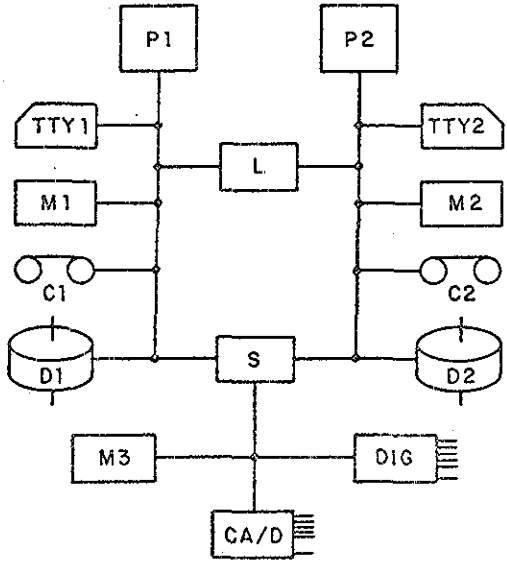
Cuando el procesador suplente se da cuenta de que ganó el control del conmutador, imprime un mensaje de alarma, abandona las tareas que estaba ejecutando, carga el sistema operativo SO1 y le entrega el control a INICIA, pasando así a la categoría de titular (ver el programa WAICHDOG para el suplente, incluido entre los listados). INICIA, por su parte, busca el último respaldo disponible para continuar la operación con un mínimo de pérdidas en el proceso.

CONCLUSIONES

Durante el desarrollo de este trabajo, y poco antes de terminarlo, se adquirió equipo adicional para el sistema, y se modificó ligeramente su distribución, quedando configurado como se muestra en la figura 7. En ella destacan dos cambios con respecto a la configuración original mostrada en la figura 3. Estos son:

Se adquirió una unidad adicional de disco magnético y, se le asignó una a cada procesador, inhabilitando la posibilidad de compartir ese medio.

También se adquirió un canal de comunicación I entre procesadores (UNIBUS LINK), que permite transferir grandes cantidades de información en lapsos de tiempo relativamente cortos.



P1, P2	Procesadores	D1, D2	Unidades de disco magnético
TTY1, TTY2	Teletipos	DIG	Canales digitales
M1, M2, M3	Módulos de memoria	CA/D	Convertidor análogo - digital
C1, C2	Unidades de cassette	S	Conmutador entre procesadores
L	Canal rápido de comunicación entre procesadores		

Fig 7 Configuración actual del sistema dual

Estas modificaciones dan como resultado un sistema mucho más poderoso y flexible que el anterior (compare las figuras 7, 3 y 2), pero impiden el uso de un disco compartido como medio de transporte para los respaldos. Por tal motivo, SO1 tuvo que ser modificarse.

En cambio resultó positivo porque, además de proporcionar las ventajas mencionadas, permitió constatar que el sistema operativo podía ser adaptado a diferentes configuraciones con relativa facilidad. Para lograrlo, bastó insertar una subrutina llamada RESPALDA en el cuerpo del programa que genera los respaldos (CHEGEN) de SO1, y otra en el programa que ejecuta el suplente. RESPALDA le transfiere a la otra vía el canal de comunicación L, una copia del último respaldo generado para que aquella actualice el suyo; con lo cual ambos sistemas quedan en igualdad de condiciones para continuar el proceso a partir de una falla.

Los mensajes entre procesadores constan de 264 palabras, de las cuales 8 son de control y 256 de datos, y sientan un precedente como esquema básico de multiproceso, ya que muestran una de las tantas formas en que es posible lograr que un procesador realice operaciones complejas bajo la supervisión de otro (los detalles de ambos insertos puedan observarse en el listado general).

En el apéndice C se muestra un ejemplo de dos tareas que operan bajo SO1 en diferentes condiciones con la última configuración. Demuestra, a pesar de su sencillez, la validez de la mayoría de las hipótesis formuladas.

Quedan pendientes, para una etapa de desarrollo posterior a este trabajo los siguientes puntos:

1) El manejo de memoria como medio de protección y relocalización dinámica de las tareas

2) La adquisición de dispositivos de memoria secundaria que permitan la sobreposición (overlaying) de programas

3) La inclusión de rutinas de entrada y salida que sustituyan a las de RIII

4) El diseño de lenguajes y compiladores que faciliten la elaboración de tareas

5) Las modificaciones que permitan que varias tareas compartan una misma prioridad

6) Los programas que permitan cambiar el curso del proceso desde el teclado

7) Protocolos de multiproceso efectivos

8) Proporcionar la opción de que las tareas respalden sus archivos en el disco del suplente en forma análoga a como se respaldan sus variables

APENDICE A

BIBLIOGRAFIA

1. Anderson I. and Randell B. (editores), "Computing Systems Reliability", Cambridge University Press, 1979

2. Blakeney, Gordon R., y otros, "An application - oriented multiprocessing system", IBM Systems Journal, vol. VI, number 2, 1967

   Es una descripción preliminar, bastante completa, del sistema IBM 9020 desarrollado por "The Federal Aviation Administration (FAA)" de los Estados Unidos , para control de tráfico aéreo. Representa un sistema típico de computadoras redundante.

3. COMIRE Corp., Enslow, Philip H., Jr., ed., "Multiprocessors and Parallel Processing", Wiley, 1974

   Describe varios sistemas multiprocesadores y menciona algunos criterios que emplearon los fabricantes para aumentar su confiabilidad.

4. Friedman, Arthur D., "Fault Detection in Digital Circuits", Prentice Hall, 1971

5. Madnick Stuart E., Donovan John J., "Operating Systems", Mc Graw Hill, 1974

6. Myers Glenford J.,Software Reliability,Principles and Practices", Wiley-Interscience, 1976

7. Watson, Richard W., "Timesharing System Design Concepts", McGraw Hill, 1970

   Incluye algunos conceptos útiles para la generación de respaldos (capítulo 6), y dedica un capítulo (el 7) a confiabilidad y recuperación de fallas.

8. Yourdon, Edward, "Design of On-line Computer Systems", Prentice - Hall, 1972

   Trata los problemas intrínsecos a los sistemas que operan en tiempo real e incluyen algunas consideraciones relativas a la recuperación de fallas (capítulo U).

9. Digital Equipment Corporation, "PDP11/40 Processor Handbook", 1973.

10. Digital Equipment Corporation, "DT03-FM/FP Unibus Switch Option Description". Aug 1974

11. Digital Equipment Corporation, "RI-11 System Reference Manual", Oct 1974.

APENDICE B. SEUDOINSTRUCCIONES (PROGRAMMED REQUESTS) DE SO1

Muchas de las operaciones de transferencia entre memoria y periféricos que ofrece SO1 son realizados por medio de subrutinas de RT11. Sin embargo, si el programador de tareas no tiene un conocimiento profundo de ambos sistemas, no es recomendable que invoque a RT11 en forma directa. Es mejor que lo haga a través de SO1 para evitar la destrucción de los arreglos del sistema.

A continuación se incluyen una lista completa y algunas consideraciones que es necesario conocer respecto a las seudoinstrucciones de SO1.

Consideraciones comunes a las seudosintrucciones de SO1

1) Todas las operaciones con memoria secundaria (disco magnético) se realizan mediante un canal de comunicación asignado por SO1

Las tareas no necesitan conocer el número de canal que se les asigna y que puede variar entre 0 y 14; sin embargo, debe proporcionar una variable para la asignación, y debe tener cuidado de no destruir esa información si se le hará referencia a él.

Los canales de comunicación pueden solicitarse con las seudoinstrucciones $BUSCA, $CREA y $DESECHA, en las cuales es necesario especificar el nombre del archivo al que se quiere tener acceso; deben liberar el canal (excepto en el caso de $DESECHA) mediante la seudoinstrucción $GUARDA, cuando no lo necesiten, con el objeto de que otras tareas puedan usarlo.

El nombre de un archivo se compone de dos partes: un nombre propio formado por un máximo de seis caracteres alfa-numéricos y una extensión o apellido formado por un máximo de tres. Es importante que tareas diferentes utilicen nombres o extensiones diferentes para evitar interferencias indeseables.

Dichos nombres, usados como argumentos en seudoinstrucciones, siempre ocupan 3 palabras de 16 bits. Si es necesario, el nombre propio se completa a 6 carácteres con espacios a la derecha, se codifica en código RADIX 50 (ver manual de RT11), y se guarda en las 2 primeras. La extensión también se completa con espacios a la derecha hasta tener 3 caracteres, se codifica en código RADIX 50, y se guarda en la tercera y última palabra.

Ejemplo de la forma en que se especifica un nombre de archivo para SO1:

```
EIIQ:   .RAD50 / NOM/
        .RAD50 / B__/
        .RAD50 / XT_/
```

Otros ejemplos:

```
EIIQ:   .RAD50 /NOMB__XT_/
NOM1:   .RAD50 /EJMPLO1__/
NOM2:   .RAD50 /EJMPLO/
        .RAD50 /2__/
```

2) Salvo indicación explícita (ver $DESBORDE, por ejemplo), todas las seudoinstrucciones de SO1 devuelven el control al monitor, y las tareas tienen que esperar un turno nuevo para continuar su ejecución.

3) Los argumentos para las seudoinstrucciones de SO1 son, en general, nombres de variables. Sin embargo, existe una variedad de argumentos en donde se puede emplear cualquier operando válido del lenguaje ensamblador de RT11. Esta opción se indica en el texto que sigue, aplicando un asterisco (*) al argumento.

Por ejemplo (ver descripción de $ADC), para pedir un dato del convertidor analógico-digital, el canal de conversión puede ser especificado en varias formas:

```
$ADC ALFA     ;   la variable ALFA contiene el número de canal
$ADC R4       ;   R4 contiene el número de canal
$ADC (SP)+    ;   el número de canal se extrae del stack
```

$ADC DIR,-(SP) DIR contiene la dirección del número de canal, y el valor adquirido se pone en el stack

etc.

4) En general, las seudoinstrucciones de SO1 cambian el valor de R0 y del CARRY, salvo en caso de indicación explícita.

Lista de seudosintrucciones:

$TERMINA

Función: Termina la ejecución de una tarea para que se vuelva a ejecutar por la dirección inicial cuando se cumpla su hora de entrada. R0 y CARRY no se afectan.

Llama: $IERMINA

Errores: Ninguno

$ESPERA

Función: Sirve como un retardo cuando una tarea desea esperar algún evento externo, y está dispuesta a ceder momentáneamente el procesador a otra tarea que lo quiera emplear.

El monitor le devuelve el control a la tarea que solicita el retardo según el esquema de prioridades y el número de candidatos que haya durante ese periodo, si no hay otro candidato,

se le devuelve de inmediato y si el sistema está saturado, se le devuelve cuando le corresponda según su prioridad. RO y CARRY no se afectan.

Lamada: $ESPERA

Errores: Ninguno

$SALE

Función: Saca de mezcla a la tarea que lo emite. RO y CARRY no se afectan

Llamada: $SALE

Errores: Ninguno

$ADC

Función: Adquiere el último valor obtenido a través de uno de los 16 canales del convertidor analógico-digital

Llamadas: $ADC

$ADC CANAL*†

$ADC CANAL*, DESTINO*†

CANAL* y DESTINO* son nombres de variables cuyos valores representan al canal (0-15) y el destino del valor adquirido, respectivamente (ver AD11

(†) Véanse también las notas generales referentes a los argumentos de las seudoinstrucciones de SO1, al principio de este apéndice.

user's guide). Si no se especifica DESTINO, el valor se entrega en RO, y si tampoco se especifica CANAL, el número de canal se toma de RO y se sustituye por el valor adquirido (RO siempre devuelve el valor transferido).

Errores: Si el canal especificado no es válido (≠ 0,1,2...15), entonces $ADC solamente prende el CARRY (C=1). Si la transferencia tiene éxito, se apaga el CARRY (C=0).

$DAX

Función: Transferir un dato al convertidor digital-analógico, canal X

Llamadas: $DAX

$DAX VALOR*

VALOR* (ver generalidades punto 3) es el nombre de una variable que contiene al número que se quiere convertir. Si no se especifica esta variable, se usa el valor contenido en RO. (RO siempre devuelve el valor transferido).

Errores: Si el número que se quiere convertir es mayor que 17600 (octal), no se efectúa la transferencia y se prende el CARRY. Si la transferencia tiene éxito, se apaga el CARRY.

$DAY

Función: Análoga a $DAX, pero por el canal Y.

Llamadas: $DAY

$DAY VALOR*

$STRANSMITE/$TRANSMITE

Función: Iransmiten un bloque de bytes a otra tarea. La diferencia consiste en que $SIRANSMIIE espera a que se transfiera al menos un byte antes de devolver el control. En cambio, $IRANSMITE puede devolver el control sin haber transmitido un solo byte.

Estas seusoinstrucciones son complementarias de $RECIBE y $SRECIBE, respectivamente. Es decir, $SIRANSMITE espera a que otra tarea emita el $RECIBE que permita continuar la ejecución del programa transmisor. En cambio, $SRECIBE espera a que otra tarea emita el $IRANSMITE que permita continuar la ejecución del programa receptor.

Iambiẽn es factible establecer una comunicación por medio del binomio ($SIRANSMIIE, $SRECIBE). En este caso, el orden en que se emitan las seusoinstrucciones es irrelevante, ya que la ejecución de las tareas se continúa a partir del momento en que se hayan emitido ambas.

Es importante hacer notar que estas seudoinstrucciones, además de permitir la transmisión de datos entre tareas, sincronizan su ejecución. Por ejemplo, una tarea puede ponerse en estado de espera (SUSPENDIDO) emitiendo un $SRECIBE hasta que otra tarea cualquiera le indique que puede continuar (le levante la SUSPENSION) por medio de un $IRANSMIIE.

Llamadas: $(S)TRANSMITE

$(S)TRANSMITE RECEPTOR, CUENTA, APUDAT

RECEPTOR es el nombre de una variable (BYTE) que contiene el número (0<# <256) de la tarea a la que se quiere transmitir

CUENTA es el nombre de una variable que contiene un número negativo de bytes que se desean transmitir

APUDAT es el nombre de un apuntador al bloque de bytes (buffer) que se desea transmitir.

Cuando se devuelve el control a la tarea, el contenido de esas variables es el siguiente:

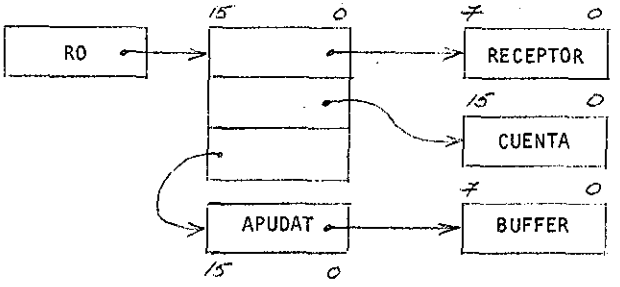
a) La variable RECEPTOR no se modifica

b) La variable CUENTA se incrementa con el número de bytes transmitidos. Es decir, al regresar, CUENTA = 0 implica que se transmitieron todos los bytes, y CUENTA = -n implica que la tarea receptora aceptó todos los bytes excepto los "n" últimos porque no cupieron en su buffer de recepción.
Note que esto permite usar buffers de transmisión y recepción de diferentes tamaños

c) La variable APUDAT se incrementa con el número de bytes transmitidos

d) El buffer no se modifica.

Estas seudoinstrucciones transfieren los argumentos a SO1 por medio de un bloque de tres palabras direccionado por RO. La primera palabra de este bloque contiene la dirección de la variable RECEPIOR y las si - guientes contienen las direcciones de la variable CUENIA y APUDAT, res pectivamente.



Cuando se devuelve el control a la tarea, dicho bloque queda como si - gue:

a) la dirección de RECEPIOR no se afecta

b) la dirección de CUENIA no se afecta

c) la dirección de APUDAI no se afecta

d) RO apunta al bloque de argumentos

Si se emplea la llamada $(S)IRANSMIIE RECEPIOR, CUENIA, APUDAI, auto - máticamente se genera el bloque con las direcciones mencionadas y se apunta por medio de RO.

Si se emplea la otra ($(S)IRANSMIIE), entonces es necesario que el programador se encargue de tomar tales medidas antes de la llamada.

El empleo de una combinación de ambas resulta cómodo cuando se desconoce el tamaño del buffer de la tarea receptora.

Ejemplo:

```
        $IRANSMIIE RECEPTOR, CUENTA, APUDAI; ARGUMENTOS EXPLICITOS
        BCS ERROR
        ISI CUENTA
        BPL FIN
SIGUE:  $IRANSMIIE                          ; ARGUMENTOS IMPLICITOS
        ISI CUENTA                          ; DETERMINADOS POR EL
        BMI SIGUE                           ; ANTERIOR
FIN:    ; aquí continúa el programa
```

Errores:a) durante el ensamblado se detecta número inválido de argumentos ($\neq$ 0,3)

b) durante la ejecución se detecta RECEPTOR inválido. Cuando devuelve el control:

C = 1 (o CARRY = 1) indica RECEPTOR inválido (RECEPTOR contiene un número R mayor que el máximo número de usuarios U posible).

