ASPECTOS HARDWARE Y SOFTWARE DEL MICROPROCESADOR 68000

Raúl Alcaraz Martínez

INDICE

CAPITULO 1

Introducción al 68000

1. Introducción al 68000.
2. Generalidades para el desarrollo del software.

1. Introducción al 68000.

El **MC68000** es un microprocesador de 16/32 bits fabricado por Motorola desde 1980.

1.1- Características Generales.

Descripción del microprocesador:

1. Alimentación única: +5V. Frecuencia de funcionamiento 4-25MHz
2. Bus de DATOS de 16 bits.
3. Bus de DIRECCIONES de 24 bits, por lo que el espacio direccionable es de 16 Megabytes.
4. Soporta 5 tipos de datos: Bits (1), BCD (4), BYTE(8), WORD (16) y LONG (32 bits)
5. Conjunto de instrucciones: 56 tipos diferentes.
6. 14 modos de direccionamiento.
7. I/O "Mapeado" en memoria: Tratamiento de los dispositivos de entrada / salida como posiciones de memoria.
8. BUS Síncrono y Asíncrono.
9. Arquitectura de la CPU:
    - 17 Registros de 32 Bits:
        - 8 Registros de datos D0-D7.
        - 8 para direcciones A0-A7
        - 1 Contador de programa.
    - 3 ALU's: una para cálculos aritmético lógicos y dos para determinar direcciones efectivas.
10. Interrupciones vectorizadas con 7 niveles de prioridad enmascarables.

El **MC68000** puede ejecutar sus instrucciones en dos modos diferentes: el modo usuario y el modo supervisor.

- El MODO USUARIO conforma un entorno de programación para la ejecución de los programas de aplicación. En este modo no se pueden ejecutar ciertas instrucciones y no se puede acceder a parte del registro de Estado.

- El MODO SUPERVISOR permite utilizar el juego de instrucciones completo y acceder a todo el registro de Estado, tanto en lectura como en escritura. Se establece así unos "privilegios" de utilidad en sistemas operativos y software de base de los sistemas.

2. Generalidades para el desarrollo software.

El **MC68000** dispone de características que convierten la programación en una actividad más sencilla, rápida y productiva de lo que era con microprocesadores anteriores.

2.1. Estructura consistente

La estructura regular del 68000 simplifica notablemente el esfuerzo que supone escribir programas en ensamblador. Podemos destacar como características notables las siguientes:

- Existen catorce modos de direccionamiento consistentes e independientes de las instrucciones.
- Todos los registros de direcciones pueden usarse de manera directa, indirecta e indexada. Todos los registros de datos y de direcciones pueden utilizarse como índices.
- Todos los registros de direcciones pueden utilizarse como punteros de pilas gracias a los direccionamientos con predecrementos y postincrementos.

2.2. Programación modular estructurada

Los principios de la programación estructurada sugieren ciertos mecanismos de utilización de las subrutinas, la posibilidad de programar código reentrante y rutinas recursivas. Todo esto se ve facilitado en el 68000 principalmente por los siguientes motivos:

- Las instrucciones **LINK** y **UNLIK** permiten la manipulación de listas ligadas de áreas de datos en el stack.
- Las instrucciones **MOVEM** reduce los tiempos de entrada y salida de subrutinas siendo a su vez un mecanismo sencillo de protección de recursos.
- Dieciséis **TRAPs** permiten complementar llamadas a rutinas de sistema operativo o simular nuevas instrucciones.
- El potente sistema de interrupciones permite la generación de rutinas reentrantes modulares. Se dispone de siete niveles de prioridad con 192 vectores y autovectores (199 de los 255 del microprocesador).
- La distinción entre modos de funcionamiento usuario y supervisor permite controlar la relación entre rutinas de sistema operativo y de aplicación.

2.3. Capacidades de test del software

En los apartados anteriores se han comentado aspectos del 68000 que permiten programar con un reducido nivel de errores, no obstante siempre se cometen algunos, por lo que este microprocesador cuenta además con algunas facilidades para detectarlos y corregirlos.

- Existen " excepciones " que indican las siguientes condiciones anormales:
  - Accesos a una dirección impar con un operando word o long word (**Address error**)
  - Instrucciones ilegales (**Illegal instruction**).
  - Accesos a direcciones de memoria no válidas (**Bus error**).
  - Divisiones por cero (**Zero divide**).
  - Reboses aritméticos (**TRAPV**).
  - Fuera de límites en registros (**CHK**).
  - Interrupciones espúreas (**Spurious interrupt**).
- El usuario dispone de **16 TRAPS** que le permiten generar sus propios sistemas de detección y corrección de errores.
- El 68000 dispone además de funcionamiento en modo " traza " que permite ejecutar instrucciones paso a paso.

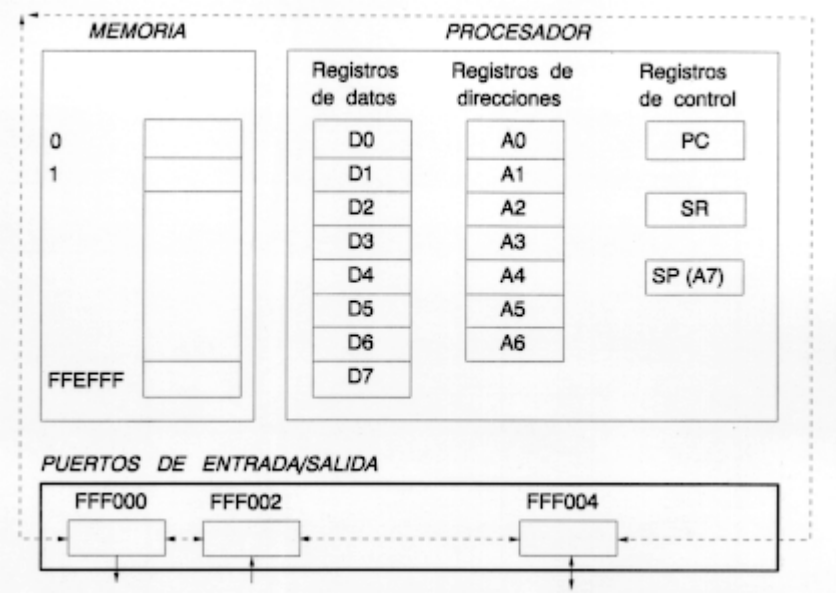
CAPITULO 2

Organización interna del 68000

1. Modelo de memoria y tipos de datos.
2. Modelo de registros.
3. Modos de direccionamiento.
4. Instrucciones.

1. Modelo de memoria y tipos de datos.

A continuación se irán describiendo los tres componentes principales del computador: la memoria, el procesador y la unidad de entrada / salida (según el esquema adjunto).



1.1. La memoria.

La memoria principal de este computador está formada por celdas de un byte (8 bits), que constituyen la unidad básica de lectura o escritura, identificándose mediante una dirección. Los procesos de lectura y escritura pueden realizarse con varias celdas consecutivas simultáneamente, debiendo indicar el procesador a la memoria principal dos parámetros, la dirección de la primera celda de memoria y la longitud de la información a la que se desea acceder. Siendo esta longitud de un byte, dos bytes (una palabra) o cuatro bytes (palabra larga).

El tamaño máximo de la memoria viene determinado por el número de bits de los registros de direcciones que tiene el procesador, siendo en el caso del Motorola 68000 de 32 bits pero, debido a limitaciones en el montaje solo pueden utilizarse 24 como máximo, así que la máxima longitud que se puede usar de la memoria principal es de 2^{24} bytes, desde 0 hasta FFFFFF .

El procesador puede leer y escribir información de diferentes tamaños, existiendo una norma para almacenar las palabras (W) y las palabras largas (L), y siendo esta la de comenzar por el byte más significativo.

1.1.1. Acceso a memoria.

Habiendo establecido este espacio lineal de direcciones, quedan reservadas las direcciones comprendidas entre la 0 y la 1024 para usos específicos esenciales del 68000, conocidas como **memoria del sistema** y **datos del sistema.** Estas direcciones contienen importantes tablas para el tratamiento de las interrupciones, cargas de operativos, etc., y no deben ser alteradas por el usuario. Algunas de estas direcciones pueden estar en la ROM, de forma que no desaparezcan al desconectar la alimentación, pero seguirán utilizando espacio de direcciones como cualquier otra posición de memoria. Habrá, con toda seguridad, otras áreas de memoria asignadas permanentemente a otras funciones no accesibles al usuario de programas y datos.

Por disponerse de un bus de dato de 16 bits, la memoria puede considerarse formada por unidades de ese tamaño. El MC68000 lee 16 bits cada vez , coincidiendo con el formato de codificación de instrucciones en palabras de 16 bits. No obstante Se podrá acceder a la memoria para leer un BYTE , una palabra WORD, ó una doble palabra LONG, siempre y cuando se tengan en cuenta las siguientes reglas:

1. Las direcciones de los bytes pueden ser pares o impares. Sus incrementos son de 1 en 1.
2. Las direcciones de las palabras son siempre pares. Sus incrementos son de 2 en 2.
3. Las direcciones de las dobles palabras son pares. Sus incrementos son de 4 en 4.
4. Cuando se accede a una palabra que está en la dirección N, siendo N un número par, el byte de la dirección N es el Byte más significativo y el de la dirección N+1 es el byte menos significativo. Lo que significa que cuando se mira una palabra de memoria, su byte más significativo está en la dirección más baja, y su byte menos significativo está en la dirección más alta.
5. Al acceder a una doble palabra también se realiza de tal manera que la palabra alta será la de dirección más baja, mientras que la palabra baja está en la dirección más alta.
6. Si se intenta acceder a una palabra o a una doble palabra en una dirección impar, se provocará un error.

Recordando lo más característico de la manera de acceder a memoria es:

- No existencia de segmentación o restricciones en el tamaño del programa, siendo utilizable todo el espacio de direcciones de manera lineal.
- Las instrucciones de los programas se almacenan como palabras.
- Los datos pueden tratarse como bytes, word ó long.
- Las direcciones de palabra(WORD) y doble palabra(LONG) son pares.

1.2. Tipos de datos.

Un microprocesador necesita, además de instrucciones, datos con los que trabajar y procesar.

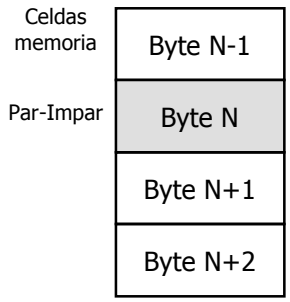
Los tipos de datos son pues las representaciones binarias con los que un valor puede ser codificado.

El microprocesador 68000 puede manejar un total de 4 formatos de datos, a saber: Bytes, Palabras (words), Palabras larga (long word) y BCD.

Byte: 8 bits

Un dato de este formato puede ser transferido en un solo ciclo de bus.

Las direcciones en memoria pueden ser tanto par como impar, y por lo tanto los incrementos del PC son de 1 celda de memoria.

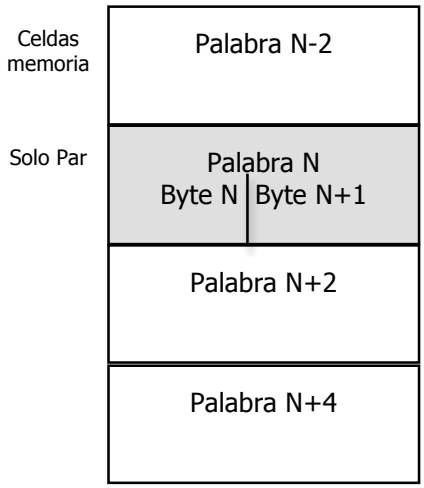


Palabras: 16 bits

Los datos bajo este formato también pueden ser transferidos en un solo ciclo de bus.

Existen sin embargo ciertas restricciones para su direccionamiento. Los datos Palabra solo pueden almacenarse en direcciones de memoria pares. De esta forma, su byte más significativo se almacena en la parte baja (par) de la celda y su byte menos significativo en la parte alta (par) de la misma.

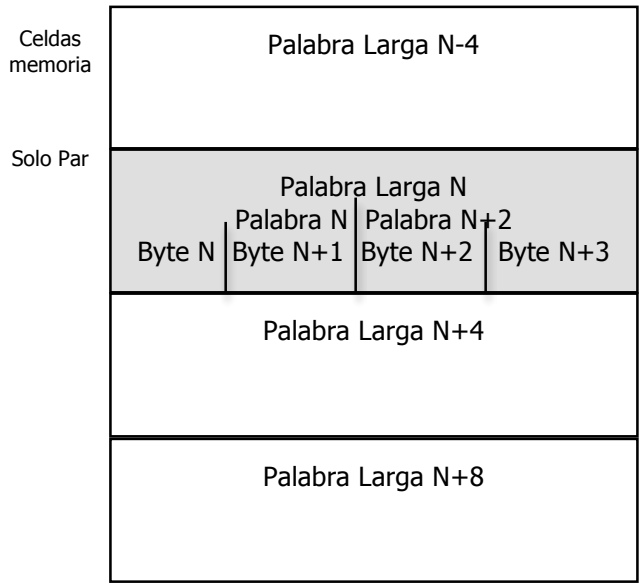
Los incrementos de memoria son ahora de 2 bytes.



Palabras largas: 32 bits

Puesto que los registros internos del 68000 son de 32 bits, se consigue también manejar este tipo de datos, pero al tener un bus de datos de 16 bits se necesitan 2 ciclos de bus para una transferencia.

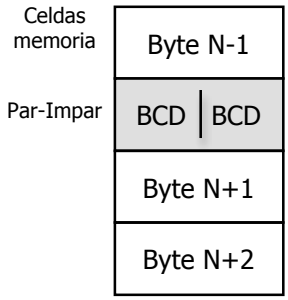
Las direcciones son ahora pares también, pero los incrementos son ahora de 4 bytes. Así, a nivel de byte, el más significativo se guarda en la parte más baja de la palabra larga y el menos significativa en la parte más alta de esta.



BCD: Binary Coded Decimal 4 - 4 bits

El 68000 es capaz de utilizar datos en BCD, utilizando cada dígito 4 bits (de 0=0000 hasta 9=1001), lo que permite almacenar 2 datos BCD en un mismo byte. Los datos en un byte variaran pues de 00 hasta 99.

En esta organización, el dato BCD más significativo se guarda ahora en la parte alta del byte y el menos significativo en la parte baja.



2. Modelo de registros.

2.1. Registro de direcciones.

Existen 7 registros de direcciones y son: **A0, A1, A2, A3, A4, A5 y A6**, siendo estos de 32 bits aunque solo se pueden utilizar 24 bits para direccionar como antes se ha mencionado.

Estos registros pueden actuar como registros de 32 ó 16 bits, con los datos alineados con los bits de menor peso, como en el caso de los registros de datos, pero a diferencia de éstos la carga o modificación de los 16 bits de menor peso, provoca la extensión de signo a los 32 bits, lo que modifica los 16 bits de mayor peso. Este comportamiento permite trabajar con desplazamientos de memoria negativos, complemento a 2.

El contenido del byte alto (bits 24 a 31) de estos registros es irrelevante, dado que el bus de direcciones del 68000 es de 24 bits, lo que le permite direccionar hasta 16 Mbytes (de $0 a -$FFFFFF). Las versiones del 68000 que ya disponen de un bus de direcciones de 32 bits utilizan plenamente los 32 bits de estos registros.

El registro de direcciones **A7** puede ser empleado como tal, sin embargo, su uso está relacionado con la gestión de la pila y por ello su modificación ha de estar justificada en el programa.

Este registro se desdobla en dos registros independientes:

- Puntero de pila de supervisor (SSP).
- Puntero de pila de usuario (USP).

Esto es necesario puesto que en modo usuario se accederá al puntero de pila de usuario, mientras que en modo supervisor se puede acceder tanto al puntero de pila de usuario como de supervisor. Además cuando se produce el salto a una subrutina la dirección de retorno se almacena en la pila de usuario, mientras que cuando se produce una interrupción la dirección de retorno y el contenido del registro de estado se guardan en la pila de supervisor. Se accederá directamente a un puntero o a otro (SSP o USP) automáticamente según el uP se encuentre funcionando en un modo o en otro (supervisor o usuario respectivamente).

2.2. Registros de datos.

El M68000 consta de 8 registros de datos: **D0, D1, D2, D3, D4, D5, D6 y D7**. Cada uno de ellos de 32 bits. En muchas instrucciones existe la posibilidad de especificar el tamaño del dato, indicándose este mediante el sufijo S (B, W y L), que va añadido al mnemotécnico de la instrucción. Dichos modificadores determinan que bytes del registro son modificados, los demás permanecen inalterados.

La forma en que se almacenan los datos en los registros, viene dada por su longitud, ya que como esta es variable, irán ocupándolos de izquierda a derecha empezando por el bit menos significativo del registro.

Estos registros también permiten la manipulación de bits individuales y operaciones con datos decimales codificados en BCD.

2.3. Registros de propósito especial.

Los registros de propósito especial son el **contador de programa PC (Program Counter)** y el **registro de estado SR (Status Register)**.

El PC tiene un tamaño de 32 bits y guarda la dirección de la siguiente instrucción a ejecutar (al menos, generalmente). La capacidad de este PC no es aprovechada debido a una limitación en el chip, el cual emplea únicamente 24 patillas para direccionar la memoria, como ya se ha explicado. De cualquier modo, la existencia del PC no afectar demasiado a nuestra labor, dado que no se esta permitido el acceso manual a dicho registro.

El SR es un registro de 16 bits que está divido en dos campos lógicos. El byte más significativo es denominado el **byte de sistema** y controla el modo de funcionamiento del 68000 y no puede ser modificado por el programador cuando nos hallamos bajo el modo usuario. Sus bits significativos son :

a) **T (Trace)**
b) **U (User)**
c) **I_0 , I_1 , I_2 (Interrupt mask bits)**

El byte menos significativo constituye el CCR (Condition Code Register) y guarda constancia de diversas situaciones que tiene lugar en el 68000. Al realizar operaciones matemáticas existe la posibilidad de desbordamiento, por lo que se realiza un test automáticamente, quedando el resultado almacenado en CCR. Este registro consta de 5 flags útiles que se almacenan en los 5

bits menos significativos y reciben las siguientes denominaciones, ordenadas desde el bit menos significativo:

a) **C** : indicador de acarreo o carry flag: indica el valor del bit de acarreo de la posición más significativa del resultado de una operación, se pone a 1 si existe desbordamiento

b) **V** : indicador de desbordamiento o overflow flag: indica si en el resultado de una operación en complemento a 2 existe desbordamiento, se pone a 1

c) **Z** : indicador de cero o zero flag, se pone a 1 cuando sea 0 el resultado de una operación aritmética o lógica

d) **N** : indicador de número negativo o negative flag, se pone a 0 si es positivo y a 1 si es negativo el signo del resultado de una operación en complemento a 2

e) **X** : indicador extendido o extended flag que funciona de la misma manera que C, pero únicamente con operaciones aritméticas o de desplazamiento

3. Modos de Direccionamiento.

Existen cuatro modos de direccionamiento:

1.- **Direccionamiento inmediato**: almacena el operando precedido del símbolo # en el registro indicado.

Ejemplo: MOVE.L #$18,D6

(el número $18 es un operando inmediato por que forma parte de la propia instrucción)

2.- **Direccionamiento absoluto (o directo)**: almacena el operando que está en la dirección de memoria especificada en el registro de datos indicado.

Ejemplo: ADD.W 1000,D2

(suma la palabra de dirección 1000 al registro D2)

3.- **Direccionamiento directo mediante registro**: en este modo de direccionamiento, los registros contienen los datos sobre los que se trabaja. Por tanto, no implica accesos a memoria.

Ejemplo: MOVE.B D3,D4

(copia el primer byte del contenido del registro D3 al primer byte de D4, sin modificación alguna de los bytes restantes)

4.- **Direccionamiento relativo a registro (indirecto)**:

a) **Direccionamiento mediante registro normal**: la dirección de uno (y sólo uno) de los operandos viene dada a través de un puntero en uno de los registros. El registro que actúa como puntero se marca mediante paréntesis y debe ser uno de los registros de dirección.

Ejemplo: ADD.B (A0),D6

(suma el contenido de la posición de memoria –byte- cuya dirección está en A0 al registro D6, guardando el resultado en este último)

Ejemplo ADD.B D6,(A0)

(suma el byte menos significativo de D6 al byte al cual apunta A0)

b) **Direccionamiento relativo a registro con post-incremento**: su funcionamiento y sintaxis es común al modo anterior, con la salvedad que incrementa en una cantidad de memoria, según sea el tamaño del operando (1 para .B, 2 para .W y 4 para .L), después de operar, el contenido de la posición de memoria indicada por el registro de direcciones.

Ejemplo: MOVE.W (A0)+,D0

(copia en D0 el contenido de la posición de memoria direccionada por A0 y luego incrementa en 2 -.W- el contenido de A0)

c) **Direccionamiento relativo a registro con predecremento**: decrementa en una cantidad de memoria, según sea el tamaño del operando, el registro de direcciones y opera después con el contenido de la posición de memoria cuya dirección es el nuevo valor de dicho registro.

Ejemplo: MOVE.B -(A0),D0

(decrementa en 1 -.B- el contenido del registro A0 y luego copia en D0 el contenido de la nueva posición de memoria direccionada por A0)

c) **Direccionamiento relativo a registro con desplazamiento**: la dirección efectiva se calcula mediante la adicción al contenido del registro de dirección el desplazamiento indicado en la palabra de 16 bits (con signo) que acompaña la sintaxis

Ejemplo: MOVE.L 12(A4),D3

(copia en los 2 bytes menos significativos de D3 el contenido de la posición de memoria cuya dirección viene dada por 12 más el contenido de A4. No se modifica A4)

Ejemplo MOVE.L –12(A4),D3

(el desplazamiento puede ser un número comprendido entre –32.768 y +32.767)

e) **Direccionamiento relativo a registro con índice**: Este modo de direccionamiento es la extensión natural del anterior. Para formar la dirección efectiva de un operando, los contenidos del registro de dirección especificado es sumado al contenido al registro general, a la vez que se añade también un desplazamiento codificado en 8 bits. El registro general puede ser un registro de datos o de direcciones, y recibe el nombre de registro índice.

Ejemplo: MOVE.B 4(A0,D1), D0

(copia en D0 el contenido del byte cuya dirección es el resultado de sumar el número 4, el contenido del registro A0 y el contenido del registro D1. No ven alterados ni A0 ni D1)

f) **Direccionamiento relativo al contador de programa con desplazamiento**: cuando es necesario hacer referencia a un operando relativo a la posición de la próxima instrucción que va a ser ejecutada. El desplazamiento puede ser un número con signo de 16 bits.

Ejemplo: MOVE.B -24(PC),D0

(copia en D0 el contenido de la posición de memoria cuya dirección es la suma de -24 y el valor del contador del programa – PC no se ve alterado)

g) **Direccionamiento relativo al contador de programa con índice**: utiliza como dirección efectiva la suma del contenido de PC, el contenido de un registro de índice (de datos o dirección) y un desplazamiento dado en un número de 8 bits con signo.

Ejemplo: MOVE.B 24(PC,D0),D1

(copia en D1 el contenido del byte direccionado por el resultado de sumar el número 24, el contador de programa y el contenido de D0)

4. Instrucciones del Motorola 68000

4.1. Formatos de instrucciones

Los formatos empleados para las instrucciones utilizan una o más palabras de 16 bits. La primera palabra especifica el código de la operación y en muchos casos la dirección de un operando. Las especificaciones para completar los operandos, cuando no es suficiente con una palabra van a continuación de la primera palabra. Cada operando utilizará como máximo dos palabras de ampliación, equivalente a una palabra larga, después de la del código de operación, por lo que la instrucción más larga del M68000 ocupa 5 palabras (10 bytes), siendo 1 para el código de instrucción, 2 palabras de ampliación para el operando origen, y 2 palabras de ampliación más para el operando destino.

La información que identifica la situación exacta del operando, denominada dirección efectiva, se codifica en los formatos de instrucción mediante dos campos, siendo uno el modo de direccionamiento (MD) y el otro el de registro. Cada campo tiene un tamaño de 3 bits y se incluyen en la primera palabra de instrucción. El campo MD identifica el modo de direccionamiento empleado y el campo CR indica el registro empleado para obtener la dirección del operando. A veces se utilizan las palabras de ampliación ya que la dirección efectiva requiere incluir más información sobre la situación del operando en la palabra de instrucción.

Antes de terminar este apartado sobre los direccionamientos, no podemos olvidarnos hacer notar que aunque el M68000 tiene una arquitectura muy regular, en el sentido de que no existen registros de datos de direcciones especiales que sean empleados únicamente con determinadas instrucciones (salvo el registro de estado - SR), no es así con la regularidad de sus modos de direccionamiento. Por ello, no todas las instrucciones permiten todos los direccionamientos presentados.

A continuación se desarrollarán las instrucciones del M68000, agrupadas según la función básica que realizan, estudiando la operación que realizan, su sintaxis, sus campos y algunos ejemplos.

4.2. Instrucciones de transferencias de datos

El conjunto de estas instrucciones permite el movimiento de datos entre registros de la CPU, entre registros y memoria y entre posiciones de memoria. Estas son las siguientes:

a) **MOVE** : realiza la transferencia de un dato desde fuente a destino. Modifica el CCR de forma que refleja el signo del dato movido y el hecho de que sea cero o no. Los flags C y V se ponen a cero y el flag X no se modifica. También el manejo de la pila se lleva a cabo mediante esta instrucción, utilizando direccionamiento indirecto con el registro A7 que actúa como puntero de pila. Para llevar un dato a la pila el direccionamiento es indirecto con predecremento, mientras que para extraerlo se emplea el indirecto con post-incremento.

b) **MOVEA (Move Adress)** : es una variante dedicada a la transferencia de direcciones. El tamaño del dato es de 16 ó 32 bits. No modifica el CCR

c) **MOVEQ (Move Quick)** : tiene por finalidad la carga rápida de un registro de datos.

d) **MOVEM (Move Múltiple)** : carga una serie de posiciones consecutivas de memoria en varios registros a la vez. No afecta al CCR.

