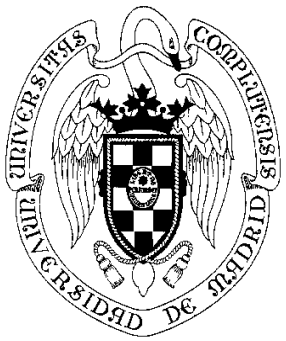
APUNTES
DE
ESTRUCTURA DE COMPUTADORES



JUAN LANCHARES DÁVILA

DEPARTAMENTO
DE
ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

1 ESTRUCTURA DE COMPUTADORES

1.1 FAMILIA DE COMPUTADORES

- Se dice que un conjunto de sistemas computadores forma una familia cuando todas ellos tienen la misma arquitectura y diferentes estructuras
- El concepto diferenciado entre Arquitectura y Estructura de un computador surge en los años 60 con el lanzamiento al mercado de la familia IBM 360.
- Cada estructura tiene una relación rendimiento/coste diferente. Generalmente las gamas altas de la familia tienen mayor rendimiento y coste.
- Con la familia de computadores surge el concepto de **compatibilidad**.
    - Un programa escrito para un modelo se puede ejecutar en otro modelo de la serie con la única diferencia del tiempo de ejecución.
    - La compatibilidad entre diferentes miembros de la familia es ascendente. Es decir funciona con programas escritos en una gama inferior que se ejecutan en una superior. Lo contrario no siempre es cierto.
- Las **características** de una familia son:
    - Repertorio de I's similar o idéntico
    - Velocidad en incremento
    - Nº de puertos I/O en incremento
    - Tamaño de la memoria creciente
    - Coste creciente

1.2 ARQUITECTURA DE UN COMPUTADOR

De las subáreas en que se descompone el estudio de un computador hay dos íntimamente relacionadas con la asignatura: la Arquitectura de Computadores y La Estructura de Computadores. Estas subáreas tienen fronteras comunes lo que puede dificultar su diferenciación.

El conocimiento de la arquitectura es imprescindible para estudiar la estructura que la implementa. De las muchas definiciones de Arquitectura que existen se damos dos que son, en cierta medida, el compendio de todas las demás. Bell y Newell, en su libro *Computer Structures. Principles and examples* [Siew82] definen arquitectura como los atributos del computador que puede ver el programador de lenguaje máquina.

Por su parte, Hennessy y Patterson realizan una interesante definición. Arquitectura es la interface entre el software de bajo nivel y el Hardware, indicando que esta interfaz es la que posibilita implementaciones de diverso coste/rendimiento en las que corre software idéntico. La arquitectura debe contener todo lo que los programadores necesitan conocer para que el programa SW (Lenguaje Máquina) funcione correctamente. Según indican Hennessy y Patterson todos los lenguajes máquina son parecidos. Esto se debe a que la mayoría de los computadores se construyen con tecnologías basadas en los mismos principios básicos, y a que deben ser pocas las operaciones que debe suministrar el computador.

El objetivo perseguido al diseñar una arquitectura es encontrar un lenguaje máquina que haga fácil la construcción del hw y del compilador, al tiempo que se maximiza el rendimiento y se minimiza el coste [Henn93][Patt95]. Ejemplos de atributos de arquitectura son:

- Repertorio de instrucciones
- Formato de las instrucciones
- Códigos de operación
- Modos de direccionamiento
- Registros y posiciones de memoria que se pueden manipular directamente
- Número de bits utilizados para representar diferentes tipos de datos
- Mecanismos de entrada/salida

1.3 LA ESTRUCTURA DE UN COMPUTADOR

La estructura de un computador estudia las unidades operacionales de un computador así como y las formas de relacionarlas para que implementen las especificaciones de la arquitectura. Atributos de la estructura son los detalles del hardware transparentes al programador como

- Las señales de control
- Interfaces entre el computador y los periféricos
- La tecnología de memoria utilizada
- El tipo de operadores aritméticos seleccionado

Con el ejemplo que se da a continuación se ven las diferencias entre los dos conceptos. Una decisión que afecta a la arquitectura es determinar si el computador va a disponer de una determinada operación aritmética, por ejemplo, el producto. Una decisión de estructura es estudiar cómo implementar dicha operación, si mediante un sistema secuencial o combinacional; mediante una unidad especial o en la UAL del computador. La decisión de diseño de la estructura se fundamenta en:

- La velocidad de ejecución
- En el tamaño
- Consumo de potencia
- Etc.

La diferencia entre arquitectura y estructura aparece de manera clara en las familias de computadores, que comparten una misma arquitectura pero tienen diferentes estructuras. Consecuentemente, los diferentes modelos de la familia tienen diferentes precios y características de rendimiento. Aún más, el concepto de estructura está ligado a la tecnología de fabricación, mientras que la arquitectura es independiente de él, de tal manera que una arquitectura puede perpetuarse durante años, mientras es muy extraño que una estructura dure demasiado tiempo. El primer ejemplo de familia de computadores fue el IBM 360.

En lo que respecta al rendimiento del computador, en un principio eran las mejoras tecnológicas las que aumentaban el rendimiento de los computadores. En la actualidad, es la mejora en las arquitecturas la que ha logrado un gran avance en el rendimiento [Henn93]. En esta dirección ha habido adelantos tan importantes como:

- La segmentación - pipeline
- El paralelismo
- Los computadores RISC

Conviene recordar que pueden existir muchas estructuras diferentes para implementar un conjunto de instrucciones pero, en definitiva, es el repertorio de instrucciones el factor principal que determina la razón coste/rendimiento [Huck 89].

Antes de pasar al siguiente apartado quiero recalcar que la frontera entre arquitectura y estructura no está tan clara como puede parecer. De hecho, existen autores como Baron y Higbie [Baro92] que consideran, que tanto lo que hemos definido como arquitectura, como lo que hemos definido como estructura, son arquitectura de computadores:

> *Arquitectura de computadores es el diseño de computadores incluyendo su conjunto de instrucciones, sus componentes hardware y su organización de sistema. Existen dos partes esenciales en la arquitectura de computadores: la arquitectura del conjunto de instrucciones (ACI) y la arquitectura del sistema hardware (ASH). La ACI incluye las especificaciones que determinan cómo el programador de lenguaje máquina interactúa con el computador. Un computador se ve generalmente en términos de su ACI que determina sus características computacionales. En contraste, el ASH está relacionado con los principales subsistemas hardware del computador, incluyendo su unidad central de proceso, su sistema de almacenamiento, y su sistema de entrada/salida. El ASH incluye tanto diseño lógico como organización del flujo de datos de dichos subsistemas, por eso el ASH determina en gran medida la eficiencia de la máquina.*

1.4 CLASIFICACIÓN DE LOS COMPUTADORES

Las clasificaciones se han basado, generalmente, en la potencia de procesamiento. Debido a los importantes avances tecnológicos acaecidos desde 1970, ninguna de las clasificaciones ha sido definitiva. Una clasificación tradicional es la siguiente:

- Microcomputador (PCs y estaciones de trabajo)
- Minicomputador
- Computador (Mainframe)
- Supercomputador

En la actualidad casi todos los sistemas utilizan el microprocesador como bloque de construcción. Además las rápidas mejoras en el rendimiento de las CPUs acercan las tradicionales familias. Los elementos diferenciadores son :

- Potencia de Entrada/Salida
- Sistema de memoria

Hennessy y Patterson basan la clasificación exclusivamente en el precio

ν **Microcomputadores**

- Pequeños computadores personales
- Estaciones de trabajo
- Unos miles de dólares

ν **Minicomputadores**

- Mas de 50.000 dólares
- Tamaño medio

ν **Mainframe**:

- Mas de medio millón de dólares
- Propósito general de altas prestaciones
- Se utiliza para tareas de gestión
- Comercial
- Aritmética decimal→ que no necesita la codificación y descodificaron para trabajar con ella. Estos sistemas no están pensados para realizar grandes cálculos aritméticos

- Soporte para grandes bases de datos
- Tratamiento de transacciones
- Soporta más terminales y discos que el minicomputador

ν **Supercomputador**:

- Mas de un millón de dólares
- Aritmética de punto flotante
- Mas caros
- Aplicaciones científicas
- Mas alto rendimiento

1.4.1 CLASIFICACIÓN DE FLYNN

(pg 605 Stalling)

Realizó la clasificación en 1972 en función de las siguientes características:

- número de procesadores
- nº de programas
- Estructura de memoria

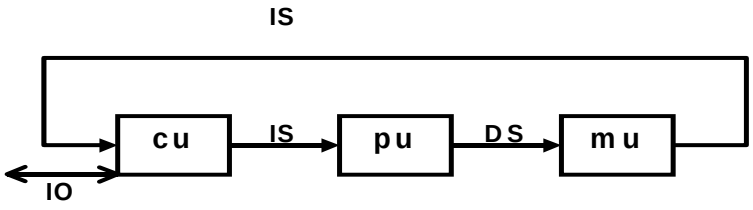
Flynn clasificó los computadores en:

SISD
SIMD
MISD
MIMD

ν **SISD**

Un único procesador interpreta una única secuencia de instrucciones para operar con los datos de una única memoria

- Solo una instrucción solo un dato
- Es típico el Von Neumann
- Una CPU que ejecuta una instrucción cada vez y busca o almacena un dato cada vez
- Es una arquitectura con un único procesador



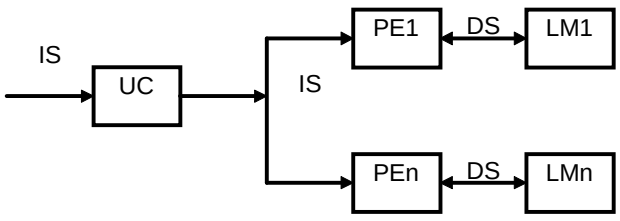
Siendo:

- **CU** la unidad de control
- **PU** la unidad de procesamiento
- **MU** unidad de memoria
- **IS** cadena de instrucciones
- **PE** elemento de proceso
- **LM** memoria local

ν **SIMD**

Una única instrucción controla la ejecución simultánea de varias unidades de proceso. Cada unidad de proceso tiene una memoria asociada. Cada instrucción es ejecutada en cada procesador por un conjunto de datos diferente. Sistemas típicos son los procesadores vectoriales y matriciales

- Solo una instrucción múltiples datos
- Una unidad de control
- Varias unidades de proceso
- Típico de arrays de procesadores
- Se ejecuta la misma instrucción sobre datos diferentes
- Distribuyen el proceso sobre una gran cantidad de hw
- Operan concurrentemente sobre muchos datos
- Ejecutan el mismo cálculo sobre todos los elementos
- La unidad de control es por si misma un computador de Von Neumann y se le llama UC porque esta diseñada para controlar los procesadores
- Además puede incluir computadores host que realizan tareas especificas como:
  - Carga de programas
  - Configuración del array de elementos de proceso
  - Supervisión de entrada/salida



Arquitectura SIMD con memoria distribuida

ν **MISD**

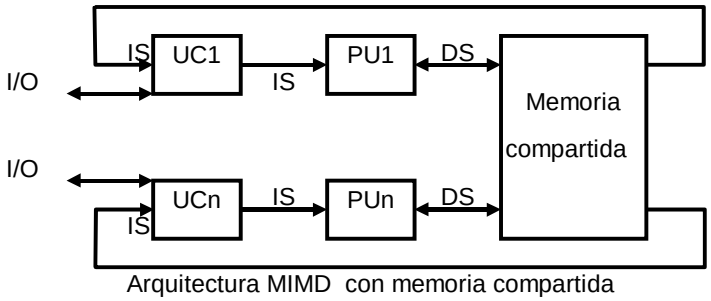
Se transmite una secuencia de datos a un conjunto de procesadores. Cada procesador ejecuta una instrucción diferente sobre el mismo conjunto de datos. Nunca se ha implementado.

- Múltiples instrucciones y solo un dato
- Ejecución de diferentes programas sobre el mismo datos
- Conocidos como arrays sistólicos

ν **MIMD**

Conjunto de procesadores que ejecutan conjuntos de instrucciones diferentes sobre conjuntos de datos diferentes. Los procesadores son de carácter general.

- Múltiples instrucciones múltiples datos
- Distribuyen el procesamiento entre un nº de procesadores independientes
- Distribución de recursos, incluyendo la MP, entre los procesadores
- Cada procesador opera concurrentemente y en paralelo con el resto
- Cada procesador ejecuta su propio programa
- Diferentes arquitecturas se diferencian por
  - Diferente red de interconexión
  - Diferentes procesadores
  - Diferente estructura de direccionamiento de memoria
  - Diferente control y estructura de sincronización
- Multiprocesadores→ tienen memoria común
- Multicomputadores→ tienen memorias locales.



Arquitectura MIMD con memoria compartida

1.5 HISTORIA DE LOS COMPUTADORES

1.5.1 ANTECEDENTES

Desde tiempos inmemoriales el hombre a sentido la necesidad de utilizar herramientas que le hicieran el trabajo más sencillo. De hecho se puede afirmar que este es el motor de la civilización. Que duda cabe que unos de los trabajos que más tediosos se puede hacer es el de los cálculos numéricos, y esta posiblemente fue la causa de que se inventara el ábaco en china en el siglo XI. Este instrumento, compuesto por un conjunto de cuentas insertadas en varillas paralelas, permite realizar cálculos aritméticos asignando diferentes valores a las cuentas y deslizándolas para que ocupen determinadas posiciones en la varilla.

Han existido con posterioridad otros intentos de diseñar máquinas de calcular, como la del escocés Napier a finales del siglo XVI que multiplicaba y dividía, o el aritmomero de Pascal diseñado a mediados del siglo XVII, que sumaba y restaba utilizando ruedas dentadas y engranajes. Basándose en la idea de Pascal, Leibnitz construyó una máquina que sumaba, restaba, multiplicaba y dividía.

Ya en el siglo XIX Jacquard ideo un sistema a partir de tarjetas perforadas que unido a un telar permitía tejer piezas complicadas, de manera que al variar la tarjeta variaba el dibujo a tejer. En 1822 Babbage diseño la máquina de las diferencias. Esta máquina calculaba tablas de funciones empleando el método de las diferencias divididas. La máquina constaba de ruedas dentadas y engranajes movidos por máquinas de vapor que permitía una precisión de hasta 20 decimales. En 1834 Babbage diseñó una máquina computadora universal y programable llamada la máquina analítica. Precisamente, es a finales del siglo XIX cuando se crea la compañía Hollerith Tabulating Machine Company embrión de la futura International Business Machines, que es considerada como la primera empresa de computadores. Esta empresa comercializaba una máquina de tarjetas basada en la máquina de Jaquard que se utilizaba para reducir el tiempo de las operaciones de clasificación y recuento del censo.

A principios del siglo veinte existían maquinas de calcular mecánicas que utilizaban ruedas movidas por motores eléctricos. También existían computadores electromecánicos que utilizaban relés para representar números.

Fue durante la Segunda Guerra Mundial cuando se produjo un gran auge de los computadores. Esto se debió a que facilitaban enormemente los cálculos balísticos. Un ejemplo de este tipo de computadores son el Z3 y Z4 desarrollados por el alemán Konrad en 1941 que se utilizaron en el diseño de las bombas volantes V2. En el

1944 Aiken de la universidad de Harvard construye el Mark I, computador electromecánico que incluía el concepto de programa almacenado.En estos años coexistían los computadores mecánicos y electromecánicos y aparecen los computadores electrónicos. Estos primeros computadores electrónicos son la base de los actuales computadores. En el siguiente punto vamos a estudiar el desarrollo de los computadores en función de los avances que se han ido produciendo en la tecnología electrónica.

Etapas tecnológicas

Es habitual clasificar los computadores en generaciones basadas en la tecnología con que se implementan. Cada nueva generación se caracteriza por una mayor velocidad, mayor capacidad de memoria y menor tamaño. Estas variaciones tienen como efecto importantes modificaciones en las Arquitecturas y Estructuras de los computadores. Permiten la utilización de estructuras o módulos funcionales que antes, ya fuera por su tamaño, velocidad o complejidad de fabricación, se desechaban. Las fechas que limitan cada generación son fechas aproximadas que marcan eventos importantes pero que no indican una clara diferencia entre la utilización de las diversas tecnologías. En la figura que viene a continuación se asocia a cada generación la tecnología que la implementa y los computadores comerciales más representativos:

PRIMERA GENERACIÓN

- VÁLVULAS DE VACÍO
- ARQUITECTURA DE VON NEUMANN
- CONTADOR DE PROGRAMA
- ACUMULADOR
- ARITMÉTICA DE PUNTO FIJO
- UNIVAC
- IBM 700

SEGUNDA GENERACIÓN

- TRANSISTORES DISCRETOS
- ARITMÉTICA DE PUNTO FLOTANTE
- CANAL DE DATOS
- PDP 1
- IBM 7094

TERCERA GENERACIÓN

- CIRCUITOS INTEGRADOS (SSI/MSI)
- DIFERENCIA ENTRE ARQUITECTURA Y ESTRUCTURA
- MICROPROGRAMACION
- SEGMENTACION
- CACHE
- FAMILIA IBM 360
- PDP 8

CUARTA GENERACIÓN

- MEMORIAS SEMICONDUCTORAS
- MICROPROCESADORES
- SUPERCOMPUTADORES VECTORIALES
- INTEL 4004
- INTEL 8086
- MC680X0
- IBM 3090
- CRAY XMP
- VAX 9000

QUINTA GENERACIÓN

- ULSI/VHSIC
- ARQUITECTURAS ESCALABLES
- CRAY MPP

Primera generación. Las válvulas de vacío (1938-1954)

La paternidad del primer computador electrónico no está demasiado clara. Hasta hace bien poco se consideraba como tal el diseñado por Turing en el Bletchey Research Establishment del Reino Unido. Este computador llamado COLOSSUS entró en servicio en 1943 y sirvió para descifrar el código *Enigma* de los alemanes, lo que ayudó de manera significativa a ganar la guerra. Pero al parecer Atanasoff diseñó dos computadores electrónicos con anterioridad, uno en 1937 y el otro en 1942. Estos computadores se utilizaron para solucionar sistemas de ecuaciones lineales. El ABC (Atanasoff-Berry Computer) , que así se llamaba, es considerado hoy como **el primer computador electrónico de propósito específico**.

El **primer computador electrónico de propósito general** fue el ENIAC (Electronic Numerical Integrator And Computer), diseñado y construido en 1943 por Mauchly y Eckert en la Universidad de Pennsylvania. El proyecto se inició durante la Segunda Guerra Mundial y su principal objetivo era el cálculo de tablas de fuego.

Las características de este computador eran: 30 toneladas de peso, 10000 pies cuadrados, 18000 válvulas de vacío y un consumo de potencia de 140 Kw. El ENIAC era más rápido que cualquiera de los computadores electromecánicos existentes, con más de 5000 sumas por segundo. La representación de los números era decimal. Su memoria consistía en 20 acumuladores, cada uno de los cuales era capaz de almacenar 10 dígitos decimales. Cada dígito se representaba mediante 10 tubos de vacío. Esta máquina era programable y aceptaba los saltos condicionales.

El principal inconveniente del ENIAC era que tenía que programarse manualmente manipulando los interruptores, y conectando y desconectando cables. Se acabó de construir en 1946, demasiado tarde para utilizarlo en el esfuerzo de la guerra. En lugar de eso, sirvió para realizar una serie de cálculos complejos que ayudaron a determinar la fiabilidad de la bomba atómica. Estuvo activo hasta 1955 en que fue desmantelado.

La máquina de Von Neumann

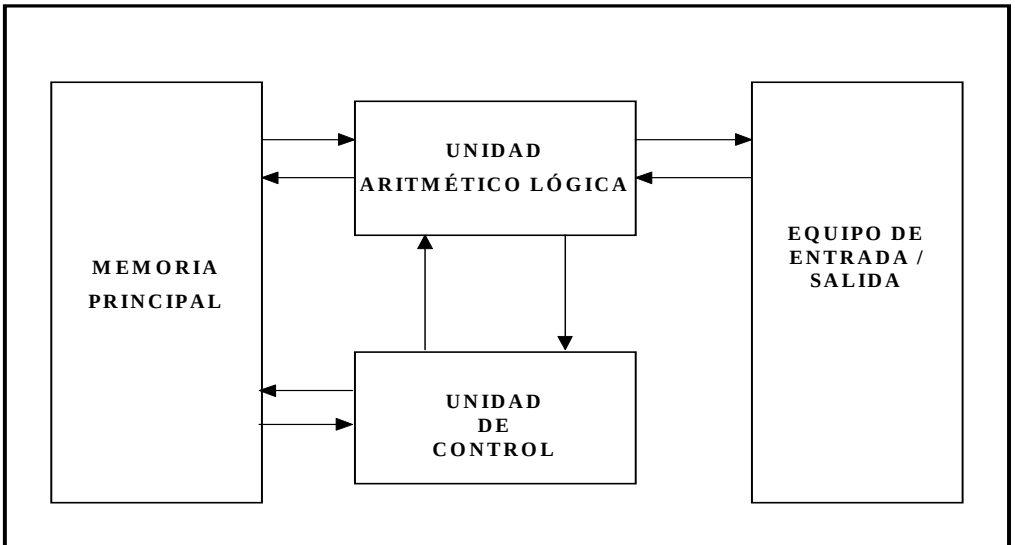
Los diseñadores del ENIAC se dieron cuenta de que la programación manual del computador era demasiado rígida y limitaba la potencia del mismo. Un asesor del proyecto propuso que tanto los programas como los datos se almacenaran en la memoria. Este asesor se llamaba John Von Neumann. Este concepto revolucionó el mundo de los computadores hasta tal punto que incluso hoy en día se utilizan estructuras basadas en esta arquitectura. La idea fue utilizada por primera vez en la descripción de una nueva máquina, la EDVAC (Electronic Discrete Variable Computer).

En 1946 Wilkes tras asistir a una serie de conferencias en la Universidad de Pennsylvania comenzó en la Universidad de Cambridge la construcción de un

computador de programa almacenado en memoria, la EDSAC (Electronic Delay Storage Automatic Calculator), que fue el primer computador de programa almacenado operativo a escala completa. El mismo Wilkes propuso también la utilización de unidades de control microprogramadas, pero debido a razones tecnológicas este concepto no se implemento hasta una década más tarde [Wilk51][Wilk53].

También en 1946, Goldstine, que había trabajado con von Neumann y Burks diseñó en el Princeton Institute for Advanced Studies un computador, el IAS, acabado en 1952 y que fue el prototipo de todos los modelos posteriores de propósito general. Sus características eran:

- Una memoria principal que almacenaba tanto datos como instrucciones
- Una unidad aritmético lógica que operaba con datos binarios
- Una unidad de control que interpretaba las instrucciones y generaba las señales para su ejecución.
- Un equipo de entrada y salida controlado por la unidad de control.



ESTRUCTURA DEL COMPUTADOR IAS

La memoria del IAS tenía 1000 posiciones de almacenamiento, llamadas palabras, de cuarenta bits cada una de ellas. Los números y las instrucciones se codificaban en binario. La unidad de control funcionaba buscando las instrucciones en la memoria y ejecutando una cada vez.

Primeros computadores comerciales

En 1951 se construye el primer computador comercial, el LEO (Lyon Electric Office) construido por Wilkes. A principio de los cincuenta había dos compañías comerciales que dominaban el mercado de los computadores, la Sperry y la IBM. En 1947, Mauchly y Eckert, los diseñadores del ENIAC, crean su propia compañía, la Eckert-Mauchly Computer Corporation. La empresa fue absorbida por la Sperry, que se dedicó a fabricar una serie de máquinas de gran éxito: la UNIVAC I, que fue el primer computador fabricado en serie, y UNIVAC II.

La UNIVAC I fue la primera máquina comercial con éxito. Se pensó para cálculos científicos y aplicaciones comerciales. La UNIVAC II apareció a finales de los 50 y tenía mayor memoria y rendimiento que la UNIVAC I. Simultáneamente a la UNIVAC II, comienza el desarrollo de la serie 1100, cuyo primer computador fue el UNIVAC 1103. La característica principal de esta serie era que estaba orientada a cálculo científico, es decir, cálculos largos y complejos.

Por otra parte, IBM, que era el principal fabricante de equipo procesador de tarjetas perforadas, desarrolla en 1953 su primer computador electrónico de programa almacenado en memoria, el 701. Este computador estaba orientado a los cálculos científicos. En 1955 IBM introdujo el 702. Este computador tenía características hardware orientadas a los negocios, es decir, al procesamiento de grandes cantidades de datos. Estos fueron los primeros computadores de la serie 700/7000 que puso a IBM en cabeza de los fabricantes de computadores. Del IBM 709 se vendieron 19 unidades. El último computador de la serie 700, el 794 se caracterizaba por incorporar operaciones en coma flotante.

En esta época las memorias que se utilizaban eran de ferrita, se introducen las cintas magnéticas como memorias de almacenamiento masivo y Wilkes propone en 1951 la idea de microprogramación, aunque esta técnica no se pudo incorporar hasta casi 10 años después al no existir memorias lo suficientemente rápidas.

En resumen, la primera generación utilizó válvulas de vacío y relés. La velocidad de proceso se mide en milésimas de segundo. Estos computadores disipaban gran cantidad de energía calorífica y necesitaban controles rigurosísimos de refrigeración y limpieza ambiental. Eran equipos de gran tamaño, escasa capacidad y difícil mantenimiento. Los trabajos se realizaban en monoprogramación y no existía sistema operativo. Se programaba en lenguaje máquina y los periféricos de entrada salida dependían directamente del procesador. En el año 1953 Nathan Rochester diseñó el lenguaje ensamblador formado por instrucciones simbólicas que se correspondía con instrucciones máquina, facilitando la programación de los computadores.

La segunda generación. El transistor (1954-1963)

El cambio más importante que se ha producido en el mundo de la electrónica ha sido la sustitución de los tubos de vacío por el transistor, inventado en los laboratorios Bell de ATT por Bardeen, Brattain y Shockley. El transistor era más pequeño, más barato y disipaba menos potencia que los tubos de vacío, pero se podía usar exactamente igual que éstos para fabricar computadores. En 1954 se construyó en los laboratorios Bell el primer computador digital con transistores, el TRADIC.

La segunda generación trajo también unidades de control y unidades aritmético lógicas más complejas, y los primeros lenguajes de alto nivel que permitían una codificación más cómoda que la realizada con lenguaje máquina. Destaca el Fortran que hacía la preparación de programas de aplicación mucho más sencillo. También se desarrollaron programas de sistema, llamados compiladores, que traducían los lenguajes de alto nivel a lenguajes ensambladores que, a su vez, había que traducir a un lenguaje máquina ejecutable por el computador.

Desde el punto de vista de las empresas, es notable la aparición de la Digital Equipment Corporation (DEC). Esta compañía se fundó en 1957 y aquel mismo año sacó al mercado su primer computador, el PDP 1. Tenía 4Kx8 bits de memoria, un terminal de vídeo de 512X512 pixels con alguna capacidad gráfica y un tiempo de ciclo de 5µs.

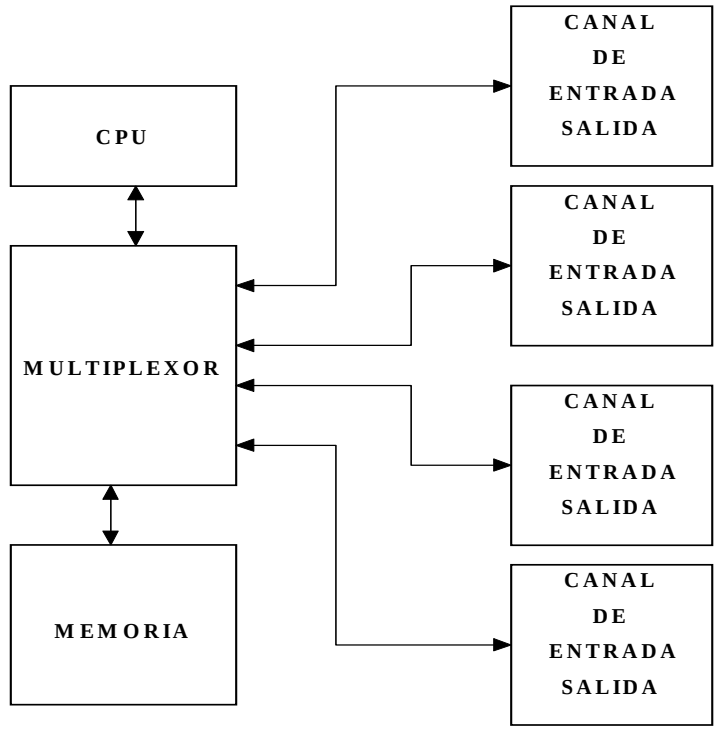
En 1962 en la universidad de Manchester se desarrolla el ATLAS que es el primer computador que utiliza un disco magnético como memoria secundaria, creando el concepto de memoria virtual. También es el primer computador que utiliza las interrupciones de entrada/salida.

El IBM 7094

Desde que se creó el primer computador de la serie 700 hasta el último computador de la serie 7000, la principal característica de los productos fue el aumento de rendimiento y de la memoria.

Lo más notable del IBM 7094, es la aparición de un módulo diferente a los explicitados en el modelo de Von Neumann, el ***canal de datos***. Un canal de datos es un módulo independiente de entrada/salida con su propio procesador y su propio conjunto de instrucciones. Los computadores con este módulo no ejecutan ellos mismos las instrucciones de entrada/salida. Éstas se almacenan en la memoria principal pero se ejecutan en el canal. Este computador tenía 32Kx36 bits de memoria y un tiempo de ciclo de 2µs .

Otra característica de esta generación es la aparición de un multiplexor que distribuía los datos que entraban hacia la memoria o hacia la CPU. A esta estructura se la conoce como centralizada, por que cualquier intercambio de datos se realiza a través del multiplexor.



CONFIGURACIÓN IBM 7094

Tercera generación. Circuitos Integrados a pequeña y mediana escala (1963-1971)

Hasta el momento los transistores, resistores y capacitores se fabricaban y empaquetaban por separado. Posteriormente se soldaban o unían a las tarjetas para implementar una función. Esto hacía que el diseño de computadores fuera complejo y caro.

En 1958, la invención del circuito integrado revolucionó de nuevo el mundo de los computadores. La importancia del circuito integrado reside en la capacidad de fabricar en una misma superficie de silicio un conjunto de transistores que implementan una funcionalidad. Esto permite que cientos de transistores puedan fabricarse simultáneamente sobre una oblea de silicio y, además, se puedan conectar por medio de un proceso de metalización. Para el fabricante de computadores la

posibilidad de una mayor densidad de los circuitos proporciona las siguientes ventajas:

- Disminuye el precio de los computadores.
- Como los elementos lógicos y de memoria se pueden situar más próximos se incrementa la velocidad.
- Los computadores son más pequeños.
- Disminución del consumo de potencia y de la temperatura.
- Conexiones más fiables que las soldaduras.

Sistema 360 de IBM

En 1964 IBM lanza al mercado una familia de computadores, ***el sistema 360***, que no era compatible con los computadores anteriores, pero que introducía un concepto revolucionario en el mundo de la informática. En definitiva, una familia de computadores no era más que varios modelos compatibles entre sí, de manera que un mismo programa se podía ejecutar en cualquiera de ellos con la única diferencia de los tiempos de cálculo. Las características de una familia son:

- Conjunto de instrucciones similar
- Sistema operativo similar
- Incremento de la velocidad en los modelos
- Incremento del número de puertos de entrada/salida
- Incremento del tamaño de la memoria
- Incremento del precio

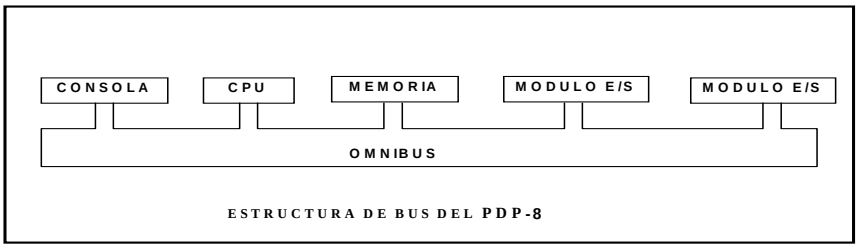
Lo revolucionario del sistema es la aparición, claramente diferenciada, de dos conceptos: la Arquitectura y la Estructura de un computador. Esta familia incluía multiprogramación, tenía 16 registros de 32 bits, y un espacio de direcciones de 16 Mbytes. Incluía además la posibilidad de programar las prioridades de las interrupciones y ciertos mecanismos de protección de la memoria. Estos fueron los primeros computadores que se comercializaron con tecnología SSI y difundieron importantes innovaciones como prioridades de las interrupciones, controladores DMA, memoria cache, protección de memoria y microprogramación.

En 1964 Control Data lanza el 6600 diseñado por Cray, que incluía segmentación de las unidades funcionales, con lo que obtenía un rendimiento un orden de magnitud superior al IBM 7094. Este computador es considerado el **primer supercomputador**. Posteriormente Cray formaría su propia compañía la Cray Research que comercializó el Cray I.

PDP 8

Construido por la casa DEC, se puede considerar el primer minicomputador. Tenía dos características importantes. Por un lado, en un momento en que la mayoría de los computadores necesitaba una habitación con sistema de ventilación para poder instalarse, el PDP se podía colocar sobre una mesa. Por otro lado, aunque no tenía la potencia de un mainframe, su reducido precio, en comparación con la familia de mainframes de IBM 360, que costaban cientos de miles de dólares, permitía que todos los laboratorios pudiesen tener uno.

Una de las aportaciones del PDP 8 a la estructura de computadores fue la utilización de un bus en lugar de la estructura centralizada que usaba IBM. En la actualidad esta estructura de bus se utiliza universalmente en microcomputadores y minicomputadores. El bus del PDP 8, llamado Omnibus, tenía 96 señales agrupadas en señales de control, de dirección y de dato. Puesto que todos los módulos tenían acceso al bus, la concesión de su uso estaba controlada por la CPU. La principal característica de esta configuración era la flexibilidad. Su continuador fue el PDP 11 que tenía 16 bits. Por último la casa DEC introdujo el VAX 11 de 32 bits.



ESTRUCTURA DE BUS DEL PDP-8

Control Data produjo varias series derivadas del 6600 que finalizaron con la aparición de la serie Cyber.

Además de todo lo anterior, aparecen importantes avances en el campo de la arquitectura como son las arquitecturas segmentadas, las paralelas y la microprogramación. Los sistemas operativos se desarrollan hasta tal punto que permiten a varios usuarios compartir el mismo computador. Sistemas operativos representativos son el MVS de IBM y el VMS de DEC. Igualmente se desarrollan las memorias cache y virtual.

Por último comentar el desarrollo de los lenguajes de alto nivel que se basan en los postulados dados por Dijkstra. Entre los lenguajes que aparecen destacan el BASIC, APL, Pascal.

Cuarta generación. Microprocesadores (1972-1987)

Hay dos avances que definen la nueva generación de computadores, la aparición de las memorias semiconductoras y el diseño del primer circuito que incluía todo un procesador.

Memorias semiconductoras

Hasta los años 60 la mayoría de las memorias eran de ferrita. Estas memorias tenían velocidades del orden de 1µ, ocupaban mucha área y eran de lectura destructiva. Por lo tanto, tenían que incluir la circuitería necesaria para restaurar el dato tan pronto como era leído.

En 1970 Fairchild produce la primera memoria semiconductora. Tenía un tamaño similar al de un corazón de ferrita, podía contener 256 bits de memoria, era de lectura no destructiva, y muy rápida. Su principal problema era lo elevado del coste por bit.

En 1974 tiene lugar un suceso muy importante. El precio por bit de una memoria semiconductora es inferior al de una memoria de ferrita. A partir de este momento las memorias se hacen más densas y baratas.

Microprocesadores

Del mismo modo que la densidad de las memorias iba aumentando, también aumentaba el número de elementos del procesador que se integraba en un sólo circuito. El salto cualitativo se produjo en 1971 cuando INTEL desarrolló el 4004. Este fue el primer circuito integrado de aplicación especifica que contenía todos los elementos de la CPU. A partir de este momento, se buscaba aumentar la potencia del microprocesador. En 1974 INTEL fabrica el 8080 que fue el primer microprocesador de propósito general. Este microprocesador tenía un bus de datos de 8 bits. Las tendencias se orientaron a conseguir mayor anchura del bus de datos. INTEL introdujo el 8086 de 16 bits y en 1985 el 80386 con 32 bits. Cabe también destacar la familia de microprocesadores de Motorola, la MC680x0.

En esta cuarta generación acaban de madurar conceptos aparecidos en la generación anterior, como son la segmentación, el paralelismo, la memoria cache y virtual, consiguiendo la fabricación de sistemas de computadores de alto rendimiento. Además llegan a ser dominantes las redes locales de estaciones de trabajo, disminuyendo la importancia de los grandes mainframes. Por último, se generaliza el uso de computadores personales, pequeños pero potentes.

Los computadores personales

La aplicación más revolucionaria del microprocesador fue la creación de los computadores personales y las estaciones de trabajo. La revolución de los computadores personales surgió con el primero de ellos el Apple II, desarrollado por un grupo de amigos y basado en el microprocesador 6502de tecnología MOS.

Posteriormente surgieron infinidad de ellos destacando los ZX80 y ZX81 considerados los primeros computadores domésticos europeos y el Spectrum de Sinclair.

RISC

En los cambios arquitectónicos introducidos en esta generación hay que destacar la aparición del concepto RISC (Reduced Instruction Set Computer) y la aparición de los computadores vectoriales, que se caracterizan por tener instrucciones que actúan directamente sobre operadores vectoriales. Generalmente se construyen en tecnologías muy rápidas como la ECL.

La quinta generación. El microprocesador como elemento básico (1988) La característica fundamental de esta época es el uso del microprocesador, para el diseño, tanto de computadores portátiles, como de supercomputadores.

En los microprocesadores actuales para aumentar la velocidad de procesamiento se utilizan técnicas de segmentación y paralelización En la segmentación se descompone la ejecución de las instrucciones máquina en pasos aislados. Con esto se consigue reducir los tiempos medios de ciclo y se consigue ejecutar varias instrucciones en paralelo. Los computadores superescalares utilizan microprocesadores que pueden ejecutar varias instrucciones en paralelo.

También en la actualidad se han extendido bastante los computadores paralelos de memoria distribuida formados por un conjunto de procesadores con memoria local conectados por una rápida red de interconexión que cooperan entre sí para resolver la misma tarea. Al principio estos computadores paralelos llevaban procesadores especializados pero el enorme tiempo de diseño y depuración de estos procesadores hace que la relación coste rendimiento disminuya si se diseñan con microprocesadores comerciales. Ejemplos son el CRAY 3TX y el IBM SP2

Por otro lado la idea de computador vectorial no se ha abandonado, solo que se tiende a utilizar tecnología CMOS en lugar de la ECL. También se utilizan los microprocesadores como elemento básico para el diseño de computadores paralelos de memoria compartida. Los microprocesadores se conectan a la memoria por medio de un bus como en el caso de las arquitecturas SG Power Challenge, Sun sparcserver, HP DEC8000. El número de procesadores de estos computadores suele ser inferior a 20.

1.6 BASES DEL DESARROLLO DE LOS COMPUTADORES

1.6.1 BASES DEL DESARROLLO DE LOS COMPUTADORES

Existen tres pilares sobre los que se asienta el desarrollo de los computadores,

- El avance tecnológico
- El avance arquitectónico
- El avance en las herramientas de diseño automático

Que duda cabe que uno de los motores del desarrollo de los computadores ha sido los importantes avances de la tecnología, que ha evolucionado desde las válvulas de vacío primitivas hasta los actuales circuitos integrados que pueden contener varios millones de transistores. Este aumento de la densidad de transistores en una oblea ha producido una disminución de los tiempos de ciclo y de los consumos de potencia debido a la reducción de los caminos y de las capacidades parásitas.

Pero este aumento de la densidad ha tenido otro efecto y es posibilitar el estudio y el diseño de arquitecturas que se debieron abandonar por necesitar mayor número de transistores de los que se podían integrar en un chip.

Esta necesidad de mayor número de transistores en un circuito es la que ha impulsado en gran media la investigación en el campo de la tecnología. Es decir, los avances tecnológicos impulsan los avances arquitectónicos mientras los avances arquitectónicos impulsan los avances tecnológicos, y ambos impulsan los avances en el campo de los computadores.

Por otro lado es importante darse cuenta que no sería posible el diseño de circuitos con varios millones de transistores si no fuera por la ocultación que se hace de los detalles de más bajo nivel para poder tratar los problemas en el ámbito de sistema, así como por la utilización de herramientas CAD que permiten la obtención de diseños correctos en tiempos de mercado infinitamente más cortos que los iniciales.

1.6.2 AVANCES TECNOLÓGICOS

La tecnología ha sido uno de los factores decisivos en la evolución que han sufrido los computadores. Los cambios tecnológicos han marcado puntos de inflexión en el desarrollo de los computadores debido a su gran influencia sobre la arquitectura.

Durante los primeros años la tecnología y la arquitectura fueron los motores del desarrollo. En cada generación la tecnología permanecía prácticamente estancada mientras la arquitectura, aprovechándose de las mejoras tecnológicas, aumentaba las prestaciones. De hecho, el ciclo de reloj reflejo de la mejora tecnológica solo varió durante estos años un orden de magnitud.

A raíz del descubrimiento del circuito integrado en 1965, el factor de mejora tecnológica ha permanecido prácticamente constante. La orientación que han seguido los circuitos integrados es la siguiente:

> *El nivel de integración de los circuitos integrados se ha multiplicado por cuatro cada tres años. Esta es la* ***ley de Moore****.*

Es decir cada tres años aparece una nueva generación con memorias cuatro veces más densas y procesadores cuatro veces más rápidos. Consecuencias del avance tecnológico son:

- Las interconexiones entre los elementos se han reducido, incrementando de esta manera la velocidad operativa.
- El precio de una oblea ha permanecido constante, mientras la densidad de integración ha aumentado lo que supone una disminución de precio por componente.
- El computador disminuye de tamaño, ampliando las aplicaciones en que puede utilizarse.
- Reducción de las necesidades de potencia y refrigeración.
- Disminuye el número de circuitos integrados por sistema por lo que disminuye el número de conexiones y los computadores se hacen más rápidos y fiables.

La disminución de la tecnología tiene dos efectos bien diferentes sobre el rendimiento de los sistemas. Por un lado, se reducen la longitud de las conexiones y las capacidades de carga de los circuitos por lo que se produce una disminución de los tiempos de carga y descarga de estas capacidades cuyo principal efecto es aumentar la velocidad del circuito. Además, esto permite que en el mismo área de silicio el número de componentes aumente cuadráticamente con la razón de disminución de la tecnología. Gracias a esto se pueden ensayar nuevas arquitecturas que suponen mayor complejidad y por lo tanto mayor número de transistores.

También debe hacerse notar que el tamaño del dado no permanece constante, sino que va aumentando, lo que da lugar a un nuevo incremento del número de dispositivos que se pueden integrar, lo que redunda en beneficio de nuevas arquitecturas.

1.6.3 AVANCES ARQUITECTÓNICOS

1.6.3.1 El modelo Von Neumann

El primer avance de la arquitectura apareció para superar la dificultad de programación del ENIAC. Esta programación se realizaba manualmente manipulando cables e interruptores. La solución fue almacenar los programas en la memoria. La arquitectura a la que dio lugar se utiliza en la mayoría de los computadores y consta de cuatro bloques básicos:

- la unidad aritmético lógica
- la unidad de control
- la memoria principal
- los sistemas de entrada/salida.

1.6.3.2 La Microprogramación

Fue propuesta por Wilkes en 1951, pero la lentitud de las memorias de aquel entonces impedía su utilización. Con este enfoque se ordenaba y sistematizaba el estudio de la Unidad de Control evitando la complejidad del diseño cableado. La microprogramación consiste en implementar una instrucción máquina mediante un conjunto de microinstrucciones, siendo una microinstrucción el conjunto de microoperaciones que se ejecutan en un ciclo de reloj. Los microprogramas se almacenan en una memoria de control. Es en 1964 cuando IBM la utiliza comercialmente en la familia 360, en la que la mayoría de los modelos (salvo los más avanzados) eran microprogramados.

1.6.3.3 La memoria principal

La tecnología a afectado de diferente manera al procesador y a la memoria. Los avances tecnológicos han supuesto un aumento importante de la densidad de las memorias pasando de memorias de 16 bits en 1965 a memorias de 16 Mbits en 1995, cumpliéndose así las previsiones de la ley de Moore. Este aumento en la densidad de componentes, no se ha visto correspondido con incrementos espectaculares en los tiempos de acceso.

Los tiempos de acceso se ven multiplicados por un factor de 1,07 cada año, mientras que las velocidades de los procesadores se ven multiplicados por un factor de 1,55. Esto provoca que cada vez sea mayor la barrera que separa los rendimientos del procesador de las velocidades de la memoria, llegando a ser el acceso a éstas uno de los principales cuellos de botella con los que se puede encontrar un diseñador.

Una de las técnicas habituales para minimizar este problema es utilizar memorias principales entrelazadas. Sin embargo la densidad de las memorias RAM está creciendo más rápidamente que la necesidad de memoria de los usuarios. Esto lleva aparejado una disminución de módulos de memoria en los computadores y por lo tanto disminuye la capacidad de entrelazamiento. Otra técnica posible es aumentar el ancho de bus, pero esto encarece mucho los sistemas.

En lugar de mejorar las interfaces de la memoria por medio de una organización externa de los módulos DRAM, en la actualidad se busca mejorar la organización interna. A continuación se comenta alguna de las organizaciones ya comercializadas:

- **EDRAM** (Enhanced DRAM) Incluye una pequeña memoria SRAM que almacena la última fila seleccionada de modo que si el siguiente acceso se realiza en la misma fila, solo se debe acceder a la rápida SRAM. Además permite realizar una lectura simultáneamente con el refresco o con una escritura.
- **CDRAM** (Cache DRAM): similar a la anterior pero con una memoria cache SRAM que almacena varias filas, siendo más efectiva para los accesos aleatorios de memoria.
- **SDRAM** (Synchronus DRAM) en lugar de ser una memoria asíncrona como el resto esta intercambia datos con el procesador sincronizada por una señal de reloj externa. Incluye un módulo SRAM que recoge la dirección y la orden y responde después de un cierto número de ciclos de reloj. Entre tanto el procesador puede ir realizando otra tarea.
- **RDRAM** (Rambus DRAM) A diferencia de las anteriores en este caso se cambia la interfaz entre la DRAM y el procesador, sustituyendo las líneas de selección de fila y de columna por un bus que permite otros accesos mientras se da servicio a un acceso, usando transferencias de ciclo partido.

Dado que todas estrategias decrementan los tiempos de acceso pero no reducen las diferencias entre procesador y memoria principal de manera significativa, se suele aprovechar el principio de localidad de los programas para introducir una memoria SRAM entre el procesador y la memoria DRAM lo que produce bueno resultados. A esta memoria se le llama cache y se comenta más adelante.

1.6.3.4 La memoria virtual

La memoria virtual apareció para facilitar la tarea del programador en los casos en que los programas eran tan largos que no entraban en la memoria. En estos casos, de manera transparente al usuario, el programa se dividía en bloques que iban de la memoria secundaria a la principal y de la principal a la secundaria según fueran las necesidades del programa. Este tipo de memoria también permitía la gestión de la multiprogramación.

Esta memoria la utilizó por primera vez el computador ATLAS diseñado en la Universidad de Manchester, aunque los primeros computadores comerciales que la utilizaron fueron los IBM/360. En 1974 la familia IBM/370 introdujo el mecanismo Translation Lookaside Buffer (TLB) para la traducción de direcciones.

1.6.3.5 La memoria cache

Uno de los problemas más importantes a solucionar en la actualidad por un diseñador de computadores es la gran diferencia que existe entre las velocidades del microprocesador y las velocidades de acceso a la memoria principal. Esto provoca que, en muchas ocasiones, el procesador esté inactivo lo que lleva aparejado una degradación del rendimiento. De las diferentes soluciones propuestas la más interesante es la utilización de una memoria interpuesta entre el microprocesador y la memoria principal que permita aprovechar la localidad que aparece en la ejecución de los programas para aumentar el rendimiento del sistema. Esta memoria debe ser pequeña y rápida para que se acerque a las velocidades de procesamiento. En la actualidad es una técnica muy generalizada.

Fue Wilkes en 1965 el que introdujo el concepto de memoria cache y poco tiempo después, en 1968 el modelo 360/85 la incluía en su arquitectura. El primer microprocesador con la memoria cache en el interior del chip fue el Motorola 68020. Posteriormente aparecieron computadores con dos niveles de cache uno interno y otro externo como por ejemplo el SUN 3/250 en 1986 y caches separadas para datos e instrucciones (4D/240 de Silicon en 1988).

El tiempo medio de acceso a la memoria es función del tiempo de acceso a la memoria cache, la tasa de fallos, y las penalizaciones por fallos. El aumento del grado de asociatividad y del tamaño de la cache disminuyen la tasa de fallos, pero pueden repercutir negativamente aumentando el tiempo de acceso a memoria cache al aumentar la complejidad de la lógica de control. Por otro lado, el aumento del tamaño del bloque disminuye la tasa de fallos, pero puede llegar a aumentar el

tiempo medio de acceso al aumentar la penalización de cada fallo. En el diseño se debe encontrar un compromiso entre todos estos factores.

1.6.3.6 Estructura RAID de memoria secundaria

Otro importante escollo que se encuentra un diseñador de computadores es la lentitud de acceso a las memorias secundarias. Este problema se puede solucionar utilizando varios discos a los que se accede en paralelo en lugar de utilizar un único disco. A esta estructura se le llama RAID (Redundant Array of Independent Disks). Además de acelerar la velocidad de acceso, se puede guardar información redundante para aumentar la fiabilidad del sistema.

1.6.3.7 Computadores segmentados

La segmentación es una de las principales estrategias para incrementar el número de instrucciones ejecutadas por ciclo (CPI). Esta técnica se basa en la explotación del paralelismo al nivel de instrucción, que es un paralelismo que permanece oculto para el programador. El procesamiento de una instrucción se puede subdividir en varias etapas que son: la búsqueda de la instrucción, la descodificación, la búsqueda de los operandos, la ejecución y la escritura de resultados. Utilizando una unidad de procesamiento segmentada es posible procesar varias instrucciones simultáneamente, manteniendo a cada una en una etapa diferente del pipe-line.

Uno de los problemas de la segmentación es la dependencia de datos y de control, que tiene como efecto la disminución del rendimiento del pipe. Para reducir estos problemas han surgido varias técnicas como la anticipación y la planificación de operaciones para el caso de dependencia de datos y la predicción o el retardo de saltos para el caso de dependencias de control.

La segmentación mejora el rendimiento de una máquina sin cambiar en principio el tiempo básico de ciclo. Esto implica que la segmentación es una buena técnica arquitectónica cuando el número de puertas que se pueden integrar en un ci aumenta más rápidamente que la velocidad de las puertas lógicas.

A finales de los años ochenta casi todas las arquitecturas incluían segmentación, y en algunos casos la tasa de CPI (Ciclos de reloj por instrucción) estaba próxima a uno. Esto se conseguía desarrollando repertorios de instrucciones poco propensos a las dependencias como es el caso de las arquitecturas RISC.

El primer computador segmentado fue el IBM 7030, que apareció a finales de los años 50, sucesor del modelo 704. El CDC 6600 presentado en 1964 y que es considerado el primer supercomputador, fue el primer que tuvo en cuenta la relación

entre repertorio de instrucciones y la segmentación. El IBM 360/91 introdujo muchas de las técnicas utilizadas hoy en día para aumentar el rendimiento de la segmentación

1.6.3.8 Procesadores superescalares y supersegmentados

Para aumentar el rendimiento del procesador se introducen nuevas técnicas como son la supersegmentación y los procesadores supersegmentados.

Un procesador supersegmentado divide cada una de las etapas de procesamiento de una instrucción en varias subetapas reduciendo el ciclo de instrucción. Esto da lugar a pipes más profundos. En resumen, mayor número de etapas pero etapas más sencillas. EJ MIPS R4000 incorporado en las estaciones Silicon Graphics.

Un procesador superescalar es aquel que es capaz de ejecutar más de una instrucción por ciclo de reloj. Lo que se está explotando es el paralelismo al nivel de instrucciones y por lo tanto para que funcione es necesario que existan instrucciones independientes entre sí que estén convenientemente ordenadas y que el procesador tenga el HW necesario para ejecutarlas en paralelo. Con ambas técnicas se consigue aumentar el número medio de instrucciones que se ejecutan por ciclo. La técnica superescalar se propuso en 1987 para mejorar las arquitecturas RISC segmentadas y se implementó en un Power 1 de IBM (línea RS/6000) en el año 1990. Otros procesadores son el TI SUPERSPARC, Motorola 88110, HP-PA 7100. Otro superescalar importante es el PENTIUM. El pentium tiene un núcleo RISC junto con un amplio soporte de la gama de instrucciones de los 80x86. Esto en realidad se traduce en una aproximación entre los RISC y los CISC. Por último comentar que existen arquitecturas que reúnen las dos técnicas como es el caso de DEC Alpha

1.6.3.9 Supercomputadores

Dentro de los supercomputadores se pueden encontrar dos tipos de computadores diferentes: los computadores vectoriales y los computadores paralelos.

Los **computadores vectoriales** son aquellos que incorporan a su repertorio instrucciones cuyos operandos son vectores. Desde el punto de vista arquitectónico la utilización de operaciones vectoriales presenta las siguientes propiedades:

- El cálculo sobre cada componente es independiente del cálculo de los demás
- Una instrucción vectorial puede desplazar a un bucle de operaciones escalares sobre los elementos de un vector. De esta manera se reduce el número de instrucciones que es necesario buscar en memoria para ejecutar un cálculo.

- Cuando los elementos de un vector están en posiciones consecutivas de la memoria, el acceso a los elementos de este vector es secuencial y conocido. Esto facilita la utilización de memorias entrelazadas para acelerar los procesos.

En definitiva los computadores vectoriales son capaces de extraer el paralelismo inherente a las operaciones con vectores. Un elemento importante en los computadores vectoriales es el estudio de las operaciones escalares. De nada serviría un computador vectorial que trabaja muy rápidamente con operadores vectores si después este rendimiento se ve degradado cuando se opera con escalares.

También hay que destacar la importancia de los compiladores en este tipo de computadores. Los compiladores deben ser capaces de extraer el paralelismo vectorial de las aplicaciones científicas que ya existían para computadores no vectoriales. Los podemos clasificar en dos grupos:

- Computadores vectoriales segmentados
- Computadores vectoriales SIMD en array

Los computadores vectoriales segmentados son capaces de ejecutar operaciones especiales de para manipular vectores. Para ellos poseen unidades funcionales segmentadas para procesamiento vectorial, capaces de procesar un componente del vector por ciclo de reloj. Se pueden distinguir dos tipos de computadores vectoriales segmentados:

- registro - registro
- memoria - memoria

Los primeros disponen de registros vectoriales para almacenar tanto los operandos como los resultados de las operaciones vectoriales. Ejemplos de estos son el Cray 1, Cray 2 X-MP, Fujitsu VP100 y VP200. En los segundos los operandos y resultados son leídos y escritos directamente en la memoria. Ejemplos son CDC STAR 100 y Ciber 205

En cuanto a los computadores SIMD en array disponen de múltiples elementos simples de proceso supervisados por una unidad de control común. Todos los elementos de proceso operan en paralelo y síncronamente sobre cada una de las componentes de un vector realizando todos la misma operación sobre distintos datos . El primer computador SIMD fue el ILLIAC IV desarrollado en 1972. Ejemplo e la actualidad son el (MaPar, MP-1, CM-2, DAP 600). Esta arquitectura ha caído en desuso debido a que solo puede obtener rendimiento en las aplicaciones con paralelismo en datos y las unidades funcionales se debían diseñar e implementar específicamente para la arquitectura, elevando enormemente los costes.

Por otro lado, el pequeño tamaño, el bajo coste y el alto rendimiento de los microprocesadores ha dado lugar al diseño e implementación de arquitecturas paralelas basadas en múltiples procesadores, que ofrecen importantes ventajas en fabricación, relación precio rendimiento y fiabilidad. Frente a los computadores más tradicionales. Se pueden encontrar dos tipos de computadores paralelos los multiprocesadores de memoria compartida y los multicomputadores de memoria distribuida por paso de mensajes.

Los multiprocesadores de memoria compartida se caracterizan por que los microprocesadores comparten un único espacio de memoria de direcciones global. Las tareas de cada procesador puede compartir datos de la memoria por eso es necesaria la sincronización y el acceso exclusivo para asegurar la consistencia de la memoria. Ejemplos típicos son el sg power (basado en el procesador R8000/R10000) y el DEC80000 basado en el procesador DEC 21164.

Su principal desventaja es la falta de escalabilidad y esto potenció el estudio de los multicomputadores de memoria compartida que consistían en múltiples procesadores conectados en red principalmente como malla o como hipercubo. Ejemplos típicos son el IBM SP2 Basado en el procesador power PC o el Vpp basado en un procesador vectorial VLSI El principal problema de estas arquitecturas es la dificultad de programación eficiente La tendencia en la actualidad es crear espacios de memoria virtuales únicos en esta línea están el Cray T3E basado en procesador alpha o el convex SPP basado en procesadores HP-PA

De los computadores vectoriales destacar que la tendencia ha sido utilizar la tecnología ECL para su implementación que es mas cara e impone estudios de consumo de potencia y disipación de calor. Esta ha sido la tendencia de los Cray. Existe otra tendencia que es la de utilización de tecnologías CMOS, que da lugar al concepto de minisupercomputadores.

1.6.3.10 Arquitecturas RISC

La arquitectura RISC es con toda probabilidad la principal aportación a la arquitectura que se ha producido en los años ochenta, debido a que a roto la tendencia evolutiva de las arquitecturas que cada vez se hacían más complejas.

Arquitectura CISC

La utilización de lenguajes de alto nivel ha provocado un salto semántico debido a la diferencia que existe entre estos lenguajes y el lenguaje máquina que ejecuta un computador. El efecto inmediato de este salto es una creciente complejidad de los computadores y un tamaño excesivo de los programas en lenguaje máquina lo que en ocasiones les hacía ineficientes.

Para reducir este salto semántico los diseñadores optaron por incrementar la complejidad de la arquitectura de los computadores incluyendo tipos de datos complejos, decenas de modos de direccionamiento y grandes repertorio de instrucciones, incluyendo algunas que implementaban directamente sentencias complejas de los LAN.

Arquitecturas RISC (Reduced Instruction Set Computer)

Los estudios realizados por algunos investigadores sobre las características de ejecución de programas en lenguaje máquina a partir de programas escritos en LAN daba como resultado que las instrucciones máquina que se ejecutan con mayor frecuencia dinámica son las de movimiento de datos, control de flujo, comparación lógicas y aritmética simple, representando estas más del 90% de las instrucciones ejecutadas. Además, los modos de direccionamiento simples representan más del 75 % de los usados dinámicamente.

Basándose en estos estudios surgen las arquitecturas RISC que reducen la complejidad de los computadores al implementar solo aquellas instrucciones más usadas, y utilizar sólo modos de direccionamiento sencillo, tipos de datos básicos y gran número de registros que sirven para almacenar datos temporales.

En 1980 Petterson de la universidad de Berkeley diseño dos máquinas, la RISC I y la RISC II, que dieron nombre al nuevo tipo de arquitectura. En 1981 Henessy de la universidad de Stanford publicó la descripción de la máquina MIPS. En 1986 aparecen los procesadores MIPS R20000, hp precision architecture e IBM RT-PC. Mas adelante aparece la arquitectura SPARC, derivada de la RISC II, el RS 6000 y el PowerPc.

1.6.4 INFLUENCIA DE LAS HERRAMIENTAS CAD

Los avances explicados tanto de la tecnología como la arquitectura de los computadores posiblemente no hubieran sido posibles si en paralelo a ellos no se hubiera producido un desarrollo en los métodos y técnicas de diseño de dichos circuitos.

La rápida evolución que han sufrido los métodos de diseño no solo está determinada por los avances tecnológicos y arquitectónicos, sino también por la necesidad de acortar los tiempos de mercado, factor crucial para que el diseño tenga éxito. Es posible que un diseño muy sofisticado sea sobrepasado en prestaciones por otro más sencillo, pero con un ciclo de diseño más corto.

Es sus etapas iniciales, el diseño se realizaba manualmente y los transistores había que diseñarlos y optimizarlos individualmente, teniendo en cuenta los que había a su alrededor. Esta técnica era difícil cuando se implementó el procesador INTEL 4004 en 1971, pero es absolutamente imposible de aplicar en la actualidad a procesadores como el pentium que tienen varios millones de dispositivos.

Uno de los primero avances que se produjo en las metodologías de diseño fue el estudio jerárquico de los sistemas lo que permitía aproximaciones mas sencillas. Un sistema se compone de módulo, los módulos de celdas y las celdas de transistores. Se buscaba la reutilización de las celdas para reducir el esfuerzo de diseño.

Otra avance importante en las metodologías de diseño fue la utilización de herramientas que automatizaban las partes más tediosas del proceso y que permitían ensayar diferentes soluciones arquitectónicas antes de la implementación definitiva. Dentro de estas herramientas destacan la captura de esquemáticos, la simulación funcional, la simulación eléctrica, place & route, la síntesis de layout a partir de esquemáticos de transistores, la compactación.

Otra herramienta importante es la de síntesis lógica que obtiene redes de puertas lógicas a partir de descripciones booleanas o tablas de verdad. Estas herramientas se encuentran en fase comercial desde hace aproximadamente 5 años. La últimas herramientas que han aparecido a nivel comercial son las de síntesis a partir de descripciones RTL, que sistemas comerciales como Synopsis o Compass ya las incorporan. El ultimo nivel que esta a punto de alcanzar la categoría comercial son las herramientas de síntesis de alto nivel.

Los lenguajes de descripción de hardware

En un principio se introdujeron con la intención de proporcionar una herramienta que permitiera la descripción de arquitecturas y estructuras de manera precisa y fiables y que además permitiera su simulación para comprobar su posible funcionamiento.

En los años ochenta se produjo un gran auge en la aparición de lenguajes de descripción de Hardware. De hecho cada grupo de investigación creaba su propio lenguaje con lo que se perdía uno de los principales objetivos de este tipo de lenguajes, la posibilidad de que una misma descripción fuera comprendida de manera universal.

Fue el Departamento de Defensa de los Estados Unidos, el que puso en marcha el programa Very High Speed Integrated Circuits (VHSIC) cuyo objetivo era la mejora de los circuitos integrados en los sistemas de defensa. En 1981 comienza el estudio de un lenguaje de descripción que fuera común a todos los proyectos que se desarrollaban dentro del marco del VHSIC. De esta manera surge el VHSIC Hardware Description Languaje (VHDL) cuyo principal objetivo era eliminar la anarquía reinante en este campo, en lo que a proyectos de defensa se trataba. El desarrollo lo llevaron acabo, IBM, Texas Instruments e Intermetrics, y concluye en 1986. Debido a la gran fuerza del Departamento de Defensa en la industria electrónica estadounidense se generalizó su uso, lo que indujo a su aprobación como un estándar del IEEE (IEEE std. 1076-1987).

De todos modos no conviene perder de vista que, aunque la tendencia del VHDL es la de ser el lenguaje dominante en años venideros, en la actualidad existen otros lenguajes de gran importancia como son el VERILOG, que utilizan el 46 por ciento de los diseñadores frente al 39 que utilizan VHDL.

2 EL RENDIMIENTO

2.1 INTRODUCCIÓN

El rendimiento del HW es clave para la efectividad de un sistema completo HW/SW, pero determinar el rendimiento de un sistema es difícil. Según que aplicación vaya a tener se debe utilizar una métrica u otra. Suele suceder que máquinas que se venden como de máximos rendimientos no dan la talla en las aplicaciones para las que se usan. La definición de rendimiento dependerá, por tanto, del sistema y de la aplicación.

Para comparar el rendimiento de dos estaciones de trabajo independientes manejadas por un único usuario serviría el tiempo de respuesta del computador, es decir el tiempo entre el comienzo y la finalización de la tarea (también llamado tiempo de ejecución). Pero si se quisiera comparar el rendimiento de dos grandes máquinas multiusuarios de tiempo compartido, la medida del rendimiento vendría dada por el número de tareas acabadas en un día, aunque esto vaya en detrimento del tiempo de respuesta (productividad). Incluso dentro de un mismo caso no es lo mismo la visión del rendimiento que tendrá el administrador que la del usuario.

2.2 MEDIDAS DEL RENDIMIENTO

Intuitivamente se puede decir que el computador que realiza la misma cantidad de trabajo que otro en menos tiempo es el más rápido. El tiempo puede definirse de diferentes formas según lo que contemos

ν **TIEMPO DE RESPUESTA**

También llamado tiempo de reloj o tiempo transcurrido, se define como el tiempo total que necesita el sistema para completar una tarea incluyendo:

- Accesos al disco
- Accesos a memoria
- Actividades de entrada /salida
- Gastos de sistema operativo

El principal inconveniente de esta métrica es que los computadores son de tiempo compartido en su gran mayoría, es decir un procesador trabaja en varios programas, por lo tanto no se puede asegurar que el tiempo de respuesta sea igual al tiempo gastado por el procesador en trabajar en un proceso. Para solucionar este problema se distingue entre tiempo de respuesta y tiempo de ejecución de la CPU siendo este el tiempo que la CPU emplea en realizar una tarea. No incluye los tiempos de entrada/salida de información, ni el tiempo de ejecución de otros programas. Este tiempo se descompone a su vez en Tiempo de CPU de usuario y Tiempo CPU de sistema, aunque esta subdivisión no se suele considerar por la complejidad de las medidas que acarrea. Otras métricas habitualmente utilizadas son la duración de los ciclos y la frecuencia. Estas métricas se pueden relacionar mediante las siguientes expresiones:

$$T_{CPU} = \text{nº ciclos de CPU para el programa} \cdot \text{duracion del ciclo}$$

Expresada de otra manera

$$T_{CPU} = \text{nºciclos de CPU para el programa/frecuencia}$$

Vistas estas expresiones, para mejorar el rendimiento habría que reducir la duración del ciclo o reducir el número de ciclos necesitados.

Otra métrica utilizada es la de Ciclos de reloj por instrucción (CPI) que es el número medio de ciclos de reloj que necesita cada instrucción para ejecutarse. Proporciona un medio de comparar dos implementaciones diferentes de una misma arquitectura ya que el recuento de instrucciones es el mismo. Se suele obtener por simulación detallada de una implementación y se relaciona con el tiempo de CPU mediante la siguiente expresión.

$$Tcpu = Nº instrucciones \cdot CPI \cdot duracion\ de\ ciclo$$

2.3 OTRAS MÉTRICAS DEL RENDIMIENTO

2.3.1 MIPS

Millones de Instrucciones por Segundo, que se puede relacionar con otras métricas a través de las siguientes expresiones:

$$\mathbf{MIPS} = \frac{\textbf{Millones de instrucciones}}{\textbf{Tiempo de ejecución}}$$

$$\mathbf{MIPS} = \frac{\textbf{Millones de instrucciones}}{\textbf{Ciclos de CPU} \cdot \textbf{Tiempo de ciclos}}$$

$$\mathbf{MIPS} = \frac{\textbf{Frecuencia}}{\textbf{CPI} \cdot \mathbf{10^6}}$$

Es una media de la frecuencia de ejecución de instrucciones para una máquina particular

$$T_{EJECUCION} = Nº instrucciones / MIPS \cdot 10^6$$

Tiene la ventaja de ser fácil de comprender. Intuitivamente se ve que máquinas más rápidas tienen mayores MIPS. Su principal problema es que especifica la frecuencia de ejecución pero no depende del repertorio de instrucciones, es decir no se pueden comparar computadores con diferentes repertorios de instrucciones ya que difieren los recuentos de instrucciones

- MIPS varia entre programas en el mismo computador
- Puede variar inversamente con el rendimiento. Puede ocurrir por ejemplo que el sistema no tenga implementada la multiplicación y tenga que implementarse por sw. Entones aunque se ejecuten muchas instrucciones el rendimiento será menor que el de un sistema que tenga un multiplicador.

- MIPS de pico o MIPS máximo se consigue utilizando una mezcla de instrucciones que difícilmente se va a dar en la realidad
- MIPS relativos se compara con otras maquinas, generalmente el VAX 11/780

2.3.2 MFLOP

Millón de operaciones en punto flotante por segundo, es el número de operaciones en punto flotante de un programa/tiempo de ejecución $10^{6.}$ Esta métrica depende del programa porque diferentes programas necesitan la ejecución de diferente número de operaciones en coma flotante. Esta medida se basa en las operaciones del programa y no en las instrucciones. Es mucho más útil para comparar diferentes maquinas que los MIPS porque el mismo programa ejecutándose en diferentes máquinas puede tener un número diferente de instrucciones, pero siempre tendrá el mismo número de operaciones en punto flotante.

Aunque esto último no es cierto porque el número de operaciones no es consistente entre máquinas y por lo tanto aunque el número supuesto de operaciones sea el mismo el número final de ellas no lo es. Ejemplo el Cray 2 no tiene operación de división mientras que el MOTOROLA 68882 sí, por lo tanto en el Cray 2 se necesitan varias operaciones en coma flotante para implementar la división.

Existen otros problemas en esta métrica, como que no todas las operaciones en punto flotante son iguales de rápidas. Por ejemplo no es lo mismo que las operaciones de punto flotante sean sumas a que sean divisiones. Para solucionar este problema se suele realizar un recuento de instrucciones en coma flotante en un programa escrito en alto nivel y ponderar las operaciones más complejas asignándolas un peso mayor. A este tipo de información se le llama MEGAFLOPS normalizados. De todos modos los resultados que se obtienen se alejan de la realidad.

2.4 PROGRAMAS PARA EVALUAR RENDIMIENTOS

La mejor manera de evaluar el rendimiento sería que el usuario que trabaja con los mismos programas diariamente, los ejecutara en el nuevo computador para evaluar su rendimiento. Pero esto es difícil que ocurra. Por eso se utilizan los Benchmark, que son programas escogidos para medir el rendimiento de una maquina. Los programas de prueba forman una carga de trabajo en teoría capaz de medir el rendimiento de la carga real de trabajo. Hoy en día se sabe que los mejores benchmark son aplicaciones reales, como las que el usuario emplea regularmente o aplicaciones típicas.

La utilización de programas reales como banco de prueba hace difícil que el diseñador pueda utilizar trucos para acelerar el rendimiento. Ha sido costumbre muy generalizada, cuando se han utilizado programas de prueba específicos, diseñar los compiladores para acelerar los segmentos de código del banco de prueba en que el esfuerzo computacional era mayor de este modo se obtienen unos resultados finales mejores de los reales.

Los bancos de pruebas aparecieron después de lo tiempos de CPU, los MIPS y los MFLOPS . No todos los bancos de prueba tenían las mismas características, y según éstas se podían clasificar en

- programas de prueba sintéticos
- núcleos
- Programas juguetes.

Los programas de prueba sintéticos eran programas totalmente artificiales que se basaban en cálculos intensivos. Ejemplos típicos eran el Whetstone, Dhrystone. Los núcleos eran trozos de programas reales que realizaban cálculo intensivo. Los más conocidos eran el Livermoore loops y el Linpack. Su objetivo era aislar el rendimiento de características individuales. Su principal inconveniente es que los resultados difieren del rendimiento real. Por último, los programas juguete eran pequeños programas de entre 10 y 100 líneas cuyos resultados finales eran conocidos. Programas típicos eran los de ordenación como el quicksort. Su principal ventaja era su facilidad de compilación y de ejecución por simuladores, permitiendo el calculo del rendimiento aproximado en las fases de diseño.

ν **SPEC**

En 1988 surge la compañía SPEC formada por IBM, DEC, INTEL Hewlett etc Su objetivo era desarrollar programas de prueba normalizados para evaluar la potencia de los computadores. Para ello lo que hacía era escoger un conjunto real de programas y entradas. Esta tarea se vio facilitada por la portabilidad de los SO y la popularidad de los LAN. Inicialmente fueron 6 bancos de prueba en coma flotante y 4 de enteros.

El uso de bancos de pruebas dio lugar a la aparición de la picaresca por parte de los fabricantes. Generalmente el rendimiento final dependía de pequeños segmentos de código que se repetían intensivamente. Para obtener mejores rendimientos los fabricantes introducían pequeñas modificaciones, o bien en la arquitectura o en los compiladores, de manera que falseaba lo resultados finales.

Este era el caso del programa matrix300, que formaba parte del banco de pruebas SPEC88. Este programa que multiplicaba matrices, estaba pensado para medir el acceso a la memoria de un sistema. Existían compiladores que reducían al mínimo los accesos a memoria consiguiendo rendimientos hasta 5 veces mayores. Este programa fue eliminado del banco de pruebas SPEC90 que se compone del SPECint, y SPECfp

SPECint

- Cálculos con enteros
- Minimizar funciones lógicas
- Traducir ecuaciones booleanas a tablas de verdad
- Cálculo de valores en una hoja de cálculo

SPECfp

- Cálculo en coma flotante
- Simulaciones de circuitos analógicos
- Cálculo de integrales derivativas
- Resolución de redes neuronales

2.5 RESUMEN DE MEDIDAS

Una vez que se ha decidido cuales son los programas de prueba a utilizar y se han ejecutado para determinar los respectivos rendimientos, hay que decidir como proporcionar la información al usuario. Si bien es cierto que un cuadro de rendimientos da mayor información, el usuario suele preferir una sola cifra que le ayude a comparar dos máquinas. Vamos a suponer que tenemos dos máquinas A y B y dos programas el programa 1 y el 2, cuyos respectivos tiempos de ejecución aparecen en el siguiente cuadro:

	Computador A	Computador B
Programa 1	1sg	10sg
Programa 2	1000sg	100sg
Tiempo total	1001sg	110sg

El rendimiento relativo sería B=1001/110=9,1 veces más rápido que B.

Otra medida es la media aritmética, es decir suma de todos los tiempos partido por el número total de procesos. A menor media aritmética mejor rendimiento. Esta media aritmética supone que todos los trabajos se ejecutan el mismo número de veces. Si se desea tener en cuenta que algunos programas se usan más que otros se debe realizar una media aritmética ponderada. Por ejemplo si el programa 1 supusiera el 20% de la carga de trabajo y el 2 el 80% los factores de peso sería 0,2 y 0,8

ν LEY DE AMDAHL

El posible aumento de rendimiento de una máquina para una mejora determinada está limitada por el uso que se de a la característica mejorada. Por ejemplo, de nada sirve mejorar muchísimo el multiplicador en coma flotante si luego solo lo utiliza el 0,001% de las instrucciones. Es mucho mejor introducir pequeñas mejoras en elementos que se usan mucho que introducir grandes mejoras en elementos que se usan poco:

HACER RÁPIDO EL CASO COMUN

Además conviene recordar que generalmente el caso común suele ser más simple y por lo tanto es más fácil de mejorar. Todas estas afirmaciones se pueden extraer de las siguientes expresiones:

$$T_{\text{CPU FINAL}} = T_{\text{CPU NO AFECTADO}} + \frac{T_{\text{CPU QUE QUIERES MEJORAR}}}{\text{cantidad de mejora parcial}}$$

$$T_{\text{CPU FINAL}} = \frac{T_{\text{CPU INICIAL}}}{\text{cantidad de mejora total}}$$

Vamos a ver un ejemplo de aplicación de la ley de Amdhal. Suponer un programa que se ejecuta en 100 segundos en una máquina. De estos 100 segundos 80 se deben a los cálculos que debe hacer un módulo multiplicador sobre los datos. Cuanto debe mejorar la velocidad de multiplicación si quiero que el programa corra cinco veces más rápido.

Tcpu inicial=100ns

Tcpu final deseado = 100/5=20

Tcpu no afectado = 20

T cpu afectado=80

20=80/n+20 → 0=80/n luego n=∞

Esto lo que indica que por mucho que se mejore el rendimiento del multiplicador no hay forma posible de reducir los tiempos a cinco veces su valor inicial.

sobrepase el tamaño de una página de memoria virtual. Por último los modos de direccionamiento deben ser sencillos, generalmente a registro, porque son fáciles de calcular.

El procesador debe tener un gran número de registros de propósito general, porque esto favorece las operaciones registro a registro, con lo que se mejoran los tiempos de acceso y se simplifican los formatos y modos de direccionamiento. La **Unidad de control debe ser cableada.** Como no son unidades excesivamente complejas, comparadas con la de los CISC, son fáciles de implementar en forma cableada que son más rápidas

En la actualidad parece que la tendencia es que las RISC incluyan características CISC como el Power PC y las CISC incluyan características RISC Pentium.

La selección de un conjunto de instrucciones demanda un equilibrio entre

- El número de instrucciones
- El número de ciclos necesarios para una instrucción y
- La velocidad del reloj

Existen cuatro principios que deben guiar al diseñador de repertorio de instrucciones para conseguir este equilibrio

- Cuanto más pequeño sea el HW más rápido será
- La simplicidad favorece la regularidad
- Los diseños siempre demandan compromisos entre diversos factores
- Hacer rápido el caso común

3 ESTRUCTURA BÁSICA DE LOS COMPUTADORES

La máquina de Von Neumann fue el primer computador de programa almacenado que se definió. Una máquina de programa almacenado es aquella en la que los datos y las instrucciones están almacenadas en memoria.

La maquina se compone de los siguientes módulos:

- La Memoria
- La Unidad Central de Proceso compuesta a su vez de la Unidad Aritmético Lógica y de la Unidad de Control
- La entrada/salida

ν **Los conceptos principales de la máquina de Von Neumman son:**

- La Memoria Principal almacena datos e instrucciones
- El sistema opera con datos binarios
- La Unidad de control interpreta las instrucciones de memoria y las ejecuta
- La entrada y salida la maneja la Unidad de Control

La primera máquina que implementó estos conceptos se llamó IAS, fue desarrollada en el Institute for Advanced Studies of Princeton, y no estuvo acabada hasta 1946-1952. En la actualidad, salvo contadas excepciones, todos los computadores tienen su misma estructura.

ν **MEMORIA PRINCIPAL.**

- Es la unidad destinada a almacenar instrucciones y datos.
- Se divide en palabras de memoria
- Cada palabra en un número de bits
- Todas las celdas son del mismo tamaño

ν **UNIDAD ARITMÉTICO LÓGICA.**

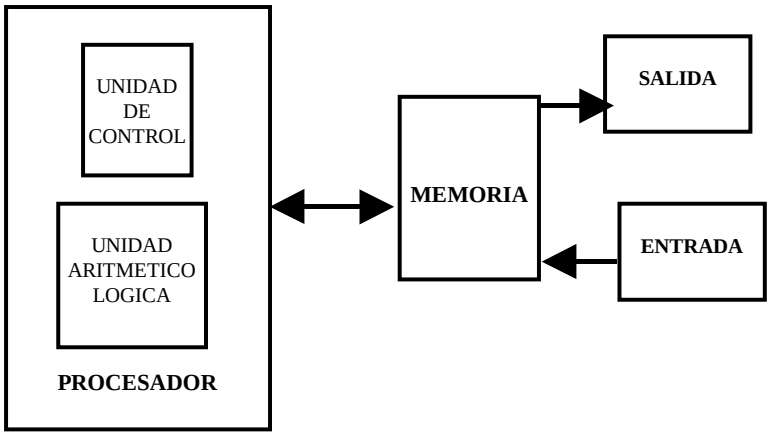
- Es la que realiza las operaciones elementales:
  - Suma
  - Resta
  - AND
  - OR
- Los datos sobre los que opera provienen de la Memoria Principal y se almacenan en la memoria principal, aunque pueden almacenarse temporalmente en los registros.

ν **UNIDAD DE CONTROL**

Es el módulo que se encarga de leer las Instrucciones, una a una, de la Memoria Principal y de generar las señales de control para que el computador las ejecute en el orden correcto. Las señales de control más habituales son de carga de registros, de selección de caminos en multiplexores o demultiplexores, de control de la memoria, apertura/cierre de puertas triestate.

ν **UNIDAD DE ENTRADA/SALIDA.**

Realiza las transferencias de información entre el computador y los sistemas periféricos



Def. Buses son caminos cuyo objetivo es hacer que las instrucciones y los datos circulen entre las distintas unidades del computador.

Def. Camino de datos

Es la suma de la unidad aritmético lógica, los registros generales accesibles por el programador, los registros particulares (CP, RI, RD...) invisibles para el programador y los caminos de conexión entre todos los elementos.

Nota: para comprender muchas de las decisiones que se toman a lo largo del curso es importante recordar que un sistema computador no se fabrica en en un único circuito integrado. Generalmente necesita muchos circuitos ubicados en tarjetas que se conectan entre sí a través de buses. En función de donde esté ubicado un circuito integrado, así será su retardo con respecto a otro circuito.

Los retardos de un sistema ordenados de menor a mayor son:

Los retardos internos a un chip

los retardos entre chips de la misma tarjeta impresa,

los retardos entre tarjetas impresas

los retardos debidos a las conexiones con periféricos (como las memorias secundarias)

Visto lo anterior, conviene recordar siempre que el microprocesador de un sistema computador suele contener la unidad central en un solo chip, y la memoria principal suele ser un conjunto de chips que se conectan al procesador a través de un bus. Por lo tanto es un millón de veces más rápido acceder a un registro del microprocesador que acceder a la memoria principal.

3.1 FASES DE EJECUCIÓN DE UNA INSTRUCCIÓN MÁQUINA

[Stalling] pp 50

La función básica de un computador es la ejecución de programas. Un programa está compuesto por un conjunto de instrucciones y datos almacenados en memoria. Las instrucciones se colocan una detrás de otra en la memoria, de manera cuando se leen se leen en el orden especificado por el programador. La forma de recordar cual es la siguiente instrucción a leer es mediante un registro que se llama contador de programa (PC).

La Unidad Central de Proceso es la encargada de ejecutar las instrucciones especificadas en un programa. Vamos a considerar con cierto detalle la forma que tiene un computador de ejecutar un programa. Desde la máxima simplificación el procesamiento de una instrucción se divide en dos fases:

- Una de búsqueda de la instrucción
- Ejecución de la instrucción

La ejecución de un programa consiste en la repetición de los dos pasos anteriores.

- **La búsqueda de la instrucción** es una fase común a todas las instrucciones que consiste en la lectura de la instrucción de una posición de memoria, es decir accede a la posición de Memoria Principal cuya dirección está contenida en el registro Pc y se carga el contenido en el Registro de Instrucciones. Este es un registro invisible para el programador que va almacenando la instrucción que se esta ejecutando.
- **La ejecución** se suele descomponer en varios pasos (o fases) que dependen de la instrucción a ejecutar. En la ejecución la CPU debe interpretar la instrucción y llevar a cabo la acción requerida que puede ser de cuatro tipos diferentes:

  transferencia de datos entre la CPU y la memoria

  transferencia de datos entre la CPU y la entrada/salida

  procesamiento de datos.- alguna operación aritmética o lógica

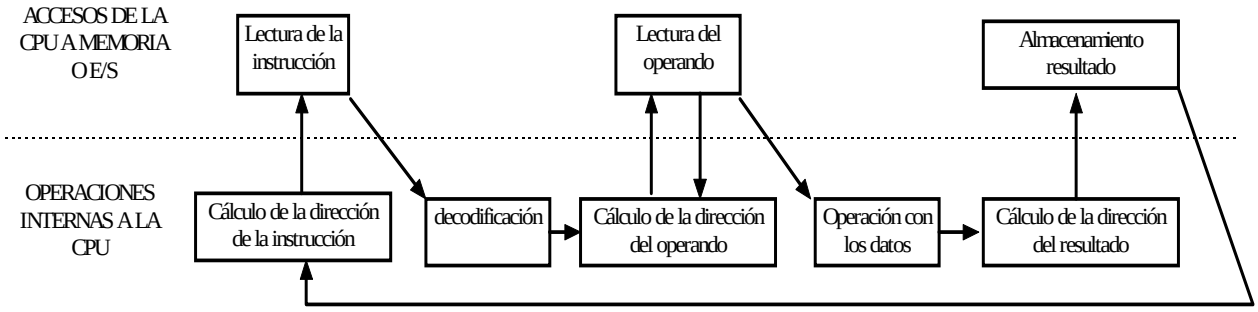
  control.- alteración de las secuencias de ejecución

Entrando más en detalle las fases de ejecución de una instrucción son

- calculo de la dirección de la instrucción
- lectura de la instrucción (carga de la instrucción almacenada en memoria en el Registro de Instrucciones)
- Descodificación de la instrucción (fase en la que La unidad de control se dedica a averiguar cual es la rama de la máquina de estados finitos debe seguir. En esta fase no se generan señales de control.)
- calculo de las direcciones de los operandos (las direcciones vienen especificadas en la instrucción, pero en ocasiones hay que realizar cálculos con ellas para hallar el valor verdadero)
- lectura de los operandos (se leen de la memoria principal y se almacenan en registros a la entrada de la Unidad Aritmético lógica)

operación con los datos (en la Unidad Aritmético Lógica)

- almacenamiento de operando (escritura del resultado en la memoria)
- el siguiente paso seria de nuevo el cálculo de la dirección de la instrucción.



4 EL REPERTORIO DE INSTRUCCIONES

4.1 CARACTERÍSTICAS Y FUNCIONES

4.1.1 INTRODUCCIÓN [STALLING]

La estructura de un computador es la implementación mediante módulos y redes de conexión de las especificaciones del sistema dadas en la Arquitectura. ¿Pero, como se define la arquitectura de un sistema computador? Mediante el repertorio de instrucciones. Éste se puede ver como la frontera en la que el diseñador y el programador ven la misma máquina. Un repertorio de instrucciones no sólo define las operaciones que el computador puede realizar (sumas, restas, etc) sino que también incluye descripciones del modelo de programación, es decir registros accesibles por el programador, modos de direccionamiento, tipos de operandos, tamaño de lo buses etc

Desde el punto de vista del diseñador el repertorio de instrucciones da las especificaciones funcionales de la Unidad Central de Proceso (CPU). El objetivo de los diseñadores de computadores es encontrar un repertorio que haga fácil la construcción del hw y del compilador al tiempo que se maximiza el rendimiento y se minimiza el coste.

Se define Lenguaje máquina como el que puede interpretar y ejecutar directamente el computador. Se compone de un conjunto de instrucciones máquina, cada una de las cuales realiza una acción específica y sencilla. En general, todos los lenguajes máquina son similares ya que todos los sistemas computadores están construidos con tecnologías similares. Para definir un repertorio de instrucciones hay que detallar:

a) Tipo de instrucciones a realizar
b) Tipos de representación de los datos
c) Modos de direccionamiento que son mecanismos utilizados para especificar un operando o la ubicación de un operando.
d) Formatos de las Instrucciones que indican como se codifica y distribuye la información en la instrucción

Además, el repertorio de instrucciones deber ser completo (que se pueda calcular en tiempo finito cualquier tarea computable) y eficaz, (que tenga velocidad de cálculo sin exigir a cambio una complicación excesiva de la unidad de control y de la unidad aritmética).