$RECIBE/$SRECIBE

Función: complementaria de $SIRANSMIIE y $IRANSMIIE, respectivamente.

La forma de llamarlas es completamente análoga a la forma en que se llaman $(S)IRANSMITE, y sus argumentos son muy similares.

Llamadas: $(S)RECIBE

$(S)RECIBE IRANSMISOR, CUENIA, APUDAT

IRANSMISOR es el nombre de una variable (BYIE) que contiene el número (0<#<256) de la tarea de la cual se quiere recibir información. Si se desea recibir datos provenientes de cualquier tarea capaz de transmitirlos, el byte IRANSMISOR puede contener cualquier número válido de tarea.

CUENIA es el nombre de una variable que contiene un número negativo de bytes que se desea recibir

APUDAI es el nombre de un apuntador al bloque de bytes (buffer) de recepción.

Aunque se puede especificar un IRANSMISOR en particular, los datos transferidos no provienen necesariamente de él; pueden provenir de cualquier tarea que desee transmitir información a este receptor en particular. Para estar seguro de la procedencia de los datos, es necesario consultar el valor de la variable IRANSMISOR cuando se devuelva el control a la tarea receptora.

En resumen, el destino de una transferencia queda especificado desde el momento de la llamada $(S)IRANSMITE (argumentos), pero la procedencia de una transferencia queda condicionada a que existe más de una tarea que trate de transmitir a la misma tarea receptora.

Cuando se devuelve el control a la tarea receptora, el contenido de las variables es el siguiente:

a) IRANSMISOR contiene el número de la tarea de la cual proviene la transmisión (no necesariamente el número invocado)

b) CUENIA se incrementa con el número de bytes recibidos

c) APUDAI se incrementa con el número de bytes recibidos

d) el BUFFER contiene la información recibida (el contenido original se pierde).

La forma de transmitir los argumentos es completamente análoga a la de $(S)IRANSMITE (argumentos). Cuando se devuelve el control a la tarea receptora, el bloque donde se transfieren las direcciones de los argumentos a SOl queda como sigue:

a) dirección de IRANSMISOR no se afecta

b) dirección de CUENIA no se afecta

c) dirección de APUDAI no se modifica

d) RO apunta al bloque de argumentos

Errores: a) número inválido de argumentos (durante el ensamblado)

b) C = 1 implica TRANSMISOR inválido

$DESBORDE SEIPAL

$RETORNO SEIPAL

Solamente son necesarias cuando se desea tener la posiblidad, dentro de una subrutina de atención a desbordes de tiempo, de continuar la ejecución de la tarea en el mismo estado en que se encontraba al momento de detectarse el desborde.

Función: $DESBORDE guarda el estado en que se encontraba la tarea al momento de detectarse el desborde (R0-R5, PC y PSW) excepto SP. Por esta razón se responsabiliza al programador a devolver SP y el stack en general, tal y como lo recibe.

$RETORNO usa la información guardada por $DESBORDE para continuar la tarea interrumpida al detectarse el desborde.

Llamadas:

```
$DESBORDE  SEIPAI ;  primera instrucción
     .             ;  resto del programa
     .
     .
subrutina de
desborde
     .
     .
$RETORNO   SEIPAI
```

SEIPAL es el nombre de un buffer de 6 palabras en donde se pueden guardar temporalmente los registros (RO-R5).

NOTAS

a) Para emplear $REIORNO es absolutamente indispensable haber guardado el estado de la tarea por medio de $DESBORDE

b) Habiendo emitido $DESBORDE, se puede emitir más de un $REIORNO

c) En caso de emplear $DESBORDE, es absolutamente necesario que sea invocada antes de ejecutar cualquier instrucción o de invocar cualquiera otra seudoinstrucción dentro de la rutina de atención a desbordes

d) $DESBORDE no implica salir de la rutina con un $RETORNO. Se puede emplear cualquier otro medio que no provoque un desbalance en la pila ($TERMINA, $SALE, etc)

e) $DESBORDE devuelve inmediatamente el control a la tarea (no tiene que volver a esperar turno)

f) RO y CARRY no se afectan.

Errores: Número inválido de argumentos (durante el ensamblado).

$BUSCA

Función: Solicita un canal de transferencia, busca el archivo especificado, y lo asocia al canal para poder manipularlo.

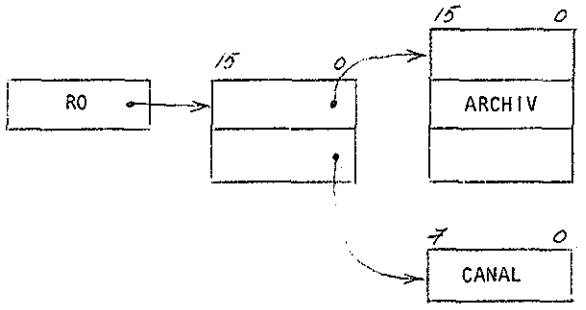
Llamadas: $BUSCA

$BUSCA ARCHIV, CANAL

ARCHIV es el nombre de un bloque de tres palabras que contiene el nombre del archivo al que se quiere tener acceso, dispuesto tal y como se especifica en la primera parte del apéndice.

CANAL es el nombre de una variable (BYTE) en la cual SO1 deposita un número (número de canal) asociado con el archivo invocado. Este número es una especie de nombre temporal que se le da al archivo para poder usarlo.

Estos argumentos son transferidos a SO1 por medio de un bloque de dos palabras apuntado por RO, que contiene las direcciones de ARCHIV y CANAL



Si se omiten, entonces el programador tiene que realizar esa construcción antes de efectuar la llamada $BUSCA.

Cuando se devuelve el control a la tarea:

a) CARRY = 0 indica que la búsqueda tuvo éxito y que CANAL contiene un número que puede ser usado para referirse al archivo

b) CARRY = 1 indica que la búsqueda no tuvo éxito, y entonces: RO = 1 implica que el archivo no existe y RO = 0 implica que, de momento, no hay un canal disponible para realizar la búsqueda.

Errores: a) Número inválido de argumentos (durante el ensamblado)

b) El archivo no fue encontrado (CARRY = 1 y RO = 1).

$CREA

Función: Solicita un canal de transferencia, crea un archivo con el nombre especificado y lo asocia al canal para poder manipular lo

Llamadas: $CREA

$CREA ARCHIV, CANAL, BLQUES

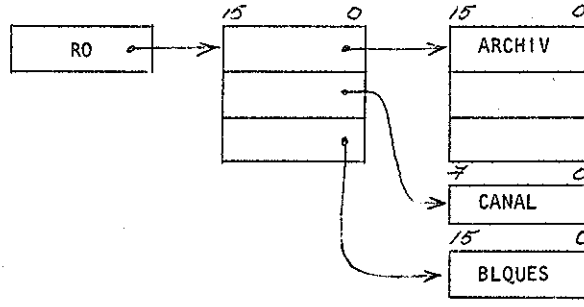
ARCHIV es el nombre de un bloque de 3 palabras que contiene el nombre del archivo que se quiere crear (consultar la primera parte del apéndice).

CANAL es el nombre de una variable (BYTE) a la cual SO1 asigna el número de canal por el cual se puede tener acceso al archivo.

BLQUES es el número (> 0) de bloques de 256 palabras que se quieren reservar para el archivo. Si SO1 encuentra que BLQUES tiene un valor menor o igual a cero, entonces automáticamente

le asigna el valor uno.

Los argumentos son transferidos a SOl en la forma normal, por medio de un bloque de tres palabras apuntado por RO, que contiene las direcciones de ARCHIV, CANAL y BLQUES, respectivamente.



Si se omiten, entonces el programador debe realizar la construcción antes de efectuar la llamada.

Cuando se le devuelve el control a la tarea:

a) CARRY = 0 implica que el archivo puede ser usado a través del canal especificado en la variable CANAL

b) CARRY = 1, RO = 0 : no hay canal disponible
CARRY = 1, RO = 1 : no hay BLQUES disponibles

Errores: Número inválido de argumentos (durante el ensamblador)

$DESECHA

Función: Solicita un canal de transferencia, desecha el archivo espe

cificado y finalmente libera el canal usado.

Llamadas: $DESECHA

$DESECHA ARCHIV

ARCHIV es el nombre de un bloque de 3 palabras que contiene el nombre del archivo que se quiere desechar (consultar la primera parte del apéndice). Si se omite, entonces R0 debe apuntar a dicho bloque antes de hacer la llamada (R0 = dirección del descriptor).

Cuando se devuelve el control a la tarea:

a) CARRY = 0 implica que la operación tuvo éxito

b) CARRY = 1, R0 = 0 : no hay canal disponible

CARRY = 1, R0 = 1 : archivo inexistente

Errores: Archivo inexistente (CARRY = 1, R0 = 1)

$SESCRIBE

Función: Transfiere un número especificado de palabras al canal de transferencia indicado. La tarea es suspendida hasta que se ejecute o rechace la transferencia.

Llamadas: $SESCRIBE

$SESCRIBE CANAL, DATOS, CUENTA, BLOQUE, TRANSFER

CANAL es el nombre de una variable que contiene un número asignado por SO1 para referirse a un archivo especificado en una

llamada $BUSCA o $CREA

DAIOS es el nombre de un arreglo (buffer) que contiene la información que se desea transferir.

CUENIA es el nombre de una variable que contiene el número de palabras que se desea transferir

BLOQUE es el nombre de una variable que contiene el número del bloques de 256 pálabras (relativo al principio del archivo) donde se quiere escribir.

IRANSFER es el nombre de una variable a la que SO1 asigna un valor igual al número de palabras que efectivamente pudo transferir (no necesariamente = CUENIA).

Estos argumentos se transfieren a SO1 en la forma normal, por medio de un bloque de cinco palabras apuntado por RO, que contienen las direcciones de las variables en el mismo orden que se especifican. Los argumentos pueden ser omitidos si el programador hace dicha construcción antes de la llamada.

Cuando se devuelve el control a la tarea:

a) CARRY = 0 implica que la transferencia tuvo éxito al menos en forma parcial (ver IRANSFER)

b) CARRY = 1 implica:

1) RO = 0, archivo completo. No cabe más información

2) RO = 1, error de equipo electrónico

3) RO = 2, canal inválido. Esto solo puede suceder si no se solicitó el canal con anterioridad, o si la tarea destruye el número que le fue asignado

c) los argumentos (excepto TRANSFER) no se afectan

Errores: 1) número inválido de argumentos (ensamblado)

2) CARRY = 1 implica que RO contiene el código correspondiente:

RO = 0, desborde

RO = 1, error de equipo

RO = 2, canal inválido

$GUARDA

Función: Guarda el archivo asociado al canal especificado y libera el canal

Llamadas: $GUARDA

$GUARDA CANAL

CANAL es el nombre de la variable que contiene el número de canal que se desea guardar. El argumento puede ser omitido si previamente se hace RO = número de canal.

Errores: C = 1, canal inválido

$SLEE

Función: Lee un número especificado de palabras del canal de transferencia indicado. La tarea es suspendida hasta que se completa la transferencia.

Llamadas: $SLEE

$SLEE CANAL, DATOS, CUENTA, BLOQUE, TRANSFER

CANAL es el nombre de una variable que contiene un número asignado por SO1 para referirse a un archivo especificado en una llamada $BUSCA o $CREA

DATOS es el nombre de un arreglo (buffer) donde se deben poner los datos leídos

CUENTA es el nombre de una variable que contiene el número de palabras que se desean leer

BLOQUE es el nombre de una variable que contiene el número de bloque de 256 palabras (relativo al principio del archivo) de donde se quiere leer

TRANSFER es el nombre de una variable a la que SO1 asigna un valor igual al número de palabras que efectivamente pudo leer (no necesariamente = CUENTA)

Estos argumentos se transfieren a SO1 de la misma forma en que lo hace $ESCRIBE (forma normal para más de un argumento).

Cuando se le devuelve el control a la tarea:

a) CARRY = 0 indica que la transferencia tuvo éxito, al menos parcialmente (hay que consultar IRANSFER para estar seguro de que se leyeron todas las palabras solicitadas)

b) CARRY = 1 implica:

1) RO = 0, se llegó al fin de archivo sin poder leer ni una palabra (archivo vacío)

2) RO = 1, error de tipo electromecánico

3) RO = 2, canal inválido (en $SESCRIBE hay una lista de causas posibles)

c) Los argumentos, excepto DAIOS y IRANSFER, no se afectan.

Errores: 1) Número inválido de argumentos (ensamblado)

2) CARRY = 1 implica que si:

RO = 0, palabras leídas = 0

RO = 1, error electromecánico

RO = 2, canal inválido

$MENSAJE

Función: Imprime, en el teletipo, el texto incluido como argumento entre paréntesis angulares

Llamada: $MENSAJE < texto >

El texto no debe incluir otros paréntesis angulares ni retorno ni avance de carro. La llamada destruye el registro general R0 y no provoca errores.

APENDICE C. OPERACION BAJO SO1 DE 2 TAREAS

A continuación se muestran 3 fragmentos de listado. El primero muestra el código en lenguaje ensamblador de dos tareas, el segundo muestra la puesta en operación de esas tareas en el procesador titular, y el tercero, la entrada en acción del procesador suplente.

La tarea TAREA1 imprime, cada 10 segundos, un mensaje ("REFERENCIA: 10 SEG.") que sirve como referencia o base de tiempo para apreciar la operación del resto del sistema. Regresa el control a SO1 empleando la seudoinstrucción $TERMINA, y es reiniciable.

La tarea TAREA2 lleva un contador interno módulo 8 en el registro general R1, el cual incrementa cada 30 segundos. Este contador es inicializado una sola vez cuando se le entrega el control por la entrada TA2, posteriormente la tarea lo incrementa e imprime su valor en el teleti -

```
 R PIP
*TT:=ESO1T.NEU
;                ESO1T.NEU                    2-SEP-80
;
;EJEMPLO DE DOS TAREAS OPERANDO BAJO SO1
;(SE EMPLEA UNA VERSION CONFIGURADA PARA 4 TAREAS)


        .LIST


        .CSECT  COMUN
TA1.DES:
        .RAD50  /TAREA1/
        .WORD   0,0             ;HORA DE ENTRADA
        .WORD   10.*60.,0       ;INTERARRIBO
        .WORD   1               ;PRIORIDAD
        .WORD   TA1             ;ENTRADA
        .WORD   TA1D            ;DESBORDE
        .WORD   PILA1           ;DIREC. DE PILA
        .BLKW   20              ;PILA
PILA1:

        .CSECT  TAREAS
TA1:    $MENSAJE        <REFERENCIA: 10 SEG.>
        $TERMINA


TA1D:   $MENSAJE        <DESBORDE TAREA#1>
        $TERMINA


        .CSECT  COMUN
TA2.DES:
        .RAD50  /TAREA2/
        .WORD   0,0             ;HORA DE ENTRADA
        .WORD   30.*60.,0       ;INTERARRIBO
        .WORD   2               ;PRIORIDAD
        .WORD   TA2             ;ENTRADA
        .WORD   TA2D            ;DESBORDE
        .WORD   PILA            ;DIREC. DE PILA
        .BLKW   20              ;PILA
PILA:

        .CSECT  TAREAS
TA2:    CLR     R1


TA2D:   INC     R1
        MOV     R1,-(SP)
        BIC     #177770,(SP)
        ADD     #60,(SP)
        .PRINT  SP
        TST     (SP)+
1$:     BR      1$

        .CSECT  COMUN
FINDAT: .END
*
```

po, y se queda esperando (no devuelve el control a SO1) para provocar un desborde intencional. Note que la malla (loop) intencional no afecta la operación de la otra tarea.

Cuando SO1 detecta el desborde, la entrega el control por la entrada IA2D, para que incremente su contador, imprima el valor correspondiente y vuelva a quedar en estado de espera. Esta tarea no es reinicia - ble, si se le reinicia entonces pierde la cuenta que llevaba y empieza de nuevo.

La impresión no se realiza con una seudoinstrucción de SO1, sino con .PRINI que pertenece a RI11. $MENSAJE no es muy adecuada para imprimir valores numéricos porque se supone que las tareas de control en tiempo real actúan sobre dispositivos especiales, y solo eventualmente imprimen mensajes no números. La razón de que estas tareas solo operen sobre el teletipo y sean muy simples responde a las necesidades de exposición, y no deben ser tomadas como ejemplo típico de tarea de control.

El listado que sigue, muestra la forma en que se ensamblan,ligan y ejecutan las tareas bajo SO1.