Ejemplo: MOVEM.L $1000,D0-D2/A2-A4/D7

(copia los contenidos de las posiciones $1000, $1002, $1004,...$100C en los registros D0, D1, D2, D7, A2,A3 y A4 respectivamente)

e) **EXG y SWAP (ExchanGe registers y SWAP register)** : la primera intercambia el contenido completo de dos registros de datos o de dirección, mientras que la segunda actúa sobre un único registro, siempre de datos, intercambiando sus palabras alta y baja. La realización de estas transferencias sin la existencia de estas instrucciones, requeriría la ejecución de tres instrucciones MOVE, y la utilización de otro registro o de la memoria como almacenamiento temporal

f) **LEA y PEA (Load Effective Adress y Push Effective Adress)** : determinan la dirección efectiva del operando fuente. La primera almacena la dirección efectiva calculada en el registro de direcciones que se especifique como operando destino y la segunda lo lleva a la pila. Ninguna de las dos afectan al CCR

g) **LINK y UNLINK** : facilitan las operaciones necesarias para el paso de parámetros a/de subrutinas a través de la pila, ofreciendo la ventaja de ser independiente de las áreas de memoria de datos de un programa, y de no necesitar el uso de etiquetas, ni de direcciones concretas para acceder a dichos parámetros, facilitando así la inclusión de las subrutinas en programas diferentes sin necesidad de hacer cambios en ellas. La instrucción LINK reserva una zona de memoria en la pila, desplazando el puntero de

pila (SP) hacia direcciones menores en el valor que se indique. La instrucción UNLINK restablece la pila en la situación que se encontraba antes de la ejecución de LINK, cargando el puntero de pila con el contenido del registro de direcciones que se indica y a continuación extrae la palabra larga apuntada por SP y la lleva al registro de direcciones, quedando así restaurado

4.3. Instrucciones aritméticas

El Motorola 68000 dispone de instrucciones para las 4 operaciones aritméticas, sobre operandos binarios, y suma y resta sobre datos codificados en BCD. Además de cambio de signo para ambos tipo de datos, instrucciones de comparación, extensión de signo y actualización de los códigos de condición (CCR) según el valor de un dato. Estas instrucciones son las siguientes :

a) **ADD** : realiza la suma de los operandos fuente y destino , quedando el resultado almacenado en destino. Modifica el CCR en función del resultado de la operación

b) **ADDA (Add Adress)** : realiza la operación de la suma, siendo el destino un registro de direcciones en vez de datos

c) **ADDI (Add Inmediate)** : realiza la operación suma mediante direccionamiento inmediato

d) **ADDQ (Add Quick)** : realiza la operación suma mediante direccionamiento inmediato siendo el destino menor o igual que 8

e) **ADDX (Add Extended)** : incluye en el resultado la suma del flag X, lo que facilita las operaciones con valores que superan la capacidad de los registros (precisión múltiple)

f) **SUB (Substract)** : realiza la diferencia entre fuente y destino, depositando el resultado en destino

g) **SUB.A (Substract Adress)** : realiza la operación de la resta, siendo el destino un registro de direcciones en vez de datos

h) **SUBI (Substract Inmediate)** : realiza la operación resta mediante direccionamiento inmediato

i) **SUBQ (Substract Quick)** : realiza la operación resta mediante direccionamiento inmediato siendo el destino menor o igual que 8

j) **SUB.X (Substract Extended)** : incluye en el resultado la resta del flag X, lo que facilita las operaciones con valores que superan la capacidad de los registros (precisión múltiple)

k) **MULS y MULU (Multiply Signed y Multiply Unsigned)** : realizan la multiplicación sobre datos de 16 bits, generando un resultado de 32 bits, siendo el destino un registro de datos. Los flags C y V siempre quedan a 0 y el flag X no se modifica, reflejando el resultado de la operación los flags Z y N

l) **DIVS y DIVU (Divide Signed y Divide Unsigned)** : realizan la división de un dato de 32 bits (en destino) por otro de 16 bits (en fuente), generando dos resultados de 16 bits: el cociente y el resto, siendo obligatorio que el destino sea un registro de datos. El cociente se guarda en los 16 bits menos significativos y el resto en los 16 bits más significativos del registro destino. Ambas instrucciones ponen el flag C a 0, y el flag X no se modifica. Puede producirse desbordamiento si el divisor es pequeño, reflejándose en el flag V. Los flags N y Z son modificados reflejando el resultado de la operación

m) **CMP (Compara)** : las instrucciones de comparación efectúan la substracción destino-fuente pero la diferencia no se lleva a ningún destino, limitándose a afectar a los flags N, Z, V y C, de forma que se puedan comprobar diversas relaciones entre los datos que se comparen con el fin de producir ramificaciones en el programa

n) **CMPA (Compare Adress)** : utiliza como destino un registro de direcciones

o) **CMPI (Compare Inmediate)** : utiliza direccionamiento inmediato para fuente. Para el operando destino puede utilizar todos los modos salvo direccionamiento directo a registro de direcciones, indirectos con predecremento o postincremento y los relativos a PC

p) **CMPM (Compare Memory)** : compara dos datos en memoria con direccionamiento indirecto con post-incremento facilitando así la comparación de bloques de memoria

q) **CLR (Clear)** : carga un cero binario en el operando destino poniendo los flags N a 0, Z a 1, V a 0 y C a 0

r) **NEG (Negate)** : esta instrucción devuelve el complemento a 2 del operando y esto supone el cambio aritmético de signo. Afecta a todos los flags del CCR

s) **NEGX (Negate Extended)** : facilita el cambio de signo de valores de precisión múltiple

t) EXT (Extend) : cuando se quiere ampliar la longitud de un dato con signo, por ejemplo de byte a palabra o de palabra a palabra larga , sin alterar su valor, el dato original debe mantenerse en el byte o palabra de menor peso, y el byte o palabra que se añade debe tener todos sus bits con el mismo valor que el bit de signo del dato original, recibiendo esta operación el nombre de extensión de signo

4.4. Instrucciones lógicas

El Motorola 68000 dispone de cuatro instrucciones que realizan funciones lógicas, que actúan bit a bit sobre datos de 8, 16, ó 32 bits y cuatro para manejar bits individuales sobre datos de 8 ó 32 bits.

a) **AND** : es la Y lógica. Admite todos los modos de direccionamiento para el operando fuente menos direccionamiento directo a registro de direcciones, mientras que el operando destino admite también todos menos los direccionamientos relativos a PC

b) **ANDI** : realiza la misma función que AND, pero su direccionamiento es inmediato en el operando fuente y todos menos el direccionamiento directo a registro de direcciones y relativos a PC

c) **EOR** : es el O exclusivo. Admite únicamente el direccionamiento directo a registro de datos para el operando fuente, mientras que el operando destino admite también todos menos los direccionamientos directo a registro de direcciones y los relativos a PC

d) **EORI** : realiza la misma función que EOR, pero su direccionamiento es inmediato en el operando fuente y todos menos el direccionamiento directo a registro de direcciones y relativos a PC

e) **NOT** : complementación lógica.

f) **OR** : es la O lógica. Admite todos los modos de direccionamiento para el operando fuente menos direccionamiento directo a registro de direcciones, mientras que el operando destino admite también todos menos los direccionamientos relativos a PC y directo a registro de direcciones

g) **ORI** : realiza la misma función que OR, pero su direccionamiento es inmediato en el operando fuente y todos menos el direccionamiento directo a registro de direcciones y relativos a PC

h) **TST** : comprueba un operando. Los flags V y C se ponen a 0

i) **SCC** : comprueba los códigos de condición y pone a 1 el operando. Es de tamaño byte

4.5. Instrucciones de desplazamiento y rotación

Se caracterizan por desplazar o rotar el operando bit a bit a la derecha o a la izquierda. El operando destino, que es el afectado por el desplazamiento o por la rotación siempre será un registro de datos.

a) **ASL (Arithmetic Shift Left)** : desplaza a la izquierda los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag V se pone a 1 si el bit más significativo cambia en algún momento y C es el valor del último bit desplazado fuera del operando destino

b) **ASR (Arithmetic Shift Right)** : desplaza a la derecha los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag V se pone a 0, y el C es el valor del último bit desplazado fuera del operando destino

c) **LSL (Logical Shift Left)** : es el desplazamiento lógico a la izquierda. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino

d) **LSR (Logical Shift Right)** : es el desplazamiento lógico a la derecha. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino

e) **ROL (Rotate Left)** : rota a la izquierda los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino

f) **ROR (Rotate Right)** : rota a la derecha los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino

g) **ROXL (Rotate with Extended Left)** : rotación a la izquierda con extensión. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino

h) **ROXR (Rotate with Extended Right)** : rotación a la derecha con extensión. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino

i) **SWAP** : intercambia el contenido de los 16 bits más significativos con el de los 16 menos significativos, ya que los operandos son de tamaño palabra. Los flags V y C se ponen a 0

4.6. Instrucciones de manipulación de bits

El Motorola 68000 permite comprobar, poner a cero , poner a uno e invertir los bits individuales de un valor entero.

a) **BTST (Bit Test)** : sirve para comprobar el estado de un bit concreto de destino. Actualiza el flag Z en función del valor del bit indicado en la instrucción (Z=1 si el bit es cero). Admite el direccionamiento inmediato y el directo a registro de datos en el operando fuente y en el operando destino todos menos el directo a registro de direcciones

b) **BCLR (Bit Clear)** : pone a 0 el bit indicado. Actualiza el flag Z en función del valor original del mismo, poniéndolo posteriormente a cero

c) **BSET (Bit Set)** : igual que BCLR, pero poniendo el bit a 1

d) **BCHG (Bit Change)** : en primer lugar actualiza el flag Z en función del valor del bit indicado y luego complementa el bit, es decir lo pone en el valor (0 ó 1) contrario al original

4.7. Instrucciones de operación en código BCD

De la misma manera que el Motorola 68000 opera con enteros, también permite programar la suma y la resta en código BCD.

a) **ABCD** : suma fuente al destino

b) **NBCD** : niega el destino

c) **SBCD** : resta fuente al destino

4.8. Instrucciones de ramificación y salto

Este microprocesador incorpora varios mecanismos para poder realizar instrucciones típicas de los lenguajes de alto nivel, como pueden ser los bucles o las instrucciones de condición, siendo el más elemental la instrucción de ramificación, que utiliza como operando una etiqueta, y que sirve para hacer que la próxima instrucción que se ejecute sea la que tenga dicha etiqueta. Cuando el procesador se encuentra con esta instrucción, sencillamente carga la etiqueta en el contador de programa, por lo que la etiqueta es la dirección de donde debe cargarse la próxima instrucción que vaya a ejecutarse.

Este tipo de instrucción presenta tres variantes:

a) **Bcc <etiqueta>** : bifurcación a <etiqueta> cuando se cumple la condición dada por "cc"

b) **BRA <etiqueta>** : bifurcación incondicional

c) **DBcc Dn ,<etiqueta>** : testea la condición "cc", decrementa y bifurca

De estas tres instrucciones, las dos primeras se encuentran disponibles en todos los microprocesadores. La última de ellas es más inusual y puede no estar disponible en todos los modelos de la familia M68000. Detallaremos a continuación cada uno de las anteriores variantes:

a) **Bcc (Branch on condition code)** : utilizan un único argumento que indica la dirección de la siguiente instrucción en el caso de que se cumpla la condición indicada por "cc". Utiliza siempre direccionamiento relativo al contador de programa (PC). Eso implica que existe un rango máximo de salto. Este máximo viene dado un numero de 8 o 16 bits desde el comienzo de la instrucción que sucede al salto. Este desplazamiento puede ser tanto positivo como negativo (avance y retroceso). Generalmente es el propio ensamblador el que calcula el valor del salto y a su vez, decide si dicho valor necesita 16 bits o únicamente 8. Sin embargo podemos forzar que las direcciones sólo sean de un byte con el sufijo .S (Bcc.S <etiqueta>). Las letras "cc" representan el sufijo que condiciona la bifurcación. La siguiente tabla contiene lo diferentes códigos de salto según las posibles opciones de "cc" variables detalladas en la siguiente tabla

MNEMONICO - cc	*CONDICION DE RAMIFICACIÓN*
CC (Carry Clear)	C = 0
CS (Carry Set)	C = 1
NE (Not Equal)	Z = 0
EQ (EQual)	Z = 1
PL (PLus)	N = 0
MI (MInus	N = 1
HI (Higher than)	(C = 0)(Z = 0) = 1
LS (Lower/same as)	(C = 1)+(Z = 1) = 1
GT (Greater than)	[(N,V=1)(Z=0)]+[(N,V,Z=0)] = 1
LT (Less than)	[(N=1)(V=0)]+[(N=0)(V=1)] = 1
GE (Greater Than)	[(N=1)(V=0)]+[(N=0)(V=1)] = 0
LE(Less than or Equal to)	[(Z=1)+(N=1)(V=0)+(N=0)(V=1)] = 1
VC (oVerflow Clear)	V = 0
VS (oVerflow Set)	V = 1
BT (always True)	Siempre
BF (always False)	Nunca

b) **BRA (BRAnch)** : bifurcación incondicional, que al igual que los anteriores, emplea direccionamiento relativo al PC. Su único argumento es el desplazamiento de 8 ó 16 bits que se suma (con signo) al PC y se obtiene la dirección de la siguiente instrucción a ejecutar.

c) **JMP (JuMP)** : Esta instrucción es equivalente a BRA, cuya única diferencia respecto a JMP estriba en que BRA emplea direccionamiento relativo al PC y JMP puede emplear los siguientes :

   a. JMP (An)
   b. JMP d16(An)
   c. JMP d8(An, Xi)
   d. JMP Dirección_absoluta
   e. JMP d16(PC)
   f. JMP d8(PC, Xi)

d) **DBcc (Decrement and Branch on Condition Code)** : facilita la realización de bucles en un programa, y tiene dos argumentos siendo el primero un registro de datos (Dn) y el segundo un desplazamiento de 16 bits respecto al PC. Como en el caso de Bcc, hay 14 instrucciones implementadas (ver tabla de mnemónicos en la descripción de Bcc). Su funcionamiento se detalla a continuación :

   a. Se comprueba la condición cc : si se cumple dicha condición, el salto no tiene lugar y la siguiente instrucción es ejecutada (es pues, el efecto contrario a las instrucciones Bcc)
   b. Si por el contrario la condición no se cumple, los 16 bits menos significativos de Dn son decrementados en 1 unidad. Si el resultado contenido en Dn es igual a –1, la siguiente instrucción de la secuencia es ejecutada, en caso contrario, se produce la bifurcación.

4.9. Instrucciones de manejo de subrutinas

Como casi cualquier microprocesador, el M68000 posee implementado por hardware un mecanismo de subrutinas que se apoya en el empleo de la pila (registro A7).

a) **BSR/JSR: (Branch to SubRutine / Jump to SubRutine)** : el operando asociado con estas instrucciones debe ser la dirección de memoria en la que se comienza la subrutina, con direccionamiento absoluto para JSR, y relativo a PC para BSR. Al ejecutarse cualquiera de las dos instrucciones, se almacena automáticamente en la pila el valor del contador de programa en el momento anterior al salto, cuando su contenido apunta a la dirección de comienzo de la instrucción siguiente a JSR o BSR.

b) **RTS / RTR (Return from Subrutine / Return and Restore condition codes)** : su uso facilita la salida de las subrutinas. Son la última instrucción que se ejecuta en aquella antes de volver al programa que llama a dicha rutina. Ambas recuperan de la pila el valor del contador del programa, lo que permite reanudar la ejecución del programa precisamente en el punto que había sido abandonado. Además, la instrucción RTR también repone el CCR extrayendo una palabra de la pila inmediatamente antes de recuperar el PC. La subrutina debe llevar a la pila, justamente encima de las posiciones que contienen la posición de retorno, una palabra cuyos 5 bits menos significativos serán llevados al CCR al ejecutarse RTR, que puede ser una copia del CCR al entrar en la subrutina, o cualquier otro valor.

4.10. Otras instrucciones

a) **Scc (Set byte conditionally)** : Existen tantas instrucciones de es tipo como vienen dadas por la tabla de valores de "cc". Su sintaxis completa es Scc <ea> , donde <ea> es una dirección efectiva. Cuando una Scc es encontrada por el ensamblador, se evalúa la condición y si ésta se cumple, fija todos los bits del byte especificado por <ea> a uno, y de no cumplirse, se fijan todos a cero. Es decir, el byte resultante es $FF si la condición se cumplía y $00 en caso contrario.

b) **CHK (CheckK)** : Su sintaxis es CHK <ea>,Dn y chequea la palabra corta de menor peso del registro de datos Dn. Si su valor está contenido entre los límites 0 < Dn < [ea] continua la ejecución de la siguiente instrucción de la secuencia. En caso contrario, una tiene lugar un llamada al sistema operativo.

c) **NOP (No Operation)** : no realiza absolutamente nada. Es una instrucción que no hace sino consumir tiempo, exactamente 4 ciclos de reloj. Puede parecer absurda la existencia de una instrucción así, pero tiene su justificación, como por ejemplo si se desea efectuar un pequeño retraso en la ejecución de un programa con el fin de sincronizar determinados acontecimientos, o cuando se desea reservar algunas posiciones del programa para poder intercalar nuevas instrucciones, etc.

d) **STOP** : Su sintaxis es STOP #n y provoca la detención de la ejecución del programa de forma controlada y en el punto deseado, devolviendo el valor de n (de 16 bits) en registro de estado. Es una instrucción privilegiada.

Otras instrucciones existentes en el M68000 que por ahora no serán detalladas son:

a) RESET
b) RTE
c) TAS
d) TRAPV

4.11. Gestión de subrutinas

Una subrutina es un conjunto de instrucciones que realizan una tarea concreta, que no puede ser ejecutada directamente sino que debe ser llamada por un programa principal. Esta subrutina denominada también subprograma, procedimiento o simplemente rutina se comienza a ejecutar cuando es llamada por el programa principal, desde la primera instrucción. Cuando se ha llegado a la última instrucción, se vuelve a ejecutar la siguiente instrucción del programa principal anterior a la invocación de la subrutina.

La gestión de las subrutinas se realiza mediante una estructura de almacenamiento de tipo de pila. El espacio en memoria que se reserva para la pila se define a partir de dos direcciones, la dirección de la posición inicial o fondo de la pila y la dirección de la posición máxima permitida o valor máximo que puede alcanzar la cima de la pila. El puntero de pila indica la posición de la cima de la pila, utilizando un registro de direcciones denominado SP (stack pointer) y es el A7.

CAPITULO 3

Programación del 68000 en lenguaje ensamblador

1. ¿Qué es un lenguaje ensamblador?
2. ¿Qué es un programa ensamblador?
3. Estructura y sintaxis de un lenguaje ensamblador.
4. Directivas del ensamblador.
5. Ejemplos.

1. ¿Qué es el lenguaje ensamblador?

El lenguaje ensamblador surge con la idea de evitar las dificultades que presenta el trabajar en lenguaje máquina, siendo la misión de este la de simplificar la programación en un computador y teniendo a la vez un control directo sobre el hardware del mismo.

El lenguaje ensamblador es un lenguaje cuyas estructuras de datos se corresponden con las estructuras físicas de los registros de la memoria principal del computador para el que está pensado, y cuyas instrucciones tienen una relación directa y biunívoca con las instrucciones del lenguaje máquina.

2. ¿Qué es un programa ensamblador?

Un programa ensamblador es un programa que traduce un texto escrito en lenguaje ensamblador de un determinado computador al lenguaje máquina de ese mismo computador, y proporciona las facilidades necesarias para simplificar la tarea de desarrollar programas en lenguajes de bajo nivel.

También, un programa ensamblador proporciona una interfaz adecuada entre el programador y la arquitectura del computador, para las tareas de programación. Si el ordenador que se utiliza para esta tarea es el mismo, o tiene la misma CPU, que el sistema que va a ejecutar el código máquina resultante de la traducción, se dice que el programa ensamblador utilizado es un auto-ensamblador o ensamblador residente, mientras que si utiliza otra CPU diferente es un ensamblador cruzado (cross assembler).

El ordenador sobre el que corre el programa ensamblador tendrá en su memoria central el programa fuente codificado en caracteres alfanuméricos y el propio programa ensamblador que se está ejecutando, debiendo reservar en memoria, además, un espacio para ir almacenando el código resultante de la traducción. El programa fuente se encuentra almacenado en una zona de memoria, llamada buffer, en forma de caracteres alfanuméricos, ocupando cada carácter un byte. El código resultante del proceso se denomina código objeto, y queda almacenado en otro lugar de la memoria. Además se utiliza otro tercera zona de memoria destinada a almacenar la tabla de símbolos. La tabla de símbolos no es sino un pequeño diccionario que construye el programa ensamblador en el que constan todas las etiquetas utilizadas junto con sus

equivalencias numéricas. La tabla de símbolos es temporal y sirve al propio programa ensamblador a la hora de efectuar la traducción. Una vez ensamblado el programa, la tabla no tiene otra utilidad que la de su posible consulta por parte del programador durante el proceso de depuración. El programa ensamblador actúa generalmente en dos pasos: durante el primero lee el código fuente y anota en la tabla de símbolos las etiquetas que se encuentran, calculando, si es necesario su equivalencia; en el segundo paso se apoya en la tabla de símbolos y, mediante otra lectura del programa fuente, va traduciendo, instrucción por instrucción y los códigos mnemónicos que encuentra.

Generalmente, los programas ensambladores están dotados de la posibilidad de detectar errores y dan el aviso correspondiente cuando encuentran un mnemónico inexistente, un operando fuera de margen, una etiqueta definida dos veces, etc.

3. Estructura y sintaxis de un lenguaje ensamblador

La sintaxis de un lenguaje ensamblador es el conjunto de reglas que debe guardar el programa fuente y que estará compuesto por una serie de instrucciones, distinguiéndose cuatro campos: etiqueta, mnemotécnico, operando y comentario. El programa fuente deberá estar en un fichero ASCII que se genera con la ayuda de un programa editor, y no un procesador de textos, ya que estos generan códigos de control que los ensambladores no son incapaces de interpretar. Un sencillo editor es el de MS-DOS. Las instrucciones en código máquina se codifican por campos, por lo que las instrucciones escritas en ensamblador se codificaran también por campos.

LINEA DE INSTRUCCION EN ENSAMBLADOR M68000			
ETIQUETA	**NEMOTECNICO**	**OPERANDO**	**COMENTARIO**
MOVE_INS	*MOVE.L*	*D0,-(SP)* ;	*Guarda D0*

3.1. Campo de etiqueta:

Es un campo opcional que se utiliza para tener una referencia de las instrucciones. El programa ensamblador va traduciendo secuencialmente las instrucciones del programa fuente, guardando los códigos traducidos en posiciones consecutivas de memoria. Cuando encuentra una etiqueta en una línea de instrucción, el ensamblador guarda dicha etiqueta en una tabla especial en la memoria, junto con la dirección de memoria en la que se ha almacenado la instrucción que la acompaña, así si alguna instrucción tiene que referenciar a la instrucción etiquetada, bastará con que se escriban los caracteres de la etiqueta en el campo de operandos correspondiente. El programa ensamblador se encargará, cuando deba traducir esta instrucción, de buscar en la tabla la dirección en memoria de la instrucción correspondiente a la etiqueta.

Las ventajas del empleo de etiquetas son:

a) Las etiquetas permiten localizar y recordar fácilmente una determinada instrucción.

b) Se puede modificar de forma sencilla, en la fase de corrección de un programa, el punto hacia donde tienen que realizarse uno o más saltos, simplemente cambiando la etiqueta de una instrucción a otra.

c) En caso de que el programa objeto se deba colocar en posiciones de memoria diferentes de las que en un principio se habían previsto, si se han asignado etiquetas, no hay que hacer ningún cambio en el programa fuente pues al hacer la nueva traducción, el programa ensamblador asigna automáticamente las nuevas posiciones a las etiquetas. Así facilita la unión de varios programas que hayan sido desarrollados por separado.

d) El programador se libera de la tarea de calcular direcciones.

Todos los ensambladores imponen algunas reglas y limitaciones en la utilización de etiquetas, siendo estas en el caso del M68000 las siguientes:

a) Únicamente son significativos los primeros quince caracteres de la etiqueta

b) Los caracteres pueden ser cualquier código ASCII mayor que 32 (espacio), salvo + - / & ! < ( ) ^ |

c) El primer carácter no puede ser un número, ni los símbolos $ y %

d) Mayúsculas y minúsculas se consideran caracteres diferentes

3.2. Campo de mnemotécnico:

Se utiliza para escribir los códigos de instrucciones ejecutables y los códigos de pseudoinstrucciones o directivos de ensamblador. Las instrucciones ejecutables son los mnemotécnicos que constituyen las instrucciones del computador.

Las pseudoinstrucciones sirven para dar al ensamblador indicaciones como la dirección de memoria a partir de la cual debe ir guardando los códigos traducidos, realizar la reserva de las posiciones de memoria donde deben guardarse los resultados, etc. Las pseudoinstrucciones reciben este nombre porque sus mnemotécnicos no se convierten en código máquina, sino que son ejecutados directamente por el programa traductor.

3.3. Campo de operandos:

Se utiliza para indicar los valores concretos de los operandos que intervienen en la operación definida por el campo mnemotécnico. Dependiendo de la instrucción indicada en el campo mnemotécnico, habrá que definir el modo de direccionamiento, que estará determinado generalmente, mediante una combinación del código de operación y la información del campo de operandos :

a) **Números y constantes alfabéticas**: Las cantidades colocadas en el campo de operandos pueden venir expresadas en diferentes bases de numeración, que son: binaria, decimal y hexadecimal, y van determinados por un símbolo que se añade al valor numérico

BASE DE NUMERACIÓN	*SÍMBOLO*
Decimal	por defecto
Binario	%
Hexadecimal	$

b) **Símbolos** : El conjunto de caracteres alfanuméricos que se utiliza en el campo de los operandos para indicar una dirección recibe el nombre de símbolo. El programador podrá definir sus propios símbolos mediante dos procedimientos:

   a. De forma implícita, mediante el empleo de etiquetas
   b. De forma explícita, mediante algunas pseudoinstrucciones

c) **Expresiones** : Una expresión es una combinación de números y/o símbolos unidos mediante operadores aritméticos o lógicos

3.4. Campo de Comentarios:

Mediante el campo de comentarios se trata de hacer más comprensivo al programa, incluyendo en ellos todos los puntos claves, definición de símbolos, reservas de memoria, propósito del programa, etc.

4.- Directivas de ensamblador o pseudoinstrucciones

Las pseudoinstrucciones dan al ensamblador indicaciones como la dirección de memoria a partir de la cual debe ir colocando los códigos traducidos, realizar la reserva de las posiciones de memoria donde deben guardarse los resultados, etc. Se utiliza el campo mnemotécnico para transmitir esta información al programa traductor y darle las indicaciones necesarias para que éste pueda realizar la traducción correctamente. Por lo que debe reservarse un conjunto de mnemotécnicos para este fin los cuales no se convierten en código máquina, sino que son ejecutados directamente por el programa traductor. Las directivas del M68000 son las siguientes :

4.1. Definición de símbolos (EQU).

Definir un símbolo consiste en asignar un valor como un número o una dirección de memoria a un nombre utilizando el directivo EQU.

Ejemplo:

```
IMPRESORA  EQU  128
```

Cuando el traductor lee este directivo, guarda la etiqueta y el valor indicado en el operando, en una zona de memoria denominada tabla de símbolos. Al encontrar un símbolo en el campo de operandos el traductor consultará esta tabla para ver el valor que le corresponde.

4.2. El contador de dirección de ensamblado (ORG).

El programa ensamblador utiliza un registro que le indica en que posición de memoria debe guardar una instrucción y que dirección debe asignar a una determinada etiqueta. Este registro se llama contador de dirección de ensamblado. Para poner el contenido de este contador a un valor determinado se usa una pseudoinstrucción denominada ORG.

Ejemplo:

```
ORG  $1500
```

Pone el número $1500 en el contador de dirección de ensamblado

4.3. Definición de constantes (DC).

La sentencia DC permite definir una posición de memoria conteniendo un valor determinado que se colocarán en orden a partir de la posición señalada. Acepta los modificadores .B, . W y .L

Ejemplo :

```
DC.B 10,66

DC.W $0A1234

DC.B "RaSHGaRoC"
```

El primero de los ejemplos sitúa consecutivamente en memoria 10 y 66, guardándolos como bytes. En el segundo el número es guardado como palabra de 16 bits y en la última, una sucesión de bytes son almacenados conformando una cadena.

4.4. Definición de datos (DS).

La sentencia DS reserva el espacio de memoria solicitado sin especificar su contenido, y su longitud viene dada por el valor del operando , viéndose afectado este por los modificadores de tamaño .B, .L y .W.

Ejemplo:

```
ORG 1000

DS.W    4
```

Mediante estas instrucciones hemos reservado 8 bytes, a partir de la dirección 1000 (cuatro elementos de 2 bytes)

4.5. Última sentencia del programa (END).

La sintaxis de este directivo es muy simple ya que consta sólo de un campo de mnemotécnico, END. Este directivo únicamente se usa una vez en cada programa y será necesariamente la última instrucción, indicando que cese de ensamblar.

4.6. Título del listado (TTL).

Esta directiva proporciona el título del listado del código que es usado en la cabecera cuando un programa es impreso.

4.7. Fijar el contador de programa a cero (SECTION).

Se fija el contador de programa a cero como si fuese equivalente a ORG $000000. Su propósito es el de forzar al ensamblador a generar modos de direccionamiento relativos al contador de programa independientes de la situación del código.

5. Ejemplos.

EJEMPLO 1

Realizar un programa que dado un número decimal almacenado en un byte como máximo se obtenga su representación en formato ASCII almacenada en 32 bits, sabiendo que

Decimal	ASCII
0	$30
1	$31
...	...
9	$39

SOLUCION.

El algoritmo que he diseñado que resuelve este problema es el siguiente:

```
*****************************************************************
* BIN2ASC.ASM (Sistema Microinstructor TM-683)
* Conversión de binario a ASCII
*****************************************************************

        absolute
        org $25000

main    move.b #111,D0 ;Dato=255
        bsr    Bin2Asc
fin     bra.s  fin

*****************************************************************
* Bin2Asc: Conversión de binario (8 bits) a ascii (4 caracteres)
* D0 - Dato binario
* D1 - Dato ascii
*****************************************************************
Bin2Asc andi.l #$FF,D0
        divu #100,D0
        clr.l D3
        move.b D0,D3
        lsl.l #16,D3
        swap D0
        andi.l #$FF,D0
        divu #10,D0
        move.b D0,D3
        asl.w #8,D3
        swap D0
        move.b D0,D3
        add.l #$30303030,D3
        rts
        end
```

EJEMPLO 2

Realizar un programa que dado un número en formato ASCII almacenado en 32 bits, obtenga su valor decimal.

SOLUCION.

El algoritmo que nosotros proponemos es el siguiente:

```
*****************************************************************
* ASCI2BIN.ASM (Sistema Microinstructor TM-683)
* Conversion de ascii a binario
*****************************************************************
        ABSOLUTE
        ORG $25000
MAIN    MOVE.L #$30313937,D2 ;CARGAMOS EL 197 A D2
        BSR ASCII2BIN
FINAL   BRA.S FINAL
*****************************************************************
* D2=NUMERO A CONVERTIR
* D0=RESULTADO
*****************************************************************

ASCII2BIN       MOVEM.L D1-D2,-(SP) ;MOVER REGISTROS A PILA
                SUB.L #$30303030,D2 ;NUMEROS DECIMALES EN D2
                CLR.L D0
                MOVE.B D2,D0 ;UNIDADES A D0
                CLR.L D1
                ASR.L #8,D2
                MOVE.B D2,D1 ;DECENAS A D1
                MULU #10,D1
                ADD.B D1,D0  ;DECENAS MAS UNIDADES A D0
                CLR.L D1
                ASR.L #8,D2
                MOVE.B D2,D1 ;CENTENAS A D1
                MULU #100,D1
                ADD.B D1,D0 ;NUMERO DECIMAL A D0
                MOVEM.L (SP)+,D1-D2 ;RECUPERACION PILA
                RTS
                END
```

EJEMPLO 3

Realizar un programa que devuelva la longitud de una cadena que comienza en la posición conocida y denominada *string* y su final está marcado por el retorno de carro, cuyo código ASCII es $0D.

SOLUCION.

Una posible solución sería el siguiente algoritmo:

```
*****************************************************************
* LONGITUD.ASM (Sistema Microinstructor TM-683)
* Medida de la longitud de una cadena de caracteres terminada con CR
*****************************************************************

        absolute
        org $25000

main    lea.l string,A6
        bsr loncad
final   bra.s final

*****************************************************************
* A6 - direccion de comienzo de la cadena
* D0 - devuelve la longitud de la cadena
*****************************************************************
loncad  clr.l D0
comp    cmpi.b #$0D,0(A6,D0)
        beq fin
        addi.b #1,D0
        bra.s comp
fin     rts
string  dc.b    'Soy un string',$0D
        end
```

EJEMPLO 4

Realizar un programa que elimine dentro de una cadena de caracteres los ceros a la izquierda por espacios, sabiendo que en código ASCII el cero el $30 y el espacio el $20.