4.1.2 PROPIEDADES DE LAS INSTRUCCIONES MÁQUINA

- Cada instrucción debe realizar una función única y sencilla esto hace la descodificación sencilla
- Cada instrucción debe tener un número fijo de operandos. Con esto se consigue que el hw sea más sencillo
- Los operandos deben tener una representación predeterminada
- Se debe sistematizar la codificación para facilitar la descodificación
- Deben ser autocontenidas e independientes es decir:
  - Deben contener toda la información necesaria para ejecutarse
  - Su interpretación no depende de la posición que ocupan en la memoria
- La Información que debe contener una instrucción:

  Operación a realizar

  Operandos que intervienen

  Destino al que se envía el resultado

  Ubicación de la siguiente instrucción

4.2 TIPOS DE INSTRUCCIONES

La selección del juego de instrucciones de un computador es uno de los puntos críticos de diseño. Las Instrucciones más frecuentes son**:**

- Movimiento de datos
- Modificación de secuencia
- Aritméticas
- Comparación
- Lógicas
- Desplazamiento
- De entrada/salida

4.2.1 MOVIMIENTO DE DATOS

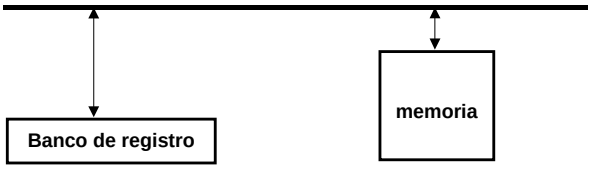
Llevan al destino la información contenida en el origen, quedando éste último sin modificar. El destino y el origen pueden ser tanto registros como posiciones de la memoria. Generalmente no modifican los biestables de estado aunque esto depende de la arquitectura. El formato con dos operandos es el siguiente:

move fuente, destino

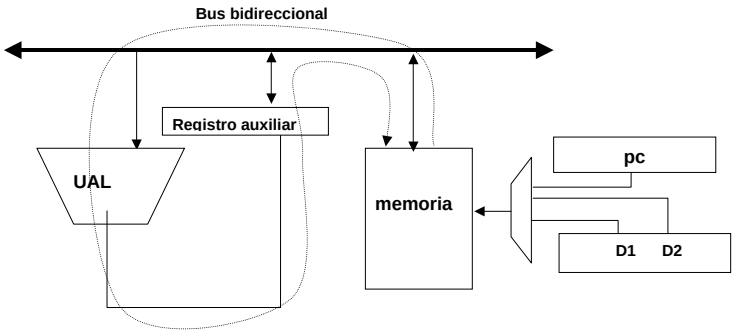
Pudiendo tener la fuente y destino las siguientes combinaciones:

- de memoria a memoria
- de memoria a registro
- de registro a memoria
- de registro a registro

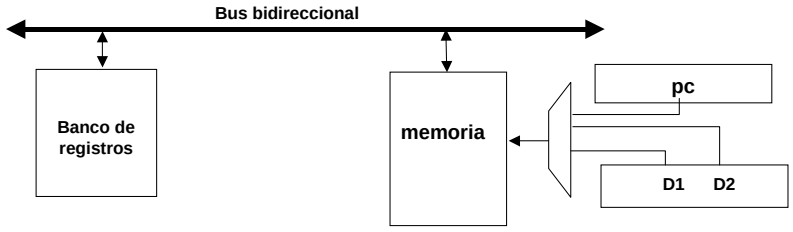
A continuación vemos como influye este tipo de instrucciones en el camino de datos. Si existe una instrucción de movimiento entre fuente y destino esto quiere decir que existe un camino físico entre ambos.



En ocasiones es posible que se necesite hw auxiliar para implementarlas. En el siguiente ejemplo suponemos una instrucción de movimiento de memoria a memoria y una estructura mínima sobre la que trabajamos que se compone de una memoria un bus bidireccional (un mismo bus para entrada y salida de datos de la memoria) y una unidad aritmético lógico. La instrucción puede implementarse utilizando la UAL como camino de paso sin modificar la información y utilizando un registro auxiliar para almacenar la información entre el paso de lectura y el de escritura de la memoria. Si no se usara este registro el dato se perdería.



En el siguiente ejemplo implementamos una instrucción de movimiento de memoria a registros

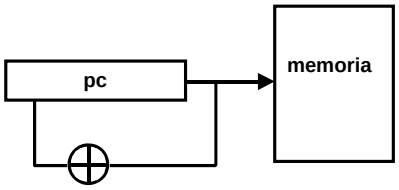


Las instrucciones de movimiento de datos tienen gran importancia en los computadores que utilizan el modelo de programación de registro-registro (típico de los RISC), porque son imprescindibles para mover la información de la memoria al banco de registros (Load) o del banco de registros a la memoria (store).

4.2.2 BIFURCACIÓN

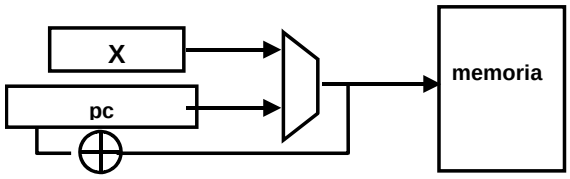
Son las instrucciones que se utilizan para tomar decisiones, tal como hacen las instrucciones if en los lenguajes de alto nivel. Los programas en lenguaje máquina se almacenan en posiciones consecutivas de memoria, y por lo tanto se ejecutan secuencialmente hasta que se encuentra una instrucción de bifurcación que altera la secuencia normal de ejecución del programa. El control de la secuencialidad lo lleva un Registro contador de programa (PC), que se suele incrementar automáticamente de la siguiente manera:

- PC ← PC+1 si la instrucción ocupa una palabra
- PC ← PC+Z si las instrucciones tienen un tamaño de Z palabras,



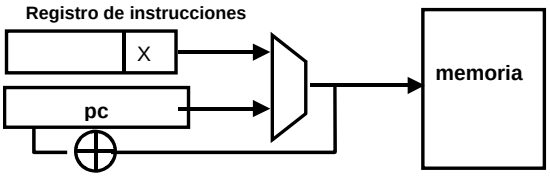
La bifurcación consiste en cargar el PC con la dirección X a la que se desea saltar

- PC ← X



- Las Bifurcaciones pueden ser:
  - Incondicionales
  - Condicionales
  - Bifurcaciones a subrutina, es decir con retorno
- **Bifurcaciones incondicionales**

En estas bifurcaciones se produce un salto siempre. Son las equivalentes al goto de los lenguajes de alto nivel



- **Bifurcación condicional**:

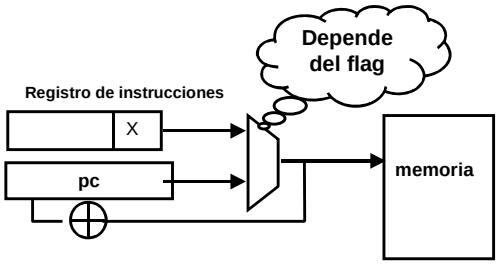
Es la que tienen dos secuencias distintas:

- Si se cumple la condición el contador de programa se carga con la nueva dirección: PC ← X
- Si no se cumple la condición el contador se autoincrementa PC ← PC+Z

Las condiciones se comprueban en los biestables de estado que almacenan información sobre operaciones realizadas con anterioridad.

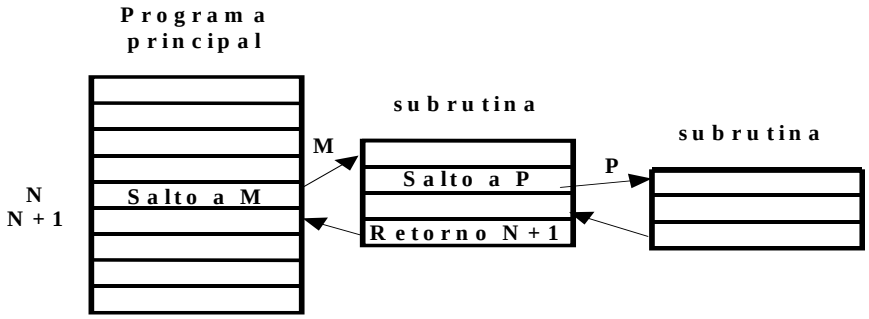
Condiciones típicas:

* Zero (Z) - Greater than (GT)
* Not Zero (NZ) - Greater than or Equal (GE)
* Equal (E) - Less than (LT)
* No Equal (NE)

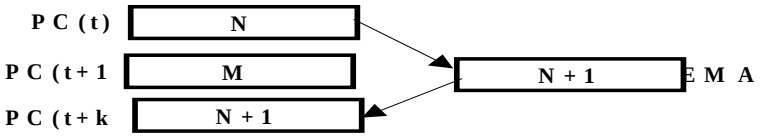


- **<u>Bifurcaciones con retorno o saltos a subrutina:</u>**

Una subrutina es un subprograma que implementa una determinada función, al que se puede saltar cada vez que se necesite. En definitiva es una forma de estructurar un programa que permite la reutilización del código. Una de sus principales características es que cuando acaba de ejecutarse vuelve al punto del programa desde el que se la llamó. La forma de implementarlo es guardar la dirección de retorno en algún elemento de memoria del procesador.



Evolución de los programas

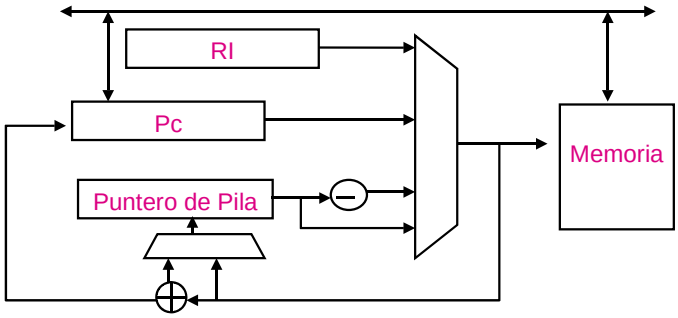


Evolución del pC y del Elemento de Memoria Auxiliar (EMA)

El problema es determinar dónde guardar la dirección de retorno de la subrutina. Para que el mecanismo sea eficaz deber permitir llamadas anidadas (una subrutina llama a otra subrutina) y subrutinas recursivas (que una subrutina se llame a sí misma.). Existen varias soluciones:

- Se guarda la dirección de vuelta en un registro especial. Su problema es que no permite anidamiento ni recursividad puesto que estas dos técnicas machacarían el contenido del registro en la siguiente llamada.
- Se guarda la dirección de vuelta en un registro general. Es similar a la anterior, con el inconveniente de utilizar uno de los registros del sistema.
- Se guarda la dirección de vuelta en la propia subrutina. Esta técnica permite llamadas anidadas puesto que al ser diferentes subrutinas las que se llaman las posiciones en las que se almacena la dirección de vuelta son diferentes, pero no permite la recursividad puesto que en este caso la posición en la que se almacena la dirección de vuelta es la misma .

- Se almacena la dirección de vuelta en una pila: permite llamadas anidadas y recursivas.



4.2.3 INSTRUCCIONES ARITMÉTICAS

La mayoría de las máquinas proporcionan operaciones aritméticas básicas para enteros con signo: Sumar, Restar,Multiplicar,Dividir

En ocasiones incluye operaciones para:

Punto flotante

Empaquetado decimal

Algunas otras operaciones típicas son las operaciones:

- Absoluto
- Negar
- Incrementar

Decrementar

Instrucciones de desplazamiento

Existe tres tipos de desplazamiento:

- Desplazamientos lógicos que son aquellos que no conservan el signo.
- Desplazamientos aritméticos que son aquellos que no conservan el signo. Son equivalentes a las operaciones de multiplicación y división por dos.
- Desplazamientos circulares.

Los tres tipos modifican los biestables de estado. La ejecución de las operaciones aritméticas suele incluir operaciones de transferencia de datos. Todas las operaciones aritméticas afectan a los bits de estado. Se verá su implementación en el tema dedicado a la unidad aritmético lógica.

4.2.4 OTRAS INSTRUCCIONES

Instrucciones de comparación

Se puede implementar o bien mediante una resta o con una operación XOR de cada bit de los 2 operandos. En esta operación no se almacena el resultado, pero sí se modifican los biestables de estado. Suele ir precediendo a una bifurcación condicional.

Instrucciones lógicas

Se realizan sobre cada bit de forma independiente. Estas operaciones modifican los bit de estado. Las más típicos son: AND, OR, NOT, XOR.

Instrucciones de E/S

Son operaciones de transferencia, cuyo destino y origen es el registro de un periférico. Pueden no existir si los periféricos están mapeados en memoria (tema de entrada/salida). Como los periféricos no se conectan directamente se transfiere el dato a un registro del interfaz de E/S que posteriormente se encarga de realizar el movimiento al periférico.

5 PARÁMETROS DE DISEÑO

5.1 TIPOS DE OPERANDOS

Los tipos de datos más importantes que se pueden encontrar en un Lenguaje Máquina son:

- Direcciones
- Números
- Caracteres
- Lógicos

NÚMEROS

Todos los lenguajes máquina incluyen datos numéricos (incluidos los procesadores de datos no numéricos). Una característica de los datos numéricos es que están limitados tanto en la magnitud representable, como en la precisión en el caso de la coma flotante. Los tipos numéricos más comunes son:

- Enteros representados en punto fijo
- Reales representados en coma flotante.
- Decimal que se utiliza en aplicaciones en las que hay una gran cantidad de operaciones de I/O y una computación comparativamente pequeña y simple.

En el tema de la Unidad Aritmético Lógica se dará un repaso a estas representaciones.

CARACTERES

Los computadores no pueden trabajar directamente con caracteres, por lo tanto se tendrán que codificar en formato binario mediante una cadena de bits. El código más común es el ASCII (American Standard Code for Information Interchange) del ANSI. En él un carácter se representa mediante 7 bits. Como se pueden representar 28 caracteres, que son más de los caracteres imprimibles, algunos de los patrones se utilizan para representar caracteres de control. En ocasiones se codifican con 8 bit, pudiéndose fijar este bit de más a cero o se puede utilizar como bit de error. La codificación que utilizan es muy sistemática:

- 011xxxx números decimales
- Desde 100 mayúsculas
- Desde 110 minúsculas
- Desde 0000000 hasta 0011111 de control

DATOS LÓGICOS

En lugar de considerar el conjunto de bits como un dato se considera cada bit un dato. Esto es útil para ahorrar memoria. Por ejemplo, si se quieren guardar matrices de 1s y 0s, en lugar de utilizar 8 bits para cada uno de ellos se utiliza solo un bit.

El tipo de dato que utiliza cada operación viene definido en el código de operación (tema 7). Por ejemplo supongamos que tenemos una suma aritmética y una suma lógica, entonces el código de operación correspondiente a la suma podría ser 010. Como tenemos que diferenciar entre los dos tipos de datos que utiliza la suma necesitaríamos un bit mas, luego los códigos de operación podrían quedar:

0100 suma aritmética

0101 suma lógica

5.2 MODOS DE DIRECCIONAMIENTO

Son procedimientos que permiten determinar un operando o la ubicación de un operando o una instrucción. Generalmente lo que se especifica es la dirección del operando. Se define dirección efectiva como la dirección en la que se encuentra el objeto, siendo un objeto cualquier elemento direccionable es decir, una

instrucción, un operando o un resultado. Los objetos pueden residir en la propia instrucción, en un registro o en la memoria principal.

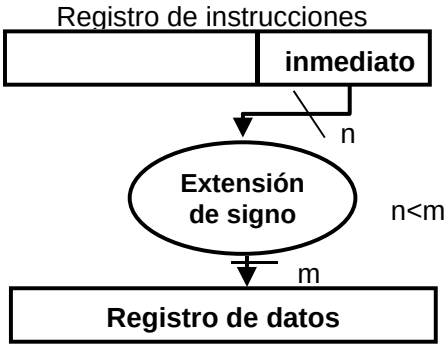
Los modos de direccionamiento existen por diversos motivos**.** Porque ahorran espacio de memoria. Por ejemplo. Si un bus de direcciones tiene 64 bits, para direccionar un elemento de memoria necesitamos instrucciones de más de 64 bits, puesto que una instrucción debe incluir mas información aparte de la dirección del operando . Si implementamos un modo de direccionamiento que solo utilice 32 bits conseguimos instrucciones más pequeñas. Cuánto más pequeñas sean las instrucciones, menos memoria se gasta. Los modos de direccionamiento suelen utilizar menos bits que los necesarios para direccionar directamente la memoria.

Además, su uso simplifica el manejo de las estructuras de datos complejas, dan flexibilidad a los programadores de lenguaje máquina y facilitan el diseño de compiladores. De las diferentes clasificaciones que existen nosotros vamos a usar la siguiente:

- Inmediato
- Directo
- Relativo o con desplazamiento que sirven para implementar código reubicable y reentrante.
- Indirecto

5.2.1 DIRECCIONAMIENTO INMEDIATO

El objeto, que es un operando, se incluye dentro de la propia instrucción. Se suele utilizar para definir y usar constantes e iniciar variables. Suele usar la representación en complemento a dos. Para representar el inmediato se utilizan un número de bits menor que el del bus de datos , por eso cuando se carga en el registro de datos se realiza una operación de extensión de signo, que debe figurar en el camino de datos. La operación de extensión de signo consiste en añadir al número inmediato tantos bits como necesite para que alcance el mismo tamaño que el registro de datos conservando el mismo valor numérico que representaba antes de la operación



Ventaja: No hace referencias a memoria, luego es un modo muy rápido

Desventaja: El tamaño del número está restringido al tamaño del campo dirección.

5.2.2 DIRECCIONAMIENTO DIRECTO

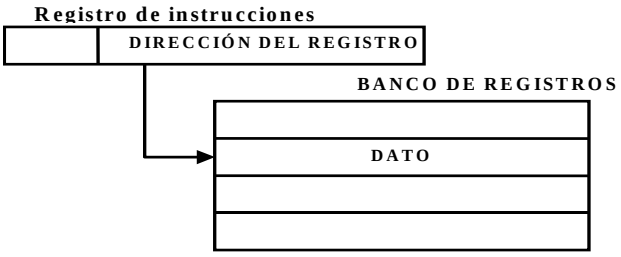
La instrucción contiene la dirección real del objeto, es decir, la dirección efectiva sin compactar. Se clasifican en

- Directa (Absoluta) de registro
- Directa (Absoluta) de memoria
- De página base

ν **Directo de Registro**

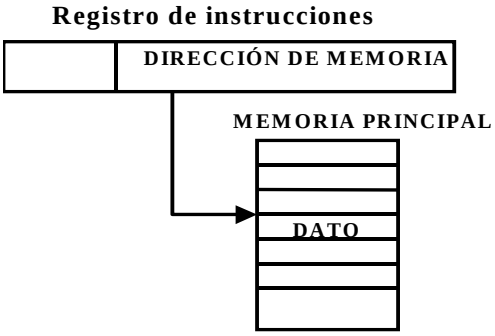
La instrucción contiene la dirección de un registro del banco de registro. Tiene dos ventajas. La primera que el campo de dirección es pequeño luego ahorra bits de memoria. La segunda que no se necesitan referencias a memoria, luego el acceso es más rápido. Su principal desventaja es que el espacio de direcciones accesibles está limitado al número de registros del banco. Además debido al pequeño número de registros de la CPU, el

uso intensivo de este direccionamiento provoca el uso ineficiente de éstos porque añade un paso intermedio a la ejecución al tener que traer el dato de MP al registro. En este último caso si es el mismo dato el que se utiliza en múltiples operaciones si se optimiza el tiempo.



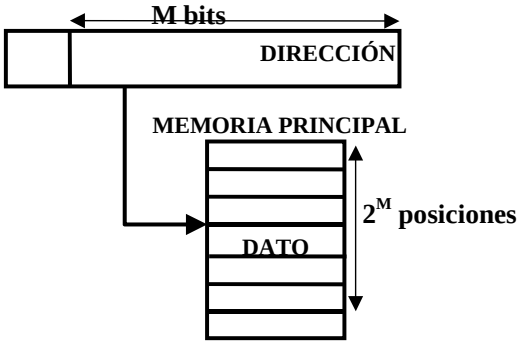
ν **De Memoria:**

La instrucción contiene una dirección completa de Memoria Principal. Su inconveniente es que el tamaño de la memoria está limitado por el número de bits asignado al campo dirección. Además el acceso es más lento que a un registro (recordar que la memoria principal se encuentran en un chip externo al microprocesador). Como ventaja tiene su facilidad de implementación y el amplio mapa de direcciones al que se puede acceder



ν **De Página Base**

La información contenida es una dirección limitada, que permite referirse solamente a una parte del mapa de memoria. Es decir tiene menos bits que los necesarios para direccionar toda la memoria. La principal ventaja que tiene es que se reduce el tamaño de la instrucción, luego consume menos bits de memoria. Su principal desventaja es que no puede acceder a todas las posiciones de la memoria. Por último sus tiempos de acceso son similares a los anteriores.



5.2.3 DIRECCIONAMIENTO RELATIVO

En estos modos la instrucción no contiene la dirección del objeto, sino un desplazamiento que se suma al contenido de un registro base para obtener la dirección efectiva. Se suelen permitir desplazamientos positivos y negativos (sumas y restas). Su ventaja es que el desplazamiento necesita menos bits para realizar el direccionamiento que el directo absoluto. Una característica importante de la implementación es que la suma no debe producir demasiado retardo en el cálculo de la dirección efectiva. Necesita más hardware para ser implementado y es más lento que el acceso directo puesto que previamente hay que realizar la suma. Es

muy interesante para tratar códigos reubicables y para recorrer de forma eficaz estructuras de datos. Se pueden encontrar los siguientes casos

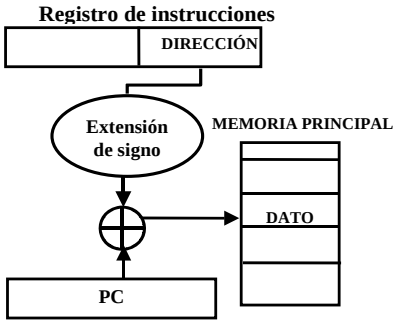
- Directo relativo a PC
- Directo relativo a registro base
- Directo relativo a índice
- Directo relativo a pila

Nota: En el Pedro de Miguel los direccionamientos relativos los clasifican como directos relativos y a los directos como directos absolutos. Como esta clasificación puede dar lugar a errores conceptuales, no la utilizo.

5.2.3.1 Direccionamiento Relativo a PC

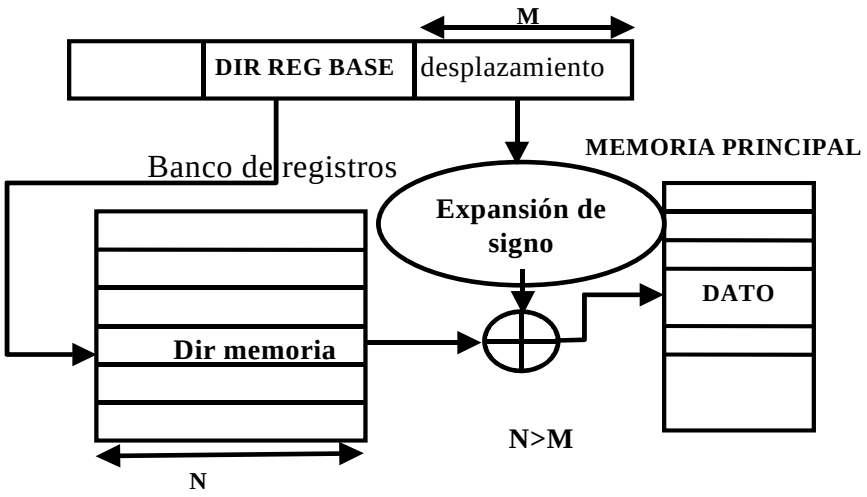
También llamado direccionamiento relativo. En este modo de direccionamiento el registro base empleado es el contador PC. Explota el concepto de localidad y por lo tanto es indicado para alcanzar instrucciones próximas a la que se está ejecutando. Ejemplo típico de esto son las bifurcaciones de bucles. Tiene la ventaja, frente otros direccionamientos relativos, de no necesitar referenciar al PC en la instrucción, luego ahorra bits.

La dirección efectiva se consigue sumando el contenido del PC y el campo dirección del registro de instrucciones. Como probablemente no tengan el mismo tamaño, habrá que utilizar un módulo extensión de signo para implementarlo



5.2.3.2 Direccionamiento Relativo a Registro Base

También llamado Direccionamiento de Registro Base, se caracteriza porque la instrucción debe contener dos informaciones: la dirección del registro base que contiene la dirección base y un desplazamiento que suele tener un tamaño inferior al de una dirección de memoria. Se utiliza cuando la dirección base no es conocida en tiempo de compilación (la información que se conoce en tiempo de compilación es la información que conoce el programador mientras escribe el programa) pero el desplazamiento relativo si que lo es.



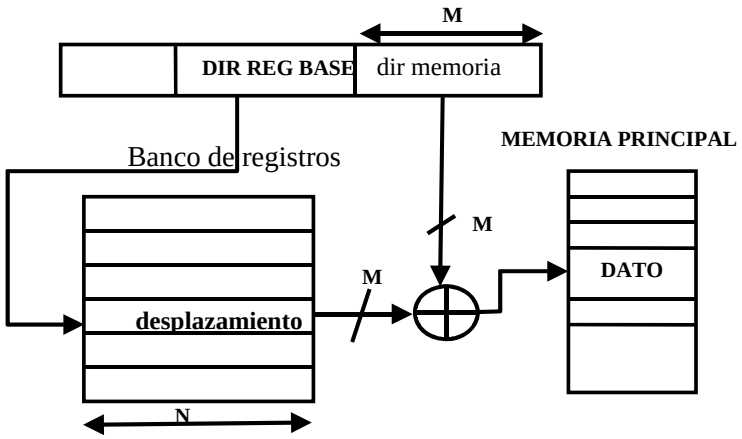
5.2.3.3 Direccionamiento Relativo a Índice

También llamado indexado. Es una modificación del anterior y por lo tanto la instrucción debe contener dos informaciones: la dirección del registro índice que contiene el desplazamiento (atención darse cuenta que es lo contrario de lo que ocurría en el indexado a registro base) y la dirección base de la memoria que debe estar referenciada con todos los bits. Es decir se consume muchos bits, puesto que la dirección de memoria es grande. Esta es la principal diferencia con el relativo a registro base. Si el nº bits de desplazamiento del registro base y los de memoria del registro índice coinciden, ambos modos de direccionamiento son indistinguibles.

En ocasiones el direccionamiento indexado se implementa con incrementos y decrementos automáticos del desplazamiento para facilitar el manejo de las estructuras de datos. Esto complica el Hardware, pero acorta los programas. Los más habituales son:

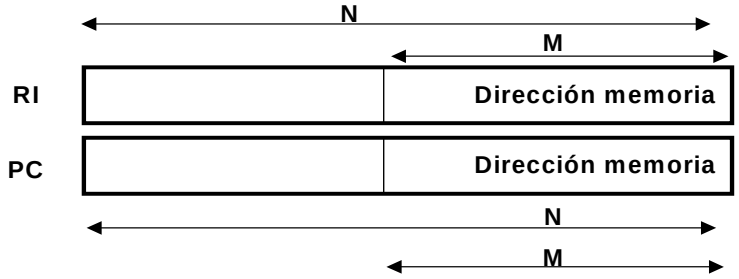
- Preautoincremento
- Preautodecremento
- Postautoincremento
- Postautodecremento

Los incrementos/decrementos deben adaptarse a la longitud de los operandos empleados. Se suele utilizar cuando se puede conocer la posición de un vector en tiempo de compilación, pero el elemento exacto al que queremos acceder no se conoce nada más que en tiempo de ejecución. Darse cuenta que en este caso y en el anterior la filosofía es similar. El valor que se conoce en tiempo de compilación y que por lo tanto es fijo es el que se guarda en la instrucción (que no se puede modificar en tiempo de ejecución). El valor variable es el que se almacena en el registro índice. Por lo tanto, antes de utilizar una instrucción que tenga un modo de direccionamiento relativo a índice o relativo a PC, habrá que poner una instrucción que cargue el registro base con el valor variable adecuado.



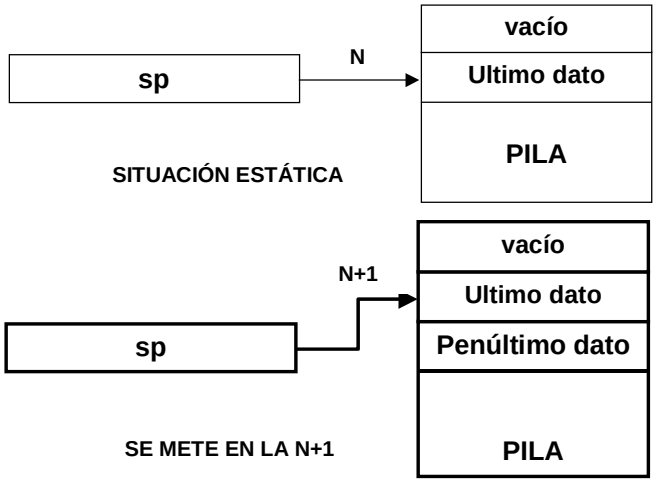
Del banco de registros solo se toman los bits necesarios para implementar la suma

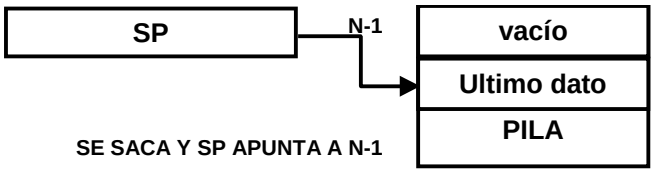
NOTA: Si el contador de programa y el registro de instrucciones tienen el mismo tamaño, y la instrucción contiene una dirección de memoria completa esto indica que no todos los bits de PC se utilizan para direccionar:



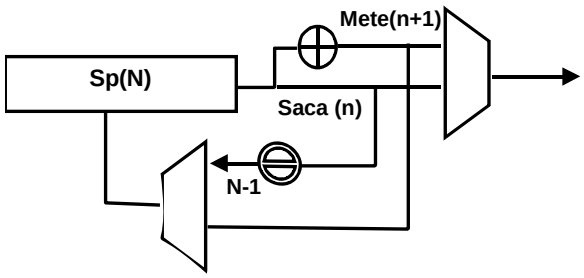
5.2.3.4 Direccionamiento Directo Relativo a Pila

Llamado también direccionamiento a Pila. Recordar que una pila es una cola del tipo primero en entrar último en salir. La máquina debe disponer de un registro SP (puntero a pila). Los accesos a pila deben tener un tratamiento especial. Para insertar un elemento en la pila, el direccionamiento relativo a SP debe generar un preautoincremento, y para sacar un elemento de la pila el direccionamiento relativo a SP debe realizar un postautodecremento. Permite instrucciones muy compactas y si sólo se dispone de un SP la instrucción no necesita información de dirección.



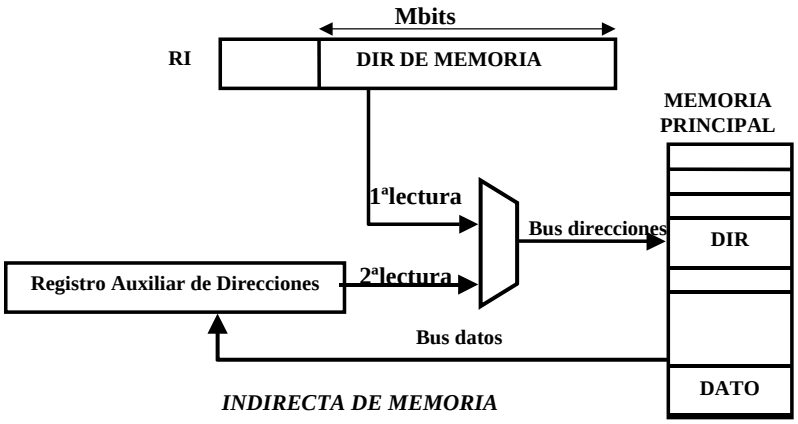


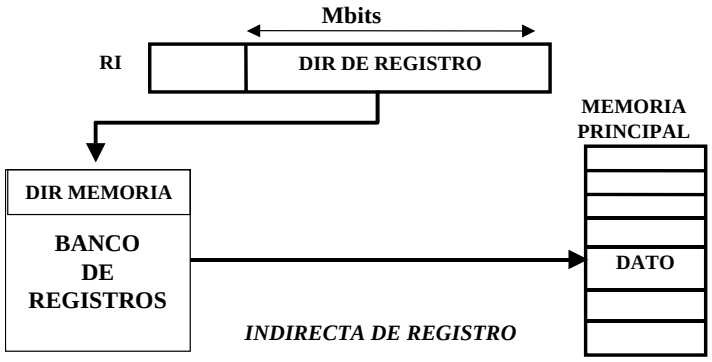
A continuación se ve una posible implementación del hw necesario para modificar el puntero:



5.2.4 DIRECCIONAMIENTO INDIRECTO:

La dirección contenida en la Instrucción **NO** es la del objeto, sino la de la Dirección del Objeto. Existen dos tipos: indirecta de memoria e indirecta de registros





En ambos casos es necesario un acceso adicional para obtener el objeto. En el indirecto a memoria se accede dos veces a la memoria, la primera vez para obtener la dirección efectiva del objeto y la segunda para obtener el objeto. No se suele utilizar la indirección multinivel porque retardaría en exceso la ejecución de los programas. Una aplicación típica de este modo de direccionamiento es el acceso a diversas informaciones mediante una tabla de punteros.

El indirecto a registro se utiliza muy a menudo por ser casi el doble de rápido que el indirecto a memoria, puesto que el acceso a registro es mucho más rápido que el acceso a memoria. Es útil cuando la dirección de una estructura de datos solo se conoce en tiempo de ejecución. Se puede encontrar con autoincremento y autodecrementos

5.3 FORMATO DE LAS INSTRUCCIONES

Es la forma en que se especifica el significado de cada bit de la instrucción. Entre las decisiones de diseño del formato, una de las más importantes es la longitud porque afecta al número de campos, tamaño de campos, tamaño de memoria etc. La longitud es el número de bits de la instrucción. Un formato debe contener la siguiente información:

- Operación que realiza la instrucción
- Dirección de los operandos
- Dirección resultado
- Dirección siguiente instrucción
- Tipos de representación de operandos

5.3.1 LONGITUD DE LA INSTRUCCIÓN

La decisión del tamaño que debe tener una instrucción es crítica en el diseño del computador. Esta decisión afecta y se ve afectada por:

- El tamaño de la memoria, ya que fija tanto el tamaño de la palabra de la memoria como el número de palabras de la memoria
- Estructura y tamaño del bus
- Complejidad de CPU
- Velocidad de CPU
- Determina la riqueza y flexibilidad de la máquina

El programador desea que su repertorio de instrucciones tenga más códigos de operación, más modos de direccionamiento, más operandos y mayor rango de direcciones, porque todo ello facilita la tarea de escribir programas. Todas estas características necesitan mayores longitudes de instrucción. El diseñador del sistema debe buscar un equilibrio entre la riqueza del repertorio de instrucciones y la necesidad de salvar espacio de memoria.

Como norma general la longitud de instrucción debe ser igual a la longitud de la transferencia de memoria o ser múltiplo de ella. En este último caso se pueden producir cuellos de botella: por ejemplo, dos accesos a Memoria para ejecutar una Instrucción.

5.3.2 CAMPOS DE LA INSTRUCCIÓN

La instrucción se divide en campos, donde un campo es una cadena de bits continuos. Cada campo proporciona información específica. Campos típicos de una instrucción son:

- Código de operación
- Dirección de operandos
- Modo de direccionamiento
- Extensión de campo

El Código de operación indica la operación a realizar por la instrucción. Es un campo con tamaño fijo. Por ejemplo si el campo tiene 8 bits esto indica que el repertorio tendrá 2^8 operaciones diferentes. El código de operación puede contener más información que la de la operación a ejecutar. Por ejemplo, los modos de direccionamiento pueden ir incluidos en este código, o tener un campo independiente.

Como no todas las instrucciones se utilizan con la misma frecuencia, en ocasiones se tienen códigos con menos bits para las instrucciones más utilizadas, de esta manera se optimiza espacio.

El campo de dirección especifica la dirección de un dato, resultado o instrucción a la que se bifurca. Lógicamente existirán tanto campos dirección como operandos tenga la instrucción. Igual que ocurría con el código de operación puede incluir implícitamente los modos de direccionamiento utilizados (si el modo de dirección se incluye en el código de operación no se incluye aquí y viceversa)

El código de modo que puede especificar alguna característica de la operación , como el tamaño de los operandos, o el modo de direccionamiento .

En algunas ocasiones se utilizan extensiones de código. En estos casos se tiene un número bits fijo y pequeño de para casi todas las operaciones y para algunas de ellas se utiliza una extensión. El código corto se suele utilizar en las instrucciones que más se utilizan. La extensión se suele utilizar en instrucciones del tipo bifurcación condicional y desplazamiento. Vamos a ver un ejemplo de esto

Supongamos qu etenemos el siguiente conjunto de instrucciones a codificar

Suma, resta, multiplicación, bifurcación incondicional, bifurcación si z, bifurcación si c, bifurcación si x, bifurcación si no x

La forma de utilizar la extensión de código seria la siguiente

Suma 00

Resta 01

Multiplicación 10

Bifurcación 11.

En el caso de la bifurcación debería existir un campo extensión que nos indicara el tipo de bifurcación

Incondicional 00

Si z 01

Ai x 10

Si nox 11

De manera que la codificación total seria

Suma 00

Resta 01

Multiplicación 10

Bifurcación Incondicional 1100

Bifurcación si z 1101

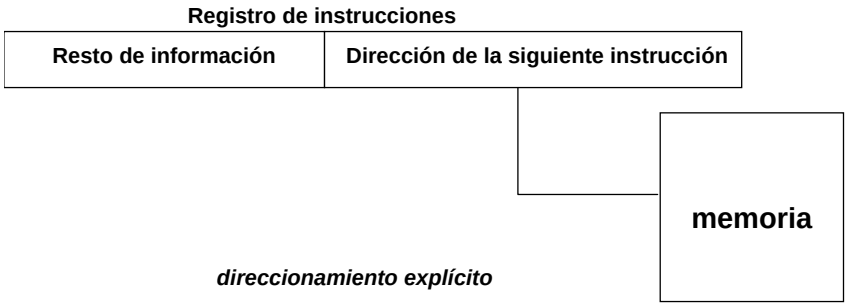
Bifurcación si x 1110

Bifurcación si no x 1111

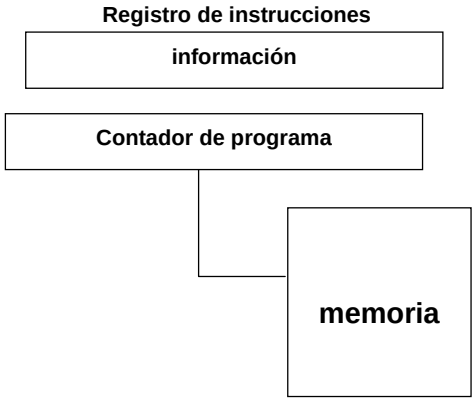
5.3.3 CARACTERÍSTICAS DEL FORMATO

Un computador tiene varios formatos y cada instrucción encaja en uno de ellos. Cuánto menos formatos tenga el repertorio, más sencilla será la implementación en hw del procesador.

Los formatos múltiples complican el HW. Esta complejidad se reduce si de un formato a otro, los campos del mismo tipo, tienen la misma longitud y ocupan la misma posición. Esto simplifica la codificación y los caminos internos necesarios para mover la información. Se suele utilizar direccionamiento implícito para ahorra tamaño de la memoria, puesto que evitas incluir la dirección de la siguiente instrucción en la instrucción actual.



direccionamiento explícito



direccionamiento implícito

El tamaño de la instrucción debe encajar con facilidad en la palabra máquina. Cuando existen varios formatos, es el código de operación el que diferencia entre ellos

Se debe intentar que instrucciones próximas utilicen codificaciones similares. Con esto se reduce la complejidad de HW

Como asignar los bits del formato. Para una longitud dada existe un compromiso entre el número de códigos de operación y la capacidad de direccionamiento. A mayor número de códigos de operación mayor cantidad de bits para codificarlo menor longitud para direccionar los operandos. Los factores que influyen en la distribución de los bits de direccionamiento son:

- Numero de modos de direccionamiento
- Número de operandos
- registros frente a memoria
- Número de bancos de registro
- Rango de las direcciones

5.4 MODELO DE EJECUCIÓN

Para poder comprender la diferencia que existe entre los distintos modelos de ejecución es importante recordar que los bancos de registro se encuentran dentro del chip procesador mientras la Memoria Principal se encuentra en otra tarjeta a parte, y se comunica con el procesador a través de un bus. Esto tiene como efecto que los accesos al banco de registros sean mucho más rápidos (orden de 8 ns) que los accesos a la memoria principal (orden de 120 ns).

Los operandos de las instrucciones pueden estar almacenados en posiciones de memoria o registros. Debido a sus grandes tamaños, las estructuras de datos como arrays se suelen almacenar en la memoria.

El modelo de ejecución especifica el dispositivo en que están almacenados los operandos para realizar las operaciones. Será el compilador el encargado de asociar variables de los programas a los registros y las estructuras de datos a la memoria.

ν **Pila**

Los datos se almacenan en la cabecera de la pila. El resultado de la operación se almacena en la pila. Según lo visto las Instrucciones no necesitan direccionar datos. Solo las de movimiento necesitan incluir direcciones para trasladar los datos de la memoria a las posiciones de la pila. Para realizar una suma primero habría que mover a la pila los dos operandos: mueve a, pila, mueve b, pila.

Después habría que realizar la suma. El microprograma se encargaría de sacar los dos operandos de la pila realizar la suma y de almacenar el resultado en la pila.

ν **Registro-registro**

La mayoría de los sistemas computadores tienen un banco de registros en el que se cargan los datos para poder operar con ellos. Estos registros son visibles por el programador. Generalmente los registros no son muchos, porque un gran número de ellos incrementaría la duración del ciclo de reloj.

La mayoría de los programas tienen más variables que registros la máquina. Y es el compilador el que intenta cargar las variables más usadas en los registros de la máquina y coloca las restantes en la memoria utilizando cargas y almacenamientos para mover las variables entre los registros y la memoria. Al proceso de colocar las variables menos utilizadas en la memoria se le llama Spilling. Las características de este modelo de ejecución son:

- Operandos están los registros
- Las Instrucciones llevan las direcciones de los registros en los que se encuentran los operandos

Como el banco de registros no es muy grande deben existir operaciones de movimiento de datos de la memoria a los registros y viceversa. Si los datos no están en los registros hace falta dos accesos a memoria para cargar los datos. Este es el modelo típico de los RISC

ν **REGISTRO-MEMORIA:**

El primer operando está en registro y el segundo operando en memoria (o viceversa). El resultado se carga en la posición del 2º operando. Solo hace falta un acceso a memoria.

ν **MEMORIA-MEMORIA**

Necesita 3 accesos a memoria, dos para traer los operandos de la memoria a la Unidad Aritmético lógica y uno para cargar el resultado en la memoria..

6 EL CAMINO DE DATOS

6.1 INTRODUCCION

Tanto la duración del ciclo de reloj como el número de ciclos de reloj por instrucción están determinados por la implementación del procesador. Para conseguir un mejor diseño es conveniente tener en mente los dos principios que a continuación se enuncian:

- Hacer rápido el caso común
- La simplicidad favorece la regularidad

En este capítulo se tratará inicialmente los tipos de registros que se pueden encontrar habitualmente en un computador, los diferentes tipos de buses que se suelen usar para implementar camino de datos y se desarrollará la implementación de un camino de datos para que el alumno tenga un modelo.

6.2 REGISTROS DEL PROCESADOR

El nivel más elevado de la jerarquía de memoria son los registros del procesador cuyas características son:

- Más pequeños
- Más rápidos
- Más caros

Existen dos tipos de registros en la CPU

a) Visibles por el usuario. Descritos en la arquitectura del computador. Los utiliza el usuario para minimizar el número de accesos a Memoria Principal.

b) De estado y de control. Son invisibles al usuario y con ellos la unidad de control controla la unidad central de proceso y el sistema operativo controla la ejecución de los programas

Los registros visibles por el usuario son los referenciables por el lenguaje máquina que ejecuta la CPU. Se pueden clasificar en:

- Propósito general.
- Datos.
- Dirección.
- Códigos de condición.

Los registros de condición son aquellos que guardan los códigos de condición, es decir los bits que activa la CPU como resultado de una operación. Se suelen comprobar en las instrucciones de bifurcación condicional. En ocasiones forma parte del registro de control: pueden ser leídos, pero no alterados.

Los registros de propósito general se usan para muchas funciones entre la que destaca: ortogonalizar el repertorio de instrucciones, es decir cualquier que registro puede contener el operando de cualquier código de operación. Por ejemplo los registros de propósito general pueden contener tanto direcciones como datos.

En ocasiones, puede haber separación entre los registros generales de datos y los de direcciones. Estos últimos pueden ser de propósito general o de propósito específico; dentro de esta última clase se encuentran los punteros a segmentos para implementar la memoria virtual segmentada que guardan la dirección base del segmento; los registro de índices y los puntero a pila.

6.2.1.1 Decisiones de diseño.

- **¿Registros de Propósito General o Específico?.** Los específicos propician el direccionamiento implícito con lo que se ahorran de bits, pero provocan falta de flexibilidad. La tendencia actual es hacia registros específicos.
- **Número de Registros.** Cuanto más registros, más bits se necesitan para direccionar el operando. Normalmente hay entre 8 y 32 registros. También es cierto que pueden acelerar la ejecución de programas, puesto que inicialmente cuanto más registros menos referencias a memoria pero llega un momento en que estas no se no reducen.

- **Longitud de Registro** Los que almacenan direcciones como poco deben tener la longitud de la dirección. Los registros de datos deben ser capaces de almacenar la mayoría de tipos de datos. Algunas máquinas permiten dobles registros.

6.2.2 REGISTROS DE ESTADO Y CONTROL

Se utilizan para controlar las operaciones de la CPU. Algunos pueden ser visibles cuando se ejecutan instrucciones en modo control. Dependen mucho de la máquina. Los más típicos:

- **PC:** contador de programa: contiene la dirección de la siguen instrucción. El contador de programa es actualizado por la unidad de control después de cada búsqueda de instrucción de manera que siempre exista un puntero a la siguiente instrucción. En los saltos y bifurcaciones, también se modifica el PC. La instrucción buscada se carga en el RI dónde se estudian el código y los operandos
- **RI**: Registro de Instrucciones: contiene la última instrucción traída de la Mp.
- **RD**: (MAR) Registro de Direcciones de Memoria.
- **MB**: (Memory Buffer Register), contiene una palabra leída de la memoria o que se va a escribir en la memoria.
- **Program Status Word (PSW)** contiene información de estado. Sus campos más comunes son: el signo de la última operación aritmética, Zero, Carry, Igual, Overflow y Supervisor

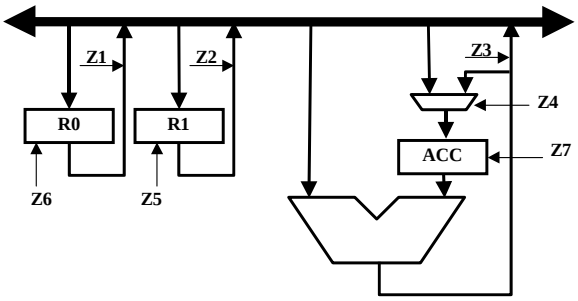
Puede existir un conjunto de registros que hacen de frontera entre el camino de datos y la UAL. Estos registros no suelen ser accesibles por el programador. Entre ellos destaca el acumulador. Su tamaño coincide con la longitud de palabra del procesador. Contiene uno de los operandos o el resultado.

6.3 CAMINO DE DATOS SECUENCIAL

Un bus es un grupo de líneas digitales que soportan un mismo tipo de información del computador como son direcciones, datos y control. Un camino de datos secuencial es aquel en el que el camino de datos sólo existe una UAL y por lo tanto para desarrollar un algoritmo complejo es necesario realizar algunas operaciones sencillas en serie

Camino de datos de un bus(Angulo)

Es la estructura más usada comercialmente debido a su bajo coste. Tiene el inconveniente de ser muy lento porque se tienen que utilizar más ciclos para implementar la misma instrucción. Como el bus es único, hay que usarlo tanto para depositar datos en los registros, como para cogerlos, por lo tanto es un bus bidireccional .



Se necesita obligatoriamente un registro acumulador que guarde uno de los datos de la operación (el otro entra directamente por el bus). Con la estructura de la figura una operación aritmética de dos operandos se realiza en tres ciclos de reloj.:

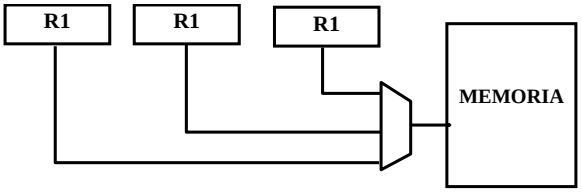
- En el primer ciclo se trae el primer operando que se debe guardar en el acumulador para no perderlo
- En el segundo ciclo se trae el segundo operando, que ataca directamente a la entrada de la ALU. Como ésta es un circuito combinacional el resultado aparece tras un retardo a la salida de la ALU. Este dato no se puede escribir en el bus porque éste está ocupado con el segundo operando, luego se debe cargar en el acumulador

- En el tercer ciclo se escribe el contenido del acumulador sobre el bus. Como el acumulador no tiene salida directa al bus, el resultado debe pasar a través de la UAL, que no realizará ninguna operación sobre el.

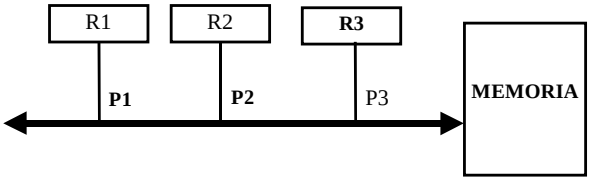
Es posible utilizar otro registro temporal a la salida de la UAL.

PUERTAS TRIESTATE (RECORDATORIO):

Se utilizan para controlar las salidas de diferentes módulos a un bus. Suponer que tenemos tres registros que envían información a la memoria. Esto se puede realizar de dos maneras diferentes. Utilizando una línea para cada registro y un multiplexor a la puerta de la memoria para seleccionar el registro cuya información quiero cargar en la memoria:



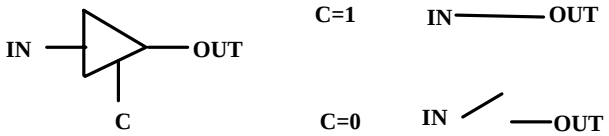
La segunda forma es utilizando una línea única a la que se conectan todos los registros



Esta última estructura tiene el problema de que los registros se pueden leer en cualquier instante de tiempo, es decir el contenido de R1 está en todo instante en el punto P1, el contenido del registro R2 esta en todo instante en P2 y el contenido de R3 está en todo instante en P3. Y como P1, P2, P3 son el mismo punto eléctrico es imposible que pueda contener tres informaciones diferentes. La solución es incluir algún tipo de barrera que me impida el paso directo del contendido de un registro al bus. Esto se consigue mediante las puerta triestate. Una puerta triestate es una puerta lógica cuya tabla de verdad es la siguiente:

C In	Out
0 0	Z
0 1	Z
1 0	0
1 1	1

Siendo C la señal de control de la puerta, *in* el valor de entrada y *out* el valor de salida. Es decir cuando la señal de control vale 1 la entrada se coloca en la salida. Cuando la señal de control vale 0 la salida es siempre de alta impedancia (Z) que es lo mismo que suponer una resistencia infinita y por lo tanto la salida está aislada de la entrada. El símbolo de una puerta triestate y su comportamiento es el siguiente:



Volviendo a la figura del bus bidireccional, las señales de control se dividen en los siguientes grupos:

Z_1, Z_2, Z_3 que controlan puertas triestate.

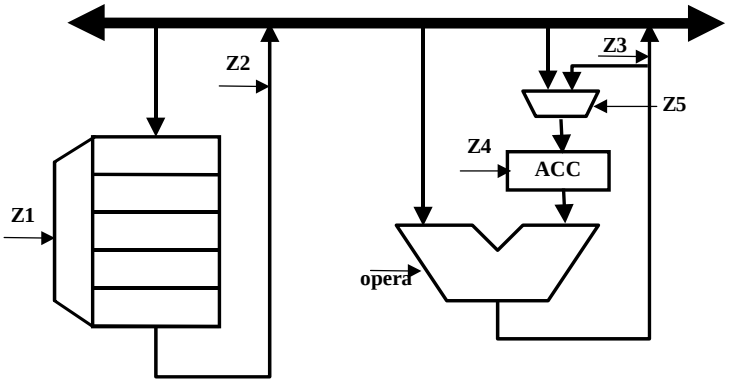
Z_5,Z_6,Z_7 que cargan los registros.

Z_4 que controla el mux

Estas señales las genera la unidad de control. El número de señales de control que se precisan para manejar registros es el doble de la cantidad de registros que existe:

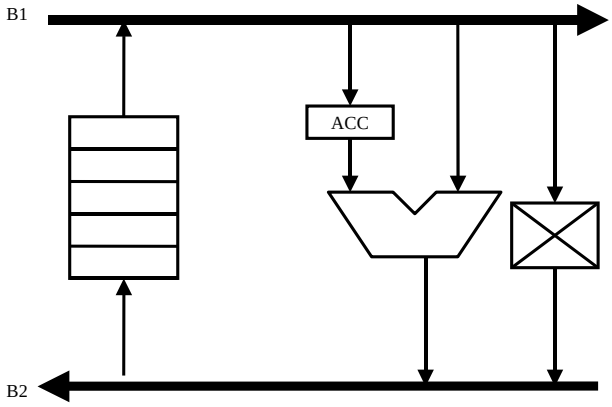
- una señal de lectura (la de la puerta triestate)
- Una señal de escritura. (la de carga del registro)

Una variante a la estructura de 1 bus, consiste en agrupar todos los registros en un banco de registros. Con ello se reduce notablemente el número de señales de control. Por ejemplo si existen 2^N registros se necesitan N señales de selección frente a las 2^N que necesitábamos y una de lectura frente a las 2^N (una de escritura optativa).



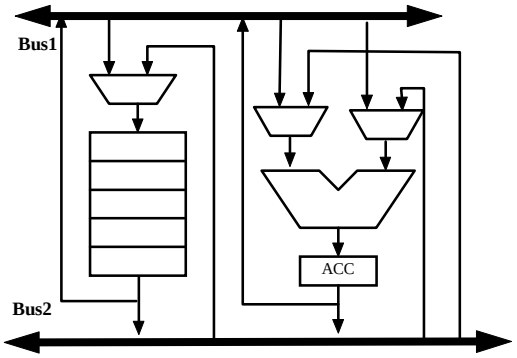
CAMINO DE DATOS SECUENCIALIZADO DE DOS BUSES

Se produce un incremento de coste pero también se reducen los ciclos necesarios para ejecutar la instrucción de operación de registro a registro, luego mejora la velocidad. Estos buses puede ser de dos tipos : los unidireccionales y los bidireccionales. En los buses unidireccionales el bus 1, lleva los datos de la memoria y el banco de registros a la UAL. Y el bus 2, lleva los resultados de la UAL a la memoria y el banco de registros. En ocasiones existe un enlace mediante el que se pueden unir el bus 1 y el bus 2.



El movimiento de información de la memoria al banco de registros se debe realizar a través de la unidad aritmética lógica o del enlace entre buses. Este enlace se puede implementar mediante un registro o mediante puertas triestate. En el primer caso se realiza el movimiento en un ciclo más para poder cargar el registro.

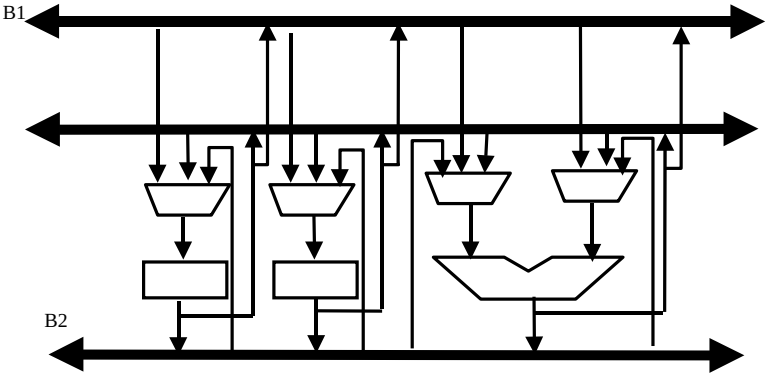
En los buses bidireccionales se produce un aumento del rendimiento y mejora de la capacidad de procesamiento pero como contrapartida encarece el producto y aumenta la complejidad de control.



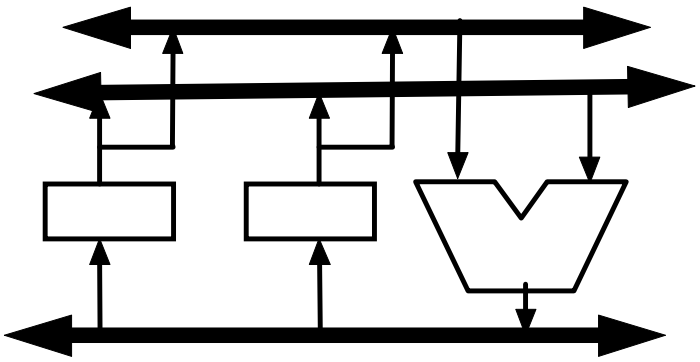
Darse cuenta que al tener dos líneas diferentes que llegan a una misma entrada se deben colocar multiplexores para decidir cual de las líneas que llegan se conecta a la entrada. En este caso el registro acumulador se utiliza debido a que como cada dato llega simultáneamente a la UAL están ocupados los dos buses, y por lo tanto necesito acumular el resultado hasta que en el siguiente ciclo de reloj los buses quedan libres. Con esta estructura reduzco los ciclos necesarios para realizar una operación de dos operandos. En el primer ciclo traigo los operandos y los proceso mediante la UAL. En el siguiente ciclo envío el resultado a su posición de memoria

CAMINO DE DATOS SECUENCIALIZADO DE 3 BUSES

Es la estructura menos usada comercialmente, dado su elevado coste. Se produce un incremento del conexionado. Solo se ve en computadoras especializadas de gran potencia de cómputo.



Existe otra posibilidad que simplifica mucho el hardware aunque resta potencia y flexibilidad al camino de datos. Y es considerar que la salida de la UAL solo puede conectarse a un bus y suponer que a cada entrada de la UAL solo puede llegar un bus, de esta manera suprimimos todos los multiplexores y un montón de líneas extra.



La principal ventaja de esta estructura es que en un único ciclo de reloj puedo ejecutar una instrucción aritmético lógica de dos operandos

6.4 DISEÑO DE LA RUTA DE DATOS

Al diseñar una máquina se debe decidir no solo como será la lógica de la máquina sino también como será su reloj. Las señales de reloj se utilizan para determinar cuando debe escribirse un elemento de estado o la memoria. Un elemento de estado puede leerse en cualquier instante

Una metodología de pulsos de reloj determina cuando pueden leerse o escribirse las señales. Es importante distinguir la temporización de la lectura, de la temporización de las escrituras porque si una señal se lee al mismo tiempo que se escribe, el dato que se lee puede ser el antiguo no el nuevo o una mezcla de ambos.

ν **METODOLOGÍA DISPARADA POR FLANCOS**

Los datos se actualizan en los flancos de reloj (o bien en el flanco de subida o en el de bajada). La entrada a un elemento combinacional debe venir de un conjunto de elementos de estado (registros o memoria). La salida de un elemento combinacional se escriben en un elemento de estado

Las entradas a un circuito combinacional son las que se han cargado en los elementos de estado en el último ciclo de reloj, las salidas del circuito combinacional son los valores que se cargan en los elementos de estado en las siguientes flanco de reloj.

La metodología disparada por flancos permite que un elemento sea leído y escrito en el mismo ciclo de reloj sin crear ningún riesgo que pueda conducir a valores de datos indeterminados. La condición que debe cumplir el sistema para implementar la metodología anterior es que el ciclo de reloj sea lo suficientemente largo como para permitir que los valores de entrada estén estables cuando se presente el flaco. Cuando no se quiere que un registro sea escrito en todos los ciclos de reloj se le debe añadir una entrada de control de capacitación de manera que sólo cuando la capacitación esté activa al llegar la señal de reloj se escriba el registro. Son estas señales de capacitación las que genera la unidad de control.

ν **CONSTRUCCIÓN DEL CAMINO DE DATOS DE UN SOLO CICLO**

Es aquel en el que se puede ejecutar una instrucción por ciclo. Esto implica que se deben de generar todos los módulos hardware que sean necesario porque ningún recurso del camino de datos puede utilizarse más de una vez por instrucción. Por ejemplo para implementar la instrucción add reg1,reg2 en un solo ciclo de reloj sabemos que si el incremento del PC se realiza con un sumador, el camino de datos debe tener dos sumadores, uno para incrementar el PC y otro para realizar la suma En este mismo ejemplo se ve que el banco de registros debe tener dos puertos de salida. Tiene el inconveniente del coste en HW y la ventaja del incremento del rendimiento CPI=1 (CPI= Ciclo Por Instrucción), aunque esto último no es del todo cierto porque dado que el ciclo de reloj es fijo e igual al tiempo que tarda en ejecutarse la instrucción más larga, esto puede producir una degradación del rendimiento, puesto que no todas las instrucciones tardan lo mismo.

La implementación de un solo ciclo viola nuestro principio clave de diseño de hacer rápido el caso común, porque trabajas siempre al ritmo de la instrucción más lenta. Este problema se pueden eliminar mediante el multiciclos.

ν **DISEÑO MULTICICLO**

Permite que una unidad funcional sea usada tantas veces como sea necesario para implementar una operación, siempre que se utilice en ciclos diferentes. Esto reduce el HW. Siguiendo el ejemplo anterior si tuviéramos sólo un sumador, necesitaríamos un ciclo para incrementar el contador y otro para sumar los datos. Además si solo se tuviera un puerto de salida se necesitarían dos ciclos de reloj para leer los datos

6.4.1 ESPECIFICACIONES DE ARQUITECTURA

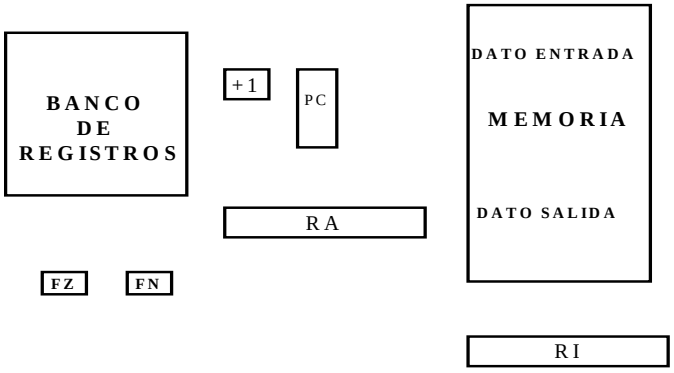
6.4.1.1 Especificaciones generales

Nota: Esta arquitectura ha sido propuesta por los profesores del Departamento de Arquitectura de Computadores y Automática, D.Francisco Tirado y D.Román Hermida.

A continuación se describe la arquitectura y estructura del computador que se diseña en la asignatura.

- Es una arquitectura de tipo Von Neumann
- Tipos de datos enteros de 16 bits codificados en complemento a dos
- Memoria RAM de 256 palabras de 16 bits cada una que puede almacenar tanto datos como instrucciones
- Bus de direcciones de 8 bits
- 2 buses de datos de 16 bits (uno de entrada a la memoria y uno de salida)
- Secuenciamiento implícito
- Registro de instrucciones de 16 bits
- Contador de programa de 8 bits
- Banco de 8 registros de 16 bits nombrados de R0 a R7, siendo el R_0 es un registro especial que contiene siempre el 0. Además tiene un puerto de entrada y un puerto de salida.
- Modelo de ejecución registro - registro
- Dos indicadores de condición: cero y signo
- Operaciones de 3 operandos funte1, fuente2, destino

Del estudio de estas especificaciones llegamos a la conclusión de que necesitamos como poco el siguiente HW



Donde todos los módulos han sido indicados explícitamente en las especificaciones salvo RA y el incrementador +1. La existencia del registro acumulador se debe a que el modelo de ejecución es registro-registro, es decir los dos operandos están en el banco de registros y como el banco de registros solo tiene un puerto de salida el primer dato leído se debe almacenar en alguna parte.

La existencia del incrementador se debe a que, como se ha indicado, el secuenciamiento es implícito por lo tanto el PC debe autoincrementarse. Este incrementador se puede implementar como un módulo independiente o se puede utilizar la UAL. Esta decisión es una decisión de diseño. En nuestro caso lo supondremos un modulo independiente. Esto añade hw pero acelera la ejecución puesto que permite incrementos del pc y operaciones aritmético lógicas simultáneamente.

6.4.1.2 Repertorio de instrucciones:

Tiene tres tipos de instrucciones diferentes:

- Aritmético lógicas con operando inmediato y sin operando inmediato
- Transferencia de datos
- Salto

Para cada instrucción vamos a estudiar el formato que propone la arquitectura, los pasos que habría que dar y el hardware extra que habría que utilizar para implementarla

OPERACIONES ARITMÉTICAS CON DOS OPERANDOS EN REGISTROS:

- **ADD Rf1, Rf2, Rd**

Para todas las instrucciones se observa que los dos operadores Rf1 y Rf2 provienen siempre del banco de registros y que el resultado Rd va siempre al banco de registros. Como el banco de registros solo tiene un bus de salida, esto quiere decir que en algún lado se debe guardar el primer operando. Esta es la razón de la existencia del registro auxiliar Ra. Este registro no viene definido en la arquitectura y aparece como consecuencia del estudio de la estructura necesaria para implementar la arquitectura.

Según esto la instrucción se tendría que realizar en dos ciclos de reloj en el primero se lee el primer operando, y en el segundo se lee el segundo operando se realiza la operación aritmético lógica y se carga el resultado en el registro destino. Observar que en este segundo paso se lee un registro y se escribe en otro. Esto es posible porque los registros se pueden leer en cualquier instante de tiempo, luego desde el comienzo de este segundo ciclo el dato de salida es correcto y es al finalizar el ciclo (en el primer flanco de subida) cuando se carga el dato en Rd.

Si ésta fuera la única operación, el ciclo debía durar lo suficiente para que el resultado correcto obtenido de la operación se encontrase a la entrada del registro Rd. Por último, se debe tener en cuenta que todas las operaciones aritméticas pueden modificar los flags de cero y signo. Los dos pasos en que se descompone la operación son:

Ra←Rf1

Rd←Ra+Rf2

Los razonamiento para el resto de las instrucciones son similares.

- **SUB Rf1, Rf2,Rd**

  Ra←Rf1

  Rd←Ra-Rf2

- **ASR Rf,Rd**

  Rd←deplaza(Rf2)

  Esta instrucción es algo diferente puesto que no necesita el registro auxiliar

- **AND Rf1, Rf2,Rd**

  Ra←Rf1

  Rd←Ra and Rf2

El formato de las instrucciones aritméticas con dos fuentes y un destino que propone la arquitectura es el siguiente:

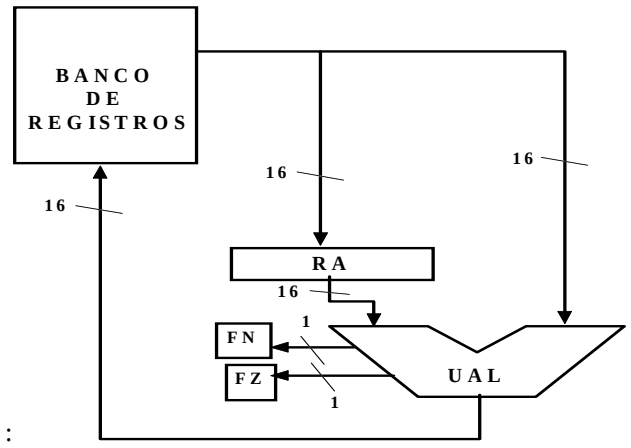
O	d	f1	f2	0	P

Siendo:

- CO el código de operación - identifica el tipo de instrucción. Utiliza dos bits
  00 load
  01 store
  10 salto
  11 aritmético - lógicas
- Rd Registro destino. 3 bits RI<13:11>
- Rf1. Registro fuente. 3 bits. RI<10:8>
- Rf2. Registro fuente. 3 bits. RI<7:5>
- OP identifica la operación aritmético - lógica concreta que se quiere realizar
  100 ADD
  101 SUB
  110 ASR
  111 AND

El hecho de que el código de operación, CO esté claramente diferenciado de la operación aritmética (OP) que se va a realizar es muy importante para el diseño de la unidad de control puesto que la simplifica mucho. De esta manera, la unidad de control solo tiene que interpretar el hecho de que se va a realizar una operación aritmética, sin preocuparse de ver cual es la que se ejecuta. Los bits OP del registro de instrucciones pueden atacar directamente a la UAL sin pasar por la unidad de control

En caso contrario el código de operación tendría más de 2 bits - puesto que cada operación aritmética tendría su propio código - y la unidad de control tendría bastante más circuitería. Teniendo en cuenta el Registro acumulador ya comentado y los flag de numero cero y número negativo el camino de datos correspondiente a estas instrucciones es el siguiente:



OPERACIONES ARITMÉTICAS CON OPERANDO INMEDIATO

ν **ADDI Rf, #n,Rd**

En el caso de estas operaciones aritméticas habrá que tener en cuenta que uno de los operandos utiliza un modo de direccionamiento inmediato, es decir el número debe venir incluido directamente en la instrucción.

En el formato propuesto por la arquitectura, que se explica más adelante, se indica que el número de bits para el operando inmediato es de 5, y la unidad aritmético lógica que se utiliza es de 16 bits, por lo tanto se debe usar un operador extensión de signo para poder operar con este inmediato.

¿Haría falta con estas operaciones registro auxiliar? La respuesta es no porque caminos diferentes, alimentan entradas a la UAL diferentes. Pero lo cierto es que para tratar las operaciones con inmediato de manera similar a las operaciones con dos operandos también suponemos que se realizan en dos pasos. Esto simplifica el diseño de la U.Control y ahorra hardware, puesto que en caso contrario tendríamos que incluir un multiplexor entre el registro Ra y la entrada de la ALU.

Ra←Rf1

Rd←Ra + exte(#n)

ν **SUBI Rf,#n,Rd**

Ra←Rf1

Rd←Ra - exte(#n)

El formato propuesto por la arquitectura es:

CO	Rd	Rf1	INMEDIATO	OP

- Rd Registro destino. RI<13:11>. 3 bits

- Rf1. Registro fuente. RI<10:8>. 3 bits
- INMEDIATO. Numero inmediato con el que se opera. RI<7:3> 5 bits. Como se utiliza en una UAL de 16 bits habrá que hacer una extensión de signo con él.
- OP

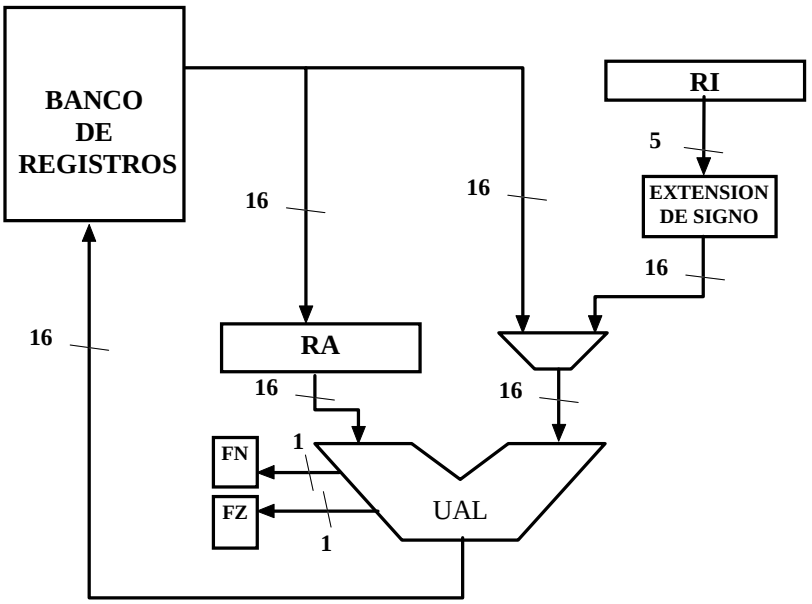
  000 suma

  001 resta

Observando esta instrucciones con operando inmediato llegamos a la conclusión que el operador de la izquierda Rf1 proviene siempre del banco de registros mientras que el de la derecha viene del registro de instrucciones.

Observando el camino estudiado antes vemos que a la entrada derecha de la UAL pueden llegar datos procedentes del banco de registros y procedentes de RI luego necesitaremos un multiplexor para seleccionar una entrada u otra.

Por último recordar que el dato inmediato solo tiene 5 bits por lo tanto se debe realizar una extensión de signo para que pueda entrara a la UAL que tiene 16 bits. El camino de datos resultante incluyendo las modificaciones es:



INSTRUCCIONES DE ACCESO A MEMORIA

LOAD A(Ri),Rd

Utilizan estas instrucciones un modo de direccionamiento a registro base**.** Es decir la dirección de acceso a memoria se obtiene sumando el contenido del registro y el desplazamiento incluido en la propia instrucción. Aunque para implementar esta instrucción no hace falta un registro auxiliar de direcciones R@, lo hemos utilizado. Mas adelante veremos el motivo. La instrucción se realiza en dos pasos

R@←<A+Ri>

Rd←M<R@>

este dato llega al banco de registros a través de la UAL

STORE Rf, A(Ri)

R@←<A+Ri>

M<R@>←Rf

Como se puede ver en esta instrucción salen dos datos del banco de registros, por un lado se utiliza el Ri para obtener la dirección de memoria y por otro lado el Rf es el dato que se quiere cargar en la memoria. Como el banco de registros solo tiene un bus de salida se tendrá que realizar la operación en dos ciclos y será

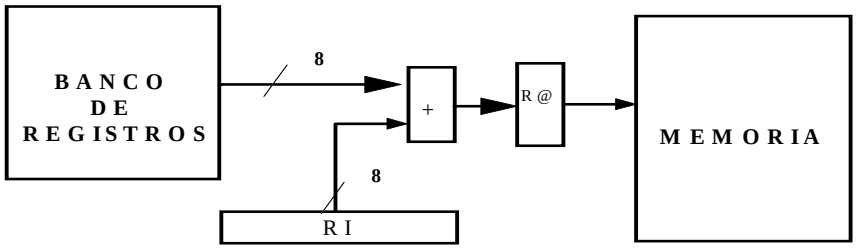
obligatorio el uso del registro R@. La dirbase (A) es la dirección base incluida en la instrucción y el desplazamiento el contenido de un registro

CO	Rx	Ri	dirbase

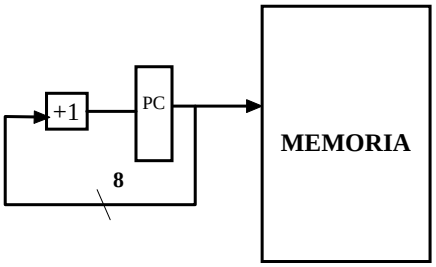
Rx Registro destino o fuente RI<13:11>

Ri Registro índice RI<10:8>

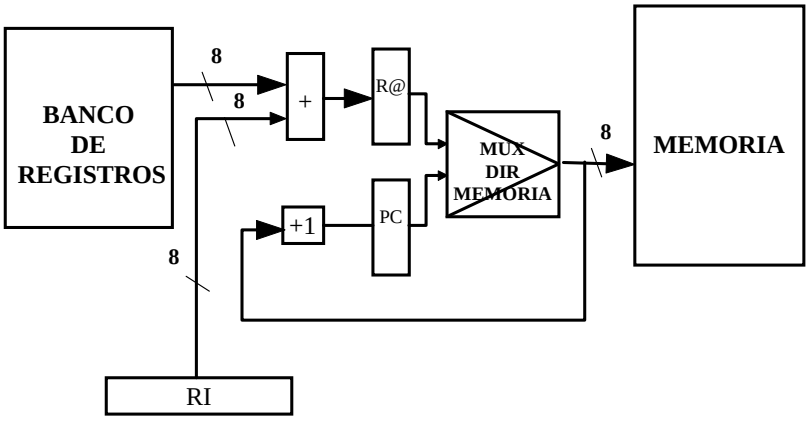
Además para calcular la dirección de acceso a memoria se necesita un sumador de 8 bits, que se va a implementar como un módulo de HW independiente de la UAL . El resultado viene dado en el siguiente esquema:



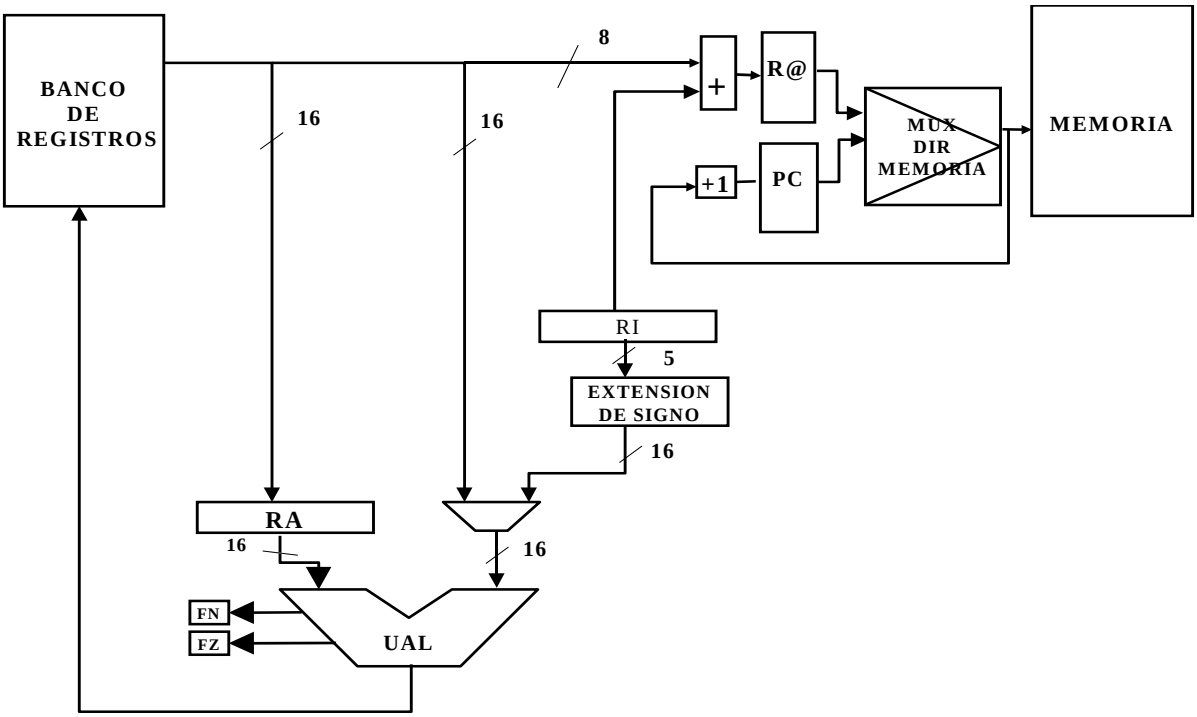
Atención: Darse cuenta que los registros del banco son de 16 bits y que en el camino de datos de la figura anterior solo se utilizan 8 de ellos. Estro se debe a que las direcciones de memoria solo tienen 8 bits. Es una decisión del diseñador indicar en que bits del registro se guarda la dirección. Lo mas común es suponer que se guardan el los 8 bits más significativos. Recordando que el direccionamiento era implícito el esquema que lo implementa es:



Es decir de los dos últimos esquemas se ve que existen dos formas diferentes de proporcionar una dirección a la memoria. Desde el contador de programa PC y desde el registro auxiliar de direcciones R@. Por lo tanto se necesitara un multiplexor que ayude a seleccionar cual de las dos direcciones es la que direcciona la memoria. Juntándolo todo queda:



Notar que la realimentación para incrementar el contador de programa se realiza desde la salida del multiplexor, en lugar de hacerlo desde la salida del PC. El porque se explica más adelante. Antes de continuar con el estudio del camino de dato vamos a unir en un solo esquema los dos subcaminos que hemos encontrado hasta el momento:

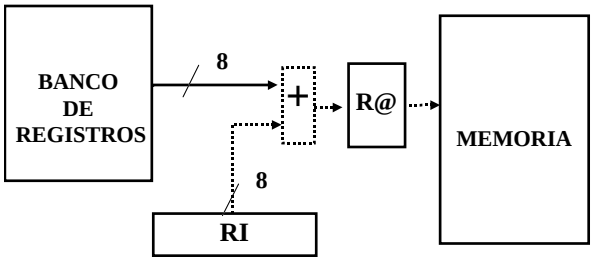


Instrucciones de salto

Existen siete instrucciones de salto:

BL bifurcar si es menor	BG bifurcar si es mayor
BEQ bifurcar si es igual	BNE bifurcar si es distinto
BLE menor o igual	BGE mayor o igual
BR salto incondicional	

Para ahorra hardware suponemos que utilizamos la unidad secuenciadora estudiada en las instrucciones de movimiento de datos. En éstas para encontrar la dirección de acceso a la memoria se sumaba el contenido de un registro y el desplazamiento incluido en el registro de instrucciones y se cargaba el resultado en R@. Para las instrucciones la dirección viene incluida directamente en el RI. El camino que debe seguir esta dirección hasta llegar a la entrada del bus de direcciones de la memoria está marcado en la siguiente figura en discontinúo:



Se ve que si queremos utilizar la estructura que ya existía, sin introducir ningún cambio la dirección debe pasar obligatoriamente por un sumador. Como la dirección que incluye el registro de instrucciones es la correcta la única manera de realizar esto correctamente es sumar un cero a esa dirección:

R@:=RI.dirección+0

Esto a su vez se puede hacer de varias formas. Como tenemos la ligadura de mantener la estructura, la única solución es cargar un cero en alguno de los registros. La solución finalmente adoptada es suponer un registro especial que siempre tendrá almacenado un cero. Este registro es el R_0. Los pasos de ejecución de todas ellas son idénticos: si condición es cierta

R@←RI<7:0>+ R_0

PC←R@+1

RI←M[R@]

si no

PC←PC+1

Debido a que el PC se carga con el contenido de R@ + 1, la realimentación del PC se realiza desde la salida del mux en lugar de realizarlo desde la salida del PC

FORMATO

CO	cond	000	dirección

- CO =10
- COND es la condición respecto a la que se hace el salto. RI<13:11>.

  000 incondicional
  001 igual
  010 menor
  011 menor o igual
  101 distinto
  110 si mayor o igual
  111 si mayor.
- Dirección es la dirección a la que se bifurca RI<7:0>

Es importante darse cuenta que todas las instrucciones están condicionadas por el hecho de que el conjunto de registros sólo tenga un puerto de salida, lo que impide que tanto las operaciones aritmético-lógicas como las de bifurcación y movimiento de datos se implementen en dos pasos. Observando las instrucciones de salto se puede ver dos nuevos módulos de hardware:

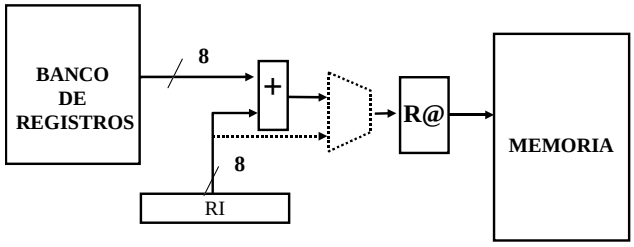
Un módulo generador de la nueva dirección de salto

Un módulo evaluador de la condición

Modulo generador de la nueva dirección de salto

Tal como se ha definido, la dirección de salto viene dada por la suma del registro R_0 del banco de registros, que siempre contiene un cero, y parte del contenido del registro de instrucciones, mas exactamente por los 8 bits menos significativos. Pero si nos fijamos en las instrucciones de load y store las direcciones base contenida en el registro de instrucciones también ocupa los 8 bits menos significativos. Es decir, no hace falta añadir nuevo HW al circuito secuenciador para implementar estas instrucciones.

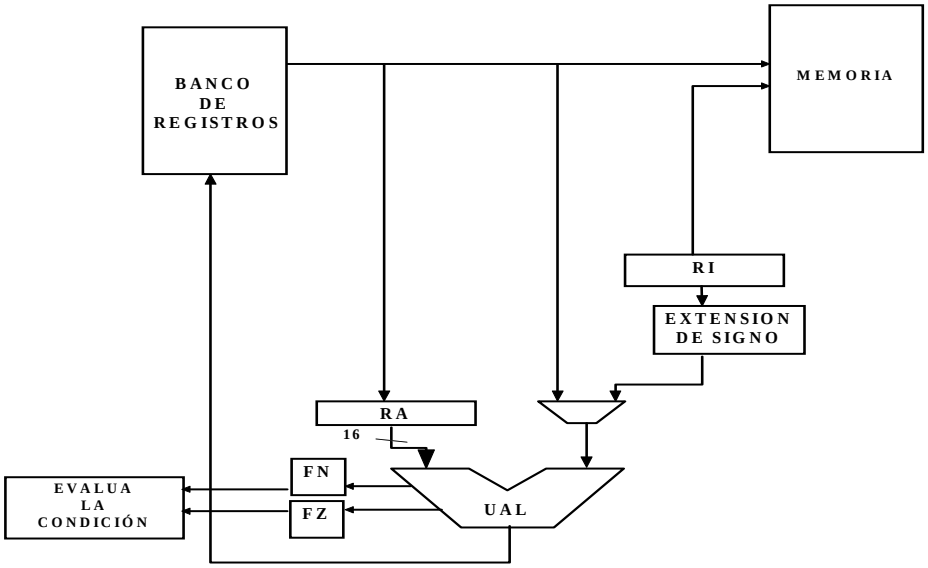
Esta solución no es la única y es una decisión del diseñador de estructura puesto que no está definida en la arquitectura. Otra posible solución es añadir un multiplexor de direcciones como muestra la parte punteada de la siguiente figura:



Modulo evaluador

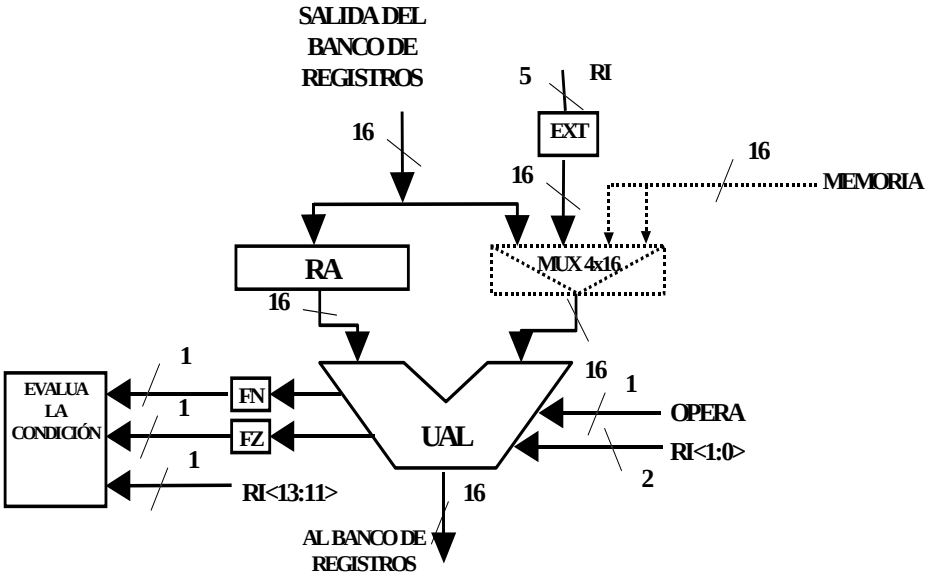
Es una decisión de diseño incluir el modulo evaluador en el camino de datos. Este modulo podría estar perfectamente incluido dentro de la unidad de control. Es decir a la unidad de control le llegaría los contenidos del flag cero y del flag de número negativo (FZ y FN) y la parte del código de instrucciones que indica que la instrucción en ejecución es una instrucción de bifurcación condicional y es la UC la encargada de evaluar esta información.