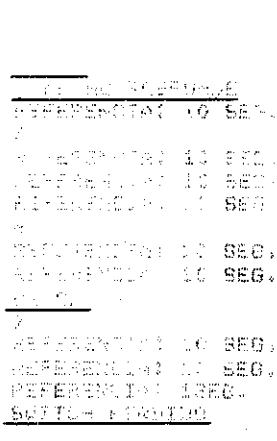
Note que, dado que esta versión de SO1 está configurado para 4 tareas, el programa LINK de RI11 advierte la ausencia de 2 de ellas, lo cual debe ser tomado como un mensaje de advertencia y no como un error. Note también que, dado que solo se ha inicializado el procesador titular, SO1 advierte al operador, mediante el mensaje "LINK NO RESPONDE", que

```
^C

.R MACRO
*MS01T.NEU=ES01C.NEU,ES01T.NEU
ERRORS DETECTED: 0
FREE CORE: 13489. WORDS

^C

.R LINK
*S01.NEU,TT:=MS01T.NEU,MS01LL.NEU,MS01M.NEU,MS01I.NEU
UNDEF GRT-11 LINK    V03-01    LOAD MAP
S01    .NEU

SECTION ADDR    SIZE    ENTRY   ADDR    ENTRY   ADDR    ENTRY   ADDR

  ABS.  000000  001000
COMUN   001000  001200  TA1.DE  002030  TA2.DE  002114  FINDAT  002200
TAREAS  002200  000120
LLAMA   002320  002310  LLA.E1  002320
MTR     004630  003264  MTR.E1  004630  MTR.E2  005064  MTR.E3  005144
INIC    010114  002660  INICIA  010114  TAR.NU  012116  PERDID  012640

UNDEFINED GLOBALS:
TA4.DE
TA3.DE

TRANSFER ADDRESS = 010114
HIGH LIMIT = 012774LBLS

.R SC1.NEU
S01.NEU                    27-MAY-80
SISTEMA OPERATIVO MULTITAREA
INICIA
EN OPERACION
REFERENCIA: 10 SEG.

REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.

REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.
CHECK
LINK NO RESPONDE
REFERENCIA: 10 SEG.

REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.

INTERRUPCION C.A.
EN OPERACION
REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.
```

lo respaldos no estan siendo copiados en la máquina suplente. Por úl - timo, es interesante notar la recuperación de SO1 a la falla de energía eléctrica.

En el último fragmento de listado que se incluye para este ejemplo se muestra, a la izquierda, la operación de la máquina titular y, a la de - recha la del suplente.

En la primera parte, el titular se encuentra operando solo. A continuación se puede observar como entra en acción el suplente y cesan los mensajes del tipo "LINK NO RESPONDE", lo cual indica que los respaldos ya están siendo copiados. En la tercera y última parte se nota la forma en que el suplente detecta una avería en el título y toma posesión del conmutador, continuando la operación a partir de la información contenida en el último respaldo generado. Note que la tarea IAREA2 no pierde completamente la cuenta, pero tampoco puede recordar aquello que no ha si - do respaldado.



```
R RESPAL.NEU
WATCHDOG EN OPERACION
```

```
S01.NEU                          27-MAY-80
SISTEMA OPERATIVO MULTITAREA
CONTINUA
EN OPERACION
LINK NO RESPONDE
7
REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.
0
REFERENCIA: 10 SEG.
REFERENCIA: 10 SEG.
CHECK
LINK NO RESPONDE
REFERENCIA: 10 SEG.
1
```

APENDICE D

Este apêndice contiene dos ejemplos de programas que ejecuta el procesador suplente para vigilar la operación del titular.

El primero (VIGILA.NEU) muestra el código que sería necesario insertar en un programa cualquiera para que, al ejecutarse en el procesador suplente, vigile al titular y tome posesión de la operación en tiempo real, en caso de falla.

Es interesante notar que dicho código puede ser insertado en el sistema operativo que se usa normalmente en el procesador suplente. Una inserción de este tipo puede ser más laboriosa que si se hiciera sobre un programa de aplicación, pero evita la necesidad de hacerla en cada programa de aplicación que se ejecute bajo ese sistema operativo modi

ficado.

Este código se compone de cuatro partes:

1) La primera parte, etiquetada como WAIDOG, imprime el mensaje "WAICHDOG EN OPERACION", hace todos los preparativos para que las interrupciones del conmutador sean atendidas por la subrutina WATHAN, inicia la vigilancia solicitando el conmutador por primera vez. Si lo obtiene, entonces toma posesión mediante la subrutina EQIP y, en caso contrario, salta al cuerpo principal del programa MAIN (la operación del conmutador puede ser consultada en el manual correspondiente {10}).

2) EQIP es la subrutina encargada de tomar posesión en caso de que el procesador titular permita que el conmutador pase a manos del suplente, lo cual se considera como prueba de que el titular no está operando correctamente. Su intervención sólo consta de dos etapas: imprime el mensaje "TOMA POSESION" y transfiere el control a SO1 mediante la seudoinstrucción .CHAIN de RT11.

3) Cada vez que el conmutador interrumpe como respuesta a la solicitud de conexión, la subrutina WATHAN investiga si la petición tuvo éxito, en cuyo caso transfiere el control a EQIP. En caso contrario renueva la solicitud y devuelve el control al programa interrumpido.

3) Los tres segmentos de código descritos funcionan cuando el WAICHDOG-TIMER interrumpe al procedador suplente para notificarle el re

```
;       VIGILA.NEU              V:12-JUN-80

;PROGRAMA QUE PUEDE SER ENSAMBLADO O LIGADO CON CUALQUIER OTRO
;PROGRAMA DE APLICACION, PARA VIGILAR EL COMPORTAMIENTO DEL
;OTRO SISTEMA Y TOMAR SU LUGAR EN CASO NECESARIO.

        .MCALL  ..V2..,.REGDEF,.PRINT,.CHAIN
        .MCALL  .ENTER,.WRITW,.CLOSE
        ..V2..
        .REGDEF

WAIDOG: .PRINT  #ME                             ;MENSAJE DE ENTRADA
        MOV     #WAIHAN,@#270                   ;PREPARA VECTOR DE INT.
        MOV     #340,@#272                      ;CON PRIORIDAD 7
        MOV     #101,@#177420                   ;SOLICITA EL SWITCH
        TSTB    @#177420                        ;CONCEDIDO?
        BMI     EQIP
        JMP     MAIN                            ;NO,ESPERA INTERRUPCION

EQIP:   .PRINT  #MT                             ;SI,MENSAJE DE TOMA
        MOV     ARCHVO,@#500                    ;ENCADENA A SO1.NEU
        MOV     ARCHVO+2,@#502
        MOV     ARCHVO+4,@#504
        MOV     ARCHVO+6,@#506
        .CHAIN

ME:     .ASCIZ  /WATCHDOG EN OPERACION/
MT:     .ASCIZ  /TOMA POSESION/
        .EVEN
ARCHVO: .RAD50  /DK SO1    NEU/
        .EVEN

WAIHAN: MOV     #101,@#177420                   ;SOLICITA DE NUEVO
        TSTB    @#177420                        ;CONCEDIDO?
        BPL     RETURN
        CLR     2(SP)                           ;SI,TOMA POSESION SALTANDO A
        MOV     #EQIP,(SP)                      ;EQIP, VIA "RTI".
RETURN: RTI

MAIN:                                           ;PROGRAMA PRINCIPAL
                                                ;QUE PUEDE REALIZAR
                                                ;CUALQUIER FUNCION
                                                ;

;*****          INSERTO PARA EL WATCHDOG-TIMER                  *****
        MOV     #101,@#177420                   ;INSISTE CON EL SWITCH
        TSTB    @#177420                        ;CONCEDIDO?
        BPL     11$
        JMP     EQIP                            ;SI,TOMA POSESION
11$:
;*****************************************************************

                                                ;CONTINUA EL PROGRAMA
                                                ;PRINCIPAL
                                                ;
        .END    WAIDOG
```

sultado de su petición de obtener el conmutador. Pero si el operador se lo transfiere manualmente, entonces el WAICHDOG-IIMER no interrumpe y, por lo tanto, se hace necesario insertar, en alguna parte del programa principal, el cuarto segmento de código mostrado en este ejemplo.

El segundo ejemplo (RESPAL.NEU) contiene los mismos cuatro segmentos explicados en el anterior, más el código necesario para que el procesador suplente haga copias, en su propio disco magnético, de los respaldos que genera el titular.

Esta adición es factible solamente cuando cada procesador cuenta con un disco propio, y se hace indispensable cuando un procesador no tiene acceso directo al disco del otro.

Los detalles de ambos ejemplos pueden ser consultados en los listados que se incluyen a continuación.

```
;       RESPAL.NEU              V:12-JUN-80

;PROGRAMA QUE PUEDE SER ENSAMBLADO O LIGADO CON CUALQUIER OTRO
;PROGRAMA DE APLICACION.  PARA VIGILAR EL COMPORTAMIENTO DEL
;OTRO SISTEMA , TOMAR SU LUGAR EN CASO NECESARIO, Y COPIAR
;ARCHIVOS DE RESPALDO BAJO CONTROL DEL OTRO PROCESADOR.


        .MCALL  ..V2.. .REGDEF .PRINT .CHAIN
        .MCALL  .ENTER .WRITW .CLOSE
        ..V2..
        .REGDEF


WAIDOG: .PRINT  #ME                             ;MENSAJE DE ENTRADA
        MOV     #WAITHAN,@#270                  ;PREPARA VECTOR DE INT.
        MOV     #340,@#272                      ;CON PRIORIDAD 7
        MOV     #101,@#177420                   ;SOLICITA EL SWITCH
        TSTB    @#177420                        ;CONCEDIDO?
        BMI     EQIP
        JMP     MAIN                            ;NO.ESPERA INTERRUPCION


EQIP:   .PRINT  #MT                             ;SI.MENSAJE DE TOMA
        MOV     ARCHVO,@#500                    ;ENCADENA A SO1.NEU
        MOV     ARCHVO+2,@#502
        MOV     ARCHVO+4,@#504
        MOV     ARCHVO+6,@#506
        .CHAIN


ME:     .ASCIZ  /WATCHDOG EN OPERACION/
MT:     .ASCIZ  /TOMA POSESION/
        .EVEN
ARCHVO: .RAD50  /DK SO1    NEU/
        .EVEN


WAITHAN: MOV    #101,@#177420                   ;SOLICITA DE NUEVO
        TSTB    @#177420                        ;CONCEDIDO?
        BPL     RETURN
        CLR     2(SP)                           ;SI.TOMA POSESION SALIENDO A
        MOV     #EQTP,(SP)                      ;EQIP. VIA "RTI".
RETURN: RTI


MAIN:   CLR     ESTADO
OTRO:   MOV     #4,@#172414                     ;CLEAR STATUS DEL LINK
1$:

;*****          INSERTO PARA EL WATCHDOG-TIMER                  *****
        MOV     #101,@#177420                   ;INSISTE CON EL SWITCH
        TSTB    @#177420                        ;CONCEDIDO?
        BPL     11$
        JMP     EQTP                            ;SI.TOMA POSESION
11$:
;*****************************************************************


        BIT     #4000,@#172414                  ;INP.INI.REQ?
        BEQ     1$
        MOV     #-264.,@#172410                 ;-WORD COUNT
        MOV     #RESCOM,@#172412                ;DIREC.INICIAL
        MOV     #5,@#172414                     ;IN DIR. BLOCK M. GO
2$:     TSTB    @#172414                        ;LISTO?
        BPL     2$
```

```
        ISI     @#172414                ;ERROR?
        BPI     3$
                .PRINT  #EI
                JMP     OTRO
3$:     MOV     RESCOM,R0               ;DECODIFICA EL COMANDO
        ASL     R0
        BGT     4$
                .PRINT  #EC
                JMP     OTRO
4$:     CMP     LIMITE,R0
        BHI     5$
                .PRINT  #EC
                JMP     OTRO
5$:     JMP     @TABLA(R0)

ENTER:  ISI     ESTADO
        BEQ     1$
        .CLOSE  #0
        CLR     ESTADO
1$:     .ENTER  #AREA,#0,#RESNOM,RESBLK
        BCC     ENTFIN
                .PRINT  #EE
                JMP     OTRO
ENTFIN: MOV     #1,ESTADO
        JMP     OTRO

WRITE:  CMP     #1,ESTADO
        BNE     2$
        .WRITW  #AREA,#0,#RESBUF,#256,,RESBLK
        BCC     WRTFIN
2$:             .PRINT  #EW
WRTFIN: JMP     OTRO

CLOSE:  CLR     ESTADO
        .CLOSE  #0
        BCC     CLOFIN
                .PRINT  EO
CLOFIN: JMP     OTRO

TABLA:  .WORD   0,ENTER
        .WORD   WRITE
        .WORD   CLOSE
LIMITE: .WORD   <LIMITE-TABLA>
EI:     .ASCIZ  /ERROR TRANSMISION/
EC:     .ASCIZ  /ERROR COMANDO/
EE:     .ASCIZ  /ERROR ENTER/
EW:     .ASCIZ  /ERROR WRITE/
EO:     .ASCIZ  /ERROR CLOSE/
        .EVEN
AREA:   .BLKW   10
ESTADO: .WORD
RESCOM: .WORD
RESBLK: .WORD
RESNOM: .BLKW   6
RESBUF: .BLKW   256.

        .END    WAITDOG
```