SOLUCION.

El algoritmo que nosotros proponemos es el siguiente:

```
*****************************************************************
* NOCEROS.ASM (Sistema Microinstructor TM-683)
* Sustituye por espacios los ceros a la izquierda de un string
*****************************************************************
        absolute
        org $25000
main    lea.l string,A6
        bsr quitaceros
final   bra.s final
*****************************************************************
* A6 - dirección de comienzo del string
*****************************************************************
quitaceros      clr.l D0
comparar        cmpi.b #$30,0(A6,D0)
                bne fin
                move.b #$20,0(A6,D0)
                addi.b #1,D0
                bra.s comparar
fin             rts
string          dc.b    '0000250320'
                end
```

EJEMPLO 5

Realizar un programa que dadas dos cadenas de caracteres, conocida su longitud, determine si son iguales o no lo son.

SOLUCION

A continuación proponemos un algoritmo que resolvería este problema, indudablemente existen más:

```
*************************************************************
* STRCOMP.ASM (Sistema Microinstructor TM-683)
* Compara dos cadenas de caracteres
*************************************************************

        absolute
        org $25000


main    lea.l string1,A6
        lea.l string2,A5
        move.w #longi,D2
        bsr compstr
        lea.l string1,A6
        lea.l string2,A5
        move.w #longi-1,D2
        bsr compstr
final   bra.s final


*************************************************************
* A6 - direccion de comienzo de string1
* A5 - direccion de comienzo de string2
* D2 - longitud a comparar
* D3 - resultado: $FF distintas
*                 $00 iguales
*************************************************************
compstr clr.l D0
        subq.b #1,D2
        cmpi.b #0,D2
        beq fin
comp    cmpm.b (A6)+,(A5)+
        bne casif
        dbf D2,comp
        move.b #$00,D3
        bra.s fin
casif   move.b #$FF,D3
fin     rts
string1 dc.b    'Cadena 1'
string2 dc.b    'Cadena 2'
longi   equ     string2-string1
        end
```

EJEMPLO 6

Realizar un programa que encuentre el valor máximo de una tabla donde se almacenan números sin signo y cada uno de ellos ocupa una palabra.

SOLUCION.

El algoritmo que se propone en este caso es el siguiente:

```
*****************************************************************
* MAXIMO.ASM (Sistema Microinstructor TM-683)
* Determina el maximo entre las palabras de una tabla
*****************************************************************

        absolute
        org $25000
main    lea.l tabla,A6
        move.w #cont,D0
        bsr maximo
final   bra.s final

*****************************************************************
* D0 - tama¤o de la tabla
* D1 - elemento maximo
* D2 - indice
*****************************************************************
maximo  move.w #0,D1
        move.w 0,D2
comp    cmp.w 0(A6,D2),D1
        bhi posic
        move 0(A6,D2),D1
posic   addi.w #2,D2
        dbf D0,comp
        rts
tabla   dc.w    $12A2,$32,$9025,$A478,$4A,$355B
ftabla  dc.w    $3500
cont    equ     (ftabla-tabla)/2
        end
```

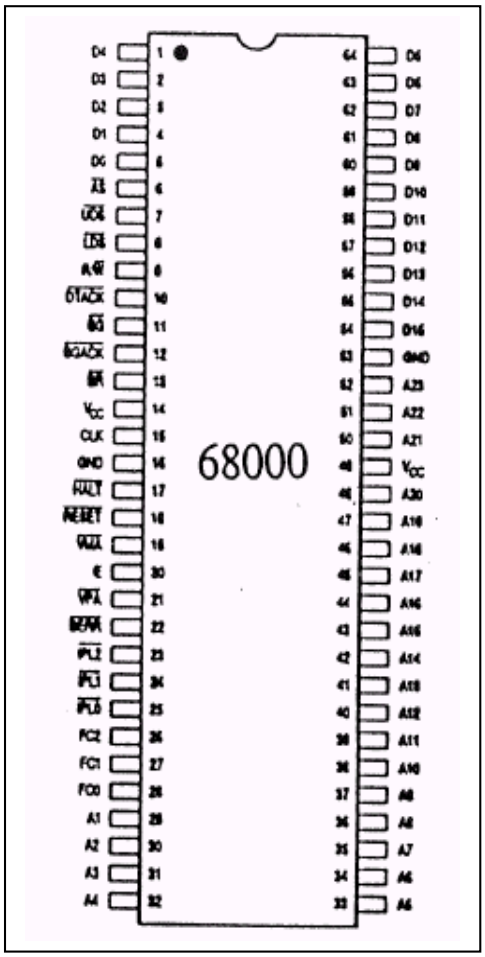
CAPITULO 4

Organización externa del 68000

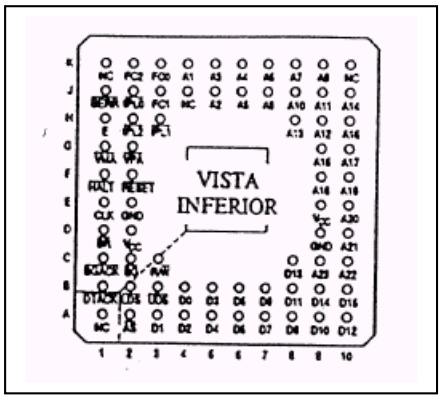
1. Encapsulados.
2. Descripción de las señales del 68000.
3. Ciclos del bus.

1. Encapsulados.

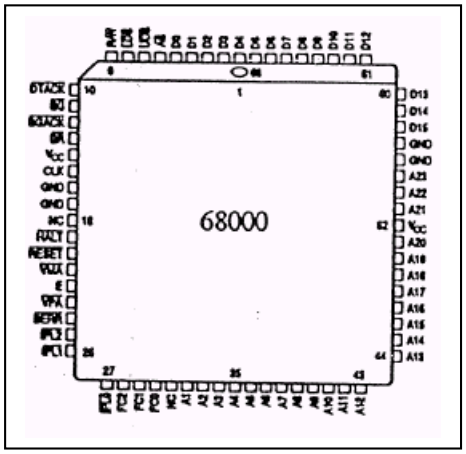
En el mercado el 68000 de Motorola lo encontramos con tres encapsulados distintos que son:



Encapsulado DIL

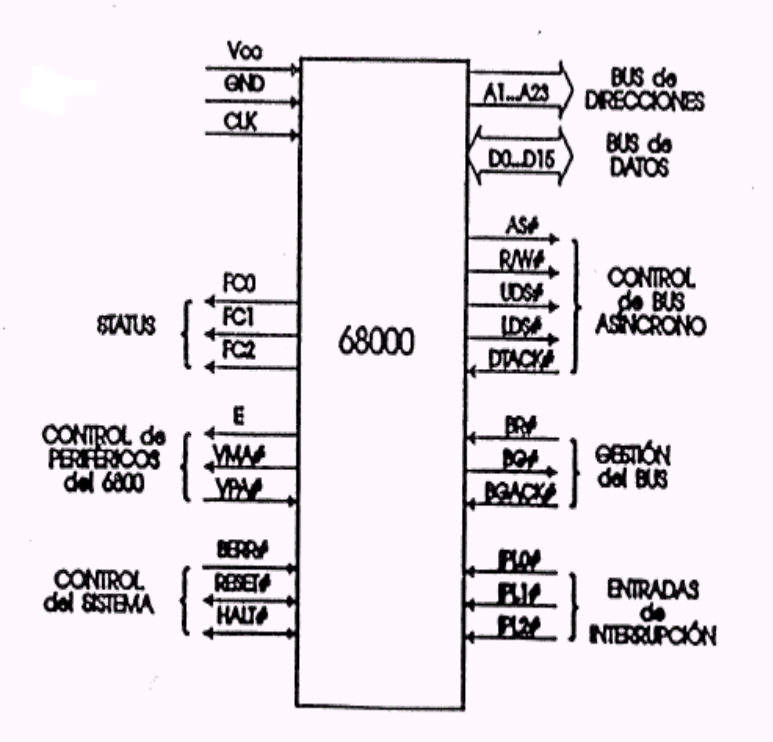


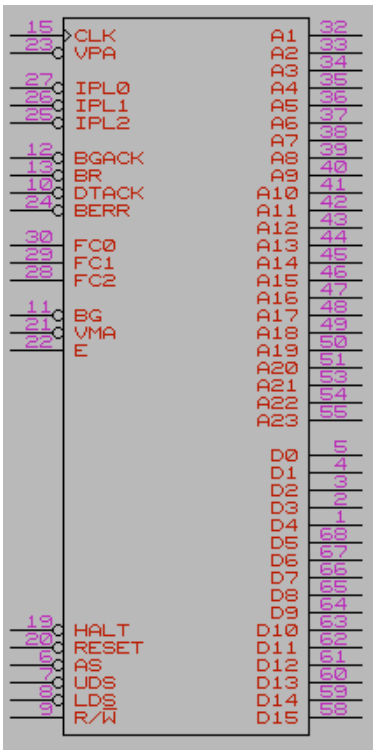
Encapsulado PGA



Encapsulado QUAD

Además en la siguiente figura podemos observar las principales patillas del 68000 agrupadas de forma funcional.


Vcc
GND
CLK
68000
A1...A23
BUS de DIRECCIONES
D0...D15
BUS de DATOS
AS#
R/W#
UDS#
LDS#
DTACK#
CONTROL de BUS ASÍNCRONO
STATUS
FC0
FC1
FC2
CONTROL de PERIFÉRICOS del 6800
E
VMA#
VPA#
BR#
BG#
BGACK#
GESTIÓN del BUS
CONTROL del SISTEMA
BERR#
RESET#
HALT#
IPL0#
IPL1#
IPL2#
ENTRADAS de INTERRUPCIÓN


15 CLK
23 VPA
27 IPL0
26 IPL1
25 IPL2
12 BGACK
13 BR
10 DTACK
24 BERR
30 FC0
29 FC1
28 FC2
11 BG
21 VMA
22 E
19 HALT
20 RESET
6 AS
7 UDS
8 LDS
9 R/W
A1 32
A2 33
A3 34
A4 35
A5 36
A6 37
A7 38
A8 39
A9 40
A10 41
A11 42
A12 43
A13 44
A14 45
A15 46
A16 47
A17 48
A18 49
A19 50
A20 51
A21 53
A22 54
A23 55
D0 5
D1 4
D2 3
D3 2
D4 1
D5 68
D6 67
D7 66
D8 65
D9 64
D10 63
D11 62
D12 61
D13 60
D14 59
D15 58

2. Líneas y buses en el 68000.

El 68000 tiene básicamente 3 buses con los que gestiona todo el sistema:

» Líneas especiales:

- Alimentación y masa
- Reloj

» Bus de direcciones

» Bus de datos

» Bus de control: control de transferencias de datos, que se puede subdividir en:

- línea de dirección de transmisión
- líneas de selección de dirección par e impar
- bus asíncrono
- bus síncrono
- control de posesión de los buses
- control del sistema
- control del estado del microprocesador
- control de las interrupciones

A continuación comentaremos cada uno de ellos.

2.1. Líneas especiales.

2.1.1. Alimentación y masa.

Todo sistema eléctrico requiere una alimentación para su funcionamiento, y el 68000 no puede ser una excepción.

VCC: Voltage CCollector - Tensión por el colector. Línea de entrada.

Esta línea se utiliza para alimentar todo el microprocesador. El 68000 posee solo una (otros micros tienen varias, al tener que alimentar muchos transistores) y su rango es de -0,3v a +7v. Se ha de preveer también el paso de corriente de hasta 1.5 A.

Esto no son más que unos valores puntuales y en realidad varían según el modelo que se utilice.

GND: GrouND - Masa

Este es el terminal al que se conecta la masa de la fuente de alimentación.

La VCC y la GND se encuentran distribuidas en el micro de forma simétrica para mantener la compatibilidad electromagnética, es decir, se colocan de esta forma para evitar que el campo electromagnético que producen afecte al resto de entradas y de salidas del 68000.

2.1.2. Reloj.

CLK: CLocK – Reloj. Línea de entrada

Esta línea se utiliza para suministrar al microprocesador la sincronización interna que él necesita para trabajar correctamente, tanto en lo referente operaciones de bus y ejecución de instrucciones.

En efecto, todas las acciones que el dispositivo efectúa se ven sometidas a una rígida temporización, sin la cual sería imposible que los procesos se ejecutaran en el instante requerido.

La señal que se introduzca debe ser perfectamente cuadrada (ciclo de trabajo del 50%) y su frecuencia no debe de superar la frecuencia nominal que el fabricante indica para el modelo utilizado.

2.2. Bus de direcciones.

Este bus se utiliza para direccionar las zonas de memoria y los dispositivos (que recordemos son tratados como si de posiciones de memoria se tratasen), de forma que, al escribir una

dirección en el bus, cierto dispositivo quede activado y sea quien reciba-envíe los datos en el ciclo de bus así empezado.

Es un bus triestado unidireccional, por lo que puede ponerse en alta impedancia e ignorar lo que ocurre en el exterior (al tiempo que no influir en el estado de las líneas), pero solo permite la escritura del bus. Esto último es razonable, puesto que la lectura del bus de direcciones no es de utilidad para el uP, al ser él mismo quien gestiona el direccionamiento.

Esta compuesto de 24 líneas (A23 - A0), por lo que puede direccionar 2^24 posiciones de memoria: un total de 16Mbytes.

De estas 24 líneas, cabe destacar que A0 se desdobla internamente en /UDS y /LDS cuando se trata de acceder a celdas de tipo byte.

Una forma simple de ver la situación que se crea es que se puede suponer al 68000 como capaz solamente de direccionar 8Mpalabras (16 bits) mediante la líneas A23 - A1. Para ayudarle entonces en el cometido de direccionar bytes se añade una línea suplementaria, A0, que permite seleccionar la parte baja o la alta de la palabra direccionada con el resto del bus. Puesto que una palabra se compone de 2 bytes, se consigue el direccionamiento a nivel de byte. El mapa es pues de 8Mpalabras, lo que equivale a 16Mbytes.

Finalmente, otro uso del bus de direcciones es en el ciclo de reconocimiento de interrupciones. Durante este ciclo, el micro coloca en las líneas A3, A2 y A1 el nivel de interrupción en curso, mientras que el resto son forzadas a un nivel lógico alto.

2.3. Bus de datos del 68000.

El bus de datos lo forman las líneas D15 a D0, siendo este un bus bidireccional de tipo triestado, lo que permite conectar multitud de dispositivos a él sin problemas.

En efecto, cuando el microprocesador quiera no inmiscuirse en los procesos que se estén desarrollando en el bus de datos, coloca este en alta impedancia y vista desde el exterior es como si ahí donde está conectado en uP no hubiese nada.

Se utiliza para efectuar las transferencias de datos entre el microprocesador y el sistema, permitiendo tanto escrituras en el bus como lecturas de este.

Al poseer 16 líneas, permite la transmisión/recepción de 8 bits (1 byte) o bien 16 bits simultáneamente (1 palabra).

Es necesario pues en el caso de palabra larga, 2 transferencias de 16 bits.

Asimismo, en el caso de transferencias (escritura) de tipo byte, el microprocesador coloca automáticamente el dato en la parte baja del bus (D7 - D0) si la celda de memoria destino es una posición impar o en la parte alta (D15 - D8) si es par.

Por otra parte, otro de los usos que se hace del bus de datos es que en un ciclo de reconocimiento de interrupciones, el periférico ha de colocar en la parte baja (D7 - D0) el número de vector correspondiente a la interrupción generada.

2.4. Bus de control.

Estará formado por un conjunto de líneas muy variadas que se encargan de controlar la transferencia de datos. A continuación se describen las principales líneas que los forman.

2.4.1. Línea de dirección de transmisión.

R\/W: Read\/Write - Lectura\/Escritura. Línea de salida, triestado.

En cualquier operación de entrada / salida de datos, se ha de dar a conocer al sistema externo que tipo de transferencia se va a efectuar, ya sea la lectura de un dato o bien la escritura del mismo en algún dispositivo.

Para ello, en el 68000 se varía el nivel de esta línea, indicando así el sentido de la transmisión.

Si R\/W=0, entonces la línea equivale a /W, lo que significa un ciclo de escritura. De esta forma, es el microprocesador el que va a escribir en el bus de datos.

Si R\/W=1, entonces la línea equivale a R, lo que significa un ciclo de lectura. En este caso el microprocesador leerá el estado del bus de datos.

2.4.2. Líneas de selección de dirección par e impar

/UDS - /LDS: /Upper Data Select - /Lower Data Select. Líneas de salida triestado, activas a nivel bajo.

Ya se ha comentado que el 68000 es capaz de direccionar un total de 8Mpalabras.

Puesto que tiene la posibilidad también de direccionar bytes, cada palabra direccionable se subdivide en 2 bytes, dando como resultado 16Mbytes. Para posibilitar la correcta identificación de cada byte en memoria, se parte del hecho de que puesto que cada palabra solo puede ocupar posiciones de memoria pares, al subdividirla en dos bytes uno de ellos tomará una dirección par y el otro una impar.

Para conseguir esta subdivisión se recurre a la línea A0 del bus de direcciones.

En el caso de trabajar con datos de tipo bite, A0 tendrá sentido y se subdividirá internamente en dos líneas, /UDS y /LDS, que indican una dirección par e impar respectivamente.

De esta forma, resulta que:

- A0=0, direccionamiento de un byte par, lo que implica /UDS=0 y /LDS=1
- A0=1, direccionamiento de un byte impar, lo que implica /UDS=1 y /LDS=0

Cuando se acceda a datos de tipo palabra, /UDS y /LDS toman automáticamente el valor 0, sea cual sea el valor de A0.

Queda claro pues que en el caso de datos tipo palabra A0 no tiene sentido y por lo tanto no se usa.

La siguiente tabla esclarece aún más el tema y las posibles combinaciones.

/UDS	/LDS	D15-D8	D7-D0	Tipo de dato
0	0	Datos válidos	Datos válidos	Palabra
0	1	Datos válidos	Datos inválidos	Byte - par
1	0	Datos inválidos	Datos válidos	Byte - impar
1	1	Datos inválidos	Datos inválidos	Los datos son inválidos

2.4.3. Bus asíncrono

Esta parte del bus de control permite la gestión de los dispositivos que trabajen de forma asíncrona en el sistema y se compone de las líneas:

/AS: Adress Strobe - Habilitación de direcciones. Línea de salida, triestado y activa a nivel bajo.

Mediante esta línea, el microprocesador indica que la dirección colocada en el bus de direcciones es válida (es estable) y por tanto el direccionamiento se puede efectuar.

Es necesario pues que esta línea sea considerada al diseñar el circuito de decodificación de los dispositivos y que estos no puedan ser seleccionados hasta que /AS se encuentre activa.

Ejemplo:

El dispositivo X posee una entrada /CS (chip select) que lo habilita. Entonces /CS= /DirMem + /AS, donde /DirMem es la combinación de líneas A23 - A0 que refieren a su dirección en el mapa de memoria.

/DTACK: Data ACKnowledge - Reconocimiento de datos. Línea de entrada, activa a nivel bajo.

Mediante la activación de esta línea, el dispositivo direccionado indica que la transferencia del dato se ha efectuado correctamente.

En un ciclo de lectura, al activarse /DTACK, el uP captura el dato del bus de datos mediante un latch y seguidamente termina el ciclo.

En un ciclo de escritura, la activación de /DTACK indica al uP que el dispositivo externo ha capturado correctamente el dato y acto seguido termina el ciclo.

Funcionamiento:

El control del bus asíncrono se puede explicar de la siguiente forma:

1. El microprocesador valida la dirección colocada en el bus de direcciones activando la línea /AS(=0).
2. Se efectúa el ciclo, donde transcurre la transferencia.

3. El dispositivo direccionado activa /DTACK(=0) indicando el fin de la transferencia, por lo que el uP desactiva /AS(=1) y concluye el ciclo de bus.

Esta forma de comunicación permite ajustar la velocidad de trabajo entre el uP y cada dispositivo conectado al sistema, puesto que estos no tienen porque tener la misma velocidad de proceso.

2.4.4. Bus síncrono (control de periféricos del 6800).

Este grupo de líneas permite la gestión de periféricos síncronos, debiendo estos ceñirse a las características que presentan los dispositivos de la familia 6800. Estos dispositivos forman parte de otra familia de Motorola (la del uP 6800), cuyo principal rasgo es su fuerte sometimiento al sincronismo.

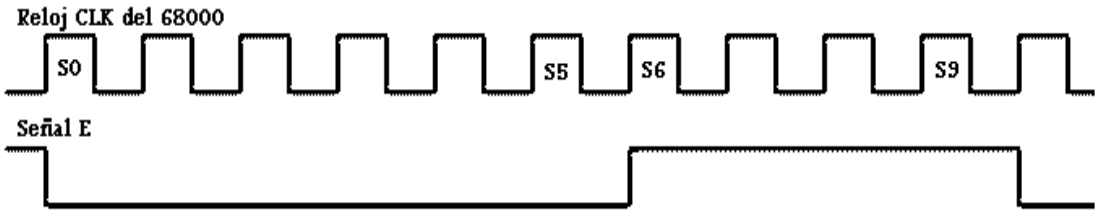
Cabe sin embargo la posibilidad de conectar otro tipo de dispositivos siempre que sean síncronos.

E: Enable – Habilitación. Línea de salida.

Esta es una línea que utilizan los dispositivos de la familia 6800 para su sincronización en las transferencias de datos. Para aquellos dispositivos compatibles con los ciclos de dicha familia desarrolla la misma función.

Independientemente del estado del microprocesador, esta línea sigue variando con el tiempo, a razón de una frecuencia 10 veces inferior a la de trabajo del uP. Es decir, FE=F68000/10.

Durante estos 10 ciclos de reloj de uP que dura 1 ciclo de E, esta línea está durante 6 ciclos a nivel bajo y 4 a nivel alto. Esto se puede observar en el diagrama.



/VPA: Validation of Periferical Address - Validación de dirección de periférico. Línea de entrada, activa a nivel bajo.

Por medio de la activación de esta línea (/VPA=0), el periférico indica al uP que se trata de un dispositivo de la familia 6800 o compatible.

Otra de sus utilidades se presenta a la hora de un ciclo de reconocimiento de interrupciones. Durante este proceso, su activación (/VPA=0) denota el uso de interrupciones autovectorizadas, pasando a obtenerse el número de vector como la suma del nivel de interrupción atendida y 24.

/VMA: Validation of Memory Address - Validación de dirección de memoria. Línea de salida, activa a nivel bajo

Esta señal indica, al activarse (/VMA=0), indica al periférico síncrono que la dirección colocada en el bus de direcciones es válida y que además el microprocesador se encuentra sincronizado con la señal E.

Esta señal solo responde a la activación de la línea /VPA, puesto que por medio de ella se indica la existencia de dispositivo síncrono.

Funcionamiento:

Cuando el microprocesador accede a un periférico síncrono, estos son los pasos:

1. El uP coloca la dirección en el bus de direcciones y activa /AS(=0).
2. La lógica de decodificación genera una salida que llamaremos, por ejemplo, /SELSinc.
3. /SELSinc se encarga de activar /VPA(=0).
4. Cuando el uP se encuentre sincronizado con E, activa /VMA(=0).
5. Por medio de la combinación de /VMA y /SELSinc, se selecciona y activa el periférico. El uso conjunto de estas dos señales asegura el correcto direccionamiento del dispositivo: aunque /VMA es una salida que puede activar a muchos dispositivos, /SELSinc solo es válido para un solo de entre ellos, por lo que la activación de más de uno es imposible.

Por lo tanto, el 68000 siempre intentará un acceso a bus asíncrono, pero cuando recibe la señal **/VPA** finalizará este ciclo y comenzará un acceso a bus síncrono.

2.4.5. Control de posesión de los buses.

Todo sistema ha de considerar la posibilidad de que otros dispositivos además del microprocesador tengan la oportunidad de controlar los buses.

Gracias a ello son posibles, por ejemplo, las transferencias directas a memoria (DMA) por parte de los Custom Chips.

Para poder pues compartir y ceder los buses (direcciones y datos), el uP requiere de unas líneas de control para otorgar la dirección a otros dispositivos.

/BR: Bus Request - Petición de bus. Línea de entrada en colector o drenador abierto, activa a nivel bajo.

Esta línea la utilizan los dispositivos que pueden necesitar los buses para efectuar una petición de cesión de estos por parte del microprocesador.

Para ello, activan /BR(=0), que al ser de tipo colector (o drenador) abierto permite efectuar una AND cableada.

Se consigue así una función /BR = BRa*BRb*....BRz.

De esta forma, mientras que los BRx de los dispositivos están a 1, /BR no se activa. Basta con que cualquiera de ellos pase a 0 para que /BR=0 y que por lo tanto reconozca una petición de control de buses.

/BG: Bus Granted - Bus cedido. Línea de salida, activa a nivel bajo.

Con esta señal, el microprocesador indica que cede los buses cuando acabe el ciclo de bus en curso, lo que significa que el uP pone el bus cedido en estado de alta impedancia.

/BGACK: Bus Granted ACKnowledge - Reconocimiento de cesión de buses. Línea de entrada, activa a nivel bajo.

Mediante esta línea, un dispositivo indica al microprocesador que está controlando los buses, llamándose a este dispositivo MASTER.

Las condiciones que permiten a un dispositivo activar esta línea y tomar la rienda de los buses son:

- se ha activado con anterioridad la línea /BG, por lo que el uP cede el bus.
- /AS se encuentre inactiva. Esto indica que el uP ha puesto sus buses en alta impedancia y que ya no los requiere.
- /BGACK no se encuentra ya activa (=0); de lo contrario otro dispositivo ya estaría controlando los buses.

Cuando se cumplen todas las condiciones, el dispositivo activa /BGACK(=0).

Funcionamiento:

La secuencia de gestión de cesión (a grandes rasgos) es como sigue:

1. El dispositivo que desea controlar los buses efectúa una petición de cesión de bus a través de /BR(=0).
2. Si el uP puede cederlos al finalizar el ciclo de bus en curso, activará /BG (=0).
3. El dispositivo, al ver que su petición ha sido atendida, tratará de activar /BGACK según se cumplan las condiciones descritas anteriormente.
4. El nuevo maestro desactiva la línea /BR(=1) y efectúa las operaciones que necesita. En respuesta, el uP desactiva /BG(=1).
5. Cuando haya acabado, el dispositivo desactiva /BGACK(=1), con lo que el microprocesador pasa a ser de nuevo el maestro de los buses.

Todo esto se encuentra muy simplificado puesto que en la práctica existen un cierto número de prioridades según los dispositivos y su importancia en el sistema, por lo que existe una lógica de asignación que tiene en cuenta estos niveles de prioridad. De esta forma, cuando coincidan dos o más peticiones, es este circuito quien se encarga de conceder la dirección de los buses.

Para que además todo esto sea posible, se le atribuye al uP una de las prioridades más bajas, lo que posibilita que pueda ceder los buses antes dispositivos de mayor prioridad.

2.4.6. Control del sistema.

Mediante este conjunto de líneas se puede controlar el sistema microprocesador (tanto el micro en sí como todo el sistema que este gestiona).

/BERR: Bus ERRor - Error de bus. Línea de entrada, activa a nivel bajo.

Mediante esta señal, el sistema externo puede indicar al micro que existe un problema en los buses. La señal la ha de generar un dispositivo externo al no haber respuesta de un otro dispositivo que ha sido direccionado en el actual ciclo de bus o en cualquier situación anormal de funcionamiento de los buses.

Según el estado de la línea **/HALT**, se producirá:

- un proceso de excepción de nivel 2 (**/HALT** inactiva),
- un ciclo de reintento de bus (**/HALT** activa). Si después de este ciclo de reintento de bus se vuelve a producir el error se entra en un estado DOBLE ERROR, en el que se intenta detectar dicho error y del que solamente se saldrá cuando se ejecute un reset externo.

/RESET: RESET – Reinicializar. Línea bidireccional, activa a nivel bajo.

Con esta se procede a la reinicialización de los distintos dispositivos del sistema.

Cuando trabaja como línea de entrada, al activarse (=0) y siempre que la línea **/HALT** también lo esté (=0), se ejecuta la inicialización del microprocesador mediante la atención de la excepción de Reset y su consiguiente ejecución.

Cuando trabaja como línea de salida, se fuerza a los dispositivos externos a una inicialización forzada (para aquellos dispositivos que posean este tipo de entrada), pero no se modifica el estado interno del uP que continua con la ejecución normal del programa. La activación de esta señal se produce con la ejecución de la instrucción RESET.

/HALT: HALT – Parada. Línea bidireccional, activa a nivel bajo.

Como entrada, el sistema externo pide la parada del microprocesador, colocando este sus buses y líneas triestado en alta impedancia y las señales de control en estado inactivo, al terminar el ciclo de bus activo.

Como salida, el microprocesador indica al sistema que ha parado. Esta parada del uP se puede conseguir mediante la instrucción STOP o tras realizar una parada como entrada.

2.4.7. Control del estado del microprocesador.

Durante la ejecución del código de los programas, el microprocesador 68000 se ve siempre manejando datos, pasando de modo Usuario a Supervisor, ejecutando instrucciones o en proceso de excepción.

Para permitir que el resto del sistema conozca el tipo de operación que está realizando se ha dotado al 68000 de unas salidas, cuyos niveles indican el estado en curso.

Esto es muy útil cuando por ejemplo se desea tener un banco de memoria exclusivo para el modo Supervisor (para su pila y variables del sistema) o para tener bancos de memoria de datos e instrucciones independientes, entre otros muchos ejemplos.

FC2, FC1, FC0: Funcion Code - Código de función. Líneas de salida, activas a nivel alto.