Pero se debe tener en cuenta que el diseño de la unidad de control ya es bastante complejo por si misma como para añadirle dificultades extras. En lugar de hacer esto, se va a incluir en el camino de datos un módulo evaluador de condición cuya misión va a ser decirle a la unidad de control si debe genera las señales de salto. El esquema quedaría



¿Falta hardware por incluir en este esquema? Nos quedan por modificar un módulo y por incluir otro. Cuando estudiamos las instrucciones de movimiento de datos desde la memoria hasta el banco de registros, modificamos el modulo secuenciador para poder generar la dirección de acceso a memoria, pero se nos olvidó tratar otro punto.

Se dijo en su momento que el movimiento de datos se hacía a través de la Unidad Aritmético - Lógica por lo tanto a la UAL le llegan datos que no habíamos tenido en cuenta. Esto quiere decir que tendrá que aparecer una línea de entrada a la UAL proveniente de la memoria y por lo tanto se debe ampliar el número de entradas al mux. Las variaciones introducidas aparecen en la siguiente figura punteadas:



Otra decisión del camino de datos que va a afectar a la unidad de control, y que se debe tomar ahora, es de donde van a partir las señales que controlan el mux que selecciona la procedencia del segundo operando. Existen dos soluciones: que las señales provengan de la unidad de control, o que provengan del RI. Vamos a ver si este segundo caso es posible. ¿Existe en el RI información para determinar si el segundo operando viene de un lado o de otro? Mediante el CO puedo determinar si el segundo operando viene de la memoria (instrucciones de movimiento de datos) o si viene de un registro (instrucciones aritmético lógica)

CO= 00 movimiento de datos

CO=11 operación aritmético lógica

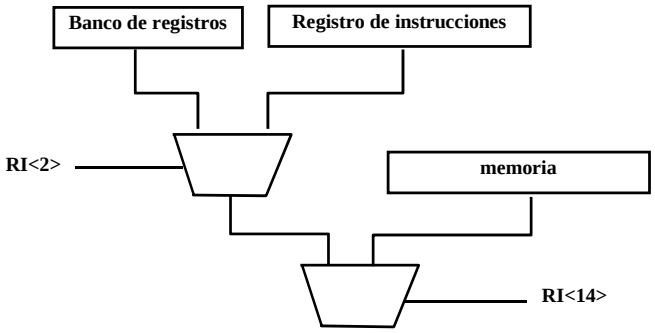
Luego RI<14>=C0<0> se puede utilizar como señal de control

Todavía existe un problema. Cuándo la operación es aritmética el segundo operando puede venir del banco de registros (dos operandos en banco de registros) o del registro de instrucciones (segundo operando inmediato) ¿Tengo suficiente información en el RI para seleccionar estas procedencias?. La operación aritmética a realizar se codifica en el campo OP:

OP 000 suma con inmediato

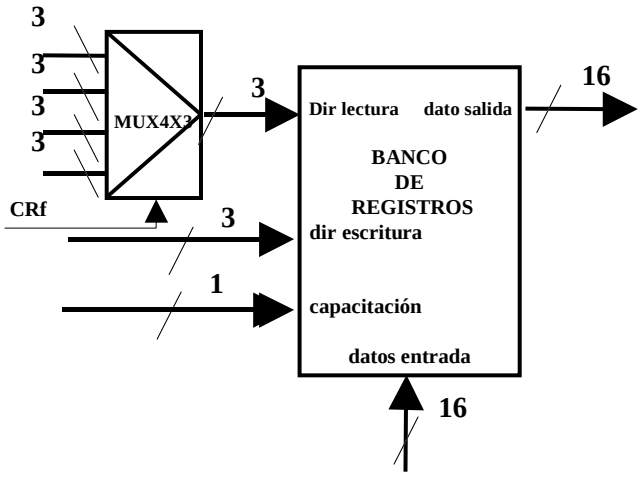
OP 100 suma normal

Luego C0<2> =RI<2> sirve para controlar esta elección esto quedaría de la siguiente manera:



DIRECCIONAMIENTO DEL BANCO DE REGISTROS

Por otro lado estamos viendo a lo largo de todo el estudio que el banco de registros se puede direccionar desde diversa posiciones del RI, es decir llegan varias líneas de direcciones a una sola entrada por lo tanto habrá que colocar un mux que seleccione la dirección adecuada:

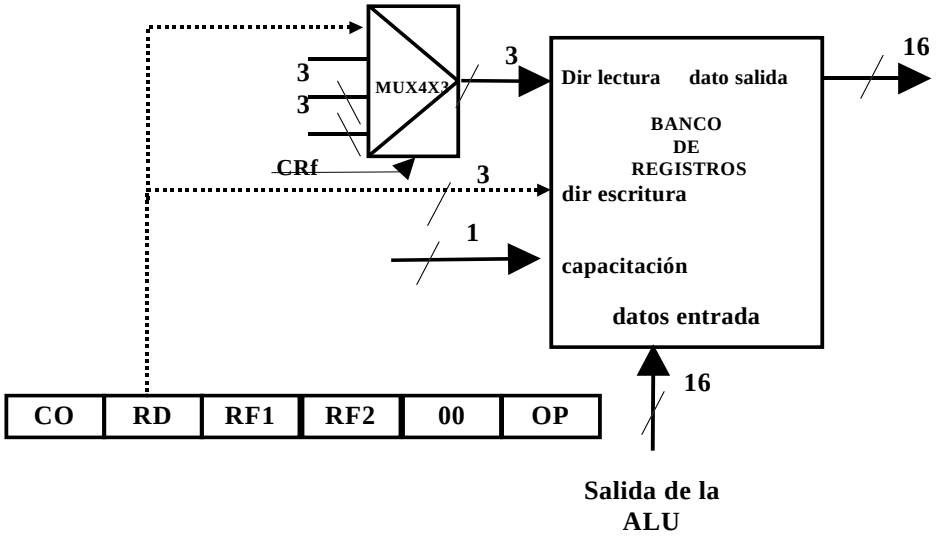


Salida de la ALU

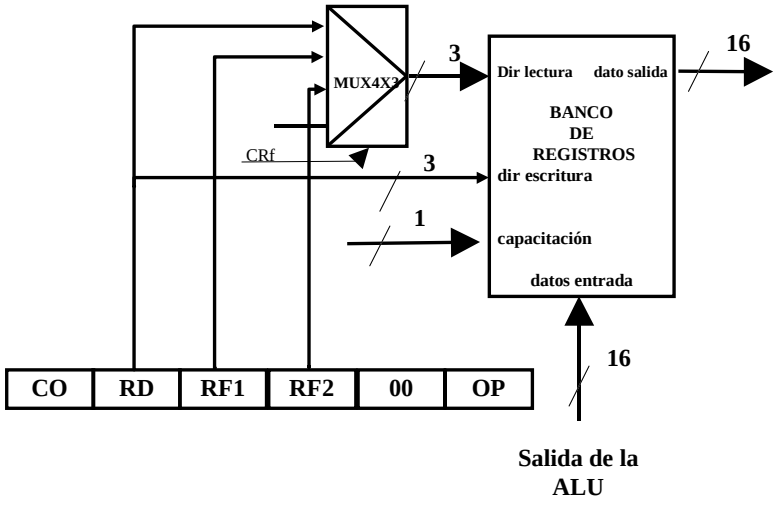
Saber que bits del registro de instrucciones tienen que llegar a cada entrada del multiplexor es una decisión que se debe tomar ahora puesto que afecta al camino de datos. Para estudiarlo recordaremos cual es el formato de las instrucciones:

15 14	13 11	10 8	7 5		
CO	RD	RF1	RF2	00	OP
CO	RD	RF1	INMED	IATO	OP
CO	RD/RF	RI	DIR	BASE	

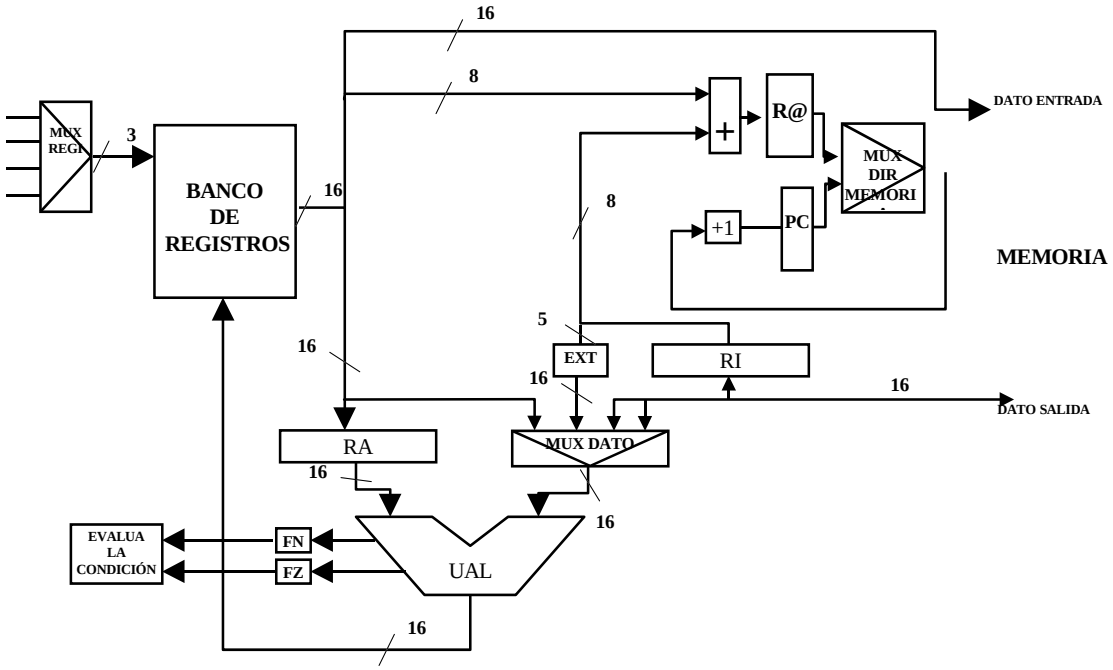
De estos formatos sólo nos interesan los campos que direccionan al banco de registros. El campo RI<13:11> direcciona en el formato 1 y 2 un registro destino. En el formato tres depende de si la instrucción es load o store. En el caso store actúa como fuente y en el load como destino. Con lo anterior se deduce ese campo deberá ir por un lado directamente conectado al bus de direcciones de escritura del banco de registros y por otro lado conectado a una de las entradas del MUX como se ve a continuación:



Volviendo a los formatos se ve que el campo RI<10:8>en los formatos 1 y 2 es el campo del primer operando que actúa como registro fuente. Para el tercer formato actúa como RI que a la postre también es un registro fuente. Luego ese campo debe ir conectado a otra entrada del mux. Por último RI<7:5> es siempre la dirección del segundo operando luego debe ir conectado al mux quedando el esquema como se ve a continuación:

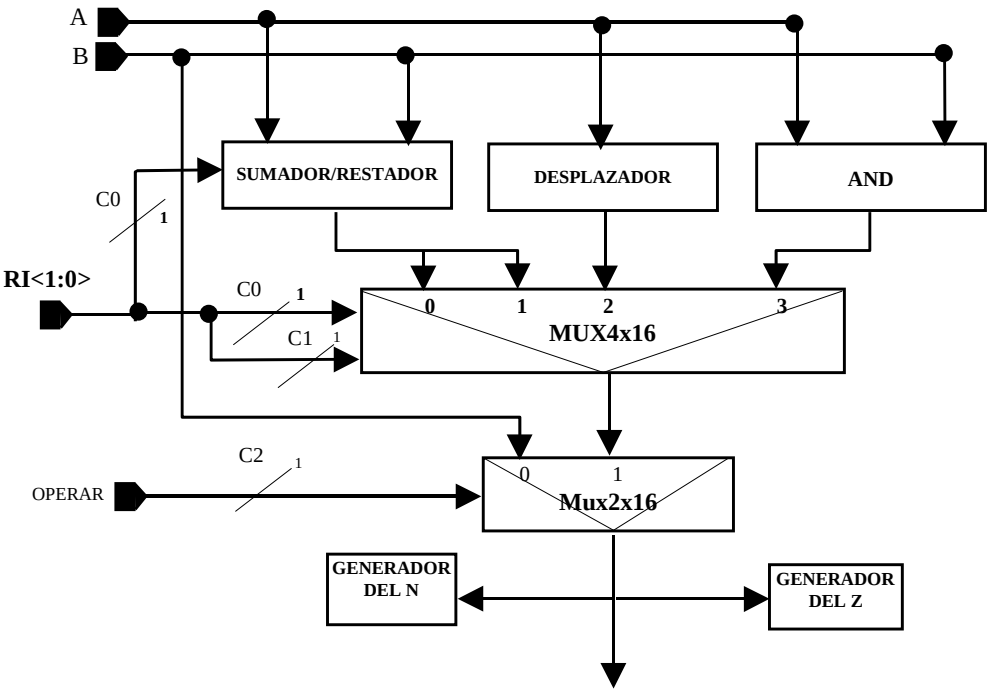


Darse cuenta que la última entrada del mux no se utiliza. La figura que aparece a continuación contiene el camino de datos completo del procesador.



6.4.2 LA UNIDAD ARITMÉTICO LÓGICA (ALU)

A continuación aparece el esquemático de la unidad aritmético lógica y de los registros y unidades funcionales que le rodean. Darse cuenta que de las tres señales que controlan la unidad solo la señal OPERAR proviene de la unidad de control y las otras dos (RI<1:0>) provienen del registro de instrucciones directamente.



Las señales de control de la unidad aritmético lógica son tres C0, C1 y C2:

C2	C1	C0	función
0	X	X	PASA B
1	0	0	A+B
1	0	1	A-B
1	1	0	DESP A
1	1	1	AND

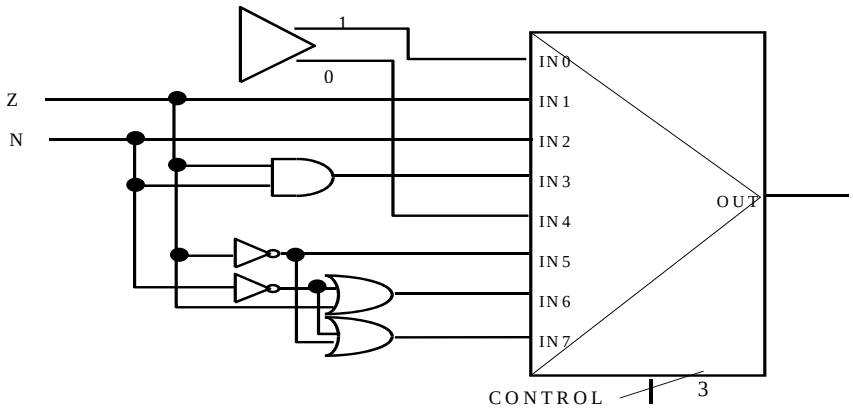
C1 y C0 vienen directamente del registro de instrucciones y C2 viene de la unidad de control. Esto simplifica el diseño del control del computador.

6.4.2.1 Evaluador condición de salto

Es un módulo combinacional que recibe como entrada el contenido del flag Z, el contenido del flag N y los bits del registro de instrucciones que indican que tipo de salto queremos. La salida es de un solo bit e indica si se cumple la condición de salto o no se cumple. Esta salida es una de las entradas a la unidad de control. La razón de realizar esta evaluación fuera de la unidad de control es que simplifica el diseño de la misma. La tabla de condiciones es la siguiente:

TIPO DE SALTO	CODIGO	CONDICION
BR salto incondicional	000	1
BEQ si igual	001	Z=1
BL si menor	010	N=1
BLE menor o igual	011	N=1 o z=1
BNE si distinto	101	Z=0
BGE si mayor o igual	110	N=0 o Z=1
BG si mayor	111	N=0 y Z=0

Esta tabla se puede implementar muy fácilmente mediante el multiplexor MUX8x1 que ya hemos estudiado. El esquemático resultante es el siguiente:


1
0
Z
N
IN 0
IN 1
IN 2
IN 3
IN 4
IN 5
IN 6
IN 7
OUT
CONTROL
3

7 LA UNIDAD ARITMÉTICO LÓGICA

Es la unidad encargada de tratar los datos ejecutando las operaciones requeridas en la instrucción en curso. La unidad de control es la que se encarga de mandarle los datos e indicarle la operación a realizar. La mayoría de los computadores basan su unidad aritmético lógica en un simple sumador-restador. Por lo general la forma de implementar operaciones más complejas es descomponerlas en pasos elementales de sumas y restas que se ejecutan rápidamente.

7.4 REPRESENTACIÓN DE ENTEROS

Se van a estudiar cuatro representaciones

- Binario puro
- Magnitud y signo
- Complemento a uno
- Completo a dos

La primera de ellas es para enteros sin signo y el resto para enteros con signo

7.4.1 BINARIO PURO

Es un sistema posicional de Base dos. Sabemos que un número en representación decimal se puede descomponer de la siguiente forma:

$N^o = 2^{n-1}b_{n-1} + 2^{n-2}b_{n-2} + ... + 2^2b_2 + 2^1b_1 + 2$

Pudiendo valer b=0 , o b=1. La representación binaria de este número es:

$b_{n-1}, b_{n-2}, ...b_2, b_1, b_0$

- El Rango de representación con N bits es $[0, 2^N-1]$
- Resolución $\rightarrow$ 1 unidad.

ν **Desventajas:** El resultado de una operación puede necesitar n+1 bits, esto sucede cuando aparece un acarreo al bit, a esto se le llama desbordamiento. Además **n**o puede representar números negativos por lo tanto antes de restar A-B se debe comprobar que A≥B. La resta incorrecta genera bits de desbordamiento. Por último el producto de dos números de N bits puede necesitar 2N bits.

ν **Ejemplo** de representación binaria pura con tres bits:

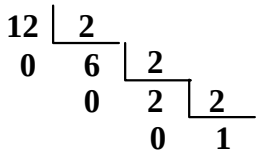
000	0	$0\cdot2^2+0\cdot2^1+0\cdot2^0$
001	1	$0\cdot2^2+0\cdot2^1+1\cdot2^0$
010	2	$0\cdot2^2+1\cdot2^1+0\cdot2^0$
011	3	$0\cdot2^2+1\cdot2^1+1\cdot2^0$
100	4	$1\cdot2^2+0\cdot2^1+0\cdot2^0$
101	5	$1\cdot2^2+0\cdot2^1+1\cdot2^0$
110	6	$1\cdot2^2+1\cdot2^1+0\cdot2^0$
111	7	$1\cdot2^2+1\cdot2^1+1\cdot2^0$

Rango $[0,2^3-1] = [0,7]$ es decir el resultado de operaciones aritméticas con binarios de 3 bits no puede superar el 7.

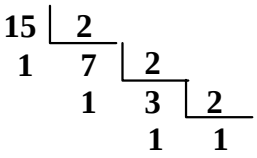
Atención en el caso de los binarios puros el acarreo y el desbordamiento en la suma coinciden pero esto no es lo normal. De manera general se puede afirmar que no siempre que hay acarreo existe desbordamiento.

La conversión de números decimales a números binarios puros se consigue dividiendo sucesivamente por 2 los restos parciales, hasta que el resto es menor de 2

ejemplos:



$$12_{10}=1100_2$$



$$15_{10}=1111_2$$

7.4.2 MAGNITUD Y SIGNO:

Se utiliza para representar números enteros con signo, por l tanto tiene Bit de signo, que es siempre el más significativo. De manera que:

- Nº positivo. Bit de signo = 0
- Nº negativo. Bit de signo = 1

El rango de esta representación es Rango para N bits [- (2^{N-1}- 1), 2^{N-1}1] y su resolución = 1.Este tipo de representación tiene la ventaja de que las operaciones de multiplicar y dividir se tratan sin dificultad: Se operan por un lado los N-1 bits de las magnitudes y por el otro los signos.

Por otro lado, también derivadas de esta representación aparecen varios inconvenientes como que el cero tenga dos representaciones. Esta ambigüedad del complica la detección de números negativos (no es suficiente analizar el bit más significativo). También aparecen problemas con las sumas y restas puesto que debe analizar previamente los signos de los operandos para saber si se debe aplicar una suma o una resta. Como en todas las codificaciones existe probabilidad de desbordamiento La operación de suma o resta depende de los operandos. A continuación aparece un ejemplo de representación.

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6

0111	7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

7.4.3 COMPLEMENTO A DOS

Representa números enteros positivos y negativos. Se caracteriza por que el bit de signo no es independiente de los de magnitud, como en el caso anterior, sino que forman un todo solidario: las operaciones aritméticas se realizan con los N bits. Los números positivos se caracterizan por tener el bit más significativo a =0 y se representan como en binario puro. Como uno de los bits es de signo el rango total es [0, 2^{N-1}-1].

En los números negativos el bit más significativo a uno. Su representan se obtiene aplicando la operación de C'2 al positivo. Esta operación consiste en restar a 2^n la magnitud del número a representar. Suponiendo que el número lo estamos representando con n bits esto quiere decir que para representar 2^n se necesitan n+1 bits. Por ejemplo para n=4 2^4 es 10000, y el complemento es 2^n-A = -A. El rango de esta representación viene dado por Rango [-2^{n-1},0), y el rango total viene dado por [-2^{n-1}, 2^{n-1}-1]. En esta representación sólo existe un 0.

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Su más importante ventaja es que se puede operar sin tener en cuenta el signo del operando. Vamos a demostrar que no es necesario hacer restas en Complemento a dos porque cualquier resta se puede convertir en suma. Para ello se estudian los siguientes casos:

1. **Cambio de signo:**

Si es positivo: A$\rightarrow$ -A = 2^n - A.

3=0011

-3= 2^4-3= 10000-0011=1101

Si es negativo: 2^n - A $\rightarrow$ A = 2^n - (2^n - A) = A

-2=1110

2=2^4-(-2)= 10000-1110=0010

2. **Suma de dos cantidades negativas:**

De hacer la suma directamente se obtiene la siguiente expresión

$(2^n - A) + (2^n - C) = 2^n + (2^n - A - C)$

La expresión que se busca es buscada es $2^n - A - C$, si se observa las expresiones sólo difieren en el término 2^n que es un bit de posición que ocuparía la posición n+1, y que por lo tanto aparece como un acarreo C_{n-1}. Basta con ignorarlo para que el resultado sea correcto.

En el caos en que $A + C > 2^{n-1}$, habrá desbordamiento.Ejemplo:

```
  1110
+1101
 11011
```

Este acarreo se debe ignorar.

3. Suma de un cantidad A positiva y una C negativa:

- **si A< C e**l resultado deberá ser negativo , $A + 2^n - C = 2^n - (C-A)$, luego se obtiene el resultado directamente. Ejemplo:

```
 1100
+0010
 1110
```

- **si A ≥ C, el r**esultado debe ser positivo $A + (2^n - C)$. Ahora bien, 2^n más una cantidad positiva no se puede representar con n bits: aparece un bit n+1; luego ignorando ese bit se obtiene la representación correcta. Ejemplo

```
  0101
+1101
 10010
```

Se observa en el ejemplo que aparece un bit que se debe ignorar.

ν ¿Cómo hacer rápidamente el complemento? Complemento binario + 1

Ejemplo:

1100→ 0011+0001=0100

0011→1100+0001=1101

ν **¿Cómo se realiza la extensión de signo?**

- Si el número es positivo, se añaden 0's a la izquierda. Ejemplo 0011→00000011
- Si el número es negativo, se añaden As a la izquierda. Ejemplo 1100→11111100

En resumen las características del complemento a dos son:

- Se realizan las sumas sin tener en cuenta los signos de los operandos.
- No confundir acarreo con desbordamiento.
- Se complica la multiplicación.
- El complemento (o cambio de signo) exige una operación de suma
- Rango de representación es asimétrico: $[-2^{n-1}, 2^{n-1}-1]$
- Representación del 0 es única.
- Resolución es 1.

7.4.4 COMPLEMENTO A UNO (C'1):

Se utiliza para representar enteros con signo. El bit de signo el más significativo determina si el número es positivo o negativo en función de los siguiente valores:

- **0 positivo**
- **1 negativo**

El número negativo se obtiene haciendo 2^n - 1 - A. La negación se consigue aplicando directamente el complemento lógico.

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-7
1001	-6
1010	-5
1011	-4
1100	-3
1101	-2
1110	-1
1111	-0

Vamos a ver que ocurre cuando se realiza diferentes operaciones. Empezamos por la suma de dos cantidades negativas: (2^n - 1 - A) + (2^n - 1 -C)= 2^n - 1 - (A + C) + 2^n - 1 - (A+C). El resultado que buscamos es 2^n - 1 - (A + C). Comparando con el que hemos obtenido vemos que da un bit de acarreo superior 2^n y un resultado una unidad inferior a lo deseado (-1). La solución es reciclar el bit de acarreo, sumándoselo al resultado de la suma.Ejemplo: 1101 +1101= *1*1010; con cuatro bits el resultado es -5 en lugar de -4 y además aparece un acarreo en el bit 5. La solución es sumar el acarreo 1010+*1*= 1011.

ν **Suma de un positivo A y un negativo C**

si (A<C): el resultado de la suma es A + (2^n -1 - C). El resultado debe ser negativo y se genera directamente. Solo tenemos que reordenar el resultado y obtenemos la representación correcta R = 2^n - 1 -(C - A). Ejemplo: 0010+1011=1101

Si A ≥ C, el resultado de sumar directamente es A + (2^n - 1 – C y reordenandolo da R = 2^n - 1 + (A - C), cuando el resultado correcto es A-C. Comparando ambas expresiones vemos que hay un acarreo del bit superior y al resultado le falta un 1. La Solución es reciclar al bit de acarreo sumándoselo al resultado. Ejemplo 0101+1100= *1*0001 , con cuatro bits se obtiene una cantidad inferior en una unidad a la correcta y además aparece acarreo. La solución es sumar el bit de acarreo 0001+*1*=0010.

En C'1 se puede sumar cantidades negativas, pero si existe acarreo al bit enésimo, se debe incrementar el resultado en una unidad.

ν **Resumen de las características:**

- El complemento a uno se reduce al complemento lógico.
- Al sumar cuando aparece un acarreo en el bit de mayor peso se debe incrementar en una unidad el resultado.
- Se complica la multiplicación y la división.
- Se puede producir desbordamiento.
- Rango de representación simétrico: $[-(2^{n-1} - 1), 2^{n-1} - 1]$
- Dos representaciones del cero.
- La Resolución = 1
- La extensión de signo $\Rightarrow$ repetir el bit de la izquierda.

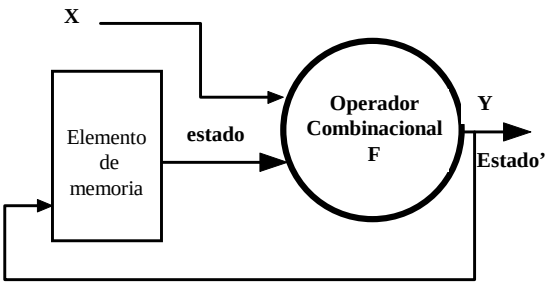
7.5 OPERADORES COMBINACIONALES Y SECUENCIALES

ν COMBINACIONAL:

- Y=F(X)
- No tiene elementos de memoria.
- Mientras las entradas fijas, la salida fija.
- Si se modifica una entrada, se modifica la salida
- El tiempo que tarda en dar el resultado es la suma de los retardos.
- Si se quiere guardar el resultado se debe utilizar un elemento de memoria.

ν SECUENCIAL:

- Y=F(estado,x)
- Estado=F(estado,x)
- Requiere varias fases para llegar al resultado final
- Debe tener la memoria necesaria para almacenar la información que debe transmitirse entre fases
- Un contador de fases
- Se necesita un algoritmo para calcular el resultado
- Debe contener las fases necesarias y la función a realizar en cada una de ellas.



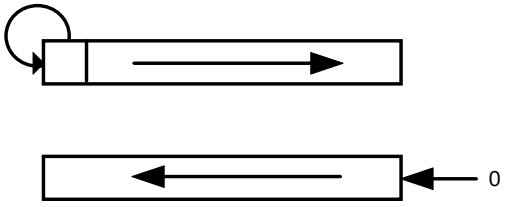
7.6 OPERACIONES DE DESPLAZAMIENTO

Consisten en mover los bits de una palabra hacia la izquierda o hacia la derecha. Los computadores sencillos sólo permiten desplazar una posición pero las máquinas más complejas permiten desplazamientos múltiples. Estos desplazamientos múltiples complican mucho el hardware y encarecen la implementación. Esta operación se implementa mediante un circuito combinacional específico.

La forma en que se tratan los extremos del desplazamiento permite diferenciar tres tipos de desplazamiento:

- Lógicos
- Circulares
- Aritméticos.

En los Desplazamiento lógico **l**os valores extremos se completan con 0's. En los Desplazamientos aritméticos el significado del desplazamiento depende de su sentido. los Desplazamiento a la izquierda son equivalente a multiplicar por una potencia de dos; y los Desplazamiento a la derecha son equivalente a dividir por dos. Es importante que conserven el signo del operando (bit izda). Esto depende del tipo de representación. En Desplazamientos en complemento a dos a la izquierda se Meten 0's por la derecha; y en desplazamientos a la derecha se Mete el signo por la izquierda

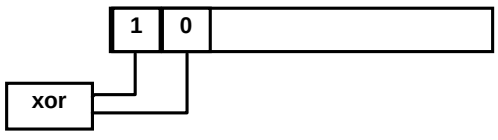


El desplazamiento a la izquierda puede producir desbordamientoEjemplo:

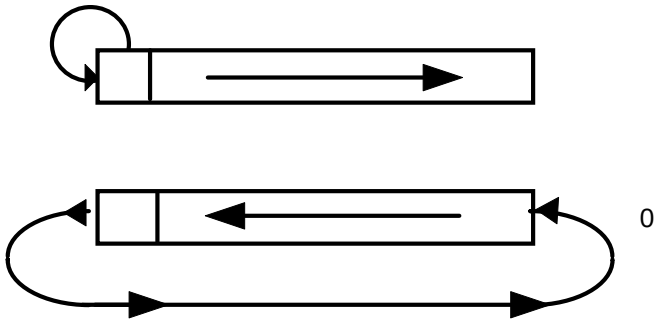
110 ;100; 000 Desbordamiento

001; 010; 100 Desbordamiento

Para detectar un desbordamientos hay que ver si los Bits [N-1] y [N-2] son diferentes.Un desplazamiento más pone un 0 en [N-1] cambiando el signo del número



Desplazamientos en C'1, si son a la derecha se mete es el signo por el bit más significativo, si son a la a la izquierda se mete también el signo.



DESPLAZAMIENTOS A LA DERECHA Y A IZQUIERDA EN C1

El desplazamiento a la izquierda puede producir desbordamiento.ejemplo:

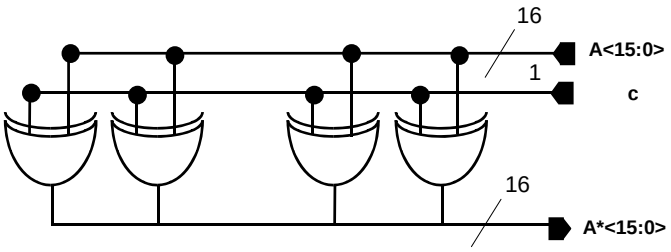
110 ;101; 011 Desbordamiento

001; 010; 100 Desbordamiento

Para detectar desbordamientos se comprueba si los Bits [N-1] y [N-2] son diferentes

7.7 OPERACIÓN DE CAMBIO DE SIGNO

ν Para Complemento a 1



Si C = 0, $a^*_{n-1} = a_{n-1}, \ldots, a^*_0 = a_0$

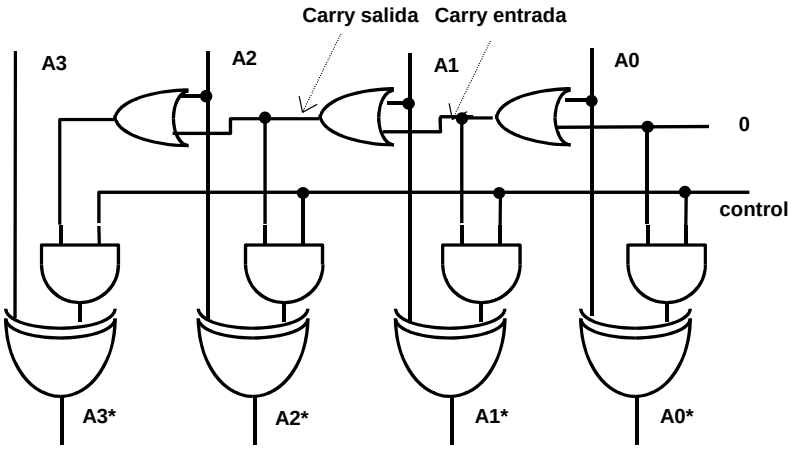
Si C = 1, $a^*_{n-1} = a_{n-1}, \ldots, a^*_0 = a_0$

ν PARA Complemento a dos (C'2)

La operación a realizar es complementar y a continuación sumar 1. Esto se podría hacer mediante puertas nor y un sumador, pero se pueden ahorrar puertas sabiendo que la operación a realizar es la siguiente: Se recorren los bitas del número a complementar de manera que lo sbits que se van obteniendo cumplen la siguiente condición: todos los bits son iguales hasta que se encuentra el primer 1 a partir de este momento se complementan todos

Para llegar al esquema final conviene darse cuenta que la operación final consiste en realizar complementos lógicos según el algoritmo anteriormente explicado, por lo tanto necesitaremos puertas xor. La señal de control de una puerta xor tendrá que ser *uno* cuando se cumplan dos requisitos:

1. La señal de control externa está 1, lo que indica que se quiere realizar el complemento a dos
2. el bit anterior es uno.



Cada bit actúa de la siguiente manera:

Comprueba si con anterioridad ha habido un bit con valor = 1. Esto lo sabe viendo si el carry de entrada es 1 o 0.

Si carry de entrada es 1, tiene que hacer dos cosas:

1.- Carry de salida =1

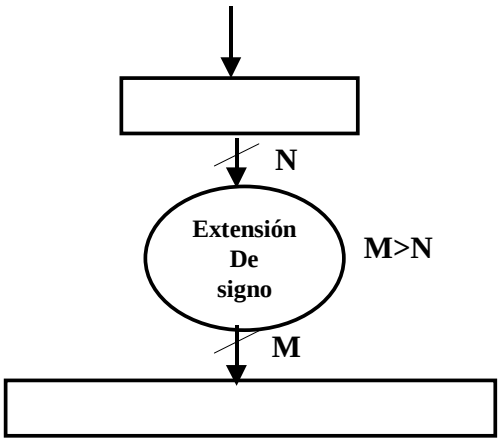
2.- Complementar la entrada.

Si carry de entrada es 0, hace dos cosas:

1. Comprueba si la entrada es 1. En este caso se transmite y no se complementa la salida.
2. Transmite el carry

7.8 OPERACIÓN DE EXTENSIÓN DE SIGNO

Se transfiere un operando de n bits, a un elemento de m bits (m>n). Si no se rellenan correctamente los bits, se cambia el valor numérico. La forma de operar depende de la representación del número.



- **SIGNO Y MAGNITUD:** Se desplaza el bit n-1 a la posición m-1,El resto de los bits se llenan con 0's
- **COMPLEMENTO A1 Y COMPLENTO A2:** Se repite el bit de signo hasta llenar todos los bits.

7.9 OPERACIONES SUMA Y RESTA

7.9.1 SUMADOR ELEMENTAL

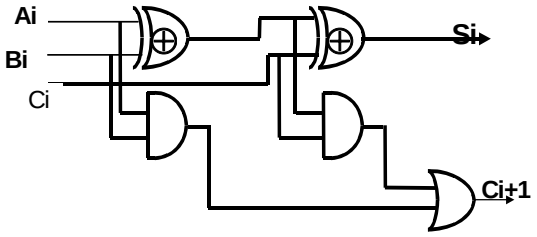
Circuito combinacional capaz de sumar dos dígitos binarios y de generar el acarreo.

Ai	Bi	Cin	Si	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$Si=Ai \oplus Bi \oplus Cin$

$Cout=AiBi+BiCin+AiCin$

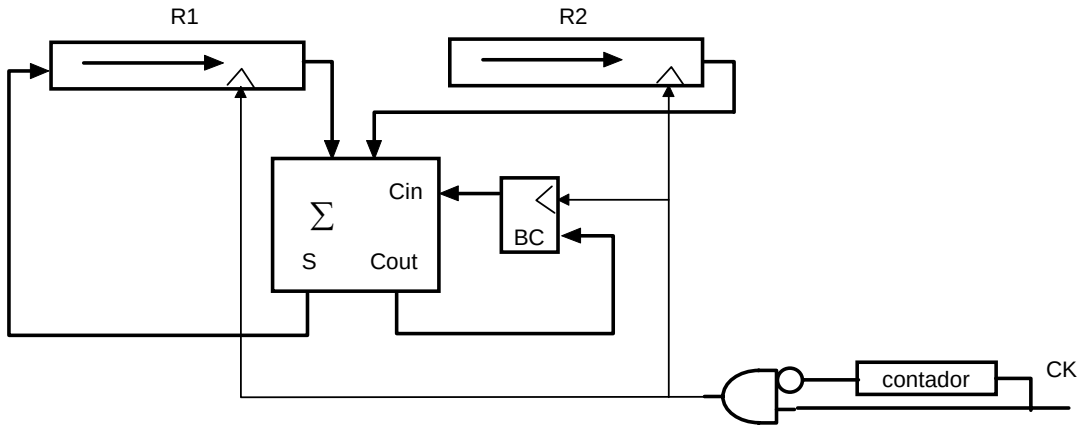
Lo más habitual es utilizar semisumadores (representación en tres niveles).



El sumador elemental se emplea normalmente como bloque constructivo para formar sumadores paralelos. También se puede utilizar para implementar un sumador secuencial.

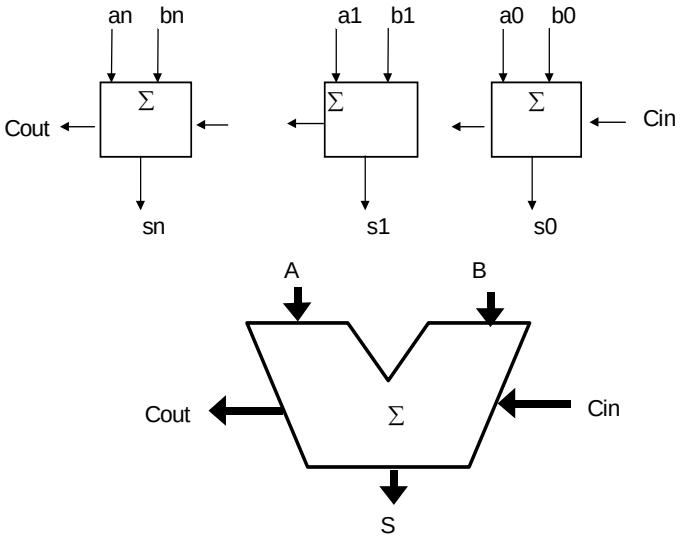
7.9.2 SUMADOR SECUENCIAL

Suponemos que se quiere sumar el contenido de los Registros R_1 y R_2 y que los registros permiten desplazamientos a la derecha.



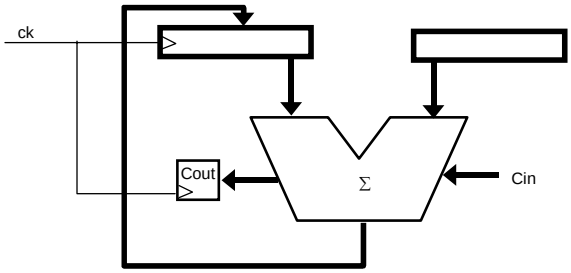
- Se inicia el biestable de acarreo (BC) a cero y el contador con el número de fases.
- Ventaja: Barato en área porque con un sumador de 1 bit implementas un sumador de n bits
- Desventaja: Bajo rendimiento. Puesto que se necesitan n ciclos de reloj para obtener el resultado final.
- En este ejemplo aparecen claramente determinadas las ventajas y desventajas de la implementación combinacional (aumento de área y de rendimiento) frente a la implementación secuencial (disminución de área y rendimiento)

7.9.3 SUMADOR PARALELO



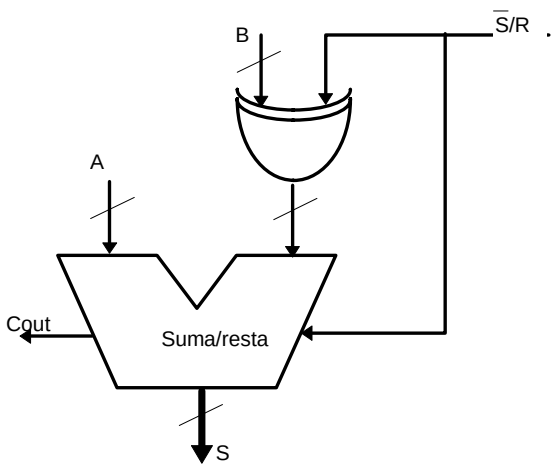
- El acarreo de entrada C_{in} depende del tipode representación usada para binario sin signo $C_{in} = 0$.
- El C_{out} se utiliza para detectar el desbordamiento.
- El retardo total viene dado por el máximo número de puertas que deben atravesar las señales: el camino más largo es el de los Acarreos.

ν Esquema general de utilización



7.9.4 SUMADOR/RESTADOR GENÉRICO

Aunque en las máquinas se podría implementar por un lado un sumador y por otro lado un restador, lo cierto es que se suelen implementar en un solo circuito. El hw necesario es Un sumador básico + circuitería complementaria + señal de control de SUMA/RESTA. Vamos a estudiar porque se puede hacer esto. Suponemos A positivo y B positivo. A - B = A - B + 2^n - 2^n = A + (2^n - B) - 2^n; pero 2^n - B es el C'2 de B. El C'2 de B se calcula haciendo la negación lógica de B y sumando 1 al resultado; y Restar 2^n es equivalente a restar 1 al bit S_n que no existe es decir ignorar el bit de acarreo C_{out}. Luego el esquema quedaría



Partiendo de este esquema se pueden generar el resto de S/R para las diferentes representaciones. Lo importante es particularizar los desbordamientos para cada representación, éstos indican que la operación no es correcta.

7.9.5 SUMADOR/RESTADOR BINARIO SIN SIGNO

El esquema que se utiliza es el de la figura anterior. Hay que particularizar el desbordamiento para la representación que se utiliza.

ν **SUMA**:

- El sumador es sencillo de particularizar puesto que es un sumador binario sin más complicaciones
- Operación: A+B
- El desbordamiento lo da el bit de acarreo $C_{out} = 1$

ν **RESTA**:

- Para particularizarlo hay que darse cuenta que el restador es un restador en C'2.
- De los dos casos que se pueden dar en la resta uno es correcto (A>B) y otro es incorrecto (A<B)
- Operación A-B con A>B
  - No existe problema el restador hace la operación $A+2^N-B=2^N+(A-B)$
  - Aparece un Cout=1, pero no se produce desbordamiento
- Operación A-B con B > A
  - Aparece desbordamiento puesto que sería un resultado negativo:
  - El restador en C'2 hace $A+2^N-B= 2^N-(A-B)$
  - Cout=0 cuando hay desbordamiento

ν Desbordamiento:

$$DE=Cout \oplus S'/R$$

7.9.6 SUMADOR/RESTADOR EN COMPLEMENTO A DOS

El esquema del sumado /restador es válido sólo hay que particularizar el desbordamiento. Como en complemento a dos la suma y la resta son lo mismo sólo se verán las condiciones de desbordamiento de la suma. Sólo se produce desbordamiento cuando se suman cantidades del mismo signo.

- **Suma de dos positivos**

* **Hay desbordamiento** cuando aparece un 1 en el bit de la izquierda del resultado
* $A_{N-1} = 0$ $B_{N-1} = 0$ $S_{N-1} = 1$ (S_{N-1} solo puede ser 1 si $C_{N-2} = 1$)
* La suma de dos números positivos no puede dar como resultado uno negativo
* $DE = A'_{N-1} \cdot B'_{N-1} \cdot C_{N-2}$

- **Suma de cantidades negativas:**

* $A_{N-1} = 1 = B_{N-1}$ y CN-2=0 (S_{N-1} solo puede ser 0 si $C_{N-2} = 0$)
* La suma de dos cantidades negativas no puede dar una positiva.
* $DE = A_{N-1} \cdot B_{N-1} \cdot C'_{N-2}$

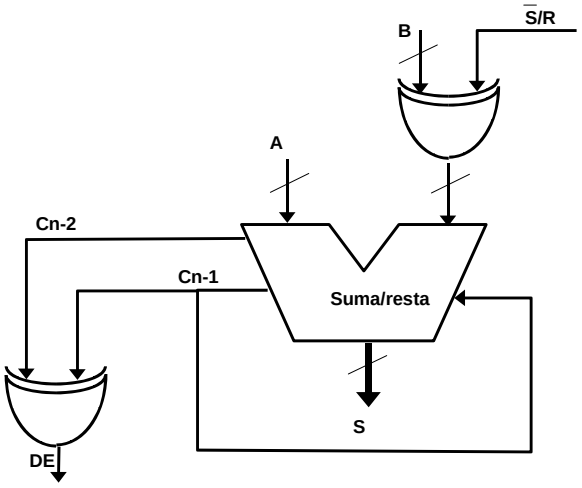
- **El desbordamiento total será:**

  $DE = A'_{N-1} B'_{N-1} C_{N-2} + A_{N-1} B_{N-1} C'_{N-2}$,

  que se puede simplificar $DE = C_{n-1} \oplus C_{n-2}$

7.9.7 SUMA Y RESTA EN COMPLEMENTO A UNO

El esquema general no se puede utilizar directamente. Hay que introducir modificaciones, teniendo en cuenta que la suma de dos números en C'1 se realiza en dos pasos: Se suman y luego Se suma el carry al resultado inicial. La negación del C'1 sólo exige la negación lógica.



Las condiciones de desbordamiento se obtienen con un razonamiento idéntico al anterior.

7.9.8 SUMADORES RÁPIDOS DE ACARREO ANTICIPADO

El principal inconveniente de los sumadores/restadores tal y como los hemos implementado es que para generar el acarreo de un sumador de 1 bit, se deben atravesar dos puertas lógicas pero para un sumador de n bits, el acarreo debe atravesar N puertas, y esto provoca importantes retrasos. Vamos a ver como implementar un sumador rápido.

- **Función generadora de acarreo G:**

* $g_i = a_i b_i$
* Indica si se genera acarreo en i

- **Función propagadora de acarreo P**

- $P_i = A_i + B_i$
- Indica si se propaga el acarreo al siguiente módulo

- Entonces
  - $C_i = C_{i-1} - P_i + G_i$
  - $S_i = P_i + C_{i-1}$

Si se desarrolla C_i se puede ver que aunque aumenta el número de puertas por nivel **NO** aumenta el número de niveles que es siempre DOS.

7.10 MULTIPLICADOR DE ENTEROS

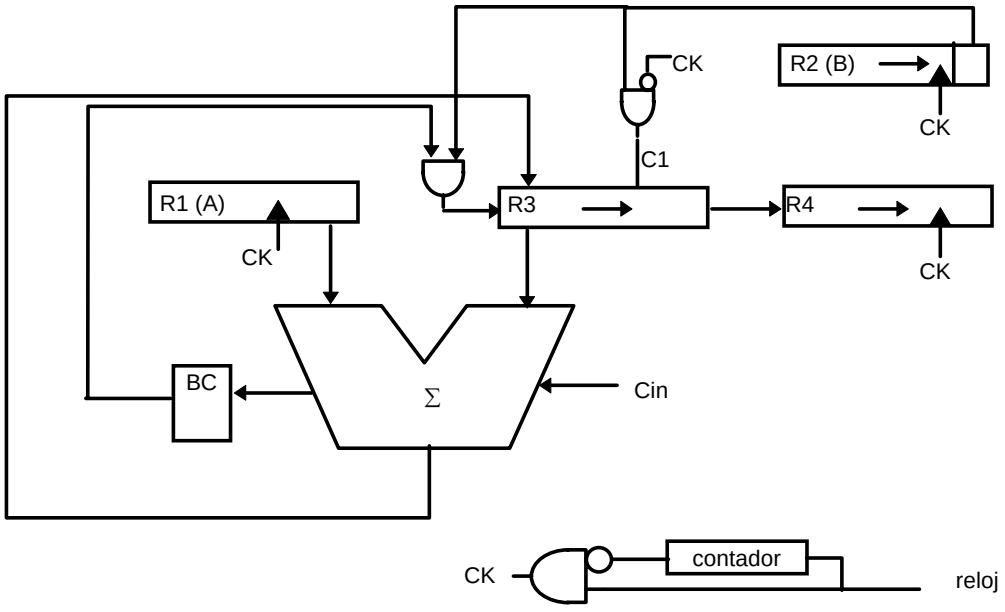
7.10.1 MULTIPLICADOR DE ENTEROS SIN SIGNO.

Se suele implementar como un operador secuencial mediante un sumador/restador y un algoritmo. Sólo las máquinas muy potentes lo implementan mediante un circuito combinacional. El producto de dos dígitos de N bits da lugar a un dígito de 2N bits.

7.10.1.1 Algoritmo de suma y desplazamiento (enteros +)

- Se utiliza para números binarios sin signo. Reproduce el método manual A x B
  1. Inspección sucesiva de los bits de B
  2. Si $B_i = 1$, se suma al resultado parcial A desplazada i-1 posiciones a la izquierda.
  3. No se hace nada si $B_i = 0$

Para evitar circuitería y control en lugar de desplazar a la izquierda el multiplicando A, se desplaza a la derecha el resultado parcial



Donde:

R1 contiene el multiplicando A

R2 Contiene el multiplicador B
Realiza desplazamientos a la derecha.

R3 y R4 son 2 registros concatenados.
Contienen los resultados parciales.

Se desplazan las dos a la derecha.

BC Flag que contiene el carry de las semisumas.

Se concatena con R3

La señal de reloj CK produce desplazamientos a la derecha de los registros R2, R3 y R4. La señal C1= CK'·B_0, produce la carga en el R3 de la suma parcial y la carga del carry producido en la suma.

ALGORITMO:

1.- Se inicia R3 ← 0 y el contador de fases con ← 0

2.- Si $R2_0$ = 1 → R3 ⇐ R3+R1(flanco negativo)

= 0 → nada R3 ⇐ R3 + 0

3.- Incrementar el contado (flanco positivo)

4.- Desplazar derecha BC - R3 - R4(flanco positivo)

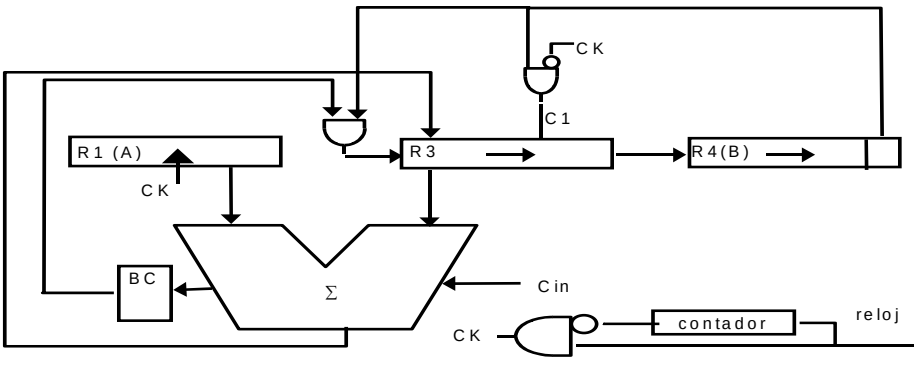
5.- Rotar a la derecha R2 (flanco positivo)

6.- Observar el contador si no ha acabado vuelta a 2

Para comprender el algoritmo es importante recordar que la suma es un operador combinacional y que solamente se puede considerar realizada cuando se carga el resultado sobre un registro. Es decir en el algoritmo R3:=R3+R1 es equivalente a decir que el resultado de la suma se carga sobre R3.

La razón por la que se utilizan los flancos positivos y negativos en el algoritmo es para ahorrar flancos tiempo. Se deben hacer dos operaciones diferentes cargar los registros y desplazarlos. Cada una de estas operaciones se debe hacer con un flanco de reloj. Si se utilizan siempre flancos positivos para realizar un bucle del algoritmo necesitamos dos ciclos de reloj(dos flancos de subida). Si se utiliza un flanco positivo y otro negativo, solo se consume un ciclo por cada bucle.

Al esquema anterior todavía se le puede eliminar hardware. Este ahorro consiste en eliminar el registroR2. Una vez eliminado el multiplicador B se puede cargar sin problemas en el registro R4.



7.10.1.2 Algoritmo de suma y resta

- Se basa en el siguiente hecho:

  Nº = 01110 = 10000 - 00010

- Ejemplo de multiplicación
  - si B = 01110 y quiero hacer A x B
  - A·(01110) = A·(10000 - 00010) = A ·10000 - A ·00010=$A\cdot 2^4$-$A\cdot 2^1$

- El producto queda reducido a una suma y resta de desplazamientos del multiplicando A

ν **GENERALIZANDO:**

- Se modifica el algoritmo de suma y desplazamiento para tratar las cadenas de 1's que contenga el multiplicador B. Esta modificación supone una modificación de la estructura que lo implementa. El nuevo algoritmo es el siguiente: avanzando de menos significativo a más significativo.
  - Se resta $A \cdot 2^i$ cuando al analizar B se encuentra el $1^{\underline{er}}$ bit de una cadena de 1's
  - Se suma $A \cdot 2^i$ cuando al analizar B se encuentra el 1^o 0 después de una cadena de 1's.
- Es importante conservar el signo de las sumas parciales.
- Como los resultados parciales pueden ser tanto positivos como negativos, el multiplicando A sólo debe ocupar N-1 bits de la palabra.
- Vamos a suponer que las formas negativas se almacenan en C'2.
- Para comprender el algoritmo es importante recordar que un desplazamiento a la izquierda de A , es decir A·2i, es equivalente a desplazar a la derecha el resultado parcial i veces.

ν ALGORITMO:

- Sabiendo que:

  Partimos del esquema anterior (con ligeras modificaciones)

  R3 almacena resultados parciales

  1. Inicializar R3 ⇐ 0 y el contador de Fases
  2. Observar B(0) (contenido en R2.)

     Si es principio de cadena de 1's R3←–R3-R1 (equivalente a restarle A2i)

     Si es el primer cero después de una de unos R3←–R3+R1(equivalente a sumarle A2i)

     Si no es ninguno de los dos anteriores no se hace nada
  3.- Incrementar el contador
  4.- Desplazar a la derecha BS → R3 → R4 (equivalente a desplazar A a la iz)
  5.- Rotar a la derecha R2
  6.- Observar el contador.
  7.- Si $B_{n-1} = 1$ se cierre la última cadena de 1's sumando R3 ⇐ R3 + R1

  (equivalente a sumarle $A \cdot 2^N$)

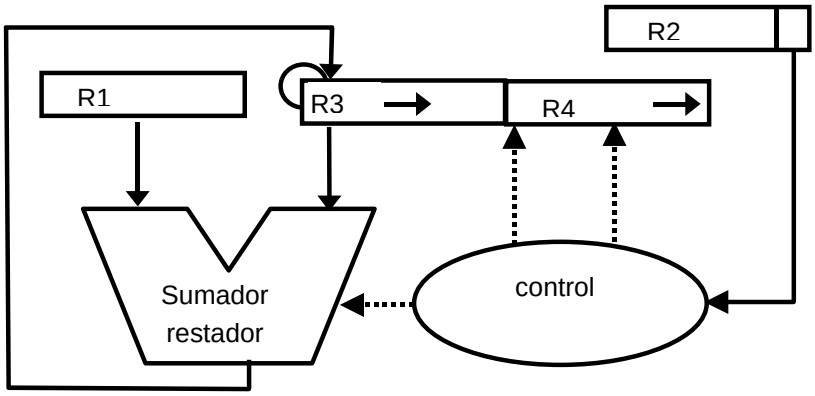
  ATENCIÓN:

  ν El punto 7 se añade para tratar correctamente el caso que el bit más significativo sea un 1. En este caso se debe cerrar la cadena

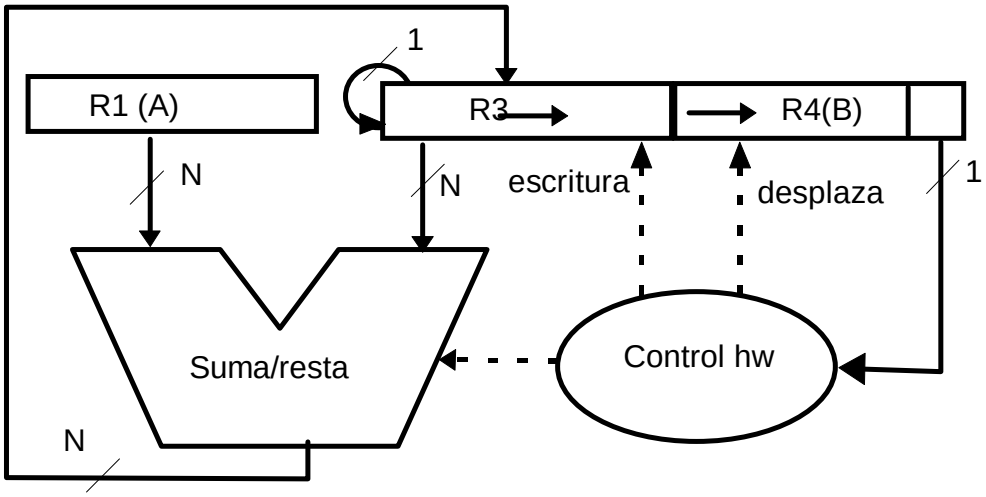
  1110 = 10000 - 0001

- Para poder aplicar este algoritmo hay que introducir cambios en el camino de datos estudiado en el algoritmo de suma y desplazamiento.

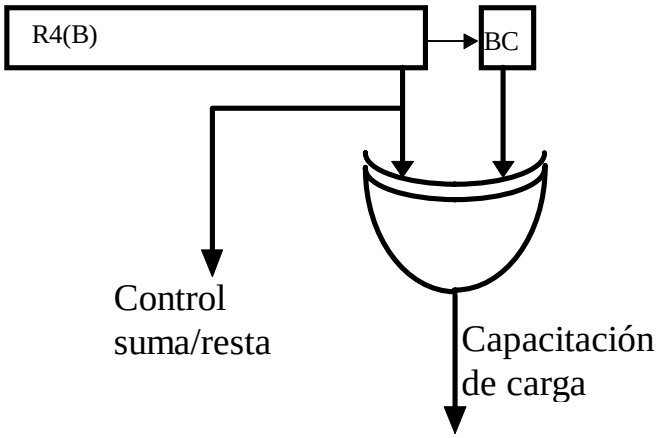
- Por ejemplo La concatenación ya no se hace con el bit de carry, sino con el de signo para conservar el signo de los productos parciales.
- También hay que añadir el control que compruebe si B(0) es el primer 1, el primer 0 o es otro de los casos para generar las señales de control que sumen y desplacen, resten y desplacen o simplemente desplacen.
- A estas modificaciones correspondería el siguiente esquema



En este esquema se puede ahorrar nuevo hw si suponemos que el multiplicador B no se carga en R2 sino que se carga en R4, luego se puede quitar R2.



Implementación sencilla del control mediante puertas XOR



7.10.2 MULTIPLICACIÓN CON SIGNO

Consiste en la transformación a multiplicación de enteros sin signo los Pasos a seguir son:

1.- Análisis de los signos de los operandos para determinar el signo del resultado.

2.- Conversión a positivo de los operadores negativos.

3.- Multiplicar

4.- En su caso, cambio de signo

El ALGORITMO DE BOOTH: Permite trabajar directamente con los operandos con signo. Se emplean datos representados en C'2 y el Algoritmo de sumas y restas con correcciones. Un multiplicando A negativo no es problema puesto que las sumas se hacer directamente, obteniéndose el resultado negativo en C'2. Siempre que se hagan las extensiones de signo en los productos parciales. Demostración:

Aunque el sumador es de N bits, la suma que realizamos es de 2N bits, por lo tanto el operando A en C'2 se representa $-A=2^{2N}-A$. Operando según el algoritmo visto se obtiene $R'= (2^{2N}-A)\cdot B= 2^{2N}B-AB$ y se debería obtener lo siguiente: $R=2^{2N}-AB$. Voy a ver que tendría que añadir o quitar a R' para que fuese igual a R

R'=22NB-AB+22N-22N=(22N-AB)+22nB-22N =(22N-AB)+22N(B-1) luego tendría que restarle 22N(B-1) al resultado, es equivalente a modificar el acarreo, por lo tanto este termino se puede ignorar. En realidad este término es la suma de todos los acarreos que se desprecian según se hacen las sumas parciales(si se hacen 2n desplazamientos a B se observa que el valor final es 0)

demostración:

$2^{2N}-A +2^{2N}-A= 2^{2N}+2^{2N}-2\cdot A$ el 2^{2N} es equivalente a un acarreo que se puede ignorar

Ahora sumamos otra vez $2^{2N}-A$ obteniendo $2^{2N}+2^{2N} +2^{2N}-3\cdot A$, apareciendo un nuevo acarreo que igual que el anterior se puede ignorar.

El problema surge cuando es el multiplicador B el que es negativo $A\cdot(-B)$. Suponemos **B negativo** luego su representación es: $2^N - |B|$. En este caso B siempre se representa con N bits, porque no interviene directamente como operador en la suma. Solo se utiliza para saber si se realizan sumas o restas, luego $A\cdot(- B) \Rightarrow R' = A (2^N - |B|)$.Que es el resultado que se obtiene al aplicar directamente el algoritmo. $A \cdot 2^N - |B|A$

El resultado correcto sería:

$R = 2^{2N} - A\ |B|$

Vamos a ver la transformación que debería realizarse sobre R* para que sea igual a R.

$R^* = A\,(2^n - |B|) = 2^n A - A\,|B| + 2^{2n} - 2^{2n} = 2^n A - 2^{2n} + 2^{2n} - A\,|B| = 2^n A - 2^{2n} + R$

Debería sumar 2^{2N} al acarreo→ se puede prescindir.

Debería restar $2^N A$ al resultado

El problema ahora es estudiar cuando el algoritmo introduce el desplazamiento $2^N A$.

En el algoritmo de sumas y restas se introduce $2^N A$ cuando $B_{N-1}=1$ ya que en este caso se debe sumar esta cantidad al resultado parcial para que el resultado sea correcto.

Como en este caso B_{N-1} siempre es 1, el algoritmo siempre introduce el error. La solución es eliminar el ultimo paso del algoritmo de sumas y restas

($A \cdot 2^N$ es el contenido de R1 justo después de n rotaciones a la derecha Luego es suficiente con no hacer la operación R3←R3+R1)

- **Algoritmo**

1.- Inicializar: R3 ← 0; contador ← 0.

2.- Observar B(0)

Si es principio de cadena de 1's R3 ⇐ R3 - R1

Si es final de cadenas de 1's R3 ⇐ R3 + R2

3.- Incrementar contador.

4.- Desplazar a la derecha BS - R3 - R4

5.- Rotar a la derecha R2

6.- Estudiar el contador para acabar.

El 7º punto no se añade para no tener en cuenta el desplazamiento $2^n A$

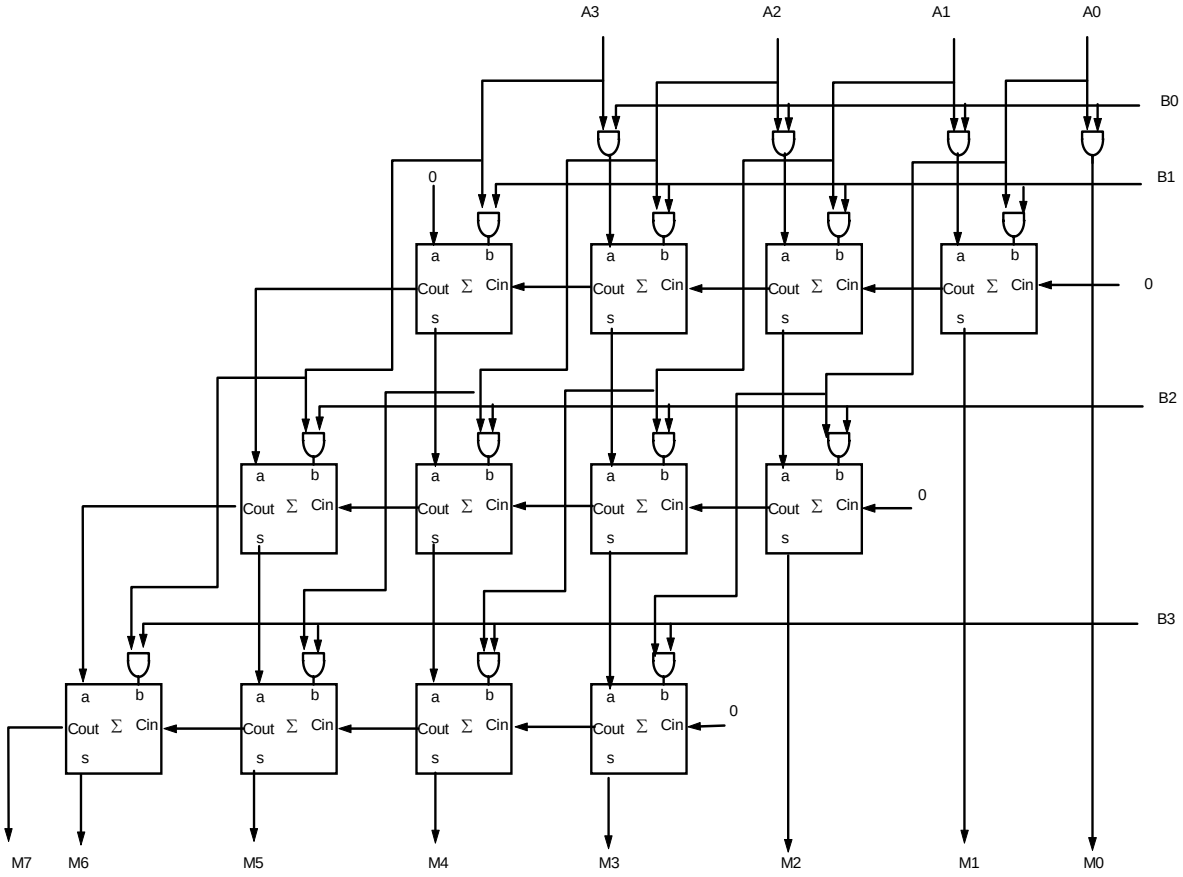
(Recuerda es un multiplicador de N bits: como los bits se numeran de 0 - (n-1) el bit n-esimo no pertenece al rango).

7.10.3 MULTIPLICADORES COMBINACIONALES:

También llamados rápidos son Circuito combinacionales que realizan directamente la generación y suma de todos los productos parciales. La generación de los productos parciales es inmediata. AND de todos los bits de A con cada uno de los bits de B.

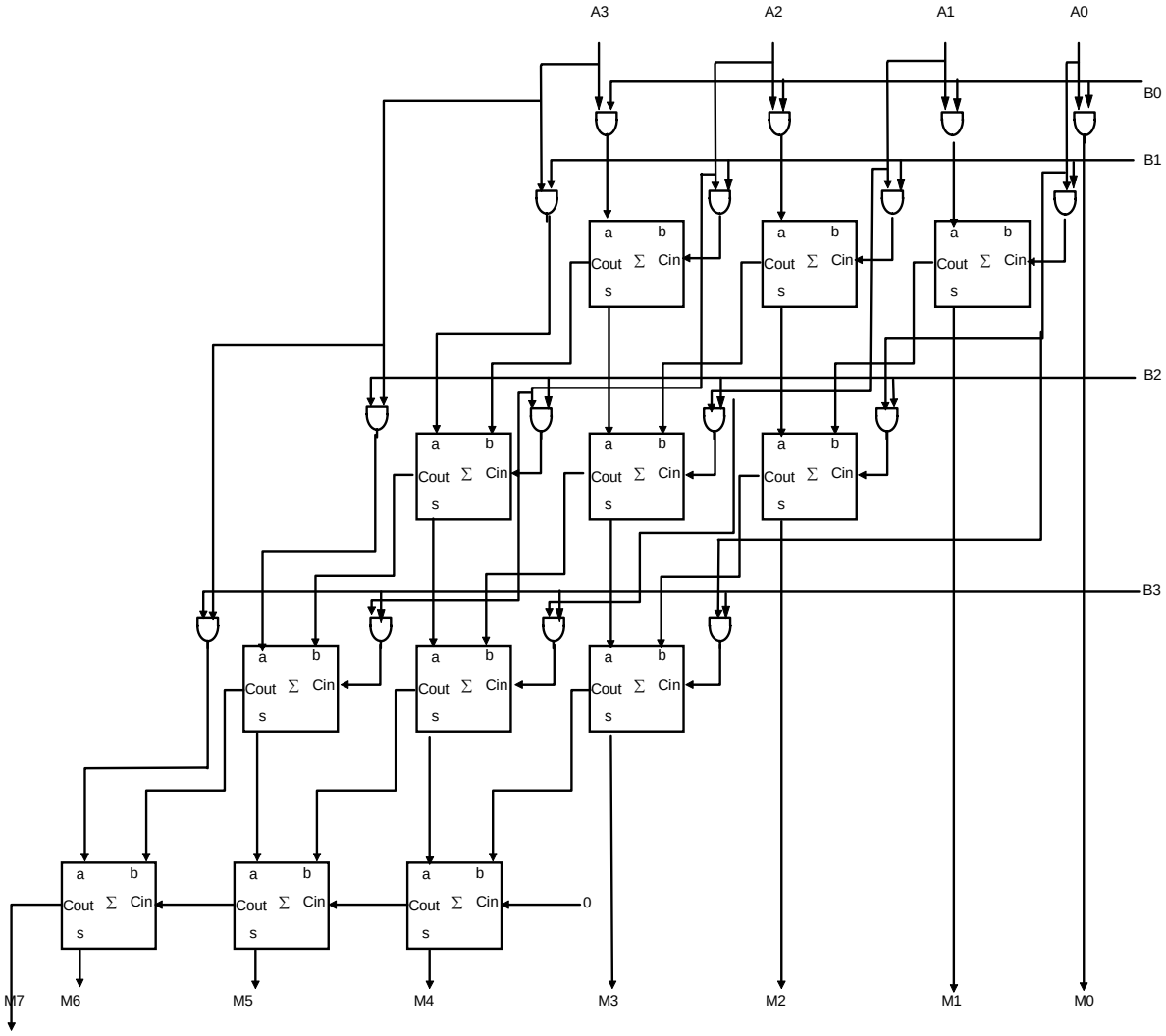
Si el número de bits es N se necesitan (N-1) sumadores de N bits cada uno de ellos.

(En lugar de poner sumadores de N bits vamos a poner n sumadores de 1 bit para que se vea más claro.



- El retardo del multiplicador viene dado por su camino crítico:
  - retardo de n-1 acarreos del primer sumador.
  - retardo del último bit de suma del 1º sumador.
    - retardo de los acarreos (n-2) y de los últimos bits de suma de las restantes filas de sumadores.

Este retardo se puede mejorar teniendo en cuenta lo siguiente: es lo mismo transmitir el acarreo del sumador y al sumador y+1 de la etapa j, que el sumador y+1 de la etapa j+1.



- El retardo viene dado por los n acarreos de los primeros bits de cada etapa, más los n-2 adicionales de la última etapa.
- Con este esquema se añade una etapa más pero es más rápido
- En este caso **NO** se pueden usar sumadores de N bits, sino sumadores de un solo bit.

7.11 OPERACIÓN DE DIVISIÓN DE ENTEROS

La división BINARIA SIN SIGNO es más compleja que la multiplicación. Se suele implementar siempre de manera secuencial es decir un sumador/restador más una algoritmo.

7.11.1 DIVISIÓN CON RESTAURACIÓN

(Hamacher + Stalling)

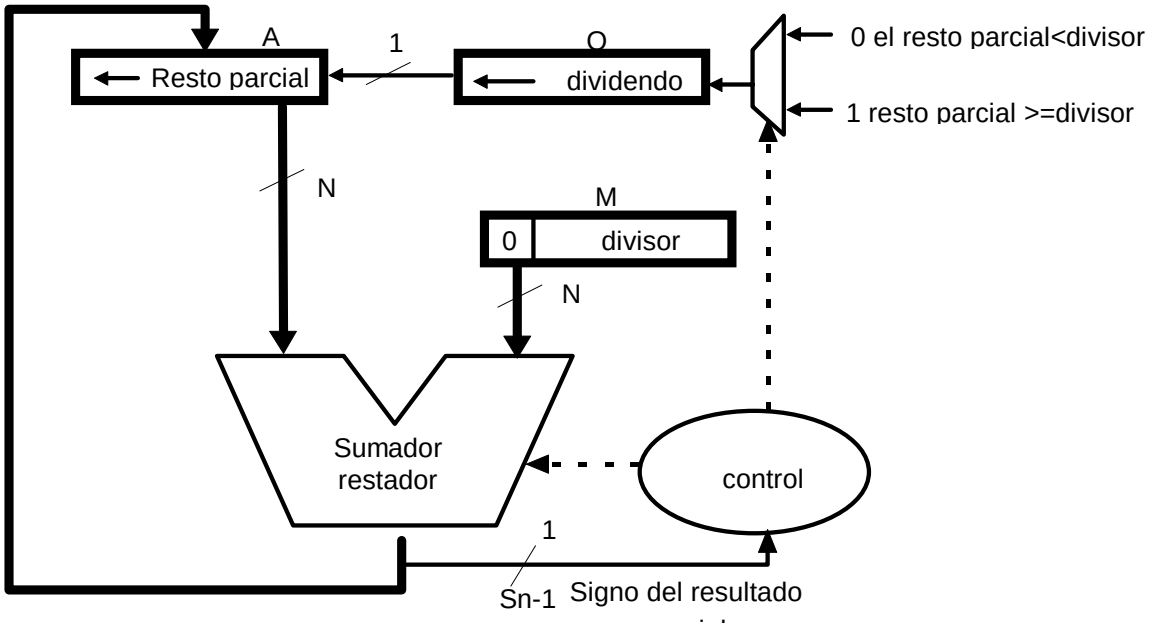
Se utiliza para dividir enteros sin signo. El dividendo, el divisor, el cociente y el resto tienen el mismo número de bits. Si los números ocupan N bits, los registros van a tener N+1 bits para tratar el caso de restos parciales negativos que se representan complemento a dos, como consecuencia: El sumador/restador debe ser de N+1 bits.El algoritmo a mano es el siguiente:

Se seleccionan bits del dividendo de izquierda a derecha hasta que el conjunto de bits seleccionados sea un número mayor o igual que el divisor.

Por cada bit seleccionado se añade un 0's al cociente
cuando el dividendo parcial es mayor o igual que el divisor se añade un 1
se resta al dividendo parcial el divisor siendo el resultado el resto parcial
A partir de este momento se repite un ciclo:
se añade bits del dividendo al resto parcial hasta que este sea mayor que el divisor
por cada bit añadido al resto parcial se añade un cero al cociente
cuando el dividendo parcial mayor que el divisor :
se añade un 1 al cociente
se resta el resto parcial y el divisor
el proceso continua hasta que se acaban los bits del dividendo
Ejemplo:

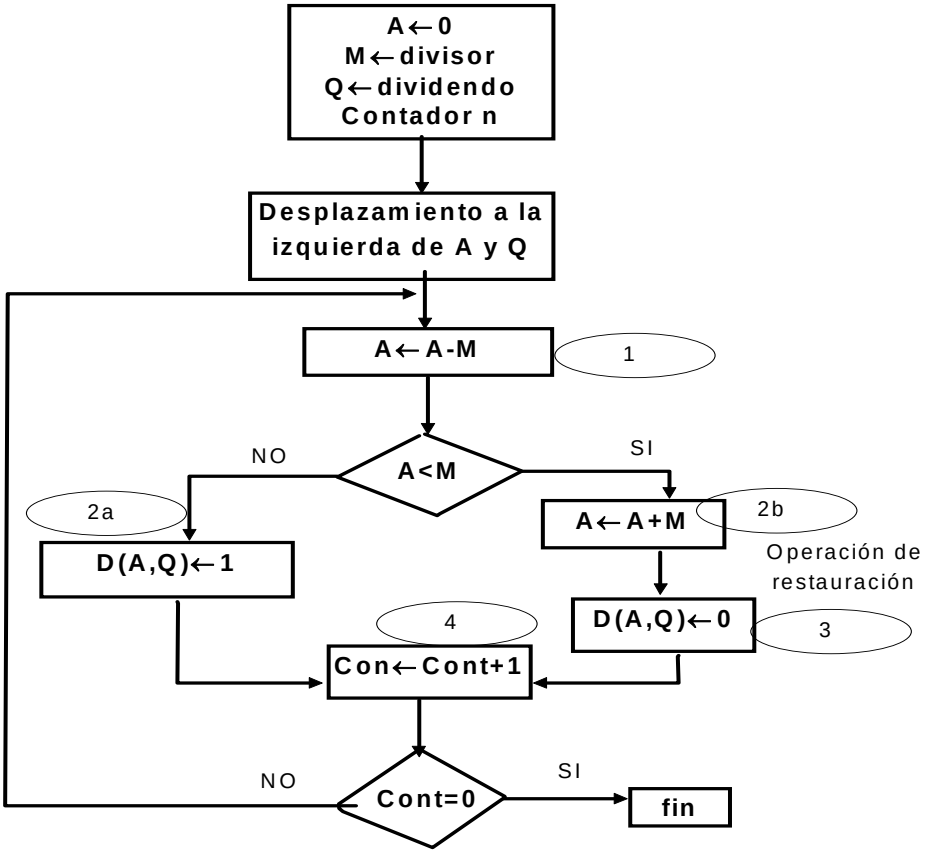
```
10010011 | 1011
 1011    | 00001101
-------
001110
 1011
------
 001111
  1011
 ------
  0100
```

La estructura de un divisor es la siguiente:



- M contiene el divisor positivo.
- Q contiene el dividendo positivo.
- A contiene los restos parciales
- Cuando termine la división el cociente de N bits estará en Q y el Resto en A.

El algoritmo se puede ver en la siguiente figura:



La razón de cargar inicialmente en A todo ceros es para que en el primer desplazamiento quede cargado en A el primer bit del dividendo y así comenzar el algoritmo

A y Q están concatenados de manera que cada desplazamiento a la izquierda carga en A un bit del dividendo en el resto parcial.

Esto se hace para comprobar si el resto parcial "cabe o no cabe"

La forma de comprobar si cabe es restar el divisor del resto parcial, si el resultado es negativo (S_{N-1} = 1) la resta no es valida, y para restaurar el resto parcial se a suma el divisor

7.11.2 DIVISIÓN SIN RESTAURACIÓN

Es posible mejorar el algoritmo estudiado eliminando la restauración. Partiendo del esquema de la división con restauración vamos a estudiar con detenimiento que se hace en cada caso. Después de una sustracción A=A-M (paso 3) se hace lo siguiente.

- Si A es positivo⇒ se desplaza A a la izquierda (paso 2) y se le resta M (paso 3) es decir **2A - M**
- Si A es negativo se restaura A + M (paso4) y después se desplaza a la izquierda (paso 2) (2A+ 2M), y se le resta M ⇒ **2A + M**.(paso 3)

En función de esto el Algoritmo sin restauración queda como sigue:

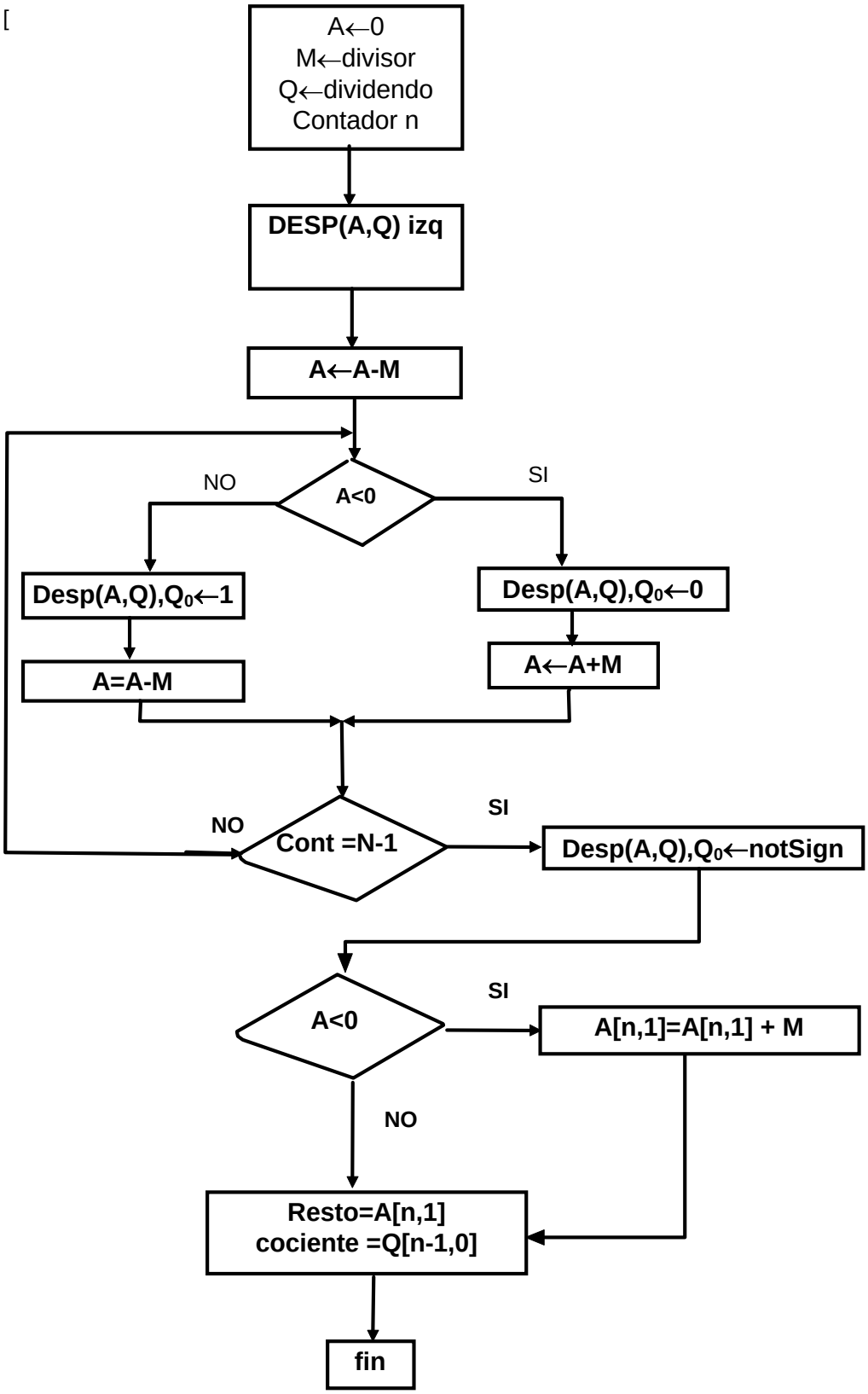
- Si A positivo

    - Poner un 1 en Q_0
    - Desplazar una posición A y Q hacia la izquierda
    - A:=A-M
- Si A es negativo,
    - Poner un 0 en Q_0
    - desplazar A y Q a la izquierda
    - A:=A+M
- Por último si A es negativo sumar A:=A+M

Atención: **es importante cargar el signo antes de realizar los desplazamientos** . En caso contrario se podían perder los signos.

Vamos a estudiar dos implementaciones diferentes de este algoritmo con un ejemplo cada de cada una para que se vea la diferencia de ambos. En ambos casos vamos a suponer que NO se utiliza un registro independiente para almacenar el cociente, sino que se introduce en la mitad de menor peso del resto parcial. El primer algoritmo es el de la siguiente figura

[



Al acabar el algoritmo el cociente queda almacenado en el registro Q de n bits y el resto queda almacenado en los n bits más significativos del registro A que tiene n+1 bits.El bucle se realiza N-1 veces

El ultimo desplazamiento se realiza para que el cociente quede correctamente cargado pero tiene el inconveniente de desplazar el resto una posición a la izquierda .Este es el motivo de que quede almacenado en A[n,1] en lugar de A[n-1,0].

En el caso de que el resto sea negativo, hay que sumarle el divisor para que quede positivo-l a división de números positivos no puede dar como resultado un resto negativo-.

Un problema de esta implementación del algoritmo es que el cociente y el resto no quedan localizado en los bits menos significativos del registro A. Y esto provoca un hardware extra a la hora de realizar la última suma.