APENDICE E

LISTADO GENERAL DE SO1

```
        .NLIST  COM
        .NLIST
        .TITLE  SO114   SISTEMA OPERATIVO MULTIAREA
        .MCALL  .PRINT .EXIT .LOOKUP .ENTER .DELETE ..V2..
        .MCALL  .WRITC .WRITW .READC .READW .CLOSE .CHAIN
        ..V2..
;       ARCHIVO:        ESO1C.NEU       20-MAY-80

        .CSECT  COMUN
;
        .SBTTL  MACROS DEL SISTEMA
;
        .MACRO REGDEF
        R0=%0
        R1=%1
        R2=%2
        R3=%3
        R4=%4
        R5=%5
        SP=%6
        PC=%7
        .ENDM REGDEF
        REGDEF
        .MACRO IF ARG1 OP ARG2 THEN BEGIN N
        CMP ARG1,ARG2
        .IIF IDN OP ,EQ.        BEQ B'N
        .IIF IDN OP ,NE.        BNE B'N
        .IIF IDN OP ,GT.        BGT B'N
        .IIF IDN OP ,GE.        BGE B'N
        .IIF IDN OP ,LT.        BLT B'N
        .IIF IDN OP ,LE.        BLE B'N
        JMP E'N
B'N:
        .ENDM IF
        ;
        ;
        ;
        .MACRO END N
        BR .+4
E'N:
        .ENDM END
        ;
        ;
        ;
        .MACRO ENDIF N
E'N:
        .ENDM ENDIF
        ;
        ;
        ;
        .MACRO BEGIN N
B'N:
        .ENDM BEGIN
        ;
        ;
        ;
        .MACRO ELSE BEGIN N
        BR .+6
        JMP E'N
        .ENDM ELSE
        ;
        ;
        ;
        .MACRO WHILE ARG1 OP ARG2 DO BEGIN N
WH'N:
        IF ARG1 OP ARG2 THEN BEGIN N
        .ENDM WHILE
        ;
        ;
        ;
        .MACRO ENDWHILE N
        JMP WH'N
E'N:
```

```
        .ENDM ENDWHILE
        ;
        ;
        ;
        .MACRO ESCRIBE, MENSAJE  TAMANO  ?ES  ?ES2
        MOV #MENSAJE,R0
        MOV #TAMANO,R4
ES:     TSTB @#177564              ;ESTA LISTO???
        BPL ES                     ;SI NO ESPERA
        MOVB (R0)+,@#177566
        DEC R4                     ;TERMINO EL MENSAJE???
        BNE ES
ES2:    TSTB @#177564
        BPL ES2
        .ENDM ESCRIBE
        ;
        ;
        ;
        ;    MACROS  <CALL>, <SUBROUTINE>, <RETURN>
        ;
        ;         <CALL>
        .MACRO CALL NAME P1 ARG1 ARG2 ARG3 ARG4 ARG5 P2 ?CAL
        .IF IDN P1,<
           .IF DIF ARG1,<>
              .IIF NB ARG1    MOV #ARG1 @#CAL
              .IF DIF ARG2,<>
                 .IIF NB ARG2    MOV #ARG2,@#CAL+2
                 .IF DIF ARG3,<>
                 .IIF NB ARG3    MOV #ARG3,@#CAL+4
                 .IF DIF ARG4,<>
                    .IIF NB ARG4    MOV #ARG4,@#CAL+6
                    .IF DIF ARG5,<>
                       .IIF NB ARG5    MOV #ARG5,@#CAL+10
                    .ENDC
                 .ENDC
                 .ENDC
           .ENDC
        .ENDC
        JSR %5  NAME
CAL:
        .IF IDN P1,<
        .IF DIF ARG1,<>
                .IF NB ARG1
                .WORD ARG1
                .ENDC
        .IF DIF ARG2,<>
                .IF NB ARG2
                .WORD ARG2
                .ENDC
        .IF DIF ARG3,<>
                .IF NB ARG3
                .WORD ARG3
                .ENDC
        .IF DIF ARG4,<>
                .IF NB ARG4
                .WORD ARG4
                .ENDC
        .IF DIF ARG5,<>
                .IF NB ARG5
                .WORD ARG5
                .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDM CALL
        ;
        ;       <SUBROUTINE>
        ;
        .MACRO SUBROUTINE NAME  P1 ARG1 ARG2,ARG3,ARG4,ARG5 P2
        .IF IDN P1,<
        .IF DIF ARG1,<>
```

```
                .IF NB ARG1
                ARG1=.
                .WORD 0
                .ENDC
        .IF DIF ARG2,]
                .IF NB ARG2
                ARG2=.
                .WORD 0
                .ENDC
        .IF DIF ARG3,]
                .IF NB ARG3
                ARG3=.
                .WORD 0
                .ENDC
        .IF DIF ARG4,]
                .IF NB ARG4
                ARG4=.
                .WORD 0
                .ENDC
        .IF DIF ARG5,]
                .IF NB ARG5
                ARG5=.
                .WORD 0
                .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDC
NAME:
        .IF IDN P1,[
        .IF DIF ARG1,]
                .IF NB ARG1
                MOV @(%5)+,ARG1
                .ENDC
        .IF DIF ARG2,]
                .IF NB ARG2
                MOV @(%5)+,ARG2
                .ENDC
        .IF DIF ARG3,]
                .IF NB ARG3
                MOV @(%5)+,ARG3
                .ENDC
        .IF DIF ARG4,]
                .IF NB ARG4
                MOV @(%5)+,ARG4
                .ENDC
        .IF DIF ARG5,]
                .IF NB ARG5
                MOV @(%5)+,ARG5
                .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDC
        .ENDM SUBROUTINE
        ;
        ;       <RETURN>
        ;
        .MACRO RETURN ARG1,ARG2,ARG3,ARG4,ARG5
        MOV %5,-(%6)
        .IF NB ARG1
          .IF NB ARG2
            .IF NB ARG3
              .IF NB ARG4
                .IF NB ARG5
                MOV ARG5,@-(%5)
                .ENDC
              MOV ARG4,@-(%5)
              .ENDC
            MOV ARG3,@-(%5)
            .ENDC
```

```
            MOV ARG2,@-(%5)
            .ENDC
        MOV ARG1 @-(%5)
        .ENDC
        MOV (%6)+,%5
        RTS %5
        .ENDM RETURN
        ;
        ; DEFINE: RESERVA UNA PALABRA E INICIALIZA CON 0
        ;
        .MACRO DEFINE ARG1,ARG2,ARG3,ARG4,ARG5
            .IRP ARG,<ARG1,ARG2,ARG3,ARG4,ARG5>
                .IF NB ARG
                ARG=.
                .WORD 0
                .ENDC
            .ENDM
        .ENDM DEFINE

         ;DATA: RESERVA UNA PALABRA E INICIALIZA
        ;       CON EL VALOR ESPECIFICADO
        ;
        .MACRO DATA ARG P1 V1,V2,V3 V4 V5 P2
        ARG=.
            .IRP V,<V1,V2,V3 V4,V5>
                .IF NB V
                .IIF DIF V,/    .WORD V
                .ENDC
            .ENDM
        .ENDM DATA
        .MACRO  DIMENSION DV P1 D1 D2 P2
DV:     .REPT   D1 * D2
        .WORD   0
        .ENDM
        .ENDM DIMENSION
                .PAGE


;
;
;       ********************************************
;       *                                          *
;       *                                          *
;       *                MACRO                     *
;       *                TYPE                      *
;       *                                          *
;       *                                          *
;       ********************************************
;
;       ESCRIBE EN LA CONSOLA LA CUERDA DE
;         CARACTERES QUE VIENE COMO ARGUMENTO.
;
;
        .MACRO  TYPE    MSG ?DATA ?PRINT
        BR      PRINT
DATA:   .ASCIZ  /MSG/
        .EVEN
PRINT:  .PRINT  #DATA
        .ENDM
;       ********************************************
;       *                                          *
;       *                                          *
;       *                MACRO                     *
;       *                TYPEN                     *
;       *                                          *
;       *                                          *
;       ********************************************
;
;       ESCRIBE EN LA CONSOLA LA CUERDA DE
;         CARACTERES QUE VIENE COMO ARGUMENTO.
;
;
        .MACRO  TYPEN   MSG ?DATA ?PRINT
        BR      PRINT
DATA:   .ASCII  /MSG/
```

```
        .BYTE   200
        .EVEN
PRINI:  PRINI   #DATA
        .ENDM


        .MACRO  LIMPII  ?L
I:      .TTINR
        BCC     I
        .ENDM   LIMPIT

        .MACRO  PSHDIG
        .TTYIN  -(SP)
        BIC     #177600,(SP)
        SUB     #60,(SP)
        .ENDM   PSHDIG

        .MACRO  POPDIG
        IST     (SP)+
        .ENDM   POPDIG

        .NLIST

        .MACRO  PUSH    A,B,C,D,E,F
        .IF     NB      A
        MOV     A,-(SP)
        .ENDC
        .IF     NB      B
        MOV     B,-(SP)
        .ENDC
        .IF     NB      C
        MOV     C,-(SP)
        .ENDC
        .IF     NB      D
        MOV     D,-(SP)
```

```
        .ENDC
        .IF     NB      E
        MOV     E,-(SP)
        .ENDC
        .IF     NB      F
        MOV     F,-(SP)
        .ENDC
        .ENDM   PUSH

        .MACRO  POP     F,E,D,C,B,A
        .IF     NB      F
        MOV     (SP)+,F
        .ENDC
        .IF     NB      E
        MOV     (SP)+,E
        .ENDC
        .IF     NB      D
        MOV     (SP)+,D
        .ENDC
        .IF     NB      C
        MOV     (SP)+,C
        .ENDC
        .IF     NB      B
        MOV     (SP)+,B
        .ENDC
        .IF     NB      A
        MOV     (SP)+,A
        .ENDC
        .ENDM   POP

        .IIISI

        .MACRO  MOVSIR  SIR,R,?D,?B,?I
        BR      B
D:      .ASCII  /SIR/
        .EVEN
B:      .NCHR   NUMCAR,<SIR>
        MOV     #NUMCAR,-(SP)
        MOV     #D,R0
I:      MOVB    (R0)+,R
        DEC     (SP)
        BGT     I
        TST     (SP)+
        .ENDM   MOVSTR

        .MACRO  AVAREN  R
        MOVB    #15,R
        MOVB    #12,R
        .ENDM   AVAREN

        .MACRO  SUMAIO  DESI,A,B,C,D,E,F
        .IF     NB,DESI
                .IF     NB,A
                .IFF
                CLR     DESI
                .IFI
                MOV     A,DESI
                .IIF    NB,B    ADD     B,DESI
                .IIF    NB,C    ADD     C,DESI
                .IIF    NB,D    ADD     D,DESI
                .IIF    NB,E    ADD     E,DESI
                .IIF    NB,F    ADD     F,DESI
        .ENDC
        .ENDC
        .ENDM   SUMAIO

        .MACRO  SUMA    S,DESI
        SUMAIO  DESI,S
        .ENDM   SUMA
```

```
;       ********************************************
;       *                                          *
;       *                                          *
;       *       M   A   C   R   O                  *
;       *                                          *
;       *       G   E   T   .   V   T              *
;       *                                          *
;       *                                          *
;       ********************************************
;
;OBTIENE LA DIRECCION DEL DESCRIPTOR DEL USUARIO EN EL VECTOR
;    DE TRANSACCIONES.
;ENTRADAS:   USUARIO(VARIABLE GLOBAL)
;            DIRE.NUM
;SALIDAS:    R0=ORIGEN DEL DESCRIPTOR DEL USUARIO.
;DESTRUYE R0
;ANTECESORES:   MTR.E1.  ACTU.VT
        .MACRO  GET.VT
        MOV     USUARIO,R0      ; VARIABLE USUARIO=2**DE USUARIO
        MOV     DIRE.NUM(R0),R0 ; PARA FACILITAR EL ACCESO AL
                                ; DIRECTORIO      DIRE.NUM
;
;
        .ENDM
;FIN DE     GET.VT

;*****************************************************************

        .MACRO  $MENSAJE        M,?D,?I
        BR      I
D:      .ASCIZ  /M/
        .EVEN
I:      MOV     #D,R0
        TRAP    18.
        .ENDM   $MENSAJE

;*****************************************************************
;
;
        .MACRO  $TERMINA
        TRAP    0
        .ENDM   $TERMINA
;
;
;*****************************************************************
;
;
        .MACRO  $ESPERA
        TRAP    1
        .ENDM   $ESPERA
;
;
;*****************************************************************
;
;
        .MACRO  $SALE
        TRAP    2
        .ENDM   $SALE
;
;
;*****************************************************************
;
;
        .MACRO  $ADC CANAL,DESTINO
        .NARG   NUM
        .IIF    NE,NUM,  MOV     CANAL,R0
        TRAP    3
        .IIF    GT,NUM-1,        MOV     R0,DESTINO
        .ENDM   $ADC
;
;
;*****************************************************************
```

```
;
;
        .MACRO  $DAX,VALOR
        .IIF    NB,VALOR,       MOV     VALOR,R0
        IRAP    4
        .ENDM   $DAX
;
;
;*****************************************************************************
;
;
        .MACRO  $DAY,VALOR
        .IIF    NB,VALOR,       MOV     VALOR,R0
        IRAP    5
        .ENDM   $DAY
;
;
;*****************************************************************************
;
;
        .MACRO  $SIRANSMIIE RECEP,CUENTA,APUDAT
        .NARG   NUM
        .IF     EQ,NUM-3
        BR      .+8.
        .WORD   RECEP
        .WORD   CUENTA
        .WORD   APUDAT
        MOV     #.-6,R0
        TRAP    6
        .MEXIT
        .ENDC
        .IF     EQ,NUM
        IRAP    6
        .MEXIT
        .ENDC
        .ERROR  NUM:NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $SIRANSMIIE
;
;
;*****************************************************************************
;
;
        .MACRO  $IRANSMIIE,RECEP,CUENTA,APUDAT
        .NARG   NUM
        .IF     EQ,NUM-3
        BR      .+8.
        .WORD   RECEP
        .WORD   CUENTA
        .WORD   APUDAT
        MOV     #.-6,R0
        TRAP    7
        .MEXIT
        .ENDC
        .IF     EQ,NUM
        IRAP    7
        .MEXIT
        .ENDC
        .ERROR  NUM:NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $IRANSMIIE
;
;
;*****************************************************************************
;
;
        .MACRO  $SRECIBE,IRANS,CUENTA,APUDAT
        .NARG   NUM
        .IF     EQ,NUM-3
        BR      .+8.
        .WORD   IRANS
        .WORD   CUENTA
        .WORD   APUDAT
        MOV     #.-6,R0
        TRAP    8.
```

```
        .MEXIT
        .ENDC
        .IF     EQ NUM
        TRAP    8.
        .MEXIT
        .ENDC
        .ERROR  NUM:NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $SRECIBE
;
;
;*****************************************************************************
;
;
        .MACRO  $RECIBE IRANS,CUENTA,APUDAT
        .NARG   NUM
        .IF     EQ,NUM-3
        BR      .+8.
        .WORD   IRANS
        .WORD   CUENTA
        .WORD   APUDAT
        MOV     #.-6.,R0
        TRAP    9.
        .MEXIT
        .ENDC
        .IF     EQ NUM
        TRAP    9.
        .MEXIT
        .ENDC
        .ERROR  NUM:NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $RECIBE
;
;
;*****************************************************************************
;
;
        .MACRO  $DESBORDE SEIPAL
        .NARG   NUM
        .IF     EQ,NUM-1
        TRAP    10.
        .WORD   SEIPAL
        .MEXIT
        .ENDC
        .ERROR  NUM:NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $DESBORDE
;
;
;*****************************************************************************
;
;
        .MACRO  $SIGUE,SEIPAL
        .NARG   NUM
        .IF     EQ,NUM-1
        TRAP    11.
        .WORD   SEIPAL
        .MEXIT
        .ENDC
        .ERROR  NUM:NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $SIGUE
;
;