Estas líneas indican si el uP está trabajando una instrucción o un dato, así como si lo está haciendo en modo Usuario o Supervisor; finalmente indica si se encuentra en un estado de reconocimiento de interrupciones.

Hay que tener en cuenta, que estas líneas solo son válidas mientras que /AS esté activada (=0).

Las posibles combinaciones son:

FC2	*FC1*	*FC0*	*Tipo de ciclo*
0	0	0	No definido
0	0	1	Datos de Usuario
0	1	0	Programa de Usuario
0	1	1	No definido
1	0	0	No definido
1	0	1	Datos de Supervisor
1	1	0	Programa de Supervisor
1	1	1	Reconocimiento de interrupciones

2.4.8. Control de las interrupciones.

Cuando un dispositivo genera una interrupción, esta pasa por cierto circuito de decodificación que ofrece a su salida el nivel que esa interrupción posee en el sistema.

En el 68000 existen 8 posibles niveles, que van desde el 0 (no petición) hasta el 7 (interrupción no enmascarable).

Para comunicar al microprocesador el nivel de interrupción que se ha generado y que este lo pueda comparar con la máscara de interrupciones del byte del sistema del SR (y ver si ha de entrar en un estado de excepción), se coloca el nivel en cuestión en ciertas líneas de entrada.

/IPL2, /IPL1, IPL0: Interrupt Pending Level 2, 1 y 0 - Nivel de interrupción pendiente. Líneas de entrada, activas a nivel bajo.

Comunican al uP cuál es el nivel de interrupción que se ha generado. El conjunto de las 3 líneas permite 8 niveles (2^3=8) de interrupción, entre los que destacan el nivel 0, que corresponde a una no-petición de interrupción, y el nivel 7, que es una interrupción no enmascarable.

Las posibles combinaciones de estas líneas y el nivel resultante es (no olvidando que ser activas a nivel bajo se emplea lógica negativa):

/IPL2	/IPL1	/IPL0	Nivel
1	1	1	0
1	1	0	1
1	0	1	2
1	0	0	3
0	1	1	4
0	1	0	5
0	0	1	6
0	0	0	7

Algo a tener en cuenta en el diseño del sistema, es que el nivel generado el /IPL2, /IPL1 e /IPL0 ha de mantenerse constante hasta que uP pase al estado de reconocimiento de interrupciones. De lo contrario la interrupción no será reconocida.

Para evitar que se atiendan otras interrupciones de inferior nivel (pero superior al de la máscara), se copia en la pila de supervisor la máscara original y se sobrescribe el SR con el nivel que se está atendiendo. Esto fuerza a que solo una interrupción de nivel superior pueda interrumpir la que se esté atendiendo en ese instante.

3. Ciclos del bus.

3.1. Introducción.

Se denomina **ciclo de bus** al tiempo total que dura el proceso de transferencia de un dato entre el uP y algún dispositivo conectado al bus. Se producen ciclos de bus cuando el uP realiza un acceso a memoria o a E/S.

Los ciclos de bus pueden ser:

- De LECTURA, cuando el uP recibe un dato procedente de algún dispositivo conectado al bus.
- De ESCRITURA, cuando el uP envía un dato que debe ser recogido por algún dispositivo conectado al bus.
- De LECTURA – MODIFICACIÓN – ESCRITURA, cuando el uP recibe un dato de memoria, lo modifica y lo vuelve a depositar en la misma posición de memoria. En realidad no es sino una secuencia de un ciclo de lectura y uno de escritura que es llevada a cabo por algunas instrucciones.

En el 68000, un ciclo de bus ocupa como mínimo 4 ciclos de reloj, comprendiendo 8 estados denominados $\mathbf{S_0,\ldots, S_7}$, equivalentes cada uno de ellos a un semiciclo de la señal de reloj.

Para que un ciclo de bus finalice es necesario que la CPU reciba la señal **/DTACK** en el momento adecuado. De no ser así, el ciclo de bus se alarga todo lo necesario, insertando entre los estados $\mathbf{S_4}$ y $\mathbf{S_5}$, unos periodos de tiempo denominados **ciclos de espera.** Cada uno de estos ciclos tiene una duración de un semiciclo de reloj y equivale a la repetición del estado $\mathbf{S_4}$ en espera de recibir la señal **/DTACK**. De esta forma, un ciclo de bus puede alargarse lo que sea necesario para dar tiempo a que la memoria o los dispositivos de E/S conectados al bus tengan tiempo suficiente para recibir o entregar el dato. Mediante este mecanismo pueden conectarse al bus dispositivos con tiempos de acceso relativamente largos.

3.2. Ciclo de lectura.

Un ciclo de lectura es un ciclo de bus en el que el uP realiza una lectura de la memoria, entendiendo como ésta bien la memoria propiamente dicha, o bien algún dispositivo de

entrada. Este tipo de ciclo se origina cuando el uP ejecuta alguna instrucción cuya **fuente** es una dirección de memoria.

Las acciones que se realizan en un ciclo de lectura son las siguientes:

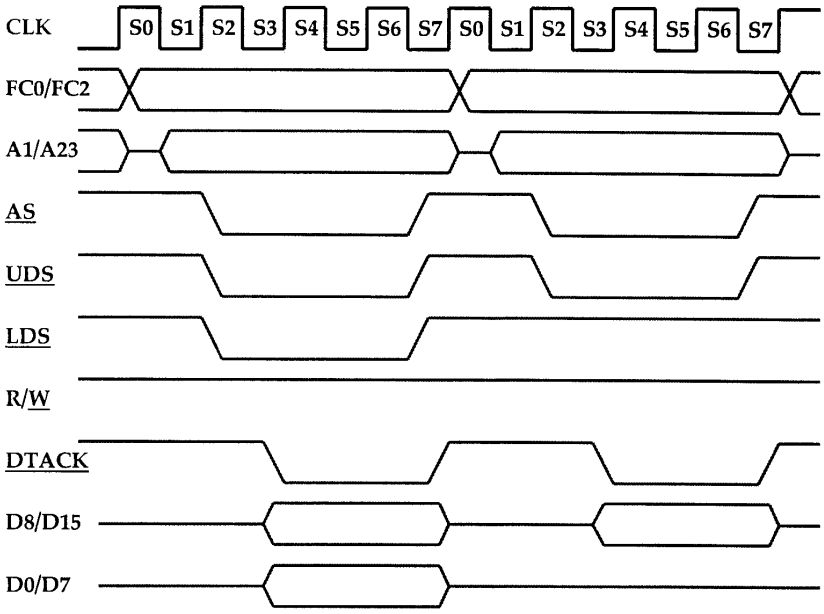
» El uP realiza las siguientes funciones:
  - Pone **R/\W** a 1 para indicar LECTURA.
  - Coloca el código de función en FC2 a FC0.
  - Pone en el bus de direcciones la dirección a acceder.
  - Válida la dirección activando **/AS**.
  - Activa la señales **/UDS**, **/LDS** según se vaya a direccionar una palabra o un byte (par o impar).

» En este momento el subsistema de memoria dispone de la información suficiente para realizar el acceso a la dirección indicada. Sus funciones serán:
  - Decodificación de la dirección.
  - Entregar el dato en la parte del bus de datos correspondiente dependiendo que se realice un acceso de tipo palabra o byte.
  - Activa la señal **/DTACK** para informar al uP que el dato está disponible.

» Una vez recibida la señal **/DTACK** el uP realizará:
  - La captura del dato que el subsistema a puesto en el bus del datos.
  - La desactivación de las señales que había activado al inicio del ciclo: **/UDS, /LDS, /AS**.

» Una vez que el uP ha realizado estas funciones el subsistema quitará el dato y desactiva la señal **/DTACK**.

En la siguiente figura se muestra el diagrama de tiempos simplificado de un ciclo de lectura, en el que se puede observar la evolución de estas señales.



Como puede observarse cada uno de los acontecimientos enumerados anteriormente tienen lugar en momento concreto del ciclo. Así a continuación indicamos las acciones que ocurren en cada estado del ciclo:

- Estado 0:
  - El uP pone el código de función en FC0/FC2.
  - El uP deja el valor por defecto de **R/\W** (=1).
- Estado 1:
  - El uP pone una dirección válida en A1-A23.
- Estado 2:
  - El uP activa **/AS**
  - El uP selecciona un acceso tipo palabra o byte con las líneas **/UDS** y **/LDS**.
- Estado 3:
  - No se altera ninguna señal.
- Estado 4:
  - Finaliza el ciclo si se activa la señal **/DTACK** o **/BERR** o inicia una lectura síncrona si se activa **/VPA**.
  - Si termina sin recibir **/DTACK** o **/BERR** se insertan ciclos de espera hasta que alguna se active.
- Estado 5:
  - No se altera ninguna señal.
- Estado 6:

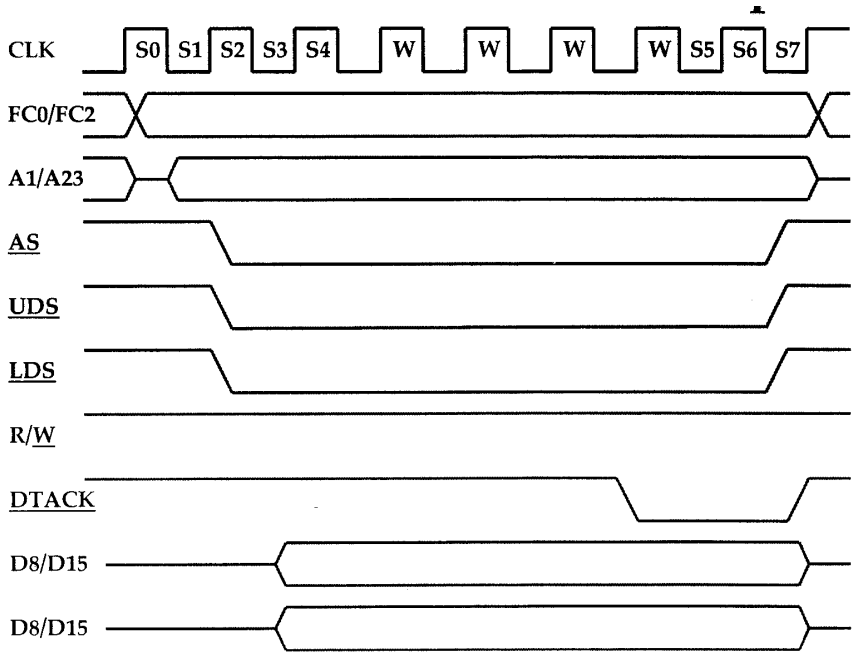
- El uP lee la palabra o byte colocada en D0/D15.

☞ Estado 7:

- El uP pone el valor por defecto en **/AS, /UDS, y /LDS** (=1).
- El dispositivo desactiva **/DTACK** o **/BERR**.

Como hemos indicado anteriormente si la señal **/DTACK** no llega antes de que finalizase el ciclo **S_4**, se alarga el ciclo de bus, insertándose ciclos de reloj adicionales entre **S_4** y **S_5**, denominados CICLOS DE ESPERA. En la figura siguiente se puede observar un diagrama de tiempos de un ciclo de lectura en el que se han insertado 4 ciclos de espera. Con la inserción de ciclos de espera el tiempo de acceso requerido se alarga y es posible acceder a pastillas de memoria o E/S más lentas y más baratas.

La inserción de los ciclos de espera finaliza cuando la CPU detecta la activación de la señal **/DTACK** y continua con la ejecución normal indicada en el caso anterior.



3.3. Ciclo de escritura.

El ciclo de escritura es similar al descrito anteriormente, pero con las siguientes diferencias:

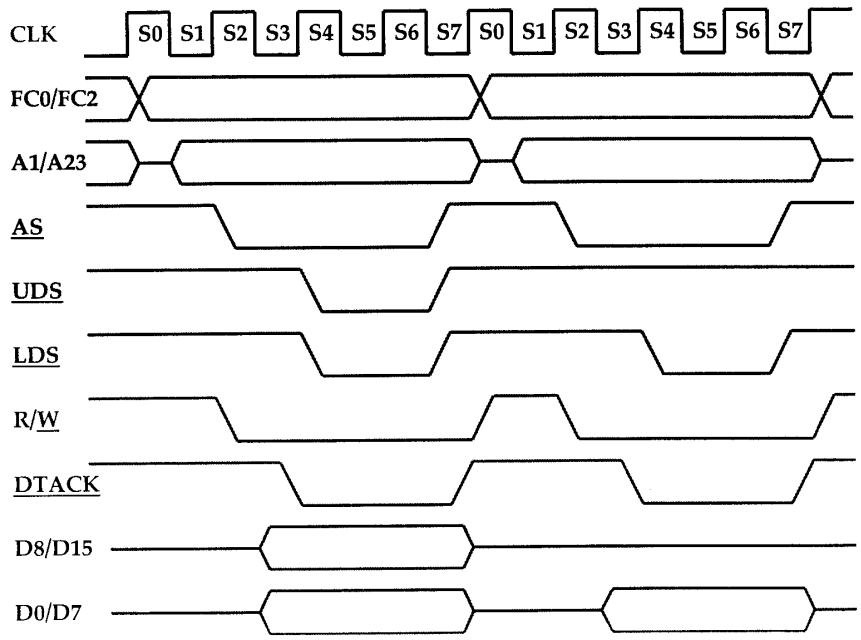
- Se produce un ciclo de escritura cada vez que el uP ejecuta una instrucción cuyo **destino** sea una dirección correspondiente a un dispositivo (memoria o E/S) conectado al bus.
- El uP pondrá en el bus de datos el dato que se quiere escribir en memoria o en E/S.
- El subsistema de memoria deberá en este caso leer el dato, en vez de escribirlo.

Así pues los principales pasos que se realizan en este ciclo de lectura son:

- El uP realizará los siguientes pasos para iniciar el ciclo:
  - Coloca el código de operación en FC0-FC2.
  - Pone el bus de direcciones la dirección válida.
  - La válida activando **/AS**.
  - Coloca el dato en la parte correspondiente del bus de datos dependiendo que la escritura sea palabra o byte (par o impar).
  - Activa adecuadamente **/LDS y /UDS** dependiendo del tipo de escritura acorde con la colocación del dato en el bus de datos.
- Una vez realizadas estas acciones el subsistema de memoria tiene la información necesaria para realizar la escritura y realiza las siguientes acciones:
  - Decodifica la dirección.
  - Almacena el dato en la dirección decodificada.
  - Activa la señal **/DTACK** para indicarle al uP que ha realizado la escritura adecuadamente.
- Después de esto el uP terminará la transferencia de datos realizando las siguientes acciones:
  - Desactiva las señales **/UDS y /LDS.**
  - Desactiva la señal **/AS.**
  - Quita el dato del bus.
  - Coloca la señal **R/\W** a su valor por defecto (=1).

- Por último el subsistema para terminar el ciclo:
  - Desactiva la señal **/DTACK**.

En la siguiente figura se representa el diagrama de tiempos de un ciclo de escritura en el que la señal **/DTACK** llega a tiempo y no es necesario insertar ciclos de espera.



Hasta ahora hemos comentado la sucesión de acontecimientos que se produce en un ciclo de escritura, pero cada uno de estos sucesos ocurre en un instante de tiempo exacto. Así a continuación indicamos qué ocurre en cada estado que constituyen este ciclo:

- Estado 0:
  - EL uP pone el código de función en FC0/FC2.
  - El uP deja el valor por defecto de **R/\W** (=1).
- Estado 1:
  - El uP pone la dirección valida en A1/A23.
- Estado 2:
  - El uP activa **/AS**.
  - El uP desactiva **R/\W.**
- Estado 3:
  - El uP pone la palabra o byte en D0/D15 (en la parte correspondiente de este bus dependiendo de si la transferencia es tipo palabra o byte par o impar).
- Estado 4:
  - El uP selecciona palabra o byte en **/UDS** y **/LDS**.
  - Finaliza el ciclo si se activan **/DTACK** o **/BERR** por parte del dispositivo o inicia una lectura síncrona si se activa **/VPA**.
  - Si termina sin recibir **/DTACK** o **/BERR** se insertan ciclos de espera hasta que alguna se active.
- Estado 5:
  - No se altera ninguna señal.

- Estado 6:
  - No se altera ninguna señal.
- Estado 7:
  - El uP pone el valor por defecto en **/AS, /UDS, /LDS, y R\/W** (=1).
  - El dispositivo desactiva **/DTACK** o **/BERR.**

3.4. Ciclo de Lectura-Modificación-Escritura.

El ciclo de lectura-modificación-escritura realiza la lectura de un dato de memoria, lo modifica, y luego lo vuelve a escribir en la misma posición de memoria. Este ciclo es un ciclo indivisible, es decir, el uP no concederá lo buses hasta que no haya terminado dicho ciclo, puesto que en caso contrario si el uP que se encontrara en un sistema multiprocesador puede ocurrir que mientras que el uP está realizado la modificación del dato cambie la dirección contenida en A1/A23 de forma que al intentar el primer uP volver a escribir el dato en su misma posición de memoria de donde lo recogió le va a ser imposible.

Este ciclo de bus únicamente se produce cuando se ejecuta la instrucción TAS (Test and Set) y únicamente permite realizar operaciones de transferencia de tipo byte. Esta instrucción TAS implementa la idea de un semáforo, que es una variable que indica si un determinado recurso está siendo utilizado o no en un determinado instante.

En el 68000 estos semáforos se almacenan en byte, puesto que la unidad de almacenamiento mínima que permite, siendo únicamente importante de este byte el bit más significativo. Un ejemplo de esta instrucción sería:

```
Espera      TAS semáforo
            BNE Espera
                        ; uso recurso
            CLR.B semáforo
                        ; sigue programa.
```

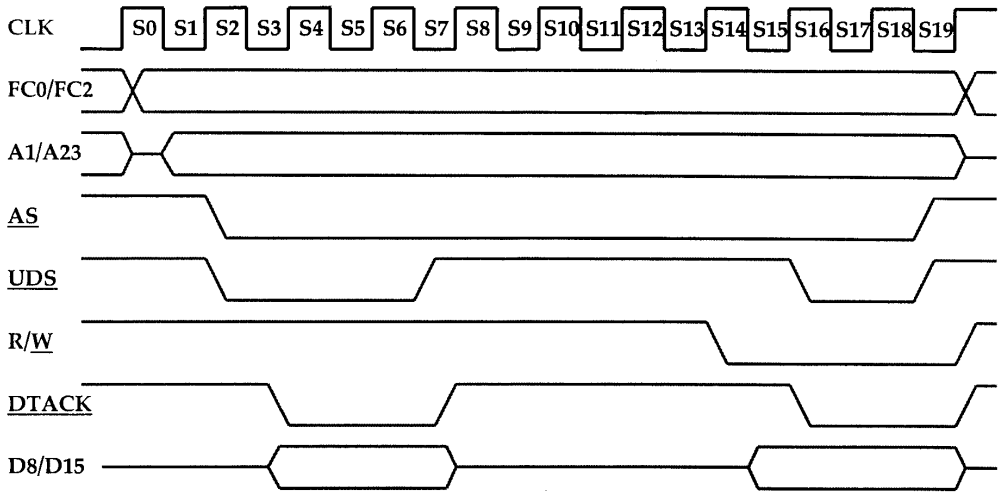
Las principales acciones que se realizan en este ciclo de bus son las siguientes:

- El uP realiza el direccionamiento del dispositivo:
  - **R\/W** =1.
  - Pone el código de función en **FC0/FC2.**
  - Coloca la dirección válida en A1/A23.

- Válida la dirección mediante la activación de **/AS.**
- Activa correspondientemente **/UDS y /LDS** para realizar una transferencia de byte par o impar.

- El dispositivo depositará el dato en el bus:
  - Decodifica la dirección.
  - Coloca el dato en la parte adecuada del bus de datos según la transferencia a realizar.
  - Activa **/DTACK.**
- El uP recibe el dato.
  - Lee el dato
  - Desactiva **/UDS y /LDS.**
  - **Inicia modificación de dato.**
- Terminar el ciclo de lectura
  - Dispositivo quita el dato.
  - Desactiva **/DTACK.**
- El uP inicializa la transferencia de salida:
  - Desactiva **R/\W** (=0).
  - Coloca el dato en la parte correspondiente del bus de datos (D15-D8 o D7-D0).
  - Activa /**UDS y /LDS** de acuerdo con la transferencia que se quiere realizar.
- El dispositivo obtiene el dato.
  - Almacena el dato.
  - Activa **/DTACK.**
- El uP termina la transferencia de salida.
  - Desactiva **/UDS y /LDS.**
  - Desactiva **/AS.**
  - Quita el dato.
  - Activa **R\/W** (=1).
- El dispositivo termina el ciclo:
  - Desactiva **/DTACK.**

A continuación indicamos el cronograma como varían las señales en el tiempo e indicamos que señales varían en cada estado.



- Estado 0:
  - El uP pone el ciclo de función en FC0/FC2.
  - Y deja el valor por defecto de **R\/W**.
- Estado 1:
  - El uP pone la dirección valida en A1/A23.

- Estado 2:
  - Válida dicha dirección activando **/AS.**
  - Y selecciona si la transferencia será a byte par o impar activando la señales **/UDS y /LDS** adecuadamente.
- Estado 3:
  - No se altera ninguna señal.
- Estado 4:
  - Finaliza el ciclo si el periférico activa **/DTACK o /BERR** o se inicia un lectura síncrona si el periférico activa **/VPA**.
  - Si termina sin recibir **/DTACK o /BERR** se insertan ciclos de espera hasta que alguna se active.
- Estado 5:
  - No se altera ninguna señal.
- Estado 6:
  - El uP lee el byte par o impar en D0/D15.
- Estado 7:
  - El uP pone a su valor por defecto **/UDS y /LDS** (=1).

- El dispositivo desactiva **/DTACK o /BERR.**

☞ Estados 8 a 11:

- No se altera ninguna señal mientras se modifica el byte.

☞ Estado 12:

- El uP pone el código de función en FC0/FC2.
- Y deja el valor por defecto de **R/\W.**

☞ Estado 13:

- No se altera ninguna señal.

☞ Estado 14:

- El uP desactiva **R/\W.**

☞ Estado 15:

- El uP pone el byte par o impar en D0/D15.

☞ Estado 16:

- El uP selecciona el byte par o impar en **/UDS y /LDS.**
- Finaliza el ciclo si el dispositivo activa **/DTACK o /BERR** o se inicia una lectura síncrona si se activa **/VPA**.
- Si termina el ciclo sin recibir **/DTACK** o **/BERR** se insertan ciclos de espera hasta que alguna se active.

☞ Estado 17:

- No se altera ninguna señal.

☞ Estado 18:

- No se altera ninguna señal.

☞ Estado 19:

- El uP pone el valor por defecto en **/AS, /UDS, /LDS, y R/\W.**
- El dispositivo desactiva **/DTACK o /BERR.**

CAPITULO 5

Excepciones

1. Introducción.
2. Las rutinas de atención a las interrupciones.
3. Las interrupciones hardware.
   3.1. EL enmascaramiento de las interrupciones.
4. El RESET.
5. El ERROR de BUS.
6. Las interrupciones software.
7. Las instrucciones de comprobación.
8. Errores internos.
9. Instrucciones de emulación.
10. Ejecución paso a paso.
11. La prioridad de las excepciones.

1. Introducción.

Las excepciones son acontecimientos, externos o internos, que dan lugar a que interrumpa la ejecución normal de un programa para dar paso a la ejecución de una rutina de atención especificada. Bajo este concepto se incluyen:

- Las interrupciones hardware.
- Las interrupciones software.
- Los errores internos.
- El reset.
- Errores hardware.
- Ejecución paso a paso.
- Etc.

La excepción es un mecanismo establecido para atender necesidades URGENTES del sistema. Cuando se produce una excepción el uP abandona su actividad normal para pasar a atender la emergencia. A estos efectos, las excepciones pueden clasificarse en:

- ☞ De **máxima urgencia.** Se atienden dentro de los dos ciclos de reloj siguientes. De este tipo son **RESET, ERROR de BUS, y ERROR de DIRECCIÓN.**

- ☞ De **media urgencia**. El uP acaba la ejecución de la instrucción que estaba tratando. Pasa a atender la excepción antes de comenzar con la siguiente instrucción. Pertenecen a este tipo la **interrupciones hardware**, la **ejecución paso a paso**, las de **código ilegal**, y la de **violación de privilegio.**

- ☞ Controladas por el programa. Son consecuencia de instrucciones insertadas por el programador. Se atienden al ejecutarse estas instrucciones.

Todas las excepciones del 68000 dan lugar a un proceso que se desarrollan en cuatro etapas:

- 1ª Etapa: se copia internamente el registro de estado. El bit S (modo supervisor) se activa. El modo **traza**, si esta activado, se desactiva.
- 2ª Etapa: se obtiene la dirección de comienzo de la rutina de atención especifica a la excepción que ha originado el proceso.

- 3ª Etapa: se salvaguardan en la pila del **supervisor** los contenidos del contador de programa y del registro de estado original, así como otros parámetros que permitan reanudar el programa. La pila queda de la siguiente forma:

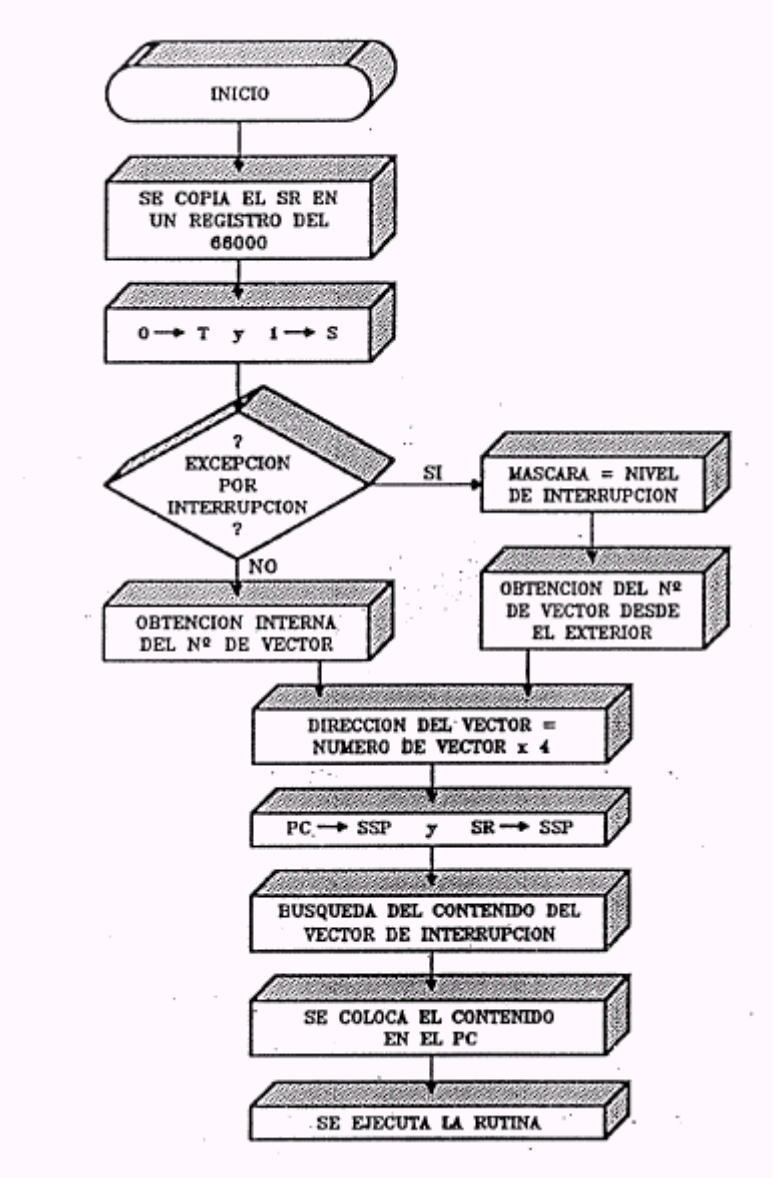
Posición de la pila	*Contenido*
SP+00	Registro de estado
SP+02	Palabra alta del PC
SP+04	Palabra baja del PC

No todas las excepciones se ajustan a este esquema. Las excepciones **ERROR de BUS y ERROR de DIRECCIÓN** llevan a la pila información adicional que sirve para facilitar una posterior investigación de las causas del error. Se llevan a la pila 7 palabras, de la cuales las 4 primeras deben ser retiradas de la pila manualmente en las rutinas de atención correspondientes, de modo que la instrucción **IRET** pueda comportarse correctamente. El contenido de la pila queda en estos casos así:

Posición de la pila	*Contenido*
SP+00	Bits 2,1,0: Códigos de función. Bit 3 si se trata de una instrucción, Bit 4 estado R/\W
SP+02	Dirección accedida (palabra alta)
SP+04	Dirección accedida (palabra baja)
SP+06	Código de operaciones (del registro de instrucción)
SP+08	Registro de Estado
SP+10	Palabra Alta del PC
SP+12	Palabra Baja del PC

- 4ª Etapa: El PC contiene un nuevo valor y la ejecución se transfiere a la rutina por él indicada. El nuevo valor del PC ha sido tomado automáticamente de la **tabla de vectores de excepciones** que se describe seguidamente.

Este proceso que se sigue cuando se produce un excepción se puede ver de una forma gráfica en la siguiente figura:



Todas las excepciones tiene asignado un **número de excepción** comprendido entre 0 y 255. Asociado a cada excepción hay 4 bytes de memoria en los que se contiene el **vector**, es decir, la dirección de comienzo de la rutina de atención correspondiente.