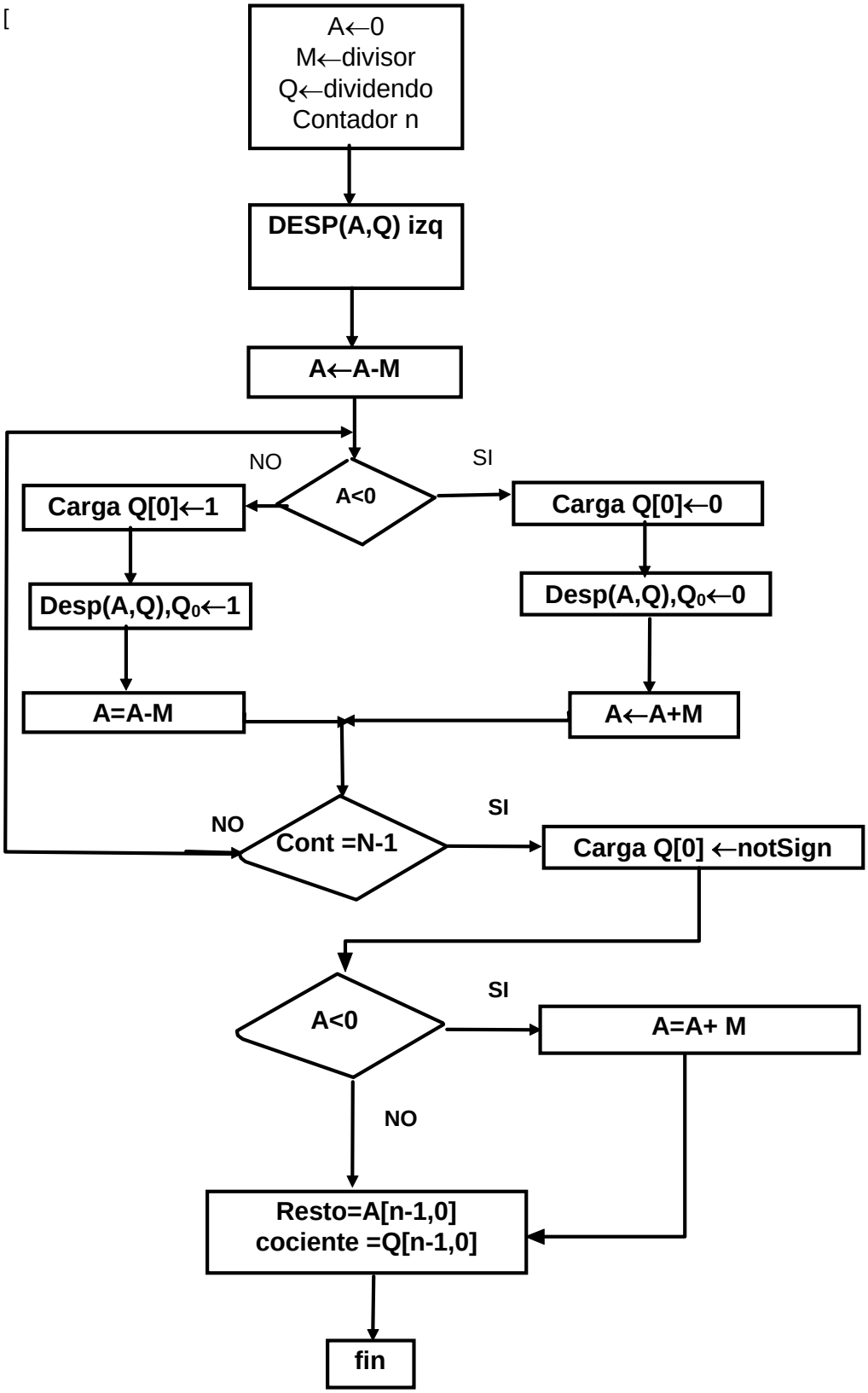
Ejemplo Dividir 7/3:

A[n]	A	Q	
	0000	0111	inicio
0	0000	1111	desplaza
-	1101		suma en c'2
1	1101	1111	
1	1011	1110	desplaza
+	0011		suma
1	1110	1110	
1	1101	1100	desplaza
+	0011		suma
0	0000	1100	
0	0001	1001	desplaza
-	1101		suma en C'2
1	1110	1001	
1	**110**1	***0010***	desplaza
0	**011**		
0	***001***		

Este último paso se utiliza para dejar el adecuado resíduo positivo en A, al final de N ciclos.(recordar que en divisiones de enteros sin signo el resto debe ser positivo)

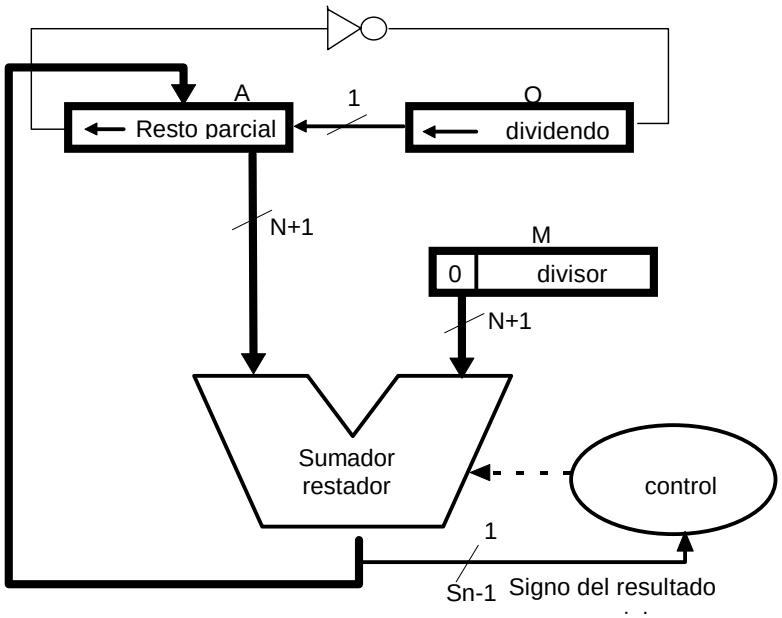
- A continuación vemos el algoritmo propuesto por Henessy-Paterson:

[



A[n]	A		
	0000		inicio
0	0000	111	desplaz
-	1101		a
		11x	suma en c'2
1	1101		
1	1101	11x	carga
1	1011		q[0]
+	0011	110	desplaz
			a
		10x	suma
1	1110		
1	1110	10x	carga
1	1101		q[0]
+	0011	100	desplaz
			a
		00x	suma
0	0000		
0	0000	10x	carga
0	0001		Q[0]
-	1101	001	desplaz
			a
		01x	suma en C'2
1	1110		
1	1	01x	carga
	110		suma
	0011	***010***	
0	***0001***		
0			

¿ que ventaja tiene este algoritmo sin restauración frente al de restauración, si incluye una suma mas ?Necesita un paso menos en el ciclo.Una estructura que trata de manera sencilla el problema de los valores que se introducen por el bit menos significativo de Q es la siguiente:



No existen algoritmos de división que efectúen la operación con signo.

- Por lo general se necesita el procesamiento previo de los operandos y el procesamiento de los resultados:
- Por los general se transforman los operandos a positivos, se trabaja con ellos como positivos y se le da el signo correcto al resultado.

7.12 REPRESENTACIÓN DE REALES. COMA FLOTANTE

7.12.1 NOTACIÓN CIENTÍFICA

Con las representaciones que se han estudiado hasta el momento se pueden representar enteros. Si queremos que un computador tenga suficiente potencia de cálculo hay que introducir notación fraccionaria.

- **PUNTO FIJO**

Para un conjunto de bits como el que sigue $b_3b_2b_1b_0.b_{-1}b_{-2}b_{-3}$ Su valor fraccionario se obtiene de la siguiente manera

$$b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0 + b_{-1} 2^{-1} + b_{-2} 2^{-2} + b_{-3} 2^{-3}$$

Es posible representar un rango de enteros positivos y negativos centrados en 0.Se puede representar números con parte fraccionaria suponiendo un punto fijo en la representación. Esto tiene dos inconveniente no se pueden representar números grandes (en magnitud) y no se pueden representar fracciones pequeñas.

ν **PUNTO FLOTANTE**

Esta limitación se puede salvar en los números decimales usando **notación científica o punto flotante**

Ej.: $976.000.000.000.000 = 9.76\ 10^{14}$

$0{,}000.000.000.000\ 976 = 9{,}76\ 10^{-14}$

Se ha movido dinámicamente el punto decimal a una localización deseada y se ha utilizado un exponente en una base 10 para indicar la posición original al punto decimal. Esto permite un rango de números muy grandes y muy pequeños con muy pocas cifras. Esta misma aproximación se puede aplicar a los binarios:

* $N^o = \pm M \cdot B^{\pm E}$
* M ⇒ Mantisa.
* E ⇒ exponente
* B ⇒ Base.

ν **REPRESENTACIÓN TÍPICA:**

ig	expo nente	m antisa

Donde Sig es el signo de la mantisa, la **BASE** se utiliza implícitamente, por eso no hace falta almacenarla y suele ser 2 o 16.

El **EXPONENTE T**iene su propio signo. Se suele representar en exceso 2^{q-1}, siendo q el número de bits del exponente.Por ejemplo suponiendo que el exponente tenga 8 bits, la representación será en exceso 128. Para hallar el valor verdadero del exponente hay que restarle 128 al valor almacenado. En este caso el rango del exponente será -128 a 127. La razón de utilizar esta representación es que el exponente mas pequeño se le asignan todo ceros y al más positivo se le asignan todo 1s, con lo que se simplifica el HW necesario para comprobar el tamaño real del número y además permite la utilización de UAL de enteros para el estudio de los exponentes. Ejemplo de exceso 2^{q-1} con q = 4

0000	-8
0001	-7
0010	-6
0011	-5
0100	-4
0101	-3
0110	-2
0111	-1
1000	0
1001	1
1010	2
1011	3
1100	4
1101	5
1110	6
1111	7

Visto lo anterior los números:

$001 \cdot 2^3 = 010 \cdot 2^2 = 100 \cdot 2^1 = 1000 \cdot 2^0 = 8$

$0.0110 \cdot 2^6 = 0.1100 \cdot 2^5 = 1.1000 \cdot 2^4 = 11.0000 \cdot 2^3 = 24$

Son diferentes representaciones del mismo.

LA MANTISA Se suele representar en modo fraccionario

$B_{-1}\, 2^{-1} + b_{-2}\, 2^{-2} + b_{-3}\, 2^{-3}$

Se suele normalizar porque Permite el intercambio de información con mucha facilidad, Simplifica el cálculo aritmético, y Aumenta la exactitud al eliminar ceros que no contiene información por bits que si la contiene. LA Forma general de la mantisa normalizada es

$\pm$ 0.1 bbb ... $b2^{\pm\epsilon}$,

donde b puede ser 0 o 1. Esto tiene un Problema: Los 2 bits de la izquierda siempre son los mismos,e s decir No aportan información y por lo tanto Consumen bits inecesariamente. Para solucionar esto de manera implícita, se supone que el bit más a la izquierda de la mantisa, es siempre 1 lo que ayuda a ganar espacio. Por ejemplo: con un campo de 23 bits, se guardan mantisas de 24 bits. Vamos a ver un ejemplo de como se trata la mantisa. Una mantisa almacenada de la forma M = 0 — 0 Es en realidad la siguiente mantisa 0,10 — 0. Puesto que hay que añadirle la coma y el 1 por defecto. El rango decimal que puede abarcar la mantisa es el siguiente:

* para M= 00----0 = 0,100----0 que es = $1 / 2^1 = 0,5$
* para M=11----1=0,11---1=

$$\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^{N+1}} = 1 - \frac{1}{2^{N+1}}$$

El rango de la mantisa, suponiéndola n bits, incluido el implícito, es:

$$\frac{1}{2^1} \leq |M| \leq 1 - \frac{1}{2^{N+1}} < 1$$

En estas expresiones estamos suponiendo que la base es 2.

El rango de representación de números así codificados con un total de 32 bits sabiendo que utiliza:

1 bit para el signo
8 bits para exponente en exceso 128
23 explícitos para la mantisa.

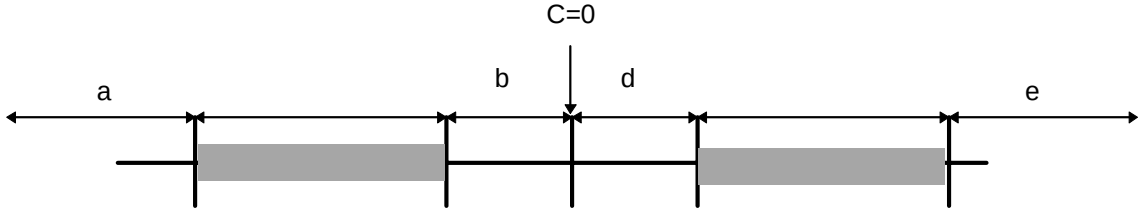
es el siguiente:

Nº negatvos $-(1 - 2^{-24}) \cdot 2^{127} \rightarrow - 0,5 \cdot 2^{-128}$

Nº positivos $0,5 \cdot 2^{128} \rightarrow (1 - 2^{-24}) \cdot 2^{127}$

Con esta representación se pueden definir 5 regiones no incluidas:

- OVERFLOW NEGATIVO: números negativos menores que:- $(1 - 2^{-24}) \cdot 2^{127}$
- UNDERFLOW NEGATIVO: números mayores que-$0,5 \cdot 2^{-128}$
- CERO que en esta representación no existe
- UNDERFLOW POSITIVO: números positivos menores que:$0,5 \cdot 2^{-128}$
- OVERFLOW POSITIVO: números positivos mayores que:$(1 - 2^{-24}) \cdot 2^{-128}$



Overflow = magnitud demasiado grande

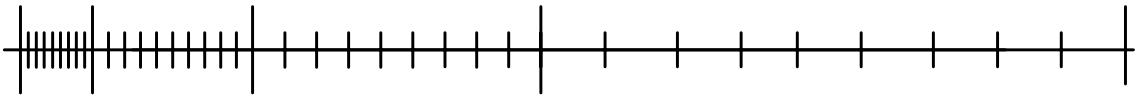
Underflow= magnitud demasiado pequeña.

ν **ALGUNAS CARACTERÍSTICAS DE LA COMA FLOTANTE**

El número de valores diferentes que se pueden representar es el mismo en cualquier formato que se utilice. Por ejemplo con 8 bits en binario puro se pueden representar 2^8 números diferentesen coma flotante distribuido de la siguiente manera:

- 1 bit de signo,
- 4 bits de mantisa y
- 3 de exponente
- los números representables son $2^1 \cdot 2^4 \cdot 2^3 = 2^8$

Los números que se representan no se distribuyen uniformente en el espacio numérico. Están más próximos en el origen.



Ejemplo:

para 3 bits de mantisa puedo representar 0,1,…,7

para 4 bits de mantisa puedo representar $2^0, 2^1, 2^2, 2^3, \ldots, 2^{15}$

entonces

para 2^0→ 0,1,2,3,…,7 de 1 en 1

para 2^1→ 0,2,4,6,…,14 de 2 en 2

para 2^2→ 0,4,8,12,…, 24 de 4 en 4

En los formatos debe existir un equilibrio entre rango y precisión: Si aumentamos el numero de bits del exponente aumentamos el rango de los números representables y disminuimos la precisión..La única manera de aumentar rango y precisión, es aumentar el número de bits total de la representación. También es importante la base implícita que se utilice. Si la base es 2el rango es menor pero tiene mayor precisión. Además un desplazamiento del exponente equivale a multiplicar o dividir por 2 luego es equivalente a un desplazamiento de un bit en la mantisa.

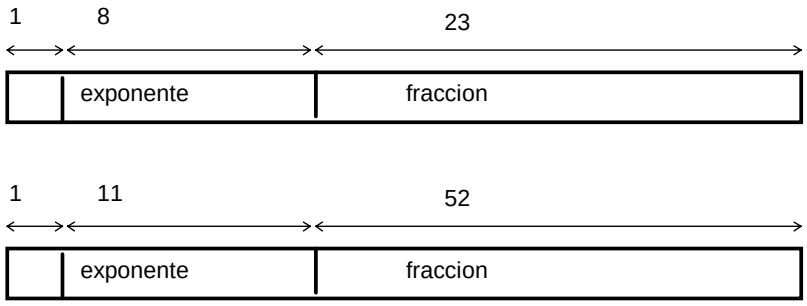
En cambio en base 16 es mayor rango, menor precisión y Un desplazamiento del exponente equivale a multiplicar o dividir por 16 es decir por 2^4 luego equivale a un desplazamiento de 4 bits en la mantisa

En la actualidad muchos computadores usan dos precisiones Precisión simple y Doble precisión

7.12.2 ESTÁNDAR DEL IEEE PARA ARITMÉTICA DE PUNTO FLOTANTE

Su Objetivo es Facilitar la portabilidad de programas de un procesador a otro y facilitar el desarrollo de programas sofisticados orientados al cálculo numérico. Define dos formatos sencillos:

- **SIMPLE:**
  - de 32 bits
    - 8 de exponente
    - 23 de mantisa
    - 1 de signo de la mantisa
    - Base implícita = 2
- **DOBLE**
  - de 64 bits
    - 11 de exponente
    - 52 de mantisa
    - 1 de signo de la mantisa
    - Base implícita = 2



El formato coloca el exponente antes que la mantisa porque simplifica la comparación de los números en coma flotante ya que se puede usar aritmética de comparación de enteros. Dada la codificación en exceso del exponente, el número de mayor exponente es el número mayor; por lo tanto con el exponente almacenado en los bits de mayor peso del formato no importa el contenido de la mantisa.

Además de lo visto incluye dos formatos extendidos que dependen de la implementación en cada máquina. Para implementarlos utiliza Bits adicionales →con lo que se consigue mayor precisión y mayor rango.

- **SIMPLE AMPLIADO**
  - Palabra >= 43
    - Exponente >=11
  - Mantisa >=31
- **DOBLE AMPLIADO**
  - Palabra>=79
    - Exponente>=15

* Mantisa >=63

Este se usa para cálculos intermedios, con lo que se evita los redondeos excesivos y Evita que se aborte en un cálculo intermedio por overflow, cuando el resultado podría incluirse en el tamaño final.

Una ventaja del formato sencillo, es que no tiene la penalización de tiempo en la que incurren las operaciones con mucha precisión.

Además de todo lo anterior conviene saber que algunos patrones de bits tienen un significado especial. Los valores extremos del exponente (todo 0's o todo 1's), definen valores expeciales. El caso general de representación es el siguiente

* los exponentes que se encuentran en el rango[000—01] y [111---10] Están representando números normalizados , no nulos en coma flotante.
* Exponentes se representan en **exceso a 127** (simple) o **1023(doble**)- es decir utiliza el exceso a 2^{q-1}-1)
* Rango de exponentes: -126 a 127 (simple) y -1022 a 1023 (doble)
* Mantisa

el primer bit de la izquierda del punto decimal es un 1 implícito 1.bbbbbbb

mantisa efectiva 24 (simple) o 53(doble)

a los bits explícitos bbbbb..bb se les suele llamar fracción

Mantisa = 1+ fracción

Para representar un número en el formato del IEEE se utiliza la siguiente expresión

Nº decimal = $(-1)^S$ (mantisa) 2^{Er}= $(-1)^S$ x(1+fracción) $2^{Ecodificado-desplazamiento}$

ν **CASOS PARTICULARES**

Según que los exponentes y fracciones toman el valor 1 y 0 se dan los siguientes casos particulares

a) Exponente = 0 fracción 0 → -0 o +0
b) Exponente =0 fracción ≠ 0 números desnormalizados. En este caso el bit de la izquierda del punto binario que antes era 1(implícitamente) ahora es cero y el verdadero exponente -126 o -1022 según sea la representación simple o doble el numero es positivo o negativo según el bit de signo.
c) Exponente =1 fracción =0 ±∞
d) Exponente =1 fracción ≠ 0 numero utilizado para señalar excepciones

Características de los números normalizados

* Rango de exponentes -126 a127 o de -1022 a 1023
* El primer bit de la izquierda del punto decimal es un 1 implícito

La tabla que figura a continuación contiene los parámetros del formato IEEE 754.

Parámetros	sencillo	sencillo extendió	doble	doble extendido
palabra	32	>=43	64	79
exponente	8	>=11	11	15

exceso	127		1023	
exp.mx	127	>=1023	1023	16382
exp.min	-126	-1022	-1022	-16382
rango	10^{-38} 10^{+38}		10^{-308} 10^{+308}	
anch mantisa	23	31	52	63
Nº exponentes	254		2046	
Nº fracciones	2^{23}		2^{52}	
Nº valores	$1.98\ 2^{31}$		$1.99\ 2^{63}$	

ν **EJEMPLOS:**

Como transformar un número real en decimal a un número real en binario

Sea 5.625 la parte entera se opera como siempre - mediante divisiones sucesivas -

$5_{10}=101_2$

En la fracción se realizan multiplicaciones por dos sucesivas. Se para cuando la parte fraccionara = 0. La representación binaria son los bits enteras que aparecen en cada multiplicación

0.625x2=1.250→ 1

0.250x2=0.50→0

0.5x2=1.00→ 1 fin

luego $5.625_{10}=101.101_2$

Representación en IEEE754

Para representar 101.101 en IEEE 754 primero hay que normalizar:

$101.101\cdot 2^0=1.01101\cdot 2^2$

Luego la mantisa será 0110100…00

El exponente real = exponente codificado - desplazamiento

2=Ec-127→ Ec=129 → 01000001

(recordar que en IEEE el desplazamiento es 2^{q-1}-1)

Ejemplo de nº real decimal no representable en binario

Debemos recordar que entre 0 y 1 existen infinitos números reales, pero que solo son representables 223 de ellos (en el caso de precisión simple) en la mayoría de las ocasiones habrá que aproximar.

0.08371

0.08371x2=0.16742→0

0.16742x2=0.33484→0

0.33484x2=0.66968→0

0.66968x2=1.33936→1 con 4 bits 0.0001=0.0625

1.33936x2=0.67872→0

0.67872x2=1.35744→1 con 6 bits 0.000101=0.078125

La precisión de los cálculos en coma flotante depende en gran medida de la precisión del redondeo.

7.13 ARITMÉTICA DE PUNTO FLOTANTE

ν **PRINCIPALES PROBLEMAS:**

- **Overflow del exponente**
  - Un exponente positivo excede del Máximo posible
  - Número demasiado grande
  - A veces se representa como +∞ o -∞
- **Underflow del exponente**
  - Un exponente negativo excede del Mx valor negativo permitido
  - Número demasiado pequeño:
  - Puede aproximarse al 0.
- **Underflow de la mantisa**
  - En el proceso de alinear mantisas los dígitos pueden salirse por la derecha de la mantisa:
  - Si existen bits de guarda se podrá redondear
- **Overflow de la mantisa**
  - La suma de dos mantisas del mismo signo puede producir un carry out del bit más significativo
  - Puede solucionarse desplazando la mantisa a la derecha e incrementando el exponente

7.13.1 SUMA Y RESTA.

- Pasos:
  1.- Comprobar los ceros
  2.- Alinear los exponentes.
  3.- Sumar/restar las mantisas.
  4.- Normalizar el resultado.
- Si el formato incluye un bit implícito, éste debe hacerse explícito para la operación.
- Los exponentes y mantisas se suelen guardar por separado y sólo se reunen en el resultado final

ν **Alinear los exponentes**

La alineación consiste en un desplazamiento del punto binario lo que implica también un cambio en el valor del exponente. Se puede alinear el más pequeño o el más grande. **Para alinear el exponente pequeño con el grande**

∗ Incrementar el exponente menor en una unidad equivale a multiplicar La mantisa por dos - cuando la base es dos -

∗ para conservar el valor de la mantisa

∗ Habrá que dividirla por dos que es equivalente a desplazarla a la derecha dividirlo por dos

ejemplo $1.0011 \cdot 2^3$ se alinea con $1.1100 \cdot 2^5$ habrá que incrementar el exponente 3 dos unidades y por lo tanto para no perder el valor de la mantisa habrá que desplazarla dos veces a la derecha:

$$0.010011 \cdot 2^5$$

Para alinear el número grande con el pequeño

∗ Se resta el exponente en una unidad

∗ Desplaza la mantisa a la izquierda

ejemplo alinear el número $1.1100 \cdot 2^5$ con el 1.002^3

Hay que restar 2 unidades del exponente 5 luego hay que hacer dos desplazamientos de su mantisa a la izquierda:

$111.0000 \cdot 2^3$

Puesto que la operación de alineación puede dar lugar a perdidas de dígitos**, debe ser el número más pequeño el que se alinee**, de esta forma los bits que se pierdan serán los de menor importancia. Si la base es 16, los desplazamientos de la mantisa deben ser de 4 bits

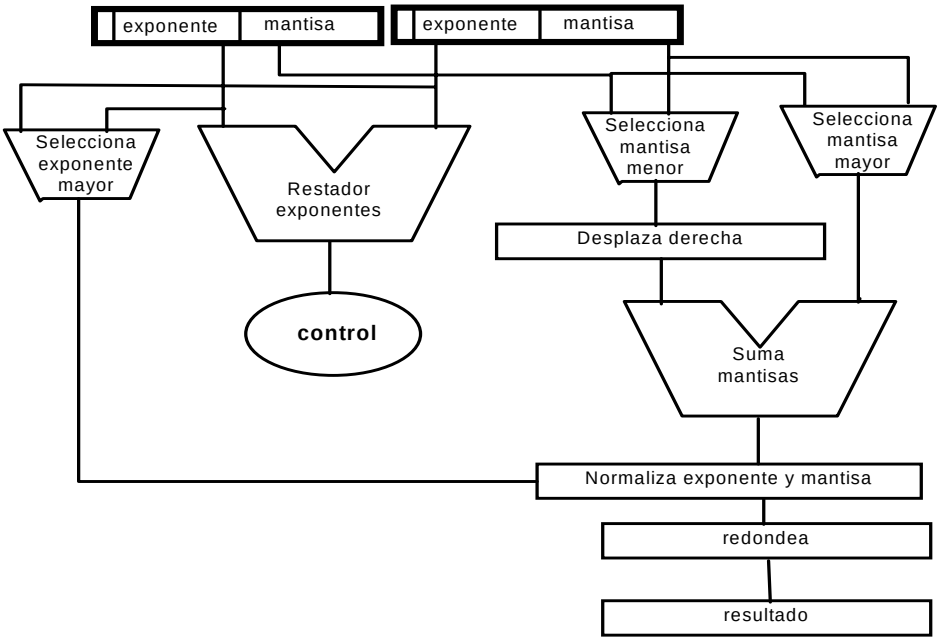
Si el resultado es 0 en la mantisa, el resultado de la operación es el otro número. Si dos números tiene exponentes que difieren mucho, el número menor se pierde.

ν **Sumar/restar las mantisas**

- Se suman las mantisas teniendo en cuenta los signos

- Problema 1
  - El resultado puede ser 0
  - Solución colocar el exponente a = para que la solución sea 0 que es la correcta
- Problema2
  - Overflow de la mantisa:
  - Solución: la mantisa se desplaza a la derecha y el exponente se incrementa
- Problema 3
    - Underflow de la mantisa
    - Solución si existen bits de guarda → redondeo
- Problema 4
  - Overflow y Underflow del exponente ¡¡alto!!

Normalizar el resultado: Desplazar los dígitos de la mantisa a la izquierda hasta que el dígito más significativo sea 1. Como para cada desplazamiento de la mantisa hay que decrementar en una unidad el exponente se puede producir un underflow del exponente. Esto no tiene solución luego se debe parar la operación. La última acción es realizar el redondeo.



7.13.2 MULTIPLICACIÓN Y DIVISIÓN

Mucho más simple que la suma y la resta.

ν **MULTIPLICACIÓN:**

1.- Comprobar si los operandos son 0

- Si uno = 0, el resultado = 0.

2.- Sumar exponentes:

- Tener en cuenta si están almacenados en exceso.
  - En este caso el valor del exceso se debe restar del resultado
  - ejemplo Ec1+Ec2= Er1+D+Er2+D sobra un desplazamiento D que habrá que eliminar
- Puede suceder overflow o underflow del exponente → ¡¡FIN!!

3.- Multiplicar las mantisas teniendo en cuenta los signos:

- Igual que con los enteros
- Se trabaja con magnitud y signo
- Doble longitud que multiplicando y multiplicador.
- problema: Overflow de la mantisa:
  - Solución: la mantisa se desplaza a la derecha y el exponente se incrementa
- Los bits menos significativos se pierden en el redondeo.

4.- Normalización del redondeo: podrían dar lugar a un underflow del exponente.

ν **DIVISIÓN:**

1.- Comprobar los ceros.

1A.- Si dividendo es cero el resultado 0

1B.- Si el divisor es cero

¡¡ERROR!!

Resultado infinito

2.- Se resta el exponente del divisor al del dividendo,

se elimina el exceso del resultado;

se debe volver a sumar

Puede dar overflow o underflow.

4.- División.

5.- Normalización.

6.- Redondeo.

7.13.3 CONSIDERACIONES DE PRECISIÓN:

ν **BITS DE GUARDA:**

Cuando se realizan operaciones en coma flotante, los operadores deben traerse a los registros de la UAL.

∗ Los exponentes y las mantisas se ponen en registros separados.

∗ Los registros que contienen las mantisas tiene mayor longitud que la de la mantisa.

∗ Los bits sobrantes se llaman bits de guarda, se utilizan para no perder precisión durante las operaciones.

- Por ejemplo:

$x = 1.\ 00000\ 2^1$

$y = 1.\ 11111\ 2^0$

Vamos a suponer que hacemos x-y.

Se debe desplazar el número menor, que es y.

$y' = 0.\ 11111\ 2^1$

$x - y = 1.\ 00000\ 2^1 - 0.\ 11111\ 2^1 = 0.\ 00\ 01\ 2^1$

al normalizar $1.0 — 0\ .2^{-22}$

En el desplazamiento a la derecha de "y", se ha perdido un bit de importancia. Si se hubieran tenido bits de guarda:

$x = 1.\ 00 — 000\quad 0000\ 2^1$

$y = 0.\ 11 — 111\quad 1000\ 2^1$

y el resultado de la resta es:

$z = 0.00\ —\ 000\quad 1000 = 1.00\ —\ 000\quad 0000 \times 2^{-23}$

7.14 TRUNCAMIENTO Y REDONDEO

- Generalmente los resultado de operaciones se guardan en registros de mayor tamaño, que el que los contendría finalmente.
- Objetivo

  Conservar en la mayor medida de lo posible la precisión de la operación.
- Efecto:

  Habrá que eliminar parte de los bits que contienen el resultado obtenido.
- **TRUNCAMIENTO:**
  - También llamado redondeo por defecto
  - Consiste en eliminar los bits de la derecha que no caben en la representación.
  - Defecto:
    - el error inducido en los resultados, es siempre por defecto.
    - El error acumulado crece rápidamente.
    -

ν **REDONDEO**:

El estándar IEEE propone 4 aproximaciones:

1.- Redondeo al más cercano,

2.- Redondeo a $+\infty$,

3.- Redondeo a $-\infty$,

4.- Redondeo a 0.

Solo estudiamos el redondeo al más cercano

ν **REDONDEO AL MÁS CERCANO:**

- Es el que utiliza por defecto el estándar.
- Se representa el valor representable más próximo al valor del resultado obtenido.
- Suponer que aparte de los 23 bits representables se han utilizado 5 bits de guarda
- Existen tres casos:

Bits de guarda= 1bbbb siendo como poco uno de los b's=1

La fracción > 0.5

Se redondea por exceso sumando 1 al ultimo representable

Bits de guarda 0bbbb

La fracción < 0.5

Redondeo por defecto --> truncar

Bits de guarda=10000 --> fracción =0.5

Forzar el nº par en el representable:

Si el representable acaba en 1 se le redondea por exceso (suma 1)

Si el representable es 0 se le deja como está

8 ESPECIFICACIÓN DE LA UNIDAD DE CONTROL

8.1 INTRODUCCIÓN

Definición: La unidad de control es la parte de la unidad central de proceso (CPU), que controla :

Las operaciones de la CPU, incluyendo las operaciones de la ALU

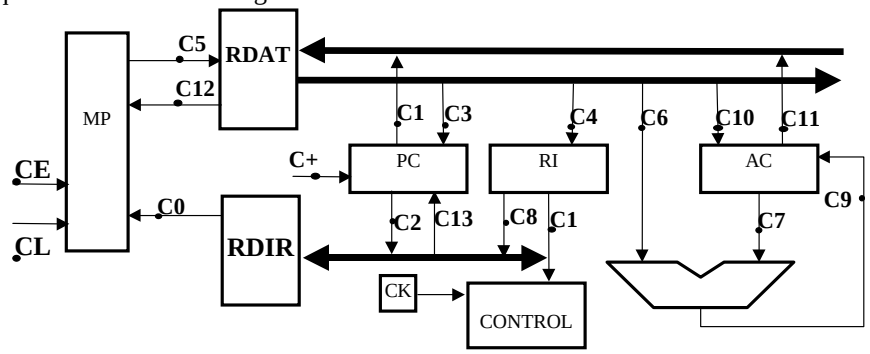
- Los movimientos de datos en el interior de la CPU
  Los movimientos de datos entre la CPU y la memoria
- Los movimientos de datos y de señales de control con los Interfaces exterior

En este tema vamos a ver como se definen las señales que genera la UC en cada ciclo de reloj para que el computador funcione. Para ello hay que estudiar, el camino de datos de la CPU, Las μoperaciones que debe ejecutar la CPU para cada fase de una instrucción y las señales de control necesarias para realizar cada μoperación

La Unidad de Control es un sistema secuencial que sirve para indicar qué señales se activan y cuando se activan. Para especificar un sistema secuencial se necesita definir:

- Las señales de entrada al sistema como son el reloj, los contenidos del Registro de Instrucciones, los Flags o las interrupciones.
- Las señales de salida
- Los estados del sistema

Además se debe indicar como se relacionan estos tres conjuntos. Para poder trabajar vamos a definir un camino de datos que utilizaremos a lo largo del estudio.



Siendo el RDAT el registro de datos de memoria y RDIR el registro de direcciones de memoria. Haremos ademas las siguientes suposiciones: la función de las señales de control es abrir caminos entre diferentes módulos y no existen señales de capacitación de carga de los registros. Para esta última existen tres excepciones, La C+ de controla el autoincremento del PC, CE que controla la escritura en la memoria y CL que controla la lectura de la memoria.

8.2 LAS μOPERACIONES

Ya se vio en su día que una instrucción se divide en fases. La división general en fases de una instrucción es:

- Búsqueda (Fetch) que carga en el registro de instrucciones la instrucción a ejecutar y actualiza el contador de programa
- Cálculo de las direcciones de los operandos , es decir generación de las direcciones efectivas
- Búsqueda de los operandos, es decir lectura del dato de la memoria o del banco de registros.
- Ejecución de la operación
- Cálculo de la dirección de los resultados
- Almacenamiento de los resultados
- Interrupción.

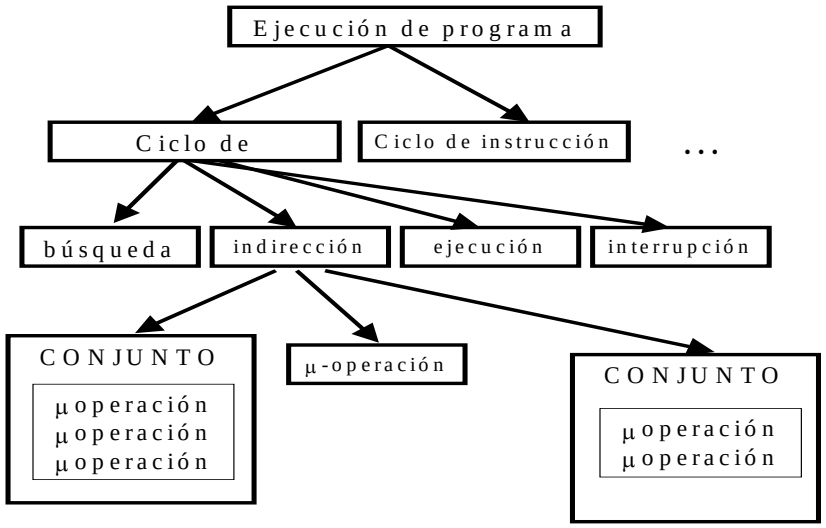
Una fase de una instrucción puede necesitar varios ciclos de reloj para ejecutarse. Para el estudio de la UC, las fases de cálculo de las direcciones de los operandos y de las direcciones de los resultados son

idénticas. Nosotros trabajaremos con el modo de direccionamiento indirecto que, por ser algo más complejo, sirve mejor como ejemplo.

ν **μOPERACIONES**

En algunas ocasiones la descomposición de la instrucción en fases no es suficiente para especificar la unidad control, y esto implica que cada fase debe descomponerse a su vez en nuevos pasos. A cada uno de estos pasos se le llama μoperación y se le define como un movimiento elemental necesario para implementar parte de la fase

Las μoperaciones de una fase se pueden agrupar en conjuntos. Las μoperaciones de un conjunto se pueden ejecutar simultáneamente y los conjuntos de μoperaciones se deben ejecutar en serie y según un orden preestablecido. El número de conjuntos de μoperaciones necesarios para implementar una fase determina el número de ciclos de reloj necesarios para implementarla y viene determinado por el camino de datos diseñado.



A continuación se estudian las μoperaciones en que se dividen lasa fases de:

- Búsqueda
- Ejecución
- Indirección
- Interrupción

Para realizar el estudio se debe tener muy presente el camino de datos sobre el que se trabaja. En este camino de datos deben estar perfectamente especificadas todas las señales que lo controlan. La especificación de la unidad de control consiste en especificar un sistema secuencial, de manera que indique las señales que debe generar la unidad de control en cada ciclo de reloj.

▪ **FASE DE BUSQUEDA:**

Es la primera fase que realiza una instrucción. Se puede descomponer en las dos siguiente microoperaciones

- Cargar el R.I. con una instrucción leída de la memoria principal.
- Actualizar el contador de programa

Vamos a ver sobre el camino de datos que movimientos elementales realizan estas dos microperaciones:

T_1: RDIR ← <PC>

T_2: RDAT ← <MEMORIA [RDIR]>; PC ← <PC> + 1

T_3: RI←<RDAT>

Vamos a describir en que consiste cada microoperación. En T1 se copia el contenido del registro PC, es decir la dirección de la siguiente instrucción en el registro RDIR. En T2 se lee la instrucción de la

memoria., para ello la dirección se coloca en el bus de direcciones. Para conseguirlo la UC coloca la señal de control CL en el bus de control. El resultado de la lectura aparece en el bus de datos y se copia en el RDAT. Como hay que incrementar el contador y esta acción no interfiere con la anterior se ejecuta a paralelo, es decir en el mismo ciclo de reloj. Esto se puede hacer porque las dos microperaciones utilizan diferentes caminos y módulos para llevarse a cabo. El ultimo paso, T3 ,es mover el contenido de RDIR al RI.

¿Porque se agrupan en tres ciclos? la μoperación RDAT←<MEMORIA[RDIR]> no se puede llevar a cabo hasta que la operación RDIR ← <PC> no ha concluido, puesto que necesita tener la dirección de memoria correctamente cargada en RDIR.

La μoperación RI←<RDAT> no se puede llevar a cabo hasta que no se ha ejecutado RDAT ← <MEMORIA [RDIR]>, puesto que necesita que el dato leído esté correctamente cargado es RDAT. En cuanto a la μoperación PC←<PC>+1 se podría ejecutar en un ciclo aparte, con lo que la fase se realizaría en cuatro ciclos de reloj, pero generalmente se intenta reducir el número de ciclos de reloj, para aumentar el rendimiento del computador. Para conseguir esto es necesario estudiar si esta μoperación se puede realizar simultáneamente con alguna de las otras μoperaciones. Según está definida la estructura, el incremento del PC no se podría realizar en el primer ciclo, puesto que se corre el peligro de cargar una dirección falsa en RDIR. En cambio no hay ningún problema en incluirla en el ciclo T_2 o en el T_3

Conclusiones: la fase de búsqueda consta de 3 ciclos de reloj y 4 μoperaciones. Cada μoperación produce movimiento de datos a, o, desde registros. Mientras que un movimiento no interfiera con otro se pueden llevar a cabo en el mismo paso. De lo anterior se pueden concluir dos reglas para agrupar μoperaciones en un mismo ciclo. La primera de ellas es que se debe seguir la secuencia de eventos correcta es decir respetar las dependencias temporales entre las microoperaciones, y la segunda es que se deben evitar los conflictos entre microoperaciones, es decir no se debe intentar escribir y leer el mismo registro en la mismo instante de tiempo. El resultado sería impredecible. No se debe utilizar simultáneamente el mismo camino de datos, no se debe utilizar simultáneamente la misma unidad funcional

NOTA: recordar que cada conjunto de μoperaciones se ejecuta en un sólo ciclo de reloj. Por lo tanto cuanto más μoperaciones se puedan ejecutar simultáneamente, menos ciclos necesitara una instrucción para ejecutarse y más rápido será el computador.

- **CICLO DE INDIRECCION**

El modo de direccionamiento usado es el indirecto a memoria, es decir el registro de instrucciones RI contiene una dirección de memoria y en esa dirección de memoria se encuentra la dirección del dato. La forma idónea de llevar a cabo esta fase sería cargar la dirección de memoria en el registro RDIR. Cargar el contenido de la memoria en el registro RDAT. Y por último cargar el contenido de RDAT sobre el registro RDIR con lo que ya tenemos la dirección del dato en el registro de direcciones.

Pero observando el camino de datos vemos que esto no es posible puesto que no existe un camino entre RDAT y RDIR. La solución que se propone es modificar el campo dirección del Registro de Instrucciones cargando la nueva dirección. Las μoperaciones correctas son:

T_1: RDIR ← <RI. DIRECCIÓN>

T_2: RDAT ← <MEMORIA [RDIR]>

T_3: RI. DIRECCION ← <RDAT>

Ninguna de las μoperaciones se puede ejecutar simultáneamente debido a las dependencias temporales.

ν **CICLO DE INTERRUPCIÓN:**

Se lleva a cabo al terminar el ciclo de ejecución. Consiste en comprobar si está activada alguna interrupción. Si no es así se sigue la ejecución normal del programa. En caso que exista una petición de interrupción, se suspende la ejecución del programa y se ejecuta una subrutina de interrupción. En el ejemplo se trata el caso en que la interrupción esta solicitada. La secuencia de μoperaciones:

T_1: RDAT ← <PC> (porque la información va a ser machacada en el siguiente ciclo)

T_2: RDIR ←Dirección de salvaguarda ; PC ← Dirección de la subrutina de interrupción

T_3: MEMORIA[RDIR] ← <RDAT>

La μoperación RDIR ←Dirección de salvaguarda se podría haber ejecutado también en el ciclo T1 y la μoperación PC ← Dirección de la subrutina de interrupción se podría haber incluido en el ciclo T3 quedando la fase como sigue

T_1: RDAT ← <PC>; RDIR ←Dirección de salvaguarda

T_2: PC ← Dirección de la subrutina de interrupción; MEMORIA[RDIR] ← <RDAT>

- **CICLO DE EJECUCIÓN**

Los ciclos de búsqueda, indirección e interrupción son simples y predecibles, cada uno implica una pequeña secuencia fija de μoperaciones. Esto no es cierto para el ciclo de ejecución. Si un computador tiene N códigos de operación diferentes, existen N secuencias diferentes. Se van a ver varios ejemplos.

a) **ADD AC, X**

Es la suma del contenido del registro acumulador y del dato contenido en la dirección X de memoria, siendo X=RI.dirección. Como se dijo en su momento, lo primero es buscar la dirección de los operandos, a continuación traer los operandos y operarlos. Por ultimo se deben generar la dirección del resultado y guardar el resultado. Según se ve en la instrucción, ésta solo tiene dos operandos y uno de ellos es el acumulador luego el resultado se debe guardar en este acumulador. Las μoperaciones agrupadas por ciclos son las siguientes:

T_1: RDIR ← <RI .dirección> busca la dirección

T_2: RDAT ← <MEMORIA[RDIR]> busca el operando

T_3: AC← <AC> + <RDAT> opera y guarda el resultado

b) **ISZ X (**Increment and Skip if Zero)

El dato almacenado en la posición X de memoria se incrementa en una unidad y se almacena en la misma posición. Si el resultado de este incremento es cero, se salta una instrucción del programa. Lo primero es buscar la dirección del operando, que en este caso es un direccionamiento directo RI.Dirección. A continuación se opera y se guarda el resultado en la misma dirección. Por último se comprueba si el resultado es cero. Las μoperaciones son:

T_1: RDIR ← <RI.dirección); búsqueda de la dirección

T_2: RDAT ← <MEMORIA[RDIR]>; búsqueda del operando

T_3: AC ← <RDAT> + 1; operación

T_4: RDAT ←<AC >

T_5: MEMORIA[RDIR]← <RDAT> ;se guarda el resultado en memoria

if <RDAT = 0> then PC ←<PC>+1

Darse cuenta que el test también puede implementarse como una μoperación a parte

b) **BSA X (**Branch and Save Address instruction).

Esta instrucción es un salto a subrutina La dirección de la instrucción que está a continuación de BSA se guarda en X y se ejecuta a partir de X+1. Esta se encuentra almacenada en la posición X, y esta primera posición de la subrutina se utiliza para guardar la dirección de vuelta. Las μoperaciones serán las siguientes:

T_1: RDIR ← <RI. Dirección>; se guarda la dirección X de subrutina, que esta almacenada en RI, en el RDIR para poder saltar posteriormente a dicha subrutina; RDAT ← <PC> se lleva el contador de programa al RDAT (para poder salvarlo posteriormente en la posición X

T_2: PC ← <RI. Dirección> se carga la dirección X de subrutina en PC; Memoria<X> ← <RDAT> se guarda en la posición X de memoria la dirección de vuelta

T_3: PC ← <PC> + 1 se incrementa el PC para que empiece a ejecutarse la instrucción X+1.

NOTA: cuidado con los diagramas de estados, no son iguales para todas las unidades de control. Por ejemplo en las unidades cableadas la fase de decodificación sobra. Si tengo como entradas el código de operación, y el ciclo de reloj en la que se está ejecutando, COMO en el ejemplo que se pone más adelante, se puede eliminar la fase de decodificación.

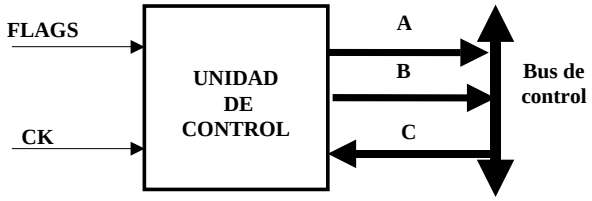
En las unidades de control microprogramadas, esa fase tiene sentido si se implementa tal y como están implementadas en el tema. Es decir el multiplexor está antes del micropc y por lo tanto es obligatorio cargar este registro antes de poder leer la micromemoria. Esto produce un ciclo más en todas las instrucciones. Por ejemplo tras la fase de búsqueda el código de operación tras haber pasado por la pla no puede atacar directamente al bus de direcciones de la micromemoria. Esto se podría mejorara con solo colocar el mux detrás del micropc, es decir con una estructura similar a l a que estamos viendo todos los días para los caminos de datos

8.3 LAS SEÑALES DE CONTROL

Una vez descompuesta una instrucción máquina en un conjunto de μoperaciones, se debe determinar con exactitud las señales de control que la U.C. genera para que se ejecute la instrucción correctamente. La unidad de control realiza dos tareas básicas:

- **Secuenciamiento**: provoca que la CPU realice las μoperaciones en la secuencia adecuada
- **Ejecución**: que se ejecute cada μoperación del programa.

Para implementar una unidad de control hace falta determinar unas especificaciones internas que son la descripción de la lógica que permite el secuenciamiento y ejecución de las μoperaciones (esto se verá en los temas siguiente) y unas especificaciones externas que es el conjunto de entradas que permitan determinar el estado del sistema en cualquier instante de tiempo y un conjunto de salidas que permitan controlar el sistema.



siendo A las señales de control internas de la C.P.U., B señales de control al bus de sistema y C las señales de control del BUS de sistema.

Las entradas a la unidad de control son las siguientes. La señal de reloj (CK) que sirve para “marcar el tiempo”. La UC ejecuta un conjunto de μoperaciones cada ciclo. Esto se conoce como tiempo de ciclo del procesador o tiempo de ciclo del reloj. Algunos de los contenidos del registro de instrucciones (RI) que contiene el código de operación de la actual instrucción y determina las μoperaciones del ciclo de ejecución. Los FLAGS que necesita la U.C. para determinar el estado de la CPU Suelen guardar datos sobre las operaciones ejecutadas por la UAL. Por último, la unidad de control puede generar las señales de control de entrada que son señales de interrupción y señales de reconocimiento

Las salidas de la unidad de control son las señales de control de la CPU que generan los movimientos de datos y las que activan operaciones de la ALU. Las señales de control que se envían al bus que contienen información para la memoria y para los módulos de E/S.

8.4 ESPECIFICACIÓN DE LAS SEÑALES DE CONTROL DE CADA μOPERACIÓN

Una vez se tiene definido el camino de datos, las conexiones con memoria y el módulo de E/S y el conjunto de μoperaciones de cada instrucción es sencillo ver cuales son las señales de control que se necesitarían para llevar a cabo cada μoperación. Ejemplos de señales de control son:

- Lectura/escritura de memoria
- Carga de registros
- Control de puertas triestate
- Operaciones de la UAL
- Selección de entradas en mux

A continuación vemos como encontrar las señales de control a partir de las μoperaciones. Se debe estudiar los caminos de comunicación que se establecen entre los módulos para poder realizar la μoperación.

Para ello hay que buscar las señales de control de puertas triestate y de multiplexores. Además se debe estudiar los registros que interviene en la μoperación determinando los que se deben cargar y generar la señal de capacitación de la carga de registros. Por último se debe realizar un estudio las operaciones aritmético/lógicas. Para que las señales de control tengan sentido es necesario que la UC recuerde en que fase de la instrucción nos encontramos

- **Búsqueda:**

T_1: RDIR ← <PC>: C_2

T_2: RDAT - MEMORIA: C_5, $C_{L;}$ PC ← <PC> + 1: C+

T_3: RI ← <RDAT> : C_4

- **Indirección:**

$T_{1:}$ RDIR ← <RI.dirección>: C_8

T_2: RDAT ←<MEMORIA[RDIR]>:C_5, C_R

T_3: RI. DIRECCION ← <RDAT>: C_4

- **Interrupción:**

T_1: RDAT ← <PC>: C_1

T_2: RDIR ← dirección de salva guarda; PC ← Dirección de la subrutina de tratamiento de interrupción

T_3: MEMORIA[RDIR]← <RDAT>: C_{12}, C_E,C0

Atención: como no está definido el camino de datos completamente, hay señales de control que faltan.

- **ADD AC,X**

T_1: RDIR ← <RI .dirección> : C_8

T_2: RDAT ← <MEMORIA[RDIR]>: C_L, C_5

T_3: R1← <R1> + <RDAT> : C_9,C_6,C_7

- **ISZ X**

T_1: RDIR ← <RI.dirección>:C_8

T_2: RDAT ← <MEMORIA[RDIR]>:C_L, $C_{5,}C_0$

T_3: AC ← <RDAT> + 1 :C_6,C_9

T_4: RDAT ←<AC >:C_{11}

T_5: MEMORIA[RDIR]← <RDAT>: C_E ,$C_{0,}C_{12}$

if <RDAT = 0> then PC ←<PC>+1: C+

- **BSA X**

T_1: RDIR ← <RI. Dirección> :C8; RDAT ← <PC>: C1

T_2: PC ← <RI. Dirección> : C13; MEMORIA[RDIR]← <RDAT> : C0, C12, CE

T_3: PC ← <PC> + 1: C+

En el ejemplo que viene a continuación se vera como se puede unificar todo lo visto hasta ahora, especificando la unidad de control mediante un diagrama de estados, que no es nada mas que la representación de una máquina de estados finitos.

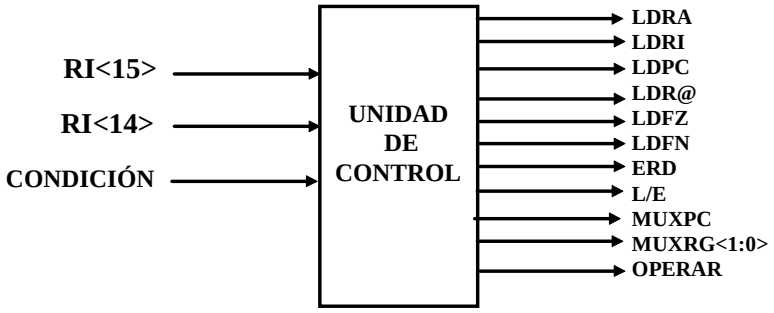
8.5 EJEMPLO

En este apartado se ve un ejemplo de como estudiar las especificaciones de una unidad de control a partir del camino de datos de la máquina simple vista en el tema de camino de datos. La unidad de control es un sistema secuencial cuya misión es determinar el orden en que los distintos elementos de la unidad de proceso deben operar para que se ejecute una instrucción. La forma de determinar este orden es activando las señales de control del camino de datos que son las siguientes

- **LDRA** capacitación del registro acumulador
- **LDRI** capacitación del registro de instrucciones
- **LDPC** capacitación del contador de programa
- **LDR@** capacitación del registro auxiliar de direcciones

- **LDFZ** capacitación del flag cero
- **LDFN** capacitación del flag de signo
- **ERD** capacitación de escritura en el banco de registros
- **L/E** señal de escritura lectura de la memoria
- **MUXPC** selecciona la dirección de memoria
- **MUXRG<1:0>** selecciona la dirección del banco de registros
- **OPERAR** selecciona el tipo de operación de la ALU

La unidad de control tiene como entradas el código de operación y el resultado de evaluar la condición de salto, y actúa sobre los elementos de la unidad de procesos: permitiendo la carga de registros, indicando si la unidad aritmético lógica debe operar e indicando a la memoria el tipo de operación que debe realizar.

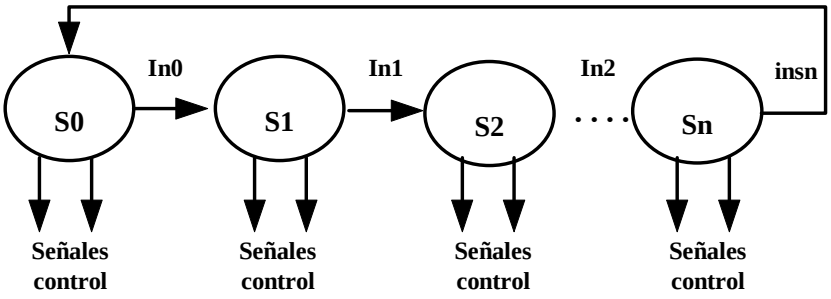


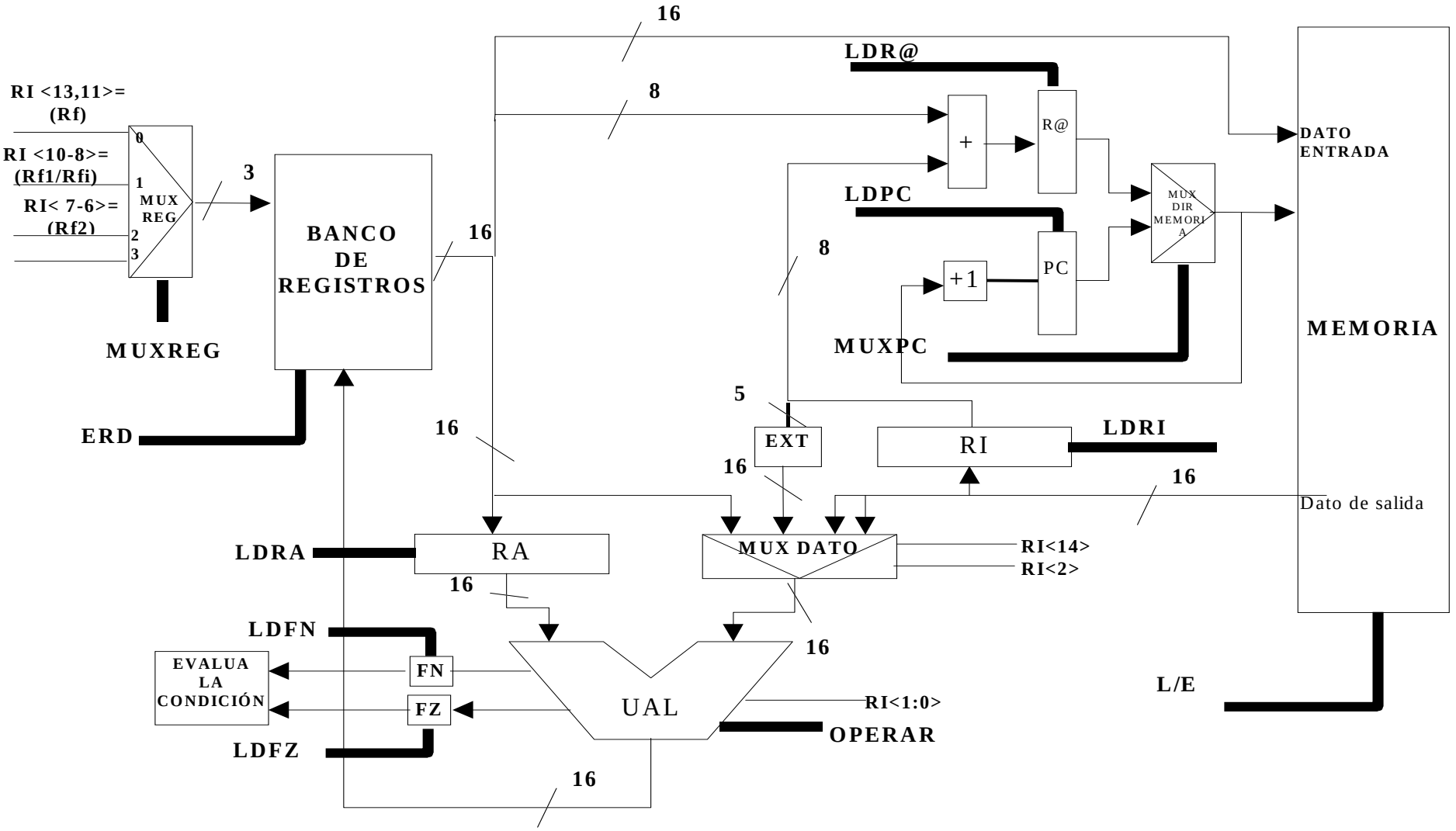
De manera general la ejecución de una instrucción se divide en varias fases:

- Fase de búsqueda de la Instrucción
- Fase de descodificación
- Búsqueda de operando y evaluación de condiciones de salto
- Fases de ejecución de la instrucción

Cada fase se divide en un conjunto de μoperaciones y para llevar a cabo una μoperación se debe activar alguna de las señales de control que se han visto. En función del camino de datos implementado habrá operaciones que se puedan realizar simultáneamente, cuando no utilizan los mismos buses ni las mismas unidades funcionales y habrá operaciones que se deben realizar en serie, cuando utilizan los mismos buses o las mismas unidades funcionales.

El número de conjunto de operaciones que se deben realizar en serie para implementar una fase determina el número de ciclos de reloj necesarios para ejecutar la fase. El número de ciclos de una instrucción es la suma de todos los ciclos necesarios para ejecutar las cuatro fases ya nombradas. En las siguientes secciones se estudian las μoperaciones, el número de ciclos, el grafo de estados y las señales de control que tiene que generar la unidad de control para implementar cada instrucción del computador. Para interpretar los grafos de estado debemos saber que los círculos representan los estados, las flechas entre círculos indican las relaciones entre estados , y los códigos sobre las flechas indican la información de entrada a la unidad de control.


S0
In0
S1
In1
S2
In2
Sn
insn
Señales control
Señales control
Señales control
Señales control


16
LDR@
8
RI <13,11>=
(Rf)
RI <10-8>=
(Rf1/Rfi)
RI< 7-6>=
(Rf2)
0
1
2
3
MUX
REG
3
BANCO
DE
REGISTROS
16
+
R@
DATO
ENTRADA
LDPC
8
MUX
DIR
MEMORIA
+1
PC
MEMORIA
MUXREG
MUXPC
5
ERD
16
EXT
RI
LDRI
16
16
Dato de salida
LDRA
RA
MUX DATO
RI<14>
RI<2>
16
LDFN
16
EVALUA
LA
CONDICIÓN
FN
FZ
UAL
RI<1:0>
L/E
OPERAR
LDFZ
16

8.5.1 FASES COMUNES

Para este procesador existen dos fases comunes a todas las instrucciones, la fase de búsqueda (fetch) y la fase de descodificación.

FASE DE BÚSQUEDA

En esta fase se realizan las siguiente operaciones

RI←m[PC]

PC←<PC>+1

En el camino de datos se puede ver con facilidad que estas dos operaciones no utilizan los mismos módulos en ningún caso y por lo tanto se pueden llevar a cabo en el mismo ciclo de reloj.

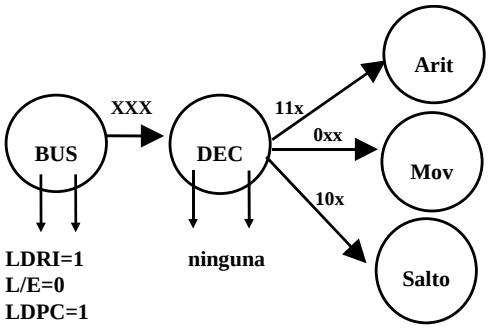
Nota: Conviene recordar que los registros se actualizan en los flancos de subida de la señal de reloj. Las señales que llegan desde la unidad de control a los registros son señales de capacitación.

FASE DE DESCODIFICACIÓN

Durante la fase de descodificación la lógica de la unidad de control evalúa el código de operación de la instrucción con el objeto de decidir cuales son las siguientes acciones a realizar. El código que corresponde a cada clase de operación es el siguiente:

00 instrucciones LOAD
01 instrucciones STORE
10 instrucciones de SALTO
11 instrucciones aritmético-lógicas

Durante esta fase la unidad de proceso no debe realizar ningún tipo de operación. El contenido de todos los registros de la unidad de proceso se preserva durante esta fase poniendo a cero sus señales de capacitación de carga.



8.5.2 INSTRUCCIONES ARITMÉTICO LÓGICAS

Por simplicidad el procesador del computador ha sido diseñado de modo que todas las instrucciones aritmético lógicas se ejecuten siguiendo las mismas secuencias de operaciones, tanto si se opera con dos operandos almacenados en registros, con un operando en registro y otro inmediato o con un sólo operando. Las dos operaciones en que se dividen las instrucciones aritmético lógicas son las siguientes:

1. Búsqueda del primer operando que se encuentra en el banco de registros
2. Búsqueda del segundo operando, ejecución de la instrucción y almacenamiento en los registros del resultado.

BÚSQUEDA DEL PRIMER OPERANDO

La operación a ejecutar es RA←RF1 es decir se carga en RA el contenido del registro RF1 del banco de registros cuya dirección proviene del registro de instrucciones. Para ello hay que activar la señal de capacitación del registro RA. En el caso de la instrucción ASR (desplazamiento aritmético a la derecha)

este primer paso se realiza para homogeneizar el tratamiento de todas las instrucciones aritmético lógicas. La información que se carga en el registro RA en este caso no tiene importancia porque no se utiliza.

BÚSQUEDA DEL SEGUNDO OPERANDO Y EJECUCIÓN

Según que tipo de instrucciones aritmético lógicas se quieran ejecutar así serán las operaciones a realizar.

- Para las instrucciones con dos operandos en el banco de registros las operaciones son:

  RD←RA OP RF2, FZ, FN

- Para las instrucciones con un operando inmediato las operaciones son:

  RD←RA OP EXT(RI<7:3>), FZ, FN

Siendo EXT una operación auxiliar de extensión de signo necesaria para transformar un número en complemento a dos codificado con 5 bits a un número en Ca2 codificado a 16 bits.

- Para las instrucciones de desplazamiento:

  RD←Rf2>>1, FN,FZ

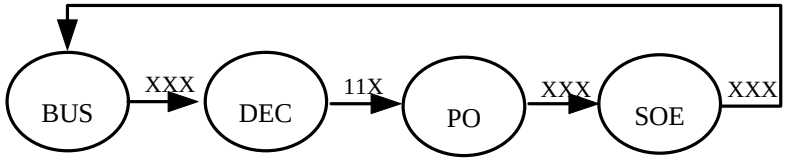
El tipo de operación que se quiere ejecutar no viene explicitado en el campo código (bits 15 y 14) sino en le campo operación (bits 1 y 0). La señal que proporciona la unidad de control en estos casos es **OPERAR** que debe estar activada a uno para que la operación aritmético lógica se realice.

DIAGRAMA DE ESTADOS DE LAS OPERACIONES ARITMÉTICO - LÓGICAS

Para interpretar los grafos de estado debemos saber que los círculos representan los estados , las flechas indican la relaciones entre estados y los códigos sobre las flechas indican la información de entrada a la unidad de control es decir RI<15:14> y condición.

Los cuatro estados en los que se divide la ejecución de una instrucción aritmético lógica son:

1. Búsqueda de la instrucción (**BUS**)
2. Descodificación (**DEC**)
3. Búsqueda del primero operando (**PO**)
4. Búsqueda del segundo operando y ejecución (**SOE**)

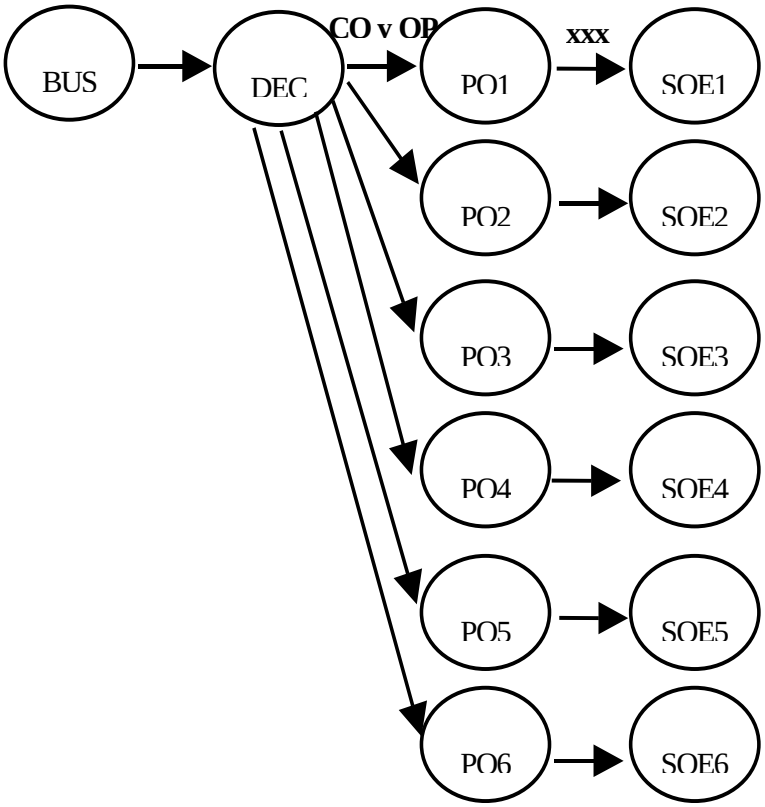


Conforme a este diagrama de estado y al camino de datos las señales de control que deberían activarse son las siguientes:

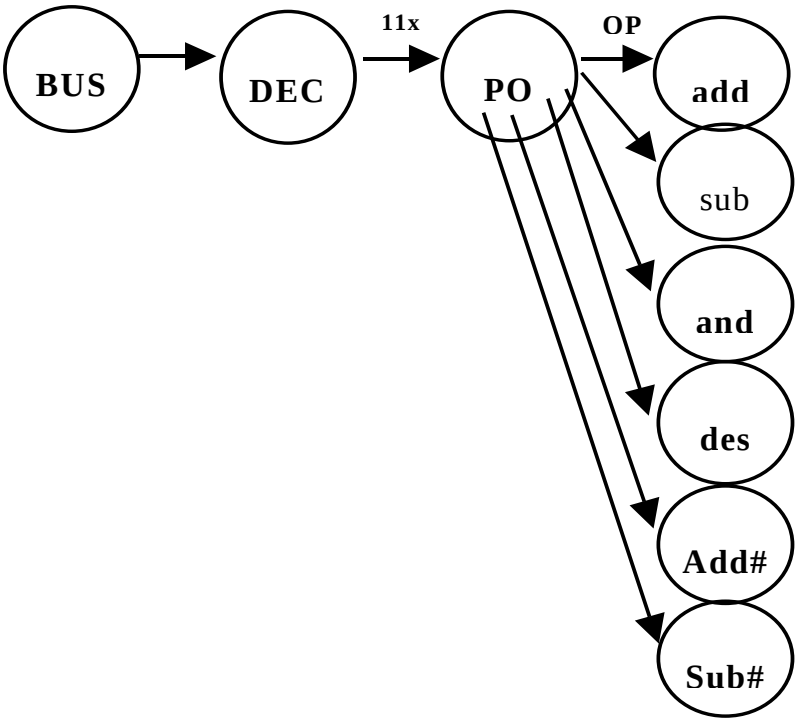
señales de control	BUS	DEC	PO	SOE
LDRA	0	0	1	0
LDRI	1	0	0	0
LDPC	1	0	0	0
LDR@	0	0	0	0
LDFZ	0	0	0	1
LDFN	0	0	0	1
ERD	0	0	0	1
L/E	0	0	0	0
MUXPC	0	X	X	X
MUXREG<1:0>	X	X	1	2
OPERAR	X	X	X	1

NOTA:El diagrama de estados que especifica las instrucciones aritmético -lógicas sale tan sencillo debido a dos decisiones tomadas en la fase de diseño de la estructura: se ha buscado la regularidad es decir todas las instrucciones tienen dos ciclos: uno de búsqueda del primer operando y otro búsqueda del

segundo operando y ejecución; y el código de operación y la operación aritmético lógica están separadas Si estas decisiones no se hubieran tomado, posiblemente nos hubieramos vistos obligados a utilizar una rama del diagrama de grafos por cada instrucción con lo que el grafo total hubiera quedado como se ve a continuación.



Es posible que este grafo se pudiera reducir al de la siguiente figura, suponiendo la búsqueda del primer operando común para todos, pero la complejidad seguiría siendo superior al utilizado por nosotros:



8.5.3 INSTRUCCIONES DE ACCESO A MEMORIA

Existen dos instrucciones de acceso a memoria diferentes:

- **LOAD** lectura de memoria
- **STORE** escritura en memoria

En ambos casos se utiliza un registro del banco de registros mas un campo de la propia instrucción para calcular la dirección de memoria a la que se accede. Estas dos instrucciones realizan dos operaciones cada una, el cálculo de la dirección de memoria y posterior almacenamiento en el registro auxiliar de direcciones R@; Acceso a memoria

- Cálculo de la dirección de memoria:

R@←Ri<7:0>+RI<7:0>

Donde Ri es el registro índice especificado por el RI y RI<7:0> son los siete bits menos significativos del registro de Instrucciones.

- Acceso a memoria

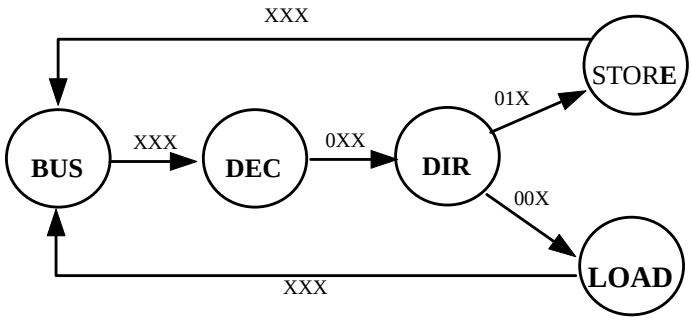
Depende si la operación es load o es store:

- **STORE** (Escritura en memoria) M[R@]←RF
- **LOAD** (Leer la memoria y cargar el contenido en el banco de registros) RD←M[R@], FN, FZ

Conviene recordar que el camino de datos de lectura de memoria pasa por la entrada B de la unidad aritmético lógica.

DIAGRAMA DE ESTADOS

En definitiva las operaciones de load y store tiene el siguiente diagrama de estados



Donde BUS y DEC son los estados comunes a todas las instrucciones, DIR es el estado en el que se calcula dirección de memoria a la que se quiere acceder y LOAD y STORE son los dos estados de acceso.

Fijándose en la figura del camino de datos se puede ver que la tabla de señales de control que se deben activar es la siguiente:

señales de control	dir	store	load
LDRA	0	0	0
LDRI	0	0	0
LDPC	0	0	0
LDR@	1	0	0
LDFZ	0	0	1
LDFN	0	0	1
ERD	0	0	1
L/E	0	1	0
MUXPC	X	1	1

MUXRG<1:0>	1	0	X
OPERAR	X	X	0

8.5.4 INSTRUCCIONES DE SALTO

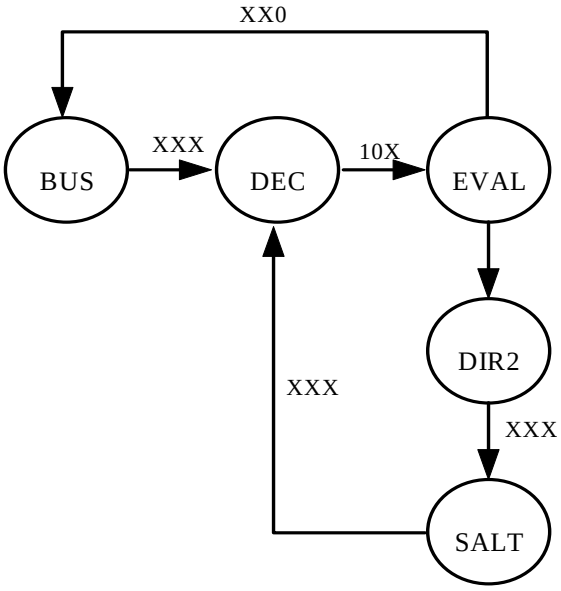
Todas las instrucciones de salto se ejecutan de forma similar. Una vez descodificada, la instrucción de salto tiene dos fases bien diferenciadas: la de evaluación de la condición de salto y en caso de realizar el salto el cálculo de la nueva dirección en caso contrario salta a la fase común de fetch.

La evaluación de la condición de salto se realiza mediante un módulo combinacional implementado en el camino de datos. Este módulo genera un único bit que indica exclusivamente si se realiza el salto o no. La fase de calculo de la nueva dirección sólo se realiza en caso que si haya salto y consiste en calcular la dirección y ejecutar el salto propiamente dicho.

primera fase R@←RI<7:0>

segunda fase PC←R@+1; RI←M[R@]

DIAGRAMA DE ESTADOS



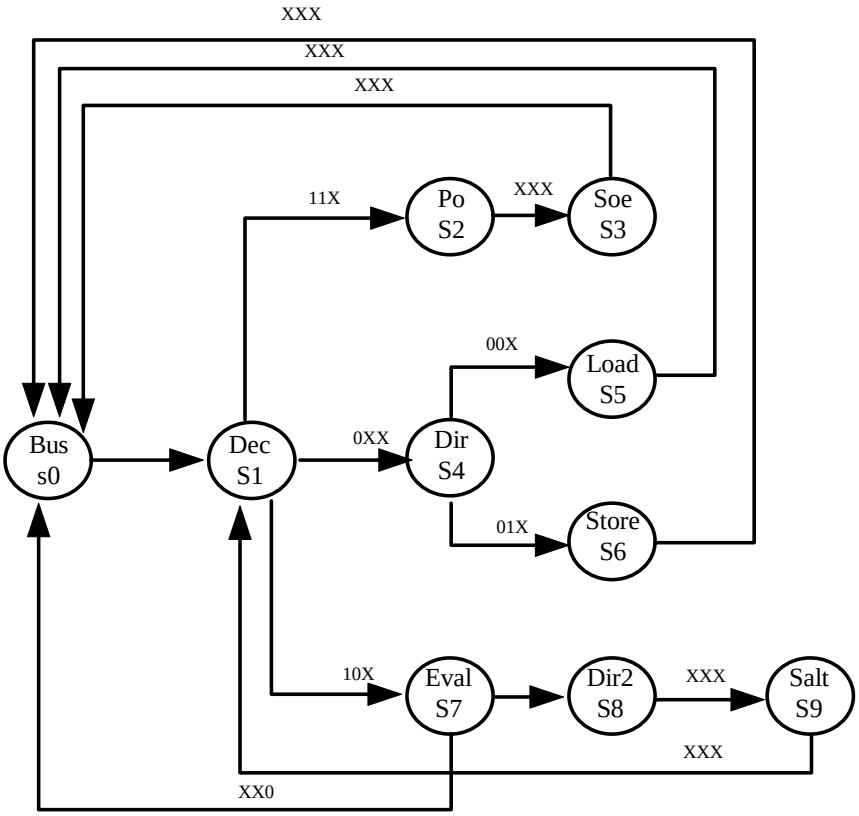
Donde **EVAL** es el estado en el que se evalúa la condición de salto, **DIR2** es el estado en el que se calcula la nueva dirección y **SALT** es de hecho el estado BUS de la siguiente instrucción.

Fijándose en el camino de datos de la figura se puede ver que las señales de control que tiene que activar la unidad de control para cada uno de los estados es la siguiente:

señales de control	**eval**	**dir2**	**salt**
LDRA	0	0	0
LDRI	0	0	1
LDPC	0	0	1
LDR@	0	1	0
LDFZ	0	0	0
LDFN	0	0	0
ERD	0	0	0
L/E	0	0	0
MUXPC	X	X	1

MUXRG<1:0>	0	1	X
OPERAR	X	X	x

8.5.5 DIAGRAMA DE ESTADOS Y TABLA DE SALIDA DE LA UNIDAD DE CONTROL



A este diagrama de estado le corresponde la siguiente tabla de salidas de la unidad de control:

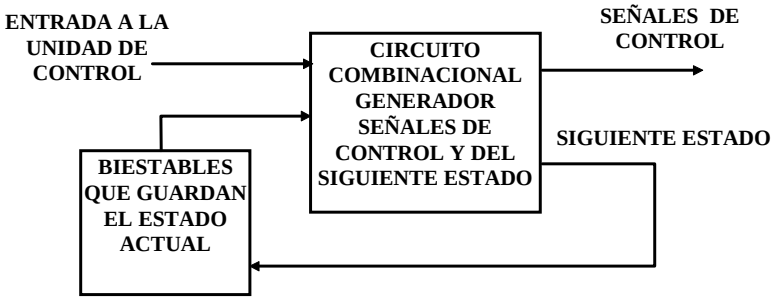
salidas	bus	dec	dir	load	store	po	soe	eval	dir2	salt
LDRA	0	0	0	0	0	1	0	0	0	0
LDRI	1	0	0	0	0	0	0	0	0	1
LDPC	1	0	0	0	0	0	0	0	0	1
LDR@	0	0	1	0	0	0	0	0	1	0
LDFZ	0	0	0	1	0	0	1	0	0	0
LDFN	0	0	0	1	0	0	1	0	0	0
ERD	0	0	0	1	0	0	1	0	0	0
L/E	0	0	0	0	1	0	0	0	0	0
MUXPC	0	X	X	1	1	X	X	X	x	1
MUXRG<1:0>	X	X	1	X	0	1	2	X	1	X
OPERAR	X	X	X	0	X	X	1	X	x	X

9 IMPLEMENTACION DE LA UNIDAD DE CONTROL

Una vez que se tienen definidas todas las señales de control que debe generar la U.C. y los instantes de tiempo en que deben generarse ya se tiene la información necesaria para implementarla. Existen dos tipos de implementación: la cableada y la microprogramada

9.1 UNIDAD DE CONTROL CABLEADA

Es un circuito secuencial clásico, en el que las señales de entrada se transforman en un conjunto de señales de salida que son las de control. Este sistema secuencial recuerda en cada momento el estado en que se encuentra la U.C.



ENTRADAS A LA UNIDAD DE CONTROL

Algunas de las entradas a la U.C. vistas en el tema anterior se pueden modificar para simplificar la implementación. Ejemplos típicos son: El registro de instrucciones y Reloj

- **Registro de Instrucciones**

La U.C. utiliza el código de operación para realizar diferentes acciones de instrucciones diferentes. La U.C. podría incluir un descodificador en su interior para tratar el código. Este descodificador puede ser exterior a la U.C. de manera que la unidad de control tenga una única entrada activa para cada código de operación. Esto se puede ver en la siguiente figura 5.1.

- **Reloj**

La unidad de control genera diferentes señales en diferentes instantes de tiempo T_I. El periodo del pulso de CK debe ser lo suficientemente largo como para permitir la propagación de las señales a lo largo del camino de datos y de la circuitería de la C.P.U. Si el CK ataca directamente a la U.C., ésta debe incluir un S.S. que indique exactamente en que instante de tiempo nos encontramos para cada instrucción. Esto se puede simplificar utilizando un contador exterior a la U.C. que indique el instante T_I en el que se encuentra cada instrucción. Al finalizar el ciclo de instrucción se debe inicializar el contador.

Estas simplificaciones se pueden hacer extensivas a otras señales de entrada a la unidad de control como son las señales de código de las operaciones aritméticas y los códigos de condición. En el caso de las señales de control de las operaciones aritméticas se suele codificar de manera que sean directamente las señales del código las que seleccionen la operación en la UAL. Si es muy compleja la unidad aritmético lógica puede que haga falta una unidad combinacional que descodifique el código que proporciona el RI.

En el caso de las condiciones de salto en lugar de realizar la evaluación dentro de la unidad de control se puede utilizar un módulo evaluador de condiciones que indique a la unidad de control si se debe producir salto o no.

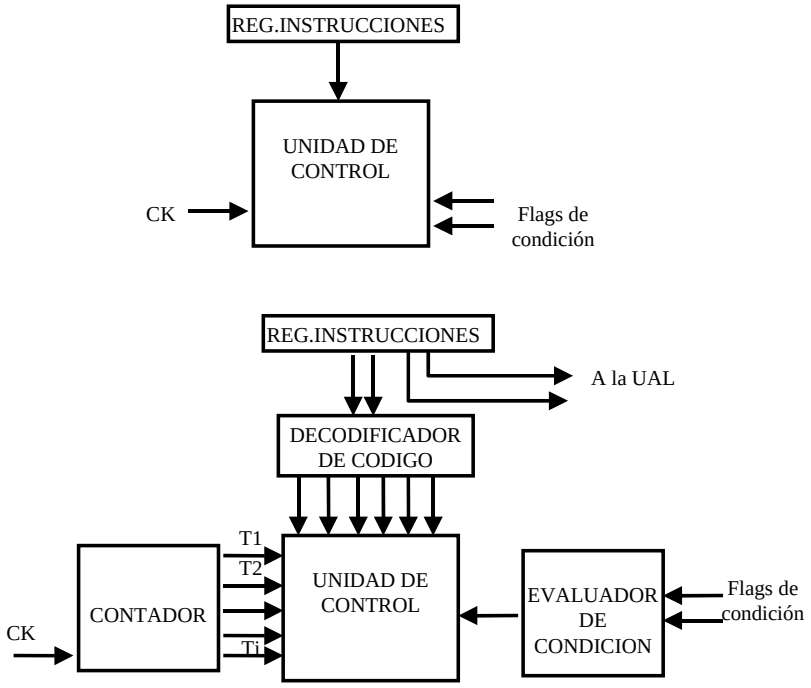
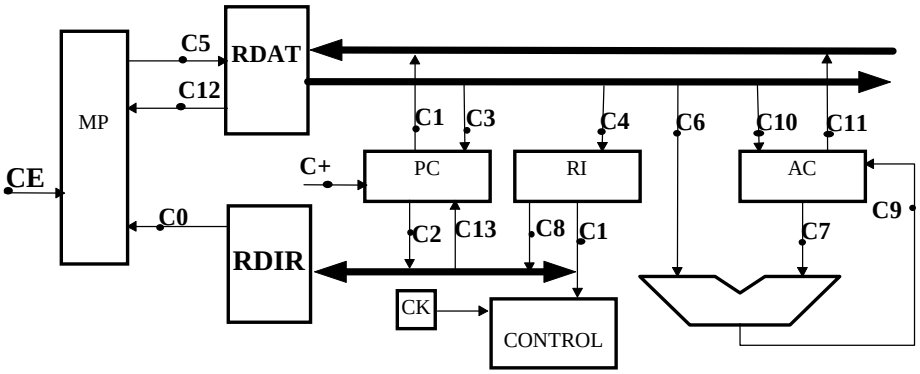


Figura 11.1

9.1.1 LOGICA DE LA UNIDAD DE CONTROL

Una vez vista como se pueden tratar las entradas a la unidad de control, se ve como se implementa la lógica combinacional de la unidad de control que genera las señales de control y el siguiente estado. Para cada señal de control se debe hallar su expresión booleana en función de las entradas. Vamos a ver como se implementan algunas de las señales que controlan la estructura:



que quedaban especificadas de la siguiente manera:

- **Búsqueda:**

  T_1: RDIR ← <PC>: $C_{2;}$

  T_2: RDAT - MEMORIA: C_5, $C_{L;}$ PC ← <PC> + 1:C+

  T_3: RI ← <RDAT> : C_4

- **Indirección:**

  $T_{1:}$ RDIR ← <RI.dirección>: $C_{8;}$

  T_2: RDAT ←<MEMORIA[RDIR]>:C_5, C_R

  T_3: RI. DIRECCION ← <RDAT>: C_4

- **Interrupción:**

  T_1: RDAT ← <PC> :C_1

T_2: RDIR ← dirección de salva guarda; PC ← Dirección de la subrutina de tratamiento de interrupción

T_3: MEMORIA[RDIR]← <RDAT>: C_{12}, C_E,C0

Atención: como no está definido el camino de datos completamente, hay señales de control que faltan.

ν **ADD AC,X**

T_1: RDIR ← <RI .dirección> :C_8

T_2: RDAT ← <MEMORIA[RDIR]> :C_L, C_5

T_3: R1← <R1> + <RDAT> :C_9,C_6,C_7

ν **ISZ X**

T_1: RDIR ← <RI.dirección>: C_8

T_2: RDAT ← <MEMORIA[RDIR]>:C_L, C_5,C_0

T_3: AC ← <RDAT> + 1:C_6,C_9

T_4: RDAT ←<AC >:C_{11}

T_5: MEMORIA[RDIR]← <RDAT>: C_E ,C_0,C_{12}

if <RDAT = 0> then PC ←<PC>+1 C+

BSA X

T_1: RDIR ← <RI. Dirección> :C8; RDAT ← <PC>:C1

T_2: PC ← <RI. Dirección> : C13; MEMORIA[RDIR]← <RDAT> :C0, C12, CE;

T_3: PC ← <PC> + 1; C+

Debido a que cada fase tiene varios ciclos, el módulo contador indica en que ciclo de una fase nos encontramos. Por ello, de alguna manera se debe indicar si el ciclo corresponde a la fase de búsqueda o a la fase de indirección. Esta es la razón de que se incluya un registro en la unidad de control que llamamos registro de fases y que, para el ejemplo, consta de dos bits que hemos llamado PQ. Este registro se supone transparente al usuario y nos indica en cada instante en que fase se encuentra la ejecución de la instrucción.

00 Búsqueda

01 Indirección

10 Ejecución

11 Interrupción

A continuación vemos el ejemplo de implementación de varias señales de control. Implementación de la señal C_5 que es la señal de carga del registro de datos de memoria (MBR). Esta señal se utiliza en tres fases diferentes: en la de búsqueda el registro de fase toma el valor P'Q' en su ciclo T2, y la ecuación lógica que la implementa la señal de control para este casoe s P'Q'T2; en la fase de indirección el registro de fase vale P'Q en su ciclo T2 y la ecuacion lógica que implementa la señal de control es P'QT2.

En la fase de ejecución el registro de fase vale (PQ'). Como ya sabemos la fase de ejecución depende de la instrucción que se esté ejecutando. Por lo tanto además de indicar la fase, también se debe indicar la instrucción. Esta señal se activa en el ciclo T2 de la Instrucción ADD y en el ciclo dos de la ASZ, luego la ecuacion logica que la imlementa es PQ'(AddT2+ASZT2), recordando que Add y Asz son señales que provienen el descodificador colocado entre el RI y la Unidad de control. Por lo tanto la señal de control C5 se implementa con la siguiente función lógica total:

$$C_5 = P'\cdot Q'\cdot T_2 + P'\cdot Q\cdot T_2 + PQ'(ADD\cdot T_2 + ISZ\cdot T_2)$$

Otras señales de control serían:

$$C_2 = P'Q'\cdot T_1$$

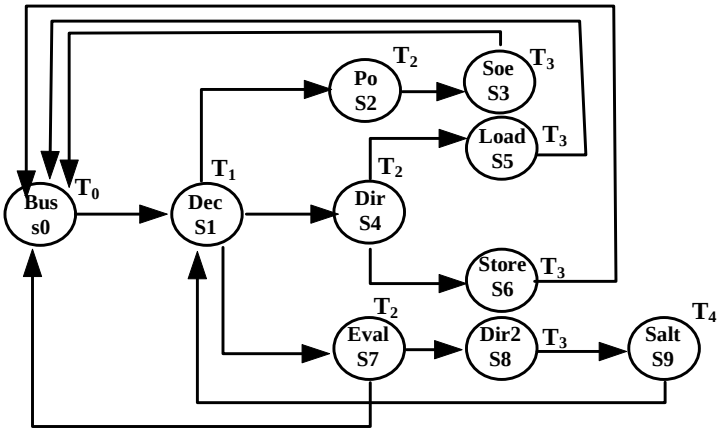
$C_0=P'Q'\cdot T_2+P'Q\cdot T_2+PQ'(ISZ\cdot T_2+BSA\cdot T_2)+PQ\cdot T_3$

$C_4=P'Q'\cdot T_3+P'QT_3$

$C_8=P'Q\cdot T_1+PQ'\cdot(ADD\cdot T_1+ISZ\cdot T_1+BSA\cdot T_1)$

Es importante recordar que al final de cada ciclo de búsqueda, indirección, etc..., la UC debe generar una señal que reinicie el contador. La UC también debe poner los valores apropiados a P y Q, para definir la siguiente fase que se ejecuta..