;*****************************************************************************
;
;
        .MACRO  $BUSCA,ARCHIV,CANAL
        .NARG   NUM
        .IF     EQ,NUM-2
        BR      .+6.
        .WORD   ARCHIV
        .WORD   CANAL
        MOV     #.-4.,R0
        TRAP    12.
```

```
        .MEXIT
        .ENDC
        .IF     EQ NUM
        IRAP    12.
        .MEXIT
        .ENDC
        .ERROR  NUM; NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $BUSCA
;
;
;*****************************************************************
;
;
        .MACRO  $CREA ARCHIV CANAL BLQUES
        .NARG   NUM
        .IF     EQ NUM-3
        BR      .+8.
        .WORD   ARCHIV
        .WORD   CANAL
        .WORD   BLQUES
        MOV     #.-6 R0
        TRAP    13.
        .MEXIT
        .ENDC
        .IF     EQ NUM
        IRAP    13.
        .MEXIT
        .ENDC
        .ERROR  NUM:NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $CREA
;
;
;*****************************************************************
;
;
        .MACRO  $DESECHA ARCHIV
        .IIF    NB ARCHIV       MOV     #ARCHIV R0
        IRAP    14.
        .ENDM   $DESECHA
;
;
;*****************************************************************
;
;
        .MACRO  $ESCRIBE CANAL DATOS CUENTA BLOQUE
        .NARG   NUM
        .IF     EQ NUM-4
        BR      .+12.
        .WORD   CANAL
        .WORD   DATOS
        .WORD   CUENTA
        .WORD   BLOQUE
        MOV     #.-10. R0
        TRAP    15.
        .MEXIT
        .ENDC
        .IF     EQ NUM
        IRAP    15.
        .MEXIT
        .ENDC
        .ERROR  NUM:NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $ESCRIBE
;
;
;*****************************************************************
;
;
        .MACRO  $GUARDA CANAL
        .IIF    NB CANAL        MOV     CANAL R0
        IRAP    16.
        .ENDM   $GUARDA
;
;
;*****************************************************************
```

```
;
;
        .MACRO  $SLEE CANAL DATOS,CUENTA BLOQUE
        .NARG   NUM
        .IF     EQ,NUM-4
        BR      .+12.
        .WORD   CANAL
        .WORD   DATOS
        .WORD   CUENTA
        .WORD   BLOQUE
        MOV     #.-10., R0
        TRAP    17.
        .MEXIT
        .ENDC
        .IF     EQ,NUM
        IRAP    17.
        .MEXIT
        .ENDC
        .ERROR  NUM; NUMERO INVALIDO DE ARGUMENTOS
        HALT
        .ENDM   $SLEE
;
;
;*****************************************************************
;
        .SBTTL  DEFINICIONES
;
;       ***********************************************
;       *                                             *
;       *                                             *
;       *              V   A   R                      *
;       *                                             *
;       *                                             *
;       ***********************************************


VI.NOMBRE       =0.             ; NOMBRE, DOBLE PALABRA R50
VI.HE           =4              ; HORA DE ENTRADA, DOBLE PALABRA
VI.TI           =8.             ; TIEMPO DE INTERARRIBO, DOBLE PALABRA
VI.PRIORIDAD    =12.            ; PRIORIDAD POR POSICION
VI.START        =14.            ;'DIRECCION DE INICIO NORMAL
VI.RETORNO      =16.            ; DIRECCION DE INICIO POR DESBORDE
VI.PILA         =18.            ; APUNTADOR AL STACK
VI.ESTADO       =20.            ; CALIFICADOR DE ESTADO (VER EDO.*)
VI.DAI          =22.            ;APUNTA A LOS DATOS
VI.LDAI         =24.            ;LONGITUD(PAL.) DE DATOS
VI.PRO          =26.            ;AREA DE INSTRUCCIONES
VI.IPRO         =28             ;LONGITUD(PAL.) DE AREA DE INSTR.
VI.INT          =30             ; CALIFICADOR DE INTERRUPCIONES
VI.SUS          =32.            ; CALIFICADOR DE SUSPENSIONES=(#DE IRAP)*2
VI.R0           =34.            ; DESPLAZAMIENTO PARA SALVAR REGISIROS
VI.R1           =36.            ; GENERALES R0-R5, SP Y PC
VI.R2           =38.            ;  DURANTE UNA INTERRUPCION O
VI.R3           =40.            ;  SUSPENSION.
VI.R4           =42
VI.R5           =44.
VI.SP           =46.
VI.PC           =48.
VI.PSW          =50.
VI.PCD          =52.            ; DESPLAZAMIENTOS PARA GUARDAR
VI.PSD          =54.            ;   PC Y PSW DURANTE DESBORDE.
VI.LINK         =56.            ;LINK PARA COLA DE PRIORIDADES
VI.UTILERIA     =60             ;AREA PARA ARGUMENTOS DE UTILERIAS
VI.LON          =62             ;LONGITUD DE AREA PARA UN USUARIO  EN PALABRAS
VI.LGB  =<VI.LON+2>


EDO.ACT         =1              ; ACTIVO
EDO.INT         =2              ; INTERRUMPIDO
EDO.SUS         =4              ; SUSPENDIDO
EDO.DES         =20             ; DESBORDE
EDO.ELE         =40             ; ELEGIBLE
EDO.HOR         =200            ; EN HORA PARA EJECUTAR
EDO.MEZ         =8200           ; PROGRAMA EN MEZCLA
```

```
SUSOINI         =<EDO.SUS + EDO.INI>
MA.USU          =30000          ; MODO ANIERIOR USUARIO
AC.USU          =140000         ; MODO ACTUAL USUARIO
PSW             =^O177776       ; DIRECCION DEI PSW

UNSWRE  =177420                 ;UNIBUS SWIICH SIAIUS REGISIER
LKSTRE  =172414                 ;LINK SIATUS REGISTER
DA.X    =170412                 ;DIRECCION PUERTO D/A CANAL X (10 BIIS)
DA.Y    =170414                 ;DIRECCION PUERTO D/A CANAL Y (10 BIIS)

MAX.PROGS       =4              ; MAXIMO NUMERO DE PROGRAMAS <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
NUM.USU         =<MAX.PROGS*2>      ; COTA PARA DIRECTORIO DE USUARIOS

CANALES=16.                     ;NUMERO DE CANALES I/O DISP.<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CAN.LIM=<CANALES-1>*2           ;LIMITE DE CANALES DISPONIBLES
LIM.INT =60.*5                  ; LIMITE DE INTERRUPCIONES CONSECUII-
                                ;    VAS AL SISTEMA.
```



```
        .SBTTL  VARIABLES

        .GLOBL  IA1.DES, IA2.DES, IA3.DES, IA4.DES :<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
        .GLOBL  INICIA LLA.E1.TAR.NULA.FINDAT
        .GLOBL  MTR.E1.MTR.E2.MTR.E3


CONI.MIR:       .BLKW   2       ; CONTADOR  DE ENTRADAS A TAR.NULA

REL.SYS:        .BLKW   2       ; RELOJ DEL SMA.(-SIGN   +SIGN.)


USUARIO:        .WORD   ^O177777
                                ; (# DE USUARIO EN OPERACION)*2
ELI.CANDI:      .WORD           ; LISTA DE CANDIDATOS A ELEGIR(BITS)

ELI.CONTA:      .WORD           ; CONTADOR PARA OBTENER PRIORIDADES

VI.BASE:        .WORD   VECI.TRANS      ; DIRECCION BASE DEL VECI DE TRANS.

VECI.TRANS:                             ; VECTOR DE TRANSACCIONES
        .RAD50  /TARNUL/
        .WORD   0 0             ; HORA DE ENTRADA
        .WORD   0.0             ; TIEMPO DE INTERARRIBO
        .WORD   0               ; PRIORIDAD POR POSICION
        .WORD   TAR.NULA        ; DIRECCION DE INICIO
        .WORD   IAR.NULA        ; DIRECCION DE INICIO POR DESBORDE
        .WORD   IAR.NULA        ; APUNTADOR AL STACK DE TAR.NULA
        .=VECI.TRANS+VI.IOB

        .BLKW   <MAX.PROGS*VI.IOW>


DIRE.NUM:
        .WORD   VECI.TRANS
        .REPT   MAX.PROGS
                .WORD   ^O177777
        .ENDM


DIRE.ORI:                       ; DIRECTORIO DE ORIGENES
        .WORD   VECI.TRANS      ;DIREC. DEL DESCRIPTOR DE TAREA
                                ;    NULA, EL CUAL COINCIDE CON EL VECTOR
                                ;    DE TRANSACCIONES YA QUE SUS PARAMETROS SE TRANSF
                                ;    REN DESDE EL ENSAMBLADO.
        .WORD   IA1.DES         ; <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
        .WORD   IA2.DES         ;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
        .WORD   IA3.DES         ;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
        .WORD   IA4.DES         ;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
DIRE.PRIOR:
        .WORD   0               ; PRIORIDAD CERO (TAREA NULA POR DEFLT)
        .BLKW   16.             ; DIRECTORIO PARA 16 PRIORIDADES


DIRE.CAN:               ;DIRECTORIO DE CANALES EN USO
        .REPI <CANALES-1>
        .WORD   -1      ;SE INICIALIZA CADA CANAL CON "-1"
                        ;   CON OBJETO DE MARCARLO DISPONIBLE.
        .ENDR

AD.AREA: .BLKW  16.     ;AREA PARA CONV. A/D 16 CANALES(10 BITS)
TUERR:  .WORD           ;BANDERA PARA DESPRECIAR ERRORES DE I/O
A.C.:   .WORD           ;BANDERA PARA RECONOCER INTERR. DE A.C.
A.C.SP: .WORD           ;STACK POINT. PARA INTERR. DE A.C.
CHECK:  .WORD           ;BANDERA DE CHECKPOINI PENDIENTE
CHEPAL: .WORD           ;# DE PALABRAS DEL CHECKPOINT
CHEBLK: .WORD           ;# DE BLOQUES DE 256 PALABRAS DEL CHECKP.
DMAPEN: .WORD           ;CONTADOR DE ACCESOS DIRECTOS PENDIENTES
AREA:   .BLKW   10      ;AREA PARA PROG.REQ. DEL SISTEMA
CHENOM: .RAD50  /DK CHECK SO1/
        .EVEN
RI.4:   .WORD           ;GUARDA LA DIREC. DE IRAP4
RI.6:   .WORD
RI.10:  .WORD           ;GUARDA LA DIREC. DE IRAP10
RI.12:  .WORD
RI.60:  .WORD           ;GUARDA LA DIREC. DE ATENCION AL TECLADO
RI.62:  .WORD
RI.100: .WORD           ;GUARDA LA DIREC. DE ATENCION AL RELOJ
RI.102: .WORD

;       TERMINA VARM.NEU
```

```
        .LISI
        .SBTIL  MONITOR (SCHEDULER)

        .CSECI  MIR
        .GLOBI  PERDIDO

;       ********************************************
;       *                                          *
;       *                                          *
;       *               M  I  R                    *
;       *                                          *
;       *                                          *
;       ********************************************

;       ARCHIVO:   ESOIM.NEU
;       FECHA:  20-MAY-80


;DISIRIBUYE EL IIEMPO DE PROCESADOR ENIRE LOS
;   DIFERENIES USUARIOS.
;
;ENIRADAS GLOBALES: REL.SYS.  USUARIO  VI.*.
;
;SALIDAS (GLOBALES): REL.SYS.  VI.*  USUARIO.
;
;ANIECESORES: INIERRUPCION DE RELOJ DE IIEMPO REAL(MIR.E1)
;                                        LLAMA(MIR.E2)
;                                        INICIA(MIR.E3)
;
;SUCESORES: GEI.VI  ACIU.VI.  EI I.US.  USUARIOS Y IAREA NULA
;
;
;REQUISIIOS: 1) INICIAR EL DIRECIORIO DEL VECIOR DE IRANSACCIONES   DIRE.NUM
;              CON LAS DIRECCIONES INICIALES DEL VECIOR DE IRANSACCIONES
;               PARA CADA UNO DE LOS USUARIOS.
;            2) INICIALIZAR EL VECTOR DE TRANSACCIONES.
;
;SUBRUIINA ******* MIR.E1 *********


MIR.E1:                 ; ENIRADA POR INIERRUPCION POR RIR
                        ;DEBE ENTRAR CON PRIORIDAD 6 PARA EVIIAR
                        ; QUE SE INTERRUMPA EL MISMO ANTES DE MIR.E2
        ADC     REL.SYS ; INCREMENTA EL RELOJ
        ADC     REL.SYS+2       ; DEL SISTEMA(DOBLE PALABRA)


;*****  INSERIO PARA DEIECIAR PERDIDA MANUAL DEL SWIICH *****
        MOV     #377,IOERR      ;AVISA QUE ES UNA PRUEBA
        TSIB    @#UNSWRE        ;HAY SWITCH?
        ISI     IOERR
        BMI     20$
        TSIB    @#UNSWRE        ;SI HAY.IO IENGO?
        BMI     20$
        JMP     PERDIDO         ;NO LO IENGO
20$:    CLR     IOERR           ;SI LO HAY.LO IENGO.NO MAS PRUEBAS
;*************************************************************


        INC     CONI.MIR+2      ;INC. CONIADOR PARA CHCKPOINI
        CMP     CONI.MIR+2, #3600.      ;UN MINUTO?
        BNE     1$
                CLR     CONI.MIR+2      ;SI.VUELVE A EMPEZAR IA CUENIA
                IST     CHECK           ;CHECK PENDIENTE?
                BEQ     2$
                        TYPE    <FALIA CHECK>
                BR      1$
2$:             MOV     #-1,CHECK
                TYPE    <CHECK>
```

```
1$:     BIT     #MA.USU,@#PSW   ; MODO ANT.=USUARIO?
        BNE     MTR.E4          ; SI. ES PROGRAMA DE USUARIO
                                ;    VE A MTR.E4
                INC     CONT.MIR
                CMP     CONT.MTR,#LIM.INT       ;5 SEGUNDOS?
                BNE     3$
                CLR     CONT.MIR
                TYPE    <MONOPOLIO>
3$:             RTI


MTR.E4: CLR     CONT.MTR
        BIC     #340,@#PSW      ;BAJA LA PRIORIDAD
MTR.E2: MOV     R0,-(SP)        ; PONE R0 EN STACK
        GET.VI                  ; TRAE DIRECCION BASE EN R0 PARA GUARDAR
        MOV     (SP)+,VT.R0(R0) ; LOS REGISTROS DEL USUARIO:
        MOV     R1,VT.R1(R0)
        MOV     R2,VT.R2(R0)
        MOV     R3,VT.R3(R0)
        MOV     R4,VT.R4(R0)
        MOV     R5,VT.R5(R0)
        MFPI    SP
        POP     VT.SP(R0),VT.PC(R0),VT.PSW(R0)
MTR.E3: TST     CHECK           ;HORA DE HACER CHECKPOINT?
        BEQ     3$
                TST     DMAPEN  ;SI.DMA PENDIENTES?
                BGT     3$
                        JSR     PC,CHEGEN;NO.GENERA CHECKPOINT
                        CLR     CHECK   ;CHECK GENERADO
3$:     JSR     PC,ACTU.VT      ; ACTUALIZA EL ESTADO DE LOS USUARIOS.
        JSR     PC,ELI.US       ; ELIGE NUEVO USUARIO
        GET.VI                  ; TRAE DIRECCION BASE DE USUARIO EN R0
        BIT     #EDO.DES,VT.ESTADO(R0)  ; DESBORDE?
        BEQ     MTR.NOD                 ; SALTA SI NO HAY DESBORDE
        SUMA    <REL.SYS,VT.TI(R0)> VT.HE(R0)
        BCC     1$
                SUMA    <#1,REL.SYS+2,VT.TI+2(R0)>,VT.HE+2(R0)
        BR      2$
1$:             SUMA    <REL.SYS+2,VT.TI+2(R0)>,VT.HE+2(R0)
2$:     BIC     #EDO.DES,VT.ESTADO(R0)  ; APAGA EL BIT DE DESBORDE
        MOV     VT.PC(R0),VT.PCD(R0)    ; GUARDA PC Y PSW EN LUGAR
        MOV     VT.PSW(R0),VT.PSD(R0)   ;ESPECIAL DENTRO DEL V.T.
        PUSH    VT.PSW(R0),VT.RETORNO(R0),VT.SP(R0)
        MTPI    SP
        MOV     VT.R5(R0),R5
        MOV     VT.R4(R0),R4
        MOV     VT.R3(R0),R3
        MOV     VT.R2(R0),R2
        MOV     VT.R1(R0),R1
        MOV     VT.R0(R0),R0
        RTI


MTR.NOD:
        BIT     #EDO.ACT,VT.ESTADO(R0)  ; ACTIVO?
        BNE     MTR.ACT                 ; SI  SALTA
        ADD     VT.TI(R0),VT.HE(R0)
        ADC     VT.HE+2(R0)
        ADD     VT.TI+2(R0),VT.HE+2(R0)
        BIS     #EDO.ACT,VT.ESTADO(R0)  ;ACTIVA
        PUSH    #AC.USU,VT.START(R0),VT.PILA(R0)
        MTPI    SP
        MOV     VT.R5(R0),R5
        MOV     VT.R4(R0),R4
        MOV     VT.R3(R0),R3
        MOV     VT.R2(R0),R2
        MOV     VT.R1(R0),R1
        MOV     VT.R0(R0),R0
        RTI
```

```
MTR.ACT:                ; RESTAURA EL ESTADO DEL USUARIO
        PUSH    VT.PSW(R0),VT.PC(R0),VT.SP(R0)
        MTPI    SP
        MOV     VT.R5(R0),R5
        MOV     VT.R4(R0),R4
        MOV     VT.R3(R0),R3
        MOV     VT.R2(R0),R2
        MOV     VT.R1(R0),R1
        MOV     VT.R0(R0),R0