Todos los vectores están agrupados en una tabla, situada en la memoria central, ocupando precisamente los 1024 bytes más bajos del mapa de memoria, es decir, desde la dirección $000000 hasta la $0003FF. Cada elemento de esta tabla tiene la siguiente estructura:

PALABRA BAJA	NUEVO PC (BYTES ALTOS)
PALABRA ALTA	NUEVO PC (BYTES BAJOS)

El número de excepción, multiplicado por cuatro, proporciona directamente la dirección en donde se encuentra la dirección de atención correspondiente. Cuando el uP tiene que atender una excepción realiza internamente esta multiplicación para localizar las posiciones de memoria donde se debe encontrar el vector.

Por ejemplo, la excepción 6 nos lleva a las posiciones hexadecimales 018, 019, 01A, 01B.

En la siguiente tabla se muestran los vectores disponibles:

Número de excepción	***Dirección***	***Tipo***
0	$000000	RESET - Valor inicial del SSP
	$000004	- Valor inicial del PC
2	$000008	ERROR DE BUS
3	$00000C	ERROR DE DIRECCIÓN
4	$000010	INSTRUCCIÓN ILEGAL
5	$000014	DIVISIÓN POR CERO
6	$000018	INSTRUCCIÓN CHK
7	$00001C	INSTRUCCIÓN TRAPV
8	$000020	VIOLACIÓN DE PRIVILEGIO
9	$000024	MODO TRAZA
10	$000028	EMULACIÓN 1010
11	$00002C	EMULACIÓN 1111
12-23		**** RESERVADOS****
15	$00003C	VECTOR INICIALIZADO
24	$000060	INTERRUPCIÓN ESPURIA
25	$000064	NIVEL 1
26	$000068	NIVEL 2
27	$00006C	NIVEL 3
28	$000070	AUTOVECTORES NIVEL 4
29	$000074	NIVEL 5
30	$000078	NIVEL 6
31	$00007C	NIVEL 7
32-47		INTERRUPCIONES SOFTWARE (TRAPS)
48-63		****RESERVADOS****
64-255		INTERRUPCIONES HARDWARE

De los 255 vectores disponibles, los numerados de 12 a 23 y de 48 a 63 están reservados para versiones futuras. Los vectores 64 a 255 pueden ser manejados libremente por el usuario. Los restantes tienen asignadas un función especifica, como se muestra en la tabla.

Las causas de generación de una excepción pueden ser internas o externas y pueden clasificarse del siguiente modo.

EXCEPCIONES	
DE ORIGEN EXTERNO	Interrupciones hardware
	RESET
	Error de bus
DE ORIGEN INTERNO	Interrupciones software
	Instrucciones de comprobación
	Errores internos
	Ejecución paso a paso

2. Las rutinas de atención a las interrupciones.

El vector de interrupción constituye la dirección de comienzo de la **rutina de atención** correspondiente, es decir, un programa construido expresamente para atender la interrupción concreta que se ha producido. El programador debe tener en cuenta los siguientes factores:

- El uP entra en modo supervisor.
- Los registros generales del uP no han sido alterados. En el caso de las interrupciones *software,* estos registros pueden utilizarse para el paso de parámetros a la rutina de atención.
- En el caso de interrupciones *hardware*, o provocadas por errores, que pueden ocurrir en cualquier momento de la ejecución del programa interrumpido, es necesario preservar el contenido de estos registros. La rutina de atención deberá salvaguardar (en la pila preferentemente) el contenido de aquellos registros que van a ser alterados por la propia rutina de atención.
- En el caso de interrupciones *hardware,* dependiendo de cómo esté construido el periférico que solicita atención, puede ser necesario que la rutina de atención le envíe un **comando** para indicarle que ya está siendo atendido, con el fin de que retire su solicitud de interrupción.
- Al atenderse una interrupción el nivel de la **máscara** se sitúa en el mismo valor que la interrupción que está siendo atendida. De este modo quedan prohibidas las interrupciones de igual o menor nivel de prioridad. Es labor de la rutina de atención el posicionamiento de la máscara en el nivel deseado por el programador.
- Antes de finalizar la rutina se deben recuperar de la pila aquellos registros que fueron salvaguardados al principio de la rutina.
- Toda la rutina de atención debe finalizar con la instrucción **RTE**. Esta instrucción da lugar a la recuperación del PC y del registro de estado que fueron llevados a la pila automáticamente al producirse la excepción. Obsérvese que al recuperar el registro de estado, el uP vuelve al modo anterior y que el nivel de la máscara de interrupción recupera su valor original.

3. Las interrupciones hardware.

Podemos diferenciar entre las interrupciones hardware de propósito general y las interrupciones RESET y ERROR de BUS, que también son interrupciones hardware pero son más especificas.

Como ya hemos visto el 6800 dispone de las entradas **/IPL0, /IPL1, /IPL2**, que permiten la producción de interrupciones *enmascarables por nivel y* sometidas a un sistema de prioridades. No se tratan de entradas independientes sino que actúan conjuntamente, admitiendo un código de tres bits. Por lo tanto, lo normal es conectar a ellas las salidas de un **codificador con prioridad**, que permite compone este código, expandiendo hasta 7 las líneas de petición de interrupción.

Será el propio codificador el que se encargue de determinar la prioridad cuando se activen 2 ó más señales simultáneamente. Así como ya comentamos en el apartado de descripción de las líneas del 68000, las de nivel 7 serán las de mayor prioridad (no enmascarables) mientras que las de nivel 1 tendrán la mínima prioridad. El nivel 0 no existe puesto es cuando estas tres señales se encuentran desactivas y no se realiza petición de interrupción.

En circuitos sencillos en los que se dispone de menos de 7 periféricos generalmente se conecta cada uno de ellos a una entrada del codificador mientras que en el caso de sistemas más complejos donde el número de periféricos es mayor de 7 a cada entrada del codificador se suelen conectar más de un periférico. Puesto que cada una de estas entradas se corresponde a un nivel de prioridad, todos los periféricos conectados a la misma entrada tendrán la misma prioridad y se van a distinguir entre sí por el número de interrupción que entregarán al uP.

La secuencia de acontecimientos que tendrían lugar cuando un periférico solicita la interrupción sería la siguiente.

El dispositivo adecuado solicita la interrupción activado la entrada del codificador a la que se encuentra conectado obteniendo a su salida el nivel de petición que llegará al uP mediante los pines **/IPL0, /IPL1, /IPL2.**

El uP comparará este valor con el que contiene en ese momento la máscara de interrupciones, de forma que si es mayor que ésta la interrupción se atenderá. Para ello el uP pone en los terminales de direcciones A1, A2, y A3 el número del nivel atendido y al mismo tiempo sus pines **FC0, FC1 y FC2** se ponen a1 para indicar que se trata de un ciclo de reconocimiento de interrupción.

El periférico que ha solicitado la petición colocará en la parte baja del bus de datos el número de interrupción que le corresponde. Y el uP obtendrá este dato mediante un ciclo parecido al de lectura en memoria, interviniendo también las señales **/AS, /LDS, y /DTACK.** La señal **/LDS** se activa debido a que el vector se transmite por los 8 bits menos significativos del bus de datos. La señal **/DTACK** debe ser activa por el periférico en cuestión para ratificar la entrega del vector.

Una vez que el uP ha recibido el número de interrupción , coloca en su PC la dirección correspondiente y lee en la tabla de interrupciones la dirección en donde ha de comenzar la rutina de atención al periférico solicitante.

Las pastillas periféricas del 68000 están preparadas para un funcionamiento como el que hemos descrito, puesto que poseen de un registro en el que el uP durante el proceso de inicialización escribe el vector asignado que pude ser cualquiera entre 0 y 255. Aunque es preferible utilizar los vectores disponibles para el usuario (64 a255) ya que los otros tiene funciones fijas que pueden activadas por otras causas.

En el caso de que en los periféricos no hayan sido inicializados con su número de vector se entregará un vector número 15, previsto por defecto que puede utilizarse para corregir errores.

En el caso en que el uP no reciba la señal de validación **/DTACK o /VMP** dentro de los ciclos de reloj correspondientes, un circuito supervisor activará la señal **/BERR** y se producirá una excepción denominada **INTERRUPCIÓN ESPURIA** que tiene asignado el vector 24 para su tratamiento.

En el caso de que solo se conecte un periférico por cada nivel , o que no se desea hacer uso pleno de las interrupciones vectorizadas, puede construirse el sistema de modo que los periféricos activen la entrada de la CPU **/VPA** durante el ciclo de búsqueda del vector. Si esto ocurre, el uP renuncia a leer un vector exterior y utiliza el denominado AUTOVECTOR correspondiente a nivel de interrupción activado. Están previstos 7 autovectores uno para cada nivel de interrupción, cuyos números corresponden a los vectores 25 a 31.

El periférico debe mantener la petición de interrupción activa hasta que reciba la señal de reconocimiento, puesto que en caso contrario pudiera ser que la interrupción no fuese reconocida.

La señal de reconocimiento **/IACK** se puede conseguir mediante un codificador cuyas entradas son A1, A2, y A3 y **/AS**, la combinación **FC0.FC1.FC2=1,** y opcionalmente que todas las señales de dirección restantes se encuentren a nivel alto.

3.1. El enmascaramiento de interrupciones.

En el SR del uP existen los bits **I0, I1, I2** que determinan el estado de la **máscara de interrupción**. Si esta máscara esta a **000**, todas las interrupciones están permitidas. Si la máscara esta a **111** están todas prohibidas excepto las interrupciones de nivel 7 que se denominan **no enmascarables.**

Para que una interrupción sea atendida su nivel ha de ser superior al que tenga la mascara en ese momento. La única excepción a esta regla es la petición de nivel 7, que es atendida aunque la máscara se encuentre a igual nivel.

Como puede observarse por tanto, las máscaras de nivel 6 y 7 producen siempre el mismo efecto, es decir, solo permiten las interrupciones de nivel 7.

Un aspecto importante a destacar es que una petición de nivel 7 puede ser interrumpida por otra petición de este mismo nivel.

Cuando se atienden interrupciones *hardware*, la máscara adopta automáticamente el nivel de esta interrupción , de modo que no pueda atenderse a una interrupción de igual o menor nivel, pero sí de mayor nivel que ésta.

No obstante durante la ejecución de cualquier programo o rutina de atención a interrupción, la máscara pude en todo momento ponerse al nivel conveniente, permitiendo una interrupciones y no otras. Esto depende del efecto que quiera conseguir el programador en cada caso.

4. El RESET.

Al igual que ocurre en todos los uP el 68000 dispone de un terminal de RESET destinado a inicializar el sistema al conectarse el equipo o para salvar alguna circunstancia problemática en la que el procesador se encuentra *perdido* y el operador no dispone de otros recursos para reconducir el funcionamiento.

En el caso del 68000, como ya comentamos en su momento, el RESET es bidireccional, comportándose como una salida cuando el procesador ejecuta la instrucción RESET. De esta forma conseguimos una inicialización de todos los elementos conectados al sistema, pero no afecta esta inicialización al uP.

Por otro lado el RESET EXTERNO, es provocado por un circuito exterior al uP que fuerza su terminal a 0.

Este reset en el 68000 se trata como una excepción especial que utiliza los números de vector 0 y 1. A diferencia del resto de excepciones ésta reserva 8 bytes para almacenar su vector, por lo que ocupa el espacio, en la tabla de excepciones, correspondiente a las excepciones 0 y 1. La dirección de comienzo de la ejecución está contenida en los bytes $000004, $000005, $000006, y $000007, es decir, las posiciones que debería contener el vector 1. Las posiciones desde $000000 hasta $000003 sirven para contener el valor inicial que el programador ha deseado dar a puntero de la pila del modo supervisor de forma que desde el momento en que arranca el P se encuentre definida la posición de pila.

El reset se puede producir en algunas de estas dos circunstancias:

- Al aplicar la alimentación al sistema cuando éste se encontraba apagado.
- Cuando el operador del sistema decide reinicializar el sistema, generalmente por alguna situación catastrófica en la que se haya perdido el control de su funcionamiento.

En ambos casos no importa el estado anterior del uP y la excepción de reset destruye el contenido de los registros del uP y lo pone de nuevo en funcionamiento. Por este motivo no es necesario que la subrutina salvaguarde el contenido de ningún registro en la pila.

Para que el sistema pueda arrancar es necesario que tanto el vector como la rutina de inicialización se encuentren ya en memoria al encender el equipo, por lo que deben estar grabadas en algún tipo de memoria no volátil. La forma de arrancar el 68000 está concebida

para que en la zona más baja del mapa de memora se instale una pastilla ROM de tamaño suficiente para contener toda la tabla de vectores de interrupción y el programa básico.

En la siguiente tabla podemos observar como el arranque tiene lugar en 6 ciclos.

Ciclo de bus	*Acción*	
1º	INACTIVO	
2º	LEE $000000	
3º	LEE $000002	Lectura del valor inicial de SSP
4º	LEE $000004	
5º	LEE $000006	Lectura de valor inicial del PC
6º	LEE (PC)	Lectura del primer código de operación de la rutina RESET.

Al ejecutar esta excepción la máscara de interrupción se pone a nivel 7 por lo que quedan prohibidas las interrupciones excepto las no enmascarables. El uP se pone en modo supervisor y el funcionamiento paso a paso queda desactivado.

Para desencadenarse el proceso de RESET EXTERNO deben activarse simultáneamente los terminales **/RESET y /HALT.**

5. El error de bus.

El uP 68000 dispone de una entrada denominada **/BERR** dispuesta para recibir una señal generada externamente por un circuito diseñado por el usuario que podríamos denominar CIRCUITO SUPERVISOR DE BUS.

Una de las aplicaciones más importantes de esta señal es la detección de accesos a posiciones de memoria no implementadas. En este caso el circuito supervisor de bus debe diseñarse para que active **/BERR** si la señal de **/DTACK**, que confirma que el ciclo de bus ha terminado, se retrasa respecto de **/AS**, más allá de un límite prefijado. Para ello, el circuito supervisor de bus tendrá como entradas estas dos señales.

La señal **/BERR** puede utilizarse también para detectar otros problemas como podrían ser intentos de escritura de ROM, accesos a zonas de memoria protegidas o para indicar error en un periférico.

Cuando el uP detecta la activación de **/BERR** puede actuar de dos modos:

- si **/HALT** está activa, da lugar a una excepción de vector número 2.
- Si el **/HALT** está activa, el procesador se para hasta que se desactiva. Entonces repite el ciclo que había originado el error.

Si en el primer caso, al procesarse la excepción se produce un segundo error de bus, el procesador pasa al estado de *doble error*, que da lugar a su detección. En ese caso, la única salida a la situación es la realización de un RESET externo.

Este modo de funcionamiento tiene el objetivo de que el uP no pase a funcionar en un modo descarriado lo que daría lugar a alteraciones de la memoria que imposibilitarían una investigación que permitiese deducir las causas del fallo.

6. Las interrupciones software.

La instrucción **/TRAP n** en donde n es un número de 0 a 15, da lugar a la producción de una excepción cuyo número de vector está comprendido entre 32 y 47. Las interrupciones *software* se producen al ejecutarse esta instrucción, es decir, en los puntos del programa en los que el programador ha previsto. Realmente son equivalentes a llamar a subrutinas. Sus uso más interesante es el permitir que programas de usuario (generalmente ejecutados en modo usuario) puedan acceder a servicios propios del sistema operativo que deben ejecutarse en modo supervisor. Las interrupciones *software*, lo mismo que todas las excepciones ponen al uP en modo supervisor, y al final de su rutina de atención, al recuperarse el SR de la pila, permiten la vuelta al modo original.

Un ejemplo típico del uso de una interrupción software podría ser el siguiente:

La rutina de atención **TRAP 0** realiza la exploración de un teclado. Al finalizar, devuelve en el registro D0 el código de la tecla pulsada. Si no se ha pulsado ninguna tecla devuelve el valor FFFF, en dicho registro. Cuando un programa desee leer del teclado utilizará simplemente la instrucción **TRAP 0.**

7. Las instrucciones de comprobación.

El 68000 dispone de varias instrucciones que pueden dar lugar a una excepción si se cumplen determinadas condiciones. Estas son:

- **TRAPV.** Da lugar a una excepción (vector numero 7) si el bit V del SR se encuentra a 1. Su uso por el programador es evidente.
- **CHK.** Esta instrucción efectúa una comparación entre el contenido de un registro de datos y el de un operando. Si este registro contiene un número menor que cero o mayor que el operando, se produce una excepción denominada CHK cuyo vector es de número 6. Como puede verse, la finalidad de esta instrucción y de su excepción asociada es la comprobación de que un dato no excede determinados limites.
- **DIVU y DIVS.** Estas instrucciones pueden dar lugar a una excepción de vector número 5 si el divisor que se pretende utilizar es cero.

8. Errores internos.

Existen varias excepciones especificadas que se activan automáticamente al producirse determinadas situaciones de error. El 68000 detecta los siguientes errores:

- **ERROR de DIRECCIÓN (vector número 3).** El 6800 exige que los datos de 16 ó 32 bits se encuentren en direcciones pares. Si por el contrario, el programa obliga a buscar uno de estos datos en una dirección impar, se produce el error correspondiente.
- **VIOLACIÓN DE PRIVILEGIO (vector número 8).** Algunas instrucciones solamente pueden ser ejecutadas cuando el uP se encuentra en el modo *supervisor*. Si encontrándose en modo usuario, se intentara ejecutar alguna de estas instrucciones (STOP, RESET,...) se produce un error de este tipo.
- **CÓDIGO ILEGAL (vector número 4).** De todas las combinaciones de 16 bits que se pueden formar hay algunas que no corresponden a códigos de operación. Si en un ciclo de lectura de código de operación, se obtiene un valor de este tipo, correspondiente a una instrucción inexistente, se produce una excepción de INSTRUCCIÓN ILEGAL.

Es importante que el programador incluya siempre en el sistema rutinas para el tratamiento de estas excepciones. Estas rutinas deberían dar lugar a un aviso para que el operador conozca, que se ha producido algún tipo de error. Durante la fase de programación y depuración, en la que se suele disponer de un sistema de visualización conveniente, este aviso puede ser la aparición de un mensaje en pantalla. Pero muchas veces el sistema que se está diseñando no va a tener ni *display* ni siquiera un operador capaz de reaccionar ante un error de este tipo. Es evidente que en estos casos el programa debe estar ya lo suficientemente depurado como para que se pueda excluir totalmente la posibilidad de que produzca algún error de este tipo.

9. Instrucciones de emulación.

Entre las instrucciones no implementadas, reciben un tratamiento diferente aquellas cuyo código de operación comienza por 1010, o por 1111, que dan lugar a excepciones separadas, de vectores 10 y 11 respectivamente. La finalidad de estas excepciones, conocidas como de **EMULACIÓN** es la de atender a instrucciones inexistentes que se pueden emular mediante rutinas *ad hoc.* Utilizando estas excepciones el programador puede inventar instrucciones nuevas, que no serían tales sino que desencadenarían una excepción de este tipo que daría paso a una rutina que proporcionase el resultado deseado.

10. Ejecución paso a paso.

El 68000 dispone de la posibilidad de ejecutar sus programas paso a paso. En este método, al finalizarse cada instrucción, se produce una excepción que da paso a la ejecución de una rutina especifica o que cede el control a un programa **monitor**. Este programa permitirá el acceso a los diferentes registros o posiciones de memoria para comprobar el efecto real de las instrucciones anteriormente ejecutadas.

La ejecución paso a paso tiene interés durante la fase de depuración de un programa, ya que permite observar a *cámara lenta* el funcionamiento del mismo. Para ello, la rutina de atención a esta excepción copia en memoria el contenido de todos los registros del uP, antes de modificarlos, de forma que conserve el estado completo del uP después de finalizada la ejecución de la instrucción que ha provocado la excepción. Después, el programa monitor, de funcionamiento interactivo con el usuario a través de teclado y pantalla, permitirá que éste pueda emitir algunos comandos, siendo los más típicos:

- Visualizar los registros del uP (la copia que se guardó en memoria).
- Visualizar el contenido de cualquier zona de la memoria.
- Alterar el contenido de algún registro de uP.
- Ejecutar la siguiente instrucción.
- Etc.

Todos los sistemas de desarrollo suelen incluir programas depuradores (*debuggers)* que hacen uso de la ejecución paso a paso.

El modo paso a paso también es llamado TRAZA. El uP lo reconoce porque el bit 15 del SR está a 1.

11. La prioridad de las excepciones.

Varias excepciones pueden producirse simultáneamente. En este caso el uP debe tomar la decisión de cuál de ellas es atendida primero. En el caso de las interrupciones externas, es el propio es el propio codificador situado en las entradas /IPLx el que determina la prioridad. Pero cuando se trata de excepciones de diferente tipo, el criterio de prioridad que sigue el uP es el siguiente:

Orden	*Excepción*	*Grupo*
1º	RESET	
2º	Dirección	0
3º	Error Bus	
4º	Modo traza	
5º	Interrupción exterior	
6º	Instrucción ilegal	1
7º	Violación privilegio	
8º	Por instrucción	2

CAPITULO 6

Programación de la VIA 6522

1. Descripción de la VIA 6522.
2. Modelos de entrada salida paralelos.
3. Interfase de la VIA con el microprocesador.
4. Generación y control de interrupciones en la VIA 6522.

1. Descripción de la VIA 6522.

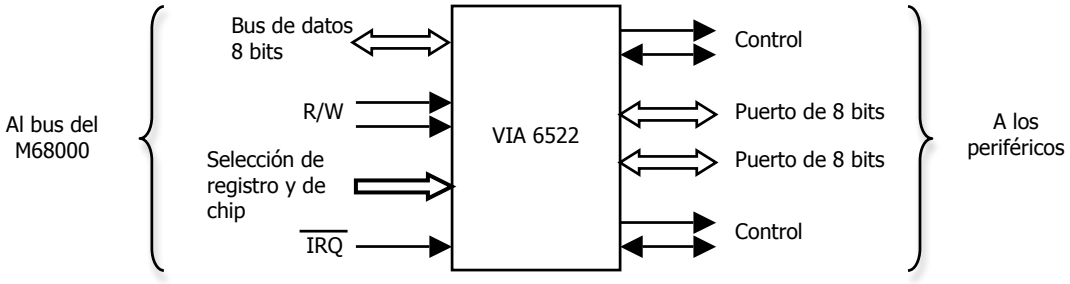
El subsistema de entrada / salida de un microcomputador suele ser más complicado y caro que el resto de los subsistemas - memoria, CPU, ... - ; además cada aplicación tiene exigencias de E/S propias que hace que se necesiten circuitos particulares para la misma. Para hacer compatible la especificidad de las aplicaciones con la generalidad que exige, por razones económicas, la construcción de un C.I.. los fabricantes de hardware ofrecen para cada familia de microprocesadores, un conjunto de chips de E/S cada uno de los cuales cubre un área más o menos extensa de funciones.

El área de aplicabilidad de un chip de E/S viene determinada por el número y tipo de circuitos en él integrados (contadores, registros de desplazamiento, líneas de E/S, etc. ...) y la particularización de una aplicación concreta se realiza por programación. Cada chip dispone de un conjunto de registros, accesibles desde programa como posiciones de memoria - en arquitecturas basadas en el *procesador MC68000 - cuyo contenido determina la interconexión interna de los circuitos, y por consiguiente, las funciones a realizar.

La VIA 6522 (Versatile Interface Adapter) es un chip de E/S fabricado por Rockwell con un amplio rango de prestaciones, es compatible con la familia 6800 y por tanto en arquitectura basada en micros posteriores tales como el 68000 funciona de manera síncrona. Descripción de la VIA 6522 En las siguientes figuras podemos ver tanto el patillaje como el diagrama de bloques de este chip.

1.1. Descripción de la VIA 6522.

En la siguiente figura podemos ver el diagrama de bloques de este chip.



El C.I. 6522 es un subsistema de E/S integrado que reúne en un sólo chip la mayoría de las funciones de E/S necesarias para un amplio rango de aplicaciones. Está diseñado para el control de periféricos en los sistemas basados en µprocesador y consta de los siguientes elementos:

1. **Dos ports bidireccionales,** denominados **PORT A y B**, de 8 bits cada uno. Cada una de estas líneas puede programarse como entrada o salida mediante el registro de direcciones de datos (DDRA o DDRB). Cada port incorpora además dos líneas de control denominadas CA1, CA2, CB1 y CB2. El registro de configuración del puerto (PCR), determina el flanco activo de las líneas CA1 y CB1, y el modo de funcionamiento de las línea CA2 y CB2. Estas líneas pueden utilizarse como entradas de interrupción.
2. **Un registro de desplazamiento (SHR)** de 8 bits, para la conversión de información de serie a paralelo y viceversa.
3. **Dos temporizadores (timers) / contadores** de 16 bits cada uno, que pueden utilizarse para contar o generar impulsos. También pueden producir ondas cuadradas de frecuencia variable, o usarse como temporizadores programables.
4. **Un registro de interrupción** que permite la producción de una determinada interrupción según el estado de los indicadores del registro (IFR). Este registro lleva asociado el correspondiente registro de validación de interrupciones (IER).

Además hay otro registro de control auxiliar (ACR), que determina el modo de funcionamiento de los timers y del SHR.

El control de dispositivos se realiza primeramente a través de los dos puertos bidireccionales de 8 bits. Siendo posible hacer que cada una de las líneas actúe como entrada o salida. Las líneas de E/S del periférico pueden controlarse directamente por medio de los timers con intervalos adecuados para generar ondas cuadradas de frecuencia programable y también para detectar y contar pulsos generados externamente.

Por otro lado el papel de los registros de control es facilitar el control de la mayoría de las funciones características de este chip, organizando los registros internos en un registro flag de interrupciones, un registro de habilitación de interrupciones y un par de registro de control de función.

Tanto los registros de control como los pertenecientes a los elementos funcionales son accesibles por programa, ocupando posiciones de memoria determinadas por la descodificación de las línea de dirección. Esta asignación se realiza en la fase de diseño de hardware.

La VIA aunque es un chip fabricado por Rockwell y no por Motorola se puede conectar directamente a los sistemas µprocesadores basados en CPUs de la familia 68xx de 8 bits. Pero además como el 68000 dispone de ciclo de bus síncrono para la conexión de periféricos de dicha familia, la VIA se puede conectar fácilmente al 68000 haciendo intervenir en el circuito de decodificación las líneas del bus síncrono.

2.- Modos de entrada / salida paralelo.

Las posibilidades de E/S de un sistema µprocesador determinan la utilidad del mismo ya que es la manera que tiene de comunicarse con el exterior. Por tanto entender el funcionamiento del sistema de E/S y sus posibilidades de programación es uno de los puntos clave en la programación de sistemas digitales. De hecho en grandes sistemas computadores utilizados actualmente para desarrollar simuladores y controladores de procesos a tiempo real o incluso realidad virtual encuentran el cuello de botella a sus posibilidades de funcionamiento en el sistema de E/S que controla la interconexión de los dispositivos periféricos encargados de la recogida de datos y el procesador o procesadores centrales.

En cuanto a la E/S paralelo tenemos dos formas básicas de realizarla.

2.1. E/S paralelo sin espera de respuesta.

El computador toma todas las decisiones de E/S, sin esperar información alguna sobre el estado de los dispositivos periféricos.

El programa no se preocupa, es decir no comprueba el conjunto de señales que le informan del estado de los periféricos. Este tipo de comunicación es útil en aplicaciones en las que el µprocesador no debe reaccionar a cambios producidos en los periféricos y estos se encuentran siempre disponibles a las peticiones del microprocesador.

La E/S sin espera de respuesta implica menor control sobre los dispositivos periféricos, siendo necesario un menor número de líneas de control al no existir ninguna señal que indique cuándo el µprocesador dispone de un dato válido procedente del dispositivo periférico para realizar la operación de entrada, o cuando el mencionado periférico está dispuesto para recibir datos procedentes del microcomputador. Las operaciones de E/S se realizan unilateralmente por parte del microprocesador, lo que necesita un conocimiento previo por parte del programador de la presencia de datos de entrada o de la disponibilidad del dispositivo para aceptar datos de salida.

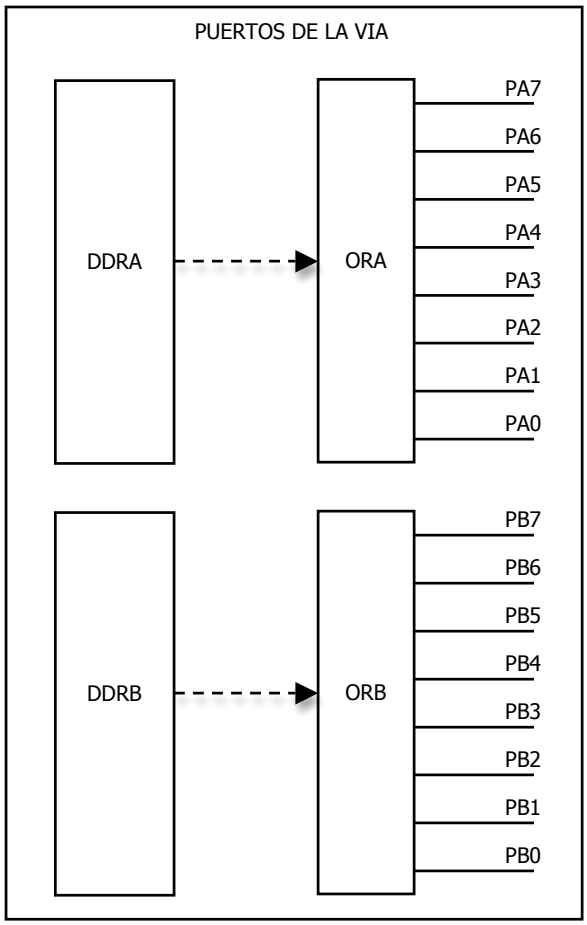
En el caso de la VIA 6522 , para este tipo de E/S sólo es necesario utilizar los ports de E/S (PORT A o PORT B), sin necesidad de usar las líneas de protocolo CA1, CA2, CB1 y CB2.