Esta técnica también se podría hacer para implementar la Unidad de Control de la máquina rudimentaria. Lo único que habría que tener en cuenta es que para este caso las fases sólo tienen un ciclo de reloj y por lo tanto el contador, no cuenta ciclos de fases sino fases propiamente dichas:



	T0	T1	t2	t3	t3	t2	t3	t2	t3	t4
salidas	**bus**	**dec**	**dir**	**load**	**store**	**po**	**soe**	**eval**	**dir2**	**salt**
LDRA	0	0	0	0	0	1	0	0	0	0
LDRI	1	0	0	0	0	0	0	0	0	1
LDPC	1	0	0	0	0	0	0	0	0	1
LDR@	0	0	1	0	0	0	0	0	1	0
LDFZ	0	0	0	1	0	0	1	0	0	0
LDFN	0	0	0	1	0	0	1	0	0	0
ERD	0	0	0	1	0	0	1	0	0	0
L/E	0	0	0	0	1	0	0	0	0	0
MUXPC	0	X	X	1	1	X	X	X	x	1
MUXRG<1:0>	X	X	1	X	0	1	2	X	1	X
OPERAR	X	X	X	0	X	X	1	X	x	X

Atención d1ebido a que se añade al camino de datos un contador de ciclos de reloj hay que generar las señales de control que lo controlan. En este caso debe salir una señal de reset contador para cargar un cero y por lo tanto saltar al estado bus, y una señal carga uno para poder implementar las instrucciones de bifurcación. Además hay que añadir un nuevo estado a continuación de estado eval. Esto se debe a que hemos supuesto el sistema secuencial como máquina de moore luego las señales que genera cada estado tienen que estar perfectamente identificadas. Al añadir la señal de resetck el estado eval tendría que generar la señal reset a uno o cero según la entrada del evaluador fuera 1 o 0. Como no puede ser se añade un estado más a continuación de eval. Luego ahora necesitamos la señal que procede del evaluador para diferenciar entre los dos estados que se pueden producir en el ciclo 3 de las instrucciones bif.

Suponemos un descodificador entre el RI y la UC. Y que el contador cuenta ciclos totales de la instrucción y no ciclos de fases como en el caso anterior. En esta implementación es el contador el que guarda la historia del sistema. Las entradas del sistema son :

T0...T4; AL; ST; LD; BIF; EVALUA

Y las señales implementadas son

$LDRA=T_2 \cdot AL$

$LDRI=T_0AL+T_0ST+T_0LD+T_0BIF+T_4BIF$

$LPC= T_0AL+T_0ST+T_0LD+T_0BIF+T_4BIF$

$LDR@=T_2ST+T_2LOAD+T_3BIF \cdot EVALUA$

$LZ=T_3LOAD+T_3AL$

$LN=T_3LOAD+T_3AL$

$ERD=T_3LD+T_3AL$

$L/E=T_3ST$

$MUXPC=T_2LD+T_2ST+T_4BIF$

$MUXREG<0>=T_2LD+T_2ST+T_2AL+T_3BIF \cdot EVALUA$

$MUXREG<1>T_3ALERD=T_3LD+T_3AL$

$L/E=T_3ST$

$MUXPC=T_2LD+T_2ST+T_4BIF$

$MUXREG<0>=T_2LD+T_2ST+T_2AL+T_3BIF \cdot EVALUA$

$MUXREG<1>T_3AL$

9.2 UNIDAD DE CONTROL MICROPROGRAMADA

9.2.1 INTRODUCCIÓN

Emplea una memoria de control para almacenar la información de las señales de control de cada periodo de tiempo. Se llama Microinstrucción a cada palabra de memoria de control que define todas las señales de control en un periodo de la instrucción. Se utilizan en **c**omputadores de tamaño medio. En los pequeños la estructura es demasiado compleja y no resulta económica. En los grandes demasiado lento. Fue propuesta por 1956 Wilkes y se utilizó por primera vez comercialmente en la familia de computadores IBM 360. Inicialmente se rechazó debido a la inexistencia de Memorias rápidas y baratas.

Una μinstrucción es una secuencia de 0 y 1 que representa los valores de la señal de control en un periodo. Una instrucción se descompone en ciclos y un ciclo en μoperaciones y cada μoperación se realiza activando señales de control. El conjunto ordenado de μinstrucciones que implementan una instrucción máquina se llama μprograma. Para ejecutar un microprograma se van leyendo cada una de las μinstrucciones y se envían las señales leídas al computador como señales de control. La cadencia de lectura es la del reloj básico del computador. Se llama firmware al **c**onjunto de μprogramas de una máquina

Observese que en la μinstrucción las señales de flanco y las de nivel tienen el mismo tratamiento. Ejemplo de señales activadas por niveles son las de selección de un mux y de señales activadas por flanco son la carga de registros. Una solución sería recortar la señal con una señal de reloj mediante una puerta AND . Esto tiene el Inconveniente :de que la señal de reloj llegue retrasada respecto a la de control ⇒ esta señal tenga tiempos de Host y Setup diferente al resto. La solución**: es** implementar los registros con dos señales de control , una de capacitación que nos indica si se realiza la carga o no (que es la que genera la U.C), y otra de reloj que nos indica el instante preciso en que esta carga se realiza. La capacitación se implementa mediante un biestable de carga por flanco de manera que la señal de capacitación este conectada a la de entrada de datos y la señal de reloj a la señal de reloj

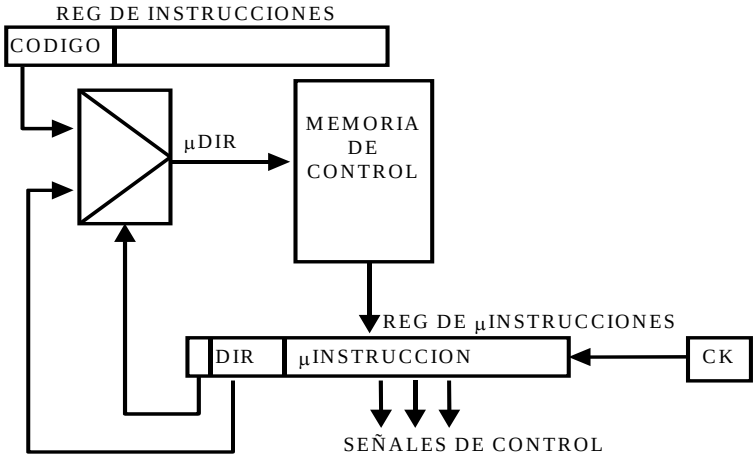
9.2.2 ESTRUCTURA DE UNIDAD DE CONTROL μPROGRAMADA:

La U.C. microprogramada debe disponer de los siguientes mecanismos, una memoria de control con capacidad para almacenar todos los μprogramas, un mecanismo que convierta el código de operación en la dirección de la memoria de control donde se almacena el μprograma, y un mecanismo de secuenciamiento y bifurcación a nuevo μprograma.

Existen dos alternativas de secuenciamiento: explícito e implícito

Secuenciamiento explícito:

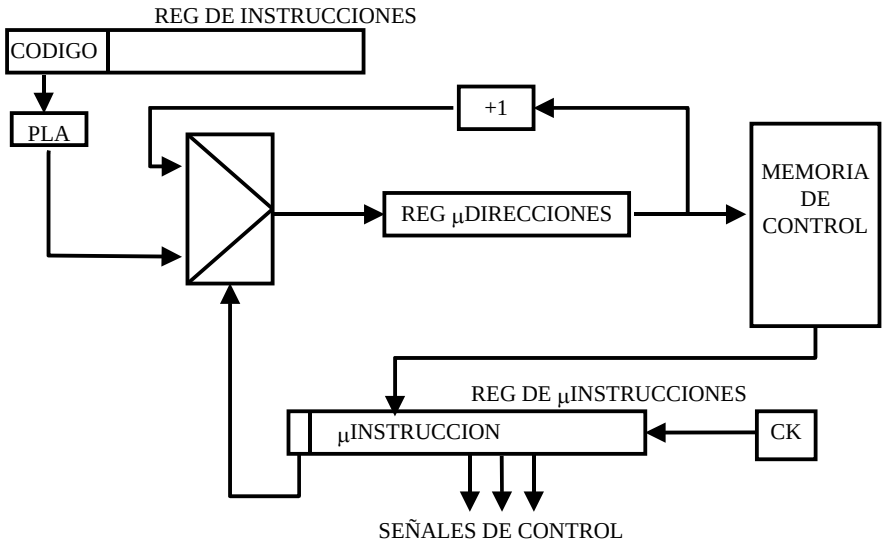
Es el que incluye en la μinstrucción la dirección de la μinstrucción siguiente. Gracias a esto La μinstrucciones de un μprograma pueden estar desordenadas. La forma de cumplir el requisito 2, es sencilla. Se guarda la μinstrucción en la posición que determine el código de operación de la Instrucción. No hace falta mecanismo de secuenciamiento. La bifurcación a un nuevo μprograma se consigue con una señal de control que toma como dirección el nuevo código de operación. Inconveniente: enorme gasto de memoria de control.



Secuenciamiento implícito:

Las μinstrucción no contienen la dirección de la siguiente μinstrucción, lo que obliga a tener ordenadas las μinstrucciones de un μprograma en posiciones consecutivas de memoria de control. Tiene la ventaja de ahorrar memoria. Debe existir un mecanismo que resuelva los problemas de secuenciamiento y bifurcación y los de generación de la dirección inicial del μprograma de una instrucción máquina, por lo tanto debe existir un modulo generador de direcciones

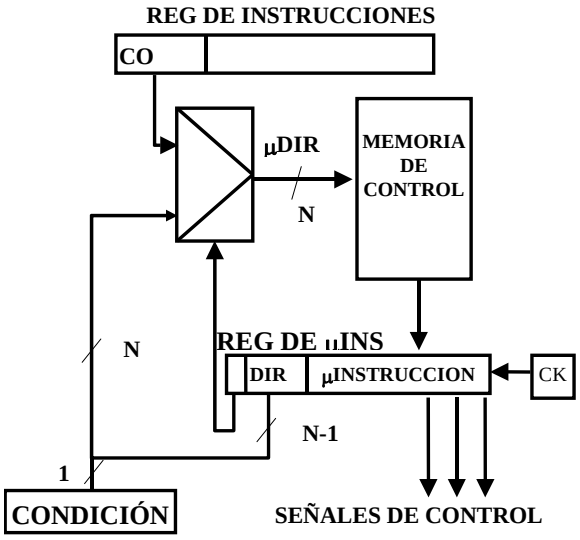
Como Generador de direcciones se va a utilizar una etapa traductora entre el código de operación y el Mux de direcciones. Suele ser una PLA o una ROM. El Secuenciamiento y bifurcación se realiza mediante un registro de μdirecciones y un incrementador.



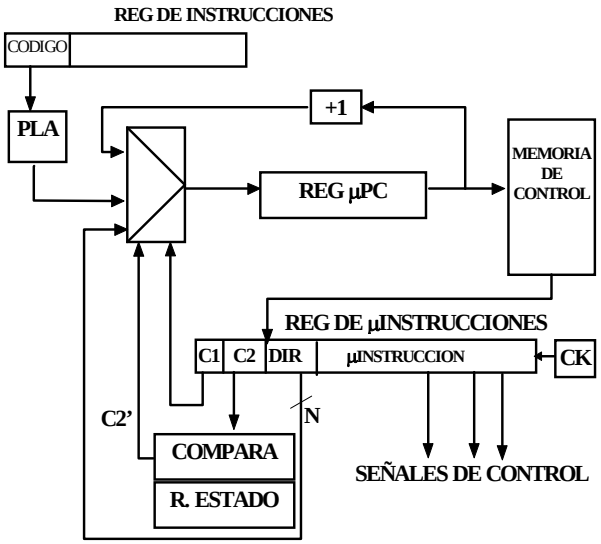
ESTRUCTURA COMPLETA CON BIFURCACIONES CONDICIONALES

Debe existir un mecanismo que permita la bifurcación a una μrama o a otra en función del valor de un bit de condición. Las instrucciones tienen con frecuencia partes comunes y existen instrucciones con partes compartidas. La unidad debe tener capacidad de μbucles y llamadas a μsubrutinas. En realidad las secuencias de μinstrucciones tienden a ser muy cortas. Cada 3 o 4 μinstrucciones suele haber una ruptura de secuencia. Es importante que las técnicas de bifurcación estén optimadas en el tiempo.

En el secuenciamiento explícito la μinstrucción debería contener dos direcciones. Pero esto es costoso en memoria. La solución es que las direcciones difieran solo en un bit. El bit toma el valor del comparador de la bifurcación.



En el secuenciamiento implícito la μinstrucción deberá contener la μdirección. Se suelen usar campos solapados (se explican más adelante).



C1 selecciona la dirección del RI o la dirección interna a la unidad de control y C2' selecciona entre dos direcciones internas de la UC, la secuencia normal (microPc) , la de salto (DIR). C2' se genera con información que viene del registro de estados del computador y del campo C2 del microinstruccion

Surge un problema. Suponiendo Dir=8 y el nº de bits de las micro instrucciones es 14 esto indica que el registro de tiene 24 bits es decir el 33% solo se usa en las instrucciones de salto

- **MICROBUCLES Y MICROSUBRUTINAS:**

Los µbucles exigen una bifurcación condicional y un contador con autodecremento que se emplea como condición de bifurcación. Este contador debe ser accesible desde la microinstruccion para poder inicializarles adecuadamente. Las µsubrutinas necesitan un almacenamiento para salvaguardar la dirección de retorno. Añadir una pila al registro µdirecciones en la que se guardan las direcciones de retorno.

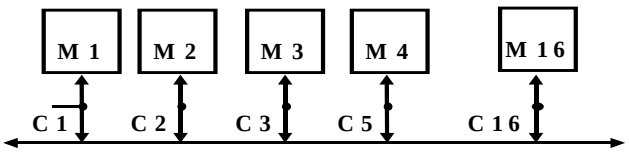
9.2.3 FORMATO Y CODIFICACIÓN DE LAS µINSTRUCCIONES

Existen dos elementos de diseño que se deben tener en cuenta al estudiar el secuenciamiento: el tamaño de la memoria de control y el tiempo de generación de la siguiente µinstrucción .

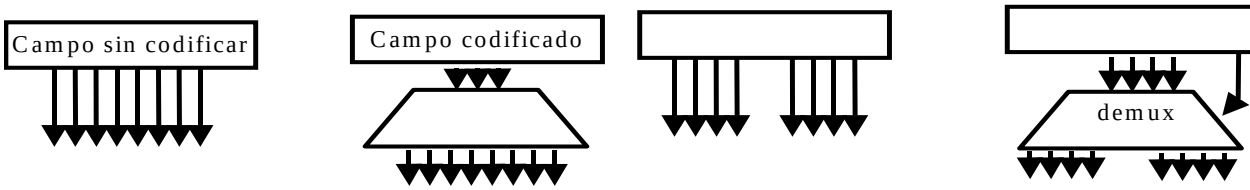
En su forma más sencilla una µinstrucción tiene 1 bit por cada señal de control y una duración de un periodo. Esta es una solución sencilla, pero poco empleada por ser muy cara. Por lo general una µinstrucción solo activa una señal de control en cada periodo de tiempo luego la µinstrucción está llena de 0's. La solución es codificar las µinstrucciones para que tengan una longitud más reducida. Se llama µprogramación horizontal a la que usa µinstrucciones no codificadas. Se llama µprogramación vertical a la que usa µinstrucciones codificadas. La horizontal se caracteriza por tener un mayor tamaño y paralelismo, la vertical no permite el paralelismo y exige descodificaciones más lentas pero usa menos memoria.

El formato especifica el número de bits que tiene la µinstrucción y el significado de cada uno de ellos. Las señales de control que sirven para accionar un mismo módulo se agrupan formando campos. Por ejemplo: señales triestado de acceso a bus, señales de la UAL, Señales de banco de registro ySeñales de memoria

Vamos a ver un ejemplo de codificación de campos. Suponer un bus al que acceden 16 módulos diferentes

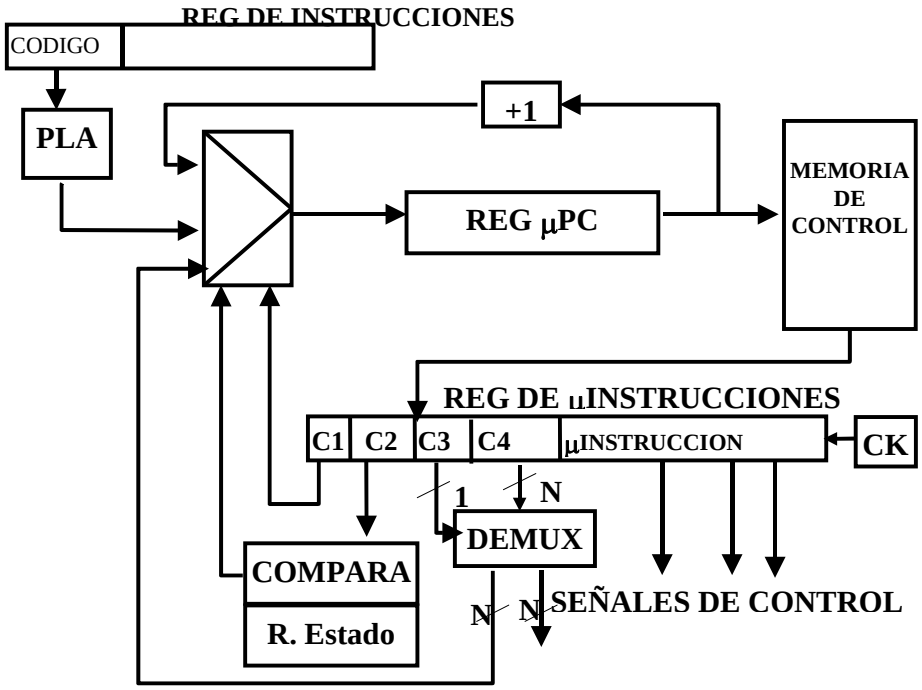


necesitaremos 16 señales de control para controlar el acceso al mismo. Esto tiene el inconveniente de que solo hay una señal activa en cada ciclo. La solución es la codificación, de manera que sólo se necesitan 4 bits. Esto tiene el inconveniente de que se necesita una etapa decodificadora , lo que produce un aumento del coste y del retardo.



solapamiento de campos:

Existen con frecuencia señales excluyentes, es decir que no se pueden activar simultáneamente. Esto Permite emplear un único grupo de bits para dos campos distintos C_1 y C_2 y se emplea un señal auxiliar C_{12} para descodificar. A continuación vemos como queda la estructura de secuenciamiento implícito suponiendo bifurcación con solapamiento de campos.



9.2.4 VENTAJAS DE LA μPROGRAMACIÓN:

El principal problema de la unidad de control cableada es que para máquinas de cierta complejidad, ésta puede tener miles de estados con cientos de secuencias diferentes, lo que dificulta enormemente su implementación. En cambio la microprogramada tiene como ventajas Su simplicidad conceptual, ser más barata y menos propensa a errores. Al estar la Información almacenada en memoria ROM y RAM las correcciones son más sencillas. Utilizando microprogramas se pueden integrar algunas funciones del S.O. a este nivel. Además un mismo computador tiene varios juegos de instrucciones Lo que se puede usar para la emulación de otros computadores y la implementación de rutinas de diagnóstico muy completas.

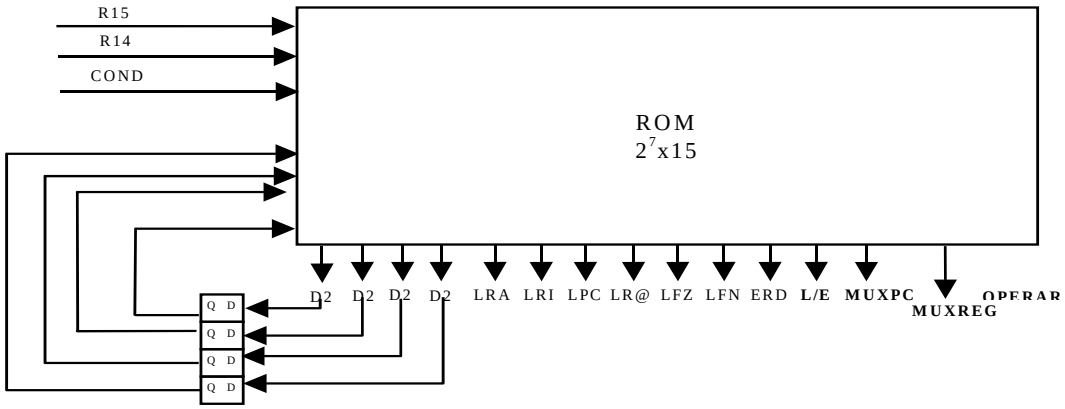
Como desventaja la microprogramación tiene el ser demasiado lenta para las máquinas complejas y demasiado compleja para máquinas sencillas. En cualquier caso es la técnica predominante en la actualidad para implementar unidades de control debido a su facilidad de implementación.

Conviene saber que no siempre la implementación mediante microinstrucciones complejas es más rápida que un secuencia de instrucciones máquina, y menos desde la existencia de memorias cache que aceleran el acceso a las instrucciones.

En la actualidad las unidades de control se implementan en el mismo circuito integrado que el resto del procesador usando las facilidades de las herramientas automáticas para introducir ROM y PLAS. Es importante saber que una memoria ROM tiene unos tiempos de acceso similares a los de las memorias RAM que se utilizan para almacenar las instrucciones del lenguaje máquina. Esto, unido al hecho de que en la memoria ROM pueden quedar muchas entradas sin utilizar, hace que sean las PLA las que en la actualidad implementen las unidades microprogramadas

Por último el diseñador debe tener en cuenta que los repertorios de instrucciones son mucho más sencillos de lo que eran en los años 70 y 80, lo que reduce la complejidad de la unidad y da la oportunidad de utilizar lo que daría la oportunidad en muchos caso de implementar la unidad cableada.

9.2.5 EJEMPLO



Para implementar la unidad de control necesitamos una memoria ROM y un conjunto de biestables. En los biestables se guarda el estado actual de la máquina de estados y en la memoria ROM se guarda el siguiente estado y las señales de control correspondientes al estado actual. Por lo tanto si tenemos diez estados necesitamos cuatro biestables para codificarlos.

La dirección a la que se accede viene dada por el contenido de los biestables y por las entradas, por lo tanto en nuestro caso se necesita un bus de direcciones de 7 bits lo que indica que la memoria tendrá 128 posiciones de las cuales no todas se utilizan. En cuanto a los bits por palabra tendrá cuatro para representar el estado y doce señales de control propiamente dichas. A continuación vamos a estudiar como habría que llenar la memoria. Las direcciones vienen dadas por los siguientes bits

$$Q3\ Q2\ Q1\ Q0\ RI_{15}\ RI_{14}\ COND$$

es decir Q3 es el bit de dirección de mayor peso y COND es el bit de dirección de menor peso:

	Q3	Q2	Q1	Q0	RI15	RI14	COND	D3	D2	D1	D0	LRA	LRI	LPC	LR@	LFZ	LFN	ERD	LE	MPC	MREG	OPER
BUS	0	0	0	0	X	X	X	0	0	0	1	0	1	1	0	0	0	0	0	0	x	x
DEC	0	0	0	1	1	1	X	0	0	1	0	0	0	0	0	0	0	0	0	x	x	x
DEC	0	0	0	1	0	X	X	0	1	0	0	0	0	0	0	0	0	0	0	x	x	x
DEC	0	0	0	1	1	0	X	0	1	1	1	0	0	0	0	0	0	0	0	x	x	x
PO	0	0	1	0	X	X	X	0	0	1	1	1	0	0	0	0	0	0	0	x	1	x
SOE	0	0	1	1	X	X	X	0	0	0	0	0	0	0	0	1	1	1	0	x	2	1
DIR	0	1	0	0	0	0	X	0	1	0	1	0	0	0	1	0	0	0	0	x	1	x
DIR	0	1	0	0	0	1	X	0	1	1	0	0	0	0	1	0	0	0	0	x	1	x
LOAD	0	1	0	1	X	X	X	0	0	0	0	0	0	0	0	1	1	1	0	1	x	0
STORE	0	1	1	0	X	X	X	0	0	0	0	0	0	0	0	0	0	0	1	1	0	x
EVAL	0	1	1	1	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x
EVAL	0	1	1	1	X	X	1	1	0	0	0	0	0	0	0	0	0	0	0	x	x	x

DIR2	1	0	0	0	X	X	X	1	0	0	1	0	0	0	1	0	0	0	0	x	1	x

En la tabla anterior aparece la información que se debe cargar en la memoria ROM. Vamos a ver un ejemplo de como se llenaría la memoria. En la tabla se ve que la información relativa al estado BUS se debe cargar en las posiciones de memoria determinadas por los valores 0000xxx, para todas las posibles combinaciones de xxx, es decir desde 0000000 hasta 00001111,

10 LAS TÉCNICAS DE ENTRADA Y SALIDA

10.1 INTRODUCCION

La forma de realizar un intercambio entre el computador y el exterior es mediante interfaces de entrada/salida. Si la CPU tuviera que conectarse directamente con todos los periféricos, el sistema tendría un rendimiento muy bajo, por eso se utilizan las interfaces, de manera que la CPU les envía la información y son éstas las que se encargan de conectarse con los periféricos. Esto tiene el inconveniente de que la CPU debe estar en constante conexión con las interfaces de entrada/salida con la consiguiente perdida de ciclos de CPU. La solución a este problema ha sido crear dispositivos, cada vez mas sofisticados, que se encargan de conectarse con las interfaces. En el límite se añaden procesadores de entrada/salida con su propia memoria. Gracias a esto se descarga de trabajo a la CPU para que siga trabajando en lo suyo sin preocuparse de las entradas/salidas.

Las principales diferencias entre los sistemas de entrada/salida y los procesadores son que los diseñadores de procesadores se centran habitualmente en mejoras del rendimiento, mientras que los diseñadores de sistemas de entrada-salida deben tener en cuenta la expandibilidad del sistema, la capacidad recuperación de fallos, la diversidad de periféricos y la finalidad del sistema

Por otro lado, el rendimiento de un sistema de ES es más complejo de medir que el de un procesador. En algunos sistemas se debe tener en cuenta las latencias de acceso, mientras que en otros es más importante la productividad. Por lo tanto la productividad puede interpretarse de dos maneras: Nº total de bits que se pueden intercambiar (p.e. en SS computadores orientados a cálculos científicos) y número total de transferencias independientes que se pueden realiza, por ejemplo transferencias las bancarias.

Además el rendimiento depende de otros factores como son las características del dispositivo, las conexiones entre el dispositivo y el resto del sistema, la jerarquía de memoria o el sistema operativo. Durante mucho tiempo ha existido la práctica generalizada de relegar los estudios de sistemas de E/S a segunda fila olvidando que uno de los fines primordiales de un computador es recibir y proporcionar información. Además, de nada sirve conseguir procesadores muy rápidos si existen cuellos de botella en los sistemas de entrada salida.

También conviene recordar que en la actualidad desde los PC más sencillos hasta los sistemas supercomputadores más potentes se construyen con la misma tecnología base, y que es la potencia de los subsistemas de entrada/salida y de memoria lo que marcan las diferencias entre unos y otros.

10.2 MEDIDAS DE RENDIMENTO

El modo de evaluar el rendimiento de un sistema de entrada/salida depende de la aplicación que se quiera dar al sistema. En algunas aplicaciones prima la productividad (el ancho de banda) sobre otros factores. La productividad de un sistema se puede interpretar de diferentes maneras. Por ejemplo se puede definir como el total de datos que se pueden intercambiar en un determinado tiempo o, por el contrario, como el total de operaciones de entrada/salida en un determinado tiempo. Por ejemplo, en el caso de los computadores dedicados a cálculos científicos una sola operación de entrada/salida maneja gran cantidad de datos. En este caso lo que cuenta es la anchura de banda de la transferencia. En cambio en los accesos a grandes bases de datos se debe tener en cuenta que hay que gran cantidad de pequeños accesos independientes entre sí.

En algunas aplicaciones interesa el tiempo de respuesta del computador. Esto último es típico de sistemas monousurario. Por último, un gran número de aplicaciones necesita una alta productividad y corto tiempo de respuesta, por ejemplo los cajeros automáticos. En estos entornos preocupa cuanto tarda cada tarea y cuantas tareas simultáneamente pueden procesarse.

Generalmente el tiempo de respuesta y la productividad son inversamente proporcionales. Para conseguir pequeños tiempos de respuesta suele procesarse las entradas/salidas en cuanto llega la petición. Para optimizar la productividad suelen agruparse las peticiones de Entrada/Salida próximas, con lo que se aumentan los tiempos de respuesta.

Existen parámetros y programas que sirven de bancos de pruebas para estudiar el rendimiento de los sistemas de entrada/salida. Estos bancos de pruebas son bastante primitivos comparados con los bancos de pruebas existentes para la medida del rendimiento de los procesadores. Estos programas se ven influidos por:

- Tecnología de los discos secundarios del sistema
- Nº de discos conectados
- El sistema de memoria
- El procesador
- El sistema de archivos que proporciona el SO

10.3 FUNCION DE UN MÓDULO DE ENTRADA/SALIDA

Junto con la CPU y la memoria, el tercer elemento clave en un sistema computador son los módulos de entrada/salida. Los módulos de entrada/salida se conectan directamente al bus de sistema y su misión es controlar uno o más periféricos, es decir son los encargados de llevar el peso de la transferencia. Para poder llevar a cabo su misión deben contener la lógica necesaria para permitir la comunicación entre el periférico y el bus.

¿Porque no se conecta un periférico directamente al bus del sistema?. Por tres motivos. El primero de ellos, la gran diversidad de periféricos que existe forzaría a incorporar demasiada lógica a la CPU. El segundo, la velocidad de transferencia de los periféricos es mucho menor que la de la CPU, lo que ralentizaría la ejecución de programas enormemente. Por último, debido a la gran diferencia de formatos entre la CPU y los periféricos. Para poder cumplir con sus misiones el módulo de E/S se compone de una interfaz interna con la CPU y la memoria, y una interfaz externa con los periféricos. Las funciones asignadas al módulo de entrada/salida son:

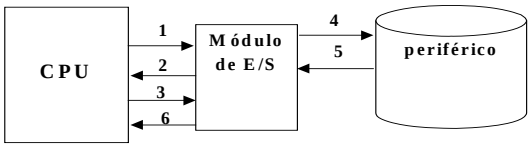
- Control y temporización de la transmisión de información
- Comunicación con la CPU
- Comunicación con los dispositivos periféricos
- Almacenamiento temporal de datos (función de buffer)
- Detección y corrección de errores.

Cada una de estas funciones se explican con más profundidad a continuación.

CONTROL Y TEMPORIZACIÓN

Los módulos de ES realizan la coordinación del trafico de información entre los recursos internos y los periféricos. La transferencia de datos de un periférico a la CPU podría implicar los siguientes pasos:

1. La CPU pregunta al módulo de E/S por el estado del periférico
2. El módulo de E/S responde a la CPU
3. Si el periférico está operativo y preparado, la CPU solicita la transferencia al módulo de E/S
4. El módulo de E/S solicita un dato del periférico
5. El periférico envía el dato al módulo
6. Los datos se transfieren del módulo a la CPU



Si el sistema utiliza un bus, cada interacción entre CPU y módulo requiere un arbitraje del bus. Como se ve el módulo debe tener capacidad de entablar comunicación tanto con la CPU como con el periférico.

COMUNICACIÓN DEL MÓDULO DE E/S CON LA CPU

Esta comunicación viene dada por la descodificación de las ordenes que el módulo acepta de la CPU, por el intercambio de datos y el intercambio de información de estado, y por el reconocimiento de dirección un módulo.

COMUNICACIÓN DEL MÓDULO DE E/S CON PERIFERICOS

Lo que implica intercambiar ordenes, información de estado y datos

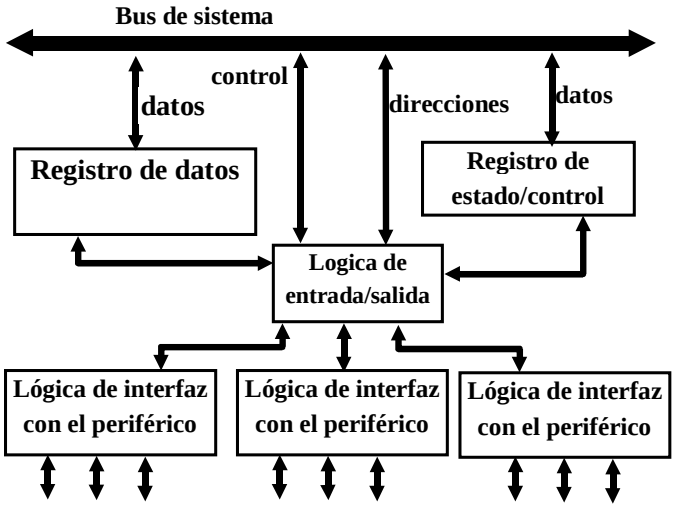
ALMACENAMIENTO TEMPORAL DE DATOS (FUNCIÓN DE BUFFER)

Utilizada para sincronizar las velocidades de la CPU y de los periféricos. Los datos se envían en ráfagas al módulo para aprovechar la velocidad de comunicación del procesador, y se almacenan temporalmente en él. Por último, se envían al periférico a la velocidad de éste. Cuando los datos se envían del periférico a la unidad central de proceso, se almacenan en el buffer para no mantener la memoria ocupada en transferencias demasiado lentas.

DETECCIÓN Y CORRECCIÓN DE ERRORES

Los errores que un módulo de entrada/salida puede detectar son de dos tipos: los errores mecánicos y eléctricos de funcionamiento (Ej atasco de papel, pista de disco en mal estado) y los cambios accidentales en los bits transmitidos. Para detectar estos errores se usan códigos de detección de errores.

10.4 ESTRUCTURA DEL MODULO DE E/S



La estructura y complejidad de los módulos de entrada/salida es muy diversa. La figura anterior muestra un diagrama de bloques de un módulo de E/S muy general. El módulo de entrada/salida se conecta al resto del computador a través del bus de sistema. Los datos que se envían o reciben se almacenan temporalmente en un registro interno, que actúa como buffer. El bloque de lógica interactúa con la CPU y genera las señales de control de los tres dispositivos internos que puede direccionar. Este bloque debe ser capaz de generar y reconocer las direcciones de los tres dispositivos.

Además de lo anterior, el módulo debe tener bloques de lógica específica de interfaz con cada periférico. El funcionamiento de un módulo de Entrada/Salida permite que la CPU acceda a una amplia gama de dispositivos de forma simplificada. Este módulo debe ocultar los detalles de temporización, los formatos y electromecánica de los periféricos externos para que la CPU pueda funcionar únicamente en término de ordenes de lectura, escritura, y de abrir y cerrar ficheros.

PROCESADOR DE E/S O CANAL DE E/S

Es un procesador que se usa como modulo de E/S y, que por lo tanto, se encarga de la mayoría de los detalles de procesamiento, presentando a la CPU una interfaz de alto nivel. Se utiliza normalmente en grandes computadores

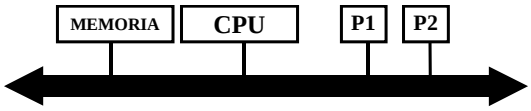
CONTROLADOR DE DISPOSITIVO

Es un módulo simple que necesita supervisión por parte de la CPU. Suele utilizarse en microcomputadores.

10.5 ORGANIZACIÓN DE LA ENTRADA/SALIDA

Existen dos organizaciones: la E/S localizada en memoria y la E/S aislada o independiente.

Entrada/salida localizada en memoria



El computador considera las direcciones de entrada/salida como direcciones de memoria, es decir no diferencia entre memoria principal y la entrada/salida. Generalmente la E/S se agrupa en una zona bien definida del mapa de direcciones. ¿Cómo se realiza el direccionamiento? Con P bits se direccionan 2^P periféricos diferentes. A estas 2^P direcciones se le llama mapa de entrada/salida.

Las operaciones de entrada/salida son operaciones de movimiento de datos (transferencia), donde la fuente (destino) es un registro de la CPU o una posición de memoria y el destino (fuente) es un registro del módulo de entrada/salida. La dirección se descodifica para convertirla en una señal de selección del dispositivo. Esto se puede hacer de dos formas:

- Un solo descodificador genera las 2^P señales de selección
- Un reconocedor de dirección en cada puerto de entrada/salida

Existen tres métodos para descodificar la dirección. Para estudiarlos suponemos 8 periféricos y una dirección de 8 bits:

Dirección	modo 1	modo2	modo3
0	0000 0000	xxxxx 000	xxxxxxx1
1	0000 0001	xxxxx 001	xxxxxx1x
2	0000 0010	xxxxx 010	xxxxx1xx
3	0000 0011	xxxxx 011	xxxx1xxx
4	0000 0100	xxxxx 100	xxx1xxxx
5	0000 0101	xxxxx 101	xx1xxxxx
6	0000 0110	xxxxx 110	x1xxxxxx
7	0000 0111	xxxxx 111	1xxxxxxx

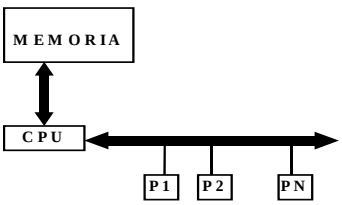
Modo1.- Relación biunivoca. A cada dirección le corresponde un periférico y a cada periférico le corresponde una sola dirección. Quedan muchas direcciones sin utilizar.

Modo2.- A cada dirección le corresponde un solo periférico a cada periférico le corresponde 32 direcciones. Se utilizan todas las direcciones.

Modo3.- Cada periférico tiene 128 direcciones, una dirección direcciona varios periféricos (tanto como 1's incluya)

Las dos últimas soluciones desaprovechan la capacidad de direccionamiento, pero si no existen más periféricos reducen el coste total del sistema. La ultima solución no necesita descodificador y por lo tanto es la más económica, pero si por error una dirección tiene dos o más 1's, se seleccionan varios periféricos y esto provoca varios accesos simultáneos al bus destruyéndose la información. Un ejemplo típico de sistema con la memoria localizada en memoria es el MC68000.

ENTRADA/SALIDA INDEPENDIENTE DE MEMORIA



Las direcciones son independientes de la memoria principal. Esto hace que no se puedan utilizar las instrucciones de movimiento de datos como ocurría en la organización anterior, y por lo tanto el repertorio debe implementar instrucciones especificas de entrada/salida. La existencia de estas instrucciones específicas provoca una mayor complejidad del HW. Además, los programas son más rígidos porque las operaciones tienen menos modos de direccionamiento y el mapa de entrada/salida más pequeño. Su principal ventaja es

que facilita la protección de las operaciones de entrada/salida. Ejemplo típico son los microprocesadores de la familia i8086.

10.6 TECNICAS DE ENTRADA/SALIDA

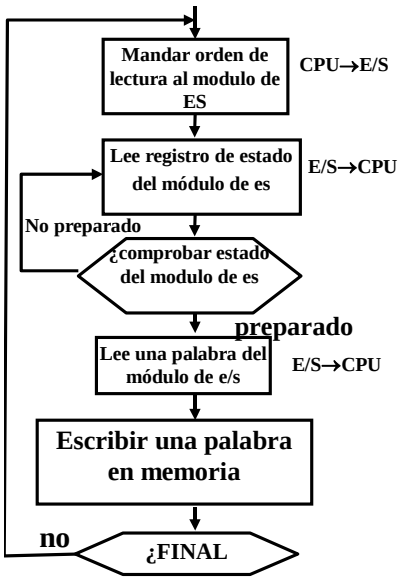
Llamamos técnicas de entrada/salida a las que utiliza el módulo de E/S para comunicarse con el procesador. Existen tres técnicas: Programada, Interrupciones, Acceso Directo a Memoria.

10.6.1 ENTRADA/SALIDA PROGRAMADA (POLLING-ESCRUTINEO)

Al proceso de comprobar periódicamente los bits de estado de los periféricos para ver si es el momento de realizar la siguiente operación de entrada/salida se le llama escrutinio. Es la forma más simple de que un dispositivo de entrada/salida se comunique con el procesador.

El periférico pone la información en el registro de estado del módulo de entrada/salida y el procesador debe venir y recoger la información, por lo tanto el procesador tiene el control y hace todo el trabajo. En la actualidad el único dispositivo que se accede por escrutinio es el ratón. El modo de operación sería el siguiente:

1. La CPU selecciona al módulo de Entrada/Salida que debe realizar la operación y le indica qué operación debe realizar
2. La CPU espera en un bucle de programa (que puede ser vacío o no) a que el módulo de IO interactúe con el periférico y modifique los bits de estado indicando que ha acabado la operación y está lista para realizar el intercambio.
3. Cuando la CPU acaba su ciclo de espera mira el registro de estado del módulo para ver si se ha realizado la operación.
4. La CPU realiza el intercambio



La principal característica de esta técnica es que la CPU permanece en un ciclo hasta que el periférico modifica el registro de estado del módulo para informar que está disponible para una nueva operación. El computador adapta su velocidad de trabajo a la del periférico (qué esta marcada por los circuitos electromecánicos), y además en muchas ocasiones pierde ciclos de trabajo efectivo debido a la espera. Esta forma de trabajo tiene algunos inconvenientes, como son:

- Pérdida de tiempo, puesto que durante la espera la CPU no hace trabajo útil. Con periféricos lentos el bucle de espera puede repetirse miles de veces.
- Esta técnica es difícil de aplicar cuando se quiere para atender a varios periféricos
- Existen tareas que no pueden esperar a que acabe el ciclo de espera.

- Una **solución** parcial es la limitación del tiempo de espera del bucle.

Solo se usa para periféricos lentos. Por ejemplo, la frecuencia del ratón es de 0,02Kbytesps, muy baja comparada con la de discos 2000Kbps o del disco óptico de 500kbps.

Existe otra modalidad de entrada salida programada llamada sin espera de respuesta. En esta modalidad se supone que cuando el procesador lee el dato, este ya se encuentra preparado. Elimina el bucle de espera pero es muy propensa a errores.

10.6.2 LAS INTERRUPCIONES (HAMACHER)

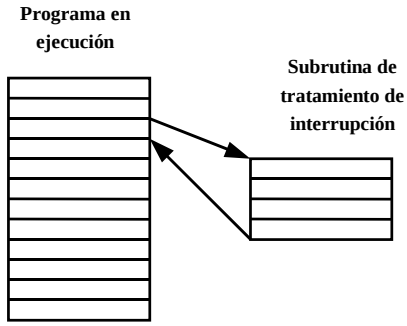
Una interrupción es una bifurcación externa al programa en ejecución provocada por una señal que viene del exterior de la CPU. Esta señal llega a través de una línea llamada de petición de interrupción. Las interrupciones también pueden ser internas a la CPU en este caso se llaman excepciones.

EL MODO GENERAL DE OPERACIÓN:

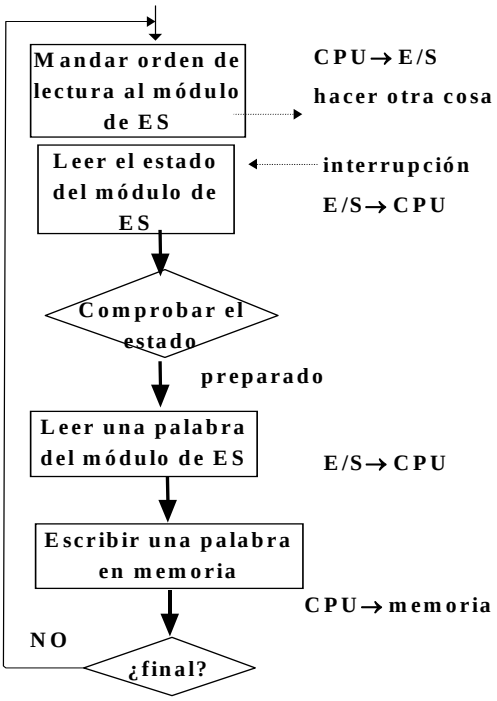
- La CPU indica al módulo de E/S el periférico al que quiere acceder y la operación que quiere realizar
- A continuación se desentiende de la operación y se pone a trabajar en algún otro proceso
- Cuando el módulo de entrada/salida acaba la operación, interrumpe a la CPU para indicarle que ya está lista para intercambiar información con ella.
- La CPU realiza la transferencia de información tras lo cual continua con el trabajo interrumpido

Una de las características más importantes de esta técnica es que debe realizarse una interrupción a la CPU por cada byte que se intercambia. Esto como veremos más adelante será una de las causas que provocan la aparición de otras técnicas de Entrada/Salida. Además una interrupción es asíncrona, es decir se produce independientemente de la señal de reloj que sincroniza el sistema. Su principal ventaja es que eliminan el ciclo de espera del procesador entre cada entrada de byte.

Se llama rutina de servicio de interrupción a la rutina que se ejecuta en respuesta a una solicitud de interrupción. Su forma de actuar es similar a la de una subrutina, debido a que rompe la secuencia de ejecución del programa, aunque es diferente porque el programador no sabe de antemano cuando se llevará a cabo.



El tratamiento de las interrupciones puede interferir en el proceso que se esté ejecutando en ese momento, por lo tanto hay que guardar el registro de estado del procesador y los registros accesibles por programa. Un detalle importante es que la CPU debe informar al periférico de que su solicitud ha sido reconocida para que éste elimine la señal de petición de interrupción (operación que se realiza generalmente accediendo a algún registro del periférico). Las interrupciones resultan particularmente útiles en sistemas operativos y en los procesos de tiempo real. A continuación se puede ver el diagrama de flujo de una operación de entrada/salida mediante interrupciones.



INCONVENIENTES DE LA E/S MEDIANTE INTERRUPCIONES.

Se sigue realizando la transferencia a través de la CPU y por lo tanto durante la transferencia la CPU queda bloqueada. Esto quiere decir que si el bloque de datos es muy grande, la CPU puede quedar bloqueada casi permanentemente

Además, la interrupción alivia al procesador de la espera, pero no se debe olvidar que la transferencia se hace byte a byte y por lo tanto debe haber una interrupción por cada byte que se desee transmitir. Recordar que las subrutinas de tratamiento de interrupción incluyen gran cantidad de instrucciones preparatoria para que la transacción se realice sin problemas.

GESTIÓN DE LAS INTERRUPCIONES

La llegada de una petición de interrupción tiene como efecto inmediato que la CPU suspenda la ejecución del programa e inicie la subrutina de tratamiento de la misma. La aceptación de una interrupción se realiza siempre al final de una instrucción (última microoperación), consultando el bit correspondiente, aunque en las instrucciones que son muy largas se puede consultar en diversos puntos de la instrucción mediante μoperaciones.

La CPU tiene capacidad para habilitar y deshabilitar interrupciones. Supongamos que las interrupciones no se pudieran deshabilitar, entonces podría ocurrir lo siguiente. Tras la petición de interrupción esta señal queda activada hasta que no llegue una señal de aceptación de interrupción. Entre tanto la CPU salta a la rutina de interrupción. En esta rutina, al final de cada instrucción se comprueba si existen interrupciones y como la señal de petición sigue activa se vuelve a solicitar una interrupción, luego entran en un bucle infinito.

La gestión correcta es la siguiente:

1. El dispositivo solicita la interrupción
2. La CPU interrumpe el programa
3. Se deshabilitan las interrupciones
4. Se informa al dispositivo que se ha reconocido su solicitud
5. Se realiza la acción solicitada
6. Se habilitan las interrupciones
7. Se reanuda el programa interrumpido

Existen tres formas de deshabilitar las interrupciones:

- La CPU las deshabilita durante la ejecución de la primera instrucción de la rutina de tratamiento.
- Se deshabilitan de manera automática
- La señal de petición de interrupción funciona por flanco no por nivel.

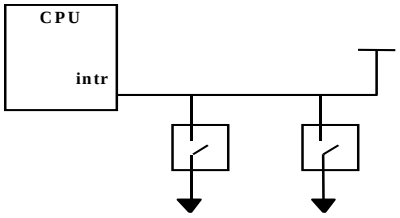
TIPO DE SISTEMAS DE INTERRUPCIONES: DIRECCIONAMIENTOS Y PRIORIDADES

Identificación de la fuente: Cuando un sistema tiene varios dispositivos que producen interrupciones se debe determinar cual de ellos interrumpe. Para ello existen dos técnicas por encuesta y por vector.

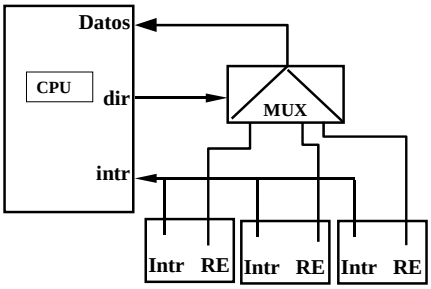
Prioridades. Cuando varios dispositivos de Entrada/Salida piden simultáneamente la interrupción se debe decidir cual se atiende primero. Existen dos técnicas ligadas al tipo de direccionamiento: por hardware y software.

Los sistemas con varios dispositivos de interrupción se pueden clasificar en sistemas con una línea de interrupción y con varias línea de interrupción.

Sistemas de una línea de interrupción. Son aquellos en los que todos los periféricos comparten una única línea de interrupción. Cuando un periférico hace una petición pone la línea a cero de manera que la línea de interrupción cumple la siguiente expresión lógica: Interrupción = not(intr1)not(intr2)not(intr3)



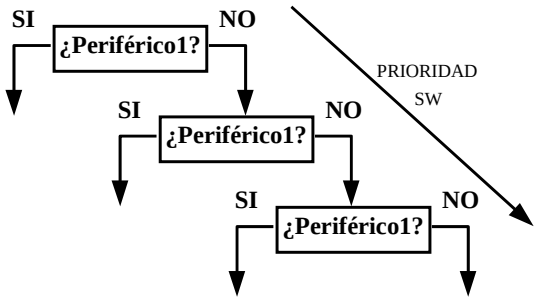
En este caso existen dos técnicas para la identificación de la fuente, por polling y por vector. En el polling se examina el registro de estado de cada periférico, en el orden que indique la subrutina de tratamiento de interrupción. Esta forma de determinar quien interrumpe fija la prioridad, que será de tipo software. El primer periférico por el que se pregunte en la rutina de tratamiento será el más prioritario, y así sucesivamente.



Modo de operación:

1. Llega una señal de interrupcion a la CPU por la línea INTR
2. La CPU ejecuta la subrutina de interrupción
3. Se van enviando direcciones sucesivas por la línea dir y para cada dirección se lee el registro de estado

El programa de tratamiento de interrupción tendrá un flujo similar al que aparece a continuación.



Las principales desventajas de esta técnicas son el tiempo que tarda en consultar el registro de estado de cada módulo de interrupción y que si el periférico n-esimo interrumpe mucho se realizan muchas consultas hasta llegar a él lo que implica que la CPU pierde mucho trabajo útil.

En la identificación vectorizada, la CPU no necesita ir preguntando uno a uno a los periféricos para ver cual a interrumpido sino que el propio dispositivo se identifica así mismo directamente o indirectamente indicando la dirección en la que comienza su rutina de tratamiento. ¿Cómo se determina la prioridad?. Se suele utilizar un protocolo de daisy chain (modulo de buses) cuyos pasos se ven a continuación:

1 El periférico que provoca la interrupción pone la línea INTR a 0

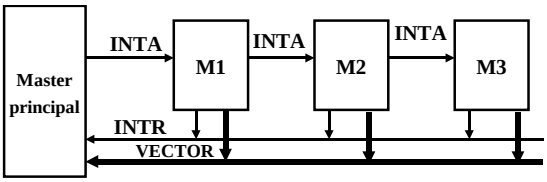
2 La CPU activa la señal INTA

3 Los periféricos que no han pedido interrupciones dejan pasar la señal INTA

4 El periférico que ha interrumpido impide el paso de la señal INTA y envía su código a la CPU.

Al código que envía el periférico a la CPU se le llama vector. Un vector puede ser un número que identifique al periférico, o una dirección de memoria en la que se encuentra la subrutina de interrupción o una dirección de memoria en la que se encuentra la dirección de memoria de la subrutina de interrupción (direccionamiento indirecto).

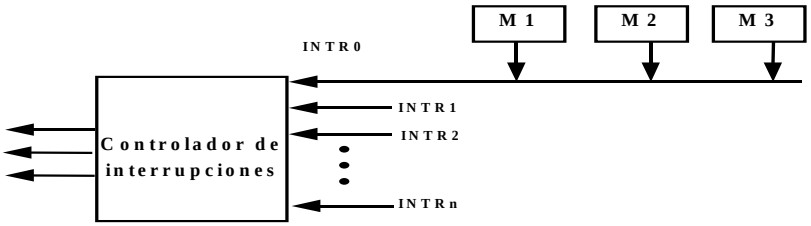
En ocasiones esta identificación del periférico se hace de manera implícita. En estos casos se dice que es una interrupción autovectorizadas. Estos dos conceptos de interrupción se verán con más claridad cuando se estudien las interrupciones del 68000. La prioridad en este caso es hardware y depende de la posición del periférico en la cadena:



Cuando el periférico da directamente la dirección de la rutina de interrupción, el sistema es demasiado rígido. El sistema con autovectores es más flexible.

En algunas ocasiones el vector de interrupción también contiene valores para el registro de estado de la máquina. Valores que suelen modificar el nivel de prioridades o deshabilita las interrupciones posteriores.

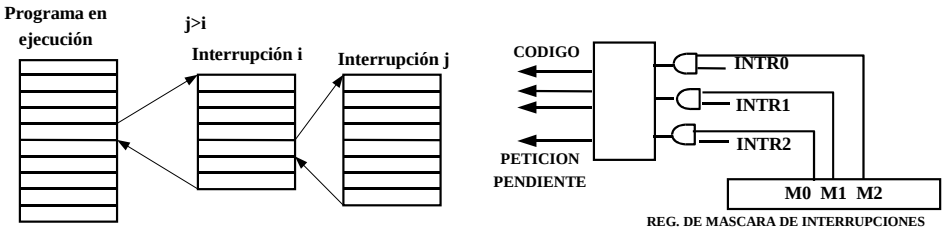
Los sistemas multilínea son aquellos en los que existen varias líneas de petición de interrupción. En cada línea, a su vez, pueden estar conectados varios periféricos. La forma de direccionarlos es sencilla. La activación de la línea de interrupción en sí misma indica la línea que interrumpe. En cuanto a las prioridades, las peticiones simultáneas en líneas diferentes se resuelve mediante un codificador de prioridades. Las peticiones simultáneas en la misma línea como se ha estudiado ya.



JERARQUÍA DE INTERRUPCIONES

Se ha visto como tratar las interrupciones cuando se producen varias simultáneamente, pero ¿qué ocurre cuando se está tratando una interrupción y llega otra?. Existen dos métodos complementarios que dependen de la prioridad de las interrupciones: las interrupciones anidadas y enmascaramiento de interrupciones.

Interrupciones anidadas. Suponer una jerarquía de interrupciones determinada por la prioridad de cada una de ellas I_1, I_2, I_3,…, I_n, teniendo mayor prioridad la de mayor índice. Si se está procesando una interrupción Ii y llega una petición de interrupción Ij, se interrumpe la rutina de Ii para ejecutarse las Ij solo cuando j>i, es decir solo cuando la prioridad de la que llega es mayor que la prioridad de la que se está tratando.



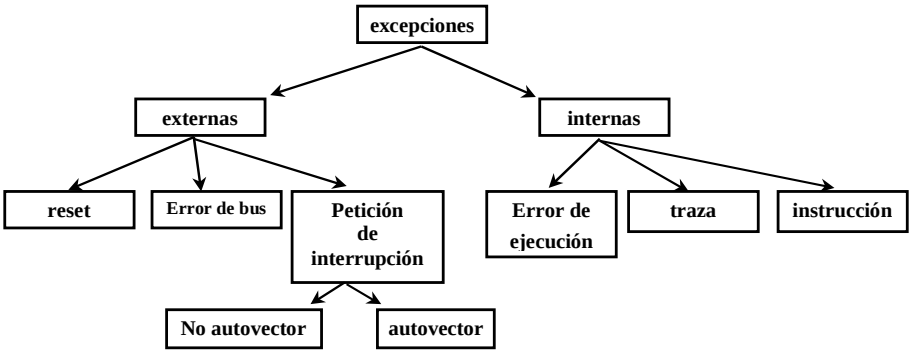
Ya se comentó con anterioridad, que lo primero que realiza la rutina de tratamiento de una interrupción es deshabilitar las interrupciones para evitar entrar en bucles infinitos. Esto tiene un problema: existen interrupciones que no pueden esperara hasta que se acaba de procesar otra interrupción, como es el caso de la señal de reloj del computador. Normalmente se asigna cierto nivel de prioridad a la CPU (programa en ejecución) de manera que sólo podrán interrumpir al programa interrupciones de mayor nivel. La prioridad de la CPU es parte del registro de estado y se puede controlar por programa

Enmascaramiento selectivo. En casi todas las máquinas se proporciona la capacidad de habilitar o deshabilitar interrupciones de manera selectiva. Esto se puede hacer bien por hardware o por sofware. En el caso hardware cada periférico o grupo de periféricos tiene un biestable de capacitación de interrupcion. Este biestable puede estar incorporado a la interfaz de dispositivo como un bit de registro de condición. En el caso software se detecta la interrupción en la rutina de identificación de la fuente pero no se trata.

En la siguiente figura se ve cómo realizar un enmascaramiento hardware de un nivel, donde M_0, M_1 y M_2 están en un registro llamado de máscara de interrupciones que se encuentra en la CPU.

INTERRUPCIONES DEL MC68000

En el MC68000 a las interrupciones se las llama de manera genérica excepciones y pueden ser externas e internas, según la señal venga del exterior o sea producida por un evento interno al mismo. Una clasificación más detallada se puede ver en el siguiente cuadro:



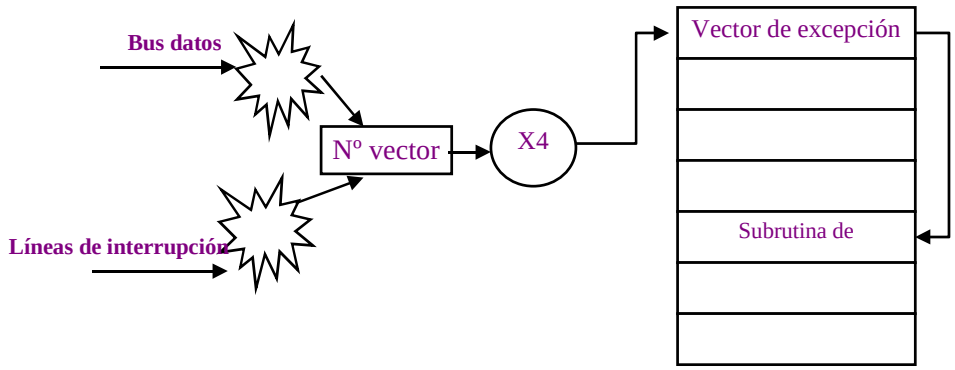
Cada excepción tiene asignada un número vector, y a cada número de vector le corresponde una dirección de memoria. En esta posición de memoria se encuentra la dirección en la que se almacena la rutina de tratamiento de la interrupción, es decir, es un direccionamiento indirecto. La dirección asociada a un número de vector se encuentra haciendo dir = nºvector·4. A continuación se pueden ver algunas de los vectores de excepción del MC68000.

Numero vector	**dirección**		**asignación**
	decimal	**hexadeci**	
2	8	008	Error de bus
3	12	00c	Error de dirección
4	16	010	Instrucción ilegal
5	20	014	División por 0
7	28	01c	Instrucción trap
9	36	024	traza
15	60	03c	Vector instrucción no inicializado

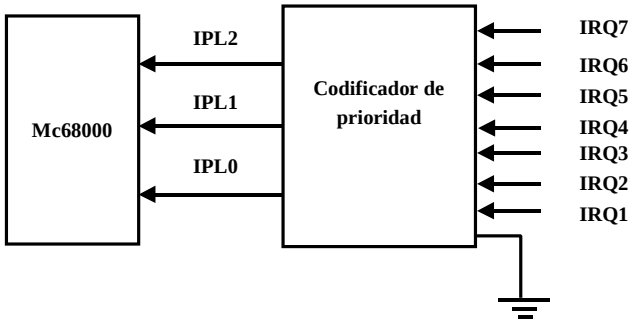
25	100	064	Autovector nivel 1
26	104	068	Autovector nivel 2
27	108	06c	Autovector nivel 3
28	112	070	Autovector nivel 4
29	116	074	Autovector nivel 5
30	120	078	Autovector nivel 6
31	124	07c	Autovector nivel 7
64-255	256-1020	100-3fc	Vectores interrupción de usuario

Cuando al MC68000 le llega una señal de excepción busca el vector correspondiente, y tras salvar en la pila los registros PC y SR bifurca a la dirección contenida en él. Este microprocesador tiene 255 posibles vectores de excepción almacenados en *la tabla de vectores de excepción* que se encuentra en la dirección $00000. De estos 255 vectores, los 64 primeros están preasignados mientras que el resto quedan libres para ser asignados a periféricos como vectores de interrupción de usuario. La secuencia de procesamiento de una excepción por el microprocesador es la siguiente:

1. Se realiza una copia del registro SR actual en un registro interno. A continuación se activa el bit S del registro SR para pasar a modo supervisior y se desactiva el modo traza poniendo el bit T a cero. Para excepciones correspondientes a interrupciones se actualiza el valor de la máscara I_2, I_1, I_0.
2. Se determina el número de vector de interrupción. Si es una excepción cuyo vector esta preasignado su valor se determina mediante lógica interna. Si es una interrupción de usuario (que no tienen un vector asociado) se realiza una búsqueda llamada reconocimiento de interrupción. A partir del número de vector encontrado se genera la dirección del vector de excepción.
3. Se salva el valor actual del contador de programa en la pila. A continuación se salva el valor de SR que se había guardado en el primer paso. Como se está en modo supervisión se usará el puntero de pila SSP.
4. Se carga en PC el valor de la dirección contenida vector de excepción.
5. La última instrucción de la rutina de tratamiento de interrupción debe ser la RTE similar a las de retorno de subrutina.



Niveles de Prioridad de interrupción

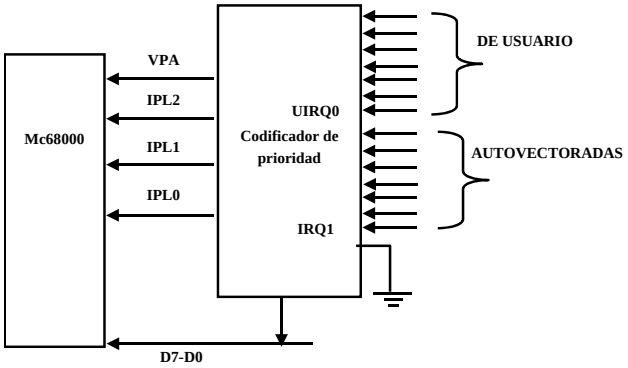


El MC68000 tiene tres entradas que permiten realizar peticiones de interrupción, con siete niveles de prioridad diferentes. El valor 000 se reserva para la ausencia de interrupciones. Gracias al codificador de

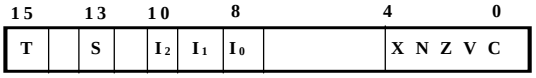
prioridad se pueden conectar siete líneas externas. El MC68000 tiene dos formas de proporcionar el vector de excepción mediante autovector y mediante usuario.

El MC68000 detecta el tipo de interrupción consultando una línea llamada VPA* al inicio del ciclo de interrupción. Si la línea no está activada (a uno) la interrupción será autovectorada. Para una interrupción autovectorada el MC68000 toma la dirección de la rutina directamente del autovector asociado al nivel de prioridad de la interrupción (vectores del 25 al 31). Se usa para periféricos de la familia mc6800.

Si se trata de una interrupción de usuario(VPA* a cero) el número de vector de excepción (comprendido entre 64-255) se obtiene del periférico a través de las líneas D0-D7 del bus de datos.



Enmascaramiento. El registro de estado del procesador incluye tres flags denominados I_2, I_1, I_0 que codifican la máscara de interrupciones activa en ese momento. La máscara identifica un nivel de prioridad a partir del cual, las peticiones de interrupción no van a ser atendidas. Es decir las peticiones de interrupción con un nivel igual o inferior a la máscara no van a producir excepción.

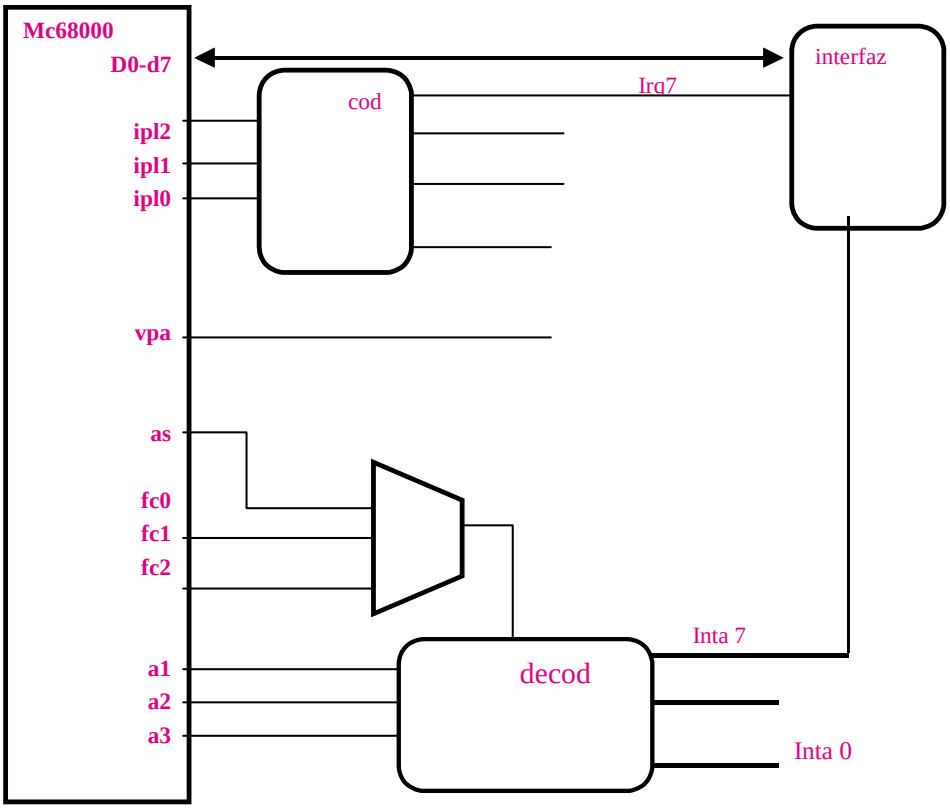


Tratamiento de una interrupción. Se produce una interrupción cuando un periférico activa una señal de interrupción (activas a la baja). Para que se trate debe estar capacitado el nivel de interrupción correspondiente. Lo primero que hace el mc68000 es determinar el tipo de interrupción y obtener el número de vector correspondiente. A continuación se realiza el tratamiento de excepciones ya visto con anterioridad, con la diferencia de que se actualiza la máscara de interrupciones al nivel de la máscara que se está tratando, para impedir que dispositivos con igual o menor prioridad interrumpan. Sí se permite que una interrupción de mayor nivel interrumpa. En este caso se produce un anidamiento de interrupciones.

Identificación de la fuente de interrupción. Cuando solo hay una fuente de interrupción conectada a cada línea, la identificación de la fuente es automática. Este caso no es lo habitual, generalmente por cada línea interrumpen varios periféricos. Para detectar cual ha sido el periférico que interrumpe se debe emplear una de las dos técnicas vistas: polling y hw.

En el caso de interrupciones autovectorizadas, se debe utilizar el polling puesto que no hay manera de recibir la identificación del periférico. En este caso el orden de prioridad vendrá dado por el orden de consulta en el programa. La rutina de servicio de interrupción debe asegurarse de borrar el flag de petición de interrupción del periférico solicitante.

Para las interrupciones de usuario se utiliza la técnica de daisy –chain en la que se envía una señal de reconocimiento de interrupción que va recorriendo todos los dispositivos. Cuando el periférico solicitante recibe la señal INTA envía por el bus de datos el número de vector.



As dirección valida
Fc estado del computador

10.7 EL ACCESO DIRECTO A MEMORIA (DMA)

Las técnicas de escrutinio y de interrupciones sólo son útiles para transferencias de pequeño ancho de banda. En estos casos, el peso de la transacción la sigue llevando la CPU. Pero cuando hay que realizar grandes intercambios de información entre un dispositivo de alta velocidad y la CPU, su utilización puede degradar enormemente el rendimiento del sistema. Este es el caso de los discos duros en los que el ancho de banda es grande (grandes bloque de información que pueden contener cientos o miles de bytes). Este es el motivo por el que los diseñadores de computadores inventaron un mecanismo para descargar al procesador de trabajo y lograr que el controlador del dispositivo transfiriese datos directamente a/o desde la memoria, sin involucrar al procesador. A este mecanismo se le denomina Acceso Directo a Memoria (DMA).

En esta nueva técnica todavía se utilizan las interrupciones para comunicarle al procesador que ha acabado la transferencia o para comunicarle un error. El ADM se implementa con un controlador especializado que transfiere datos entre un dispositivo de entrada/salida y la memoria independientemente del procesador.

MODO DE OPERACIÓN

1. Cuando la CPU desea leer o escribir un bloque de datos, envía un comando al controlador de DMA comunicándole la siguiente información:
   - Tipo de operación (si es entrada o salida)
   - Dirección del módulo de Entrada/Salida
   - Primera posición de la memoria a la que se desea acceder
   - Nº de palabras a leer o escribir
2. A continuación la CPU reanuda su trabajo, y es el controlador de DMA el encargado de manejar la E/S.

3. Cuando la transferencia ha terminado el controlador de DMA envía una señal de interrupción a la CPU.

Como se puede ver la CPU solo interviene al principio y al final de la operación

TÉCNICAS DE IMPLEMENTACION DE DMA

Existen dos técnicas de implementación de ADM: memoria multipuerto y el robo de ciclo

Memoria multipuerto. Un puerto de una memoria está compuesto de un bus de direcciones, un bus de datos y un bus de control. Como la memoria tiene más de un puerto, se utiliza uno de ellos para la conexión con la CPU y el otro para la conexión con el periférico a través del controlador DMA.

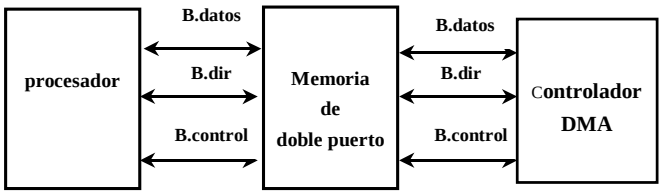
En esta implementación los periféricos acceden a la memoria principal sin intervención de la CPU, para ello la memoria suele dividirse en bloques que permiten el acceso paralelo, de tal manera que las peticiones de acceso a la memoria pueden tratarse en paralelo mientras no direccionen el mismo bloque.

Aunque es la implementación más eficiente, tiene un elevado coste debido a lógica necesaria para implementar cada puerto, y al dispositivo de gestión de prioridades de acceso a la memoria por cada uno de los puertos. El controlador de DMA debe generar señales de

- Dirección de memoria
- Dato
- Señal de Lectura/Escritura
- Inicio de ciclo

La memoria debe contestar con señales

- Datos
- Fin de ciclo que realiza la sincronización

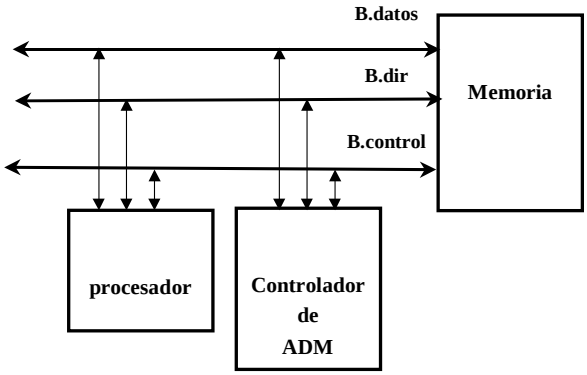


Robo de ciclo. La memoria sólo tiene un puerto al que deben acceder el DMA y la CPU, y por lo tanto ambos módulos deben compartir el bus de memoria. Esto hace al sistema más económico, ya que las memorias multipuerto son muy caras. Como inconveniente tiene que la CPU y el DMA deben ponerse de acuerdo para obtener el control del bus. En realidad es la CPU la que cede el control al DMA. El modo de operación general:

- El procesador envía al DMA la información sobre la transferencia a realizar. En este momento el DMA actúa como esclavo, puesto que el procesador controla la DMA como un periférico más.
- El procesador retorna a sus tareas mientras el DMA prepara la transferencia.
- Cuando el DMA necesita acceder a la memoria pide al arbitro el control del bus.
- El arbitro, que es el procesador, concede el control al DMA que pasa a actuar como master
- Cuando el DMA completa la operación de Entrada/Salida advierte al procesador con una interrupción

El procesador reconoce y trata la operación.

La CPU sólo puede aceptar robos de ciclo cuando está al final de una de las fases que forman la ejecución de la instrucción. Esto tiene dos consecuencias, por un lado el periférico debe esperar a que le concedan el bus y, por otro lado, el robo de ciclo puede aceptarse en medio de una instrucción.



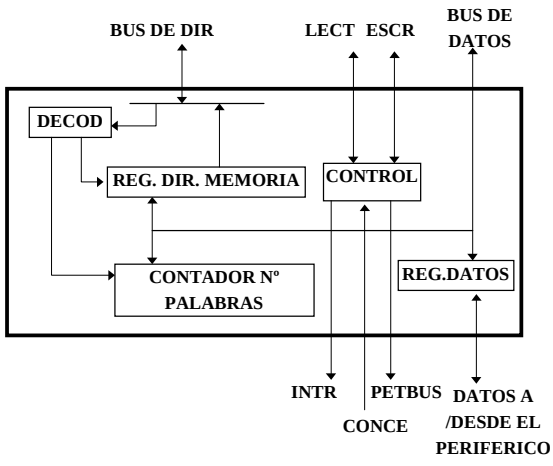
Existen tres implementaciones diferentes para el robo de ciclo, que se detallan a continuación. *Robo de ciclo sencillo*. El DMA solo toma el control en un ciclo sencillo. Una vez que tiene el control de bus trasmite una sola palabra, devuelve el control de bus al procesador y solicita nuevamente el bus así hasta que se acaba la operación.

*RÁFAGAS***.** El DMA toma el control del bus hasta que acaba la transferencia de todo el bloque. Esta implementación tiene la ventaja de la alta velocidad de transferencia, y el inconveniente de que puede llegar a parar al procesador.

En general el DMA se suele utilizar para entradas síncronas de alta velocidad (lectura de cintas magnéticas) con velocidades próximas a las de la memoria principal . El inconveniente de estas dos técnicas es la degradación del rendimiento del procesador porque el bus es compartido por la CPU y el DMA, luego cuando no lo ocupa la CPU lo ocupa la DMA.

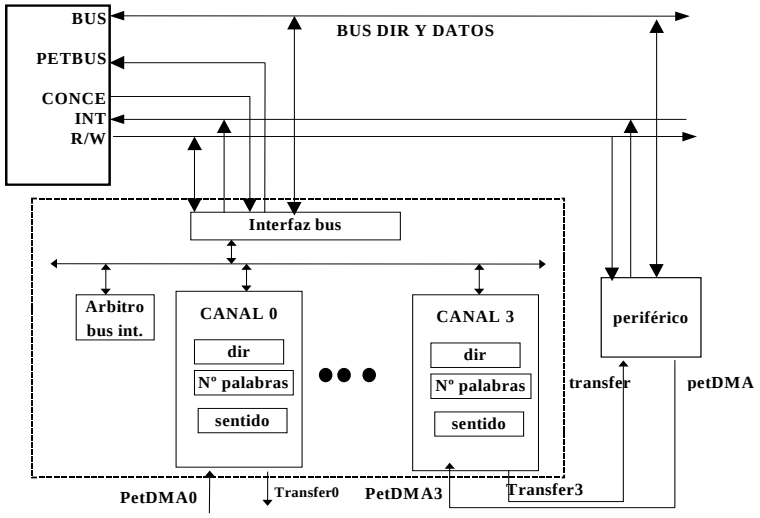
Existe una solución a este problema. Debe tenerse en cuenta que el procesador no necesita usar el bus en todas las fases de ejecución de una instrucción, luego el bus se cede al DMA solo en las fases en las que no lo utiliza la CPU. A este modo de implementación se le llama *Transparente***.** En él**,** el ADM accede al bus sólo cuando no lo usa la CPU por lo tanto la velocidad de trabajo de la CPU no se ve alterada significativamente, aunque transferencias son algo más lentas. Además, al utilizar caches el procesador puede evitar los accesos directos a la memoria la mayor parte del tiempo dejando libre el ancho de banda de la memoria para el ADM

ESTRUCTURA DE UN CONTROLADOR DE DMA



Esta estructura es consecuencia del comportamiento funcional descrito anteriormente. En ella se puede observar el funcionamiento del DMA como esclavo con los datos entrando al controlador, el funcionamiento del DMA como maestro en el que las direcciones salen del registro y señales de control de Lectura/Escritura y las líneas de arbitraje del bus.

Estructura de un controlador de DMA con 4 canales independientes:



Para reducir la necesidad de interrumpir al procesador y ocuparlo en el tratamiento de una petición de entrada/salida, el controlador de ADM puede hacerse más inteligente. Los controladores inteligentes con frecuencia se denominan procesadores de E/S o controladores de canal. Estos procesadores especializados ejecutan básicamente una serie de operaciones de entrada/salida, denominada programa de entrada/salida. Este programa puede almacenarse en el procesador de entrada/salida o puede almacenarse en la memoria.

Cuando se utiliza un procesador de entrada/salida, el SO normalmente inicia un programa de entrada/salida indicando las operaciones a realizar así como el tamaño y la dirección de la transferencia para cualquier lectura/escritura. El procesador de entrada/salida realiza entonces las operaciones del programa e interrumpe al procesador solo cuando se completa el programa entero. El procesador DMA suelen ser de propósito especifico (chips simples no programables) mientras que los procesadores de entrada/salida se implementan habitualmente con microprocesadores de propósito general que corren programas especializados de entrada/salida [Patterson pg510]

EJEMPLOS DE CONTROLADOR DMA: MC68440 Y MC68540

[Julio Septien] Se utilizan cuando las transferencias tienen que realizarse en forma de bloques de datos. El controlador de DMA puede ser accedido por el MC68000 como un dispositivo interfaz esclavo más, para ser cargado con la información relativa a la transferencia. Y también puede actuar como un maestro que solicita ciclos de bus del MC68000. El número de canales de un controlador determina el número de periféricos que pueden estar realizando operaciones DMA en paralelo.

Motorola ofrece varios controladores de DMA (DMAC) compatibles con el MC68000, como son el MC68440 de dos canales y el MC68450 de cuatro canales. Ambos controladores disponen de las señales necesarias para implementar los protocolos de bus y de arbitraje. Con el fin de minimizar el número de patillas de entrada/salida, emplean buses de direcciones y datos multiplexados en el tiempo: la misma línea es compartida por D_0-D_{15} y A_8-A_{23}. Para cada periférico controlado se utilizan tres líneas específicas y dos compartidas con el resto.

- **REQi*** . Es la línea de línea de petición y es de entrada. Procedente de la interfaz conectada al canal i que solicita una transferencia DMA.
- **ACKi***. Es la señal de reconocimiento y es de salida. Se activa cuando el DAMC recibe el control del bus y le corresponde a la interfaz conectado al canal i una transferencia DMA
- **PCLi** son líneas bidireccionales de control del periférico. Son líneas que pueden programarse para diferentes funciones, por ejemplo, como señal de reloj o como señal de preparado de un dispositivo lento.

Las líneas compartidas entre los canales son:

- **DTC*** sirve para indicar que la transferencia con dispositivo ha sido completada, por lo tanto indica el fin señal que indica el final de un ciclo DMA (ciclo DMA transferencia de un byte).
- D**ONE*** Es una línea bidirecional que indica que se ha completado la transferencia de un bloque.

Los DMCA disponen de un registro de control general GCR de 8 bits y de un conjunto de 17 registros específicos para cada canal. Entre estos registros destacan:

- Registro de estado
- Registro de control
- Registro de interrupciones
- Registro de modo de operación de canal

Registro de direcciones de memoria MAR de 32 bits que almacena la dirección de memoria a la que debe accederse en el próximo ciclo DMA del canal. Después de cada transferencia su valor se incrementa en el tamaño de dato transferido.

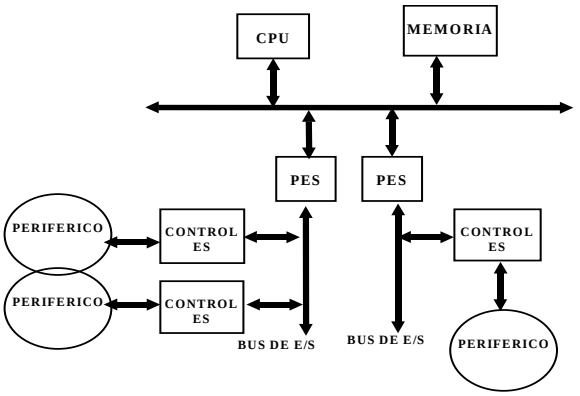
Registro de direcciones de dispositivo. DAR de 32 bits que almacena la dirección en donde se encuentra ubicado el registro de la interfaz al que se va a acceder.

Contador de transferencia de memoria MTC de 16 bits que es cargado inicialmente por el mc68000 con el número de datos a transferir. Se decrementa en uno después de cada ciclo DMA. Cuando se alcanza el valor 0, y siempre que el controlador haya sido programado para ello se genera una petición de interrupción del MC68000. La interrupción es de tipo no autovectorada, y cada canal almacena dos vectores de interrupción, uno para la terminación normal de la transferencia y otro para el caso de error.

10.8 PROCESADORES DE ENTRADA/SALIDA (PES)

[Stalling] Aparecen en los grandes sistemas computadores por razones de rendimiento. Estos procesadores auxiliares dedicados a la E/S permiten dedicación máxima de la CPU a las tareas de computación. A los procesadores en IBM se les llama canales.

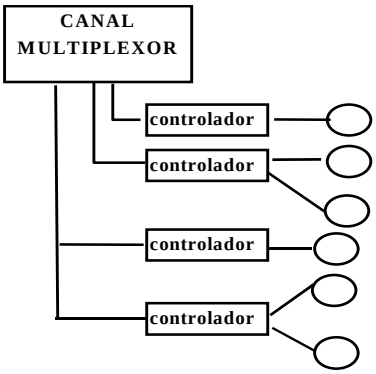
PES es un procesador con un repertorio limitado especializado en la realización de operaciones de E/S, que es supervisado por un procesador central. Ejecutan programas de Entrada/Salida almacenados en la memoria del procesador central. Por lo tanto en la memoria principal coexisten programas y datos del procesador y programas del PES con formatos máquina diferentes. A continuación podemos ver la estructura general de un sistema PES.



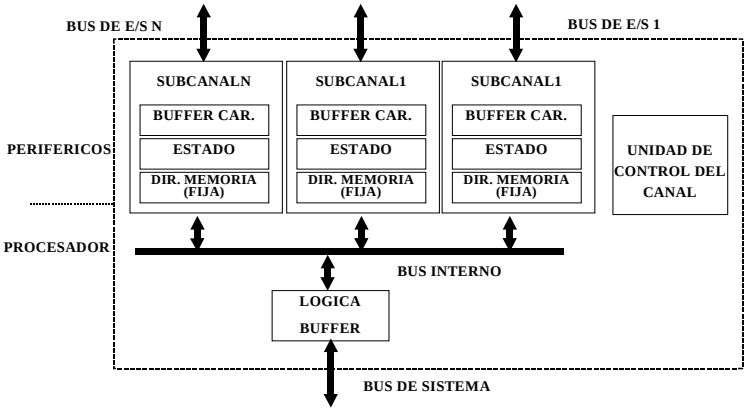
Hay tres tipos de procesador de entrada/salida: Multiplexor, Selector Multiplexor por bloques

PES MULTIPLEXOR

Se suele utilizar en la conexión de periféricos de velocidad media-baja como son las terminales de impresora. Intercambia datos con el procesador central a una velocidad mucho mayor que los periféricos. Su modo de funcionar consiste en atender alternativamente los distintos periféricos conectados, durante cortos periodos de tiempo. Su estructura general es



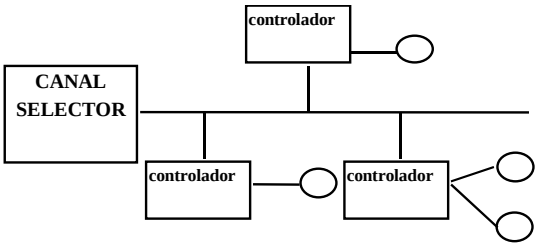
Esquema interno del canal multiplexor:



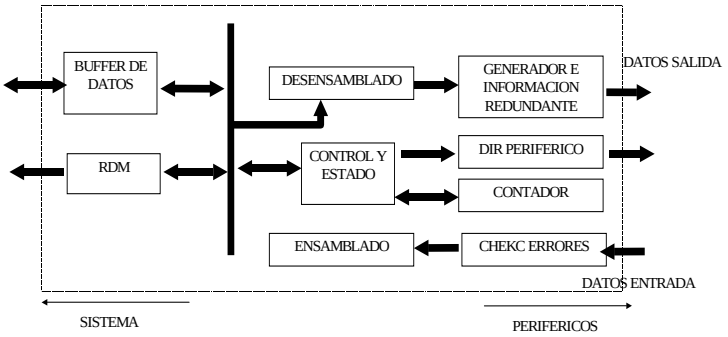
Donde dir. Memoria (fija) indica la dirección de memoria en la que se guardan los parámetros de la transferencia. La información relativa a la transferencia que gestiona cada subcanal (dirección inicial y nº de palabras) está almacenada en una zona fija de memoria a la que el canal puede acceder. El número de canales determina el número de operaciones de entrada/salida simultáneas.

PES SELECTOR

Controla varios dispositivos de alta velocidad. Su modo de funcionamiento es el siguiente, cuando comienza una transferencia con un periférico la acaba antes de atender a otro. A diferencia del multiplexor, mantiene los parámetros de la transferencia en registros internos. El esquema general es el siguiente:



- Estructura interna del canal selector:



Tanto en el canal multiplexor como en el selector hay unos módulos llamados ensamblador-desensamblador cuya misión es convertir los bytes al tamaño de palabra de computador y viceversa (en el caso que los tamaños difieran)

CANAL MULTIPLEXOR POR BLOQUES

Trata de combinar las características del multiplexor y del selector. Su modo de funcionamiento: comunicación multiplexada con varios dispositivos bloque a bloque. Su principal **Ventajas** es aprovechar las fases mecánicas de los dispositivos de alta velocidad para atender otras peticiones.

11 INTERFACES PROGRAMABLES

La misión de los circuitos de interfaz de E/S es la conversión de los formatos de datos de los periféricos a formatos aceptables por el procesador, y la sincronización de transferencias. Los factores que dificultan el diseño de interfaces son las diferencias de formatos entre la CPU y los periféricos, las diferencias de velocidad y las diferencias de niveles eléctricos. A continuación aparece un cuadro con los diferentes tipos de interfaz.