        RTI

CHEGEN: .ENTER  #AREA,#CANALES-1,#CHENOM,CHEBLK
        BCS     1$
                .WRITW  #AREA,#CANALES-1,#1000,CHEPAL,#0
                BCS     1$
                        .CLOSE  #CANALES-1
                        BCS     1$
                                MOV     #CHENOM,R1
                                JSR     PC,RESPALDA
                                BCC     2$
                                TYPE    <ERROR RESPALDO>
2$:                             RTS     PC
1$:     TYPE    <ERROR CHECK>
        RTS     PC


RESPALDA:                                       ;RESPALDA EN LA OTRA COMPUTADORA
                                                ;(EN DISCO) EL ARCHIVO CUYO NOMBRE
                                                ;APUNTA EL REGISTRO R1
        .LOOKUP #AREA,#CANALES-1,R1             ;BUSCA EL ARCHIVO A RESPALDAR
        BCS     RESFIN                          ;ERROR?
        PUSH    R0                              ;NO,CONTINUA,PILA=#BLOQUES
        MOV     #1,RESCOM                       ;COMANDO PARA LA OTRA COMP.=ENTER
        MOV     R0,RESBLK                       ;#BLOQUES
        MOV     (R1),RESNOM                     ;NOMBRE DEL ARCHIVO
        MOV     2(R1),RESNOM+2
        MOV     4(R1),RESNOM+4
        MOV     6(R1),RESNOM+6
        JSR     PC,RESEND                       ;TRANSMITE COMANDO ENTER
        BCS     RESERR                          ;ERROR?
        MOV     #2,RESCOM                       ;COMANDO=WRITE
        CLR     RESBLK                          ;A PARTIR DEL BLOQUE 0
        WHILE RESBLK .LT. (SP) DO BEGIN R1
                .READW #AREA,#CANALES-1,#RESBUF,#256.,RESBLK
                BCS     RESERR
                JSR     PC,RESEND
                BCS     RESERR
                INC     RESBLK
        ENDWHILE R1
        MOV     #3,RESCOM                       ;COMANDO=CLOSE
        JSR     PC,RESEND
RESERR: POP     R0
RESFIN: .CLOSE  #CANALES-1
        RTS     PC
RESCOM: .WORD
RESBLK: .WORD
RESNOM: .BLKW   6
RESBUF: .BLKW   256.


RESEND:                                         ;TRANSMITE VIA "LINK"
                                                ;# PAL. DE COMANDO Y 256 DE DATOS
                                                ;AVISA A EMRGEN QUE ES PRUEBA
        MOV     #377,IOERR
        CLR     @#172414
        IF IOERR .LT. #0 THEN BEGIN R2
                CLC
                RTS     PC
        ENDIF R2
        MOV     #-264.,@#172410                 ;-WORD COUNT
        MOV     #RESCOM,@#172412                ;DIREC. INICIAL
        MOV     #11.@#172414                    ;OUT REQ.,OUT DIR.,BLOCK M  GO
        MOV     #-1,R0                          ;ESPERA LA RESPUESTA
        WHILE R0 .NE. #0 DO BEGIN R3
                TSTB    @#172414                ;LISTO?
                BMI     RESIGE
                DEC     R0
        ENDWHILE R3
```

```
        TYPE    <LINK NO RESPONDE>
        SEC
        RTS     PC
RESIGE: IF @#172414 .GE. #0 THEN BEGIN R4
                CLC
                RTS     PC
        ENDIF R4
        TYPE    <ERROR LINK>
        SEC
        RTS     PC


;       ***************************************
;       *                                     *
;       *       S U B R U I I N A             *
;       *                                     *
;       *             A C I U . V I           *
;       *                                     *
;       ***************************************


;       DIAGRAMA  DE  FLUJO :   25-NOV-77 /NEUMAN

;ACTUALIZA VI Y DETECTA DESBORDES DE USUARIO
;ENTRADAS:      USUARIO  (IO DESTRUYE).
;       VT.*.
;       REI.SYS
;SALIDAS:       USUARIO=NUM.USU+2.
;       VT.*.
;       ELI.CANDI
;ANTECESORES:   MIR.EI
ACTU.VT:
;NODO 1
        MOV     R1,-(SP)
        MOV     R0,-(SP)
        CLR     ELI.CANDI
        MOV     #2.USUARIO
WHILE USUARIO .LE. #NUM.USU. DO BEGIN 1$
        GET.VT                  ;  IRAE DIR. USU. EN VT
        MOV     VT.ESTADO(R0),R1 ; GUARDA SU EDO EN R1
;NODO 2
        BIT     #EDO.MEZ R1     ; EN MEZCLA?
        BEQ     ACT.17          ; NO, SALTA
        BIT     #EDO.DES R1     ; DESB. DETECTADO?
        BEQ     ACT.3           ; NO, SALTA.
;               SE MARCA COMO CANDIDATO AL USUARIO
                BIS     VT.PRIORIDAD(R0).ELI.CANDI
                BR      ACT.17

;NODO 3
ACT.3:
        IF REI.SYS+2 .LT. VT.HE+2(R0) THEN BEGIN 2$
                BR      ACT.4           ; TODAVIA NO LE TOCA.
                END 2$
        ELSE BEGIN 3$                   ; REI.SYS+2=>VT.HE+2
                IF REI.SYS+2 .NE. VT.HE+2(R0) THEN BEGIN 4$
                        BR ACT.11       ; REI.SYS+2>VT.HE+2. YA LE TOCA.
                        END 4$
                ELSE BEGIN 5$           ; REI.SYS+2=VT.HE+2, SIGUE COMPARANDO
                        CMP     REI.SYS,VT.HE(R0)
                                BLO     ACT.4   ;REI.SYS<VT.HE, TODAVIA NO.
                                BR ACT.11; REI.SYS=>VT.HE  YA LE TOCA ENTRAR.
                    ENDIF 5$
                ENDIF 3$
        ACT.17:

        MOV     R1,VT.ESTADO(R0)
        ADD     #2.USUARIO
ENDWHILE 1$

;
;
;
```

```
;       SECUENCIA
        MOV     (SP)+,R0
        MOV     (SP)+,R1
        RTS     PC
;
;
;
;NODO 4 EL USUARIO NO ESTA EN HORA
ACT.4:
        BIC     #EDO.HOR,R1     ; SE MARCA "NO EN HORA"
;NODO 5 SE CHECA SI ESTA ACTIVO
        BIT     #EDO.ACT,R1     ; USUARIO ACTIVO?
        BNE     ACT.7           ; SI, SALIR
;
;NODO 6                         USUARIO NO ACTIVO
;                               SE MARCA NO ELEGIBLE
                BIC     #EDO.ELE,R1
                BR      ACT.17
;
;NODO 7                         USUARIO ACTIVO:
;                               SE CHECA SUSP. O INT.
ACT.7:  BIT     #SUSOINT,R1     ; NI SUS NI INT?
        BEQ     ACT.9           ; SI, SALTA
;
;NODO 8          USUARIO SUSPENDIDO O INTERRUMPIDO
                BIC     #EDO.ELE,R1
                BR      ACT.17
;
;
;NODO 9 USUARIO NI INTE. NI SUSP.
;       SE MARCA COMO CANDIDATO AL USUARIO
ACT.9:
        BIS     VT.PRIORIDAD(R0),ELI.CANDI
        BIS     #EDO.ELE,R1
        BR      ACT.17
;
;
;
;NODO 11
;       USUARIO EN HORA
;       SE MARCA EN HORA
ACT.11:
        BIS     #EDO.HOR,R1
;NODO 12
;       SE ELIGE AL USUARIO
        BIS     VT.PRIORIDAD(R0),ELI.CANDI
;NODO 13
;       SE MARCA ELEGIBLE
        BIS     #EDO.ELE,R1
        BIT     #EDO.ACT,R1     ;USUARIO ACTIVO?
        BEQ     ACT.17          ; NO SALIR
;
;NODO 15         EL USUARIO ESTA ACTIVO
;               SE MARCA DESBORDE DETECTADO:
                BIS     #EDO.DES,R1
                BR      ACT.17
;
;
;
;       TERMINA     ACTU.VT
;       ***********************************
;       *                                 *
;       *           E  I  I               *
;       *                                 *
;       ***********************************



;       SI SE HIZO LAS SUBSTITUCIONES       ELI.US
;                                           PRIORIDAD
;                                           ELIGE
```

```
;
;SUBRUTINA     ******   ELI.US     *****
;
;ELIGE USUARIO PARA CONCEDERLE EL PROCESADOR
;
;ENTRADAS:      ELI.CANDI.
;               DIRE.PRIOR.
;               ELI.CONTA
;
;SALIDAS:       USUARIO=NUEVO USUARIO.
;               ELI.CONTA.
;ANTECESORES:   MTR.*
;SUCESORES:     PRIORIDAD Y ELIGE
;
;
        .RADIX
ELI.US:
        MOV     R0,-(SP)        ; GUARDA R0
        JSR     PC,PRIORIDAD    ; OBTEN PRIORIDAD
                                ; R0 =  PRIORIDAD
        JSR     PC,ELIGE        ; ELIGE USUARIO
                ; USUARIO=NUEVO USUARIO
        MOV     (SP)+,R0        ; RESTAURA R0
        RTS     PC      ; REGRESA
;
;FIN DE ELI.US/MTR.E1
;
;
;
;
;SUBRUTINA   ****  PRIORIDAD  ****(LOCAL DE  ELI.US)
;
;OBTIENE EL # DE PRIORIDAD DEL NUEVO USUARIO POR MEDIO
;    DEL METODO DEL CONTADOR
;ENTRADAS:      ELI.CANDI.
;               ELI.CONTA
;SALIDAS:       R0=ELI.PRIOR.
;               ELI.CONTA
;ANTECESORES:   ELI.US(SUBRUTINA LOCAL)
;
PRIORIDAD:
        ELI.TM=R1       ;DEFINICIONES PARA APEGARSE
        ELI.PRIOR=R0    ; AL DIAGRAMA DE FLUGO.
;
;NODO1  DEL DIAGRAMA
        MOV     R1,-(SP)
        TST     ELI.CANDI       ; HAY CANDIDATOS?
        BNE     PRI.3           ; SI.  SALTA.
;NODO 2         NO HAY CANDIDATOS
                CLR     ELI.PRIOR
                BR      PRI.7
;NODO 3 SI HAY CANDIDATOS. DETERMINA PRIORIDAD NOMINAL:
PRI.3:
        MOV     ELI.CONTA,R0    ; TRAE CONTADOR
        INC     ELI.CONTA
        MOV     ELI.CONTA,ELI.TM ; ELI.TM=CONTADOR ACTUALIZADO.
        XOR     R0,ELI.TM       ; ELI.TM=LOS QUE CAMBIARON.
        MOV     ELI.TM,R0       ; R0=LOS QUE CAMBIARON.
        ASR     R0              ; R0=LOS QUE CAMBIARON-EL NOMINAL.
        XOR     R0,ELI.TM       ; ELI.TM=EL NOMINAL.
        BIT     ELI.TM,ELI.CANDI; EL USUARIO DE PRIORIDAD NOMINAL
                                ;   ES CANDIDATO?
        BNE     PRI.6           ; SI.  SALTA.
;NODO 5         LA PRIORIDAD NOMINAL NO ES CANDIDATO
                MOV     ELI.CANDI,ELI.TM
;NODO 6
PRI.6:  MOV     #^D16,ELI.PRIOR


;       SE DETERMINA LA PRIORIDAD NUMERICA FINAL:
PRI.65: TSI     ELI.TM
        BMI     PRI.7
        ASL     ELI.TM
        DEC     ELI.PRIOR
```

```
        BR      FEI.85
;NODO 7 TERMINA
PRI.7:
        MOV     (SP)+ R1
        RTS     PC
;FIN DE LA SUBRUTINA PRIORIDAD, SIGUEN SUBRUTINAS
;     ASOCIADAS A ELI.USUARIO.R1
;
;
;
;
;
;SUBRUTINA     *** ELIGE ***     (LOCAL DE ELI.US)
;
;ELIGE EL USUARIO CON LA PRIORIDAD QUE HAY EN R0
;ENTRE TAREAS DE **DIFERENTE PRIORIDAD**
;ENTRADAS:      DIRE.PRIOR
;       ELI.PRIOR=R0
;SALIDAS:       USUARIO
;ANTECESORES:   ELI.US   (SUBRUTINA LOCAL)
ELIGE:
        ASL     R0
        MOV     DIRE.PRIOR(R0),USUARIO
        RTS     PC
;FIN DE ELIGE


        .END
```

```
        .LIST
        .SBTTL  LLAMADAS          (INTERRUPCIONES PROGRAMADAS)
        .GSECT  LLAMA

;       ********************************************
;       *                                          *
;       *       L     L     A     M     A          *
;       *                                          *
;       ********************************************

;       ARCHIVO:        ESOLLL.NEU

;       FECHA:          22-MAY-80

LLA.E1:
        MOV     R0,-(SP)                ;GUARDA R0
        MOV     R1,-(SP)                ;GUARDA R1
        MOV     4(SP),R1                ;TRAE DIREC.DE RETORNO
        MOVB    -2(R1),R1               ;TRAE NUMERO DE TRAP
        ASL     R1                      ;COMPRUEBA VALIDEZ
        CMP     R1,LLA.LIM
        BIT     LLA.CON                 ;NUMERO VALIDO CONTINUA
LLA.ER:
        .PRINT  #LLA.M1                 ;INVALIDO SE IMPRIME
        MOV     #4,R1                   ;Y SE SACA DE MEZCLA
LLA.CON:
        GET.VI                          ;VECTOR DE TRANS. EN R0
        JMP     @LLA.TAB(R1)            ;GOTO CALCULADO
LLA.TAB:
        .WORD   .TERMINA                ;TRAP 0
        .WORD   .ESPERA                 ;TRAP 1
        .WORD   .SALE                   ;TRAP 2
        .WORD   .ADC                    ;TRAP 3
        .WORD   .DAX                    ;TRAP 4
        .WORD   .DAY                    ;TRAP 5
        .WORD   .STRANSMITE             ;TRAP 6
        .WORD   .TRANSMITE              ;TRAP 7
        .WORD   .SRECIBE                ;TRAP 8
        .WORD   .RECIBE                 ;TRAP 9
        .WORD   .DESBORDE               ;TRAP 10
        .WORD   .RETORNO                ;TRAP 11
        .WORD   .BUSCA
        .WORD   .CREA
        .WORD   .DESECHA
        .WORD   .SESCRIBE
        .WORD   .GUARDA
        .WORD   .SLEE
        .WORD   .MENSAJE                ;TRAP 18.
LLA.LIM:
        .WORD   <LLA.LIM-LLA.TAB>       ;# DE TRAPS * 2
LLA.M1:
        .ASCIZ  /UTILERIA INVALIDA SE SUSPENDE AL USUARIO/
        .EVEN
.SALE:
        BIC     #EDO.MEZ,VI.ESTADO(R0)  ;SACA DE MEZCLA
.TERMINA:
        BIC     #EDO.ACT,VI.ESTADO(R0)  ;DESACTIVA
.ESPERA:
        MOV     (SP)+,R1                ;RESTAURA R0
        MOV     (SP)+,R0                ;RESTAURA R0
        JMP     MTR.E2                  ;SALIA AL MONITOR
.MENSAJE:
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        .PRINT
        JMP     MTR.E2
.ADC:
        MOV     (SP)+,R1                ;RESTAURA R1
        MOV     (SP)+,R0                ;RESTAURA R0
        BIT     #177760,R0              ;CANAL VALIDO?
        BEQ     1$                      ;SI.CONTINUA
        BIS     #1,2(SP)                ;PRENDE EL CARRY
        RTI                             ;Y REGRESA
```

```
1$:     ASL     R0                      ;INDICE=#CANAL*2
        MOV     AD.AREA,R0),R0          ;ACCESA TABLA CON EL INDICE
        BIC     #1,2(SP)                ;APAGA EL CARRY
        JMP     MIR.E2                  ;Y REGRESA
.DAX:
        MOV     (SP)+,R1                ;RESTAURA R1
        MOV     (SP)+,R0                ;RESTAURA R0
        BIT     #176000,R0              ;VALOR VALIDO?
        BEQ     2$                      ;SI,CONTINUA
        BIS     #1,2(SP)                ;PRENDE EL CARRY
        JMP     MIR.E2                  ;Y REGRESA
2$:     MOV     R0,@#DA.X               ;TRANSFIERE EL VALOR
        BIC     #1,2(SP)                ;APAGA EL CARRY
        JMP     MIR.E2                  ;Y REGRESA
.DAY:
        MOV     (SP)+,R1                ;RESTAURA R1
        MOV     (SP)+,R0                ;RESTAURA R0
        BIT     #176000,R0              ;VALOR VALIDO?
        BEQ     3$                      ;SI,CONTINUA
        BIS     #1,2(SP)                ;PRENDE EL CARRY
        JMP     MIR.E2                  ;Y REGRESA
3$:     MOV     R0,@#DA.Y               ;TRANSFIERE EL VALOR
        BIC     #1,2(SP)                ;APAGA EL CARRY
        JMP     MIR.E2                  ;Y REGRESA
.SIRANSMITE:
        BIS     #EDO.SUS,VI.ESTADO(R0)  ;SUSPENDE LA TAREA
        MOV     R1,VI.SUS(R0)           ;CAUSA=NUMERO DE TRAP*2
.TRANSMITE:
        BIC     #1,6(SP)                ;CLEAR CARRY
        MOV     R2,-(SP)                ;GUARDA R2
        MOV     R3,-(SP)                ;GUARDA R3
        MOV     6(SP),R2                ;R2=AREA DE TRANSMISOR
        MOVB    @(R2)+,R1               ;R1=# DE RECEPTOR
        ASL     R1                      ;R1=INDICE DE RECEPTOR
        CMP     #NUM.USU,R1             ;RECEPTOR VALIDO?
        BPL     REC.VAL                 ;SI,CONTINUA