2.1.1. Programación de los PORTS de la VIA.

La única diferencia que existe entre los ports A y B es de tipo eléctrico . El Fan-Out del port B es superior al del port A y mientras que al port B pueden conectarse transistores Darlington utilizados en la activación de solenoides y relés, el port A solo admite una carga TTL, por lo que siempre que sea posible conviene utilizar el port A como entrada y el port B como salida.

En cuanto a programación ambos ports son idénticos y haremos referencia a los dos conjuntamente.

Cada port de la VIA se compone de 8 líneas (PA0-PA7 y PB0-PB7) que pueden ser individualmente programadas para actuar como entrada o salida bajo el control de un registro de dirección de los datos (DDRA o DDRB) y un registro (ORA o ORB) donde se reciben los datos o donde se colocan los datos para ser enviados por las líneas al exterior, tal como se muestra en la siguiente figura:



ORA y ORB : Son los **registros de datos** y funcionan como emisor o receptor de datos hacia o desde las líneas **PA0-PA7** y **PB0-PB7** respectivamente para cada uno de los ports A y B. Cada bit individual de los registros va asociado a una línea física exterior. El modo de funcionamiento de cada línea independientemente está controlado por el contenido de los registros **DDRA** y **DDRB**.

DDRA y DDRB : son los **registros de dirección de datos**. Cada bit en estos registros indica cómo se comporta el bit correspondiente de los registros **ORA** y **ORB**, bien como entrada o como salida.

2.1.2. Funcionamiento en el modo de E/S sin espera de respuesta.

Los registros descritos anteriormente son los únicos usados para acceder a cada uno de los 8 bits del puerto del periférico. Cada puerto tiene su registro de dirección de datos (**DDRA** ó **DDRB**) para especificar cuando los pines del periférico van a actuar como entradas o como salidas.

- Un "0" en el **bit del registro de dirección de dato** causa que el correspondiente pin del periférico actúe como una **entrada.**
- Un "1" causa que el pin actúe como **salida**.

Cuando el pin es programado para actuar como una salida, el voltaje del pin es controlado por el correspondiente bit del registro de salida de dato (ORA ó ORB).

- Un "1" en el registro de salida de datos causa que el pin se ponga a nivel lógico alto , 5 voltios.
- Un "0" pone el pin a nivel lógico bajo, 0 voltios.

Los valores escritos en los bits del registro de datos correspondientes a pines programados para que actúen como entrada no afectaran el valor de esas líneas(pines).

Cuando el pin se programa como entrada el estado de la línea ("1" = 5 volt, "0" = 0 volt.) determinará el valor correspondiente del bit de los registros de datos ORA ó ORB.

Cuando se leen los registros ORA y ORB, los bits correspondientes a líneas programadas como salidas no tienen ninguna significación válida.

La secuencia de operaciones a realizar en una operación de E/S sin espera de respuesta será:

1. Establecer el sentido de la operación, entrada o salida, para cada una de las líneas del port, cargando la correspondiente configuración de bits en el registro DDRA y DDRB.
2. Realizar la transferencia de datos hacia o desde el exterior escribiendo o leyendo los registros ORA y ORB.

2.1.3. Direcciones y programación de los registros.

El MC68000 no dispone de instrucciones especificas para operaciones de E/S -al contrario de lo que sucede con microprocesadores de otras familias, p.e. la 8086 de INTEL- y por tanto es necesario emplear instrucciones de referencia a memoria con operandos cuya dirección efectiva coincida con la del registro sobre el que se quiere actuar.

La VIA tiene asignada por hardware una dirección del microprocesador. Esta dirección es la dirección base para acceder a todos los registros de la VIA. Para acceder a cada uno los registros incrementaremos en 2 unidades la dirección base.

En el caso del entrenador TM683 la VIA1 es accesible a partir de la dirección $60001 y podremos acceder a los registros con las direcciones

```
VIA EQU $60001
ORB EQU VIA + 0
ORA EQU VIA + 2 ;$60003
DDRB EQU VIA + 4 ;$60005
DDRA EQU VIA + 6 ;$60007
```

Donde se han definido etiquetas para facilitar la programación con solo añadir estas líneas al código en nuestros programas.

Así, en nuestro caso, la instrucción:

MOVE.B #$0F,$60007

programa como salidas las 4 líneas menos significativas PA0, PA1, PA2 y PA3 de la VIA1 y las 4 más significativas como entradas.

Para transferir los datos desde los registros al periférico usaríamos:

MOVE.B #$05,$60003

Hace que las líneas PA0 y PA2 de la VIA1 se pongan a "1" lógico (5 volt.) y las líneas PA1 y PA3 pasan a "0" lógico (0 volt.). El estado de las líneas PA4,...,PA7 no se verá afectado.

Para leer los datos del periférico haremos:

MOVE.B $60003,D0

Carga en los 4 bits menos significativos del registro D0 con el valor lógico al que se encuentran las líneas PA4, ... , PA7 programadas como entrada .

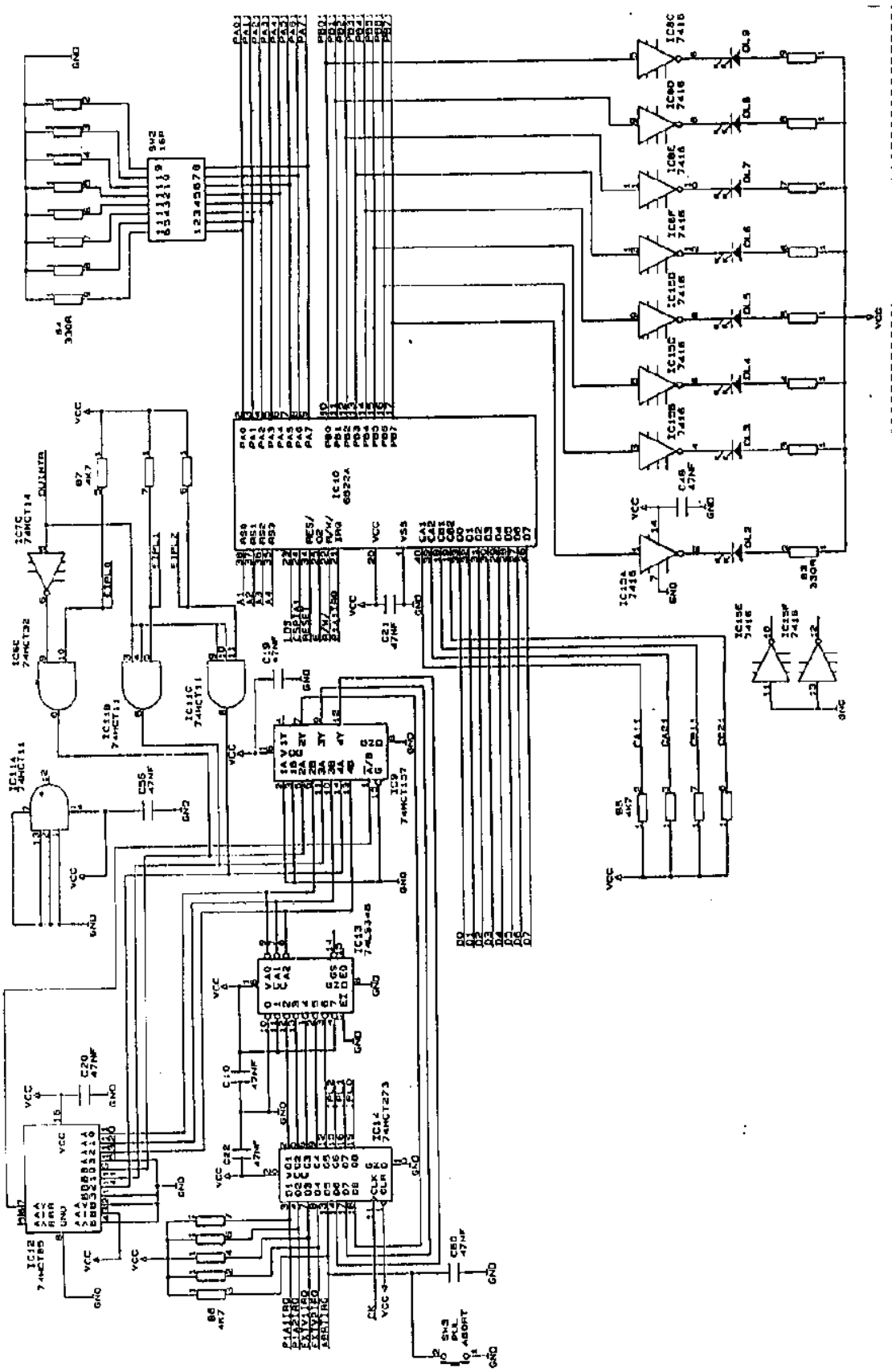
2.1.4. Conexión de la VIA con dispositivos externos.

En la placa base del entrenador TM386 tenemos conectados las 8 líneas del puerto A de la VIA1 a un conjunto de 8 interruptores (switches) y las 8 líneas del puerto B a 8 leds a través de 8 inversores-drivers 7416 que además de invertir la señal proporcionan a los leds la corriente que necesitan.

En la siguiente página se muestra el esquema de conexión.

Con esta configuración podemos emplear el puerto A como entrada de datos que pondremos mediante los switch y el puerto B como salida para visualizarlos mediante los leds.

Para la conexión con otro dispositivos exteriores tales como tarjetas de aplicación o cualquier otro periférico disponemos de dos conectores de aplicación que nos dan acceso a las líneas PA1,... PA7 y PB1,... PB7 tanto de la VIA1 como de la VIA2.



2.1.5. Ejemplos de programación de la VIA.

Coger datos de un port de entrada (PORT A) y almacenarlos en la posición de memoria $26000.

```
VIA EQU $60001
PORTA EQU VIA + 2
DDRA EQU VIA + 6
MEM EQU $26000
MOVE.B #0,DDRA ;Inicializar port A como entrada, DDRA = 0
MOVE.B PORTA,MEM ; Leer el dato y almacenarlo en memoria
```

Mandar el dato almacenado en la posición de memoria $26000 a un port de salida (PORT B).

```
VIA EQU $60001
PORTB EQU VIA + 0
DDRB EQU VIA + 4
MEM EQU $26000
MOVE.B #$FF,DDRB ;Inicializar port B como salida, todos los bits de DDRB a 1
MOVE.B MEM, PORTB ; Leer el dato de memoria y sacarlo por las líneas PB0,..., PB7
```

2.2. E/S paralelo con espera de respuesta.

En esta forma de E/S, se utilizan mecanismos a través de los cuales los dispositivos periféricos informan al computador de su estado, y éste actúa en consecuencia. Además la E/S con espera de respuesta puede llevarse a cabo de dos maneras diferentes.

1. Sin interrupciones.
2. Con interrupciones.

2.2.1. E/S con espera de respuesta sin interrupciones.

El programa debe estar en todo momento pendiente - o al menos, chequear a intervalos de tiempo lo más reducido posible - del conjunto de señales que le informan del estado de los periféricos. La ventaja de este tipo de comunicación es que el procesador cuando detecta algún cambio puede reaccionar casi de manera inmediata.

2.2.2. E/S con espera de respuesta con interrupciones.

El programa no se ocupa en absoluto de chequear es estado de los periféricos, ya que se dispone de un mecanismo capaz de interrumpir al programa en su tarea en el momento en que se produce un cambio en dicho estado, de forma que éste no vuelve a retomar su tarea hasta que termina de procesar la interrupción.

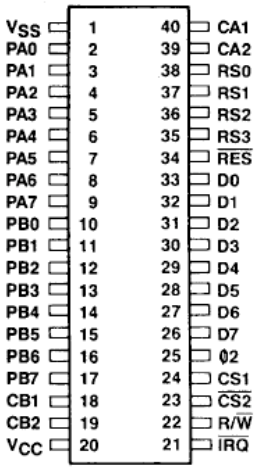
Para utilizar la VIA en este tipo de E/S ya hemos indicado que dispone para este fin de 4 líneas exteriores de control: CA1, CA2, CB1 y CB2. Las dos primeras están asociadas al port A, y las otras dos al port B, aunque el funcionamiento de ambos pares es idéntico. Las líneas CA1 y CB1 se utilizan para detectar el estado del dispositivo periférico, mientras que a través de CA2 y CB2 se generan las señales para el control del mismo.

La VIA puede programarse para reconocer tanto transiciones de 0 a 1 como de 1 a 0 a través de estas líneas, pero para ello se necesita conocer el funcionamiento de otros registros involucrados tal como el PCR y el IFR. Por tanto antes de continuar con la E/S con espera de respuesta, vamos a estudiar como funcionan los distintos registros que componen la VIA tanto sus características como su funcionabilidad.

3.- Interfase de la VIA 6522 con el procesador.

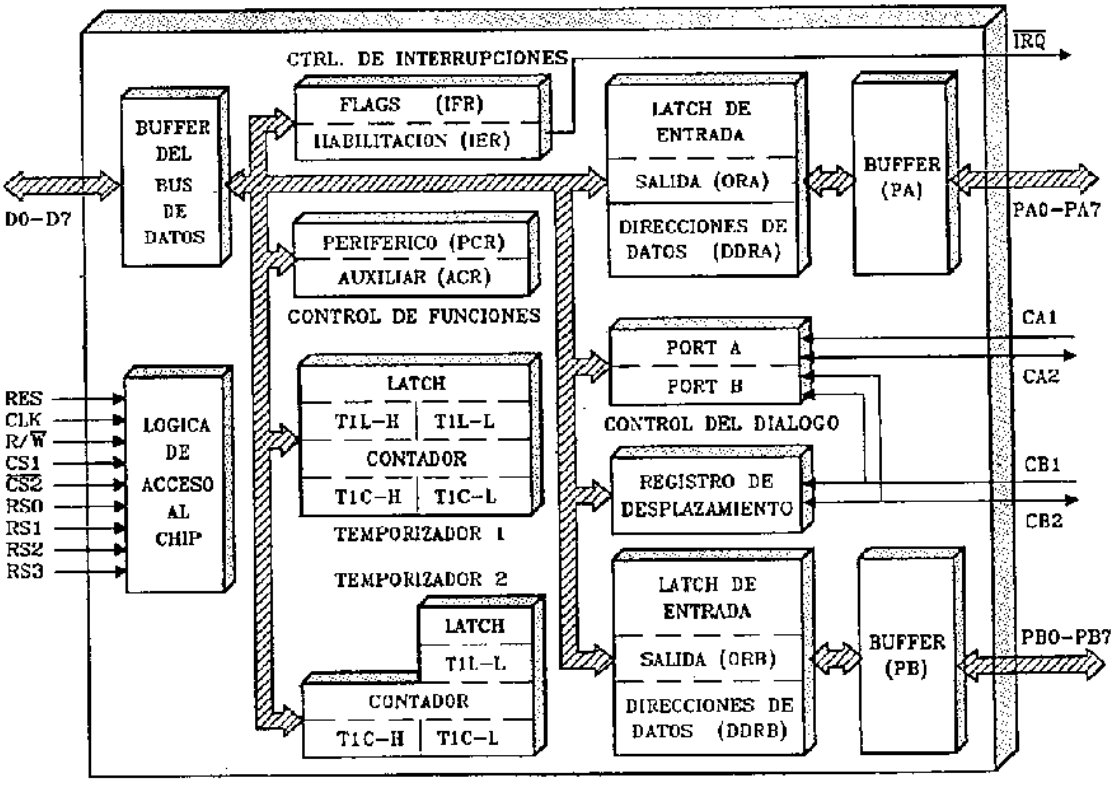
La conexión al sistema se realiza a través de los siguientes terminales:

- Terminales del bus de datos. Son ocho pines D0...D7 bidireccionales con la posibilidad de alta impedancia, para lo cual cuentan con los correspondientes buffers.
- Terminal R/W. Sirve para indicar a la VIA si la operación realizada es de lectura o de escritura.
- Entrada de reloj F2 . Se conecta en sistemas basados en el 68000 a la señal E, utilizándose para coordinar las transferencias entre los registros internos de la VIA así como para actuar como el reloj que gobierna las transiciones de los temporizadores y del registro de desplazamiento.
- Terminales de selección de pastilla. Son los pines CS1 y CS2 . En sistemas basados en el 68000 la lógica de selección debe tener en cuenta las señales UDS o LDS, según se conecte la parte alta del bus de datos -los bytes de dirección par- o la parte baja - dirección impar.
- Terminales de selección de registros. Los pines RS0, RS1, RS2 y RS3 sirven para seleccionar uno de los 16 registros internos. Se conectan a los terminales más bajos del bus de direcciones del sistema para adjudicar direcciones consecutivas a los registros. En el caso del 68000 se conectan RS0 a A1, RS1 a A2, RS2 a A3 y RS3 a A4. Cada VIA ocupa 32 bytes del mapa de memoria de los que solo se utilizan la mitad, en nuestro caso los impares porque tenemos conectado la parte baja del bus de datos.
- Terminal de IRQ. Es un terminal de salida de la VIA que se conecta al nivel deseado de la entrada del codificador de interrupciones de la CPU. A través de esta conexión se mandan las peticiones de interrupción que la VIA puede generar.
- Terminal de RESET. Es un terminal de entrada de la VIA que se conecta a la línea de RESET del sistema, actuando simultáneamente al de la CPU. Da lugar a que todos los registros de la VIA se pongan a cero.



R6522 Pin Configuration

Desde estos terminales podemos acceder a los elementos que integra la VIA y recordando que es un chip especializado en funciones de entrada / salida tenemos dos puertos de E/S configurables, dos temporizadores, un registro de desplazamiento y los registros de control necesarios para dirigir su funcionamiento como se muestran en el siguiente diagrama de bloques.



3.1. C.I R6522 de ROCKWELL : VIA (Versatil Interface Adapter).

En la siguiente tabla se dan las direcciones relativas de estos registros.

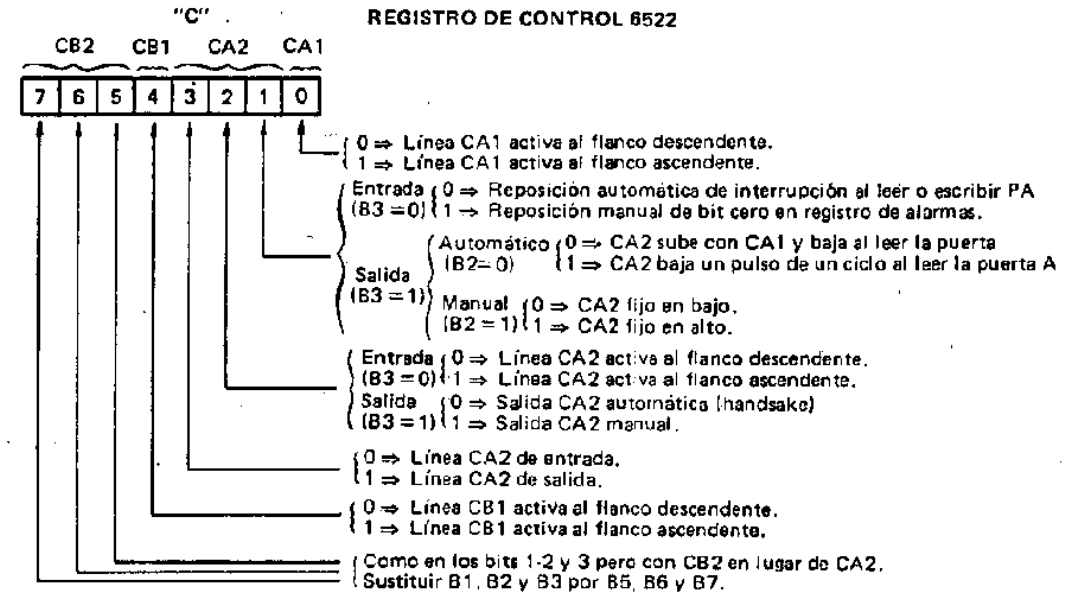
DIRECCIÓN RELATIVA	REGISTROS	REGISTRO ACCEDIDO
0	PB	REGISTRO DE ENTRADA o SALIDA PUERTO-B
1	PA	REGISTRO DE ENTRADA o SALIDA PUERTO-A Utiliza los terminales CA1 y CA2
2	DDRB	REGISTRO DE DIRECCIONES PUERTO-B: 0 -> Entrada y 1 -> Salida
3	DDRA	REGISTRO DE DIRECCIONES PUERTO-A: 0 -> Entrada y 1 -> Salida
4	T1C_L	BYTE BAJO DEL REGISTRO CONTADOR DE T1 Escribe el valor inicial (byte bajo) del contador T1 Lee el valor actual (byte bajo) del contador T1
5	T1C_H	BYTE ALTO DEL REGISTRO CONTADOR DE T1 Escribe el valor inicial (byte alto) del contador T1 Inicializa la cuenta. Se repone la interrupción. Lee el valor actual (byte alto) del contador T1
6	T1L_L	BYTE BAJO DEL REGISTRO LATCH DE T1 Guarda el valor inicial (byte bajo) del contador T1
7	T1L_H	BYTE ALTO DEL REGISTRO LATCH DE T1. Guarda el valor inicial (byte alto) del contador T1. Al escribir repone la interrupción. No inicializa la cuenta.
8	T2C_L	BYTE BAJO DEL REGISTRO CONTADOR DE T2. Escribe el valor inicial (byte bajo) del contador T2. Lee el valor actual (byte bajo) del contador T2.
9	T2C_H	BYTE ALTO DEL REGISTRO CONTADOR DE T2. Escribe el valor inicial (byte alto) del contador T2. Inicializa la cuenta. Se repone la interrupción. Lee el valor actual (byte alto) del contador T2
A	SR	REGISTRO DE DESPLAZAMIENTO. Actúa como memoria en el modo 000. Al leer o escribir, repone la interrupción
B	ACR	REGISTRO AUXILIAR DE CONTROL.
C	PCR	REGISTRO DE CONTROL (CA1,CA2,CB1,CB2)
D	IFR	REGISTRO DE ALARMAS DE INTERRUPCIONES.
E	IER	REGISTRO DE ACTIVACIÓN DE INTERRUPCIONES.
F	PA	REGISTRO DE ENTRADA o SALIDA PUERTO-A. No utiliza los terminales CA1 y CA2

La correspondencia entre las direcciones relativas de memoria y los registros internos de la VIA viene determinada por la conexión de las 4 líneas de menos peso del bus de direcciones con las patillas RS0, RS1, RS2 y RS3.

La misión de los diferentes registros de la VIA se detalla a continuación.

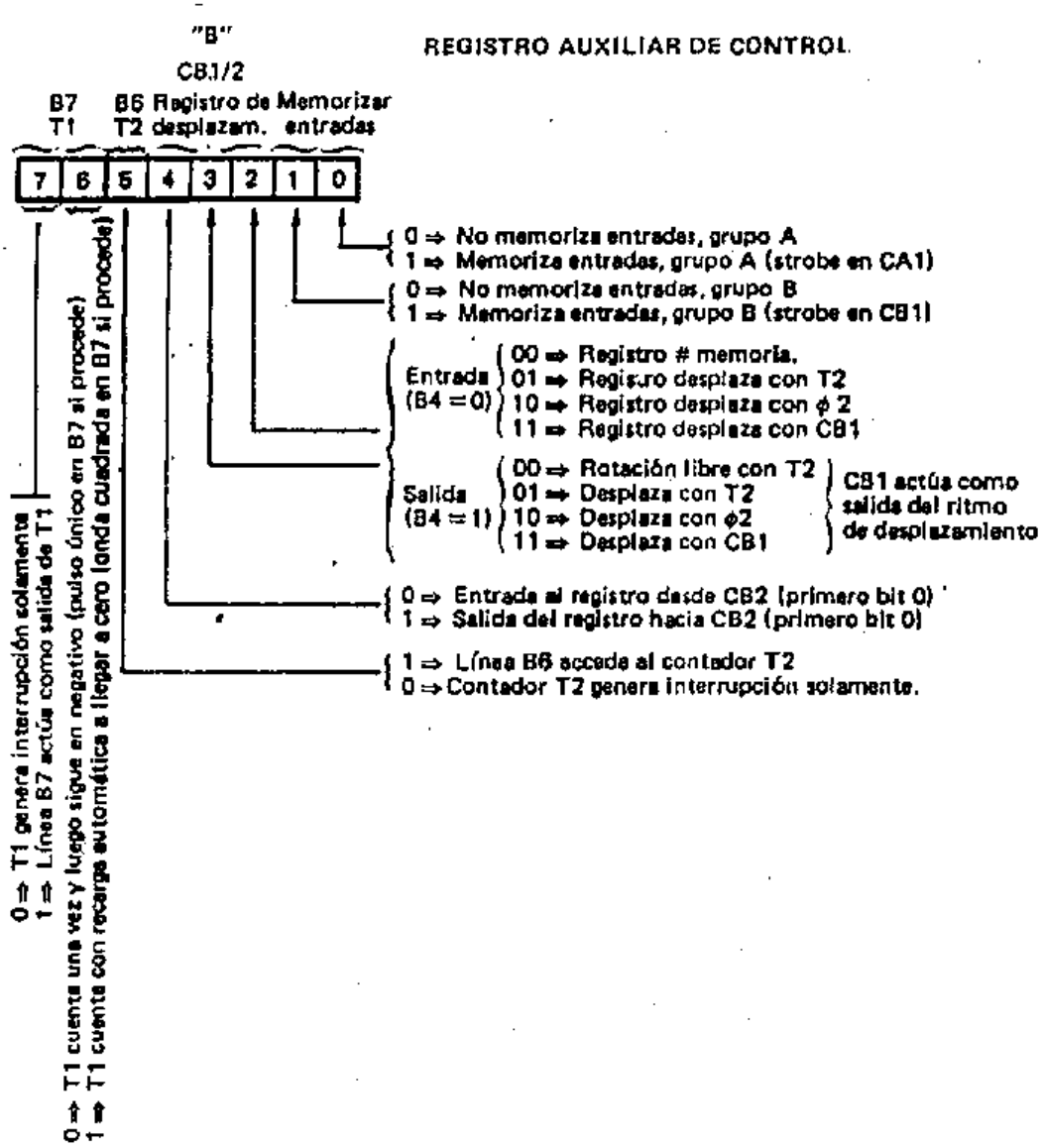
- **Registro de dirección de líneas del PORT A.** Su contenido determina los pines del puerto A que actúan como entrada o como salida. Dirección relativa 3.
- **Registro de dirección de líneas del PORT B.** Su contenido determina los pines del puerto B que actúan como entrada o como salida. Dirección relativa 2.
- **Registro de control de periféricos (PCR).** Determina la polaridad de las tensiones activas sobre las líneas de entrada de estado CA1, CA2, CB1 y CB2. Dirección relativa C.

La función de cada bit del registro de control (PCR) se detalla en la siguiente figura.

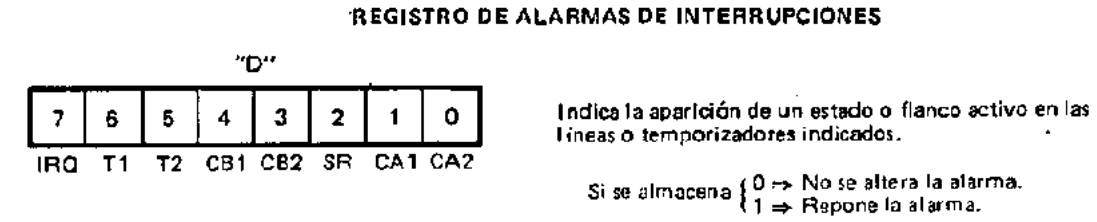


- **Registro auxiliar de control (ACR).** Determina la forma de trabajo de las puertas de entrada y salida, de los contadores/temporizadores y del registro de desplazamiento. Dirección relativa B.

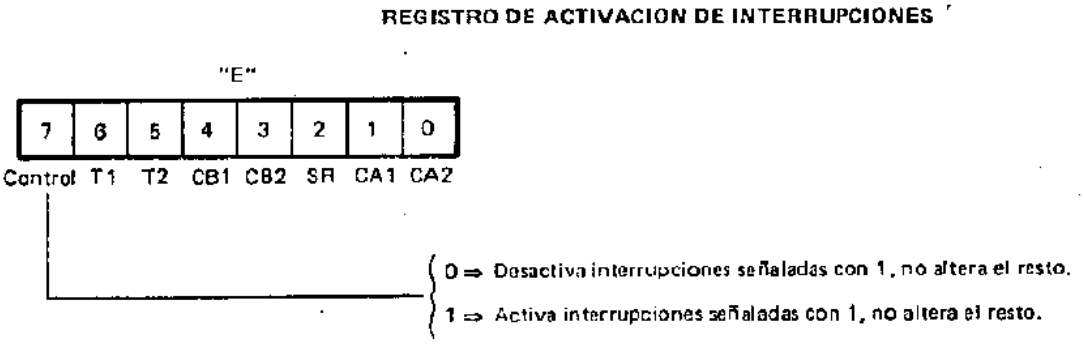
Su configuración interna que determina la misión de cada uno de los bits que lo componen se expone en la figura.



- **Registro de banderas de interrupciones (IFR).** Este registro contiene las banderas (flags) de cada una de las posibles fuentes de interrupción de la VIA. Cada uno de sus bits advierte de la presencia de una interrupción en los diversos elementos de la VIA tal como se indica en la figura. Para determinar si ha tenido lugar alguna interrupción, bastará con examinar si el bit 7 de IFR está a 1. Dirección relativa.