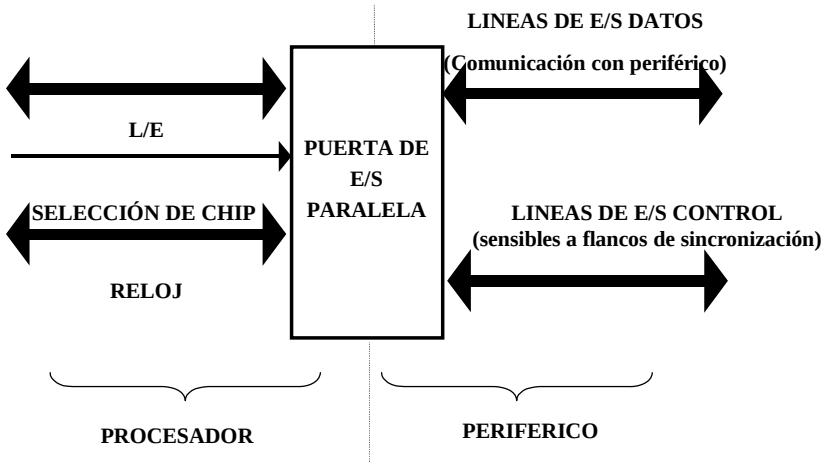
<table>
<tr><th colspan="3"></th><th>TRATAMIENTO DE LOS DATOS EN EL INTERFASE</th></tr>
<tr><td rowspan="3">FORMATO DE DATOS EN LOS PERIFERICOS</td><td colspan="2">ANALOGICOS</td><td>E: CONV A/D
S: CONV D/A</td></tr>
<tr><td colspan="2">DIGITALES</td><td rowspan="2">E: CONV S/P
S:CONV P/S</td></tr>
<tr><td>SERIE</td><td>PARALELO</td></tr>
</table>

El motivo de la aparición de las interfaces programables fue el costo en tiempo y la complejidad del diseño de interfaces específicas para cada periférico. La solución al problema fueron los circuitos integrados de interfaz universales. Esto se consigue haciéndolos programables para adaptarlos a una aplicación concreta. La forma de programarlos es escribir en algunos de sus registros.

Las ventajas de este tipo de interfaces son su versatilidad de utilización en diversos entornos con sólo alterar el contenido de sus registros, la fiabilidad del sistema debido a la reducción del número de componentes y, como efecto colateral, la estandarización de periféricos al intentar adaptarse estos a las características de las interfaces programables.

11.1 INTERFASE DE E/S PARALELO

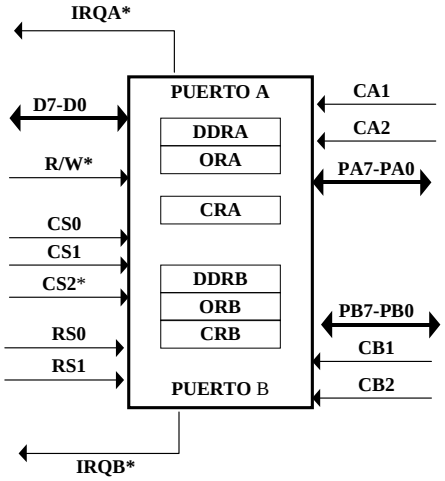
Su misión consiste en acoplar las velocidades de transferencia y en sincronizar el procesador y el periférico. Su estructura general y las señales de interfaz son:



Las líneas de ES de datos y de control son programables como de entrada o salida mediante un registro. Se pueden programar para que ciertos flancos (de subida o bajada) alteren el contenido de los registros de la puerta y/o provoquen peticiones de interrupción.

EJEMPLO DE ES PARALELA: MC6821 PIA (Peripheral Interface Adapter)

Es una interfaz paralela programable de 8 bit que puede conectarse al MC68000 a través de un bus síncrono para periféricos del MC6800



LA PIA dispone de dos puertos de entrada/salida paralelos muy parecidos, identificados como A y B. Cada puerto consta de los siguiente elementos:

- Un registro de datos ORX que funciona como emisor y receptor de datos hacia o desde las líneas exteriores. Su dirección depende del contenido del registro DDRX.
- Registro de dirección de datos, DDRX. Cada bit del registro indica como se comporta el bit correspondiente del ORX.
- Dos líneas de control CX1, CX2 para examinar el estado de los dispositivos periféricos y generar señales de control sobre los mismos.
- Un registro de control CRX que contiene información de estado. Determina el funcionamiento de las líneas CX1 y CX2, cuando actúan como entradas y su capacidad para generar peticiones de interrupciones.

RS1	RS0	CRA2	CRB2	REGISTRO SELECCIONADO
0	0	0	X	DDRA
0	0	1	X	ORA
0	1	X	X	CRA
1	0	X	0	DDRB
1	0	X	1	ORB
1	1	X	X	CRB

Por otro lado, la señal de entrada RS1 selecciona el acceso al puerto A o B, y la RS0 indica si se accede a uno de los registros de datos (OR o DDR) o al de control CRX. El acceso a ORX o DDRX viene determinado por el bit 2 de CRX

Registro de datos:

Cada bit individual del registro ORX va asociado a una línea física exterior (PX0-PX7) que puede programarse como de entrada salida escribiendo un 0 un 1 en el registro DDR.(Entrada bit 0,Salida bit 1)

Registro de control CRX

7	6	5	4	3	2	1	0
IRQX1	IRQX2	CONTROL CX2			DDRX ORX	CONTROL CX1	

REGISTRO DE CONTROL CRX

- Los bits 7 y 6 son flags que se activan cuando se detecta la transición programada por la línea correspondiente (para CX2 solo cuando se programa como entrada).

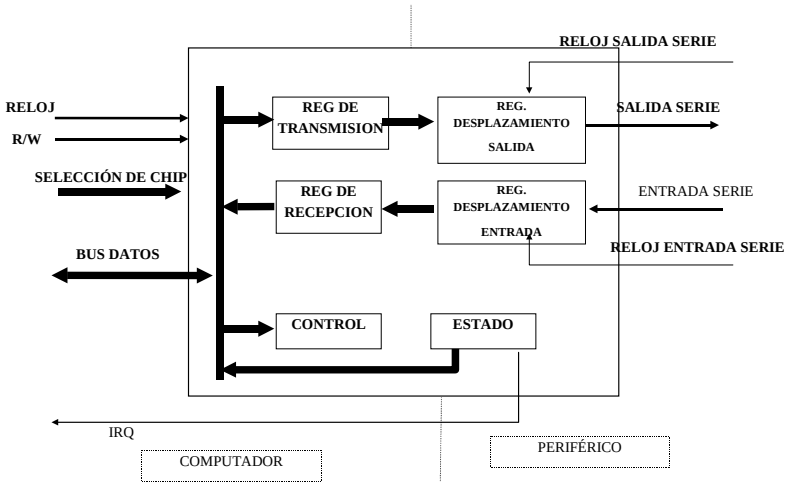
  ⇒ El bit 7 corresponde a CX1

⇒ El bit 6 corresponde a CX2

- Los bits del 5 al 3 programan las líneas CX2.
  - ⇒ El 5 si es indica si es de entrada (0) o de salida (1).
  - ⇒ Si es de entrada el 4 determina el tipo de transición. Positiva(1) o negativa(0).
  - ⇒ El 3 capacita la petición de interrupción
- El bit 2 determina la selección de acceso a DDRX/ORX
- bits 0 y 1 programan Cx1.
  - ⇒ El bit 1 determina si la transición que detecta el flag es positiva o negativa
  - ⇒ El bit 0 capacita la interrupcion.

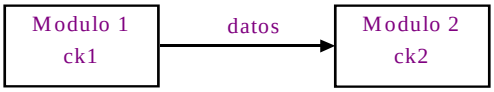
11.1.1 ENTRADA/SALIDA SERIE

La transmisión de datos paralela permite el envío de información a cortas distancias de forma rápida. Sin embargo, el transmitir información a distancias medias o grandes, supone un elevado coste debido a la necesidad de utilizar cableado resistente a errores de transmisión cada vez más probables al aumentar la longitud de las conexiones. Esto hace que se opte por la transmisión serie. La medida del rendimiento se realiza en baudios que son bits por segundo. En la siguiente figura se puede ver una estructura típica de una puerta serie:



En la transmisión serie el tiempo asignado a cada bit se mide con una señal específica que determina la frecuencia de transmisión. El receptor debe muestrear la línea con una frecuencia igual a la de transmisión para que no haya errores. Según el método que se utilice para generar la señal de reloj podemos clasificar las transmisiones en síncronos y en asíncronos.

Transmisión asíncrona es aquella que utiliza relojes diferentes de la misma frecuencia en el emisor y el receptor. Tiene el problema de que siempre acaba introduciendo desincronizacion al cabo de unos pocos bits. Para evitar esto generalmente sólo se transmiten los 10 o 12 bits necesarios para representar un carácter. El siguiente carácter se transmite un tiempo arbitrario después, luego necesita de una nueva sincronización. La frecuencia de trabajo es inferior a 20.000 baudios.

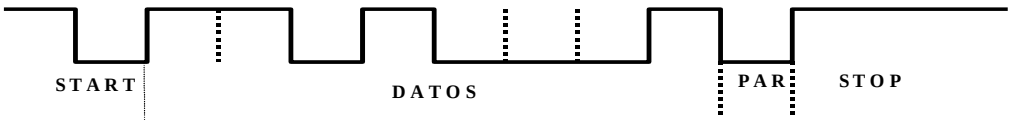


Transmisión síncrona utiliza la misma señal de reloj. El problema consiste en que es imposible enviar una señal de reloj a distancia sin que se produzcan distorsiones. La única forma posible es codificar las señales de reloj en los propios bits de datos que se transmiten. Esta técnica ayuda a mantener la sincronización durante la transmisión de bloques de bits de tamaño considerable, permitiendo mayores velocidades de transmisión de la información.

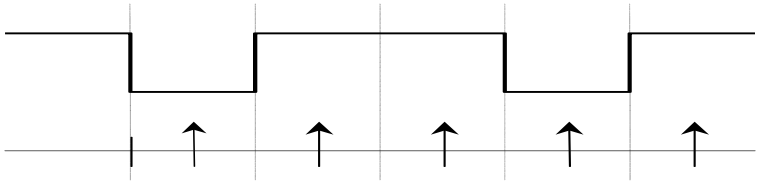
TRANSMISIÓN SERIE ASINCRONA

Es una transmisión orientada a la transmisión de caracteres. Los interfaces añaden algunos bits para asegurar la sincronización y para facilitar la detección de errores de transmisión. Hay que fijar un determinado formato de carácter como el que se explica a continuación:

- Un bit START siempre a 0
- De 5 a 8 bits para representar el dato dependiendo del código empleado
- Un bit de paridad, que sigue al último bit de datos y que ayuda a detectar errores simples.
- Bit de parada que consiste en uno o varios bits a 1



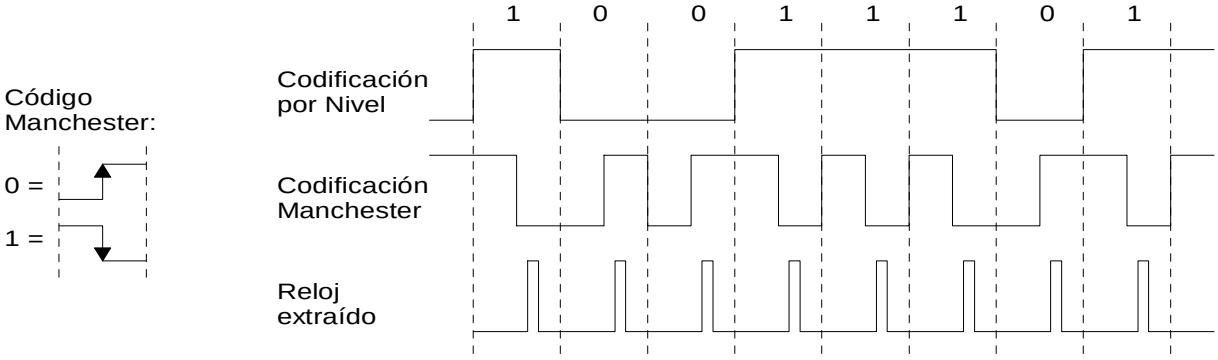
Después del bit de parada la línea queda a 1 de esta forma se detecta con facilidad el bit de arranque. En la transmisión asíncrona sigue existiendo el problema de cuando muestrear en el receptor. Ya hemos indicado que no pueden tener un reloj común luego debe existir un reloj en cada módulo**,** y lo normal es que sus frecuencias no sean idénticas. La solución consiste en que el receptor use un reloj múltiplo de la frecuencia de transmisión de manera que se detecte el centro del bit start con lo que el muestreo coincide siempre aproximadamente en el centro de cada bit de información. En la figura las flechas indican los instantes de muestreo del receptor.



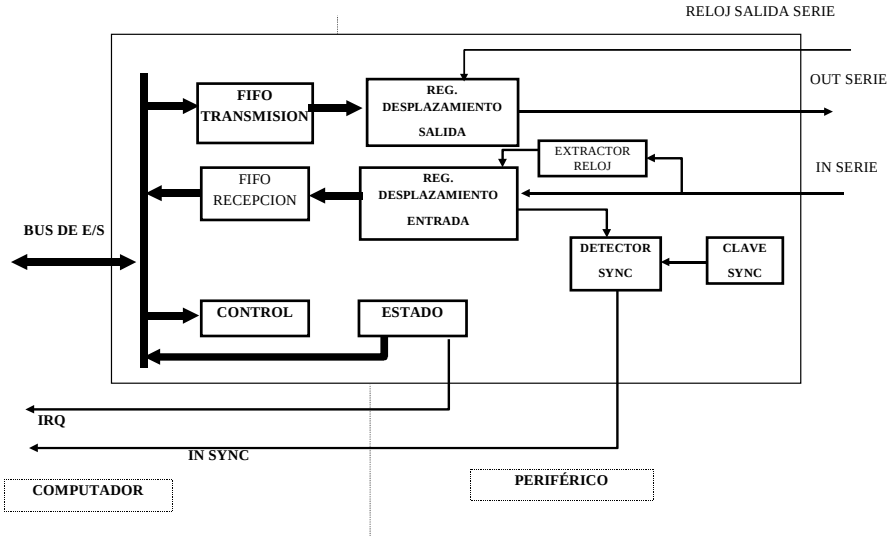
EJEMPLO INTERFAZ SERIE ASINCRONO MC6850 ACIA

TRANSMISIÓN SERIE SÍNCRONA

Al contrario de la transmisión asíncrona, que está pensada para el intercambio de caracteres, la síncrona está pensada para el intercambio de bloques de caracteres. Se basa en una codificación que permita incorporar la señal de reloj en los propios datos que se transmiten. Esto puede conseguirse mediante un código que en lugar de identificar los bits por nivel los identifique mediante transiciones, como el código Manchester que se muestra a continuación.



Los protocolos síncronos más extendidos son el BISYNC (BINARY SYNCHRONOUS COMMUNICATIONS) que está algo obsoleto. Lo usan equipos IBM. Usa 1 o 2 caracteres de sincronización que determinan el comienzo del bloque; el HDLC y el ADCCP. A continuación podemos ver una estructura típica



11.1.2 CONFIGURACIONES DE CONEXIÓN

Existen dos formas de conectar los módulos de entrada/salida y los dispositivos externos la Punto a punto y la Multipunto.

CONFIGURACIÓN PUNTO A PUNTO

Tienen una línea dedicada entre el módulo de entrada/salida y el periférico. Se utiliza en pequeños sistemas como PC o estaciones de trabajo. Para teclados, impresoras o modems. Ejemplo típico es el eia-232

MULTIPUNTO

En esta configuración una línea conecta un modulo de entrada salida con un conjunto de periféricos. Generalmente se utiliza para dispositivos de almacenamiento masivo como cintas o discos magnéticos y para los dispositivos multimedia, cdroms vídeo audio, de hecho se comporta como un bus externo. Los protocolos y arbitrajes son similares a los ya vistos en el capítulo de buses. Ejemplo SCSI y p1394

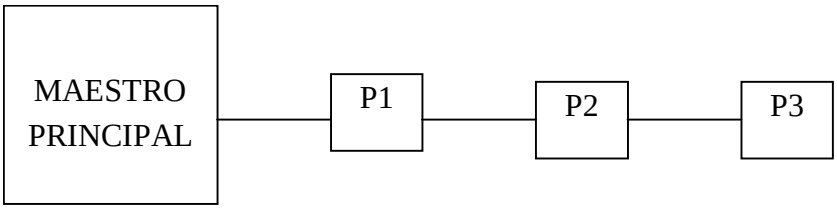
11.1.3 EJEMPLO DE BUS DE ENTRADA/SALIDA

SMALL COMPUTER SYSTEM INTERFACE (SCSI) (stalling pg207)

Fue Macintosh (1984) la que la desarrolló. Actualmente lo usan los Macintosh, los sistemas compatibles pc y las estaciones de trabajo. Es una interfaz estándar para:

- CDROM,
- Equipos de audio,
- Para almacenamientos masivos

Es un interfaz paralelo con 8,16 o 32 líneas de datos . Aunque se suele nombrar como bus en realidad es una configuración de tipo Daisy Caín (como el de la figura) cada dispositivo SCSI tiene un conector de entrada y uno de salida



Todos los sistemas funcionan independientemente y se puede intercambiar datos entre ellos y con el host. La longitud del cable es de 25 metros, se pueden conectar un total de 8 dispositivos. Cada dispositivo tiene asignada una dirección única como parte de su configuración.

EJEMPLO DE BUS DE ES SERIE: P1394

(STALLING PG 215)

Es un bus serie de alto rendimiento, se caracteriza por su alta velocidad, su bajo coste y su sencillez de implementación. Lo usan sistemas electrónicos que no son computadores: cámaras digitales, VCR, televisores. Transporta imágenes digitalizadas. Las ventajas de la transmisión serie son, tiene menos hilos, menos pines que se pueden doblar y romper, evita transferencias entre hilos y evita problemas de sincronización entre las señales

Proporciona una interfaz de IO con un conector simple que puede manejar numerosos dispositivos a través de un puerto sencillo.

Configuración:

daisy chain con 63 dispositivos conectados

usando puentes entre diferentes buses se pueden conectar hasta 1022 dispositivos

permite conectar y desconectar dispositivos sin tener que desconectar el sistema o reconfigurarlo

- Configuración automática
- No necesita terminadores
- No tiene que ser un daisy chai estricto puede utilizar una estructura de árbol
- Especifica tres capas de protocolo para estandarizar la forma en que el host actúa con los periféricos.

12 DISPOSITIVOS PERIFÉRICOS

12.1 DISCOS MAGNETICOS

DEFINICIONES Y CARACTERÍSTICAS

Son los almacenamientos exteriores más rápidos, tienen forma circular y están fabricados de metal o plástico recubierto de material magnético. Los datos se escriben/leen mediante un rollo conductor llamado cabeza.

El mecanismo de escritura se basa en los campos magnéticos producidos por el flujo de corriente en el rollo conductor generando patrones magnéticos en la superficie del disco

El mecanismo de lectura se basa en que los cambios de campo magnético producen cambios de corriente eléctrica en la cabeza.

ORGANIZACIÓN DE DATOS

Los datos se organizan en Pistas o tracks, formando anillos concéntricos de información. Todas las pistas contienen la misma cantidad de información, por lo tanto la densidad de las pistas interiores es mayor que la de las extriores, esto simplifica la electrónica . Las pistas se separan por espacios llamados intertrack gap. Como los datos se transfieren por bloques a la CPU conviene que los datos también se almacenen por bloques en el disco

Cada pista se divide en zonas llamada sectores, teniendo cada pista entre 10 y 100 sectores. Estos pueden ser de longitud fija o variable. Los sectores se separan por espacios llamados inter record gap

La Identificación de un sector de una pista. Se debe consigue con un punto inicial de comienzo de la pista, y con marcas de principio y fin de sector, esto implica un formateado del disco con información extra

CARACTERÍSTICAS FÍSICAS DE LOS DISCOS

Cabezal puede ser móvil o fijo. En los cabezales fijos existe una cabeza de lectura escritura por cada pista, y los de cabeza móvil solo tiene un transductor que debe ser colocado sobre la pista a la que se desea acceder

Existen tres mecanismos de cabeza: la cabeza se encuentra a distancia fija de la superficie, la cabeza esta en contacto con la superficie (floppy) y de cabeza móvil

Existe una relación entre la densidad de los datos y la distancia entre la cabeza y la superficie. La cabeza debe generar o sentir el campo magnético con la suficiente intensidad como para poder leer o escribir. Cuanta más cantidad de información se desea almacenar en el disco más estrechas deben ser las pistas y para que las pistas sean estrechas los cabezales deben ser estrechos. Si los cabezales son muy estrechos deben estar muy próximos a la superficie para generar y sentir los campos y cuanto más cercana la cabeza a la superficie más posibilidad de fallos debido a las irregularidades de la superficie.

- **Disco winchester**

Es un disco precintado libre de contaminación en el que la cabeza esta muy cerca de la superficie con lo que se consigue alta densidad de datos. En este tipo de disco la cabeza báscula volando sobre la superficie debido al aire que desplaza el disco al rotar a gran velocidad

- **Tiempos de acceso al disco**

Para leer o escribir la cabeza debe encontrar primero la posición inicial de la operación. La selección de pista implica movimiento de cabeza y por lo tanto tiempo de búsqueda.Una vez alcanzada la pista se debe buscar el sector lo que supone una latencia de rotación. Se define el tiempo de acceso como el tiempo de búsqueda más latencia de rotación.

RAID REDUNDANT ARRAY OF INDEPENDENT DISCK

Aprovechan el concepto de paralelismo en la organización de discos. Consiste en un arrays de discos que operan independientemente y en paralelo. Al haber múltiples discos puede haber múltiples peticiones de entrada/salida. Una petición se puede realizar en paralelos si el bloque que se desea acceder se distribuye a lo largo de todos los discos. La redundancia de la información se usa para asegurar la fiabilidad del sistema.

Existen seis niveles del 0 al 5. No son niveles jerárquicos sino diferentes arquitecturas las Características comunes a todas estas arquitecturas son que el conjuntos de discos son vistos por el sistema operativo como un único disco lógico, que los datos se distribuyen a través del array de discos y que la capacidad redundante de los discos se utiliza para almacenar paridad de información. Las dos primeras características son las que determina los diferentes tipos de RAIDS.

El Objetivo delos RAIDS es acortar distancias entre la velocidad del procesador y la velocidad electrodinámica de los discos. Las estrategias de implementación usadas son la sustitución de un disco de gran tamaño por varios de tamaño menor , la distribución de datos de manera que se pueda acceder simultáneamente a varios discos mejorando el rendimiento de entrada salida,

12.2 MEMORIA OPTICA

ν CDROM

Estan fabricados con resina policarbonatos recubierta de una superficie muy reflectiva como el aluminio. La información se guarda mediante pequeños agujeros en la superficie. el disco maestro se crea con un láser de alta densidad , el resto se genera por estampado .Hay que proteger su superficie para evitar el deterioro

La lectura se hace mediante un láser de baja potencia que recoge la variación de intensidad del rayo láser mediante un fotosensor y se convierte a señal digital

Ls información se oraniza de dos formas diferentes:

CAV(Constant Angular Velocity), es una organización similar a los discos magnéticos. Exite la misma cantidad de información en todos los sectores y tiene una velocidad de giro constante. Es poco usada porque desaprovecha espacio

En CLV(Constant Linear Velocity) la densidad de información constante en todos los sectores, por lo tanto hay que variar la velocidad de giro para realizar la lectura ,gira más lentamente en los exteriores. En esta organización no existen varios track aislados sino uno solo en espiral y tiene una capacidad aproximada 774.57Mb es decir aproximadamente 550 disquetes de 3,5

Formato de bloque

- SYNC.- Identifica el comienzo del bloque
- Header.- Dirección del bloque y modos de la información
  - ⇒ Modo 0.- Sin datos
  - ⇒ Modo 1 con datos y código de error
  - ⇒ Modo 2 con datos sin código de error

Ventajas, contiene mas información que el disco magnético, es mas facil de replicar la información y es removible.

Entre las desventajas figuran el ser solo de lectura y tener unos tiempos de acceso mayores que el magnético

ν WORM

- Write once read memory
- Se prepara el disco para que se pueda escribir una vez con un láser de baja densidad
- Velocidad angular constante

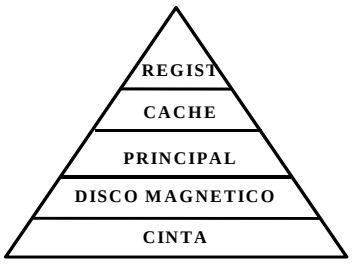
13 LA JERARQUIA DE MEMORIA

13.1 INTRODUCCIÓN

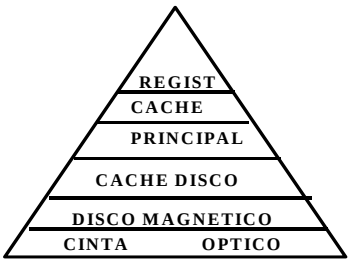
La elección de la memoria de un sistema viene determinada por las siguientes ligaduras: la Cantidad, la Rapidez y el Precio. Discutir sobre cantidad de memoria necesaria para un sistema es un sin sentido. Los usuarios de los sistemas siempre quieren más. Cuanto mayor es la memoria, mayores son las aplicaciones y cuanto mayores son las aplicaciones más necesidad de memoria. La rapidez de la memoria es muy importante. El principal objetivo es que la memoria y la CPU trabajen a velocidades parecidas para eliminar tiempos de espera. Al final el tema que determina el sistema de memoria es el económico. Las memorias grandes y rápidas son muy caras. El diseñador del sistema debe buscar un equilibrio entre las tres ligaduras:

- Mayor rapidez: más caras
- Mayor capacidad más baratas
- Mayor capacidad mayor tiempo de acceso

La solución es diseñar una jerarquía de memoria, es decir diferentes tipos de memoria organizados y relacionados de manera que optimen todas las ligaduras.



tradicional



actual

Al descender en la jerarquía decrementa el coste/bit, incrementa la capacidad, Incrementan los tiempos de acceso y decrementa la frecuencia de accesos. En la siguiente tabla aparecen los valores de tiempos de acceso y costes para diferentes tipos de memoria:

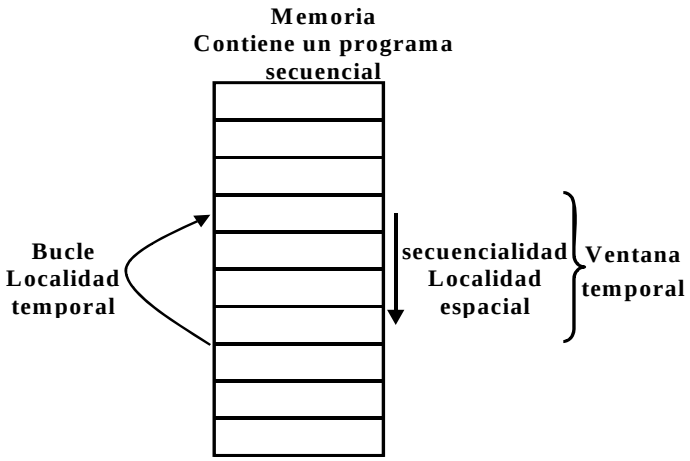
.

TECNOLOGÍA	TIEMPOS DE ACCESO	DOLARES POR MBYTE(1993)
SRAM	8-35 NS	100-400
DRAM	90-120 NS	25-50
MAGNETICOS	10.000.000-20.000.000	1-2

El punto clave de la jerarquía es el decremento de las frecuencias de acceso a los niveles inferiores y la base teórica que lo apuntala es el principio de localidad de referencia a la memoria en la ejecución de los programas.

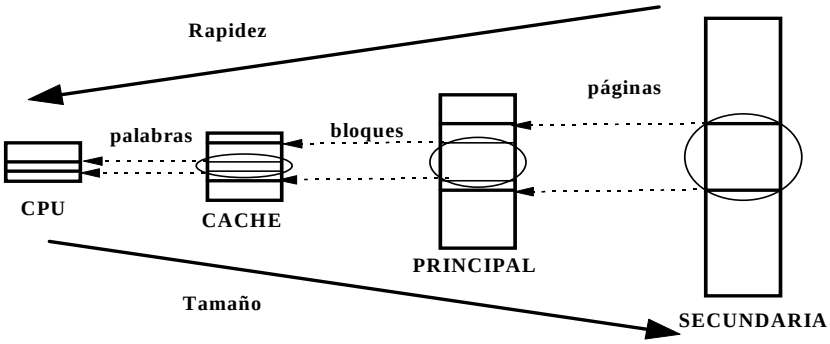
13.2 PRINCIPIO DE LOCALIDAD

Si se examina cuidadosamente el comportamiento dinámico de los programas, en especial la secuencia de referencias a memoria, se observa que esta secuencia de referencias a memoria no es aleatoria. No solo eso sino que además puede llegar a ser predecible. Este examen llevó a la postulación del principio de localidad: *Durante un intervalo de tiempo un programa tiende a agrupar sus referencias a memoria en una pequeña porción del espacio de direcciones disponible. (Denning 1970)*



La localidad se descompone en localidad temporal y localidad espacial. En la temporal existe una alta probabilidad de referenciar en el futuro páginas referenciadas en el pasado. Se suele dar en ciclos, subrutinas, pilas, variables para contar y totalizar. Se ve facilitada por el hecho de tener muchos bloques de memoria en el nivel superior de la memoria. En la localidad espacial existe una probabilidad elevada de referenciar un objeto próximo a otro ya referenciado. Se da en arrays de datos o en la ejecución secuencial de código. La localidad espacial se facilita si se dispone de bloques de memoria grande.

Visto lo anterior es posible organizar los datos en la jerarquía, de manera que el porcentaje de accesos a cada nivel inferior, sea substancialmente menor al del nivel superior. Una jerarquía de memoria se organiza de manera que el espacio de direcciones del nivel i es un subconjunto del espacio i+1. Esto trae consigo un problema en la consistencia de la información, ya que generalmente el nivel i+1 se modifica con la información del i.



El rendimiento de la jerarquía de memoria se mide con la función de la tasa de aciertos que se define como la probabilidad de encontrar la información solicitada en un determinado nivel de la memoria. La tasa de aciertos dependerá, entre otras cosas, del tamaño de la memoria. Se produce un fallo si el dato no se encuentra en el nivel superior. En función de lo anterior se puede definir la tasa de fallos como el tanto por ciento de accesos no encontrados en el nivel superior. Es igual a (1 - tasa de aciertos)

Como el rendimiento es la razón principal de una jerarquía de memoria, la velocidad de aciertos y de fallos es importante. Se llama tiempo de aciertos al tiempo que se tarda en acceder al nivel superior de la jerarquía de memoria. Incluye el tiempo necesario para comprobar si el acceso es un acierto o un fallo. Se llama penalización de fallos el tiempo que se necesita para sustituir un bloque de nivel superior por uno de nivel inferior mas el tiempo para entregar este bloque al procesador. Esta penalización depende del tamaño del bloque que se quiere intercambiar con el nivel superior y del tiempo de acceso a la memoria del nivel inferior. Ya veremos en su momento que esta es una de las principales características diferenciadoras entre la memoria cache y la memoria virtual. El tiempo de acceso a la memoria secundaria que utiliza la memoria virtual como nivel inferior es del orden de 20 millones de nano segundos, mientras los tiempos de acceso a la memoria principal que utiliza la cache como nivel inferior de jerarquía es del orden de 100ns.

13.2.1 TIPOS DE MEMORIA EN LA JERARQUÍA

- **REGISTROS:**
  - Rápidas
  - Pequeñas
  - Caras.
  - Volátil
  - Semiconductor
- **CACHE:**
  - Tecnología cmos estáticas
  - Entre los registros y la Memoria Principal.
  - No accesible por el usuario
  - Volátil
  - Tamaño pequeño
- **MEMORIA PRINCIPAL:**
  - Semiconductor, dinámica
  - Alta densidad
  - Tecnología CMOS
  - Volátil
  - Gran tamaño, velocidad media
- **LA CACHE DE DISCO:**
  - No es una memoria físicamente separada de la principal es una porción de la MP usada como almacenamiento temporal de datos que deben escribirse en el disco
  - Mejoras: en lugar de realizar muchas transferencias pequeñas, solo realiza una grande, mejora del rendimiento del disco, reduce la intervención del procesador en i/o. En ocasiones algunos datos se pueden referenciar antes de hacer la escritura a disco, lo que evita hacer una nueva lectura del disco.
- **MEMORIA SECUNDARIA:**
  - Almacenamiento permanente de datos y programas
  - Dispositivos externos
  - Información en forma de ficheros y registros
  - Discos, cintas y ópticos
  - Disco como extensión de M.P., da lugar a la virtual
  - La de mayor tamaño

13.3 PARAMETROS QUE MIDEN EL RENDIMIENTO

Tiempo de acceso. Para memorias aleatorias, el tiempo que tarda en ejecutarse una operación de lectura/escritura, desde el instante en que la dirección está presente en la memoria, hasta el instante en que el dato se ha escrito o leído

Tiempo de ciclo. Es el tiempo de acceso, más el tiempo adicional que se necesita para realizar el siguiente acceso. La existencia de este tiempo adicional se debe a la Regeneración del dato en las lecturas destructivas y a la precarga.

Razón de transferencia. La razón a la que los datos pueden transferirse a, o , desde la memoria. En memorias aleatorias viene dado por la expresión $1/T_{CICLO.}$ Para memorias de acceso no aleatorias viene dada por $T_N = T_A + N/R$ donde:

$T_N \rightarrow$ promedio en leer n-bits

$T_A \rightarrow$ tiempo de acceso promedio

N $\rightarrow$ número de bits

R $\rightarrow$ razón de transferencia en bits por segundo.

El cálculo del rendimiento de un nivel de la jerarquía de memoria es complejo porque depende de muchos factores, pero se puede utilizar una aproximación que sirve para calcular a grosso modo los tiempos de acceso promedio de la jerarquía:

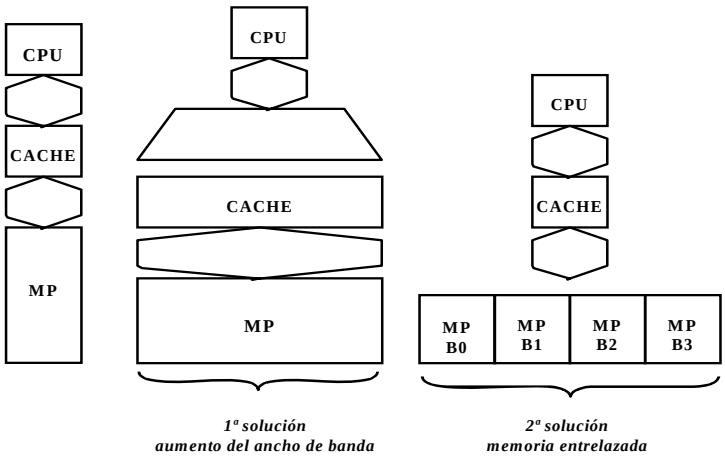
TF=Ti+Tasa de fallos·Ti+1

Ya veremos en los temas de memoria cache y virtual que la expresión que calcula los tiempos de acceso de la jerarquía son más complejos

LA MEMORIA ENTRELAZADA

En el tiempo de penalización de un fallo influyen dos factores diferentes: la latencia y el tiempo tardado en acceder a todo el bloque. La latencia se debe principalmente la tecnología y mejora lentamente con ésta. Por tanto si se quiere mejorar la penalización se debe intentar mejorar el tiempo de acceso al bloque. Este tiempo de acceso se puede reducir si se consigue aumentar el ancho de banda de la memoria principal

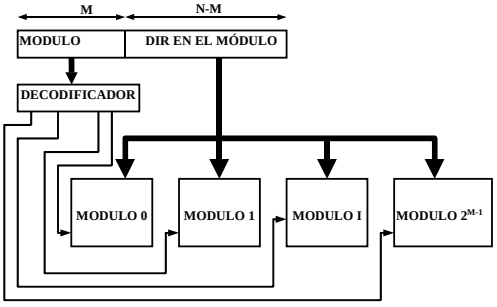
Una solución es incrementar el tamaño de la memoria y del bus con lo que aumenta proporcionalmente la anchura de banda de memoria. El coste principal de esta mejora está en la mayor anchura del bus. El coste secundario está en los buffers adicionales de la memoria. Otra posible solución es organizar los bancos de memoria para leer o escribir múltiples palabras en un solo acceso, en lugar de necesitar un acceso para cada palabra. Al enviar la misma dirección a varios bancos les permite leer a todos simultáneamente. A esto se le llama memoria entrelazada



ESTRUCTURA DE UNA MEMORIA ENTRELAZADA

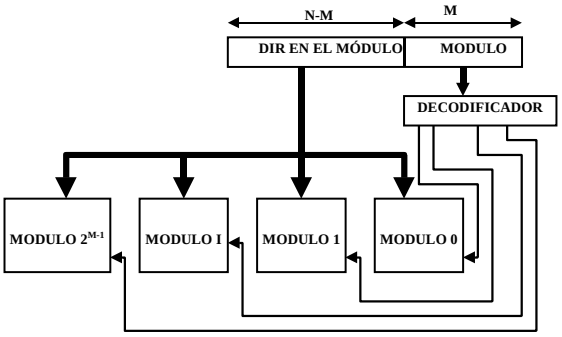
Una memoria principal de N palabras se divide en M módulos de N/M palabras cada uno de ellos. Estos módulos puede trabajar en paralelo con el resto. Existen dos organizaciones el entrelazamiento de orden alto y el entrelazamiento de orden bajo.

En el entrelazamiento de orden alto la memoria principal de 2^N palabras se divide en 2^M módulos cada uno de ellos con 2^{N-M} palabras. La forma de direccionarla y organizarla se puede ver en la siguiente figura:



Cada módulo tiene 2^{N-M} palabras consecutivas. Con los bits más significativos de la dirección se selecciona el módulo. Por lo tanto direcciones consecutivas de memoria se almacenan en el mismo módulo. Con este entrelazamiento no se consigue paralelismo de acceso que es lo que se busca.

En el entrelazamiento de orden bajo, con los bits mas significativos de la dirección se seleccionan las palabras y con los bits menos significativos se seleccionan los módulos, de manera que direcciones de memoria consecutivas corresponden a palabras almacenadas en la misma dirección de distintos módulos.

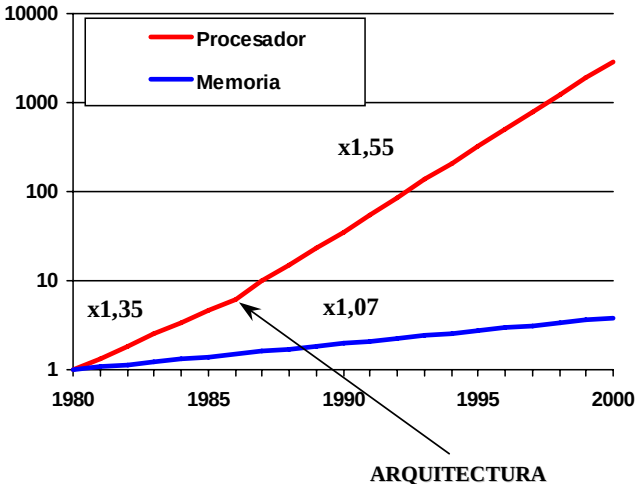


14 LA MEMORIA CACHE

14.1 **INTRODUCCION**

Uno de los principales problemas de los sistemas computadores es la imposibilidad de la memoria de intercambiar información con el procesador al ritmo que éste puede procesarla. Si se observa como han evolucionado los rendimientos de ambos sistemas se puede comprobar que el crecimiento del rendimiento anual de las memorias ha sido constante e igual al 7%, mientras que el de los procesadores fue del 35% anual hasta el año 86 fecha en la que aparecieron las primeras arquitecturas RISC y en la que el aumento del rendimiento paso al 55% anual.

Este salto se reduce con las jerarquías de memoria, destacando el nivel de la memoria cache que es una memoria rápida y pequeña situada entre el procesador y la memoria principal que almacena la información actualmente en uso de la memoria. La evolución del uso de caches ha sido muy importante. El primer computador que la usó fue el IBM 360/85 (1969). En 1980 casi ningún microprocesador utilizaba caches, mientras que en 1996 a menudo usaban dos niveles de memoria cache.

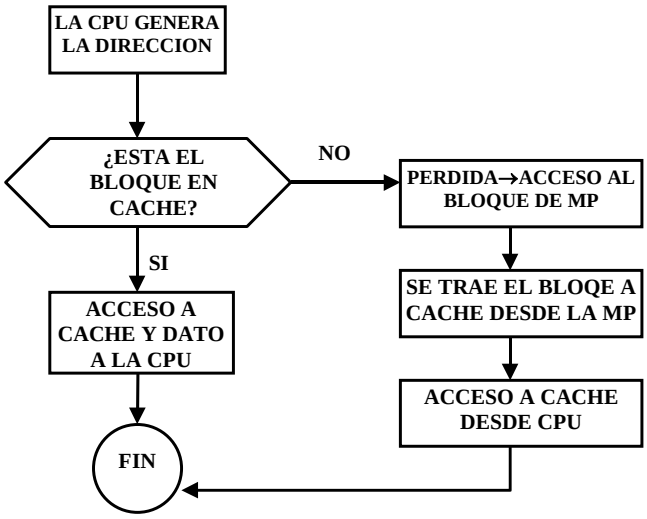


A continuación se estudia el modo de operación. Vamos a suponer una jerarquía de memoria compuesta por dos niveles, el superior la memoria cache y el inferior la memoria principal. Para estudiar el modo de operación de la memoria cache, debemos saber que la memoria principal y la cache se dividen en conjuntos de palabras de igual tamaño. A los conjuntos de la memoria principal los llamamos bloque y a los conjuntos de la cache los llamamos marcos de bloque (en ocasiones también se les llama líneas o slots).

Sabemos que en una jerarquía de memoria los datos que contiene el nivel superior de la misma (la cache) son un subconjunto de los datos de la memoria de nivel inferior. Además como la memoria cache es mucho más pequeña que la principal debe existir un flujo de información entre ambos niveles. El tamaño de la información que se intercambia es de un bloque de manera que los bloques de la memoria principal vienen a ocupar los marcos de bloque de la cache.

La forma de operar es la siguiente. La Unidad central de proceso genera una dirección de memoria. Con esta dirección se accede primero a la memoria cache, para comprobar si esta el dato solicitado. En el caso de que esto sea así, se produce un acierto de cache, y se envía el dato a la UCP, que a continuación genera la siguiente dirección.

En caso que no se encuentre en la memoria cache el dato buscado, se accede a la memoria principal y se trae a memoria cache todo el bloque que contiene el dato solicitado. A esto se le llama fallo de cache. A continuación se envía el dato de la cache a la UCP. En la siguiente figura se puede ver el flujo de esta operación. SE define la tasa de fallos como el tanto por ciento de accesos a memoria cache que son fallos.



14.2 RENDIMIENTO DE LA MEMORIA CACHE

El tiempo de CPU se divide en ciclos de ejecución de programa y ciclos que la CPU espera a que se produzca un acceso a la memoria llamados ciclos de parada. Generalmente se supone que los T_{ACCESO} a cache forman parte de los ciclos de ejecución de programa: T_{CPU}= (Ciclos $_{EJECUCIÓN}$ + $Ciclos_{PARADA}$)·T_{CICLO}

Los Ciclos de parada de memoria se deben a fallos de cache. En estos casos se deben realizar accesos al siguiente nivel de la jerarquía de memoria y por lo tanto la CPU debe esperar. Los ciclos de parada se dividen en ciclos de lectura y ciclos de escritura. Para calcular los ciclos de parada de lectura (fallos de lecturas) se aplica la siguiente expresión $Ciclos_{PARADA\text{-}LECTURA}$ = N^o $_{LECTURAS}$ · $Tasa_{FALLOS\text{-}LECTURAS}$ ·Penalización.

Los Ciclos de parada de escritura los estudiamos con detenimiento en el epígrafe de políticas de escritura. Ahora, para simplificar, suponemos que las penalizaciones de escritura y lectura son iguales y que las tasas de fallos de lectura y escritura se suman en la tasa de fallos de accesos. Con estas condiciones se obtiene la siguiente expresión Ciclos $_{PARADA}$= Accesos a memoria ·$Tasa_{FALLOS}$ ·Penalización.

Cuando aumenta la frecuencia de la unidad central de proceso, las penalizaciones por fallos de cache se incrementan .Esto ocurre porque la CPU debe estar mayor número de ciclos esperando a que la memoria le envíe información. Si la mejora del procesador se produce en la disminución del número de ciclos por instrucción(CPI) también se incrementa la penalización por fallos. Esto se debe a que cuanto menor sea el número de ciclos por instrucción de la máquina, mayor será el impacto de los ciclos de parada en el rendimiento. Supongamos que la penalización son 100 ciclos y que CPI son 10, entonces durante la penalización se podrían ejecutar 10 instrucciones, pero si el CPI son 5 se podrían ejecutar 20 es decir disminuye el rendimiento. Es decir las mejoras de la UCP si no van acompañadas de mejoras en la cache producen aumentos de penalizaciones en los fallos de cache. Otra expresión que se suele utilizar para analizar el rendimiento de la cache es la del tiempo de ciclos promedio de una cache :

$$T=Tc_{ACHE} + Tasa_{FALLOS} \cdot Penalización$$

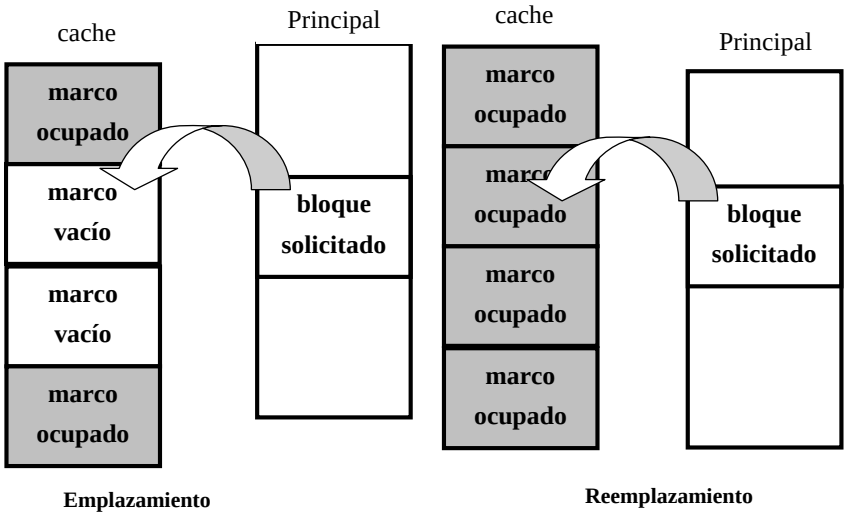
Sabiendo que la penalización depende del tiempo de acceso a la memoria principal y del tamaño de los bloques.

14.3 ELEMENTOS BÁSICOS DE DISEÑO

Como la memoria cache es mucho más pequeña que la memoria principal, un marco de bloque de cache no puede estar ocupado permanentemente por el mismo bloque. Esto da lugar a las políticas de

- Emplazamiento
- Reemplazamiento
- Actualización

El emplazamiento consiste en determinar dónde se coloca un bloque de información traído de la memoria principal cuando hay espacio en la memoria cache. El reemplazamiento selecciona el bloque que se sustituye cuando la cache está llena. En cuanto a las políticas de actualización determinan como se actualizaran los datos en la jerarquía de memoria cuando la Unidad central de proceso quiere realizar una operación de escritura en memoria. En los siguientes apartados estudiaremos cada una de estas políticas, indicando como afectan sus parámetros de diseño al coste rendimiento del sistema.



14.4 POLÍTICAS DE EMPLAZAMIENTO

Como la memoria principal es mucho mayor que la memoria cache hay que decidir en que marco se ubica un bloque. Existen tres políticas de emplazamiento la directa, la asociativa, la asociativa por conjuntos. La elección de una política de emplazamiento condiciona la organización de la memoria cache.

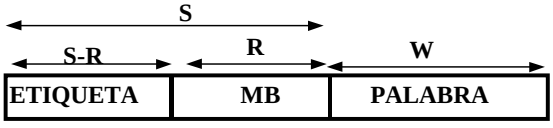
14.4.1 POLÍTICA DE EMPLAZAMIENTO DIRECTA

Consiste en que cada bloque de Memoria Principal ocupa siempre el mismo marco de la cache. Este marco viene dado por la expresión MB= B mod M, donde :

- MB es el nº de marco de la cache
- B el número de bloque de memoria principal
- M el numero de marcos de la memoria cache (tamaño de la cache)

Implementación

La dirección física que proporciona el procesador se interpreta de la siguiente manera



Donde el campo *palabra* (offset) selecciona una palabra de las varias que tiene el bloque, el campo MB (índex) indica el marco de la cache que ocupa el bloque y el campo etiqueta(TAG) se utiliza para la identificación del bloque. El número de marcos de la cache es $M=2^R$, el número de bloques de la memoria principal 2^S y el número de palabras por bloque es 2^w .

Modo de identificación del bloque

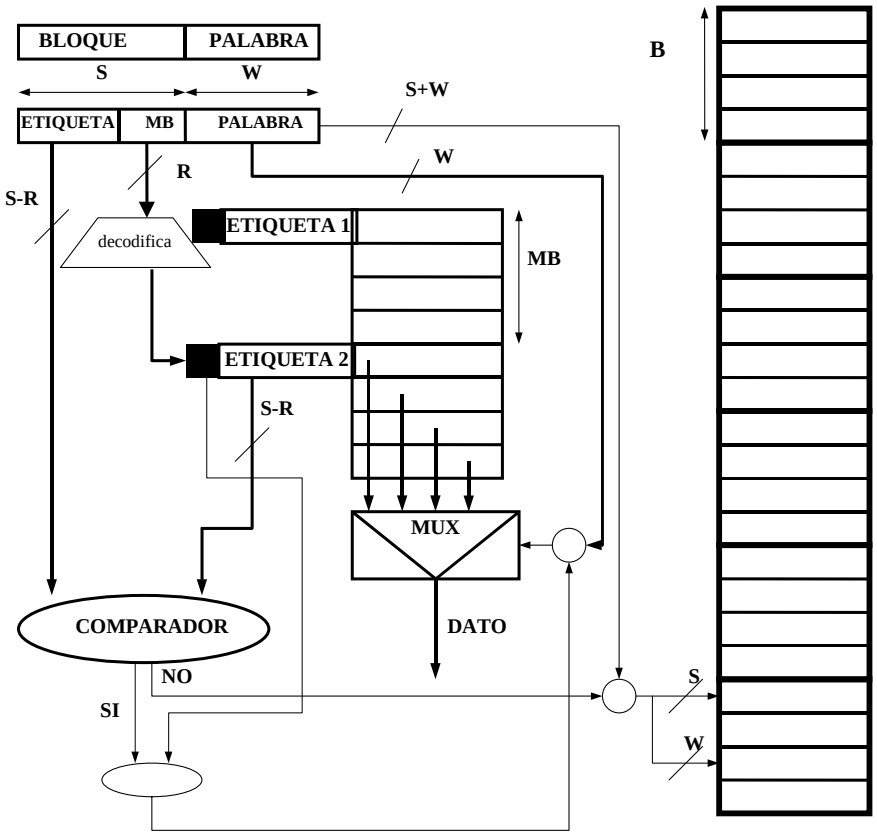
Para poder identificar el bloque que ocupa un marco, se asocia a los marcos de cache unos bits de memoria llamados etiqueta del marco, en los que se almacenará el campo etiqueta de la dirección cada vez que se guarde un bloque en un marco. Para ver cual es el bloque almacenado en un marco, se compara la etiqueta de la dirección con la etiqueta del marco.

Modo de operación:

- Se accede de manera aleatoria al marco de bloque de la cache con los MB bits de la dirección física

- Se comprueba que el marco contiene el bloque deseado comparando el campo ETIQUETA (o TAG) de la dirección física con la ETIQUETA (o TAG) de la cache mediante un mecanismo similar al de una memoria asociativa.
- Si coinciden las etiquetas se ha producido un *acierto de cache* y se realiza el acceso al dato, utilizando el desplazamiento w que fija la palabra del marco.
- Si no coinciden, esto indica que el bloque buscado no se encuentra en el marco consultado y se produce *un fallo de cache,* es decir un acceso a la memoria principal para traer a la cache el bloque solicitado.

A los marcos se les asocia un bit de validez que sirve para indicar si la información contenida en el bloque es válida o no. Por ejemplo, cuando un procesador arranca, la cache estará vacía y por lo tanto los campos de etiqueta no tendrán sentido. El esquema del HW que implementa esta política se puede ver a continuación.



Ventajas (En comparación con las otras poíticas de emplazamiento)

- Baja complejidad hardware. Necesita un decodificador y un comparador muy sencillo, esto tiene como consecuencia que el hardware sea más barato.
- Alta velocidad de operación (el tiempo de cache es bajo)
- No necesita algoritmos de reemplazamiento

Desventajas

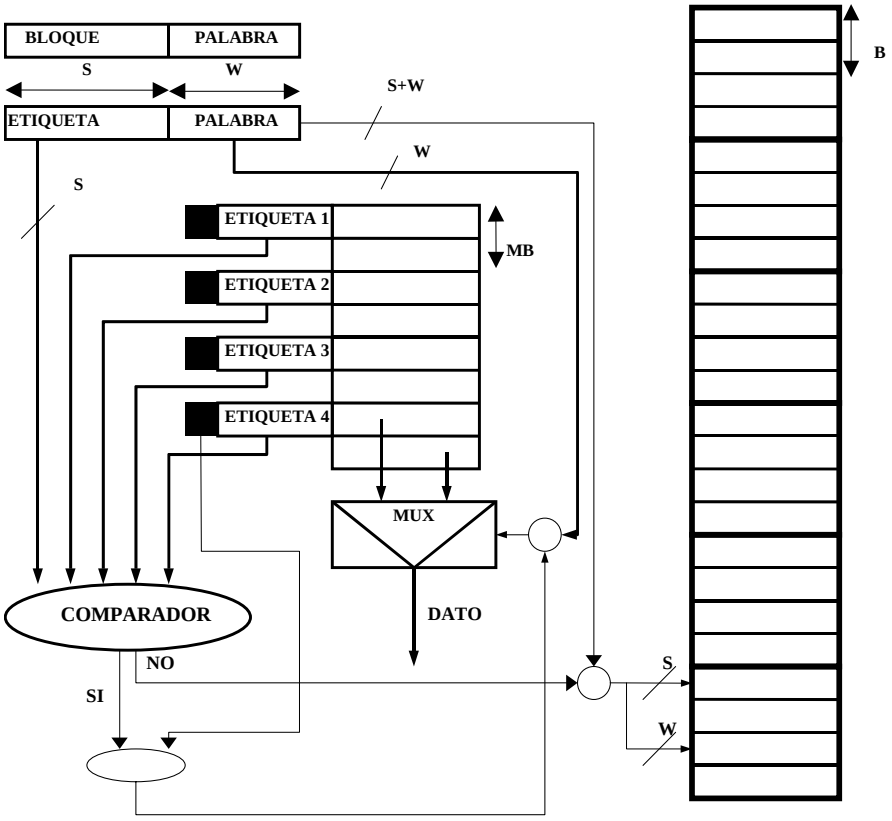
- Si un programa hace referencias repetidas a dos objetos que se encuentran en dos bloques diferentes a los que les corresponde el mismo marco, estos bloques tendrán que estar permanentemente saliendo de la memoria cache, por lo tanto la tasa de fallos será elevada.

14.4.2 POLÍTICA DE EMPLAZAMIENTO ASOCIATIVA

En esta política un bloque de la memoria principal puede situarse en cualquier marco de bloque de la cache. Ésta interpreta la dirección física proporcionada por el procesador de la siguiente manera:

ETIQUETA	PALABRA

La etiqueta de la cache contiene todos los bits que identifican al bloque de la memoria cache; de esta manera se elimina el paso de identificación aleatoria del marco que tenía la ubicación directa, luego los pasos son primero una identificación del marco mediante la comparación y después una lectura.



Para determinar si un bloque está en la cache, su lógica debe comparar simultáneamente las etiquetas de todos los marcos de cache con el campo etiqueta de la dirección.

Ventajas:

- Más flexible que la ubicación directa, lo que tiene como principal consecuencia una tasa de fallos baja porque los bloques no necesitan competir por el mismo marco.

Desventajas

- Complejidad del circuito HW. El circuito comparador tiene que tener tantas entradas como marcos de bloque tiene la cache.
- La rapidez de identificación del bloque disminuye porque los circuitos comparadores de gran tamaño son muy lentos.
- Se produce una sobrecarga del hardware de la memoria puesto que las etiquetas necesarias son de gran tamaño.
- Más cara debido a la complejidad de los circuitos.
- El permitir la libre elección del bloque que se elimina para introducir otro obliga a tener unas estrategias de reemplazamiento.

14.4.3 POLÍTICA DE EMPLAZAMIENTO ASOCIATIVA POR CONJUNTOS

Es un compromiso entre el emplazamiento directo y el asociativo. En ella la memoria cache se divide en *conjuntos de marcos* de manera que cada bloque de memoria principal se puede ubicar en un solo conjunto de marcos de la memoria cache pero en cualquiera de los marcos de bloques del conjunto. Una memoria asociativa por conjuntos de grado de asociatividad E= N es aquella en la que cada conjunto contiene N marcos de bloque.

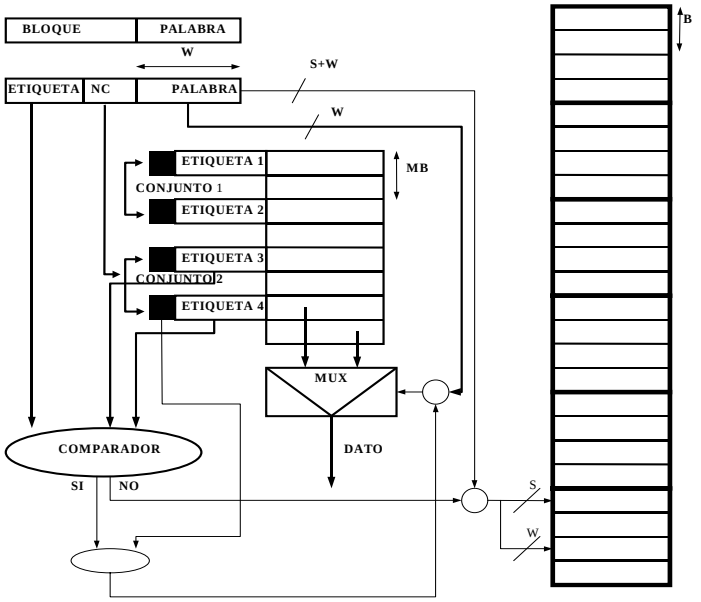
Si se supone la cache dividida en C conjuntos de marco, el conjunto en el que se coloca el bloque M de memoria principal es NC=M mod C. La dirección de memoria física se interpreta de la siguiente manera:

etiqueta	conjunto	palabra

Las características de esta política de emplazamiento son las siguientes:

- Complejidad hardware media. Es más complejo que la ubicación directa pero menos que la asociativa. El número de comparadores viene dado por el número de marcos de bloque de cada con junto.
- Rapidez en la identificación media (entre la directa y la asociativa).
- Sobrecarga de memoria media debido a la etiqueta.
- Tasa de fallos media.

El emplazamiento asociativo por conjuntos se encuentra situado entre el emplazamiento directo y el totalmente asociativo de manera que si se selecciona tamaños de conjuntos de un solo bloque estamos en emplazamiento directo y si solo hay un conjunto estamos en el caso de emplazamiento totalmente asociativo. A continuación aparece la estructura que implementa el emplazamiento asociativo por conjuntos.



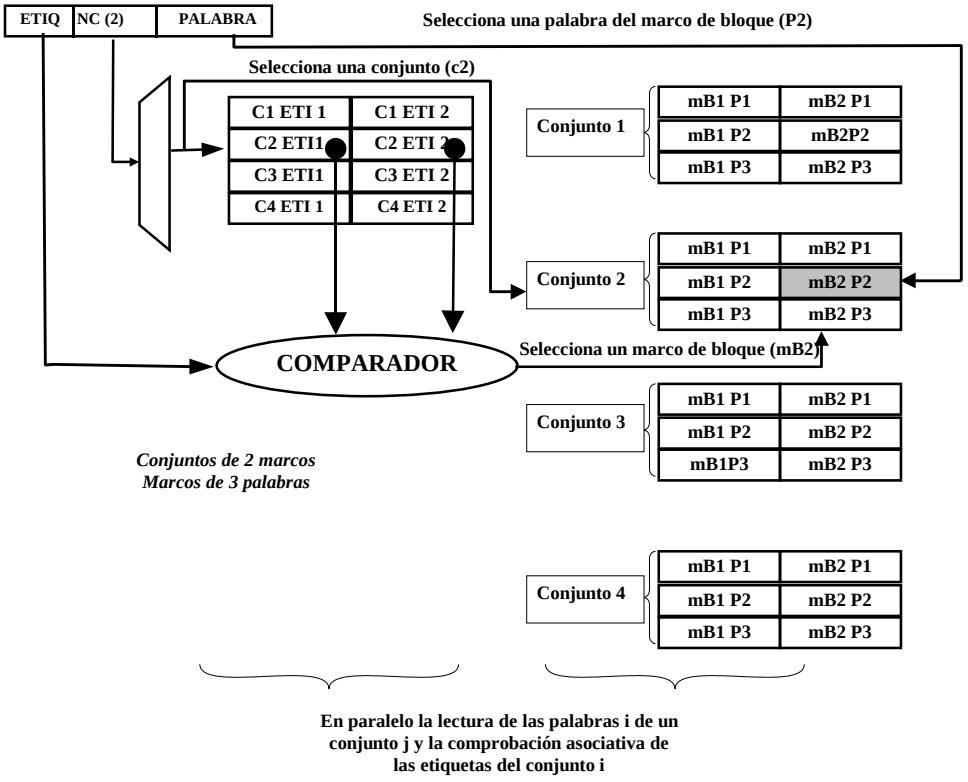
PARALELISMO DE ACCESO

Con la política de emplazamiento asociativa por conjuntos se pueden realizar un acceso paralelo a la palabra que se desea leer mientras se comprueba que el bloque cargado en el marco es el correcto. Para ello se utilizan dos memorias diferentes. Una de ellas contiene las etiquetas de cada conjunto y la otra contiene los marcos de bloque de cada conjunto.

La forma de trabajar es la siguiente: por un lado se accede a la memoria de etiquetas para comprobar que el bloque que buscamos se encuentra en ese conjunto. Esta comprobación se hace de manera asociativa con todas las etiquetas del conjunto. Por otro lado se conoce el conjunto j que al que se quiere acceder y la palabra i que se quiere leer, lo que no se conoce es el marco de bloque al que se quiere acceder. Lo que se hace es leer las palabras i de todos los marcos del conjunto y esperar a que el comparador nos indique cual de ellas es la válida. En resumen

- se comprueban asociativamente todas las etiquetas del conjunto j

- en paralelo se accede a la palabra i de todos los bloques del conjunto j
- cuando la etiqueta ha sido identificada se lee la palabra i del bloque seleccionado.
  En la escritura este acceso paralelo no se puede llevar a cabo.



14.4.4 PARÁMETROS DE DISEÑO Y RENDIMIENTO

[Página 390 de Hennessy]

Ya se vio que un fallo de cache se produce cuando se referencia un bloque que no se encuentra en ningún marco de bloque de la cache, y por lo tanto hay que ir a la memoria principal a buscarlo. Existen diferentes tipos de fallos.

Fallos iniciales (compulsor). Se producen en los primeros accesos a la memoria cache, por ejemplo cuando empieza a ejecutarse un proceso por primera vez. En estos casos, el bloque no está en la cache todavía. También se llama de arranque en frío o de primera referencia.

Fallos de capacidad (capacity). Son debidos a que la cache no puede conservar todos los bloques que se usan en un programa. Esto produce una falta de localidad temporal. En estos casos los bloques tienen que descargarse y posteriormente se tienen que volver a cargar en la memoria cache.

Fallos de conflicto (conflict). Son debidos a la competencia por los marcos del conjunto en emplazamientos asociativo por conjuntos o emplazamiento directo.

Los parámetros de diseño de una memoria cache son:

- El grado de asociatividad
- El tamaño de la memoria cache
- El tamaño de los marcos de bloque

Estos parámetros de diseño influyen en

- El rendimiento
- Coste HW, que depende de la tecnología

ν **EL RENDIMIENTO**

El tiempo de acceso a una jerarquía de memoria compuesta de cache como nivel superior y memoria principal como nivel inferior viene dado por:

$$T=T_{ACCESO}+Tasa_{FALLOS}*P$$

Donde:

- **T** es el tiempo promedio de acceso a la jerarquía de memoria
- $\mathbf{T_{ACCESO}}$ es el tiempo de acceso de la cache, que depende de la tecnología
- $\mathbf{Tasa_{FALLOS}}$**.-** es el tanto por ciento de fallos que se producen en los accesos a cache.
- **P** (penalización) es el tiempo de acceso al nivel inferior de la jerarquía y se desglosa de la siguiente manera: tiempo que se tarda en encontrar la primera palabra del bloque más el tiempo que se tarda en traer todo el bloque.

ν INFLUENCIA DEL GRADO DE ASOCIATIVIDAD SOBRE LOS PARÁMETROS DE RENDIMIENTO:

Recordemos que el grado de asociatividad es E=nºmarcos/nº conjuntos y que coincide con lo que llamamos el número de vías de la memoria cache.

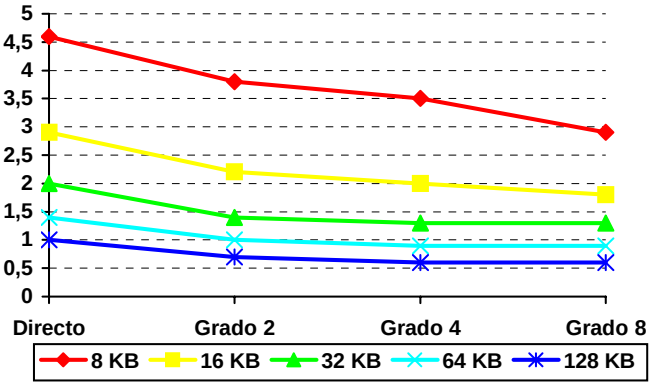
A mayor grado de asociatividad:

- Menor tasa de fallos porque es menor la competencia por los marcos del bloque del conjunto
- Mayor tiempo de acceso porque hay mayor complejidad en los comparadores
- Mayor coste de hardware porque crece el número de comparadores

De los estudios experimentales se pueden sacar dos conclusiones:

- Un conjunto de 8 vías es, a efectos prácticos, tan efectivo en la reducción de fallos como una memoria totalmente asociativa.
- Una cache de emplazamiento directo de tamaño N tiene la misma tasa de fallos que una asociativa por conjuntos de 2 vías y tamaño N/2.

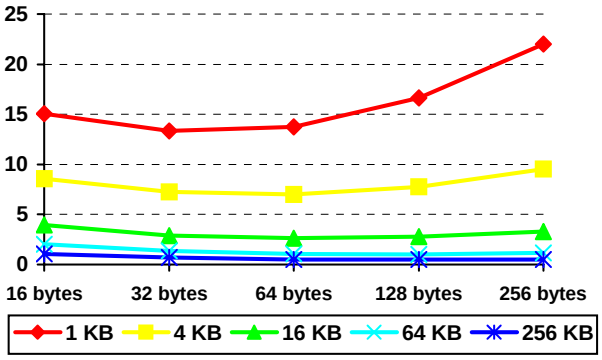
En la figura se puede ver como evoluciona la tasa de fallos en función del grado de asociatividad y el tamaño de la memoria cache.



ν INFLUENCIA DEL TAMAÑO DE BLOQUE SOBRE LOS PARÁMETROS DEL RENDIMIENTO

En el siguiente estudio se supone un tamaño de memoria cache constante. Hay que tener cuidado porque el tamaño del bloque afecta tanto a la tasa de fallos, como a la penalización.

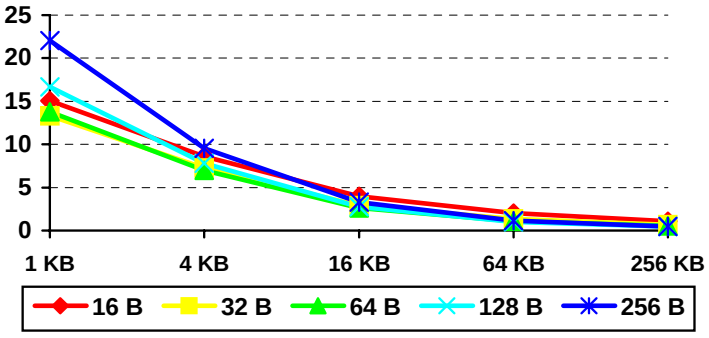
Inicialmente al crecer el tamaño del bloque disminuye la tasa de fallos debido a que se captura mejor la localidad espacial y se reducen los fallos de arranque en frío. Al seguir aumentando vuelve a aumentar *la tasa de fallos*. Si crece el tamaño del bloque disminuye el número de marcos de bloque de la cache, por lo tanto al haber menos marcos se captura mal la localidad temporal. Además aumentan los fallos de conflicto si la cache es pequeña. Las razones por las que ocurre esto son las siguientes: el número de bloques es menor y se producirá una gran competencia por esos bloques por lo tanto un bloque saldrá de la cache antes de que sean accedidas muchas de sus palabras. En la figura podemos ver como evoluciona la tasa de fallos en función del grado de asociatividad y el tamaño de la memoria.



Por otro lado, al crecer el tamaño del bloque aumenta la penalización de los fallos. La penalización del fallo está determinada por el tiempo necesario para buscar el bloque en el nivel inferior de jerarquía (la latencia) y el tiempo de transferencia del bloque. Cuanto mayor es el bloque mayor es su tiempo de transferencia. Luego en este caso la selección del tamaño del bloque depende de la latencia y del ancho de banda con el nivel inferior de memoria. Para latencias grandes y anchos de banda grandes se recomiendan bloques grandes porque en un fallo la cache obtiene más bytes por un pequeño aumento de penalización, mientras que para bajas latencias y pequeños anchos de banda son mejores los bloques pequeños porque se disminuye la penalización y se disminuyen los fallos de conflicto. Por ejemplo, con las características de pequeñas latencias y pequeños anchos de banda la penalización de un bloque grande es dos veces la penalización de un bloque pequeño.

ν **INFLUENCIA DE TAMAÑO DE LA MEMORIA CACHE SOBRE LOS PARÁMETROS DE RENDIMIENTO**

La tasa de fallos disminuye con el tamaño de la memoria**,** más concretamente se produce una disminución de la tasa de fallos de conflicto porque aumenta el número de conjuntos C, y de la tasa de fallos de capacidad porque aumenta el número de marcos por conjunto.[Pg 458 del Peterson]. El siguiente figura se puede ver la evolución de la tasa de fallos para caches de tamaño creciente y bloque también de tamaño creciente.



ν **CONCLUSIONES GENERALES:**

- El grado de asociatividad influye sobre la tasa de fallos y el coste HW
- El Tamaño de bloque influye sobre tasa de fallos y la penalización
- El Tamaño de la memoria cache influye sobre tasa de fallos

Todos estos parámetros están sometidos a ligaduras que se deben respetar, tales como:

- Los tiempos de acceso a la cache deben ser menores que los tiempos de ciclo de la memoria principal.
- La integración en un mismo circuito del microprocesador y de la memoria cache limita el tamaño de la memoria cache pero reduce los tiempos de acceso.

Las tendencias actuales en la organización de la memoria caché son grados de asociatividad pequeños (E= 2 o 4) e incluso emplazamientos directos, y tamaños de bloque también pequeños, de 2 o

cuatro palabras. En la siguiente tabla parace un conjunto de microprocesadores, indicando el año de fabricación , el tamaño, el grado de asociatividad y número de conjuntos.

	Fecha	Tam.(KB)	Tam. del bloque	E	C
UltraSparc (167 Mhz)	9/95	16	4 (8 bytes)	1	512
HP-PA 7100 (99 Mhz)	2/92	256 (ext.)	8 (4 bytes)	1	8192
HP-PA 7200 (120 Mhz)	9/95	2+256 (ext.)	8 (4 bytes)	1	8192
HP-PA 8000(180 Mhz)	5/96	1024 (ext.)			
MIPS R8000 (75 Mhz)	6/94	16	4 (8 bytes)	1	512
MIPS R10000 (200 Mhz)	3/96	32	8 (8 bytes)	2	256
DEC 21064 (150 Mhz)	6/93	8	4 (8 bytes)	1	256
DEC 21164 (333 Mhz)	-/95	8	4 (8 bytes)	1	256
DEC 21264 (500 Mhz)	12/97	64		2	
PowerPC 601 (80 Mhz)	4/93	32	8 (4 bytes)	8	128
PowerPC 620 (130 Mhz)	-/96	32	8 (8 bytes)	8	64
Pentium (75 Mhz)	3/93	8	8 (4 bytes)	2	128
Pentium Pro (180 Mhz)	1/96	8	8 (4 bytes)	2	128

14.5 POLÍTICAS DE REEMPLAZAMIENTO

Cuando se intenta acceder a una palabra y ésta no se encuentra en la cache es necesario trasladar todo el bloque de información que se encuentra en la memoria principal a la memoria cache. Si existe espacio en la cache, la política que se utiliza es la de emplazamiento. En el caso que no exista espacio en la cache se debe seleccionar el bloque que se quiere eliminar para introducir el nuevo. A estas políticas se las llama de *reemplazamiento***, y** deben cumplir los siguientes requisitos:

- Deben implementarse totalmente en hardware. La implementación en software es muy lenta y la razón de ser de la cache es aproximar la velocidad de trabajo de la memoria a la del procesador.
- Se debe intentar que la selección se realice en el ciclo de memoria principal, durante el cual se está trayendo el nuevo bloque
- Se debe intentar que el bloque reemplazado no tenga que utilizarse en el futuro. Este último requisito es el más difícil de cumplir puesto que **no** se conoce el comportamiento futuro de los programas.

En cuanto al espacio posible de reemplazamiento depende de la política de emplazamiento utilizada. En el caso del emplazamiento directo es trivial pues cada bloque tiene asignado su marco. Para el emplazamiento Asociativo: el espacio de reemplazamiento es toda la cache, y para el emplazamiento asociativo por conjuntos el espacio son los marcos del conjunto

Es importante darse cuenta de que todas las estrategias tratan de capturar la localidad temporal y espacial de referencia a memoria de los procesos, es decir intentar averiguar, basándose en la historia de los accesos a memoria, cuales son los bloques que van a ser referenciados a continuación para evitar sacarlos de la memoria cache. Algoritmos típicos son el de reemplazamiento aleatorio, y el de reemplazamiento del bloque menos recientemente usado (LRU)

ν **ALEATORIO**

En esta política el bloque a reemplazar se escoge aleatoriamente, esto hace facil su construcción en hardware y da lugar a tiempos de acceso bajos. Su principal inconveniente es que tiene una tasa de fallos elevada con relación al LRU.

ν **MENOS RECIENTEMENTE USADO (LRU)**

Se reemplaza el bloque menos recientemente usado. Para conjuntos asociativos de dos bloques es fácil de implementar

- Cada bloque incluye un bit de uso
- Cuando un bloque se referencia su bit se pone a uno y el del otro bloque a cero
- Se reemplaza el bloque que tiene su bit a cero

Cuando el número de bloques es mayor en lugar de un bit se utiliza un contador llamado registro de edad. El algoritmo a implementar es el siguiente:

- Si se referencia el bloque j del conjunto, el contador de j se pone a cero mientras que el contador del resto de los bloques que tenían un valor inferior al de j se incrementa
- Los contadores con valor superior no se incrementan
- Se reemplaza el bloque que tenga el valor mayor, y su contador se pone a cero
- Se incrementa el resto de los contadores en uno.

[Hwang en pg129 propone más técnicas de implementación]

ν **CONCLUSIONES: LRU VS ALEATORIO**

Para caches grandes la diferencia entre ambas estrategias disminuye enormemente hasta hacerse prácticamente idénticas. En una cache asociativa por conjunto de dos vías la tasa de fallos del algoritmo aleatorio es 1,1 veces mayor que el LRU.

El LRU obtiene mejores rendimientos con mayores grados de asociatividad pero es difícil de implementar. Para un grado de asociatividad mayor que 4 es excesivamente costoso en tiempo y en almacenamiento, e incluso puede llegar a suceder que el tiempo de actualización de los contadores sea mayor que el tiempo de acceso a la cache.

El Aleatorio tiene mayor tasa de fallos, menor coste hw y menores de tiempos acceso
El LRU tiene menor tasa de fallos y mayor tiempos de acceso y mayor coste hw

Por último indicar que se ha demostrado que en general el algoritmo de reemplazamiento tiene una influencia secundaria sobre el rendimiento del sistema, sobre todo si lo comparamos con las políticas de emplazamiento. La cache totalmente asociativa es la más sensible a los algoritmos de reemplazamiento. En el siguiente cuadro aparece un estudio de la tasa de fallos de los algoritmos LRU y aleatorios para distintos tamaños de cache y distintos grados de asociatividad.

	Grado 2		***Grado 4***		***Grado 8***	
	LRU	***Aleatorio***	***LRU***	***Aleatorio***	***LRU***	***Aleatorio***
16 KB	5,18%	5,69%	4,67%	5,29%	4,39%	4,96%
64 KB	1,88%	2,01%	1,54%	1,66%	1,39%	1,53%
256 KB	1,15%	1,17%	1,13%	1,13%	1,12%	1,12%

14.6 POLÍTICAS DE ACTUALIZACIÓN

(Hwang) Antes de entrar en consideraciones se debe saber que las operaciones de lectura predominan sobre las de escritura. Todos los accesos a instrucciones son lecturas y la mayoría de las instrucciones no escriben en memoria. Siguiendo el consejo de hacer siempre más rápido el caso común - corolario de la ley de Amdahl - se deben optimizar las caches para mejorar las lecturas. Pero también esta ley nos indica que el tiempo de mejora es proporcional al tiempo de ejecución no afectado por la mejora. Con esto se quiere indicar que no se deben despreciar nunca las mejoras que se puedan obtener en los accesos de escritura. La escritura tiene dos características que la diferencian de la lectura:

- No se puede acceder al dato en paralelo a la comparación del etiqueta, ya que se podría sobreescribir un dato incorrecto

- En la escritura la CPU especifica el tamaño de la escritura (de 1 a 8 bytes), y por lo tanto, sólo esa porción del bloque puede cambiarse. Es decir si el bloque se compone de cuatro palabras, se escribe esa única palabra, y no todo el bloque como ocurre en los otros casos.

El estudio de las políticas de actualización se ha dividido en dos partes. En la primer se estudia como se actúa cuando se produce un acierto de escritura y como afecta esto a la coherencia de la información en los diferentes niveles de la jerarquía y a los fallos de lectura. En este sentido se estudian las políticas de escritura directa y de post-escritura. Por otro lado se estudia como actuar en los casos de fallo de escritura. En esta parte se estudia escritura con asignación de marco y la escritura sin asignación de marco

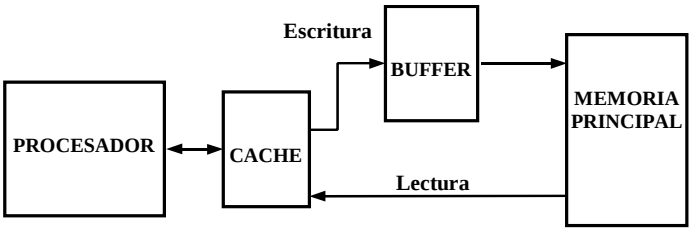
ESCRITURA DIRECTA

Cuando se produce un acierto de escritura (es decir cuando el dato que queremos modificar está en la cache), se actualiza el dato tanto en la memoria principal como en la memoria cache. En los fallos de lectura, independientemente de que el bloque de cache a sustituir haya sido modificado o no siempre se sobreescribe, porque existe coherencia.

La principal ventaja de esta política es que no existe inconsistencia nunca, esto tiene como efecto que los fallos de lectura sean más rápidas porque no necesitan escribir en el nivel inferior. Es más sencilla de diseñar y de manejar. Su principal desventaja es que no produce buenos rendimientos de escritura puesto que tiene que estar realizando constantemente accesos a la memoria principal.

Una solución a este problema consiste en añadir un buffer de escritura que almacena el dato mientras que éste espera ser escrito en la memoria principal. Una vez escrito el dato en la cache y en el buffer, el procesador puede seguir la ejecución desentendiéndose de la relación entre el buffer y la memoria principal.

El buffer de escritura tiene un número fijo de palabras que suele oscilar entre 1 y 10. Si el buffer está lleno, cuando se realiza una escritura desde el procesador éste debe detenerse hasta que haya una posición vacía. Lógicamente si la frecuencia a que la memoria principal completa escrituras es inferior a la que las genera el procesador el buffer no sirve de nada.



POST -ESCRITURA

En los aciertos de escritura actualiza sólo el dato que se encuentra en la cache de manera que el bloque modificado sólo se escribe en el nivel inferior cuando es sustituido, es decir cuando se produce un fallo.

En los fallos de lectura la forma de actuar depende de si el bloque se ha modificado o no. Cuando el bloque de cache a reemplazar no se ha modificado, ese bloque se puede sobreescribir sin problemas. Si el bloque se ha modificado antes de reemplazarlo hay que escribirlo en la memoria principal. Para distinguir si se ha modificado o no utiliza un bit que se llama dirty.

Las ventajas de esta política son:

- Las palabras individuales las escribe el procesador a la velocidad de la cache
- Múltiples escrituras en un bloque requieren sólo una escritura en el nivel más bajo de jerarquía, luego reduce el trafico entre Memoria principal y cache.
- Se hace mejor uso del ancho de banda de comunicación con el nivel inferior de jerarquía

Desventaja: compleja de implementar y produce inconsistencia

Para tratar los fallos de escritura existen dos políticas: con asignación de marco y sin asignación de marco.

CON ASIGNACIÓN DE MARCO.- Cuando se produce el fallo se trae el bloque de la memoria principal a la cache y a continuación se actúa como si hubiera habido un acierto. Esta técnica tiene una penalización elevada porque hay que gastar un tiempo en traerse el bloque para poder seguir trabajando.

SIN ASIGNACIÓN DE MARCO.- El bloque se modifica directamente en la memoria principal sin modificarlo en la cache, y por lo tanto cuando se produce un fallo de escritura no se debe traer el bloque previamente a la cache sino que basta con escribir el dato en la principal. Además se invalida en bloque correspondiente en la memoria cache mediante el bit de validez. Esta política es más rápida.

Ambas estrategias se podrían aplicar a la escritura directa y a la postescritura, pero en la práctica no es así. La post escritura suele utilizar la asignación de marcos esperando que posteriores escrituras puedan ser capturadas por la cache. La escritura directa suele utilizar la política de no asignación de marco, puesto que en cualquier caso las siguientes escrituras también se llevaran inmediatamente a la memoria principal.

Escritura directa VS post-escritura

Tanto la escritura directa como la post escritura tienen sus ventajas. Por ejemplo, la postescritura, escribe al ritmo de la cache, sólo se accede al nivel inferior de memoria de tarde en tarde y diversas escrituras en un mismo bloque se reflejan con una sola escritura en un nivel inferior. Como no todas las escrituras van al nivel inferior, necesita menor ancho de banda, haciéndola atractiva para los multiprocesadores.

Con la escritura directa las pérdidas de lectura nunca acaban en escrituras del nivel inferior, es más fácil de implementar y el nivel inferior siempre tiene sus datos actualizados. Esto es importante tanto para temas de entrada/salida, como para temas de multiprocesadores. En el modelo 800 del sistema DEC 3000 AXP el primer nivel utiliza escritura directa y el segundo nivel utiliza post-escritura.

14.6.1 PARÁMETROS DE DISEÑO Y RENDIMIENTO DE LA ESCRITURA

[Hwang]

Para describir el efecto de las políticas de actualización en los tiempos medios de acceso a memoria vamos a suponer que:

- $Aciertos_{ESCRITURA}$ es la fracción de escrituras conseguidas del sistema.
- Tm el tiempo de ciclo de memoria de segundo nivel
- Tb el tiempo de transferencia de bloque
- T_{ACCESO} el tiempo promedio de acceso a cache
- $T_{ACCESO}<Tm$

ESCRITURA DIRECTA CON ASIGNACIÓN DE MARCO

Recordemos que en estos casos cuando se produce un fallo de escritura primero se gasta un tiempo (considerado una penalización) en traer el bloque a la cache, y después se debe escribir el dato modificado tanto la cache como la memoria principal. El tiempo medio para completar una referencia cuando no hay buffer de almacenamiento intermedio viene dado por la siguiente expresión:

$$T=(1-Aciertos_{ESCRITURA})\cdot T_{ACCESO}+ Aciertos_{ESCRITURA} \cdot Tm+Tasa_{FALLOS}\cdot P$$

Siendo (1-E) las fracciones de lectura acertadas en las que solo se accede a cache y E la fracción de Escrituras en las que se accede a cache y a principal. Como en principio sólo habria que acceder a una palabra del bloque es correcto poner Tm. Operando sobre esta expresión queda

$$T=Tc+ Aciertos_{ESCRITURA} \cdot(Tm-Tc) +Tasa_{FALLOS}\cdot P$$

$Tasa_{FALLOS}\cdot P$ hace referencia al tiempo que tarda en traerse un bloque a la memoria cache en el caso en que se produce un fallo de escritura o de lectura**.**

ESCRITURA DIRECTA SIN ASIGNACIÓN DE MARCO

El rendimiento viene dado por la expresión:

$$Tc+ Aciertos_{ESCRITURA} \cdot(Tm-T_{ACCESO})+Tasa_{FALLOS}\cdot(1- Aciertos_{ESCRITURA})\cdot P$$

Tasa $Aciertos_{ESCRITURA}$ ·(1- $Aciertos_{ESCRITURA}$) indica la tasa de fallos de lectura que son los únicos casos en los que hay penalización ya que son los únicos casos en que se traen bloques a la cache. En los dos casos de escritura directa se puede observar que tienen una cota mínima cuando siempre se producen aciertos. Esta cota viene dada por la expresión T_{ACCESO}+ $Aciertos_{ESCRITURA}$ (Tm-T_{ACCESO}).

La escritura directa se mejora añadiendo un buffer intermedio. Para que el aprovechamiento de este buffer sea máximo, las transferencias de bloques debido a fallos de lectura deben tener prioridad sobre las escrituras. Es decir cuando se produce un fallo de lectura se detiene el envío de datos desde el buffer a la memoria principal para dar prioridad a la recuperación de información desde la memoria principal a la cache. En el caso extremo de que el buffer tuviera tamaño infinito los tiempos sería: Tc+Tfallos*P

POST ESCRITURA CON ASIGNACIÓN DE MARCO DE BLOQUE

Hay que recordar que en la postescritura se escribe en memoria principal sólo cuando se produce un fallo y que en la asignación de marco cada vez que se produce un fallo de lectura se trae un bloque de memoria. Luego:

$$T= T_{ACCESO}+Tasa_{FALLOS\text{-}LECTURA}\cdot P+Tasa_{FALLOS\text{-}ESCRITURA}\cdot P$$

Un de los términos se debe a la escritura y el otro a la lectura. Suponiendo la tasa de fallos de lectura y de escritura iguales el tiempo sería $T_{ACCESO}+2\cdot Tasa_{FALLOS}\cdot P$. Esta política de escritura se podía mejorar utilizando un bit dirty

$$T_{ACCESO}+ Tasa_{FALLOS}\cdot P+Tasa_{FALLOS}\cdot P\cdot Eb$$

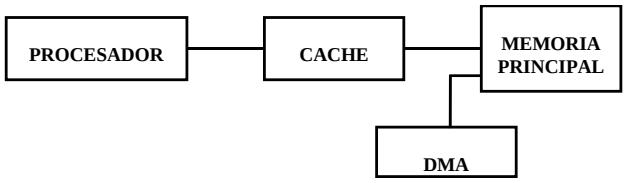
Donde Eb es la probabilidad de que un bloque reemplazado haya sido modificado.

14.7 COHERENCIA CACHE

El que existan instantes de tiempo en los que el contenido de la memoria principal y el contenido de la memoria cache sean diferentes puede ocasionar problemas. Se van a ver dos casos, cuando el sistema tiene un procesador y una cache, y cuando el sistema tiene varios procesadores cada uno con su cache.