        BIC     #EDO.SUS,VI.ESTADO(R0)  ;NO,LEVANTA SUSPENSION
        BIS     #1,10.(SP)              ;PRENDE EL CARRY
IRAESP: MOV     (SP)+,R3                ;RESTAURA REGITROS...
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        JMP     MIR.E2                  ;Y SALIA AL MONITOR
REC.VAL:MOV     DIRE.NUM(R1),R1         ;R1=DIRECC.DEL RECEPTOR EN
                                        ;EL VECTOR DE TRANSACCIONES
        BIT     #EDO.SUS,VI.ESTADO(R1)  ;RECEPTOR SUSPENDIDO?
        BEQ     TRAESP                  ;NO,ESPERA
        CMP     #8.*2,VI.SUS            ;ESPERA TRANSMISION?
        BNE     TRAESP                  ;NO ESPERA,ESPERA TU
        BIC     #EDO.SUS,VI.ESTADO(R0)  ;LEVANTA SUSPENSION AL IRANS.
        BIC     #EDO.SUS,VI.ESTADO(R1)  ;LEVANTA SUSPENSION AL RECEP.
        MOV     USUARIO,R0              ;R0=# DE TRANSMISOR*2
        ASR     R0
        MOV     VT.R0(R1),R3            ;R3=AREA DE RECEPTOR
        MOVB    R0,@(R3)                ;ANOTA PROCEDENCIA DE TRANSMISION
        MOV     @4(R2),R0               ;TRAE APUDAT
        MOV     @4(R3),R1               ;TRAE APUDAT R
4$:     MOVB    (R0)+,(R1)+             ;TRANSMITE
        INC     @2(R2)                  ;ACTUALIZA CUENTA DE TRANS.
        BIT     3$                      ;CUENTA AGOTADA(=0)?
        INC     @2(R3)                  ;SI,ACTUALIZA CUENTA DE RECEP.
        BR      5$                      ;
3$:     INC     @2(R3)                  ;NO,ACTUALIZA CUENTA DE RECEP.
        BLT     4$                      ;CUENTA DE RECEP,AGOTADA?
5$:     MOV     R0,@4(R2)               ;ACTUALIZA APUDAT
        MOV     R1,@4(R3)               ;ACTUALIZA APUDAT R
        BR      TRAESP                  ;SALTA AL MONITOR
.SRECIBE:
        BIS     #EDO.SUS,VI.ESTADO(R0)  ;SUSPENDE LA TAREA
        MOV     R1,VI.SUS(R0)           ;CAUSA=NUMERO DE TRAP*2
.RECIBE:
        BIC     #1,6(SP)                ;CLEAR CARRY
        MOV     R2,-(SP)                ;GUARDA R2
        MOV     R3,-(SP)                ;GUARDA R3
        MOV     6(SP),R2                ;R2=AREA DE RECEPTOR
```

```
        MOVB    @(R2),R1                ;R1=# DE TRANSMISOR
        ASL     R1                      ;R1=INDICE DE TRANSMISOR
        CMP     #NUM.USU,R1             ;TRANSMISOR VALIDO?
        BPL     TRANS.VAL               ;SI CONTINUA
        BIC     #EDO.SUS,VT.ESTADO(R0)  ;NO.LEVANTA SUSPENSION
        BIS     #1,10.(SP)              ;PRENDE EL CARRY
RECESP: MOV     (SP)+,R3                ;RESTAURA REGITROS...
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        JMP     MTR.E2                  ;Y SALIA AL MONITOR
TRANS.VAL:
        MOV     DIRE.NUM(R1),R1         ;R1=DIRECC.DEL TRANSMISOR EN
                                        ;EL VECTOR DE TRANSACCIONES
        BIT     #EDO.SUS,VT.ESTADO(R1)  ;TRANSMISOR SUSPENDIDO?
        BEQ     RECESP                  ;NO.ESPERA
        CMP     #6.*2,VT.SUS            ;ESPERA RECEPCION?
        BNE     RECESP                  ;NO ESPERA.ESPERA IU
        MOV     VT.R0(R1),R3            ;R3=AREA DE TRANSMISOR
        MOV     USUARIO,-(SP)           ;(SP)=# DE TAREA*2
        ASR     (SP)                    ;(SP)=# DE TAREA
        CMP     (SP)+,@(R3)             ;# DE TAREA=DESTINO DE TRANS.?
        BNE     RECESP                  ;NO.ESPERA
        BIC     #EDO.SUS,VT.ESTADO(R0)  ;LEVANTA SUSPENSION AL TRANS.
        BIC     #EDO.SUS,VT.ESTADO(R1)  ;LEVANTA SUSPENSION AL RECEP.
        MOV     @4(R3),R1               ;TRAE APUDAT DE TRANSM.
        MOV     @4(R2),R0               ;TRAE APUDAT DE RECEP.
4$:     MOVB    (R1)+,(R0)+             ;TRANSMITE
        INC     @2(R2)                  ;ACTUALIZA CUENTA DE RECEP.
        BLT     3$                      ;CUENTA AGOTADA(=0)?
        INC     @2(R3)                  ;SI.ACTUALIZA CUENTA DE TRANS.
        BR      5$                      ;Y SALIA AL MONITOR
3$:     INC     @2(R3)                  ;NO.ACTUALIZA CUENTA DE TRANS.
        BLT     4$                      ;CUENTA DE TRANS.AGOTADA?
5$:     MOV     R1,@4(R3)               ;ACTUALIZA APUDAT DE TRANS.
        MOV     R0,@4(R2)               ;ACTUALIZA APUDAT DE RECEP.
        BR      RECESP                  ;SALTA AL MONITOR
.DESBORDE:
        MOV     #-6,R1                  ;R1=CUENTA DE TRANSFERENCIAS
        ADD     #VT.R0,R0               ;R0=DIREC.DE REGISTROS DEL DESBORDE
        MOV     @4(SP),-(SP)            ;PILA=DIRECCION DE 6 PALABRAS
        ADD     #2,6(SP)                ;ACTUALIZA DIR.DE RETORNO A LA TAREA
1$:     MOV     (R0)+,@(SP)             ;GUARDA REGISTROS A PARTIR DE LA
                                        ;DIRECCION DE 6 PALABRAS
        ADD     #2 (SP)         ;ACTUALIZA DIREC. PARA GUARDAR
        INC     R1              ;INCREMENTA CUENTA DE TRANSFERENCIA
        BMI     1$              ;FIN?
        TST     (SP)+           ;SI LIMPIA LA PILA
        MOV     (SP)+,R1                ;RESTAURA R1 Y R0
        MOV     (SP)+,R0
        RTI
.RETORNO:
        MOV     2+(SP),R1               ;R1=DIRECCION DE 6 PALABRAS
        TST     (SP)+           ;DESECHA R1 Y R0 VIEJOS
        ISI     (SP)+
        MOV     VT.PCD(R0),(SP) ;SUSTITUYE DIREC.DE RETORNO
        MOV     VT.PSD(R0),2(SP)        ;SUSTITUYE PSW ANTES DEL DESBORDE
        MOV     R1,R0           ;R0=DIREC.DE REGISTROS ORIGINALES
        MOV     12.(R0),R5      ;SUSTITUYE REGISTROS ANTES DEL DESBORDE
        MOV     10.(R0),R4
        MOV     8.(R0),R3
        MOV     6(R0),R2
        MOV     4(R0),R1
        MOV     2(R0),R0
        JMP     MTR.E2
.BUSCA:
        JSR     PC,GET.CANAL            ;PIDE CANAL DISP.
                                        ;  EN R1
        BCC     1$
        JMP     ARCH.0                  ;SALIA SI HUBO FRACASO
1$:     MOV     R3,-(SP)                ;PUSH R3
        MOV     4(SP),R3                ;APUNTADOR AL AREA
        MOVB    R1,@2(R3)               ;CANAL = R1
        MOV     #1*^O40+VT.UTILERIA(R0);CLAVE
        MOVB    R1,VT.UTILERIA(R0)      ;CANAL PARA RT11
```

```
        MOV     @(R3)+,VI.UTILERIA+2(R0) ;ARCHIV
        MOV     (SP)+,R3                 ;POP R3
        CLR     VT.UTILERIA+4(R0)        ;CERO
        ADD     #VI.UTILERIA,R0          ;APUNTADOR A VI.UTILERIA
        .LOOKUP                          ;REQUEST
        BCC     2$
        JMP     ARCH.3                   ;ERROR?,SALIA
2$:     ASL     R1                       ;R1=R1*2
        MOV     USUARIO.DIRE.CAN(R1)     ;COLOCA USUARIO
        JMP     ARCH.4


.CREA:  JSR     PC,GET.CANAL             ;PIDE CANAL DISPONIBLE
                                         ; EN R1
        BCC     1$
        JMP     ARCH.0                   ;SALIA SI NO HAY DISPON.
1$:     MOV     R3,-(SP)                 ;PUSH R3
        MOV     R4,-(SP)                 ;PUSH R4
        MOV     6(SP),R3                 ;APUNTADOR AL AREA
        MOVB    R1,@2(R3)                ;CANAL = R1
        MOV     #2*^O400,VI.UTILERIA(R0);CLAVE
        MOVB    R1,VT.UTILERIA(R0)       ;CANAL
        MOV     @(R3)+,VI.UTILERIA+2(R0) ;ARCHIV
        MOV     @4(R3),R4                ;BLQUES
        TST     R4                       ;ES POSITIVO BLQUES?
        BPL     NO.CERO                  ;SI,SALIA
        MOV     #1,R4                    ;COLOCA UNO
NO.CERO:
        MOV     R4,VT.UTILERIA+4(R0)     ;BLOQUE > 0
        MOV     (SP)+,R4                 ;POP R4
        MOV     (SP)+,R3                 ;POP R3
        CLR     VT.UTILERIA+6(R0)        ;CERO
        ADD     #VI.UTILERIA,R0          ;APUNTADOR A VI.UTILERIA
        .ENTER                           ;REQUEST
        BCC     1$                       ;SKIP SI NO HAY ERROR
        JMP     ARCH.3                   ;ERROR?,SALIA
1$:     ASL     R1
        MOV     USUARIO.DIRE.CAN(R1)     ;PON DUENO DEL CANAL
        JMP     ARCH.4


.DESECHA:
        JSR     PC,GET.CANAL             ;PIDE CANAL DISPONIBLE
                                         ;   EN R1
        BCC     1$
        JMP     ARCH.0                   ;SALIA SI NO HAY DISPON.
1$:     MOV     R3,-(SP)                 ;PUSH R3
        MOV     4(SP),R3                 ;APUNTADOR AL AREA
        CLR     VT.UTILERIA(R0)          ;PON CLAVE
        MOVB    R1,VT.UTILERIA(R0)       ;PON CANAL
        MOV     @(R3)+,VI.UTILERIA+2(R0) ;PON ARCHIV
        MOV     (SP)+,R3                 ;POP R3
        CLR     VT.UTILERIA+4(R0)        ;CERO
        ADD     #VI.UTILERIA,R0          ;APUNTADOR A VI.UTILERIA
        .DELETE                          ;REQUEST
        BCC     2$
        JMP     ARCH.3                   ;ERROR? SALIA
2$:     JMP     ARCH.4


.ESCRIBE:
        MOV     R2,-(SP)                 ;PUSH R2
        MOV     R3,-(SP)                 ;PUSH R3
        MOV     6(SP),R3                 ;APUNTADOR A AREA
        MOV     @(R3),R2                 ;CANAL
        ASL     R2                       ;R2=R2*2
        CMP     DIRE.CAN(R2),USUARIO     ;EL CANAL PERTENECE A
                                         ; ESTA TAREA?
        BNE     ESC.INV                  ;NO, SALTA
        BIS     #EDO.SUS,VI.ESTADO(R0)   ;SUSPENDE LA TAREA
        MOV     #11.*^O400,VT.UTILERIA(R0);CLAVE
        MOVB    @(R3),VT.UTILERIA(R0)    ;CANAL
```

```
        MOV     @6(R3),VT.UTILERIA+2(R0);BLOQUE
        MOV     @2(R3),VT.UTILERIA+4(R0);DATOS
        MOV     @4(R3),VT.UTILERIA+6(R0);CUENTA
        MOV     #COMPLETA,VT.UTILERIA+8.(R0);DIRECCION DE COMPLETA
        ADD     #VT.UTILERIA R0          ;APUNTADOR A VT.UTILERIA
        .WRITC                           ;REQUEST
        BR      ARCH.5
ESC.INV:
        CLR     @8.(R3)                  ;TRANSFER=0
        MOV     #2,6(SP)                 ;CANAL INVALIDO
        MOV     (SP)+,R3                 ;POP    R3
        MOV     (SP)+,R2                 ;POP    R2
        BR      ARCH.1


.GUARDA:
        MOV     R2 -(SP)                 ;PUSH R2
        MOV     R3,-(SP)                 ;PUSH R3
        MOV     6(SP),R3                 ;APUNTADOR AL AREA
        MOV     @(R3),R2                         ;#CANAL
        ASL     R2                       ;R2=R2*2
        CMP     DIRE.CAN(R2),USUARIO     ;EL CANAL PERIENECE
                                         ;   A ESTA TAREA?
        BNE     GUARD.INV                        ;NO. SALIA
        MOV     #?*^0400 VT.UTILERIA(R0);CLAVE
        MOVB    @(R3),VT.UTILERIA(R0)    ;CANAL
        ADD     #VT.UTILERIA R0          ;APUNTADOR A VT.UTILERIA
        .CLOSE                           ;REQUEST
        MOV     #-1,DIRE.CAN(R2)         ;LIBERA EL CANAL
        BR      ARCH.4

GUARD.INV:
        MOV     (SP)+ R3                 ;POP R3
        MOV     (SP)+,R2                 ;POP R2
        BR      ARCH.1


.SLEE:
        MOV     R2,-(SP)                 ;PUSH R2
        MOV     R3,-(SP)                 ;PUSH R3
        MOV     -6(SP),R3                ;APUNTADOR AL AREA
        MOV     (R3),R2                  ;CANAL
        ASL     R2                       ;CANAL * 2
        CMP     DIRE.CAN(R2).USUARIO     ;EL CANAL PERIENECE A
                                         ;  ESTA TAREA?
        BNE     LECI.INV                 ;NO, SALIA
        BIS     #EDO.SUS,VT.ESTADO(R0)   ;SUSPENDE LA TAREA
        MOV     10.*^0400 VT.UTILERIA(R0);CLAVE
        MOVB    @(R3),VT.UTILERIA(R0)    ;CANAL
        MOV     @6(R3) VT.UTILERIA+2(R0);BLOQUE
        MOV     @2(R3) VT.UTILERIA+4(R0);DATOS
        MOV     @4(R3),VT.UTILERIA+6(R0);CUENTA
        MOV     #COMPLETA,VT.UTILERIA+8.(R0);DIR DE COMPLETA
        ADD     #VT.UTILERIA.R0          ;APUNTADOR A VT.UTILERIA
        .READC                           ;REQUEST
        MOV     (SP)+ R3                 ;POP R3
        MOV     (SP)+,R2                 ;POP R2
        BR      ARCH.5
LECI.INV:
        CLR     @8.(R3)          ;TRANSFER = 0
        MOV     #2,VT.R0(R0)     ;CANAL INVALIDO
        MOV     (SP)+ R3                 ;POP R3
        MOV     (SP)+,R2                 ;POP R2
        BR      ARCH.1


ARCH.0:
        CLR     VT.R0(R0)                ;LIMPIA R0(TAREA)
        BR      ARCH.1
```

```
ARCH.3:
        MOV     @#52,VI.R0(R0)          ;COLOCA ERROR EN RO(TAREA)

ARCH.1:
        BIS     #1,VI.PSW(R0)           ;PRENDE CARRY(TAREA)
        BR      ARCH.2

ARCH.4:
        BIC     #1,VI.PSW(R0)
        BR      ARCH.2

ARCH.5:
        MOV     R0,@8.(R3)
        MOV     (SP)+,R3                ;POP R3 (DE SIEE O SESCRIBE)
        MOV     (SP)+,R2                ;POP R2 (DE SIEE O SESCRIBE)
        BNE     ARCH.2
        GET.VI
        BIC     #EDO.SUS.VI.ESTADO(R0)  ;LEVANTA SUSPENSION
        BR      ARCH.3

ARCH.2:
        MOV     (SP)+,R1                ;POP R1
        MOV     (SP)+,R0                ;POP R0
        JMP     MIR.E2                  ;REGRESA A MIR.E2


COMPLETA:
        ASL     R1                      ;#CANAL * 2
        MOV     DIRE.CAN(R1),R1         ;#TAREA
        MOV     DIRE.NUM(R1),R1         ;IDENTIFICA LA TAREA
        BIC     #EDO.SUS.VI.ESTADO(R1)  ;LEVANTA SUSP.
        BIT     #1,R0                   ;HARDWARE ERROR?
        BNE     HARD.ERROR              ;SI SALTA
                                        ;NO HUBO ERROR
        BIC     #1,VI.PSW(R1)           ;CARRY(TAREA)=0
        RTS     PC                      ;REGRESA A RTI1

HARD.ERROR:
        MOV     #1,VI.R0(R1)            ;R0(TAREA)=1
        BIS     #1,VI.PSW(R1)           ;CARRY(TAREA)=1
        RTS     PC                      ;REGRESA A RTI1


;       SUBRUTINA     GET.CANAL
GET.CANAL:
        SEC                             ;PRENDE CARRY
        CLR     R1                      ;R1 = 0

OTRO:   CMP     R1,CAN.LIM              ;AUN HAY CANALES?
        BGE     NO.CAN                  ;NO, SALTA
        TST     DIRE.CAN(R1)            ;ESTA DISPONIBLE?
        BMI     SI.CAN                  ;SI, SALTA
        ADD     #2,R1                   ;R1=R1+2
        BR      OTRO                    ;BUSCA OTRO CANAL
SI.CAN: ASR     R1                      ;R1=R1/2
        CLC                             ;LIMPIA CARRY
NO.CAN: RTS     PC


        .END
```

```
        .LIST
        .SBTTL  INICIA
        .CSECT  INIC
        .MCALL  .SRESET,.TTYOUT
        .GLOBL  PERDIDO
INI.CH= 1
INI.CA= 2

;       ********************************************
;       *                                          *
;       *                                          *