- **Registro de activación de interrupciones (IER).** Cada uno de sus bits sirve para activar o desactivar las interrupciones correspondientes a los elementos de la VIA. El significado de cada bit se da en la siguiente figura.



Una vez conocido las posibilidades de cada registro, vamos a ver como se aplican al funcionamiento de la VIA como generador y controlador de interrupciones, dejando para más tarde el estudio de E/S paralelo con protocolo y el tratamiento de la información en serie mediante el registro de desplazamiento.

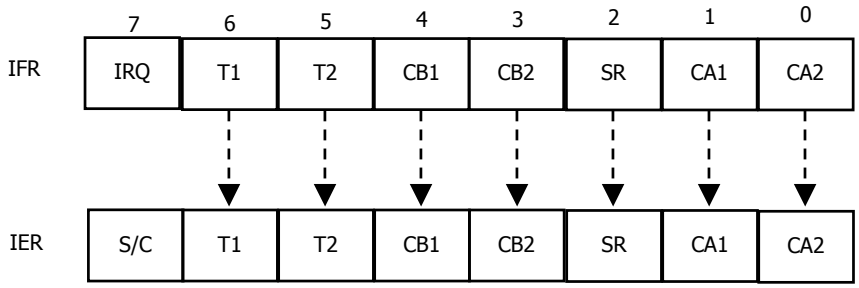
4. Generación y control de interrupciones en la VIA 6522.

La VIA puede manejar 7 posibles fuentes de interrupciones:

1. Las procedentes de la línea CA1.
2. Las procedentes de la línea CA2.
3. Las procedentes de la línea CB1.
4. Las procedentes de la línea CB2.
5. Las procedentes del temporizador T1.
6. Las procedentes del temporizador T2.
7. Las del registro de desplazamiento SR.

En principio, cada una de estas fuentes activa una bandera (flag), situado en el registro de banderas de interrupción IFR (Interrupt Flag Register). El que la interrupción se transmita o no a la CPU, a través del terminal IRQ, depende de que estén o no habilitadas. Este papel lo desempeña el registro de habilitación de interrupciones IER (Interrupt Enable Register) cuyos bits se corresponden de uno en uno con los del IFR, actuando como una máscara que permite habilitar o prohibir cada una de las posibles fuentes de interrupción.

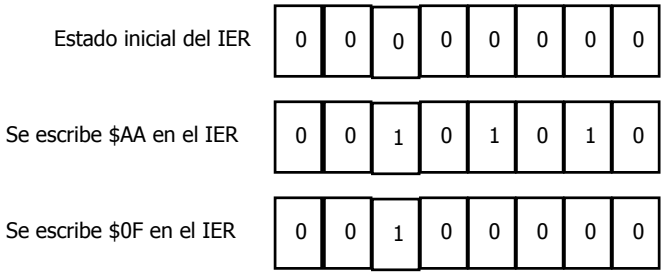
En la figura se muestra esta disposición donde podemos ver como el registro IFR contiene los flags de cada una de las posibles fuentes de interrupción de la VIA y como el registro IER (Interrupt Enable Register) permite enmascarar individualmente a cada una de ellas.



Para que una interrupción esté habilitada es preciso que el bit correspondiente de IER esté a 1. El registro de máscara IER puede ser escrito desde la CPU, aunque de una manera algo especial:

- Al escribir sobre IER un byte cuyo bit 7 es 1 (modo SET), se ponen a 1 todos los bits de IER que se corresponden con bits a 1 del byte escrito.

- Al escribir sobre IER un byte cuyo bit 7 es 0 (modo RESET) se ponen a 0 todos los bits de IER que se corresponden con bits a 1 del byte escrito.
- En ambos casos, los bits de IER que se corresponden con bits a 0 del byte escrito permanecen en su estado anterior. De este modo es fácil cambiar el contenido de bits aislados de IER sin necesidad de leer el registro previamente ni de llevar un meticuloso control de su contenido anterior.



EJEMPLO

Al final de este proceso sólo resulta permitida la interrupción proveniente de T2. Notar que al leer IER el bit 7 siempre aparecerá como cero, ya que su única misión es de ayuda para programar los restantes.

El bit 7 de IFR también tiene un sentido diferente del resto de bits del registro. Su estado 0 o 1 es un resumen de los restantes e indican el estado definitivo del terminal IRQ de la vía. Esta bit denominado IRQ cumple la función lógica:

IRQ = IFR6×IER6 + IFR5×IER5 + … + IFR0×IER0

Su utilidad se encuentra en que cuando la CPU recibe una petición de interrupción, puede explorar más fácilmente cuál de las VIAs conectadas al sistema es el origen de la misma. Para ello leerá sus registros IFR y comprobará el estado del bit 7 de cada una.

Independientemente de que la máscara permita o no la que la activación de un bit de IFR se transmita a la CPU, cada uno de ellos puede actuar como flag para identificar la ocurrencia de un suceso en el interior de la VIA.

Es importante tener en cuenta que después de activarse el bit correspondiente de IFR por que ha sucedido un suceso y generarse la correspondiente interrupción debemos ser nosotros los encargado de ponerlo a cero.

No existe la posibilidad de poner a 1 los flags desde el programa. Por el contrario, sí es posible ponerlos a 0. Para ello, el programa escribirá en IFR un byte que tenga 1 en aquellos bits que se desea desactivar.

MOVE.B #$0F,IFR

Pondrá a 0 los bits 0,1,2 y 3 en caso de que estuvieran a 1.

Además existen automatismos de puesta a 0 que simplifican los programas que manejan la VIA. En la tabla siguiente se detallan que posibles sucesos afectan a los bits de IFR

BIT	*Se pone a 1 por*	*Se pone a 0 por*
0	Flancho activo en CA2	Leer o escribir en puerto A (con direccion 0001)
1	Flanco activo en CA1	Idem.
2	Al 8º desplazamiento en SR	Leer o escribir en SR
3	Flanco activo en CB2	Leer o escribir en puerto B
4	Flanco activo en CB1	idem.
5	Paso por 0 de T2	Lectura del byte bajo o escritura en el byte alto de T2
6	Paso por 0 de T1	Lectura del byte bajo del contador de T1 o escritura en el byte alto del lacth de T1

El bit 7 de IFR sólo funciona de forma indirecta. Por tanto, no puede ser borrado por este procedimiento, sino que se necesita desactivar los bits que lo han originado.

De nuevo decir que es el programador el encargado de hacer el reset del flag y poner IRQ a 0 en el proceso de atención a la interrupción solicitada. Si no se hace permanecerá a 1 y siempre entrara en la interrupción bloqueando al microprocesador.

4.1.- Funcionamiento de los TEMPORIZADORES.

Los dos temporizadores que dispone la vía T1 y T2 son ligeramente diferentes. Vamos a describir en primer lugar el T2 que aunque dispone de menos funciones que el T1 es más simple de manejar.

4.1.1.- Temporizador T2.

Consta de un contador de 16 bits (T2C-H y T2C-L), que cuenta hacia atrás, y de un latch de 8 bits (T2L-L) que permite almacenar el byte bajo del valor inicial. Ocupa dos posiciones de memoria, una para los 8 bits menos significativos y otra para los 8 más significativos.

Puede programarse para realizar varias funciones dependiendo del valor que escribamos en el bit 5 del registro de control ACR:

- Generación de retardos, haciendo el bit 5 de ACR = 0.
- Detección de un número determinado de pulsos negativos introducidos por PB6, haciendo el bit 5 de ACR = 1.

La conclusión de cualquiera de las funciones indicadas ocasiona la puesta a 1 del bit 5 del registro de banderas de interrupción IFR. Entonces si tenemos habilitadas las interrupciones correspondientes al T2, es decir si el bit 5 del registro de habilitación de interrupciones está a 1 se provoca una petición de interrupción por la patilla IRQ de la VIA.

La rutina de atención a IRQ debe ser la encargada de volver a poner el flag de T2 a cero. Este bit se pone a cero automáticamente como consecuencia de la lectura de T2-L o la escritura de T2-H.

Modo de operación.

La puesta en marcha de T2 se efectúa en dos etapas:

1. Se carga la parte baja T2C-L del temporizador con el valor deseado
   MOVE.B #$42,$60031
2. Se carga la parte alta T2C-H lo que provoca el inicio del decremento del valor almacenado.
   MOVE.B #$05,$60032

Cuando se utiliza como generador de retardos, se carga en T2 el número de ciclos que se desea retardar, teniendo en cuenta que la VIA funciona con la señal de reloj generada por la patilla E del microprocesador. Cuando se utiliza como detector de pulsos se carga el número de pulsos a detectar. Teniendo en cuenta la precaución de cargar en segundo lugar la parte alta de T2, ya que en ese instante comienza a decrementarse y cuando llegue a 0 se pone a 1 el bit 5 de IF.

Ejemplo : Generación de un retraso de 2048 microsegundos suponiendo que el reloj que utiliza la VIA es de 1 MHz.

```
MOVE.B #$00,$60037 ;Programa T2 en modo temporizador.
MOVE.B #$00,$60031 ;Carga T2 con 2.048 (hex 800).
MOVE.B #$08,$60033 ;Pone el flag T2 = 0 y comienza a decrementar.
L1: MOVE.B $6003B,D0
AND.B #%00100000,D0 ;Mira si el flag T2 = 1
BEQ L1 MOVE.B $600031, D0 ; Lee T2C-L para poner el flag T2=0
```

4.1.2.- Temporizador T1 .

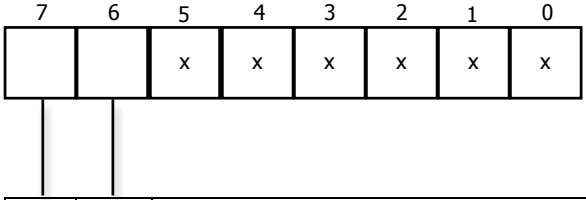
Consta de un latch de 16 bits (T1L-H y T1L-L) y de un contador de 16 bits asociado (T1C-H y T1C-L) que cuenta hacia atrás. Ocupa cuatro posiciones de memoria, una para los 8 bits menos significativos y otra para los 8 más significativos.

El latch asociado tiene la misión de almacenar el valor inicial de la cuenta. Su contenido no es decrementado y puede utilizarse para restituir el valor incicial cuando T1 se programa en el modo de recarga automática.

El contador/temporizador T1 tiene dos modos básicos de funcionamiento.

- Modo repetitivo con recarga automática (AESTABLE)
- Modo cuenta única (MONOESTABLE)
- Además, puede ser programado para que sus efectos se transmitan al exterior, para lo que utiliza el terminal PB7 del puerto B. Este terminal se comporta como salida cuando se selecciona este modo de funcionamiento independientemente de cómo haya sido configurado previamente.

Los modos de funcionamiento vienen controlados por los bits 6 y 7 de registro de control ACR, de la siguiente forma:



0	0	Modo MONOESTABLE sin salida hardware
0	1	Modo AESTABLE sin salida hardware
1	0	Modo MONOESTABLE con salida hardware
1	1	Modo AESTABLE con salida hardware

Modo de operación.

La puesta en marcha de T1 se efectúa en dos etapas:

1. Se carga la parte baja T1C-L del temporizador con el valor deseado
   MOVE.B #$42,$60029
2. Se carga la parte alta T1C-H lo que provoca el inicio del decremento del valor almacenado.
   MOVE.B #$05,$6002B

Para conocer en cualquier momento el estado del contador podemos leer las posiciones de memoria de T1C-L y T1CH mientras que si queremos ver el valor cargado inicialmente miraremos en T1L-L y T1L-H.

La carga de valores en T1C-L y T1C-H provoca el paso de su contenido al contador y el inicio del decremento. En cambio, la escritura en T1L-L y T1L-H sólo se transfiere al contador, cuando trabaja en recarga automática y se alcanza el valor cero.

Funcionamiento en modo de cuenta única (MONOESTABLE)

En este modo sólo es de interés la cuenta atrás del contador desde un valor inicial hasta cero. Cuando se carga el valor inicial, al iniciarse la cuenta, el flag de T1 se pone a 0 y la salida en PB7, si está habilitada se pone a cero.

La cuenta atrás prosigue hasta el valor 0000. En ese momento ocurren los siguientes sucesos:

- Se pone a 1 el flag de T1 en IFR.
- Se transmite una interrupción si está habilitada.
- La salida PB7 (si está habilitada) vuelve a su estado de reposo 1.

Una vez alcanzado el primer paso por cero, el temporizador continúa contando hacia atrás a partir del valor FFFF (-1) y el proceso se repite indefinidamente, pero sucesivos pasos por cero no dan lugar a una repetición de los procesos que acabamos de describir.

Funcionamiento en modo recarga automática (AESTABLE).

El modo de recarga automática se caracteriza por la repetición del proceso de cuenta atrás a partir de un valor inicial hasta cero. Al llegar a cero, el contador recupera el valor inicial, almacenado en el latch (de ahí su necesidad), repitiendo el mismo ciclo.

Los efectos del paso por 0 son:

- Si está habilitada la salida PB7, este terminal cambia de estado cada vez que el contador pasa por cero (Genera una señal de frecuencia conocida).
- El flag T1 se pone a 1 cada vez que se produce un paso por cero. Es preciso que el programa (posiblemente la rutina de atención a IRQ) se encargue de hacer el reset de este bit en cada ciclo para que este efecto sea apreciado.

Si están habilitadas las interrupciones de T1, se producirá una interrupción IRQ en cada ciclo, coincidiendo con la activación del flag de T1.

CAPITULO 7

Programación de la DUART 68681

1. Introducción.

2. Interfaz Normalizada.

3. Descripción de la DUART 68681.

4. Programación como Emisor y Receptor.

5. Ejemplos.

1. Introducción.

La transmisión de datos en serie no requiere tantas líneas como la transmisión en paralelo, ya que cada uno de los bits que componen el dato se transmiten consecutivamente de manera secuencial siguiendo un orden establecido. Aunque la transmisión en serie es más lenta que la transmisión en paralelo es ampliamente utilizada, por motivos económicos, cuando la distancia física que separa al transmisor del receptor es grande.

Además es apropiada cuando se utilizan como soporte de comunicación líneas telefónicas. Al igual que en la transmisión paralelo se establecen los conceptos de comunicación síncrona y asíncrona.

- La **comunicación síncrona** es aquella que está sometida a una rígida temporización, la cual va a permitir que el elemento receptor sea capaz de conocer en que instante la señal que le llega tenga validez.
- En la **comunicación asíncrona**, los datos pueden ser transmitidos en cualquier momento, pero el dispositivo transmisor debe enviar al receptor una señal auxiliar que indique cuando son válidos los datos, es decir, se emplean señales auxiliares de protocolo.

2. Interfases normalizadas.

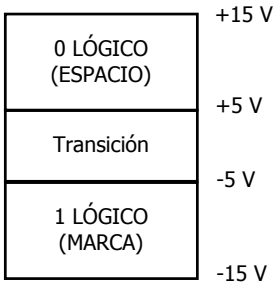
La transmisión de datos a través de redes de telecomunicación públicas se hacen en serie utilizando MODEMS (dispositivo modulador-demodulador que convierte los datos digitales en una señal modulada dentro del ancho de banda que requiere el canal utilizado) consiguiéndose la transmisión de señales digitales de una forma eficiente. Para que esta conexión funcione se debe seguir una norma entre receptor o transmisor y el equipo que se encarga de comunicar. La conexión entre TERMINAL emisor o receptor, Data Terminal equipment (DTE), y COMUNICADOR Data Comunication Equipment (DCE) se encuentra normalizada por organismos oficiales.

2.1 RS232C

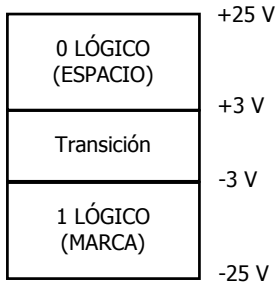
La norma RS232C es una norma americana de EIA muy difundida. Originalmente fue definida para realizar transmisiones de datos utilizando módems a través de líneas telefónicas ya instaladas. Define las características que deben cumplir las líneas de interconexión entre el equipo transmisor o receptor de datos (DTE) y el equipo que se encarga de comunicar estos datos a través de la línea telefónica (DCE), el cual será un MODEM. Actualmente esté tipo de conexión se adopta para otras funciones bien distintas tal como establecer una línea de transmisión entre dos equipos transmisores o receptores de datos (conexión DTE DTE) utilizando la norma RS232 , como pueden ser dos ordenadores comunicados entre si.

Características

Niveles Lógicos : Se utilizan un niveles bipolares, con un amplio margen de tensiones definidos por el siguiente esquema



Niveles lógicos para SALIDAS



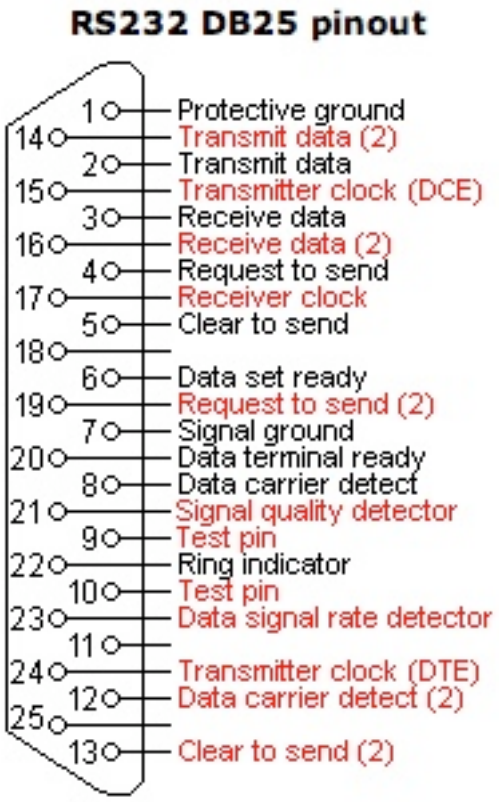
Niveles lógicos para ENTRADAS

Las características eléctricas se resumen en el siguiente cuadro.

DESCRIPCIÓN		MIN	MAX
Tensión de salida del transmisor:	***Nivel alto=0***		+25 V
(en circuito abierto)	***Nivel bajo=1***	-25 V	
Tensión de salida del transmisor:	***Nivel alto=0***	+ 5V	+ 15 V
(con carga de 3 a 7 kΩ)	***Nivel bajo=1***	-15 V	-5 V
Impedancia de salida del transmisor (sin alimentación)			300 Ω
Corriente del salida del transmisor (en cortocircuito)		-0.5 A	+0.5 A
Slew rate (pendiente de subida o bajada) (dV/dt)		4 V/μs	30 V/μs
Impedancia de entrada del receptor		3 kΩ	7 kΩ
Tensión de entrada del receptor:	***Espacio = 0***	+3 V	+25 V
	Marca = 1	-25 V	-3 V

Se establecen especificaciones destinadas a lograr seguridad frente a cortocircuitos accidentales.

Se define un conector específico de 25 terminales:

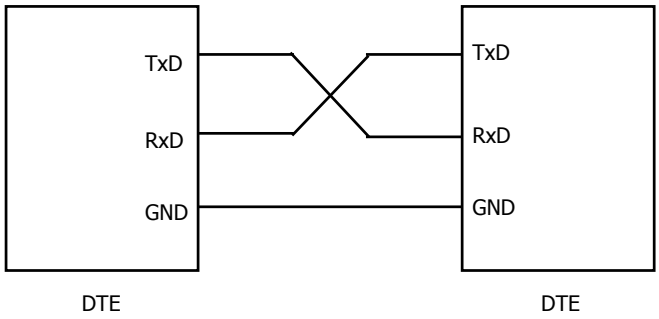


Los 25 terminales están montados en dos filas de 13 y 12 patillas. Los terminales más importantes son los siguientes:

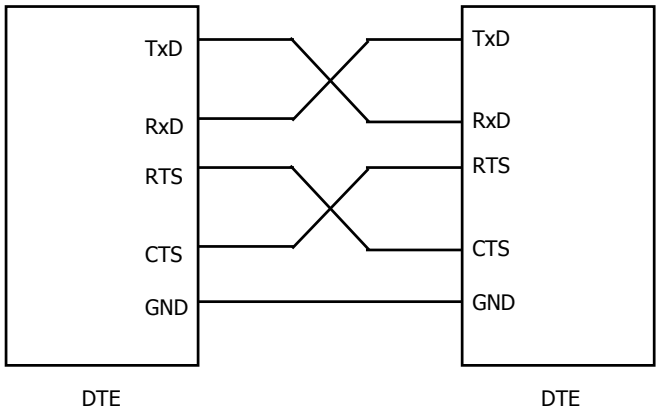
- Nº Terminal Nombre Funcion (como DTE)
- 2 TxD Salida de transmisión de datos (del DTE al DCE)
- 3 RxD Entrada de recepción de datos (del DCE al DTE)
- 4 RTS Salida de petición para enviar (del DTE al DCE)
- 5 CTS Entrada de habilitación para enviar (del DCE al DTE)
- 7 GND Terminal común de Masa.

La distancia que garantiza esta norma es de unos 16 metros, aunque con probabilidad se alcanzan distancias mayores. Con una velocidad de transmisión de 20 Kbaudios.

La conexión más sencilla para la transmisión puede hacerse con tres hilos. Para ello el protocolo debe establecerse a través de estos hilos enviando caracteres que habiliten o inhabiliten la transmisión. Este procedimiento, llamado XON/XOFF, generalmente utiliza los caracteres ASCII $13 para detener la transmisión (XOFF) y $11 para habilitar la transmisión (XON). El receptor cuando está en disposición de recibir, envía el carácter $11 al transmisor y cuando no lo está envía el carácter $13.



El protocolo puede implementarse mediante las líneas RTS y CTS



3. Descripción de la DUART 68681.

El SCN68681 es un circuito de comunicaciones que posee 2 transmisores universales en un único encapsulado. Es compatible con otros equipos de la familia MC68000 y se puede utilizar en sistemas con interrupciones.

El modo de operación y el formato de los datos de cada canal puede programarse independientemente. Adicionalmente cada receptor y transmisor puede seleccionar su velocidad de transmisión entre 18 velocidades fijas, un reloj derivado de un timer/contador programable, o un reloj externo.

Características:

- Compatible con el bus asíncrono M68000.
- Doble receptor/transmisor asíncrono completo.
- Cuatro registros receptores de datos.
- Formato de datos y velocidad de transmisión programable.
- Detección de errores (Paridad, trama, exceso, bit de comienzo falso, Break).
- Modo del canal de comunicación programable.
- Timer/contador multifunción programable de 16 bits.
- Puerto de entrada de 6 bits y de salida de 8 bits.
- Sistema de interrupción versátil.
- Alimentación única a Vcc = + 5 V.

En las siguientes figuras podemos ver tanto el patillaje como el diagrama de bloques de este chip.

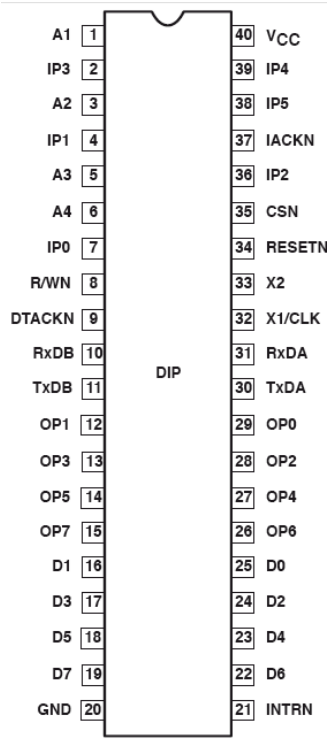
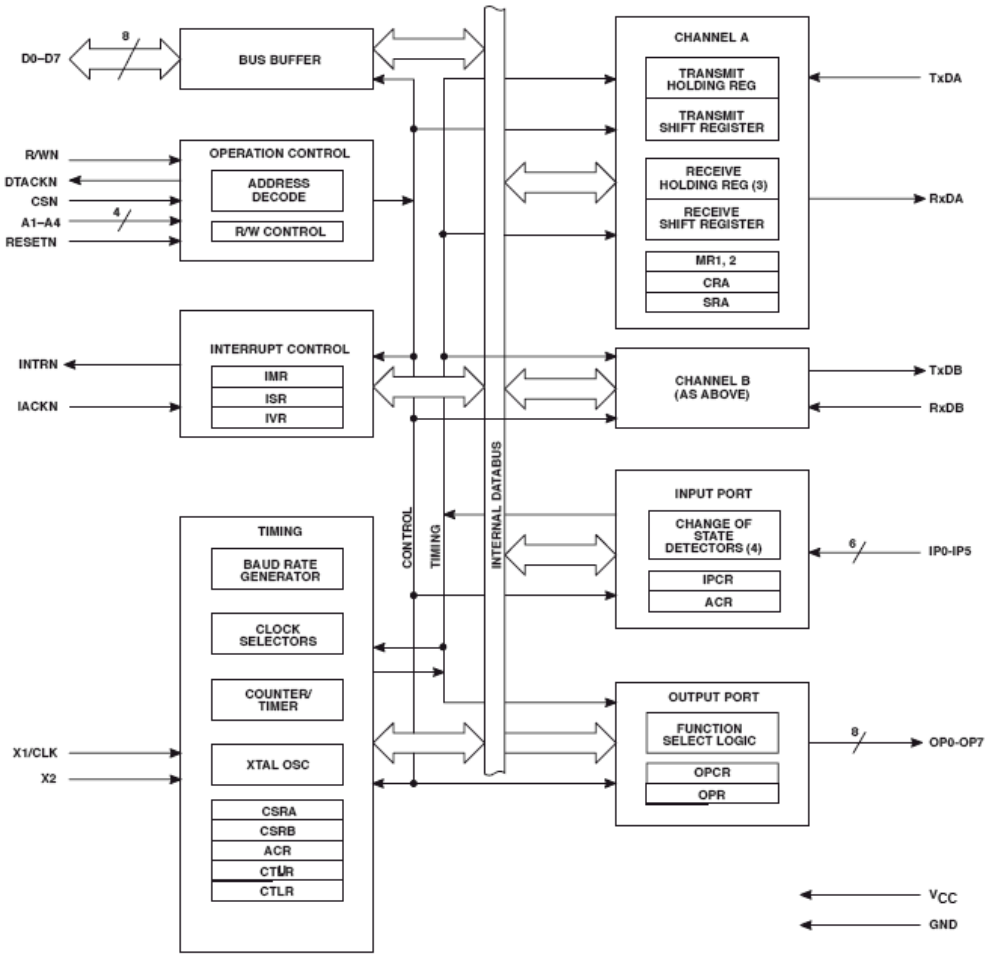


Diagrama de bloques de la DUART



La Duart consta principalmente de los siguientes elementos:

a.- Buffer del Bus de datos que proporciona la interconexión entre los buses de datos de la DUART y la CPU permitiendo operaciones de lectura y escritura.

b.- Lógica de control de operaciones que recibe las ordenes de operación de la CPU y genera las señales adecuadas a las distintas secciones internas. Contiene un decodificador de direcciones y circuitos de lectura y escritura para permitir la comunicación con el mP a través del buffer del bus de datos.

c.- Control de interrupciones. Sólo tiene una única salida de interrupciones, activa a nivel bajo, \INTR la cual se activa si ocurre cualquiera de los ocho casos internos. Asociado con el sistema de interrupciones está el registro de la máscara de interrupciones (IMR), el registro de estado de interrupciones (ISR), el registro de control auxiliar (ACR) y el registro de vector de interrupciones IVR.

El IMR debe programarse para seleccionar las condiciones que provocarán la activación de \INTR. El (ISR) se puede leer mediante el mP para determinar todas las condiciones interrupción activas.

d.- Circuitos de temporización. Consta de un oscilador, un generador de velocidades de transmisión discretas, un timer/contador de 16 bits y 4 selectores de reloj.

e.- Canales de comunicación A y B. Cada canal de comunicación del SCN68681 comprende un receptor/transmisor asíncrono (UART). La frecuencia de operación para cada receptor y transmisor puede seleccionarse independientemente.

El emisor, acepta datos en paralelo de la CPU y los convierte en un conjunto de bits en serie, inserta el bit adecuado de arranque y de parada y opcionalmente el bit de paridad y saca los datos en serie por el pin TxD. El emisor dispone de dos registros, uno el registro de espera THR, donde se escribe el dato a transmitir, otro el registro de desplazamiento TSR, donde se van desplazando cada uno de los bits para su transmisión. El receptor, acepta datos en serie por el pin RxD, los pasa a un formato paralelo, chequea el bit de arranque, el de parada y el bit de paridad (si lo hay), o la condición de ruptura y envía un carácter paralelo a la CPU.

El receptor dispone de 4 registros, uno de ellos es el registro de desplazamiento, donde se reciben los bits y se van desplazando hasta formar el carácter completo, los otros 3 son registros de espera en donde se sitúan los caracteres después de haberse recibido y donde el mP leerá los datos; siempre se lee el primer dato que se ha recibido.

f.- Port de entrada. Las entradas de este puerto de 6 bits, no latcheado pueden leerse en la direccion relativa $D. El bit 7 siempre se leerá como 1 y el bit 6 del dato reflejará el nivel de la línea \IACK, el resto de bits leerán un 1 lógico si la entrada está a nivel alto y un 0 lógico si la entrada está a nivel bajo.