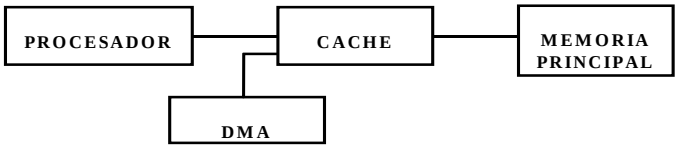
UN PROCESADOR Y UNA CACHE

Vamos a estudiar como pueden aparecer inconsistencias en este sistema. Supongamos un controlador de DMA conectado directamente a la memoria principal:



Si el DMA escribe un dato directamente en la memoria principal, independientemente de la política de escritura aparece una inconsistencia entre la memoria principal y la cache. Además en el caso de la post escritura puede suceder que el DMA lea un dato directamente de la memoria principal, y este dato puede ser inconsistente. Este segundo caso no se daría para el caso de escritura directa.

Una solución podría ser conectar los procesadores de entrada salida directamente a la memoria cache



La principal desventaja de esta solución es que el inmenso tráfico de entrada salida entre el DMA y la cache hace disminuir su rendimiento. Además, las necesidades de sincronización de los dos puertos de la cache y del arbitraje producen importantes degradaciones del rendimiento.

Otra solución: La idea clave es asegurar que el procesador lea siempre el dato correcto y que cada dato escrito por el DMA esté disponible en el procesador. Si el DMA escribe un dato en la memoria principal, debe comprobar si está en la cache. En el caso que así sea la cache debe invalidarlo de manera que si el procesador quiere leerlo, lo deba recuperar de la memoria principal.

Cuando el procesador DMA quiere realizar una lectura, comprueba si el dato ha sido modificado en la cache, en caso que así sea se escribe el dato correcto en la memoria principal y se lee.

ν **VARIOS PROCESADORES CADA UNO CON SU CACHE**

En una estructura en la que hay que varios procesadores, cada uno de ellos con su cache, compartiendo bus y una memoria principal, aparece también el problema de la coherencia cache. Si se modifican datos en una cache, se invalida no solo la palabra correspondiente de la memoria principal, sino también la misma palabra en otras caches. Incluso si se utiliza una política de escritura directa, el resto de las caches pueden contener datos no validos.

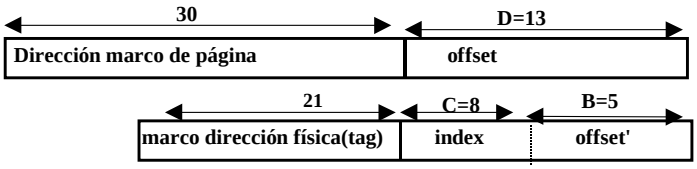
Soluciones:

- *Vigilancia del bus en escritura directa*. Es una política apropiada para sistemas con un solo bus. Cada controlador de cache comprueba las líneas de direcciones para detectar operaciones de escritura en memoria por parte de otros maestros del bus. Si otro maestro escribe en una posición de memoria compartida que también reside en la memoria cache, el controlador invalida la memoria cache.
- *Transparencia hardware.* Se utiliza circuitería especial para asegurarse de que todas las actualizaciones de memoria principal, vía cache, quedan reflejadas en todas las caches.
- *Memoria excluida de cache***.** Solo una porción de memoria principal puede ser accedida por más de un procesador, y esta se diseña como no transferible a cache. En un sistema de este tipo todos los accesos a memoria compartida son pérdida de cache, porque la memoria compartida no se copia nunca a cache.

14.8 EL ALPHA AXP 21064

El microprocesador Alpha 21064 tiene dos caches una de datos y otra de instrucciones. En este epígrafe describimos exclusivamente la cache de datos. La cache tiene 8 Kbytes (8192 bytes) en bloques de 4 palabras de 64 bits cada una (32 bytes). Utiliza el Emplazamiento directo y la escritura directa con un buffer de escritura de 4 bloques sin asignación de marco en fallos de escritura.

El microprocesador proporciona una dirección virtual de 43 bits que se traduce mediante el TLB de datos a una dirección física de 34 bits que se interpreta como se ve en la figura:



El 21064 tarda dos ciclos de reloj en realizar los siguientes pasos en una lectura:

- Llegada de la dirección
- Acceso aleatorio al marco
- Comparación de las etiquetas
- Envío de una señal a la CPU para que cargue el dato de la cache

Las escrituras son más complejas que las lecturas. Si hay un acierto de escritura los tres primeros pasos son comunes pero el último es diferente. El dato se escribe simultáneamente en el nivel inferior de la jerarquía y en la cache. Para ello se envía el dato a un buffer de escritura que puede almacenar cuatro bloques de cuatro palabras cada una de ellas. Si el buffer esta vacío aquí se acaba la escritura desde el punto de vista de la CPU. Si el buffer contiene otros bloque modificados se chequean sus direcciones para ver si coinciden con el nuevo que se quiere cargar en el buffer. En caso de que así sea se procede a realizar un *write merging* , es decir se recolocan las palabras del buffer que pertenecen al mismo bloque, de manera que se pueda optimizar el uso de ancho de banda del buffer con la memoria principal. De esta manera se optimiza el uso del buffer. Por último si el buffer está lleno la cache y la CPU deben esperar a que se vacíe.

	V	PALABRA	V	PALABRA	V	PALABRA	V	PALABRA
100	1	Ocu	0		0		0	
104	1	Ocu	0		0		0	

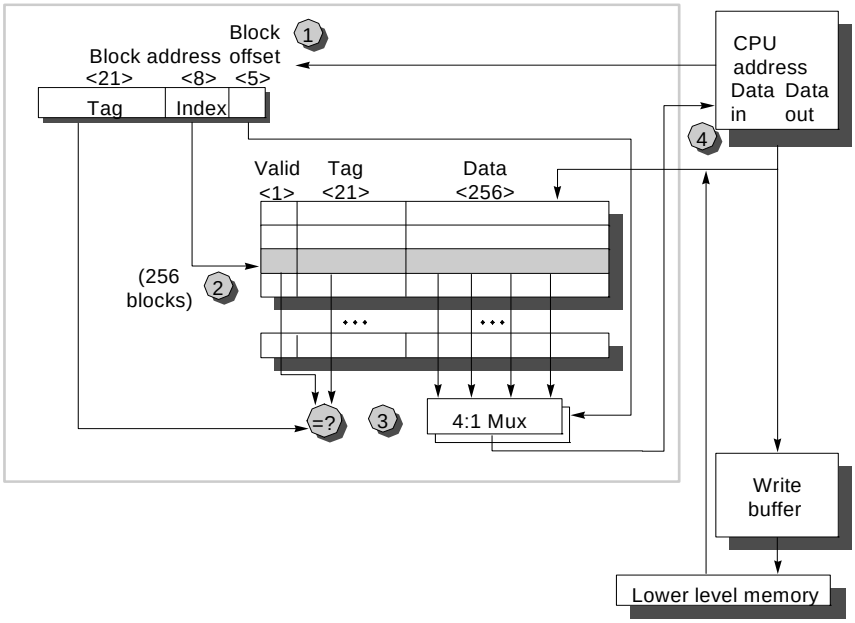
108	1	Ocu	0		0		0	
101	1	ocu	0		0		0	
	PALABRA		**PALABRA**		**PALABRA**		**PALABRA**	
100	1	ocu	1	ocu	1	ocu	1	ocu
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

En una pérdida de lectura la cache envía una señal de parada a la CPU y se lee un bloque de 4 palabras (32 bytes) del nivel inferior (cache de 2º nivel). El bus que conecta la cache con el nivel inferior es de 16 bytes de anchura en el sistema DEC 3000 model 800, y tarda 5 ciclos de reloj en realizar una transferencia, es decir una parada de 10 ciclos de reloj para trasladar todo el bloque a la cache.

Debido a que el emplazamiento es directo, no existen algoritmos de reemplazamiento. Por lo tanto la actualización del correspondiente marco de cache consiste en la actualización de los datos, la etiqueta de la memoria y el bit de validez. En los fallos de escritura se escribe directamente en el nivel inferior

Darse cuenta que el Alpha tiene cache de datos y de instrucciones separadas. Con esto se evitan cuellos de botella. Por ejemplo cuando se están realizando instrucciones de load y de store, al tiempo que se solicitan datos, también se solicita una instrucción. Con una única cache esto forzaría una parada. Hoy en día la técnica de caches separadas está muy extendida. La cache de instrucciones del Alpha 21064 tiene una estructura idéntica a la de datos que hemos explicado.(incluye un buffer de Prebúsqueda de instrucciones)

La CPU sabe si está direccionado una instrucción o un dato de manera que puede separar puertos para una y para otra, doblando el ancho de banda entre la jerarquía de memoria y la CPU .



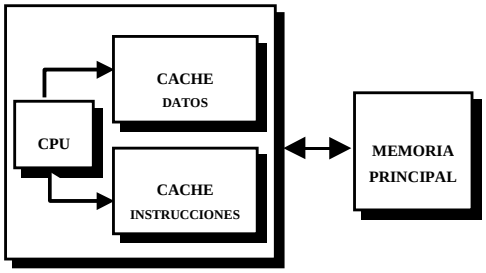
14.9 MEJORAS AL DISEÑO BÁSICO

14.9.1 LAS CACHES PARTIDAS [STALLING PG133]

Se llama cache unificada la que tiene una sola memoria que almacena datos e instrucciones. Se llama cache partida la que tiene una cache de datos y una cache de instrucciones.

Ventajas de la cache unificada

- Para un tamaño dado de memoria cache tiene mayor tasa de aciertos que la partida, porque nivela automáticamente la carga de captura entre instrucciones y datos. Es decir si una ejecución implica más instrucciones que datos se cargará de instrucciones, en caso contrario se llenará de datos.
- Sólo se necesita diseñar e implementar una cache



Ventaja de la cache partida

- Elimina la competición por la cache entre el procesador de instrucciones y la unidad de ejecución. Esto es importante en los sistemas segmentados. Normalmente el procesador captará instrucciones anticipadamente y llenará un buffer con las instrucciones que van a ejecutarse. Si el sistema tiene una cache unificada cuando la unidad de ejecución realiza un acceso a memoria para cargar y almacenar datos realiza una petición a la cache. Si al mismo tiempo la unidad de prebúsqueda de instrucciones emite una petición de lectura de una instrucción a la cache, dicha petición será temporalmente bloqueada para que la cache pueda servir primero a la unidad de ejecución. Esta disputa puede llegar a degradar las prestaciones.[pg 383 Stalling]
- Puesto que el procesador sabe si está trabajando con datos o con instrucciones, puede tener puertos separados para cada uno de ellos, y por lo tanto se dobla el ancho de banda entre la jerarquía de memoria y la CPU
- Tiene mejores tiempos de acceso efectivo
- Existe la posibilidad de optimización de los parámetros de diseño de cada cache por separado.

Desventaja: Mayor tasa de fallos. Fija el tamaño de la parte dedicada a datos e instrucciones

La tendencia actual son las caches partidas.

Ejemplo de esta arquitectura son las máquinas superescalares, como el Pentium o el PowerPC. Pg 383 del [Hennesy][pp133 Stalling]. Para poder compara resultados entre caches partidas y unificadas el tamaño total debe ser el mismo.

14.9.2 TÉCNICAS PARA DISMINUIR LA TASA DE FALLOS

A continuación se van a exponer una serie de técnicas que se pueden utilizar para reducir la tasa de fallos de una memoria cache. Algunas de ellas ya se han visto (Las señaladas con asterisco) y por lo tanto no se va a entrar en mayores consideraciones:

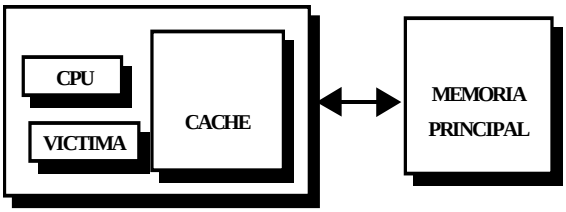
- Aumento del tamaño de bloque *
- Aumento del grado de asociatividad*
- Caches víctimas
- Caches pseudoasociativas
- Prebúsqueda hw

Las técnicas de modificar el tamaño de los bloques y el grado de asociatividad son clásicas pero tienen el inconveniente de aumentar el ciclo de reloj, en el caso de aumento del grado de asociatividad, o aumentar la penalización, en el caso de utilizar bloques más grandes. El resto de las técnicas que se tratan no modifican ninguno de estos dos parámetros de rendimiento.

CACHE VÍCTIMA

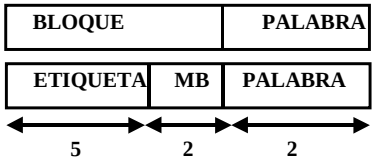
Se coloca una pequeña cache totalmente asociativa al lado de la cache de manera que siempre que se sustituye un bloque, se carga en el nivel más bajo de la jerarquía y en la cache víctima. Siempre que se produce un fallo se comprueba si el bloque deseado se encuentra en la cache víctima. Si es así, el bloque de

la cache y el de la cache víctima se intercambian. Especialmente útiles para pequeñas caches de emplazamiento directo. Para una cache de 4K con una memoria víctima de cuatro entradas se puede eliminar entre el 20% y el 95% de fallos.



CACHE PSEUDO ASOCIATIVA

Se comporta de maneras diferentes si hay acierto o si hay fallo. Si hay un acierto se comporta como una memoria de emplazamiento directo. Si hay un fallo se busca en otro marco de página. ¿En que marco? Lo más sencillo es cambiar el bit más significativo de los bits de marco de la dirección física. Por ejemplo, siendo la interpretación de la dirección física la que aparece a continuación



y suponiendo que la dirección inicial era 111111 11 11 primero se buscaría en el marco 11 y se comprobaría la etiqueta 11111. En caso de fallo se accedería al marco de bloque 01 y se comprobaría la etiqueta 11111. Estas caches tienen dos tiempos de acierto, uno, el normal, cuando se accede a la primera como un emplazamiento directo, y el otro, el de pseudoacierto, cuando el datos buscado se encuentra en el segundo acceso.

Se llama pseudoasociativa en el sentido que se forman conjuntos pseudoasociativos de 2 bloques de manera que no se consultan los dos a la vez, sino que primero se consulta uno y si falla se consulta el otro.

PRE BÚSQUEDA HARDWARE

Consiste en traer a la cache bloques que todavía no ha demandado el procesador. Esta técnica reduce la tasa de fallos. El dato o instrucción prebuscado se puede almacenar, en la propia cache o en un buffer que puede ser más rápidamente accedido. La técnica más habitual es la OBL (One Block Lookahead), en la que se carga el bloque solicitado y el siguiente. Suele ser eficiente siempre que el tamaño del bloque sea pequeño. Existen dos técnicas diferentes. En una de ellas se trae el bloque i+1 solo la primera vez que se referencia i. En la otra, llamada *Prebúsqueda sobre fallo* se trae el bloque i+1 siempre que se produce un fallo sobre el bloque i. Esta técnica se puede implementar de dos formas diferentes, o bien interna o bien externa a la cache. Por ejemplo el Alpha AXP 21064, carga el bloque referenciado en memoria cache y el otro en un buffer externo, si el bloque deseado se encuentra en este buffer se cancela el fallo de cache, se accede a este bloque y se produce una nueva prebúsqueda.

Se ha demostrado que para una cache de instrucciones de 4kb, con bloque de 16 bytes, con emplazamiento directo y un buffer de un bloque se puede reducir la tasa de fallos de un 15 a un 25%. Con un buffer de 4 bloque se puede reducir un 50% y con uno de 16 un 72%.

Una aproximación similar se puede realizar para una cache de datos. Un buffer sencillo de datos captura el 25% de fallos para una cache de emplazamiento directo de 4kb. Para 4 buffers se incrementa los aciertos en un 43%. Otros estudios indican que para 8 buffers compartidos de datos e instrucciones capturarían entre el 50 y 70% de todos los fallos para un procesador de dos caches asociativas por conjuntos de 4 caminos y 64Kb.

14.9.3 TÉCNICAS PARA DISMINUR LA PENALIZACIÓN DE PÉRDIDAS

- **ENVÍO DIRECTO DE LA PALABRA SOLICITADA AL PROCESADOR**

Hasta el momento hemos supuesto que cuando se produce un fallo de lectura hay que enviar todo el bloque de información desde el nivel inferior de la jerarquía a la memoria cache. Esto hace que la espera sea

elevada, sobretodo para bloques de gran tamaño. Este problema se puede evitar con el envío directo de la palabra solicitada al procesador. Existen en la actualidad dos técnicas que implementan esta política: la carga anticipada y primero la palabra solicitada.

Carga anticipada

Consistente en reanudar la ejecución del proceso tan pronto como llegue la palabra pedida. Por lo tanto no se espera la llegada completa del bloque. Ahorramos tiempo puesto que la ejecución del proceso y la transferencia del resto del bloque continúan en paralelo. Este mecanismo funciona mejor con instrucciones debido a que su carga es más secuencial.

Primero la palabra solicitada

Se organiza la memoria para que la palabra solicitada sea siempre la primera en llegar a la memoria cache. En cuanto esta palabra llega a la memoria cache, esta sigue funcionando. A continuación se transfiere el resto del bloque y se vuelve al principio del bloque.

- **DAR PREFERENCIA A LAS PÉRDIDAS DE LECTURA**

Con la escritura directa la mejora más importante que se puede realizar es la utilización de un buffer intermedio. Este tipo de mecanismo, sin embargo complican los accesos a memoria puesto que pueden almacenar el dato actualizado de una posición que se necesita en una pérdida lectura, es decir, puede ocurrir que el dato actualizado todavía no se haya escrito en MP y por lo tanto esté en el buffer. La solución más sencilla es detener la lectura hasta que se vacía el buffer. Pero esto aumenta mucho la penalización.

Solución: se comprueba si el dato está en el buffer y si está se recupera. Si no está se da prioridad a la lectura del dato de la memoria principal.

También se puede dar prioridad a la lectura para mejorar los tiempos de espera en la post escritura. Suponer que se tiene que reemplazar un bloque modificado (dirty bit) porque se ha producido un fallo de lectura. El proceso habitual es escribir todo el bloque en su posición de memoria principal y a continuación realizar la lectura. Para dar prioridad a la lectura habría:

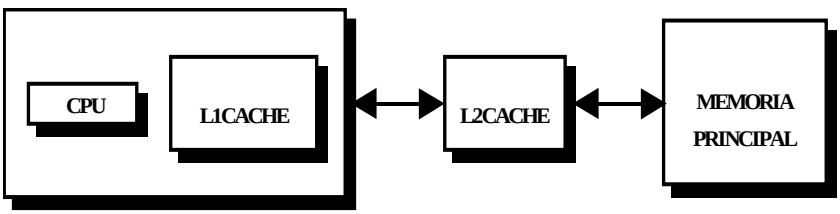
1. Depositar el bloque en un buffer intermedio
2. Se realiza la lectura
3. Se realiza la escritura.

- **CACHES NO BLOQUEANTES**

Se utilizan para reducir las paradas en los fallos de cache**.** Existen máquinas segmentadas en las que la CPU no para cuando se producen fallos de cache. Esto se puede conseguir haciendo que la CPU continue buscando instrucciones en la cache de instrucciones mientras espera que la cache de datos le devuelva un fallo de datos.

Las caches no bloqueantes extienden este esquema de manera que la memoria cache continua proporcionando datos mientras gestiona un fallo de lectura de datos. Esta optimización llamada "Acierto bajo fallos" reduce la penalización efectiva. Otra opción mas compleja es la llamada *acierto bajo múltiples fallos o fallos bajo fallos.*

CACHES DE DOS NIVELES



Esta técnica ignora la CPU y se concentra en la interfaz entre la cache y la memoria principal. Con el aumento de la densidad de integración es posible tener una memoria cache incluida en el chip del procesador. Esta cache interna reduce la actividad del bus externo del procesador y por lo tanto reduce los tiempos de ejecución e incrementa las prestaciones globales del sistema. Los accesos a las caches internas son mucho más rápidos y el bus permanece libre para realizar otras transferencias.

En la actualidad aparte de la cache interna los sistemas tienen también caches externas. A esta estructura se le conoce como caches de dos niveles. La cache de primer nivel afecta al ciclo de reloj de la CPU y la cache de segundo nivel afecta a la penalización de una pérdida de cache. Esto tiene su importancia porque decisiones de diseño que no son aceptables en la cache de primer nivel por afectar al ciclo de reloj, si que pueden serlo para la de segundo nivel.

Por ejemplo, el primer nivel de cache debe ser lo suficientemente pequeño para emparejar con el ciclo de reloj de la CPU, mientras que el segundo nivel de cache puede ser los suficientemente grande como para capturar la mayoría de los accesos que deberían ir a la memoria principal, y de esta manera disminuir la penalización. Las características de la cache externa son:

- Tamaño Grande (256KB-8MB) que disminuye la tasa de fallos de capacidad y conflicto.
- Un tamaño de bloque: grande (64-512 bytes) que disminuye los fallos de inicio.
- Un grado de asociatividad bajo (1-4), esto puede parecer contradictorio, pero como vimos con anterioridad Cuanto mayor es la memoria cache menos influye el grado de asociatividad en la disminución del tanto por ciento de fallos. Como las memorias de segundo nivel son muy grandes (como poco 10 veces mayores) las asociatividad no disminuye casi la tasa de fallos pero si que aumenta los tiempos de acceso y por lo tanto las penalizaciones de primer nivel.
- Política de Reemplazamiento aleatorio.

14.9.4 TÉCNICAS PARA DISMINUIR LOS TIEMPOS DE ACIERTO

Los tiempos de acceso a la memoria cache son importantes porque afectan directamente al ciclo de reloj.

Caches pequeñas y simples. Como ya se sabe, Cuanto más simple y pequeño el hardware es más rápido. Esto también ayuda a que se puedan incluir en el chip. En cuanto a la simplicidad, los emplazamientos directos (que son los más sencillos de implementar) además permiten solapar la comprobación de la etiqueta con la transmisión del dato.

Caches virtuales Se tratan en el tema de memoria virtual.

Segmentación de la escritura para hacer los aciertos de escritura más rápidos. La escritura habitualmente se toma más tiempo que la lectura porque no se puede realizar en paralelo la comprobación de la etiqueta y la escritura del dato porque se corre el peligro de acceder al dato erróneo.

Una técnica que puede evitar este problema es la segmentación de la escritura. Como primera medida los datos y las etiquetas se separan de manera que puedan ser accedidas independientemente. En una escritura la cache compara la etiqueta con la actual dirección de escritura, como viene siendo habitual. La diferencia aparece cuando se escribe el dato durante la comparación. En lugar de escribir sobre la cache se escribe sobre un buffer de escritura retrasada. De tal manera que para escribir sobre la cache se utiliza, la dirección y el dato de la escritura anterior, viniendo el dato del buffer de escritura retardada. Es decir, la escritura se segmenta en dos etapas. En la primera mientras se comprueba la etiqueta se guarda el dato y la dirección en el buffer de escritura retrasada. En la siguiente fase si la escritura ha sido un acierto se escribe en la cache.

15 LA MEMORIA VIRTUAL

15.1 INTRODUCCIÓN

A lo largo del tiempo, la cantidad de memoria principal presente en un sistema se ha incrementado notablemente, pero el tamaño de los programas ha crecido mucho más rápidamente que la memoria. Esto ha obligado a estudiar la manera de ampliar la memoria sin que esto sea excesivamente gravoso para los usuarios.

Por otro lado, en los sistemas de computadores se suele compartir la memoria principal entre varios usuarios, por lo tanto debe existir un modo de proteger y de compartir la memoria. En este tema se estudia la manera de optimizar el uso de la memoria del sistema computador para alcanzar las características explicadas. Para optimizar el uso de la memoria hay que fijar la organización y la administración de la memoria de manera que se consigan:

- Máximo número de procesos en la memoria.- Para evitar que la CPU esté inactiva
- Protección de la memoria.- Para evitar que al ejecutarse un proceso acceda a zonas de memoria que pertenecen a otro proceso
- Comparticion de la memoria.- Para permitir que procesos diferentes compartan información

La administración de la memoria virtual la realiza el sistema operativo. Definimos el sistema operativo cómo un programa almacenado en memoria que gestiona los recursos del ordenador proporcionando servicios a los programadores y supervisando la ejecución de otros programas. Los objetivos del sistema operativo son proporcionar comodidad al programador y obtener un sistema eficaz.

15.2 ORGANIZACIÓN DE LA MEMORIA VIRTUAL

INTRODUCCIÓN

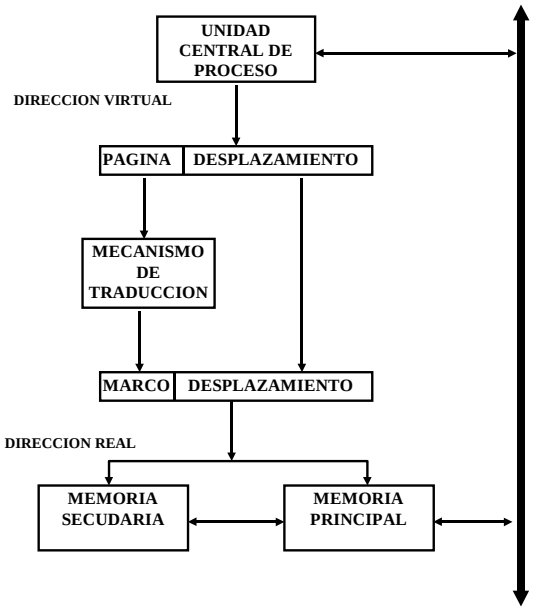
El objetivo de la memoria virtual es que el programador utilice una memoria mayor de la que realmente tiene el sistema. Para conseguirlo el procesador genera direcciones virtuales con un tamaño superior a las direcciones reales de acceso a la memoria física. Por lo tanto, las direcciones virtuales deben convertirse en direcciones reales mientras el programa está en ejecución. Esta función se realiza mediante mecanismos de traducción dinámica de direcciones (DAT) que debe ser tan rápido como sea posible para evitar la degradación del rendimiento del sistema. Esta traducción de las direcciones permite implementar la reubicación que facilita la carga de los programas.

Esto presenta el siguiente problema, si esta traducción se hiciera palabra a palabra, la información de mapa sería tan voluminosa que requeriría tanto o más almacenamiento real que los propios procesos. Por eso, el programa se trocea en bloques y la correspondencia con la memoria principal se realiza por bloques. Cuanto mayores sean los bloques menos información de mapeo se necesitará, pero menos procesos activos entraran en la memoria.

Como el espacio de direcciones virtuales es mayor que el espacio de direcciones real, parte del proceso debe estar almacenado en memoria principal y parte debe estar en la memoria secundaria. Luego para ejecutar un proceso íntegramente debe aparecer un flujo de bloques entre la memoria principal y la secundaria. Esto también lleva a pensar que la dirección virtual unas veces se traduce a una dirección de memoria principal (cuando el bloque está en memoria principal) y otras a una dirección de memoria secundaria.

El intercambio de bloques entre la memoria principal y la secundaria plantea problemas de gestión que se solucionan mediante las estrategias de ubicación, reemplazamiento y búsqueda.

Por otro lado, aunque los conceptos de funcionamiento de la cache y de la memoria virtual son similares, sus raíces históricas son diferentes lo que ha motivado que se utilicen terminologías diferentes, que conviene conocer. En memoria virtual un bloque se llama página, y una pérdida (miss) se llama fallo de página.



La mayoría de las opciones de diseño en los sistemas de memoria virtual están motivadas por el alto coste de los fallos. Un fallo de página necesita cientos de miles de ciclos para subsanarse. Esta enorme penalización de fallos dominada por el tiempo necesario para conseguir la primera palabra (acceso a memoria secundaria entre 10 y 20 millones de nanosegundos) conduce a varias decisiones de diseño claves de sistemas de memoria virtual: las páginas deben ser lo suficientemente grandes como para amortizar el elevado tiempo de acceso a memoria secundaria. Tamaños de página típicos son de 4kb a 16kb. En la actualidad se está considerando tamaños de hasta 64kb. Las organizaciones de la memoria que reducen la frecuencia de fallos son las más atractivas. La mejor técnica es la colocación flexible de las páginas, como la totalmente asociativa. Los fallos de memoria virtual pueden ser tratados por software porque el gasto de ciclos de CPU que se realiza es muy pequeño comparado con el tiempo de acceso al disco. Además el software permite la utilización de algoritmos inteligentes para elegir la forma de colocar las páginas, porque incluso pequeñas reducciones en la tasa de fallos compensan la utilización de tales algoritmos. La utilización de la escritura directa para gestionar las escrituras en la memoria virtual no funcionará, ya que llevan mucho tiempo. En su lugar necesitaremos un esquema que reduzca el número de escrituras en el disco.

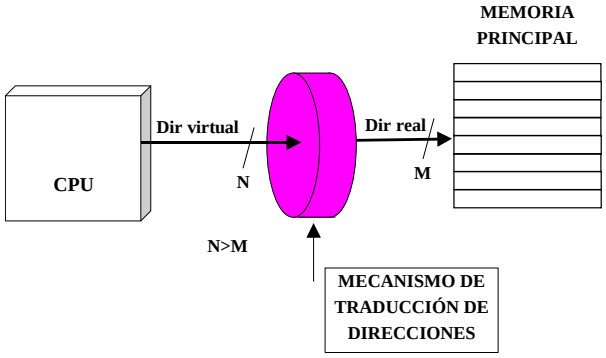
En definitiva se puede definir la memoria virtual como un sistema de almacenamiento jerárquico de dos niveles administrado por el sistema operativo que da al usuario la sensación de tener un espacio de direcciones superior al que realmente tiene. Su objetivo es tener una velocidad de acceso similar a la de la memoria principal con un coste por bit similar al de la memoria secundaria. Además ayuda a realizar la compartición y la protección de información entre procesos. El modo de operación es

- Se divide un proceso en bloques
- Se divide la memoria real en marcos de bloques
- Se carga en la memoria real un pequeño conjunto de bloques del proceso
- Se van trayendo nuevos bloques según se van necesitando. A esto se le llama demanda de página.

Para poder implementarlo se necesita un mecanismo de traducción de direcciones que se realiza mediante tablas de página y un conjunto de estrategias de administración. Se dice que se ha producido un fallo de pagina cuando se intenta acceder a una página que no está en memoria principal. Los fallos de página los gestiona el sistema operativo.

MECANISMO DE TRADUCCIÓN DE DIRECCIONES

Es el mecanismo que convierte las direcciones virtuales generadas por el procesador en las direcciones reales en tiempo de ejecución.



Debido a la alta penalización de los fallos de página, a los diseñadores les gustaría reducir el número de fallos de página optimizando la colocación de las páginas en la memoria principal. Si permitimos que una página virtual se corresponda con cualquier página física, el sistema operativo puede realizar entonces la sustitución de cualquier página cuando se presente un fallo de página. En estos casos el sistema operativo puede utilizar un algoritmo sofisticado para seleccionar una página que nos se vaya a utilizar durante mucho tiempo. Esto lleva al diseño de sistemas llamados totalmente asociativos, en los que cualquier página virtual se puede colocar en cualquier marco de página de la memoria real.

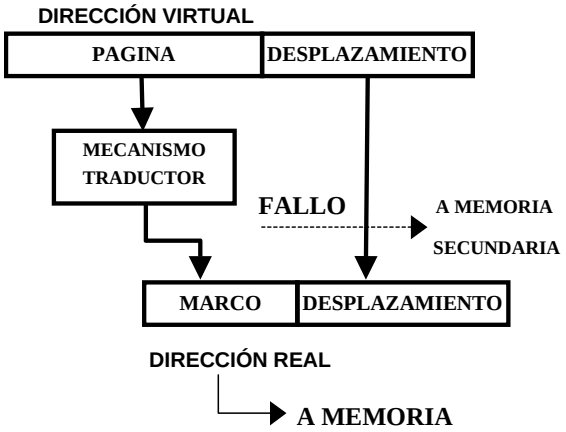
Si una página puede residir en cualquier marco de la memoria principal se necesita un mecanismo para encontrarla. Este mecanismo es la tabla de páginas que es una tabla que relaciona cada dirección virtual que proporciona el procesador con la dirección real donde se almacena la información. Los mecanismos de traducción de direcciones se pueden clasificar en directos, asociativos y T LB.

Nota no confundir la tabla de datos asociativa con la memoria virtual totalmente asociativa En el primer caso hace referencia al tipo de memoria que se utiliza para implementar la traducción mientras que en el segundo indica el hecho de que cualquier pagina se pueda colocar en cualquier marco. De hecho independientemente de que la tabla de página sea directa, asociativa o asociativa directa la memoria virtual que implementan es totalmente asociativa.

15.3 MEMORIA VIRTUAL PAGINADA

Según las características de los bloques en que se dividen los procesos y la memoria, los sistemas de memoria virtual se pueden clasificar en paginados, segmentados y paginados segmentados. De la combinación de estas características de bloque con los diferentes mecanismos de traducción de direcciones se obtiene diferentes tipos de memoria virtual.

En la memoria virtual paginada la memoria principal se divide en conjuntos de posiciones de igual tamaño denominados marcos de página. A su vez los programas se dividen en conjuntos de instrucciones, llamados páginas. Estas páginas tienen el mismo tamaño que los marcos de página. No es necesario que todas las páginas de un programa estén dispuestas de forma consecutiva en la memoria principal, ni es necesario que todas las páginas estén en MP. El espacio virtual está compuesto por páginas de tamaño fijo. La dirección virtual viene dado por un par (p,d) donde p indica el número de página virtual y d el desplazamiento respecto a la posición inicial de la página. Por otro lado, una dirección física es un par (m,d) donde m es un número de marco y d un desplazamiento que coincide con el de la dirección virtual.



15.3.1 MECANISMO DE TRADUCCIÓN DE DIRECCIONES

La traducción de direcciones se realiza mediante una tablas de páginas que relaciona el número de página virtual con el número de marco de página. Este número de página virtual es la dirección de acceso a la tabla que contiene la dirección del marco en el que se almacena. El sistema operativo, que es el que genera la tabla cuando se activa por primera vez un proceso, debe mantener tantas tablas como procesos activos haya.

La tabla de páginas más sencilla es la que tiene una entrada por cada página virtual. Por lo tanto cuanto mayor es el proceso mayor es la tabla de páginas, con lo que se pueden producir problemas de excesivo gasto de memoria. La implementación de la tabla de páginas se pude realizar de tres formas diferentes directa, asociativa y TLB.

TABLA DE DIRECCIONES DIRECTA

En un registro se guarda la posición en la que se inicia la tabla de direcciones. El número de página virtual es un desplazamiento relativo al registro anterior. No se debe confundir este desplazamiento (p) con el que aparece en la dirección virtual(d).

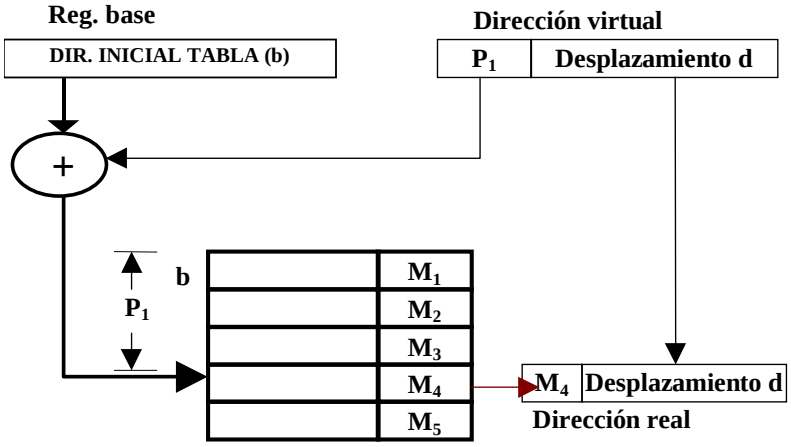


Tabla de página directa

La tabla de páginas se puede implementar en la memoria principal cuando las tablas son grandes, aunque esto relentiza el acceso porque para obtener un dato se tienen que hacer dos accesos a memoria principal. También se puede implementar en un banco de registros cuando la tabla es pequeña. Esta opción es mucho más rápida puesto que los tiempos de acceso a los registros son menores que los tiempos de acceso memoria principal. En realidad esta opción no se utiliza debido al gran tamaño de las tablas de páginas. Se llama *entrada de la tabla de páginas* a una posición de la tabla. Esta entradas debe contener la siguiente información

- Nº de Marco de página en el que se guarda la página.

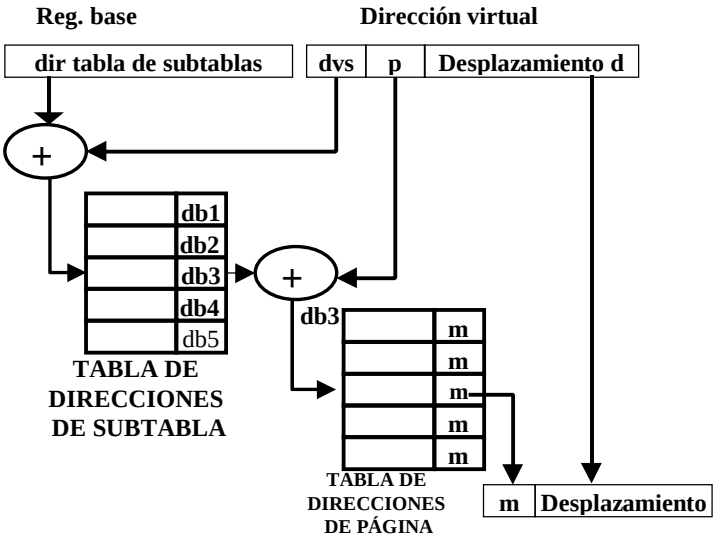
- F: indica si la página por la que se pregunta existe o no.
- M: indica si la página se encuentra en la memoria principal o no
- C: indica si el contenido de la página se ha modificado o no

Cuando los procesos son muy grandes se suelen guardar las tablas de página en memoria principal. Sin embargo si el proceso es excesivamente grande, la tabla de página puede ocupar demasiado espacio. Un ejemplo. Es la arquitectura VAX en la que cada proceso puede ocupar exactamente 2Gbytes (2^{31}). Si el tamaño de las páginas es de 512 bytes (2^9) esto indica que se necesitan tablas con 2^{22} entradas por cada proceso. Estas tablas son demasiado grandes para almacenarlas en memoria principal. Para solucionar este problema se utiliza las siguientes estrategias:

- Tabla multinivel
- Tablas de página invertida
- Registro límite

TABLAS DE 2 NIVELES:

Esta solución es la que utiliza el microprocesador Pentium de Intel para tablas demasiado grandes. En ella la tabla de páginas se divide en subtablas de manera que una tabla tiene el tamaño de una página. Algunas de ellas se almacenan en memoria principal y otras en memoria secundaria. No es necesario que las subtablas estén en posiciones consecutivas de memoria. Cada subtabla tiene su propia dirección base que se almacenan en un directorio de subtablas. En el directorio existen tantas direcciones de subtablas como subtablas. La dirección del directorio se almacena en un registro base. El mecanismo de traducción se puede ver en la siguiente figura:



Donde:

- Dvs es la dirección virtual de la dirección de la subtabla
- dvs + DIR BASE = La dirección en la que se encuentra la dirección de la subtabla tabla (db3)
- db3 +p = Será la posición de la tabla que contiene parte de la dirección física (m)
- m + DESPLAZAMIENTO = Dirección física

En este mecanismo existen dos tablas. La primera corresponde a las direcciones de las subtablas y la segunda es la subtabla a la que se accede. Ahora, por lo tanto se pueden producir dos fallos de página, uno por cada tabla. Aunque aquí solo se ha explicado la tabla de dos niveles es habitual encontrar en los sistemas tablas de varios niveles, como es el caso de la memoria virtual del sistema dec 3000.

Nota para llegar hasta esta estructura es suficiente imaginarse una tabla muy grande que se divide en subtablas de tamaño página. Cada una de estas tablas necesitaría un registro base para almacenar la dirección en la que comienza. En lugar de utilizar tres registros almaceno las direcciones de las subtablas en la memoria principal. Ahora necesito un puntero que me indique donde se almacenan las direcciones de

subtabla. De esta manera implemento una tabla de direcciones de subtabla. Como estas subtablas tiene tamaño página podrían estar en Mp o Ms. Luego una entrada de la tabla de direcciones de subtabla también debe contener información de si la subtabla está en MP o MS.

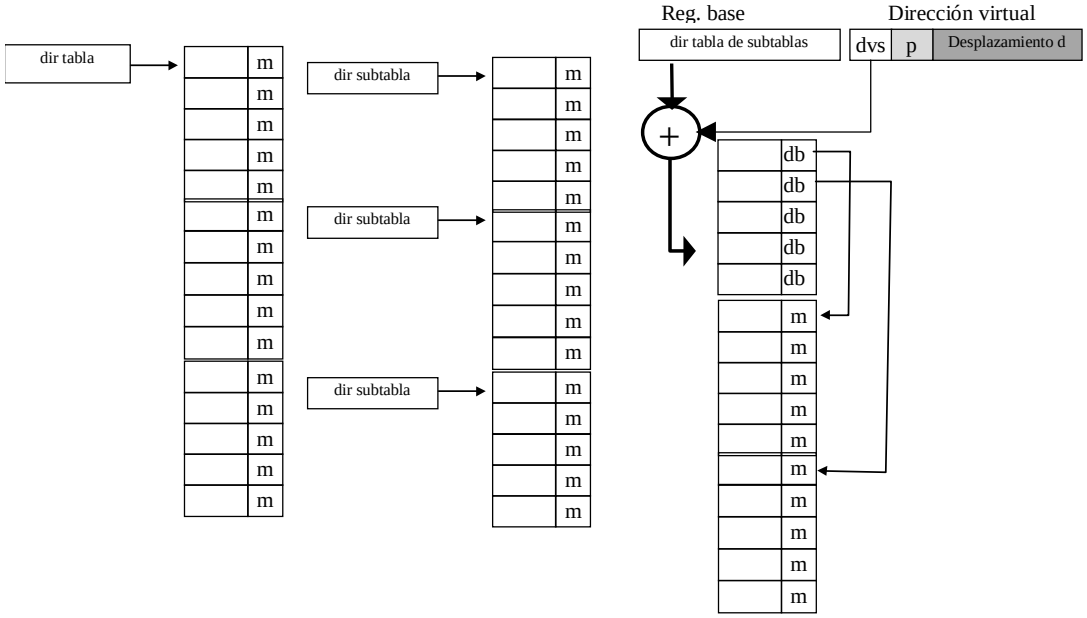


TABLA DE PÁGINAS INVERSA

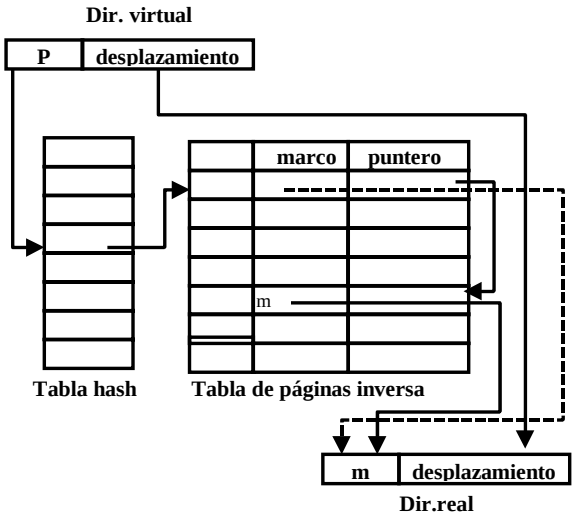
Esta es la solución que utilizan los equipos AS/400 de IBM y los RISC PowerPC. En lugar de una entrada por cada pagina virtual tiene una entrada por cada marco de página de la memoria física asignada al proceso. Es decir la tabla inversa solo da información de las páginas guardadas en los marcos. Luego si se produce un fallo debe existir un mecanismo de traducción de la información a memoria secundaria.

Se utiliza una tabla hash para implementar la tabla. El número de página virtual se mappea a esta tabla mediante una función hashing sencilla. Recordemos que una función hash hace corresponder números que se encuentran en un rango 0-m a números que se encuentran en el rango 0-n siendo m>n. La salida de esta función se utiliza como índice de la tabla hash.

Es posible que más de una entrada apunte a la misma posición de la memoria principal. Cuando una posición está ocupada se crea un puntero a otra posición. Las cadenas que se crean con las técnicas hashing son habitualmente cortas. En estos casos para saber qué marco contiene una página hay que recorrer toda los enlaces hasta el final. Nota esta demostrado que las cadenas que se formas son pequeñas. La tabla hash o tabla inversa, contiene la siguiente información:

- Pagina: indica la página almacenada en esa dirección
- ETP: Entrada de tabla de página que contiene el número de marco. Esta entrada de tabla es equivalente a la ya vista y contiene información similar.
- Puntero: a otra dirección de la tabla

En las tablas de páginas normales se tienen tantas entradas como páginas virtuales luego pueden crecer enormemente. En la tabla de páginas inversa hay tantas entradas como marcos de página asignados al proceso el sistema operativo. Su principal ventaja es que el tamaño de la tabla es constante, independientemente el tamaño del proceso. Como inconveniente tiene que la paginación es un poco más compleja pues hay que aplicar previamente la función hash.



REGISTRO LÍMITE

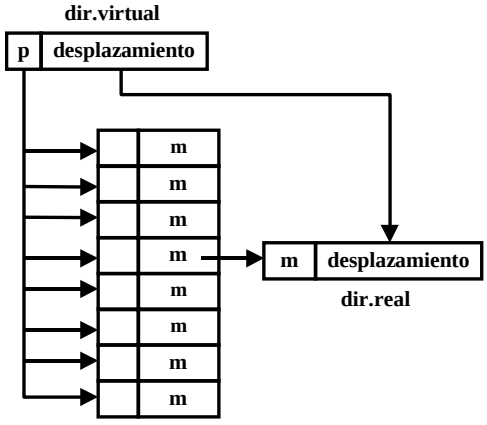
La idea es utilizar un registro limite que acote el tamaño máximo de la tabla de páginas para un proceso dado. Si el número de páginas virtuales se hace más grande que el contenido del registro límite deben añadirse entradas a la tabla de páginas. Esto permite que la tabla crezca cuando un proceso consuma mucho espacio. Esta técnica necesita que el espacio de direcciones se expanda solo en una dirección.

INCONVENIENTES DE LA TRASLACIÓN DIRECTA

Como las tablas suelen ser muy grandes se implementan casi siempre en memoria principal. Esto provoca que cada lectura de una instrucción necesite dos accesos a memoria principal, uno para obtener el número de marco, otro para leer la instrucción propiamente dicha. Por lo tanto, los programas se ejecutan a la mitad de su velocidad normal. Esto es intolerable, por lo que deben utilizarse métodos más rápidos de traducción. Una solución es implementar la tabla en la memoria cache en lugar de la memoria principal.

TABLA DE DIRECCIONES ASOCIATIVA

Una solución podría ser implementar la tabla mediante una memoria asociativa. La asociación se realizaría con el campo página. Esto solo reduciría tiempos de acceso cuando la memoria asociativa fuera muy pequeña, porque en caso contrario, aumenta notablemente el costo del circuito así como los retardos y la complejidad.



La forma de actuar es comprobar todas las posiciones de la tabla simultáneamente. De los dos acceso necesarios para obtener el objeto el primero es muy rápido. Su principal inconveniente es que si la tabla es muy larga el coste es muy elevado y se degrada el rendimiento. En realidad no existen sistemas virtuales implementados con memorias asociativas, es muy caro y muy lento, puesto que la memoria tendría que tener en ocasiones hasta 2^{22} . Se estudia para comprender el mecanismo de TLB que se explica a continuación.

ACELERACIÓN DEL MECANISMO DE TRADUCCIONES. (TLB)

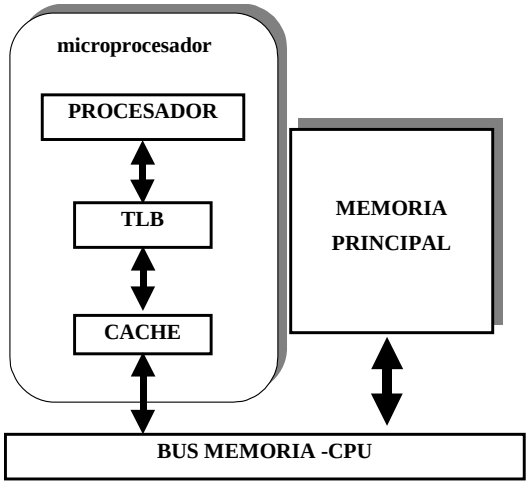
Translation Lookaside Buffer

NOTA: es muy importante darse cuenta que la principal ventaja del TLB es estar integrada dentro del mismo chip procesador y que por lo tanto sus tiempos de acceso son un orden de magnitud inferiores a los de la memoria principal.

El TLB es una memoria asociativa pequeña y rápida incluida en el chip de microprocesador que contiene una tabla de las páginas referenciadas más recientemente (captura la localidad temporal). Esto produce un efecto cache para la tabla de páginas de la memoria principal. La forma general de funcionamiento es la siguiente: cuando el procesador proporciona una dirección, se mira en la TLB. Si está la página se accede a ella y si no está se mira en a la tabla directa almacenada en la memoria principal. Es importante comprender que el TLB no contiene todas las paginas de la tabla, sólo las páginas accedidas más recientemente luego se aprovecha de la localidad de referencias a memoria.

Una entrada al TLB contiene menos información que una entrada a la tabla de páginas. Esta información es la siguiente: la dirección de memoria principal en la que se encuentra la página que se busca (no incluye la dirección de memoria secundaria) , el bit de escritura (dirty) que nos indica si se ha realizado alguna escritura sobre la página y el bit de referencia que indica si se ha referenciado alguna vez y que sirve para implementar las estrategias de reemplazamiento.

Su principal ventaja es que al encontrarse en el chip del microprocesador y además ser pequeña es muy rápida. Con ella se consigue acelerar la traducción de direcciones al máximo, acercando los tiempos de acceso a memoria virtual a los tiempos de acceso a Memoria Principal (de los dos accesos de la direccionamiento directo se reduce a uno muy rápido y el normal de acceso a memoria

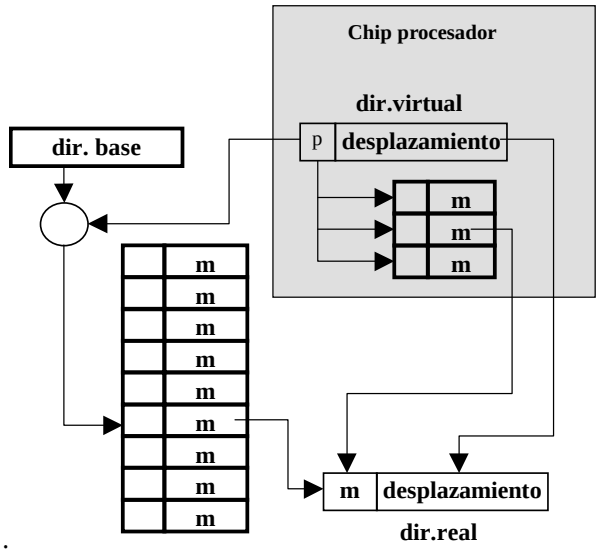


Si se presenta un fallo en el TLB se debe determinar si es un fallo de página o un fallo de TLB. Como el TLB tiene muchas menos entradas que las tablas de páginas, los fallos de TLB serán mucho más frecuentes que los verdaderos fallos de página (aun así se sigue esperando que sea una tasa relativamente pequeña).

Cuando hay un fallo de TLB hay que acceder a la tabla almacenada en la memoria principal para conocer la página de memoria principal a la que se quiere acceder. Una vez realizado el acceso hay que seleccionar una entrada del TLB para sustituirla por los datos de la última página accedida. Antes de sustituir el TLB hay que realizar una operación de escritura sobre los bits de escritura y referencia de la entrada correspondiente en la tabla de direcciones. (esto se debe a la política de post -escritura que se explicará más adelante)

Como las perdidas de TLB son más frecuentes que los fallos de pagina deben tratarse de manera más rápida. Por eso aunque algunos sistemas implementan estrategias LRU para seleccionar la entrada a sustituir en el TLB generalmente se hace una selecciona aleatoria. Ejemplo de TLB : El DECStation 3100 tiene un procesador un MIPS R2000 que incluye el TLB en el microprocesador. El procesador usa páginas de 4KB, el nº de paginas virtuales es 2^{20} el espacio de direcciones físicas es igual que el virtual, el TLB contiene 64 entradas, es totalmente asociativo y es compartido por instrucciones y datos. Cada entrada tiene 64 bits de los cuales 20 son de etiqueta, 20 son el marco de página física un bit de validez, un bit de escritura y varios bits de información auxiliar. Cuando se produce una perdida se trata mediante software (microcódigo). Aunque

una perdida puede tener un tiempo de tratamiento mínimo de 10 ciclos, lo normal es que utilice 16. Las entradas recomendadas para reemplazar se seleccionan mediante hardware de manera aleatoria.



15.3.2 ¿CÓMO SABER SI HA HABIDO UN FALLO DE PÁGINA?

La entrada de tabla de páginas contiene información de dos tipos, número de marco donde se almacena la página e información para controlar el proceso de fallo de página. Esta información viene especificada de la siguiente forma

V	C	LEX	M	P	DMP

Donde:

- V: bit válido
  - V=1: puede tener 2 significados en función de M
    - ⇒ tabla activa (memoria principal)
    - ⇒ no nula (memoria Secundaria)
  - V=0: página inactiva hay que crearla
- M: bit de memoria de Disco
  - M=1: página activa, en memoria principal
  - M=0: página no nula, en memoria secundaria
- DMP: dirección del marco de página, depende del valor de M
  - Si M=1 indica una dirección de la memoria principal
  - Si M=0 indica una dirección de la memoria secundaria
- LEX: conjunto de bits que indican los tipos de acceso permitidos
- P: indica la privacidad, pues una misma página puede pertenecer a diferentes procesos

15.4 ¿CUÁL DEBE SER EL TAMAÑO DE MEMORIA?

El tamaño de memoria debe ser un compromiso entre:

- La eficiencia de la memoria
- Tamaño de las tablas páginas
- Tamaño medio de las entidades lógicas del programa

Para páginas grandes:

- Tablas de páginas más pequeñas
- Mayor fragmentación
- Se captura la localidad espacial
- Mayor lentitud en cargar las nuevas páginas (por lo tanto mayor penalización)

Para páginas pequeñas

- Tablas de páginas mayores
- Acceso por memoria asociativa limitado
- Captura la localidad temporal

15.5 MEMORIA VIRTUAL SEGMENTADA

En este tipo memoria un proceso puede ocupar varios bloques separados de almacenamiento real. Estos bloques pueden ser de diferente tamaño y el segmento se ajusta al tamaño variable de las estructuras de programa. Las instrucciones del segmento se almacenan en posiciones consecutivas de memoria.

Sirve para organizar programas y datos y asociar privilegios y atributos a programas. Explota la técnica de modularidad de los programas estructurados. Necesita rutinas de transferencia más complejas que las paginadas porque deben tener en cuenta el tamaño variable del segmento. La paginación es invisible al programador y su único objetivo es implementar la memoria virtual. La segmentación es visible al usuario. La memoria virtual segmentada se puede ver como múltiples espacios de direcciones. La asignación de programas y datos al segmento la pueden realizar tanto el programador como el sistema operativo.

15.5.1 MECANISMO DE TRADUCCIÓN DE DIRECCIONES

S= nº de segmento

d= desplazamiento

Al igual que en el caso de la paginación la traducción puede ser directa, asociativa, TLB. El mejor rendimiento se consigue con la TLB por las mismas razones que se daban en la memoria virtual paginada. Solo estudiamos la traducción directa. En este caso la dirección virtual segmentada se compone de dos campos uno s que indica el numero de segmento y otro d que indica el desplazamiento en el segmento. El hw necesario se puede ver en la siguiente figura.

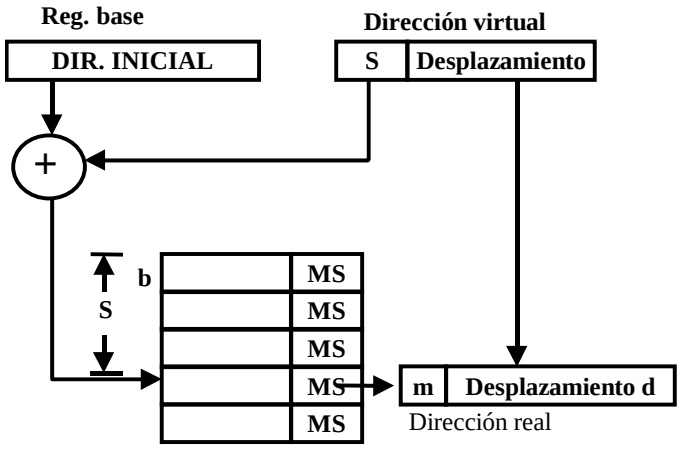


Tabla de segmentos directa

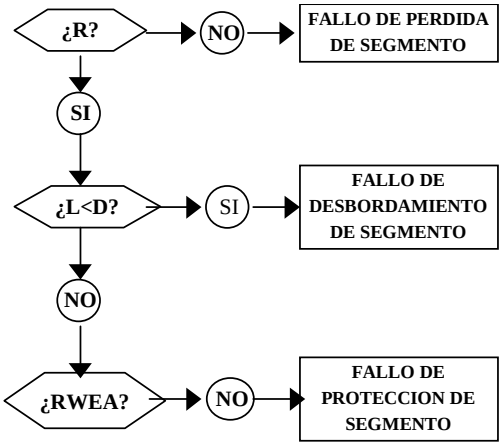
En al tabla existe una posición por segmento llamada Entrada de tabla de segmentos (ETS), que contienen la siguiente información:

r	A	L	RWEA	S´

Donde:

- r: bit de residencia del segmento
- A: dirección del almacenamiento secundario (si el segmento no está en la dirección real).
- L: longitud del segmento.
- RWEA: bits de protección
  - ⇒ R: escritura
  - ⇒ W: lectura
- E: acceso de ejecución.
- A: acceso de adición.

- Fallo de pérdida de segmento . Si el segmento no está en el almacenamiento real r =0. Entonces el S.O obtiene el control y carga el segmento desde la dirección del almacenamiento ¿secundario?
- Para descodificar la dirección hay que comprobar si el desplazamiento D de la direcciona virtual es menor que la longitud L del segmento.
  - L<D Fallo de desbordamiento de segmento
    - ⇒ S.O toma el control y suspende la ejecución del proceso.
  - L>D se comprueban los bits de protección para comprobar que la operación está permitida. Si operación no está permitida fallo de protección de segmento.



VENTAJAS DE LA SEGMENTACIÓN:

- Simplicidad del manejo de las estructuras de datos. El S.O maneja su crecimiento y decrecimiento.
- Permite modificar y compilar programas de manera independiente, sin necesidad de lincar y cargarlos todos.
- Permite la comparticion entre procesos
- Facilita la protección

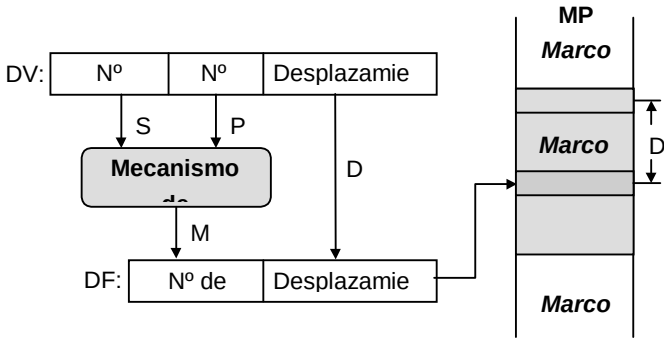
15.6 MEMORIA VIRTUAL PAGINADA- SEGMENTADA

Tanto la paginación como la segmentación tienen sus ventajas. La paginación es transparente al programador, elimina la fragmentación externa, y por lo tanto aprovecha la memoria principal de manera más eficiente. Además como las páginas son del mismo tamaño se pueden diseñar algoritmos de gestión de memoria sofisticados. Por otro lado la segmentación, que es visible para el programador, tiene las ventajas que acabamos de ver. Para aprovechar las ventajas de ambas algunos sistemas permiten las memorias segmentadas –paginadas, en las que el espacio de direcciones se divide en segmentos que a su vez se dividen en páginas. Las características generales de la segmentación-paginación son:

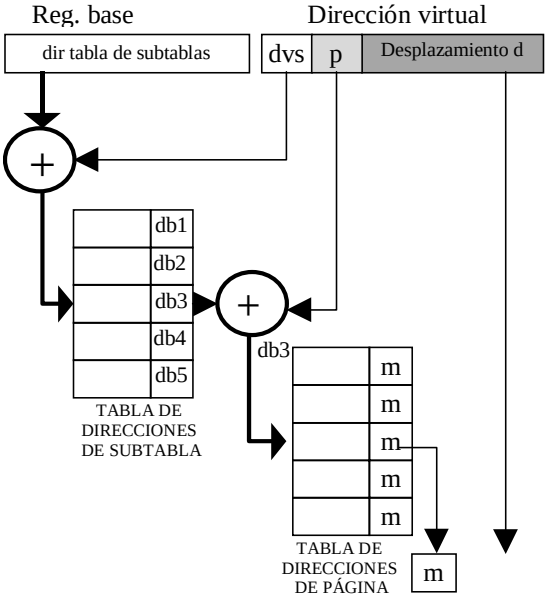
- Tamaño de los segmentos múltiplo de las paginas luego su tamaño mínimo es una página

- No es necesario que todos los segmentos del programa estén en MP.
- No es necesario que todas las páginas del segmento estén en MP.
- Segmentos de un mismo programa no necesitan ser continuos.
- Páginas del mismo segmento no necesitan ser continuas
- Es la forma más habitual de la más habitual de implementar la memoria virtual
- ejemplo Power PC y pentium

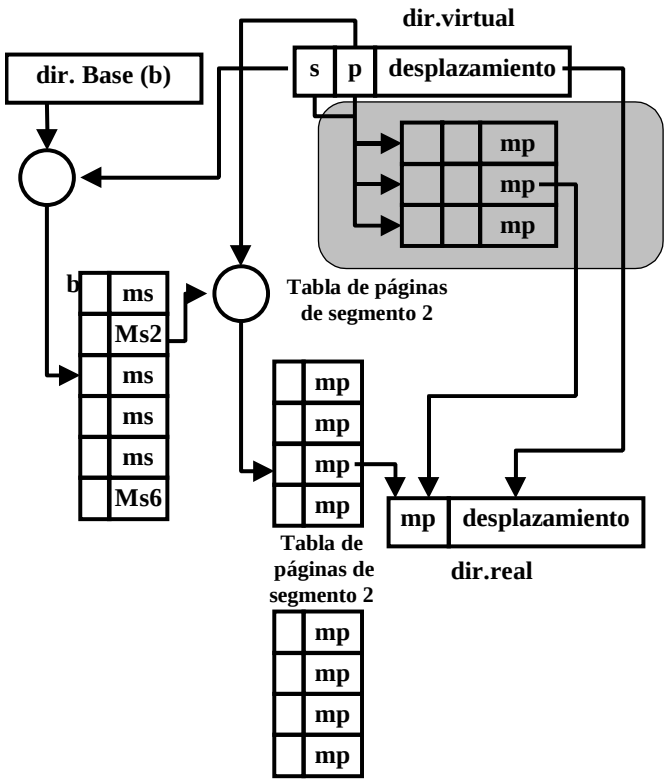
A continuación vemos como hacemos para implementar esto. Vamos a suponer una dirección virtual dividida en tres campos (S;P;D) de manera que S me indica el segmento del trabajo, P la página del segmento y D la palabra del segmento. El mecanismo de traducción debe ser algo parecido a lo siguiente:



Si recordamos como era la paginación directa de dos niveles vemos que la estructura era muy parecida:



Vista esta analogía vamos a ver como se implementaría una segmentación-paginación con TLB



Nota: la asociación del TLB se hace tanto con la página como con el segmento.

Conviene saber lo siguiente: La tabla de segmentos de un trabajo esta siempre en memoria principal. Las tablas de páginas correspondientes a un determinado segmento pueden estar o no estar en memoria principal, por lo tanto una entrada a la tabla de segmentos debe indicar si la tabla de páginas del segmento esta en Mp o en MS. Un segmento está en MP si su tabla de páginas está en MP (aun cuando no haya ninguna página de dicho segmento en MP). Si la tabla de páginas de un segmento no esta en MP se puede producir un fallo de segmento Y como tal tendrá que tratarse puesto que las tablas de páginas son de tamaño variable. El tratamiento se hace página a página. Conceptualmente es idéntico al de dos niveles solo que las subtablas de tamaño variable en lugar de tamaño fijo.

15.7 ESCRITURA EN LA MEMORIA VIRTUAL

[Patterson]

Recordemos que en la memoria cache la diferencia entre el tiempo de acceso a la cache y la memoria principal es de decenas de ciclos y por eso puede utilizarse escritura directa, (se actualizan la MP y la cache simultáneamente). En ocasiones se utiliza un buffer para ocultar la latencia de la escritura del procesador. En memoria virtual las escrituras en el disco necesitan cientos de miles de ciclos (20.000.000 ns) luego la escritura directa es impracticable.

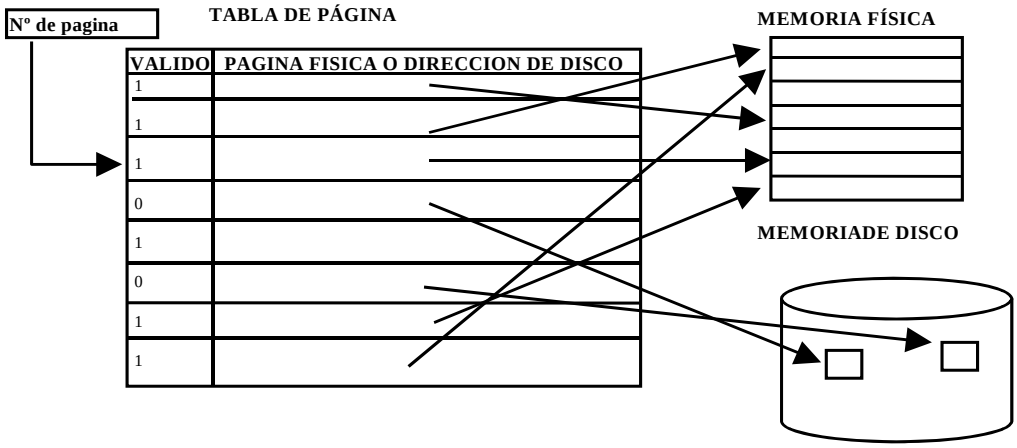
La estrategia que utiliza la memoria virtual es de post-escritura. Las escrituras individuales se acumulan en una página y cuando la página se reemplaza se escribe en el nivel inferior, en lugar de eliminarla sin más. Esto tiene la Ventaja adicional de que como el tiempo de transferencia del disco es pequeño comparado con su tiempo de acceso, las post escritura de una pagina completa es más eficiente que la escritura de palabras sueltas.

Esta técnica es todavía mejorable si se puede indicar si la página necesita post escritura o no. Esto se consigue añadiendo un bit de modificación. Este bit se pone a uno la primera vez que se modifica la página.

15.8 ESTRATEGIAS DE ADMINISTRACION

Un fallo de referencia ocurre cuando se referencia un objeto que no está en memoria real (bit de validez a cero). Cuando esto ocurre se produce una llamada al sistema operativo para que traiga un nuevo objeto mediante una excepción. El sistema operativo debe encontrar la página en el nivel de la jerarquía inferior y decidir en que parte de la memoria física se coloca la página requerida.

La dirección virtual no es suficiente para conocer la dirección del disco en la que se encuentra la página, luego de alguna forma se debe proporcionar esta dirección y relacionarla con la dirección virtual para que cuando exista un fallo se sepa a donde hay que ir. Como no se sabe cuando se va a sustituir una página el sistema operativo researva un espacio en el disco para guardar todas las páginas del proceso cuando lo crea. En ese instante crea una estructura de datos para indicar la zona de disco en la que se guarda cada página. Esta estructura puede incluirse dentro de la tabla de páginas o puede ser una estructura auxiliar similar.

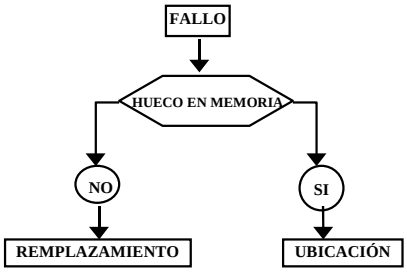


Gestión de fallos:

- Estrategias de ubicación
- Estrategias de reemplazamiento
- Estrategias de búsqueda

¿Cuándo se utiliza una estrategia u otra?

- Cuando se produce un fallo de referencia lo primero es comprobar si existen elementos en la lista de huecos
- Si existen huecos se aplican las estrategias de ubicación
- Si no existen huecos se aplica las estrategias de reemplazamiento



- **ESTRATEGIAS DE ADMINISTRACIÓN DE LA MEMORIA.**

De búsqueda:

- Anticipada
- Demanda

Ubicación

- First
- Best

- Worst
- Bussy

Reemplazamiento

15.8.1 ESTRATEGIAS DE UBICACION

Los sistemas paginados toman las decisiones de manera trivial, pues una página nueva se puede colocar en cualquier marco que esté disponible. Los sistemas segmentados necesitan estrategias similares a las de multiprogramación de partición variable de la memoria real. Estas estrategias se exponen a continuación

- Primero disponible (FIRST)
- Mejor ajuste (BEST)
- Peor ajuste (WORST)
- Buddy

PRIMERO DISPONIBLE

- Almacena el programa en el primer espacio libre que pueda contenerlo.
- Fácil de implementar.
- No necesita recorrer el espacio de direcciones completo.
- Lista de huecos libres (No necesita ordenarla)

MEJOR AJUSTE

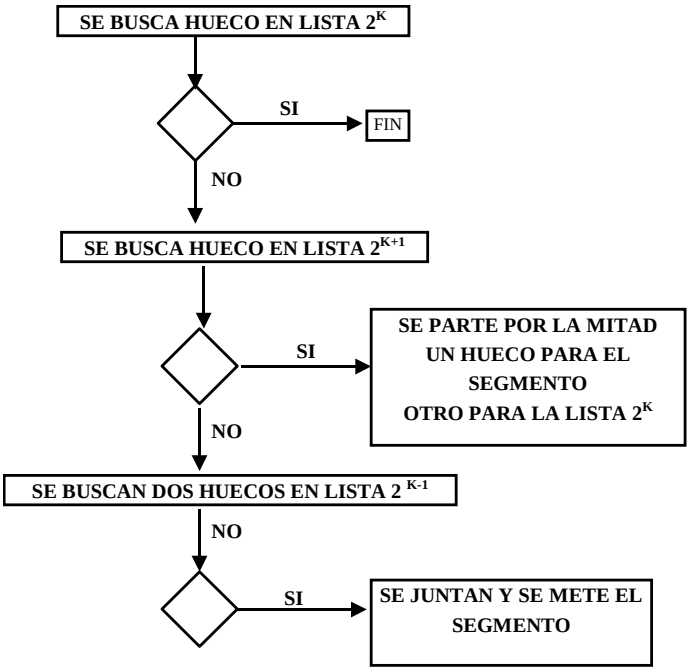
- Se le da el hueco en que encaja mejor.
- Objetivo. Evitar el desperdicio de memoria.
- Lista de huecos disponibles ordenada.
- **Problemas: es d**ifícil de implementar y puede ser necesario recorrer todo el espacio de memoria para encontrar el hueco óptimo. Divide los huecos existentes en huecos más pequeños

PEOR AJUSTE

- Hueco en el que encaja peor.
- Evita la aparición de huecos pequeños.
- Lista ordenada de más a menos.

BUDDY

- Se mantienen n listas de huecos de tamaño diferentes, potencia de 2.
  Lista1 huecos de tamaño 2^1
  Lista2 huecos de tamaño 2^2
  ListaN huecos de tamaño 2^N
- Un hueco se puede dividir en 2 de igual tamaño, y dos huecos se pueden unir en otro de doble tamaño. La estrategia es la siguiente:



15.8.2 ESTRATEGIAS DE REMPLAZAMIENTO

Se aplican cuando no existen huecos en la memoria principal El S.O debe decidir que página desplazar para hacer sitio a una nueva página

Estrategias**:**

- Reposición de páginas al azar
- Primero en entrar primero en salir
- Menos recientemente usada
- Menos frecuentemente usada
- No usada recientemente
- Conjunto de trabajo

Para obtener un rendimiento óptimo, la pagina que se reemplaza es una que no va a ser utilizada en el futuro durante un tiempo largo. Esto es una utopía pues no podemos conocer el futuro, por lo tanto las estrategias se basan en lo que ha ocurrido en el pasado.

ν **REPOSICIÓN DE PAGINA AL AZAR**

- Poca sobrecarga
- No es discriminatoria con ningún usuario
- Todas las páginas tienen la misma probabilidad de ser remplazadas
- Se utiliza poco.

ν **PRIMERO EN ENTRAR PRIMERO EN SALIR (FIFO)**

- Se pone una marca de tiempo en cada página al entrar en la Memoria principal.
- Se reemplaza la que lleva más tiempo almacenada.
- Desventaja: se puede remplazar páginas muy usadas
- Se implementa una cola en el S.O.
- Anomalía FIFO cada proceso tiene un conjunto de marcos de página en Memoria principal predeterminado. Cuanto mas marcos de página tenga cada proceso más fallos de referencia se producen

ν **MENOS RECIENTEMENE UTILIZADA (LRU)**

- Reemplaza la que lleva más tiempo sin ser usada
- Necesita información de cuando se utilizo por última vez cada página. Esto trae consigo una sobrecarga adicional.
  - Poco usado
  - Fallo en ciclos largos que incluyen varias páginas. La página menos recientemente usada puede ser la próxima en referenciarse.

ν **MENOS FRECUENTEMENTE USADA (LFU)**

- Cuidado. La menos frecuentemente usada puede ser la ultima referenciada.

ν **NO USADA RECIENTEMENTE (NUR)**

- Es una aproximación a la LRU, pero con menos sobrecarga.
- Heurística: la páginas no usadas recientemente tienen pocas probabilidades de usarse en un futuro próximo
- Cada página se implementa con 2 bits
  - ∗ bit referenciado:
    - ⇒ 0 si la página no ha sido referenciada.
    - ⇒ 1 si la página ha sido referenciada.
  - ∗ bit modificado:
    - ⇒ 0 si la página no ha sido modificada.
    - ⇒ 1 si la página ha sido modificada.

- Modus Operandi:
  - ∗ Inicialmente todos los bits a 0.
  - ∗ Al referenciarse el bit referenciado a 1
  - ∗ Cuando se modifica el bit modificado a 1

  Cuando una página se reemplaza se busca en este orden

  1. no refenciado no modificado
  2. no referenciado modificado
  3. referenciado no modificado
  4. referenciado modificado

El segundo grupo no es un caso real. Esta estrategia tiene un peligro: llega un momento en que casi todos los bits referenciados a 1 por lo tanto se precisa capacidad de decisión. La solución es la siguiente: Periódicamente se referencian todos los bits 0. Como los bits modificados no se reajustan esto da lugar al grupo 2.

15.8.3 ESTRATEGIAS DE BÚSQUEDA

PAGINACIÓN POR DEMANDA

Ninguna página se trae al almacenamiento principal hasta que se haya referenciado explícitamente. Es la estrategia más común. Garantiza que las únicas páginas que se traen son las que necesita el proceso. Tiene varias desventajas, entre ellas que el sistema se sobrecarga poco. Además el proceso acumula sus páginas una a una por lo tanto cada vez que se referencia una página nueva debe transcurrir un tiempo a que se cargue en MP. Cuanto más páginas en MP más larga la espera por lo tanto cada vez más procesos en espera (Buscar el sitio donde se guarda o estudiar cual se reemplaza)

PAGINACIÓN ANTICIPADA

Intenta predecir las páginas que un proceso va a necesitar y las precarga cuando hay espacio disponible. Si las decisiones son correctas se reduce el tiempo de ejecución. lo

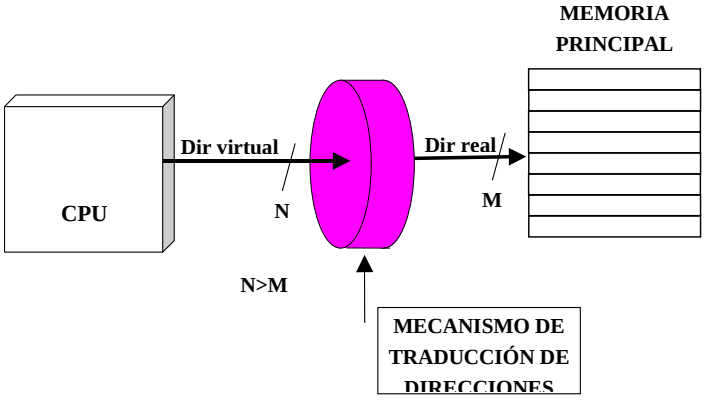
Aspectos que debe tratar: ¿Cuando hacer la prebúsqeda?, ¿Qué páginas deben prebuscarse?, ¿Qué estado de reemplazamiento debe asignarse a estas páginas?

- **Algoritmo OBL (One Block Cook Ahead)**
  - pg 108 Wang
  - Se busca en la localidad espacial
  - Se utilizan pilas de páginas
  - Cuando se produce una demanda de página se trae la página pi que se coloca en lo alto de la pila y la pi+1 que se coloca ???
  - Bien con buena secuencialidad de datos y programas de la tarea.

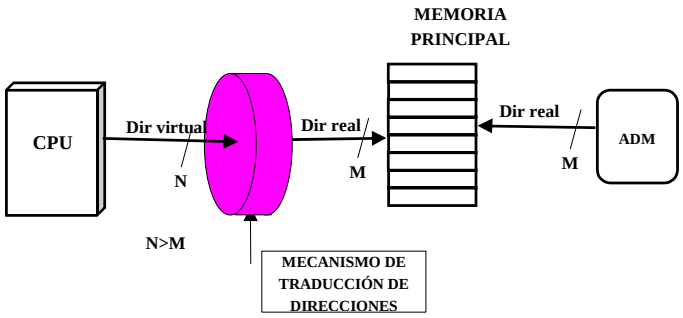
15.9 LA MEMORIA VIRTUAL Y EL ADM

pg527 del Hennessy

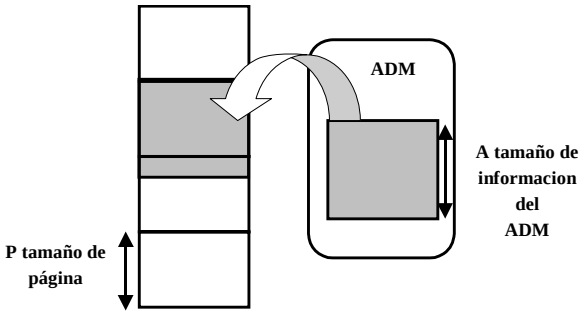
En este apartado se estudia como cargar la información en memoria principal a través del ADM (acceso directo a memoria), sabiendo que esta memoria forma parte de una virtual paginada. Como la dirección viene del ADM no atraviesa el mecanismo de traducción de direcciones luego siempre proporciona direcciones físicas. Cuando se incorpora al sistema la técnica de Entrada/Salida de acceso directo a memoria la relación entre el sistema de memoria y el procesador cambia. Sin ADM todos los accesos al sistema de memoria provienen del procesador y la traducción de las direcciones y los accesos a la cache se realizan desde el procesador.



Con el ADM esto ya no ocurre así, aparece otro camino que no va a través del mecanismo de traducción de direcciones o de la jerarquía de cache.



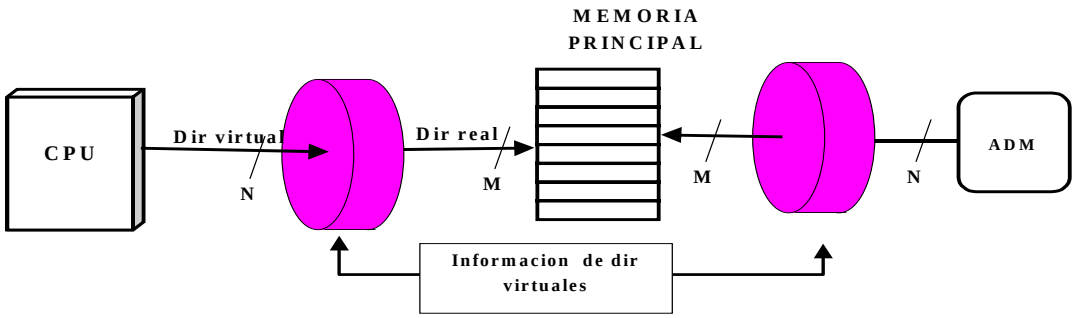
Esto provoca problemas tanto en la memoria virtual como en la memoria cache.En un sistema con memoria virtual ¿debería el ADM funcionar con direcciones virtuales o físicas? Existen dos problemas en la utilización de ADM física: Un problema aparece cuando se quiere transferir información desde la ADM a la memoria principal y esta información tiene un tamaño mayor que el tamaño de una página. Tal y como funcionan los ADM esta información se almacenaría en posiciones consecutivas de memoria principal y por lo tanto se rebosaría el tamaño de una página. La solución no permitir transacciones de tamaño superior a una página.



El otro problema aparece si el sistema operativo elimina alguna de las páginas de la memoria (o la recoloca). El ADM podría transferir o recibir datos erróneos. Este problema aparece porque el ADM no ve las operaciones de gestión que hace el sistema operativo sobre la memoria principal.

Una solución al problema es la utilización de direcciones virtuales que es un método para permitir que el sistema que inicie transferencia ADM cruce limites de páginas. Se pagina la información que envía el ADM de manera que ya no hay problemas de la cantidad de información que se envía. Para poder hacer esto el ADM debería usar también direcciones virtuales que deberían convertirse en direcciones físicas durante el ADM. De esta forma el buffer puede tener su contenido secuencialmente colocado en memoria virtual pero las páginas que lo componen no estar secuencialmente colocadas en memoria real.

En esta técnica debe haber comunicación de información entre los sistemas de dirección virtual para no machacar la información de memoria principal. El sistema operativo podría actualizar las tablas de página del ADM si un proceso se mueve usando ADM virtual, o el sistema operativo podría encerrar las paginas en la memoria hasta que el ADM se hubiera completado.

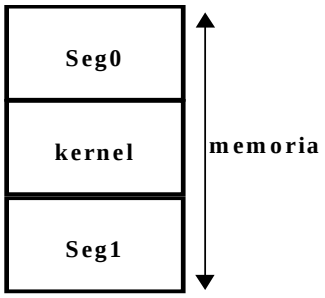


15.10 EJEMPLO ALPHA AXP 21064

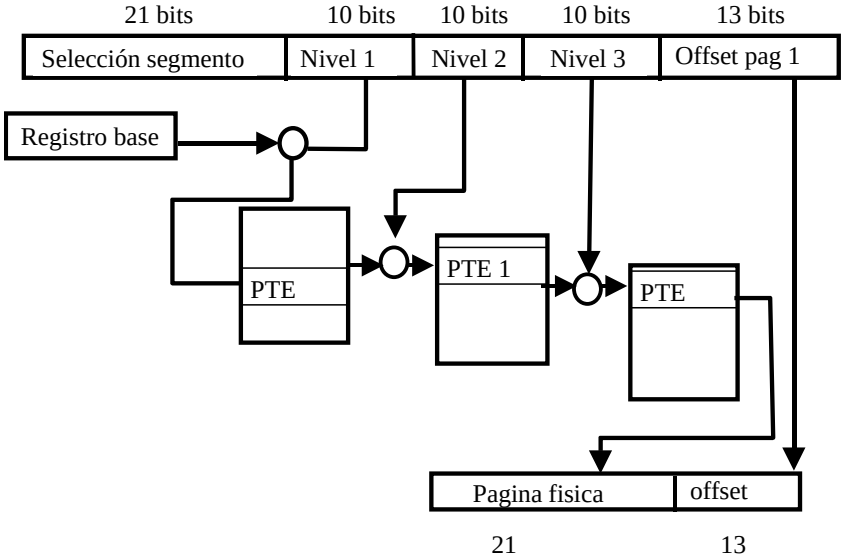
El alpha AXP usa una combinación de paginación y segmentación, de manera que proporciona protección al tiempo que reduce el tamaño de las tablas de página. El espacio de direcciones de 64 bits se divide en tres segmentos:

- Seg0 caracterizado por tener a cero el bit mas significativo de la dirección virtual (bit 63=0)
- SegK (bit 63 y 62 =10). Este segmento está reservado para el núcleo del sistema operativo, tiene una protección uniforme para todo el espacio y no usa el gestor de memoria, es decir se usa sin traducción de direcciones.
- Seg1: caracterizado por tener a 1 los dos bits mas significativo de la dirección virtual (bit 63 y 62=11).

Los procesos de usuario usan los segmentos 0 y 1 que se mapean en páginas con protección individual



El tamaño de las de página es de 8KB=2^{13} Byte. El tamaño de las tablas de página para el espacio de direcciones de 64 bits con una tabla de sólo un nivel sería muy grande, por esto el Alpha utiliza tablas de páginas jerárquicas de tres niveles.



La traslación comienza sumándole al nivel 1 el contenido de registro base de tablas de página. A continuación se lee la memoria para obtener la dirección base de la segunda tabla de página. El nivel 2 se suma a esta dirección base y se vuelve a hacer un acceso a memoria para determinar la dirección base de la tercera tabla de página. Esta dirección se suma al nivel 3 y se lee de nuevo la memoria para obtener la dirección física de la página referenciada. Esta información se concatena con el offset (desplazamiento) para obtener la dirección total. Es decir para conseguir la dirección física se necesitan tres accesos a memoria.

Cada tabla de página debe tener el tamaño de una página. Aquí hay que tener cuidado con los desplazamientos de la dirección virtual. El offset accede a bytes (para permitir después las instrucciones sobre tamaño byte) por eso utiliza 13 bits, mientras que los niveles acceden a palabras. Como cada palabra tiene 8 bytes los desplazamientos en estos casos son de 10 bits. **Nota**: La documentación del alpha permite extender el tamaño de mínimo de la página a 64 kb y por lo tanto se incrementa el tamaño de la dirección virtual a 3*13+16 =55 bits y la dirección física a 32+16=48 bits

La entrada de tabla de página (PTE Page Table Entry) es de 8 bytes (una palabra). Los primeros 32 bits contienen el número de marco de página física y el resto contienen los siguientes bits de protección:

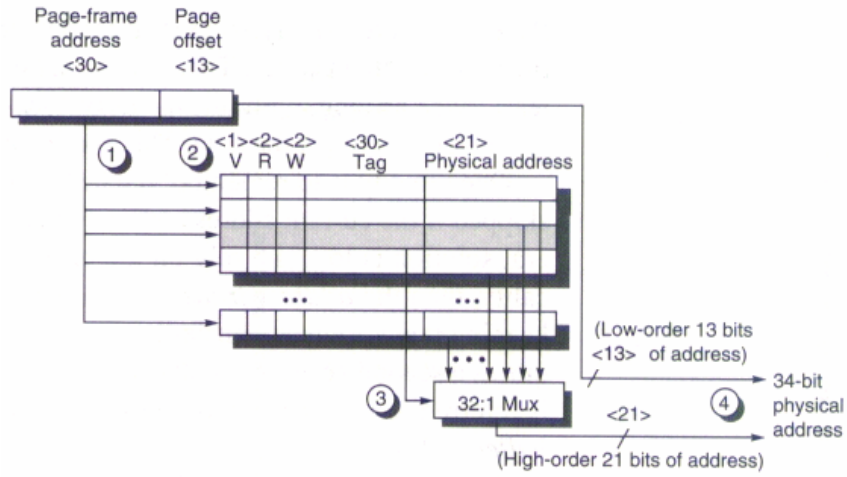
- *Valid* indica que el número de marco de página es válido para la traslación HW
- *User read enable*.- permite al usuario leer los datos de esa página
- *Kernel read enable*.- permite al kernel leer el dato de la página
- *User write enable* permite al usuario escribir datos en esta página
- *Kernel write enable* permite al núcleo escribir datos en esta página

Además el PTE tiene bits reservados para el SW de sistema. Puesto que hay que recorrer tres niveles de tablas cuando se produce un fallo de TLB existen tres lugares en los que potencialmente habrá que chequear las restricciones de protección.