;       *            I   N   I                     *
;       *                                          *
;       *                                          *
;       ********************************************


;       ARCHIVO:        ESOII.NEU
;       FECHA:  20-MAY-80


;INICIALIZA LAS TABLAS Y VARIABLES GLOBALES REQUERIDAS PARA EL
;       FUNCIONAMIENTO DEL MIR CON DOS PROCESADORES.
;
;ENTRADAS:
;       NUM.PROGS(SE DETERMINA EN LA ETAPA DE ENSAMBLADO)
;       DIRE.ORI(SE DETERMINA EN LA ETAPA DEL LINKER)
;
;SALIDAS:
;       DIRE.NUM
;       DIRE.PRIOR
;       VECT.TRANS
;       USUARIO
;       REL.SYS
;       ELI.CONTA
;


INICIA: .SRESET
        TYPE <SO1.NEU                           27-MAY-80>
        TYPE <SISTEMA OPERATIVO MULTITAREA>
        MOV     #FINDAT,CHEPAL
        SUB     #1000,CHEPAL
        ASR     CHEPAL                          ;CHEPAL=# DE PALABRAS DEL CHECKPOINT
        MOV     CHEPAL,CHEBLK
        BIC     #377,CHEBLK
        SWAB    CHEBLK
        ADD     #1,CHEBLK                       ;CHEBLK=# DE BLOQUES DE 256. PAL.
        .LOOKUP #AREA,#CANALES-1,#CHENOM;EXISTE CHECK. PARA CONTINUAR?
        BCC     LEECHE
        JMP     NOCHEC
LEECHE:         .READW  #AREA,#CANALES-1,#1000,CHEPAL,#0;SI,LEELO
                BCC     3$
                JMP     ERRCHE
3$:             .CLOSE  #CANALES-1
                BCC     4$
                JMP     ERRCHE
4$:                     TYPE    <CONTINUA>
                        MOV     #INT.CH,R1
                        JSR     PC,INTERR
                        JSR     PC,INICAN
                        JSR     PC,INIVI
                        BIS     #MA.USU,@#PSW
                        JSR     PC,INIPER
                        JMP     MTR.E3
NOCHEC:         TYPE    <INICIA>
                JSR     PC,INIVT
                JSR     PC,INIVAR
                JSR     PC,INIVI
                BIS     #MA.USU,@#PSW
                JSR     PC,INIPER
```

```
                JMP     MIR.E3
ERRCHE: TYPE    <ERROR INICIA>
        .EXIT


INIERR: CLR     DMAPEN
        PUSH    USUARIO
        MOV     #2.USUARIO              ;AVISA A LOS AFECTADOS
        WHILE USUARIO .LE. #NUM.USU DO BEGIN AC1
                GET.VI
                TSI     VI.SUS(R0)
                BEQ     3$
                BIC     #SUSOINI.VI.ESTADO(R0)
                BIS     #1 VT.PSW(R0)
                CLR     VI.SUS(R0)
                MOV     R1.VT.INT(R0)
                CLR     VI.R0(R0)
3$:             ADD     #2 USUARIO
        ENDWHILE AC1
        POP     USUARIO
        RTS     PC


INICAN:                                 ;INICIALIZA LOS CANALES I/O
        CLR     R1
        WHILE R1 .LT. #CAN.IIM DO BEGIN I1
                MOV     #-1.DIRE.CAN(R1)
                ADD     #2.R1
        ENDWHILE I1
        RTS     PC


INIVT:                                  ;INICIALIZA EL VECT.DE TRANS.
        MOV     #VECT.TRANS+VI.LOB.R0   ; DIRECCION DEL VECTOR DE TRANSACCIONES
        MOV     #2.USUARIO              ; PRIMER USUARIO.
WHILE USUARIO .LE. #NUM.USU DO BEGIN 11$
;       INICIALIZA ENTRADA EN EL DIRECTORIO POR NUMERO
        MOV     USUARIO.R1
        IF DIRE.ORI(R1) .NE. #0 THEN BEGIN 12$
                MOV     R0.DIRE.NUM(R1)
;               SE TRANSFIERE EL DESCRIPTOR DE TAREA AL VECTOR
;               DE TRANSACCIONES
                MOV     DIRE.ORI(R1).R1         ; EN R1 DIRECCION DEL DESCRIPTOR.
                MOV     (R1).VT.NOMBRE(R0)      ; NOMBRE DE LA TAREA.
                MOV     2(R1).VI.NOMBRE+2(R0)
                MOV     4(R1).VI.HE(R0)         ; HORA DE ENTRADA.
                MOV     6(R1).VI.HE+2(R0)
                MOV     8.(R1).VI.TI(R0)                ; TIEMPO DE INTERARRIBO.
                MOV     10.(R1) VI.TI+2(R0)
                CLR     R3
                MOV     12.(R1).R2              ; PRIORIDAD POR NUMERO.
                BEQ     INI.2                   ;SI PRIOR=0.SALIR
                                                ; SE TRANSFORMA LA PRIORIDAD
                                                ;    DE NUMERO A POSICION
                                                ;    DENTRO DE LA PALABRA.
                SEC
INI.1:
                ROL     R3
                DEC     R2
                BNE     INI.1
INI.2:          MOV     R3.VI.PRIORIDAD(R0)     ; PRIORIDAD POR POSICION.
                MOV     14.(R1).VT.STAR(R0)     ; DIRECCION ABSOLUTA DE INICIO
                MOV     16.(R1).VI.RETORNO(R0)  ; DIRECCION ABSOLUTA RETORNO
                MOV     18.(R1).VI.PILA(R0)     ; AREA PARA EL STACK
;               INICIALIZA ENTRADA EN EL DIRECTORIO POR PRIORIDAD
                MOV     12.(R1).R2              ; PRIORIDAD POR NUMERO.
                ASL     R2                      ; MULTIPLICA POR DOS .
                MOV     USUARIO.DIRE.PRIOR(R2)  ; COLOCA EN DIRE.PRIOR.
;               INICIALIZA ESTADO DE LA TAREA
                MOV     #EDO.MEZ.VT.ESTADO(R0)
                CLR     VI.SUS(R0)      ;DESCRIPTOR DE SUSPENSIONES
                CLR     VI.INT(R0)      ;DESCRIPTOR DE INTERRUPCIONES
        END.12$
```

```
        ELSE BEGIN 13$
                MOV     %VECT.TRANS DIRE.NUM(R1)
        ENDIF 13$
        ADD     #VT.IOB.R0      ; MUEVE EL APUNTADOR
                                ;   DEL VECTOR DE TRANS.
        ADD     #2.USUARIO      ; SIG. USUARIO
ENDWHILE 11$
        RTS     PC

INIVAR:                                 ;INICIALIZA VARIABLES
        CLR     USUARIO
        CLR     CHECK
        CLR     DMAPEN
        CLR     CONT.MIR
        CLR     CONT.MIR+2
        MOV     #-1.A.C.
        CLR     REL.SYS
        CLR     REL.SYS+2
        CLR     ELI.CONTA
        RTS     PC

INIVT:                                  ;PREPARA VECTOR DE INTERRUP.
        MOV     @#4.RT.4                ;GUARDA LA DIREC. DE TRAP4
        MOV     #T4.10HAN.@#4           ;INST.INVAL.,BUS ERROR,STACK OVF.
        MOV     @#6,RT.6
        MOV     #340.@#6
        MOV     @#10.RT.10              ;GUARDA LA DIREC. DE TRAP10
        MOV     #T4.10HAN,@#10
        MOV     @#12.RT.12
        MOV     #340,@#12
        MOV     #A.C.HAN,@#24           ;POWER FAILURE
        MOV     #340.@#26
        MOV     #LLA.E1.@#34
        CLR     @#36                    ; LLAMA PRIORIDAD 0, MODO KERNEL
        MOV     @#60,RT.60              ;PREPARA TECLADO
        MOV     #TECHAN.@#60
        MOV     @#62.RT.62
        CLR     @#62
        CLR     @#177546                ;DESACTIVA RIR
        MOV     @#100,RT.100            ;GUARDA LA DIREC. DE ATENCION A RIR
        MOV     #MIR.E1,@#^O100         ; CAMBIA DIRECCION DE ATENCION
        MOV     @#102,RT.102
        MOV     #301,@#102              ;MIR.E1 PRIORIDAD 6, MODO KERNEL, CARRY
        MOV     #LNKHAN.@#124           ;INTERPROC.LINK
        MOV     #240,@#126
        MOV     #SWIHAN,@#270           ;UNIBUS SWITCH
        MOV     #340,@#272
        MOV     #AD.HAN.@#VI.AD         ;DIRECCION DEL HANDLER CONVERSOR AD
        MOV     #300,@#VI.AD+2          ;KERNEL PRIORIDAD=6
        MOV     #INTNES,@#VI.CK         ;VECTOR DE INI. PARA RELOJ PROG.
        CLR     @#VI.CK+2               ;       "       "
        MOV     #INTNES,@#VI.DA         ;VECTOR DE INT. PARA CONVERSOR D/A
        CLR     @#VI.DA+2               ;       "       "
        RTS     PC

INIPER:                                 ;ACTIVA PERIFERICOS
        CLR     @#177560                ;DESACTIVA TECLADO
        MOV     #377,IOERR              ;AVISA QUE SE TRATA DE UNA PRUEBA
        MOV     #1,@#UNSWRE             ;PIDE EL SWITCH SIN INTERRUP.
        TST     IOERR
        BPL     3$
        TYPE    <FALTA SWITCH>
        BR      6$
3$:     BIS     #101,@#UNSWRE           ;PIDE EL SWITCH
        TSTB    @#UNSWRE                ;LO TENGO?
        BPL     3$
6$:     MOV     #377,IOERR              ;AVISA QUE SE TRATA DE UNA PRUEBA
        MOV     #AD.GO.@#AD.STA         ;HABILITA EL CONVERSOR
        TST     IOERR
        BPL     1$
        TYPE    <FALTA CAD>
```

```
)$:     MOV     #CK.1600,@#CK.BUF       ;PREPARA EL RELOJ PROG. PARA ACTIVAR BI
        MOV     #CK.GO.@#CK.STA         ;CONVERSOR 1600 VECES POR SEG.
        CLR     @#DA.STAT               ;INICIALIZA CONVERSOR D/A
11$:    MOV     #377,IOERR
        MOV     #0,@#IKSTRE             ;PRUEBA EL LINK
        TST     IOERR
        BPL     2$
        TYPE    <FALIA LINK>
2$:     CLR     IOERR                   ;NO MAS PRUEBAS DE I/O
        TYPE    <EN OPERACION>
        MOV     #100.@#177546           ;HABILITA RIR
        RTS     PC



;**********    TAREA   NULA    *********
; LA TAREA NULA NO NECESITA DESCRIPTOR   PORQUE SUS PARAMEIROS SON PUESTOS
;EN VI.*   DESDE EL MOMENTO DEL ENSAMBIADO.
        .BIKW   20                      ;STACK PARA TAREA NULA (ACEPTA 8 NIVELES
                                        ;DE INTERRUPCION)
TAR.NULA:
        BR      TAR.NULA

;*********** S E R V E S **************************

INTNES=0                                ;DIREC.INTERRUP.NO ESPERADA
VI.AD=340                               ;VECTOR DE INT. PARA CONVERSOR
                                        ;ANALOGO DIGITAL
VI.CK=344                               ;VECTOR DE INI. PARA RELOJ PROG.
VI.DA=350                               ;VECTOR PARA EL CONVERSOR D/A
AD.STA=170400                           ;STATUS DEL CONVERSOR
AD.BUF=170402                           ;BUFFER DE ENTRADA DEL CONVERSOR
AD.GO=^B1000000100000                   ;BIPOLAR.CANAL0.INI.ENAB.CK.OVERFLOW
CK.STA=170404                           ;STATUS DEI RELOJ
CK.BUF=170406                           ;PRESET DEI RELOJ
CK.GO=^B100000101                       ;REPEAI.OV.INT.ENAB.100K.ENABLE
CK.1600=62.                             ;PRESEI PARA 1600HZS. APROX.
DA.STA=170410                           ;STATUS DEL CONVERSOR DIGITAL ANALOGO
DA.X=170412                             ;PUERTO DIGITAL ANALOGO X
DA.Y=170414                             ;PUERTO DIGITAL ANALOGO Y


AD.HAN:
        MOV     R0,-(SP)                ;GUARDA R0
        MOVB    @#AD.STA+1.R0           ;OBTEN # DE CANAL LEIDO
        BIC     #177760.R0              ;AISLA EL # DE CANAL(4 BITS)
        ASL     R0                      ;MULTIPLICALO POR 2 PARA USARLO COMO INDICE
        MOV     @#AD.BUF,AD.AREA(R0)    ;GUARDA LA LECTURA DONDE CORRESPONDE
        INCB    @#AD.STA+1              ;INCREMENTA EI # DEI CANAI
        MOV     (SP)+.R0                ;RESTAURA R0
        RTI                             ;REGRESA AI INIERRUMPIDO


IECHAN:                                 ;HANDLER DEL IECIADO
        RTS     PC


INKHAN:                                 ;HANDLER DEL LINK
        RTS     PC

;*********** E M R G E N  **************

I4.10HAN:                                       ;IRAPS A MEMORIA 4 Y 10
        BIT     #MA.USU.@#PSW           ;INFRACION=TAREA O SISIEMA?
        BNE     EXPULSA
        TSTB    IOERR                   ;ERA SOLO UNA PRUEBA?
        BPL     RENUNCIA
        MOV     #177400.IOERR           ;SI.AVISA QUE HUBO ERROR
```

```
RENUNCIA:                                   ;NO ERA PRUEBA ALGO ANDA PESIMO
        JSR     PC,RI.NOVC                  ;PREPARA RETORNO A RTI1
        BIC     #340,@#PSW                  ;BAJA LA PRIORIDAD
        TYPE    <RENUNCIA>
1$:     .ITYOUI #7                          ;ALARMA
        BR      1$


EXPULSA:                                    ;SACA DE MEZCLA AL INFRACTOR
        PUSH    R0,R1
        GET.VI                              ;LOCALIZALO EN VECT.TRANS.
        BIC     #EDO.MEZ.VI.ESTADO(R0)
        CLR     R1                          ;QUITALE SUS CANALES DE I/O
        WHILE R1 .LT. #CAN.LIM DO BEGIN X1
                IF DIRE.CAN(R1) .EQ. USUARIO THEN BEGIN X2
                        MOV     #-1 DIRE.CAN(R1)
                        ASR     R1
                        .CLOSE  R1
                        ASL     R1
                ENDIF X2
                ADD     #2,R1
        ENDWHILE X1
        TYPE    <EXPULSION>
        POP     R1,R0
        BIC     #340,@#PSW                  ;BAJA LA PRIORIDAD
        JMP     MTR.E2


A.C.HAN:                                                ;INTERRUPCION DE A.C.
        TST     A.C.                        ;VA O VIENE?
        BEQ     REESTABLECE
        CLR     A.C.                        ;SE VA.PONTE A INVERNAR
        PUSH    R0,R1,R2,R3,R4,R5           ;GUARDA REGISTROS VOLATILES
        MFPI    SP                          ;GUARDA SP DEL MODO ANTERIOR
        PUSH    @#PSW                       ;GUARDA PSW
        MOV     SP,A.C.SP                   ;INCLUYENDO EL STACK POINTER
1$:     BR      1$
REESTABLECE:                                ;VIENE.REINICIA EL PROCESO
        RESET                               ;I/O SE PERDIO
        MOV     A.C.SP,SP
        POP     @#PSW                       ;RESTAURA PSW
        MTPI    SP                          ;RESTAURA SP DEL MODO ANTERIOR
        MOV     #-1,A.C.                    ;BANDERA REESTAB.
        MOV     #101,@#UNSWRE               ;NO SUELTES EL SWITCH
        BIC     #340,@#PSW                  ;BAJA LA PRIORIDAD
        MOV     #10.,R1                     ;ESPERA 5 SEG. A QUE SE ESTABILICE
1$:     MOV     #-1,R0
2$:             DEC     R0
                BNE     2$
        DEC     R1
        BNE     1$
        TYPE    <INTERRUPCION C.A >
        MOV     #INI.CA,R1
        JSR     PC,INTERR
        JSR     PC,INIPER                   ;REINICIALIZA PERIFERICOS
        POP     R5,R4,R3,R2,R1,R0
        RTI


SWIHAN:                                     ;HANDLER DEL SWITCH(RASCATE)
        MOV     #101,@#UNSWRE               ;APAGA WATCHDOG TIMER
        TSTB    @#UNSWRE                    ;PERDI EL SWITCH?
        BPL     PERDIDO
        RTI                                 ;NO,CONTINUA LA OPERACION
PERDIDO:                                    ;SI,TOMA EL LUGAR DEL SUPLENTE
        BIS     #340,@#PSW
        JSR     PC,RI.NOVO                  ;PREPARA RETORNO A RTI1
        BIC     #340,@#PSW                  ;BAJA LA PRIORIDAD
        TYPE    <SWITCH PERDIDO>
1$:     .ITYOUI #7                          ;ALARMA
        BR      1$
```

```
RI_NOVO:
        MOV     RI_4 @#4
        MOV     RI_6,@#6
        MOV     RI_10 @#10
        MOV     RI_12,@#12
        MOV     RI_100 @#100
        MOV     RI_102 @#102
        RESET
        .SRESET
        RTS     PC

        .END    INICIA
```