Además asociados a las entradas IP3...IP0 hay detectores de transición de alto-bajo o bajo-alto de una duración mayor de 25-50 ms que ponen activo el bit correspondiente en el registro de cambios de port de entrada (IPCR). Los bits son desactivados cuando la CPU lee el registro. Cualquier cambio de estado puede programarse para generar una interrupción.

g.- Port de salida de 8 bits multipropósito. Las salidas son las complementarias del registro del puerto de salida OPR. OPR(n)=1 da lugar a un nivel bajo en OP(n) y viceversa. Los bits de OPR pueden desactivarse y activarse individualmente. Un bit se activa, pasa a 1, realizando una operación de escritura en la dirección relativa $E, con el dato de la siguiente forma, 1= activa el bit, 0 = no cambia. Un bit se desactiva, pasa a 0, escribiendo en la dirección relativa $F con el dato de la siguiente forma: 1= resetea(pone a cero), 0= no cambia.

Además se pueden asignar a estas salidas funciones especificas mediante la programación adecuada de MR1A,MR2A, MR1B ,MR2B y OPCR.

4.- Funcionamiento.

4.1- Emisor.

El SCN68681 transmite datos cuando se habilita, a través del registro de ordenes CRx. Indica a la CPU cuando está listo para aceptar un carácter activando el bit TxRDY (transmisor listo) del registro de estados SRx. Esta condición puede generar una petición de interrupción si se programa en el registro de máscara de interrupciones IMR. Cuando se carga un carácter en el registro de espera de transmisión THR, la condición anterior se desactiva.

El dato se transfiere del registro de espera THR al registro de desplazamiento de transmisión TSR, cuando este está inactivo o se ha completado la transmisión del carácter anterior. El bit TxRDY se activa cuando comienza la transmisión del bit menos significativo del dato, entonces se puede cargar otro dato en el THR y por tanto disponemos de un buffer de un carácter completo. Los caracteres no pueden cargarse en el THR cuando el transmisor está inhabilitado.

El transmisor, mediante el registro de desplazamiento TSR, convierte los datos paralelos, que la CPU ha cargado en el registro de espera THR, en un conjunto de bits serie, que son transmitidos mediante el pin de salida TxD. Automáticamente envía el bit de inicio, seguido por el número programado de bits de datos, el bit de paridad opcional y el número programado de bits de parada. El bit menos significativo se transmite primero. Después de la transmisión del bit de parada, si no se encuentra ningún carácter en el THR, la salida TxD permanece a nivel alto y el bit TxEMT (transmisor vacío) del registro de estado SR se activa a 1. La transmisión continua y el bit TxEMT se desactiva cuando la CPU carga un nuevo carácter en el THR. Si el transmisor está inhabilitado, continua operando hasta que el carácter actual que se transmite se envía por completo. Se puede forzar al transmisor a enviar continuamente un nivel bajo, enviando una orden BREAK al CRx.

4.2- Receptor.

El SCN681 queda configurado para la recepción cuando lo habilitamos ,mediante los bit 0 y 1 del registro de ordenes CRx. El receptor espera la transición de nivel alto a bajo del bit de inicio en el pin de entrada RxD. Si se detecta la transición, se muestrea el estado del pin RxD. Si RxD muestra un nivel alto, el bit de inicio no es valido y la búsqueda de un nuevo bit de inicio comienza otra vez. Si RxD está todavía a nivel bajo, se asume como válido el bit de inicio y el receptor continua muestreando la entrada a intervalos de tiempo de 1 bit, teóricamente en el

centro del bit, hasta que se lean todos los bits de datos, el bit de paridad (si lo hay) y se detecte el bit de parada. El bit menos significativo se recibe primero.

Después se transfiere el dato al registro de espera de recepción RHR y el bit RxRDY del SR se activa a 1. Se puede programar el registro de modo MR1x y el de máscara de interrupciones para que la activación de RxRDY genere una petición de interrupción.

Después de detectar el bit de parada, el receptor queda esperando el siguiente bit de inicio de otro carácter.

El registro de espera de recepción (RHR) consiste en una pila del tipo FIFO, con una capacidad de 3 caracteres. El dato se carga desde el registro de desplazamiento del receptor RSR en la primera posición vacía, de la FIFO. El bit RxRDY del registro de estado se activa cuando uno o más caracteres están disponibles para ser leídos, y el bit FFULL, se activa si las tres posiciones de la pila están completadas con datos. Cada uno de estos bits puede seleccionarse para generar una petición de interrupción.

Una lectura del RHR saca los datos de la primera posición de la FIFO.

Si la FIFO está llena, cuando se recibe un nuevo carácter, ese carácter, se almacena en el registro de desplazamiento del receptor RSR, hasta que se queda libre una posición de la FIFO. Si se recibe otro carácter mientras se está en este estado, el contenido de la FIFO no se afecta; el carácter que había en el registro de desplazamiento se pierde y el bit de error de exceso del registro de estados SR(4) se activa.

Si el receptor se inhabilita, los caracteres de la FIFO se pueden leer. Sin embargo no se puede leer ningún otro carácter hasta que el receptor se habilite de nuevo.

4.3 Programación de los REGISTROS.

El modo de operación de la DUART se programa escribiendo las palabras de control en los registros adecuados.

El contenido de ciertos registros de control, se inicializa a 0 con la operación de RESET. Se debe tener cuidado al cambiar los contenidos de un registro mientras está operando, ya que puede causar problemas. En general, el contenido del registro de modo MR1x, MR2x, el registro de selección de velocidad CSRx y el registro de control de puerto de salida OPCR solo debe de cambiarse mientras el receptor y el Transmisor estén inhabilitados.

Los registros de modo MR1x y MR2x de cada canal se acceden vía punteros auxiliares independientes. El puntero se activa a MR1x de dos modos , mediante una inicialización RESET

o mediante una orden de inicialización de puntero, programando el registro de control Crx. Cualquier lectura o escritura del registro de modo mientras el puntero está en MR1x cambia el puntero a MR2x.

La dirección de los registros se describe en la tabla 1.

A4-A3-A2-A1	LECTURA(R/W =1)	ESCRITURA (R/W = 0)
0000	MR1A , MR2A	MR1A , MR2A
0001	SRA	CSRA
0010	-	CRA
0011	RHRA	THRA
0100	IPCR	ACR
0101	ISR	IMR
0110	byte alto contador CTU	parte alta latch CTUR
0111	byte bajo contador CTU	parte baja latch CTLR
1000	MR1B y MR2B	MR1B y MR2B
1001	SRB	CSRB
1010	-	CRB
1011	RHRB	THRB
1100	IVR	IVR
1101	IPR	OPCR
1110	inicio de contador	activación OPRS
1111	parada del contador	desactivación OPRR

MR1A : Registro de modo 1 del canal A

Se accede a MR1A cuando el puntero MR de canal A apunta a MR1. El puntero se inicializa a MR1 mediante un RESET o mediante una orden de "inicialización de puntero" aplicado en CRA. Después de leer o escribir en MR1A, el puntero cambia a MR2A.

Control de RXRTS	Selección de RxINT	Modo del error	Modo de la paridad	Tipo de paridad	Bits por carácter
Bit 7	Bit 6	Bit 5	Bit 4 y Bit 3	Bit 2	Bit 1 y
0 = No	0 = RxRDY	0 = Carácter	00 = Con paridad	Con paridad	Bit 0
1 = Si	1 = FFULL	1 = Bloque	01 = Paridad forzada	0 = par	00 = 5
			10 = Sin paridad	1 = impar	01 = 6

	11 = Multiconexión		10 = 7
			11 = 8
		Paridad Forzada	
		0 = Nivel bajo	
		1 = Nivel alto	
		Multiconexión	
		0 = Dato	
		1 = Dirección	

MR2A : Registro de modo 2 del canal A Se accede a MR2A cuando el puntero MR de canal A apunta a MR2, lo cual ocurre después de cualquier acceso a MR1A. Acceder a MR2A no cambia el puntero.

Modo del canal	***Control de TxRTS***	***Control del Tx por CTS***	***Longitud de bit de parada***	
Bit 7 y Bit 6	Bit 5	Bit 4	Bit 3, Bit 2, bit 1, y Bit 0	
00 = Normal	0 = No	0 = No	6-8 bits/ carácter	5 bits/ carácter
01 = Eco automático	1 = Si	1 = Si		
10 = Bucle local			0000 = 0.563	1.063
11 = Bucle remoto			0001 = 0.625	1.125
			0010 = 0.688	1.188
			0011 = 0.750	1.250
			0100 = 0.813	1.313
			0101 = 0.875	1.375
			0110 = 0.938	1.438
			0111 = 1.000	1.500
			1000 = 1.563	1.563
			1001 = 1.625	1.625
			1010 = 1.688	1.688
			1011 = 1.750	1.750
			1100 = 1.813	1.813
			1101 = 1.875	1.875
			1110 = 1.938	1.938
			1111 = 2.000	2.000

MR1B : Registro de modo 1 del canal B Se accede a MR1B cuando el puntero MR de canal B apunta a MR1. El puntero se inicializa a MR1 mediante un RESET o mediante una orden de "inicialización de puntero" aplicado en CRB. Después de leer o escribir en MR1B, el puntero cambia a MR2B. Las definiciones de los bits para este registro, son idénticas a las de los bits de MR1A, excepto que todas las acciones de control se aplican a receptor y al transmisor del canal B.

MR2B : Registro de modo 2 del canal B Se accede a MR2B cuando el puntero MR de canal B apunta a MR2, lo cual ocurre después de cualquier acceso a MR1B. Acceder a MR2B no cambia el puntero. Las definiciones de los bits para este registro, son idénticas a las de los bits de MR2A, excepto que todas las acciones de control se aplican a receptor y al transmisor del canal B.

CSRA : Registro de selección del reloj del canal A *Dependen de ACR*

Velocidad de recepción Reloj = 3.6864 MHz			***Velocidad de transmisión Reloj = 3.6864 MHz***		
Bit 7, Bit 6, Bit 5 y Bit 4			Bit 3, Bit 2, Bit 1 y Bit 0		
	ACR[7]=0	ACR[7]=1		ACR[7]=0	ACR[7]=1
0000	50	75	0000	50	75
0001	110	110	0001	110	110
0010	134.5	134.5	0010	134.5	134.5
0011	200	150	0011	200	150
0100	300	300	0100	300	300
0101	600	600	0101	600	600
0110	1200	1200	0110	1200	1200
0111	1050	2000	0111	1050	2000
1000	2400	2400	1000	2400	2400
1001	4800	4800	1001	4800	4800
1010	7200	6800	1010	7200	6800
1011	9600	9600	1011	9600	9600
1100	38400	19200	1100	38400	19200
1101	Timer	Timer	1101	Timer	Timer
1110	IP4-16x	IP4-16x	1110	IP3-16x	IP3-16x
1111	IP4-1x	IP4-1x	1111	IP3-1x	IP3-1x

CSRB : Registro de selección del reloj del canal B (ver CSRA)

CRA : Registro de control del canal A CRA permite especificar diferentes funciones en el canal A. Se pueden especificar varias órdenes realizando una única escritura en CRA, siempre que las órdenes no entren en conflicto.

No utilizado	*Diversas órdenes*	*Órdenes de transmisión*	*Órdenes de recepción*
Bit 7	Bit 6, Bit 5 y Bit 4 000 = Ninguna orden 001 = Inicacilización de MR a MR1 010 = Inicializa receptor 011 = Inicializa transmisor 100 = Inicializa estado de errores 101 = Inicializa interrupción causda por cambio de BREAK 110 = Inicio de BREAK 111 = Paro de BREAK	Bit 3 y Bit 2 00 = No hace nada 01 = Habilita transmisor 10 = Inhabilita transmisor 11 = No utilizar, indeterminado	Bit 1 y Bit 0 00 = No hace nada 01 = Habilita receptor 10 = Inhabilita receptor 11 = No utilizar, indeterminado

CRB : Registro de control del canal B (Idem a CRA para el canal B)

SRA : Registro de estado del canal A

Recepción de BREAK	*Error de trama*	*Error de paridad*	*Error de exceso*	*Transmisor vacío TxEMT*	*Transmisor preparado TxRDY*	*FIFO llena FFULL*	*Receptor preparado RxRDY*
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 = No 1 = Si	0 = No 1 = Si	0 = No 1 = Si	0 = No 1 = Si	0 = No 1 = Si	0 = No 1 = Si	0 = No 1 = Si	0 = No 1 = Si

SRB : Registro de estado del canal B (Idem a SRA para el canal B)

OPCR : Registro de configuración del puerto de salida.

OP 7	*OP6*	*OP5*	*OP4*	*OP3*	*OP2*
Bit 7 0 = \OPR[7] 1 = \TxRDYB	Bit 6 0 = \OPR[6] 1 = \TxRDYA	Bit 5 0 = \OPR[5] 1 = \RxRDYB \FFULLB	Bit 4 0 = \OPR[4] 1 = \RxRDYA \FFULLA	Bit 3 y Bit 2 00 = \OPR[3] 01 = Salida C/T 10 = TxCB 1x 11 = RxCB 1x	Bit 1 y Bit 0 00 = \OPR[2] 01 = TxCA 16x 10 = TxCA 1x 11 = RxCA 1x

ACR : Registro de control auxiliar.

Selección de conjunto de velocidades de transmisión (BRG)	*Modo y reloj del timer contador*	*\INT generada por cambio en IP3*	*\INT generada por cambio en IP2*	*\INT generada por cambio en IP1*	*\INT generada por cambio en IP0*
Bit 7 0 = Conjunto 1 1 = Conjunto 0	Bit 6, Bit 5 y Bit 4 000=Contador Externo IP2 001=Contador TxCA 1x Reloj transmisor canal A 010=Contador TxCB 1x Reloj transmisor canal B 011=Contador (X1/CLK) dividido por 16 100=Timer Externo IP2 101=Timer Externo IP2 divido por 16 110=Timer (X1/CLK) 111=Timer (X1/CLK) dividido por 16	Bit 3 0 = No 1 = Si	Bit 2 0 = No 1 = Si	Bit 1 0 = No 1 = Si	Bit 0 0 = No 1 = Si

CTUR y CTLR Registros del Timer/contador.

Estos registros son de solo escritura y no se pueden leer. Carga los 8 bits más significativos y los 8 menos significativos respectivamente.

IPCR : Registro de cambios del puerto de entrada.

Cambio en IP3	*Cambio en IP2*	*Cambio en IP1*	*Cambio en IP0*	*Nivel en IP3*	*Nivel en IP2*	*Nivel en IP1*	*Nivel en IP0*
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 = No	0 = No	0 = No	0 = No	0 = Bajo	0 = Bajo	0 = Bajo	0 = Bajo
1 = Si	1 = Si	1 = Si	1 = Si	1 = Alto	1 = Alto	1 = Alto	1 = Alto

ISR : Registro del estado de las interrupciones. Este registro proporciona el estado de todas las fuentes de interrupción. Su contenido está enmascarado por el registro de máscara de interrupciones IMR. Si un bit ISR está a 1 y el bit correspondiente de IMR también está a 1, se activará la salida \INTR. Si el bit correspondiente de IMR está a cero el estado de ISR no afecta a la línea \INTR. El IMR no enmascara la lectura de ISR. El Contenido de este registro se inicializa a 0 cuando se resetea la DUART.

Cambio por el puerto de entrada	*Cambio del BREAK en el canal B*	*RxRDY FFULL*	*TxRDYB*	*Timer o contador listo*	*Cambio del BREAK en el canal A*	*RxRDYA FFULL*	*FxRDYA*
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 = No	0 = No	0 = No	0 = No	0 = No	0 = No	0 = No	0 = No
1 = Si	1 = Si	1 = Si	1 = Si	1 = Si	1 = Si	1 = Si	1 = Si

IMR : Registro de máscara de interrupciones. La programación de este registro, selecciona que bits de ISR provocan la petición de interrupción. Si un bit ISR está a 1 y el bit correspondiente de IMR también está a 1, se activará la salida \INTR. Si el bit correspondiente de IMR está a cero el estado de ISR no afecta a la línea \INTR. La línea de salida \INTR en función de los bits de los registros ISR e IMR cumple la siguiente ecuación.

INTR = IMR7*ISR7+IMR6*ISR6+…+ IMR0*ISR0

Cambio por el puerto de entrada INT	*Cambio del BREAK en el canal B INT*	*RxRDY FFULL INT*	*TxRDYB INT*	*Timer o contador listo INT*	*Cambio del BREAK en el canal A INT*	*RxRDYA FFULL INT*	*FxRDYA*
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 = No	0 = No	0 = No	0 = No	0 = No	0 = No	0 = No	0 = No
1 = Si	1 = Si	1 = Si	1 = Si	1 = Si	1 = Si	1 = Si	1 = Si

IVR : Registro de vector de interrupciones.

Este registro contiene el vector de interrupciones. Se inicializa al valor de $0F mediante la activación de la línea de RESET . El contenido de dicho registro, se coloca en el bus de datos cuando la DUART responde, en un ciclo de reconocimiento de interrupciones.

El número de vector $0F (15) corresponde al la excepción de vector de interrupción no inicializado, por este motivo se coloca este dato en el IVR al inicializar la DUART, para avisar de un fallo de programación en el caso de utilizar interrupciones.

El valor a almacenar en el registro IVR debe ser un dato comprendido entre 64 y el 255, ya que son los números de vector correspondientes a los vectores de interrupción de usuario que el microprocesador tiene reservados en su tabla de vectores de excepción.

5. Ejemplos.

5.1. Emisor.

1.- Mandar por el port serie (CANAL B) mediante interrupciones, una cadena de caracteres hasta encontrar un carácter nulo, a 9600 baudios, sin paridad, un bit de stop y 8 bits por carácter.

SOLUCION

```
RAM     EQU  $24000
PROG    EQU  $25000

DUART   EQU  $60041

MR1A    EQU  DUART+0
MR2A    EQU  DUART+0
SRA     EQU  DUART+2
CSRA    EQU  DUART+2
CRA     EQU  DUART+4
RHRA    EQU  DUART+6
THRA    EQU  DUART+6
IMR     EQU  DUART+10
IVR     EQU  DUART+24
IPR     EQU  DUART+26
OPCR    EQU  DUART+26
STACNT  EQU  DUART+28
OPRS    EQU  DUART+28
STOCNT  EQU  DUART+30
OPRR    EQU  DUART+30

VIA     EQU  $60001

ORB     EQU  VIA+0
ORA     EQU  VIA+2
DDRB    EQU  VIA+4
DDRA    EQU  VIA+6

        ABSOLUTE

        ORG  $20098 ; Direcci¢n del pseudovector correspondiente al
n£mero de
                    ; vector 64, asignaremos este vector al reg. IVR
de la DUART

        DC.L    INT_DUART

        ORG     RAM
FLAGT   DS.B    1

        ORG     PROG
        MOVE.B  #%00010000,CRA  ; RESET del MR1B. El puntero apunta a
MR1B.
        MOVE.B  #%00010011,MR1A ; Modo car cter. Sin paridad, 8
bits/car cter.
                                ; RxRDY =>IRQ
```

```
        MOVE.B  #%00000111,MR2A ; Un bit de stop.
        MOVE.B  #%10111011,CSRA ; 9600 baudios.

        MOVE.B  #64,IVR         ; N£mero de Vector

        MOVE.B  #$FF,DDRB
        MOVE.B  #$FF,DDRA

        MOVEA.L #$26000,A1      ; Inicializar direcc. contador

        CLR.W   (A1)

        MOVE.L  #$01,D2         ; Direccionar parte baja.

        MOVE.B  #%00010000,IMR  ; Habilitar interrup. Bit 4

        MOVE.B  #0,FLAGT

        MOVE.L  #MENSAJE,A4

        MOVE.B  #%00000100,CRA  ; Habilitar el transmisor

INCR:   ADDI.W  #1,(A1)
        MOVE.B  (A1),ORA
        MOVE.B  $0(A1,d2),ORB
        TST.B   FLAGT
        BEQ     INCR

        TRAP #5



INT_DUART       MOVEM.L D0-d2/A0,-(SP)
                MOVE.B  (a4)+,D0
                CMP     #0,D0
                BEQ INHAB
                MOVE.W #$300,d1
esperar         MOVE d0,d2
                MOVE d2,d0
                DBF D1,esperar
                MOVE.B d0,THRA  ; Mandar Car cter
                BRA QUEDAN

INHAB           MOVE.B #%00001000,CRA  ;Deshabilitar transmisor.
                MOVE.B #0,IMR           ;Deshabilitar interrupci¢n.
                MOVE.B #1,FLAGT        ;Para salir al monitor

QUEDAN          MOVEM.L (SP)+,D0-d2/A0
                RTE

MENSAJE         DC.B 'Prueba de transmisi¢n con interrupciones
$$$$$$$$$$$'

         DC.B '1Prueba de transmision con interrupciones $$$$$$$$$$$'
         DC.B '2Prueba de transmision con interrupciones $$$$$$$$$$$'
         DC.B '3Prueba de transmision con interrupciones $$$$$$$$$$$'
         DC.B '4Prueba de transmision con interrupciones $$$$$$$$$$$'
         DC.B '5Prueba de transmision con interrupciones $$$$$$$$$$$'
         DC.B '6Prueba de transmision con interrupciones $$$$$$$$$$$'
         DC.B '7Prueba de transmision con interrupciones $$$$$$$$$$$'
```

```
        DC.B '8Prueba de transmision con interrupciones
$$$$$$$$$$$$',0

        END
```

5.2. Receptor.

2.- Realizar un programa que reciba por el port serie (canal A) de la DUART y los almacene en memoria hasta encontrar un carácter nulo, a 9600 baudios, sin paridad, un bit de stop y 8 bits por carácter.

SOLUCION

```
        RAM     EQU  $24000
PROG    EQU  $25000

DUART   EQU  $60041

MR1A    EQU  DUART+0
MR2A    EQU  DUART+0
SRA     EQU  DUART+2
CSRA    EQU  DUART+2
CRA     EQU  DUART+4
RHRA    EQU  DUART+6
THRA    EQU  DUART+6
IPCR    EQU  DUART+8
ACR_D   EQU  DUART+8
ISR     EQU  DUART+10
IMR     EQU  DUART+10
CTU     EQU  DUART+12
CTUR    EQU  DUART+12
CTL     EQU  DUART+14
CTLR    EQU  DUART+14
MR1B    EQU  DUART+16
MR2B    EQU  DUART+16
SRB     EQU  DUART+18
CSRB    EQU  DUART+18
CRBA    EQU  DUART+20
RHRB    EQU  DUART+22
THRB    EQU  DUART+22
IVR     EQU  DUART+24
IPR     EQU  DUART+26
OPCR    EQU  DUART+26
STACNT  EQU  DUART+28
OPRS    EQU  DUART+28
STOCNT  EQU  DUART+30
OPRR    EQU  DUART+30

        ABSOLUTE

        ORG  $20098 ; Direcci¢n del pseudovector correspondiente al
n£mero de
                    ; vector 64, asignaremos este vector al reg. IVR
de la DUART

        DC.L    I_DUART

        ORG     RAM

PUNT_REC        DS.L 1 ;zona de memoria reservada para almacenar la
direccion
```

```
                        ;de memoria donde se guardar n los datos es
decir el
                        ;puntero de recepci¢n

MENSAJE         DS.B 1000 ;memoria reservada para almacenar ls datos
recibidos
        ORG     PROG
        ;inicializaci¢n del puntero

        MOVE.L  #MENSAJE,PUNT_REC

        MOVE.B  #%00010000,CRA  ; RESET del MR1B. El puntero apunta a
MR1B.
        MOVE.B  #%10000011,MR1A ; Control de RTS sin paridad, 8 bits
de datos

        MOVE.B  #%00000111,MR2A ; Un bit de stop.
        MOVE.B  #%10111011,CSRA ; 9600 baudios.
        MOVE.B  #%00000010,IMR  ;Habilitamos interrupciones de RxRDY
(receptor listo)

        MOVE.B  #64,IVR         ; N£mero de Vector
        MOVE.B  #%00000001,CRA  ; Habilitamos el receptor del Canal A

        MOVE.B  #%00000001,OPRS ; Se activa la linea RTS a 0 para
permitir al transmisor mandar datos

ESPERA  BRA     ESPERA          ; Aqu¡ ir¡a el resto del programa
        TRAP    #5

I_DUART       MOVEM.L D0/A0,-(SP)
              MOVE.L  PUNT_REC,A0       ; Leer el puntero de recepci¢n


              MOVE.B RHRA,D0

              CMPI.B #0,D0      ; ¨Dato es cero?
              BEQ    FIN_REC

              MOVE.B D0,(A0)+
              MOVE.L A0,PUNT_REC

                BRA     FIN_INT

FIN_REC         MOVE.B  #%00000010,CRA

FIN_INT         MOVEM.L (SP)+,D0/A0
                RTE
                END
```

CAPITULO 8

Evolución del 68000

El 68000, tal y como se ha descrito detalladamente en este libro, no es sino la versión básica de una familia que ha evolucionado desde 1980 hasta el momento. La evolución de las tecnologías de fabricación de circuitos integrados ha permitido incorporar conceptos *informáticos* propuestos con anterioridad pero cuya realización práctica, y sobre todo en un solo chip no era económicamente factible hasta el momento. Describiremos muy brevemente cada uno de sus miembros.

El 68008.

Es una versión *reducida* del 68000 del que únicamente difiere en la anchura del bus de datos, que es en este caso de 8 bits. Internamente es idéntico al 68000 y por tanto, es capaz de ejecutar el mismo código.

Su interés está basado en la construcción de sistemas más pequeños en los que realmente no se requiere un bus de 16 bits, simplificándose el diseño de la memoria especialmente. Su campo de aplicación puede estar en coincidencia con el de los microprocesadores de 8 bits.

Como contrapartida hay que señalar que todos aquellos accesos al bus para efectuar una lectura o una escritura de una palabra de 16 bits deben realizarse en dos ciclos de bus, accediendo en cada uno de ellos a uno de los bytes que la componen. Esto se traduce en una cierta pérdida de velocidad que será menor cuantos menos accesos al bus haga el programa que se ejecuta.

El 68010.

Presentado en 1982 es un 68000 mejorado en los siguientes aspectos:

- Por un lado, un diseño interno más cuidadoso y elaborado ha permitido disminuir el tiempo de ejecución de algunas instrucciones, mejorando las prestaciones de velocidad del 68000.
- Se le añaden posibilidades para el manejo de memoria virtual.
- Permite manejar varias tablas de vectores de interrupción. Para ello dispone de un registro llamado **VBR** (Vector Base Register) en el que se escribe la dirección de origen de la tabla de vectores. Si el contenido de este registro fuese 000000 la tabla estaría en el mismo lugar que en el 68000 original. El sentido que tiene este registro hay que buscarlo en sistemas **multitarea,** en donde cada tarea puede tener su propia tabla de vectores de excepción.
- Soporte para MAQUINA VIRTUAL. Se trata de una mejora pensada para que la CPU pueda manejar *concurrentemente* dos sistemas operativos, funcionando ambos

aparentemente en el **modo supervisor**, aunque en realidad están en el modo usuario pero pueden ejecutar algunas instrucciones nuevas que afectan a los *códigos de condición.*

El 68020.

Data de 1983 y constituye el primer microprocesador de 32 bits de Motorola. En efecto, sus buses de datos y direcciones son ambos de 32 bits. Sus características más importantes se resumen seguidamente:

» Es compatible a nivel de código máquina con los anteriores. Por tanto puede ejecutar programas creados para el 68000.
» Su arquitectura mejora muy sensiblemente al incorporar mecanismos de *paralelismo* que le permiten que sus circuitos internos efectúen simultáneamente varias operaciones. En consecuencia su velocidad de ejecución de programa se incrementa notablemente.
» Aunque su bus de datos es de 32 bits maneja señales *hardware* que le permiten reconfigurar su anchura a 8, 16, 24 ó 32 bits según el dispositivo al que se acceda.
» Con 32 bits de direcciones es capaz de manejar un espacio de memoria de 4 Gbytes.
» Soporte para maquina virtual y para memoria virtual.
» Señales de interfase para un coprocesador matemático (68881).
» Incorpora *cache* de instrucciones 64 x 32 bits.
» Añade un puntero de pila adicional: el *Interrupt Stack Pointer* que queda independizado del *Supervisor SP* ahora llamado *Master SP*.
» Incorpora nuevas instrucciones, nuevos tipos de datos y nuevos modos de direccionamiento.

El 68030.

Se trata de una CPU que introduce mejoras adicionales sobre las del 68020 y que Motorola define como la **Segunda Generación**. Aparte de los perfeccionamientos propios de su posterior diseño, sus características diferenciales más notables son:

» *Cache* de instrucciones de 256 bytes.
» *Cache* de datos de 256 bytes.
» Señales de control para memoria *cache* externa.

» Integración de un MMU (Memory Management Unit) equivalente al *chip* externo anterior 68851. Se trata de un mecanismo *hardware* de traslación de direcciones basado en *páginas*. El tamaño de las páginas puede ser desde 256 bytes hasta 32 kBytes.

El 68040.

Constituye lo que Motorola denomina **Tercera Generación** de su familia 68000. Su diseño ha sido mejorado, incluyendo conceptos de *Arquitectura Harvard, pipelinning,* paralelismo, múltiples buses internos ... Alcanza una velocidad de proceso de 3.5 MFLOT y 20 MIP. Sus características más sobresalientes, por encima de las del 68030, son:

» Integración en la misma pastilla del coprocesador matemático.
» Dos unidades de manejo de memoria (MMU) independientes: una para instrucciones y otra para datos.
» Memorias *cache* de instrucciones y datos independientes, de 4 kB cada una.
» Soporte para multiproceso con garantías para el contenido de las *caches (Snooping).*

No son éstos los únicos miembros de la familia. Para completar la presentación habría que incluir las versiones CMOS, de bajo consumo, y los microcontroladores especializados que de esta familia se han derivado.