¿Cómo se asegura el sistema que no se realizan traslaciones erróneas?. Las tablas de página no pueden ser modificadas por el usuario, de manera que éste puede manejar cualquier dirección virtual, pero es el sistema operativo quien se encarga de controlar las PTE, y por lo tanto es el que se encarga de controlar que dirección de memoria física es accedida.

Los TLBs

Ya hemos visto que la obtención de una dirección física supone tres accesos a la memoria. Si se usara únicamente este mecanismo de traducción de direcciones el retardo de ejecución sería enorme. El Alpha 21064 emplea dos TLB para reducir los tiempos de traslación. El TLB de datos tiene 32 PTE y el de TLB de instrucciones de 12 PTE (8 para páginas de 8kb y 4 para pg 4mb). Estas últimas páginas se usan para programas largos tales como los de los sistemas operativos o bases de datos. El alpha permite al sistema operativo decirle al TLB que secuencia continuas de páginas pueden actuar como una. Las opciones son 8, 64 y 512 (esto sirve para ajustar tamaño de página).



Los TLB del Alpha utilizan emplazamiento totalmente asociativo de manera que la traslación comienza cuando se envía la dirección virtual a todos los tags simultáneamente. No es necesario incluir los 13 bits de offset en el TLB. Un fallo de TLB invoca al software PAL (Privileged Architecture Library) que lo actualiza. Este SW son rutinas de lenguaje máquina que permiten el acceso a HW de bajo nivel. El código PAL se ejecuta tras descapacitar las excepciones.

Parámetros	*Descripción*
Tamaño de bloque	1 pte (8 bytes=una palabra)
Tiempo de acierto	1 ciclo de reloj
Penalización de fallos	20 ciclos de reloj
Tamaño TLB	Instrucción : 12=8 PTE para pg de 8kb+ 4 PTE para páginas de 4MB
	Dato: 32 PTE para páginas de 8kb
Selección de bloque	Aleatoria
Emplazamiento	Totalmente asociativo

15.11 LA MEMORIA VIRTUAL Y LA MEMORIA CACHE

15.11.1 DIFERENCIAS Y COINCIDENCIAS ENTRE LA MEMORIA CACHE Y LA MEMORIA VIRTUAL STONE PG 103

Los sistemas de memoria virtual cumplen un cometido similar al que cumple la memoria cache, con la diferencia de que se gestionan diferentes niveles de la jerarquía de memoria. Los algoritmos de mantenimiento de la memoria cache intentan hacer óptimo el uso de memorias de alta velocidad para las que

la memoria principal sirve de memoria auxiliar, de manera que los bloques activos se trasladan de la memoria principal a la cache y los inactivos de la cache a la principal.En los sistemas de memoria virtual se intenta hacer óptimo el uso de la memoria principal y la memoria secundaria se utiliza como memoria auxiliar. Los bloques activos se mueven de memoria secundaria a memoria principal y los bloques inactivos de memoria principal a memoria secundaria. Los principios que gobiernan la memoria cache y la memoria virtual coinciden:

- Guardar los bloques activos en la memoria de mayor velocidad
- Guardar los bloques inactivos en la memoria más lenta

Si el algoritmo tiene éxito el rendimiento se aproxima al de la memoria más rápida y el coste tiende al de la memoria más barata.

La diferencia entre la memoria cache y la memoria virtual se basa en la diferencia de penalización entre un fallo de página (virtual) y una perdida de cache. Una perdida de cache supones una penalización de 4 a 20 veces el tiempo de acceso a la cache. Una fallo de página supone un coste entre 10000 y 100000 veces un acceso a memoria principal. Estas diferencias se deben a que la memoria auxiliar de la memoria virtual son los discos magnéticos. Aunque los tiempos de lectura /escritura de los discos magnéticos a mejorado de 1000 a 1 en los últimos tiempos, las latencias debidas a las limitaciones mecánicas permanecen prácticamente constantes. (mejora en un factor de 10). Para el futuro el coste de un fallo de página sigue aumentando debido a que las memorias semiconductoras (MP) continúan mejorando su rendimiento mientras que no se consiguen mejoras significativas en los accesos a disco. Estas diferencias pueden llegar a tener un profundo impacto en los sistemas de memoria virtual.

Este alto coste de los fallos de página tiene como consecuencia las diferencias en las estrategias de administración de la cache y de la virtual. Durante una pérdida de cache el procesador debe permanecer ocioso puesto que la penalización es relativamente pequeña. Por lo tanto una perdida de cache no viene acompañada de un cambio de tarea. En un fallo de página, la penalización es tan grande que se carga otro proceso para no desaprovechar CPU. Por ejemplo en un sistema típico, la latencia de espera puede estar comprendida entre 1ms y 100 ms. En este tiempo el procesador 10MIPS puede ejecutar entre 10.000 y1.000.000 instrucciones. Esta última afirmación conviene matizarla, según el objetivo al que se dedique el sistema. Supongamos que se pretende que los tiempos de respuesta de todos los procesos sean lo más pequeños posibles. El rendimiento de la CPU aumenta al desalojar el proceso en espera del procesador, pero esto empeora notablemente el tiempo de respuesta puesto que este proceso debe esperar en una cola a que el procesador se pueda volver a hacer cargo de él.

15.11.2 DIRECCIÓN VIRTUAL Y MEMORIA CACHE

En los sistemas con memoria virtual y memoria cache aparece el problema de cómo se deben relacionar. El procesador da direcciones virtuales a la Memoria Principal. Estas direcciones tienen que pasar primero por la memoria cache. Este problema se puede tratar de dos formas diferentes:

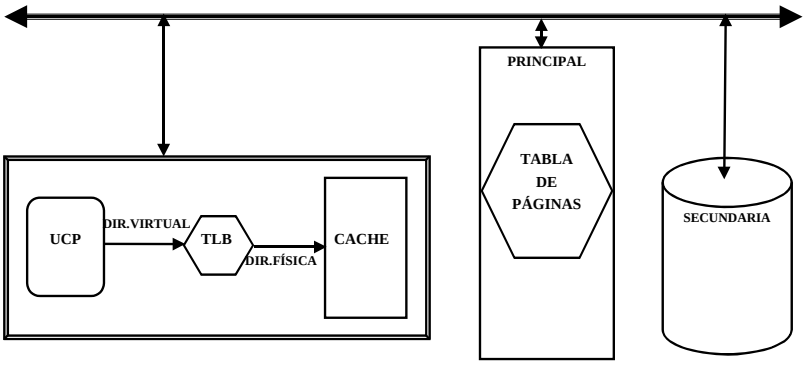
- Direccionamiento de la cache a través del TLB (cache física)
- Direccionamiento de cache a través de direcciones virtuales (cache virtual)

La diferencia radica en que en el primer caso las etiquetas que guarda la cache son reales mientras que en la segunda son virtuales.

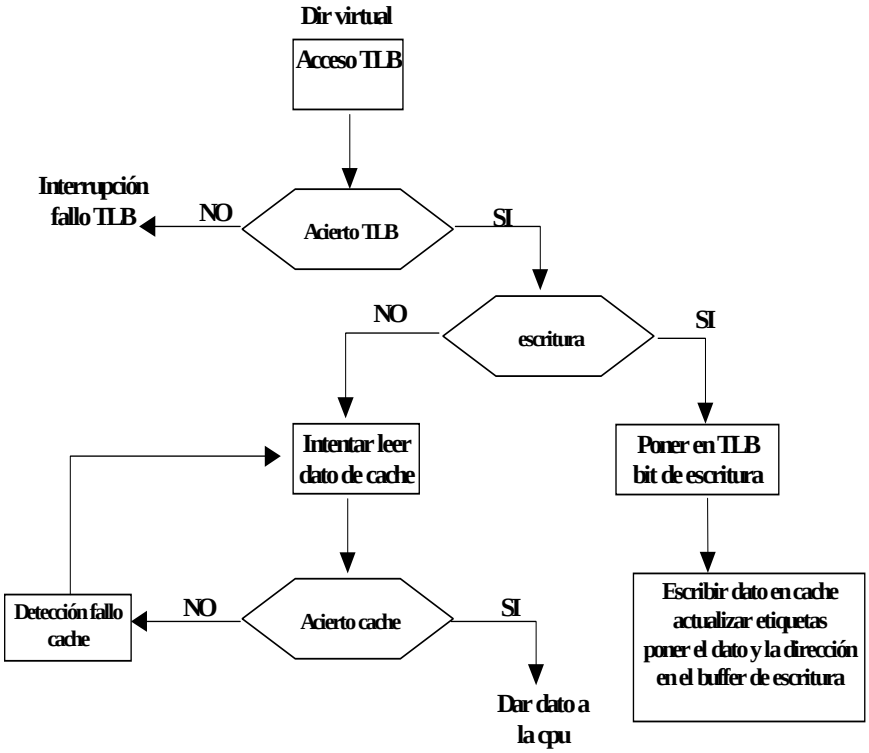
15.11.3 CACHE FÍSICA

Peterson pp436. Stalling pg91

La dirección física que direcciona la cache se obtiene a través del TLB. Su principal ventaja es que se puede mantener información en la memoria cache de diferentes procesos, puesto que ya trabajas con posiciones reales de memoria asignadas a diferentes trabajos. Su principal inconveniente que tiene que pasar antes la dirección por el TLB lo que retarda la lectura.



A continuación aparece un diagrama en el que se explica el tratamiento de lectura o escritura con el TLB y la cache de la DECstation 3100



15.11.4 CACHE VIRTUAL

Wlkinson pg91, pg 117 y123 Hwang y Hennesy pg423

Una cache puede utilizar la memoria virtual de dos formas diferentes,

* Direccionamiento virtual parcial
* Direccionamiento virtual total (prácticamente no se utiliza)

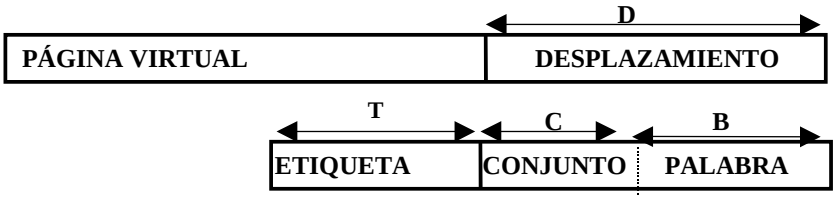
15.11.4.1 Direccionamiento virtual parcial (solapamiento)

El tiempo de acierto de la memoria cache es uno de los parámetros que más influyen en el rendimiento de un sistema computador ya que determina el tiempo de ciclo. Cuando se utiliza una cache física, se utilizan las direcciones físicas que ya han pasado por el mecanismo de traducción de la memoria virtual. Este paso de traducción se puede reducir si somos capaces de realizar simultáneamente la búsqueda del bloque en la cache y la traducción virtual. Se sabe que la dirección virtual se divide en dos partes: La página virtual que debe pasar por el mecanismo de traducción, y el desplazamiento utilizado directamente en la dirección física que con posterioridad se utiliza en la dirección de cache. Si se ponen restricciones al tamaño del desplazamiento y del número de página se puede realizar este acceso paralelo. Vamos a ver como sería el modo de operación. Se realizan en paralelo las siguiente operaciones:

- Mientras se traduce la dirección de la página a través de la tabla de páginas

- Se utiliza el desplazamiento de la dirección virtual para seleccionar el conjunto y la palabra de la dirección de cache

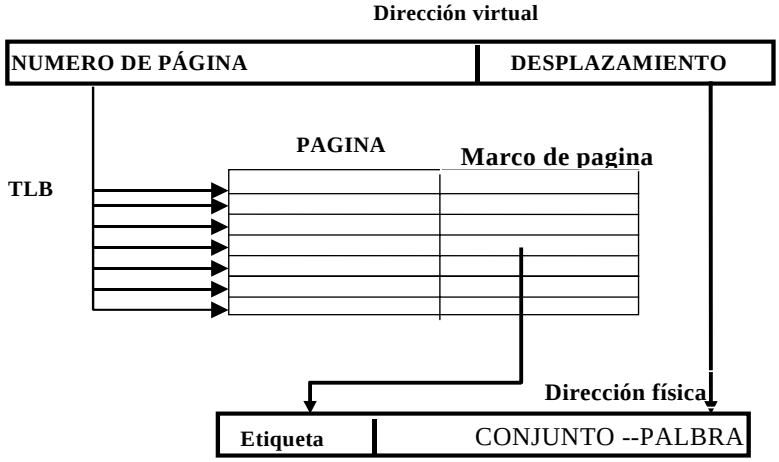
El principal efecto es que se ahorra tiempo



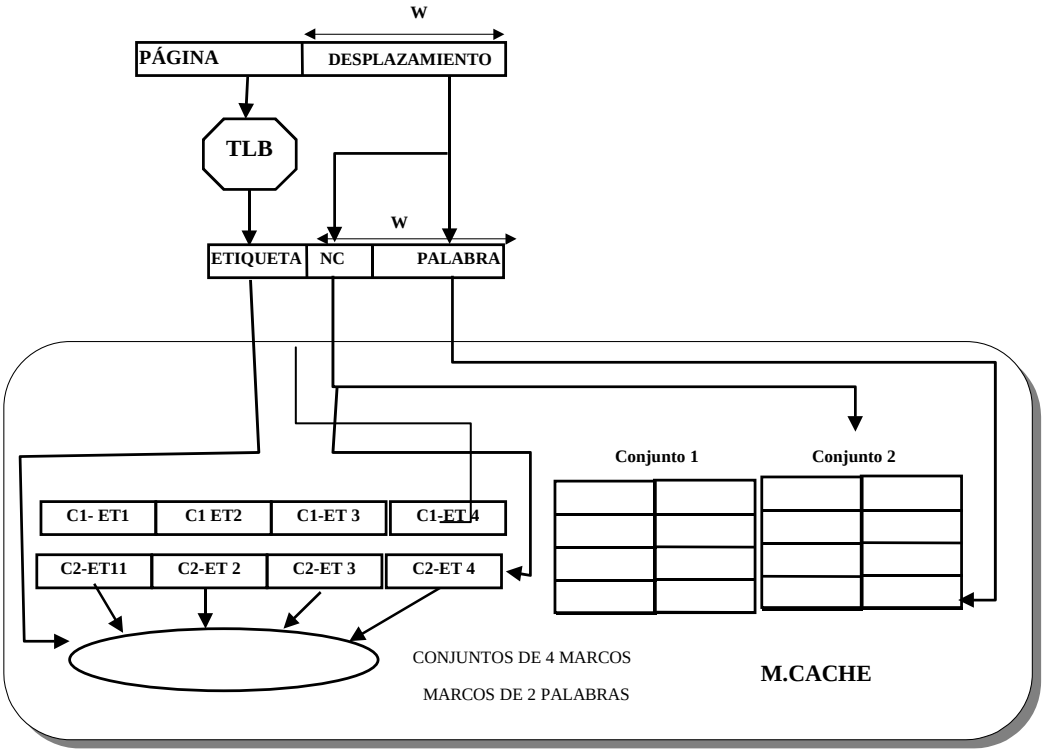
Vamos a explicarlo con más detalle. El desplazamiento de la dirección virtual, que no se modifica durante la traducción, se puede descomponer en dos campos un campo conjunto y un campo palabra. La dirección física que utiliza una cache asociativa por conjuntos se descompone en:

- Un nº de conjunto que selecciona el conjunto de bloques
- En etiqueta que realiza la búsqueda asociativa de los bloques
- La palabra que selecciona la palabra del bloque.

Con este esquema se consigue que mientras se realiza la traducción de la página, la memoria cache accede al conjunto y selecciona la palabra **(Selecciona la palabra de cada bloque).** Cuando se conoce la dirección real se comprueba la etiqueta. Si en la traducción se produce un acierto el dato seleccionado en la cache se acepta. En caso de fallo se deshecha el dato seleccionado y se lee el autentico.Para poder hacer todo lo que hemos explicado se tiene que dar una condición que D=C+B



La principal limitación de esta alternativa es que la cache no puede ser mayor que el tamaño de página virtual. Esto es una ventaja para las caches de 8kb del alphaAXP 21064 , el mínimo tamaño de página es de 8kb de manera que los 8 bits que indexan pueden tomarse de la parte física de la dirección. El Hennessy pone también como ejemplo el IBM 3033

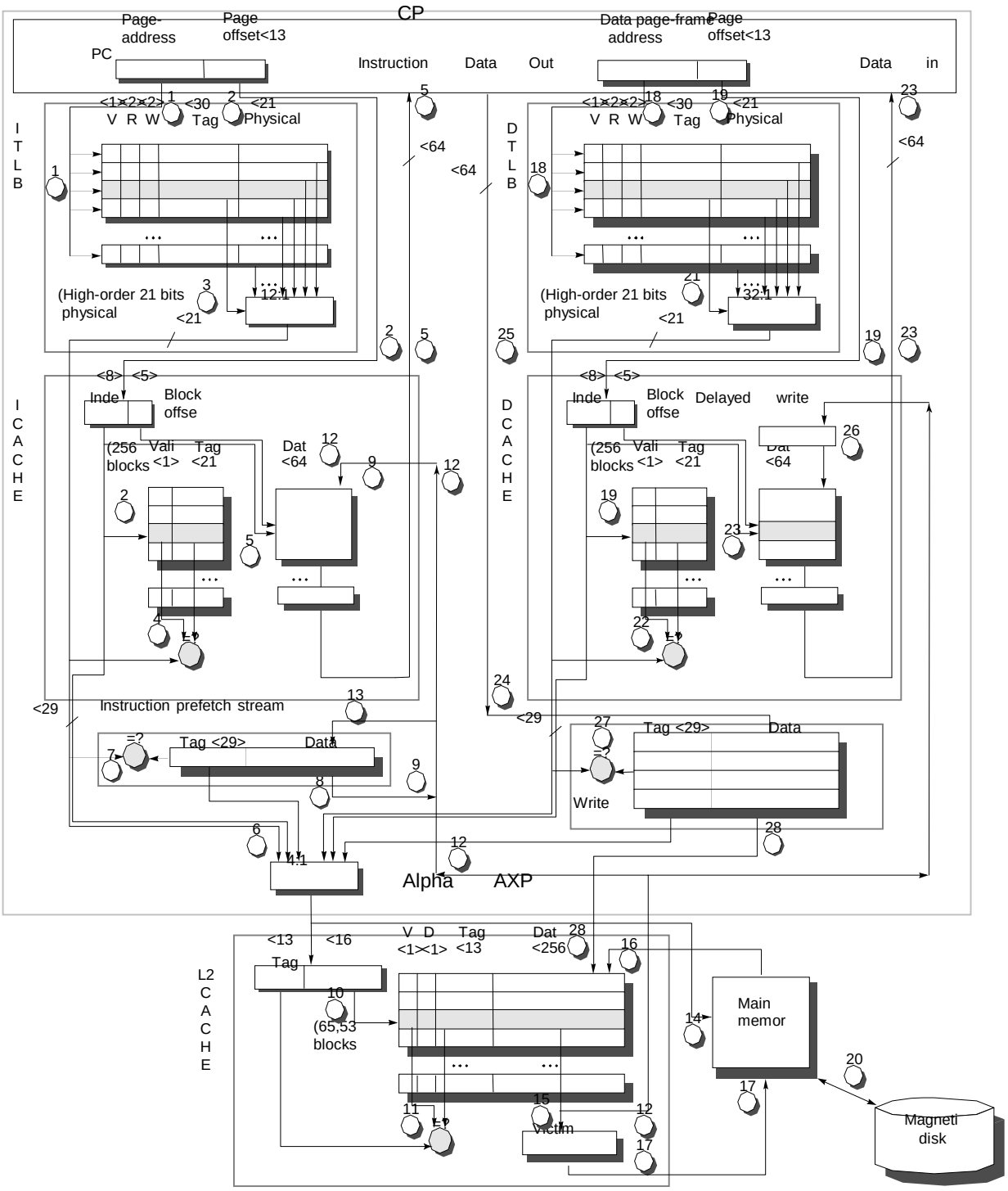


15.11.4.2 Direccionamiento Virtual Total

Este direccionamiento prácticamente no se utiliza. En el se suprime el paso de traducción, porque se utilizan las páginas proporcionadas por la dirección virtual como TAG de la dirección de cache. Lógicamente para estos casos los tamaños de dirección virtual y cache deben coincidir. El problema es que como pueden existir procesos diferentes con la misma dirección virtual, puede ocurrir que la misma dirección virtual tenga diferentes direcciones físicas. La solución es realizar un vaciado de la cache cada vez que se produce un cambio de contexto. Esto tiene el inconveniente de la perdida de tiempo que supone y la degradación del rendimiento. Otra solución es aumentar el tamaño del TAG de la dirección de cache añadiéndole un identificador de proceso (PID). Si el sistema operativo asigna este TAG al proceso no se necesita vaciar la cache cada vez que se carga otro proceso.

Problema de los sinónimos (o alias): Aparece cuando varios procesos comparten información. Puede existir el mismo dato, en diferentes bloques de cache, cada uno de ellos para un proceso diferente. Aparece el problemas de coherencia de cache, cuando un proceso actualiza un dato, y el resto de los procesos no se da cuenta. La solución es evitar que existan varias copias del mismo dato mediante la detección de los sinónimos cuando se produzca. Cuando la dirección virtual se traduce a dirección física a través del TLB es fácil evitar que existan sinónimos, con solo dar las misma dirección física a las diferentes direcciones virtuales, esto no es posible si la cache utiliza directamente la dirección virtual. La única solución es realizar la traducción física en paralelo al acceso a la cache.

15.11.5 JERARQUÍA DEL DEC ALPHA 21064 (HENNESSY)



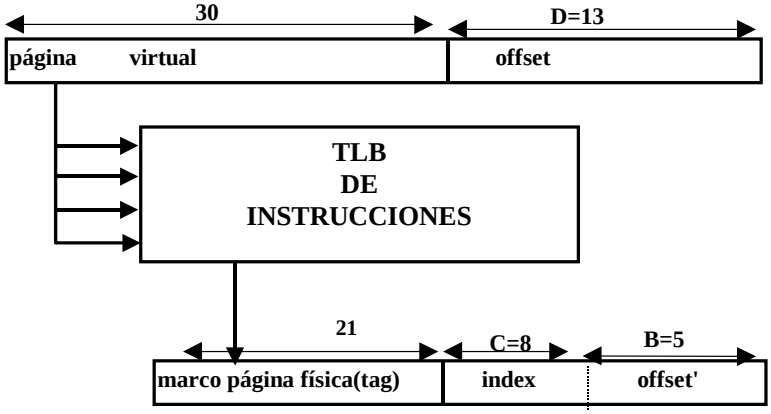
Vamos a empezar la descripción de lo que ocurre desde el principio. Cuando se enciende el sistema, el HW carga la cache de instrucciones desde una PROM externa. Esta iniciación permite a la cache de instrucciones omitir el bit válido porque siempre hay instrucciones válidas en la cache (aunque siendo validas pueden no interesar al programa que se está ejecutando en ese momento).

El PC se pone en el segmento Kseg de la memoria (bits más significativos a 10) de manera que las direcciones de instrucciones no se trasladan a través del TLB. Uno de los primeros pasos es actualizar el TLB de instrucciones con entradas de tabla de página válidas (PTE) para el proceso. El Kernel es el encargado de actualizar convenientemente los TLB. Una vez que el sistema operativo está preparado para ejecutar un proceso se pone el PC con el valor del segmento cero apropiado.

A continuación vamos a seguir la jerarquía de memoria en acción. Para ello estudiamos tres operaciones diferentes. La primera es la lectura de una instrucción que pone en movimiento la parte de la jerarquía que atañe a las instrucciones. A continuación veremos lo que ocurre con las lecturas de datos (operación de load) y las escrituras de datos (store).

LECTURA DE INSTRUCCIONES

La CPU proporciona direcciones de 43 bits que son direcciones virtuales. Tiene dos registros, por un lado el registro contador de programa PC que contiene direcciones de instrucciones y por otro lado un registro que contiene direcciones de datos. Como ya sabemos, para acceder al dato mientras que se traduce el marco de página, la dirección física y la virtual cumplen la siguiente condición

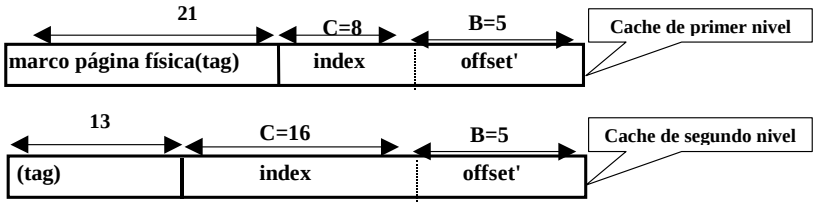


La dirección de página virtual proporcionada por el PC se lleva al TLB de instrucciones. Éste proporciona el marco de página física que a su vez utiliza la cache como TAG de la dirección. En paralelo los 8 bit de index que se obtienen del offset se envían a la cache de instrucciones, que los utiliza para realizar un acceso aleatorio y obtener la tag de la cache que se compara con el tag de dirección proporcionada por el TLB. Si hay acierto se lee la palabra de 8 bytes de la memoria cache de instrucciones y se lleva a la CPU. (En la pg 383 del Hennesy indica que todo este proceso lleva dos ciclos de reloj)

- **Fallos de lectura en la cache de instrucciones**

Si se producen fallos de cache de instrucciones se realizan simultáneamente dos acciones: por un lado se inicia una búsqueda de la instrucción en el *buffer de prebúsqueda de instrucciones* y por el otro lado se inicia un acceso a la *cache de 2º nivel.* Si la instrucción se encuentra en el buffer de prebúsqueda, los 8 bytes mas críticos se envían a la CPU inmediatamente, mientras que los 32 bytes del buffer se escriben en la cache de instrucciones y se cancela la búsqueda en la cache de segundo nivel, todo ello en un ciclo de reloj. Todas estas acciones se realizan en un ciclo de reloj.

Si la instrucción no está en el buffer de prebúsqueda, se continua la búsqueda en la cache de segundo nivel. El sistema DEC 3000 model 800 alpha AXP tiene 2Mb (65.536 bloques de 32 bytes) de manera que los 29 bits necesarios para identificar el bloque se dividen en 13 bits de tag y 16 de index.



- **Acierto en cache de segundo nivel**

Cuando se produce un acierto en la cache de segundo nivel se devuelve a la cache de primer nivel los 16 bytes críticos en 5 ciclos de reloj y los otros 16 en los siguientes 5 ciclos. La anchura del camino entre los dos niveles de caches es de16 bytes. Al mismo tiempo se realiza una petición del siguiente bloque de instrucciones, que se almacenan en el buffer de prebusqueda de instrucciones en los siguientes 10 ciclos. El buffer de instrucciones no utiliza la traslación del TLB para cargar los nuevos 32 bytes. Simplemente utiliza la dirección de pérdida y la incrementa asegurándose que se encuentra en la misma página. En el caso que esta nueva dirección sobrepase la página se suspende la operación de prebúsqueda.

- **Fallos en la cache de segundo nivel**

Si la instrucción tampoco está en la cache de segundo nivel, la dirección física se envía a la memoria principal para traer la información que se solicita. El promedio de transferencia de memoria a segundo nivel de cache es de 32 bytes en 36 ciclos después que el procesador hace la petición. La cache de segundo nivel carga 16 bytes al tiempo.

Como la cache de segundo nivel utiliza una política de **post escritura** un fallo de lectura podría provocar que la nueva información borrara información todavía no actualizada en la memoria. A este bloque de información, que se podría perder, se le llama bloque víctima; y para evitar esto la cache de segundo nivel incluye un buffer víctima El mecanismo de funcionamiento es el siguiente: el bloque víctima se carga en el buffer víctima. Los nuevos datos se cargan en la cache de segundo nivel tan pronto como llegan y por último los datos viejos de la víctima se escriben en la principal. Esta víctima sólo tiene un bloque por lo tanto si se produce un nuevo fallo, se produciría una parada hasta que se vaciara el buffer víctima.

- **Fallo de página**

Podría suceder que la información buscada no se encuentre en la memoria y que halla que traerla desde el disco. Como durante esta operación se pueden ejecutar millones de instrucciones, el sistema operativo normalmente cambia de proceso activo.

Falta por ver que ocurre si se produce un fallo de TLB. La gestión de este problema corresponde a la memoria virtual. Cuando se produce una perdida de TLB se invoca al software PAL (Privileged Architecture Library) que se encarga de actualizar el TLB. Cuando el código PAL se activa se deshabilitan la interrupciones y no se comprueba las violaciones de los accesos a memoria, para permitir que el TLB se cargue correctamente.

INSTRUCCIONES DE LOAD

Supongamos que la instrucción es de LOAD, es decir se trae información de memoria a la CPU. La forma de actuar es la siguiente. Se envía el marco de página virtual del dato al TLB de datos y simultáneamente se envían los 8 bits del index y los 5 de offset a la cache de datos.

- **Fallo de TLB de datos.**

Se puede deber a dos motivos, la TLB no contiene la información correcta o la información no está en memoria. Este último es el peor de los casos porque hay que traer la página desde la memoria secundaria. En el primero de los casos el fallo de TLB produce una llamada al código PAL que carga un PTE válido.

- **Acierto en la TLB de datos y acierto en la cache de datos**

Suponiendo que tenemos un dato valido en el TLB de datos se obtiene el marco de página física que se compara con la tag de la cache (obtenida a su vez con el offset de página). Si coinciden se envía 8 bytes a la CPU. Si se produce un fallo, se accede a la cache de segundo nivel que actuará de manera idéntica a un fallo de instrucción.

INSTRUCCIONES DE STORE

Supongamos que la instrucción es de STORE, es decir se envía información de la CPU a la memoria . De nuevo la parte de dirección correspondiente a la página virtual de datos se envía al TLB de datos y a la cache de datos, que comprueba las violaciones de protección y traduce las direcciones. La dirección física se envía a la cache de datos. La cache de datos usa escritura directa y por lo tanto los datos a escribir son enviados simultáneamente a la cache de datos y al buffer de escritura. La cache de datos utiliza la técnica de escritura segmentada, que consiste en comprobar la etiqueta de la escritura actual mientras que se escribe el dato del acierto anterior. Para ello se tiene *un buffer de escritura retrasada* en la cache de datos que guarda la información necesaria para realizar esta operación.

Si el acceso a cache es una acierto el dato se escribe en *este buffer de escritura retrasada.* **En caso de fallos** de escritura, el dato solo se envía al buffer de escritura. (Los caminos de datos para implementar estas operaciones no aparecen en la figura)

El buffer de escritura tiene 4 entradas, y cada una contiene un bloque de cache. Si este buffer está lleno, entonces la CPU debe esperar hasta que un bloque se escribe en el segundo nivel de cache. Si el buffer no esta lleno la CPU prosigue trabajando y la dirección del dato se presenta en el buffer de escritura. Éste comprueba si la palabra pertenece a algunos de los bloques que ya están en el buffer, con el objetivo de completar bloques antes de mandarlos al nivel inferior, y de esta manera optimizar el uso del ancho de banda de escritura entre el primer y el segundo nivel.

Todas las escrituras se escriben en el segundo nivel de cache a través del buffer de escritura directa. Como este segundo nivel utiliza post escritura no se puede segmentar. Cuando se produce un acierto de escritura, se escribe el dato solamente en este nivel . La escritura de un bloque de 32 bytes lleva 5 ciclos de reloj para comprobar la dirección y 10 para escribir el dato. Una escritura de 16 o menos bytes toma 5 de comprobación de dirección y 5 de escritura. En ambos caso se marcan los bloques como dirty.

Cuando en la escritura se produce un fallo, se debe recordar que utiliza la política de asignación de marco. Se comprueba el bit dirty del bloque víctima. Si lo tiene activo se envía al buffer víctima, se trae el bloque nuevo desde memoria principal y por ultimo se envía el contenido del buffer víctima a la memoria principal. En caso que el bloque víctima no tuviera el bit dirty activo se sobreescribe el bloque, puesto que hay coherencia de la información entre los dos niveles.

¡Error! Referencia de hipervínculo no válida.
¡Error! Referencia de hipervínculo no válida.
¡Error! Referencia de hipervínculo no válida.
¡Error! Referencia de hipervínculo no válida.
¡Error! Referencia de hipervínculo no válida.

16 BUSES DEL SISTEMA

16.1 DEFINICIONES

Desde el punto de vista del alto nivel un computador consta de los módulos de memoria, procesador compuesto de camino de datos, unidad de control y unidades de entrada/Salida. Estos componentes se conectan para conseguir la función básica de un computador que es ejecutar programas. Por lo tanto, para describir un sistema computador hay que describir El comportamiento de cada uno de los módulos , La estructura de interconexión y el control necesario para manejarlo.

Esta visión de alto nivel es importante, por un lado debido a su capacidad explicativa de cara a la comprensión del computador y por otro para la evaluación de los rendimientos porque ayuda a ver donde se encuentran los cuellos de botella.

Un bus es un camino de comunicación que conecta dos o más dispositivos. Este camino se caracteriza por ser un medio de transmisión compartido. Si existen muchos dispositivos conectados al bus, la señal que se transmite está disponible para todos ellos. Dado que la línea se comparte, si dos dispositivos transmiten simultáneamente se pierde la información porque se produce una colisión**,** por lo tanto sólo un dispositivo puede transmitir cada vez.

Sus principales ventajas son su versatilidad y bajo coste; y su principal desventaja es que se comporta como un cuello de botella de comunicación, limitando la máxima productividad de la entrada/salida. La máxima velocidad del bus está determinada por factores físicos tales como su longitud y el número de dispositivos conectados. La red de conexión de un computador debe dar cobertura a los siguientes tipos de transferencias:

- Memoria $\rightarrow$ CPU , CPU $\rightarrow$ memoria
- E/S $\rightarrow$ CPU, CPU $\rightarrow$ E/S
- Memoria $\rightarrow$ E/S , E/S $\rightarrow$ memoria. (DMA)

Se llama bus de sistema al que conecta los principales componentes del sistema, es decir a la unidad central de proceso, a la memoria, y a los módulos de entrada/salida. Los elementos que se deben determinar para diseñar un bus son

- Paralelismo de la línea de datos
- Temporización o sincronización
- Arbitraje de acceso al bus
- Anchura de banda
- Tipos de transferencias de datos

16.1.1 ESTRUCTURA DE UN BUS

Un bus se compone de entre 50 y 100 líneas separadas, cada una de las cuales tiene una función. Estas líneas se pueden clasificar en tres grupos funcionales que son el bus de datos, el de direcciones y el de control.

El bus de datos proporciona el camino para mover datos entre diferentes módulos del sistema. Suele tener entre 8, 16 o 32 líneas, y a este número de líneas se le llama anchura de bus y determina el número de bits que se pueden mover simultáneamente y es determinante para estudiar el rendimiento del sistema. Cada línea sólo transmite un dato al tiempo

El bus de direcciones determina la dirección de la fuente o el destino de los datos. Su anchura fija la máxima capacidad de la memoria del sistema. También direcciona puertos I/O (como veremos en el tema correspondiente el bit de mayor peso de la dirección es el que indica si el acceso es a ES o si es a memoria.). El bus de control tiene varias funciones, que indican el tipo de operación a realizar :

- Lectura de memoria, cuando el dato de la dirección se coloca en el bus
- Escritura de memoria, cuando el dato del bus se escribe en la posición direccionada
- Escritura de E/S cuando el dato se transfiere a través del puerto de e/s direccionado

- Lectura de E/S cuando dato del puerto de E/S direccionado se coloca en el bus

También se utiliza este bus para las siguientes acciones:

- Consulta del estado de un dispositivo
- Reconocimiento de la transferencia, es decir para comprobar que el dato ha sido aceptado o colocado en el bus
- Petición de control del bus
- Cesión de bus
- Petición de interrupción
- Interrupción reconocida
- Reloj
- Inicio (reset)
- Controlar la sincronización entre los elemento al principio y al final de cada transferencia.
- Arbitraje: determinar cual de los elementos conectados al bus puede usarlo (resolución de conflictos de acceso).

Las funciones del bus son **s**oportar la información a transmitir, realizar el tratamiento electrónico y físico de las señales, evitar los ruidos garantizar la correcta comunicación entre los elementos que comparten el bus, realizar el arbitraje y la sincronización

16.1.2 MODO DE OPERACIÓN

Los elementos implicados en una transferencia son el master que inicia y dirige la transferencia (la CPU y DMA) y el slave que obedece y accede a las peticiones del master (memoria y Interfaz de entrada/salida). Existen dos tipos de transferencia, la escritura en la que el master envía un dato al slave; y la lectura en la que el slave envía un dato al master. Se llama ciclo de bus a cualquier transferencia completa, ya sea de lectura o de escritura. El modo de operación de una transferencia si el módulo desea enviar información a otro es el siguiente:

1.- Obtiene el uso del bus.

2.- Transfiere los datos por el bus.

Y para un modulo que desea obtener información de otro el modo de operación es:

1. Obtiene el uso del bus.
2. Transfiere la petición de información con las líneas de dirección y control adecuadas.
3. Espera a que le envíen los datos

Esta espera se debe al tiempo que necesita el slave para dejar el dato correcto en el bus. Por ejemplo en el caso de una memoria se debe a los inevitables tiempos de acceso. Estos tiempos de espera hacen que el rendimiento del bus se degrade puesto que no puede ser utilizado por ningún otro módulo.

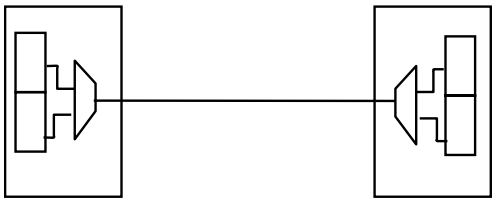
El control de transferencia se realiza mediante el arbitraje para evitar los conflictos de acceso al bus (es decir varios accesos simultáneos que tendrían como consecuencia la perdida de información que se transmite) y la sincronización del master – slave.

Los parámetros que caracterizan al bus son

- Capacidad de conexión: nº máximo de módulos que se pueden conectar al bus
- Protocolo de arbitraje. - resolución de conflictos de acceso
- Protocolo de bus.- sincronización master slave
- Ancho de bus.- Nº total de líneas
- Ancho de banda.- Nº de bits de información que es capaz de enviar por unidad de tiempo
- Ancho de datos.- Nº de líneas de datos.

16.2 PARALELISMO DEL BUS

En función dela cantidad de información que envía y como realiza la operación los buses se pueden clasificar en paralelos, multiplexados y serie. Un bus paralelo es aquel en el que el ancho de palabra coincide con el ancho de la información a transmitir. También se le llama dedicado. El bus multiplexado surge del hecho de tener que ahorrar patillas en los circuitos integrados. Se caracteriza por transmitir tipos de información diferentes, en tiempos diferentes por las mismas líneas. Esto provoca la necesidad de señales adicionales para indicar el tipo de información que se envía. Suele ser necesario añadir un demultiplexor para realizar la demultiplexación temporal y un multiplexor para realizar la multiplexación temporal:



Por último el bus serie es un caso extremo de multiplexación. La información se envía bit a bit. Se utilizan registros de desplazamiento para implementarlo.

16.3 PROTOCOLOS DE ARBITRAJE

Su objetivo es garantizar el acceso al bus sin conflictos cuando existen varios master que pueden solicitarlo, definiendo master como todo modulo capaz de solicitar un acceso al bus. Ejemplos típicos de master son el procesador y los controladores de DMA, procesador de E/S, Coprocesadores matemáticos, y los sistemas multiprocesador. Existen dos clases de protocolos de arbitraje del bus los centralizados y los distribuidos. Los protocolos centralizados son los que tienen un arbitro de bus o master principal que controla el acceso al bus. A su vez se pueden clasificar en:

- Daisy chain de dos hilos
- De tres hilos
- De cuatro hilos

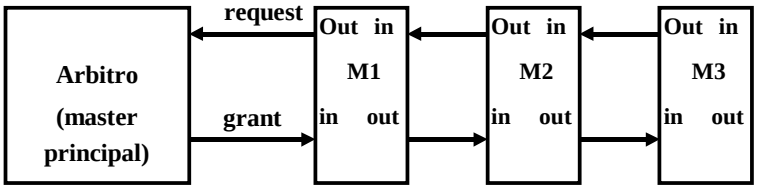
En los protocolos distribuidos el control de acceso al bus se lleva a cabo entre todos los masters de una forma cooperante. Se clasifican en:

- Protocolos de líneas de identificación
- Protocolos de códigos de identificación

16.3.1 PROTOCOLOS CENTRALIZADOS

16.3.1.1 Daisy chain de dos hilos

Tienen dos líneas de arbitraje común a todos los masters: la bus request de petición de bus; y la bus grant de concesión del bus.



El modo de operación es el siguiente

- M_i activa petición request y la petición se propaga hasta el arbitro a través del resto de los master

- El arbitro concede el bus activando grant. La concesión grant se propaga por los elementos no peticionarios. Si un master recibe grant y no pidió el bus, propaga grant al siguiente
- M_i toma el control cuando le llega la concesión (grant). Si un master recibe grant y tiene una petición pendiente toma el control del bus
- Se sabe que el control del bus está asignado cuando la señal de petición (request) de bus esta activa y la de concesión (grant) de bus también esta activa.
- Para que la señal de concesión (grant) se desactive primero se tiene que desactivar la señal de petición que llega al master principal.

En este protocolo la prioridad la determina el orden en que se conectan los master al bus.

ν Situación conflictiva 1

M2 pide el bus

M2 detecta grant in y toma el control del bus

M1 pide el bus y detecta grant a 1 luego también toma el control del bus

solución: grant funciona por flanco, no por nivel.

ν Situación conflictiva 2

- M2 controla el bus
- Cuando M2 deja de controlar el bus su señal de request se desactiva pero la señal grant tarda en desactivarse desde el arbitro y en este espacio de tiempo pueden ocurrir los dos siguientes sucesos:

a) M1 pide el control

b) M3 pide el control. Como M2 tiene grant todavía activada se propaga grant a M3, que toma el control. Al mismo tiempo M1 recibe grant_in del arbitro y M1 toma el control. Es como si hubiesen dos grant independientes, la que propaga M2 y la que genera nuevamente el master principal.

Solución un master M solo puede propagar grant al terminar de usar el bus si request_in se activó antes de que M desactivase su petición local. Con esto se evita que el master principal genere una nueva señal de concesión que cree un conflicto. Esto provoca que la concesión del bus sea secuencial. (round robin)

La principal ventaja de este protocolo es que sólo necesita dos líneas de arbitraje. A cambio tiene varias desventajas. A saber:

* Los retardos en la propagación de grant y request
* Múltiples situaciones conflictivas.
* No puede solicitar el control del bus un módulo mientras otro tiene el control puesto que ambas señales están activadas

Un ejemplo de este protocolo es el i8086

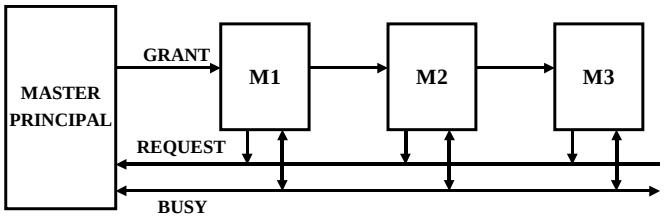
16.3.1.2 Protocolo de tres hilos

En este protocolo la señal de petición ya no se transmite a través de los módulos, por lo tanto se transmite más rápidamente. En este caso necesitamos tres líneas de arbitraje:

Bus request.- línea de petición de bus

Bus grant.- línea de concesión de bus

Bus busy.- línea de bus ocupado



Modo de operación:

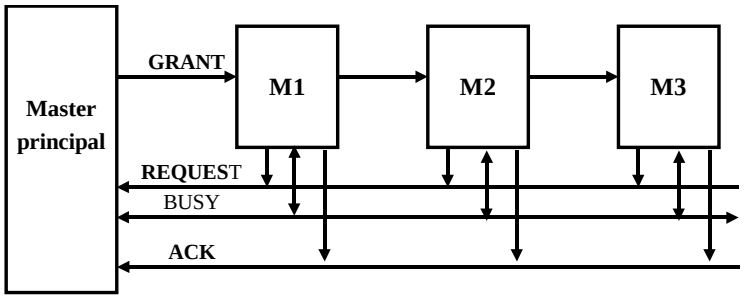
1. Cuándo un módulo toma el control se activa la señal ocupado (busy)
2. Un módulo pide el bus activando request
3. El arbitro activa grant cuando detecta request activado y busy desactivado
4. Si un módulo recibe grant y no ha realizado la petición lo transmite al siguiente
5. Un módulo toma control cuando se cumplen tres condiciones
    - La petición propia esta activa
    - La señal busy está inactiva
    - Detecta el flanco de subida de grant

El módulo desactiva request cuando ve llegar el flanco de grant y activa busy. grant se desactiva en cuanto se activa bus ocupado. Su ventaja frente al protocolo de dos hilos es que cuando un master tiene el control la única línea activa es la de ocupado, esto permite que mientras el bus esté ocupado se pueda solicitar el control del bus activando la señal petición. El control del bus no se concede hasta que se desactiva ocupado. Su principal inconveniente es que al cumplir la línea ocupado dos misiones diferentes, indicar que el bus está ocupado e indicar que módulo tiene el control, no se puede conceder el bus mientras un módulo tiene un control. Una solución es añadir una línea más.

16.3.1.3 Protocolo de cuatro hilos

Permite solapar la transferencia de ciclo actual con el arbitraje del ciclo siguiente. Las líneas de arbitraje necesarias son:

- Bus request.- línea de petición de bus
- Bus grant.- línea de concesión de bus
- Bus busy línea de bus ocupado
- Bus ack.- línea de confirmación. Esta línea la activa el master que solicitó el bus, en respuesta a bus grant, cuando el bus está ocupado**.** Cuando está activada el arbitro queda inhibido



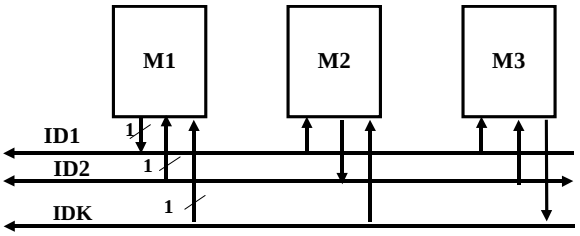
16.3.2 PROTOCOLOS DISTRIBUIDOS

Protocolo de líneas de identificación

Cuando un master quiere tomar el control del bus, activa su línea de identificación. Cada línea de identificación tiene asignada una prioridad:

Prioridad (ID1) <Prioridad (ID2)<…<Prioridad(Idk)

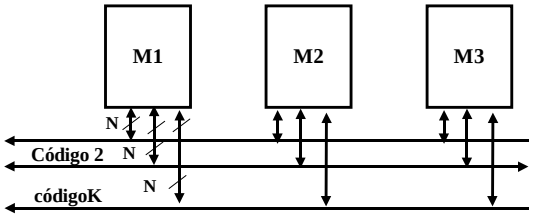
Si varios masters activan simultáneamente sus líneas de identificación, gana el de mayor prioridad. Para conseguir esto los módulos leen el resto de las líneas y comprueban si su petición tiene mayor prioridad que el resto de las peticiones. Un funcionamiento alternativo sería que las prioridades pueden ser variables.



Como principal desventaja está que el número de dispositivos que se pueden conectar al bus está limitado por el número de líneas de arbitraje. Ejemplos: Prioridad fija vax sbi, scsi, Prioridad variable: dec70000/10000 axp, alpha server 8000

Protocolo de códigos de identificación

Cada master tiene un código de identificación de N bits, luego se permite un máximo de de 2^N masters. Existen N líneas de arbitraje. Cuando un master quiere tomar el control del bus pone su código en las N líneas de arbitraje. Los módulos leen el resto de los códigos y comprueban si su petición tiene mayor prioridad que el resto de las peticiones. Si varios masters compiten por el bus gana el de mayor identificador.



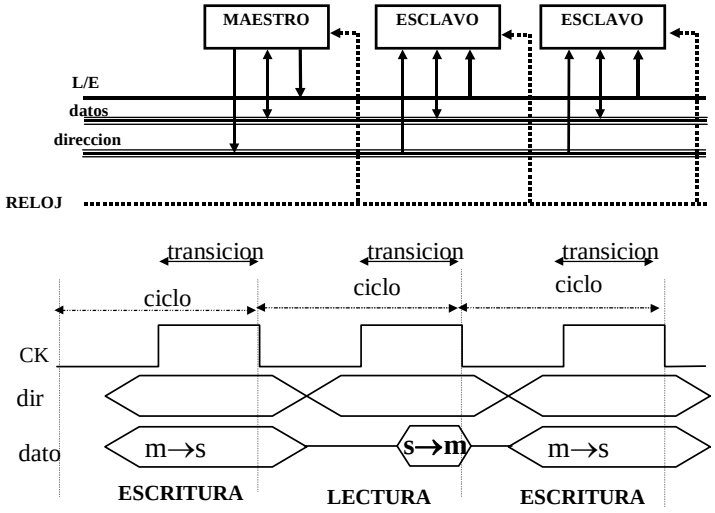
Ejemplos: de este protocolo son los usados por el MULTIBUS II y el FUTURE BUS+

16.4 PROTOCOLOS DE BUS (SINCRONIZACIÓN)

Se llama protocolo de bus, a la forma de sincronizar los elementos implicados en una transición, indicando el conjunto de líneas que lo implementan. Existen cuatro clases: Síncrono, Asíncrono, Semisíncrono y de Ciclo partido.

16.4.1 PROTOCOLO SINCRONO

Es aquel en el que la transferencia está gobernada por una única señal de reloj (CK). Esta señal es compartida por todos los elementos conectados al bus. Los flancos de subida y bajada determinan el comienzo y final de la transición, de manera que cada transferencia se realiza en un ciclo de CK. En la siguiente figura aparece un esquema de protocolo síncrono con la señal de reloj punteada.



- Operación de escritura
  - El dato lo pone el maestro
  - Se debe dar la DIR y DATO un tiempo de setup antes de que suba CK.
  - Debe permanecer DIR y DATO un tiempo de hold después que baje CK.
- Operación de lectura:
  - El Maestro pone la dirección.
  - El dato lo pone el esclavo
  - El DATO aparece en el Bus con retardo debido al tiempo que se tarda en acceder al dato (Tiempo de acceso)
  - La señal de CK debe estar en alta el tiempo necesario para que se produzca la transferencia.

Tiempo de descodificación es el tiempo necesario para que el esclavo descodifique la dirección. Tiempo de estabilización (set Up), antes de aplicar el flanco de subida las señales deben estabilizarse y deben permanecer estables durante un intervalo de tiempo para asegurar su correcto almacenamiento. Tiempo de permanencia (hold). Después de aplicar el flanco de bajada, las señales deben permanecer estables durante un intervalo de tiempo para asegurar su correcto almacenamiento.

Tiempo de desplazamiento relativo de las señales. A este tiempo también se le llama skew. Supongamos dos señales que parten de un emisor al mismo tiempo. Estas señales pueden llegar en instantes de tiempo diferentes a su destino debido a que atraviesan un número diferente de niveles de puertas o a que atraviesan circuitos con diferente tecnología. Las principales ventajas del protocolo síncrono son:

- Sencillez de diseño
- Solo necesita una señal para realizar la sincronización
- Mayor velocidad, sobre todo en la transferencia de grandes bloques de datos

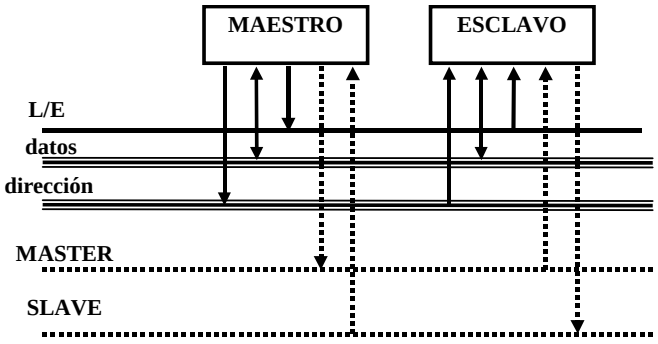
Su principal inconveniente es su inflexibilidad en lo que a tiempos de acceso se refiere, porque la velocidad la marca el dispositivo más lento. Además no se tiene información sobre si se ha realizado bien la transferencia.

16.4.2 PROTOCOLO ASINCRONO

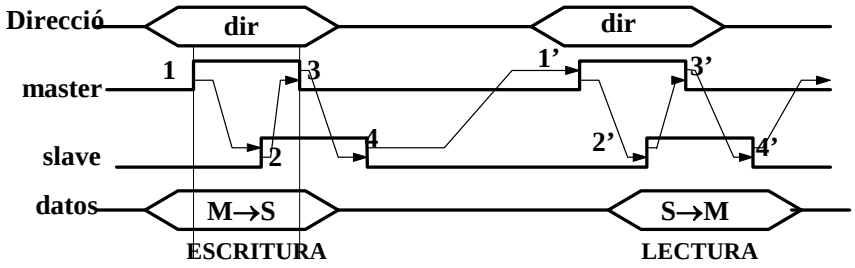
Se caracteriza por no usar una señal de CK. La forma de sincronizar el intercambio de información es mediante un conjunto de señales que se intercambian el maestro y el esclavo para indicar

- Cuando comienza la transferencia
- Como se desarrolla
- Cuando acaba la transferencia

A continuación vemos un esquema de bus asíncrono que utiliza las señales master y slave de sincronización:



Vamos a ver a continuación como es el intercambio de señales:



Pasos de escritura:

1.- M a S: Pongo un DATO y una DIRECCIÓN en el Bus

2.- S a M: Lo he cogido

3.- M a S: Veo que lo has cogido

4.- S a M: Veo que lo has visto.

Tras este último paso el bus queda libre para la siguiente transferencia. Durante toda la transferencia el bus está ocupado.

Pasos de Lectura

1'.- M a S: Quiero el DATO de la DIR.

2'.- S a M: Ya te he puesto el DATO en el Bus.

3'.- M a S: Cojo el DATO.

4'.- S a M: Veo que lo has cogido.

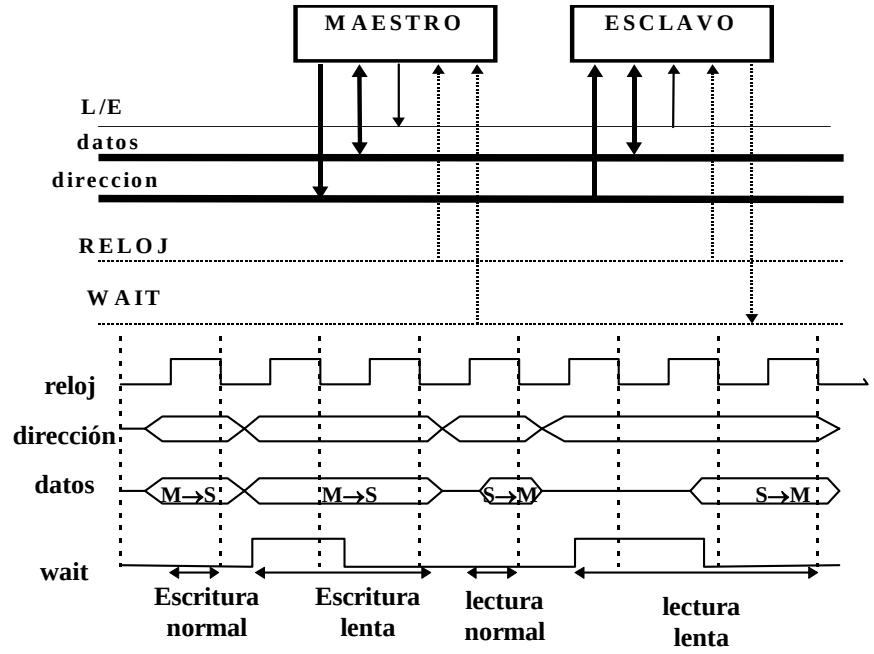
El tiempo entre 1' y 2' es el tiempo que se tarda en acceder al dato en el esclavo más el tiempo que se tarda en colocarlo en el bus. Como ya indicamos con anterioridad, durante este tiempo el bus está ocupado pero no contiene información útil. Un objetivo del resto de los protocolos que estudiemos es eliminar este tiempo de espera para poder aumentar el rendimiento del bus.

A este protocolo asíncrono que hemos visto se le llama protocolo totalmente interbloqueado porque a cada flanco del master le sigue un flanco del slave. Sus ventajas son su fiabilidad puesto que la recepción del dato siempre se confirma y la facilidad para conectar elementos de diferentes velocidades. Su principal inconveniente es ser un protocolos más lentos que los síncronos (a igualdad de dispositivos) debido precisamente al intercambio de señales. Ejemplos típicos de este protocolo son UNIBUS PDP11 , MC68000,10,20,30, Bus VME y el FUTUREBUS+

16.4.3 PROTOCOLO SEMISINCRONO

Su objetivo es intentar adaptar los buses síncronos a módulos de muy diversas velocidades. las transferencias se rigen por una única señal de reloj. existe una señal wait que puede activar cualquier slave. Su modo de funcionamiento es el siguiente. Con los dispositivos rápidos opera como un bus síncrono. Con los dispositivos lentos actúa de la siguiente manera:

- Activan la señal wait que congela la actuación del master. Con esta señal activada una transferencia puede ocupar varios ciclos de reloj
- La señal wait se debe activar antes de que la señal de reloj llegue al master, en caso contrario no tiene efecto
- La operación se finaliza con la llegada de la siguiente señal de reloj una vez que wait se ha desactivado.



Ejemplos de este protocolo son**:** i80x86, MC68040, multibusII, bus PCI, y el dec7000/10000 AXP.

16.4.4 PROTOCOLO DE CICLO PARTIDO:

Su objetivo consiste en eliminar la inactividad del bus en las fases de lectura. Con ello se mejora el rendimiento del bus en las operaciones de lectura. En definitiva lo que intenta es aunar las ventajas de los síncronos y de los asíncronos. Su modo de operación es el siguiente:

- El tiempo del bus se divide en una serie de fracciones llamadas ranuras, de manera cada ranura permite enviar un mensaje. La duración de la ranura viene fijada por las características de transmisión del bus, no por el tiempo de respuesta de los dispositivos.
- Las operaciones de lectura se dividen en dos mensajes: petición e inicio de transferencia y fin de transferencia.
- Cada mensaje sólo ocupa una ranura.
- La transferencia la inicia un maestro y la finaliza un esclavo.
- Los esclavos deben tener la capacidad de solicitar y emplear ranuras del bus.
- El ciclo partido se puede aplicar a transferencias sincronas y asíncronas.

LECTURAS SINCRONAS

ranura	ranura	ranura	ranura
Master peticion de lectura pone la direccion	Bus libre	Bus libre	Esclavo solicita la ranura y pone el dato

el maestro dispone de un temporizador de espera. Pasado un tiempo sin contestación genera una señal de error.

Este protocolo tiene varios inconvenientes. La lógica de acceso al bus es más compleja, luego crece y se encarece el hw. El tiempo para completar una transferencia crece porque hay que conseguir dos veces el bus. Protocolos mas caros y difíciles de implementar debido a la necesidad de seguir la pista a la otra parte de la comunicación.

Su principal ventaja es que al liberar el bus se permite que otro peticionario lo utilice luego mejora la anchura de banda efectiva del bus si la memoria es lo suficientemente sofisticada para manejar múltiples transacciones solapadas.

16.5 CLASIFICACIÓN DE BUSES

Pedro de Miguel califica los buses en 5 tipos diferentes:

- Interno de una pastilla. A este bus no tiene acceso el usuario. Ni los fabricantes suministran información, ni los usuarios la necesitan.
- Interconexión de pastillas de un circuito impreso. Es un bus restringido a la placa, tiene una longitud de decenas cm. No existen problemas de velocidad salvo en logias rápidas. Suelen ser síncronos, con solo un maestro.
- Panel posterior. Sirve para la interconexión de placas. Su longitud tiene aproximadamente 1 metro. El Número de señales ≅ 100. consigue una abstracción de las peculiaridades de las pastillas que forman la tarjeta.
- De sistema. Conceptualmente es idéntica a las anteriores, solo que son más largos con una longitud aproximada de 10 metros. Hay que tratarlo como una línea de transmisión. Debido a la longitud necesita repetidores (Buffers).
- E/S paralela. Conexión de perifericos en paralelo mediante una norma preestablecida diferente a la del bus (SCSI)
- Conexión serie de periféricos. Cubren las distancias más largas y sus velocidades de transmisión más lentas. Se usa en la conexión de terminales y periféricos de baja velocidad: redes locales. Un ejemplo es el RS -232-C.

Petterson-Hennesy realizan la siguiente clasificación:

- Bus procesador-memoria
- Bus de plano posterior
- Bus de entrada/salida

Los buses procesador - memoria son cortos y de alta velocidad y están adaptados al sistema de memoria para maximizar la anchura de banda memoria procesador. Su diseño es exclusivo y al diseñarse se conocen perfectamente todos los dispositivos que van a conectar

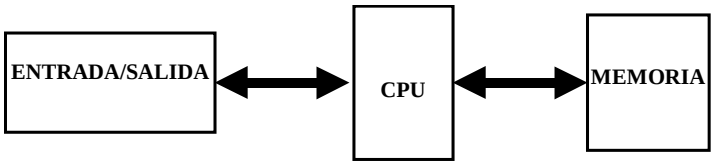
Los buses de plano posterior se diseñan para permitir que coexistan en un solo bus la memoria, el procesador y los dispositivos de entrada salida. Generalmente necesitan lógica adicional para conectarse a un bus de entrada salida o a un dispositivo, esta logica se añade mediante controladores. En los Macintosh el bus que conecta el procesador y la memoria es un bus de plano paralelo llamado NUBUS. A este tipo de bus se le puede conectar directamente un bus SCSI de entrada/salida. Son buses estándar

Los buses de entrada/salida son largos, con muchos tipos de dispositivos conectados y no tienen una interfaz directa con la memoria. Se conectan a ésta mediante buses de plano posterior o mediante buses de procesador memoria. Su interfaz es sencilla por lo que los dispositivos necesitan pocos elementos electrónicos para conectarse a él. Son buses estándar .

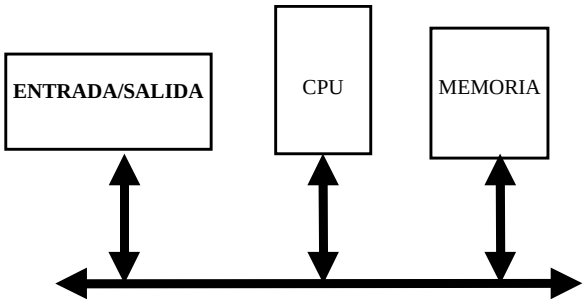
Los buses generalmente se conectan entre sí mediante adaptadores. Tanto los buses de plano -paralelos como los buses de entrada/salida se diseñan desconociendo los dispositivos que se van a conectar. Los sistemas expandibles de alto rendimiento utilizan los tres tipos de buses. Ej multiprocesadores IBM RS/6000 y Silicon Graphic. Características de esta organización jerárquica son que el bus procesador- memoria es mucho más rápido que uno de plano posterior y la jerarquía aprovecha esta velocidad. Por otro lado, el sistema de entrada salida se expande con facilidad conectando muchos controladores o buses al plano posterior y esta expansión no afecta al rendimiento del bus de procesador-memoria.

16.6 ESTRUCTURA DE LOS BUSES DE SISTEMA

hamacher simplificando habla de 2 estructuras muy básicas de Bus. La de dos buses típica de una organización de entrada -salida aislada de memoria, que es poco flexible y presenta dificultades para añadir nuevos módulos.



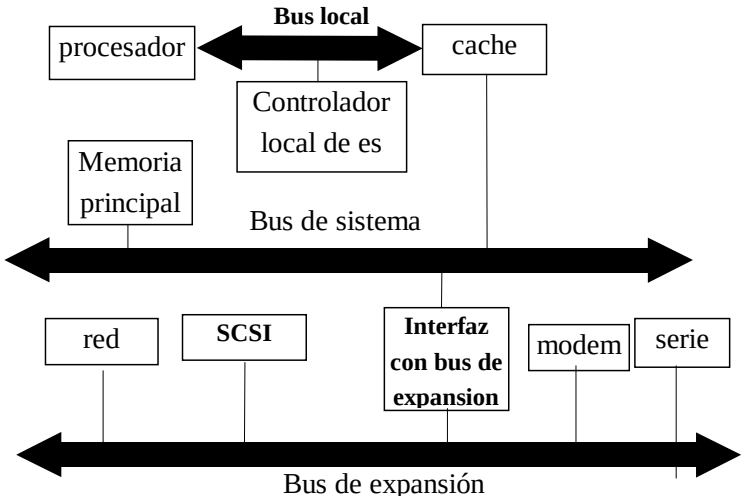
y la de un bus:



Este es un bus de bajo coste que tiene gran flexibilidad para conectar nuevos módulos, una velocidad de operación baja y lo suelen llevar máquinas pequeñas

Estas dos estructuras de bus son demasiado generales y se dan pocas veces. La razón es que dan lugar a importantes problemas (Stalling) Por ejemplo, si se conecta un gran número de dispositivos al bus, las prestaciones disminuyen debido al mayor retardo de propagación. Este retardo de propagación es importante porque es el que determina el tiempo necesario para coordinar el uso del bus. Además si el control pasa frecuentemente de un dispositivo a otro, empeoran notablemente las prestaciones y el bus se convierte en un cuello de botella. La solución es utilizar varios buses organizados jerárquicamente. A esta estructura se la llama estructura multibus y su objetivo es acelerar el sistema

16.6.1 MULTIBUS TRADICIONAL



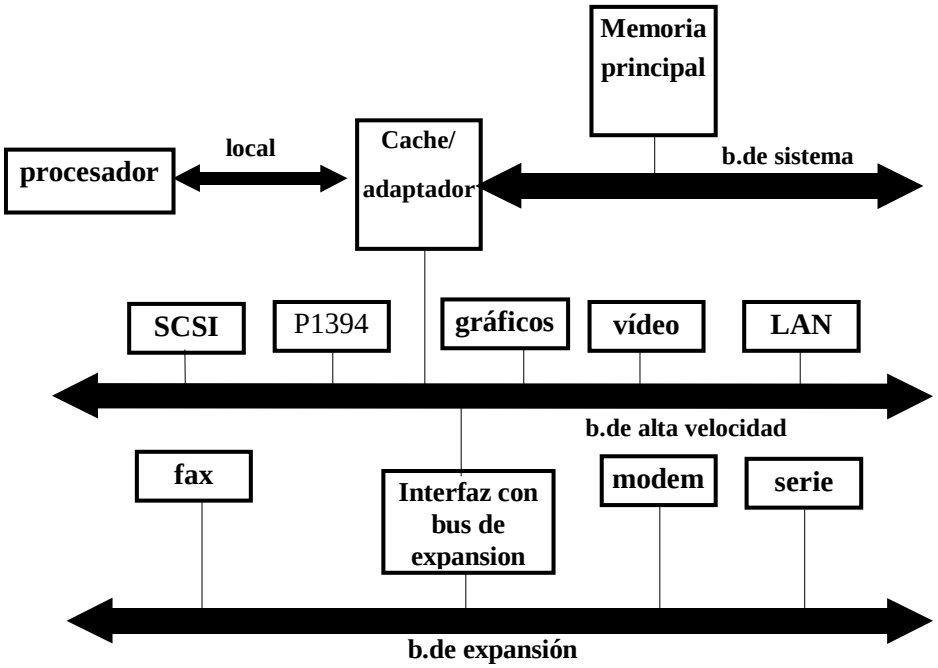
Los componentes de esta jerarquía son los siguientes: un bus local que conecta el procesador a la memoria cache. La memoria cache se conecta a través de su controlador con el bus local y el bus de sistema. El uso de cache alivia los accesos del procesador a la memoria principal. Otra componente es un bus de sistema que conecta la memoria principal y al procesador a través de la memoria cache. Como ventaja tiene que la transferencia de entrada salida no interfiere en la actividad del procesador. Se podrían conectar los controladores de E/S directamente al bus de sistema pero esta opción no es eficiente y degrada el rendimiento.

Por último en la jerarquía aparece un bus de expansión al que se conectan los controladores de entrada/salida. La interfaz del bus de expansión con el bus de sistema controla las transferencias entre los

dispositivos de entrada /salida y el bus de sistema. De esta forma se conecta gran variedad de dispositivos al sistema al tiempo que se aísla la memoria y el procesador del trafico entre la memoria y los dispositivos de entrada/salida

Esta jerarquía tiene el inconveniente de que a medida que mejoran las prestaciones de los dispositivos de entrada/salida se degrada la eficiencia de esta estructura MULTIBUS. La solución está en diseñar estructuras más complejas utilizando buses de alta velocidad, como se ve a continuación.

16.6.2 ARQUITECTURA DE ALTAS PRESTACIONES:



Esta jerarquía añade un cuarto bus a la estructura tradicional. Sigue teniendo el bus local que conecta el procesador y la memoria cache. Ésta conecta a través de un adaptador el bus local con el bus de sistema y el de alta velocidad. Al bus de alta velocidad se conectan todos los periféricos de altas prestaciones como las SCSI entrada salida paralela y P1394 (bus diseñado para dispositivos de e/s de alta velocidad). Los dispositivos de menor velocidad se pueden conectar al bus de expansión .

16.7 BUSES ESTÁNDARES

Las razones de la aparición de buses estándar son diversas. En algunos casos se debe a que algunas de las máquinas se hacen tan populares que sus buses se convierten en estándares es el caso IBM PS_AT. En otras ocasiones se reúnen un conjunto de fabricantes para solucionar problemas comunes. Es el caso IPI (Intelligent Peripheral Interface) o del SCSI (Small Computer System Interface). Ambos son buses de entrada /salida. Los buses Nubus, VME , Futurebus, PCI son buses de plano posterior de propósito general diseñados para interconectar procesadores memoria y dispositivos de entrada/salida.

En la siguiente tabla aparecen algunas de las opciones de diseño de un bus indicando la decisión que se debería adoptar en el caso que se deseara un bus de alto rendimiento o de bajo coste:

Opción	alto rendimiento	bajo coste
Anchura de bus	Separadas datos direcciones	Mulitplexa datos direcciones
Anchura de datos	Mas ancho es mas rápido(32)	Mas estrecho mas barato (8)
Tamaño de transferencia	Múltiples palabras menos gasto de bus	Mas simple transferir una palabra
Maestros del bus	Múltiples(arbitraje)	Único

Reloj	Sincrono	Asíncrono

características	**VME**	**nubus**	**Futurebus**	**IPI**	**SCSI**
	plano posterior	plano posterior	plano posterior	ES	ES
	no mux	mux	mux		
	16-32bits	32	32	16	8
nº maestros	múltiple	múltiple	múltiple	único	múltiple
arbitraje	daisy chain	autoseleccion	autoseleccion		autoseleccion
	asincro	sincro	asincro	asincro	las dos
mx nº dispositivos	21	16	20	8	7
longitud	0.5m	0.5m	0.5m	50	25
ancho de banda de una palabra	12,9MB/s	13,2MB/S	15,5	25	5 o 1,5
ancho de bandamultiples palabras	13,6	26,4	20,8	25	1,5mb

Para dar la anchura de banda de un bus conviene indicar también los tiempos de memoria en el cuadro se supone memoria de 150ns

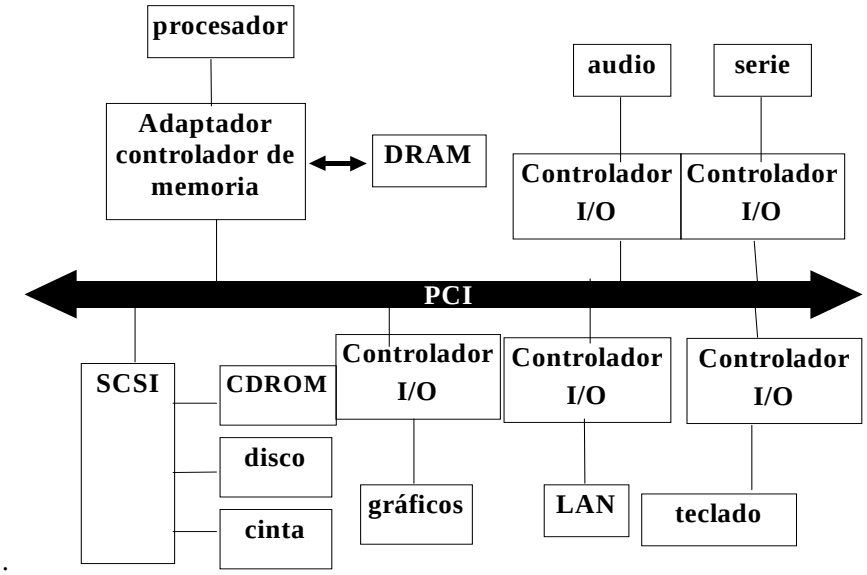
16.8 EJEMPLOS DE BUS: EL PCI PERIPHERAL COMPONENT INTERCONNECT

Es un bus de plano posterior que utiliza un arbitraje centralizado asíncrono. Fue propuesto en 1990 por INTEL para los sistemas basados en Pentium. Posteriormente hicieron las patentes de dominio público y se promueve la sociedad PCI SIG para su estudio desarrollo y promoción. Como consecuencia el PCI ha sido ampliamente adoptado en computadores personales, estaciones de trabajo y servidores. Además de bus de plano posterior también se puede utilizar como bus de E/S y como bus de arquitectura entre planta (alta velocidad).

Fue pensado para dar prestaciones a los sistemas de entrada/salida de alta velocidad como son los adaptadores de pantalla gráfica, controladores de interfaz de red, etc. Su característica más importante es que se ajusta, económicamente, a los requisitos de E/S de los sistemas actuales. Es decir se implementa con pocos circuitos integrados y permite la conexión a otros buses. Su línea de datos puede seleccionarse de 32 o de 64 bits, su frecuencia es de 33Mhz y su velocidad de transferencia de 264Mbps o 2.11Gbps. Esta pensado para soportar cierta variedad de configuraciones basadas en microprocesadores

- **ESTRUCTURA TRADICIONAL**

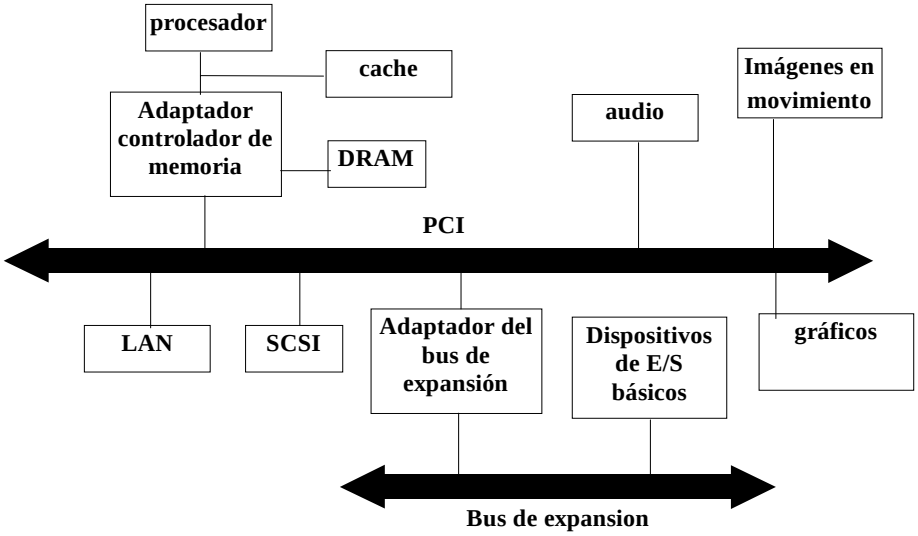
Se usa como bus de expansión colgando de PCI todos los periféricos del sistema esta estructura es usada en el APPLE Macintosh serie 7200



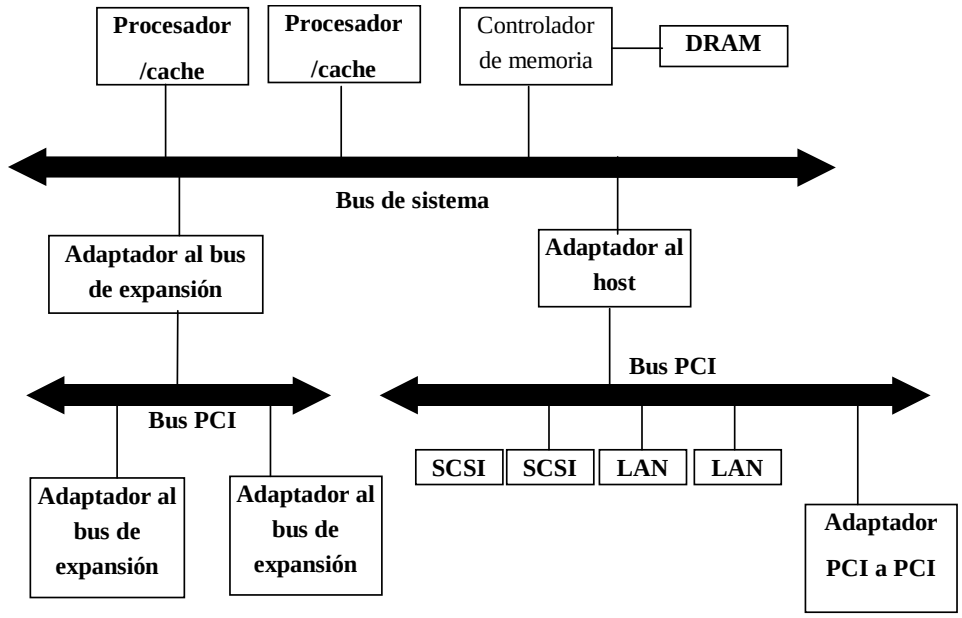
.

ν **BUS DE ALTA VELOCIDAD**

Solo se conectan al PCI los periféricos de alta velocidad. El adaptador actúa como un buffer de datos puesto que las velocidades de los dispositivos de entrada salida y del procesador pueden diferir



ν **SISTEMA MULTIPROCESADOR**



En esta configuración al bus de sistema solo se conectan los módulos procesador, cache, la memoria principal y los adaptadores a PCI. El uso de adaptadores mantiene al PCI independiente de los procesadores. El PCI se utiliza como bus de expansión y como bus de alta velocidad.

Se puede configurar como de 32 o 64 bits. Tiene los siguientes grupos funcionales:

- Terminales de sistema, reloj e inicio
- Terminales de direcciones y datos 32 líneas de datos y direcciones multiplexadas en el tiempo
- Terminales de control de la interfaz controlan la temporización de la transferencias y y proporcionan coordinación entre quienes las inician y los destinatarios.
- Terminales de arbitraje. No son líneas compartidas. Cada maestro tiene sus propias líneas. Que conectan directamente al arbitro.
- Terminales para señales de error.

Además PCI define otras 50 señales opcionales:

- Terminales de interrupción. No son líneas compartidas, cada dispositivo tiene las suyas.
- Terminales de soporte de cache,. Necesarios para permitir memorias cache en el PCI.(permiten el uso de protocolos de coherencia cache de sondeo de bus (snoopy cache)).
- Terminales de extensión de bus a 64 bits.
- Terminales de test .(jtag/boundary scan) se ajustan al estándar ieee149.1

La actividad del bus consiste en transferencias entre dos elementos, denominándose maestro al que inicia la transición.

- Reconocimiento de interrupción
- Ciclo especial.- Inicia la difusión de un mensaje a uno ovarios destinos.
- Lectura y escritura es intercambio de datos entre el modulo que inicia la transferencia y y un controlador de entrada salida. Cada dispositivo de entradasalida tiene su propio espacio de direcciones
- Lectura de memoria, lectura de línea de memoria, lectura múltiple de memoria según se lean medio bloque un bloque o varios bloques de cache.
- Escritura de memoria
- Escritura e invalidación de memoria indica que se ha escrito al menos una línea en la cache. (permite el funcionamiento de cache con postescritura)
- Lectura y escritura de configuración. Permite a un dispositivo maestro configurar un dispositivo conectado al PCI. Cada dispositivo PCI puede disponer de hasta 256 registros para configurar dicho dispositivo.
- Ciclo de dirección dual. Indicar que se usan direcciones de 64 bits.

Sistema de arbitraje centralizado asíncrono en el que cada maestro tiene una única señal de petición REQ y cesión GNT del bus. Estas líneas se conectan a un arbitro central y se utiliza un simple intercambio de señales de petición - cesión. El PCI no especifica un algoritmo especial de arbitraje.

17 JERARQUIA DE BUSES Y BUSES ESTANDAR

Pedro de Miguel califica los Buses en 5 tipos diferentes:

Interno de Pastilla.

- Conexión de componentes en la PCB.
- Panel posterior
- Sistema.
- E/S paralela
- E/S serie

ν **Tipo 0.- INTERNO DE UNA PASTILLA**

- No tiene acceso el usuario
- Ni los fabricantes suministran información, ni los usuarios la necesitan.

ν **Tipo 1.- INTERCONEXIÓN DE PASTILLAS DE UN CIRCUITO IMPRESO**

- Restringido a la placa
- Longitud: decenas cm
- Impedancia capacitiva.
- No existen problemas de velocidad salvo en logias rápidas
- Suelen ser síncronos, con solo un maestro

ν **Tipo 2.- PANEL POSTERIOR**

- Interconexión de placas.
- Longitud de 1 metro
- Número de señales ≅ 100
- Abstracción de las peculiaridades de las pastillas que forman la tarjeta.

ν **Tipo 3.- DE SISTEMA**

- Idénticas a las anteriores.
- Más largos: l = 10 metros. Hay que tratarlo como una línea de transmisión.:
  - Dispositivos terminales.
  - Repetidores ((Buffers)).

ν **Tipo 4.- E/S PARALELA**

- Conexión de periferias en paralelo mediante una Norma preestablecida diferente a la del Bus ((SCSI)).

ν **Tipo 5.- CONEXIÓN SERIE DE PERIFÉRICOS**

- Cubren las distancias más largas.
- Velocidades de transmisión más lentas.
- Conexión de terminales y periféricos de baja velocidad: redes locales.
- RS -232-C.

Petterson-Hennesy realizan la siguiente clasificación:

- Bus procesador-memoria
- Bus de plano posterior
- Bus de entrada/salida

ν **CARACTERÍSTICAS DEL PROCESADOR - MEMORIA**

- Cortos y de alta velocidad
- Adaptados al sistema de memoria para maximizar la anchura de banda memoria procesador.

- Diseño exclusivo
- Al diseñarse se conocen perfectamente todos los dispositivos que van a conectar

ν **BUS DE PLANO POSTERIOR**

- Se diseñan para permitir que coexistan en un solo bus
  - La memoria
  - El procesador
  - Los dispositivos de entrada salida.
- Generalmente necesitan lógica adicional para conectarse a un bus de entrada salida o a un dispositivo→ controladores
- En los Macintosh el bus que conecta el procesador y la memoria es un bus de plano paralelo llamado NUBUS.
- A este bus se puede conectar directamente un bus SCSI de entrada/salida.
- Bus estándar

ν **BUS DE ENTRADA/SALIDA**

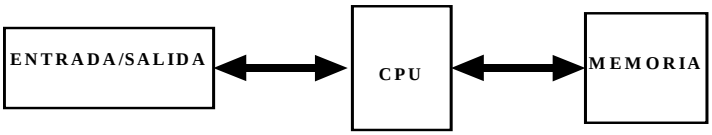
- Largos
- Con muchos tipos de dispositivos conectados
- No tienen una interfaz directa con la memoria
- Se conectan a la memoria mediante buses de plano posterior o mediante buses de procesador memoria..
- Interfaz sencilla.
- Los dispositivos necesitan pocos elementos electrónicos para conectarse a él
- Bus estándar

Los buses generalmente se conectan entre sí mediante adaptadores. Tanto los buses de plano -paralelo como los buses de entrada/salida se diseñan desconociendo los dispositivos que se van a conectar. Los Sistemas expandibles de alto rendimiento utilizan los tres tipos de buses, Ej multiprocesadores IBM RS/6000 y Silicon Graphic. Características de esta organización jerárquica son que el Bus procesador- memoria es mucho más rápido que uno de plano posterior y la jerarquía aprovecha esta velocidad, Por otro lado, el Sistema de entrada salida es expandible con facilidad conectando muchos controladores o buses al plano posterior y esta expansión no afecta al rendimiento del bus de procesador-memoria

17.4 ESTRUCTURA DE LOS BUSES DE SISTEMA

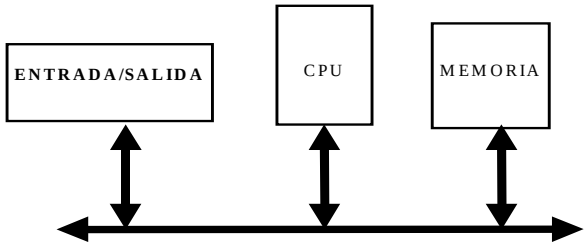
ν HAMACHER, simplificando, habla de 2 estructuras muy básicas de Bus:

a) Dos Buses:



Típico de una organización de entrada -salida aislada de memoria, que poco flexible y presenta dificultades para añadir nuevos módulos.

b) Un Bus:

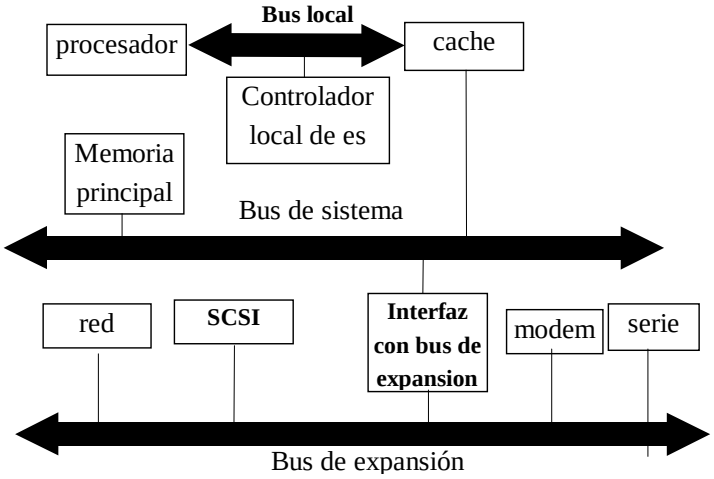


- Bajo coste

- Flexibilidad para conectar nuevos módulos
- Velocidad de operación baja
- Máquinas pequeñas

Estas dos estructuras de Bus son demasiado generales y se dan pocas veces. La razón es que dan lugar a importantes **Problemas** (Stalling) Por ejemplo, si se conecta un gran número de dispositivos al bus, las prestaciones disminuyen debido al mayor retardo de propagación. Este retardo de propagación es importante porque es el que determina el tiempo necesario para coordinar el uso del bus. Además si el control pasa frecuentemente de un dispositivo a otro, empeoran notablemente las prestaciones y el bus se convierte en un cuello de botella. **La solución** es utilizar varios buses organizados jerárquicamente. A esta estructura se la llama estructura MULTIBUS y su objetivo es acelerar el sistema

17.4.1 MULTIBUS TRADICIONAL:



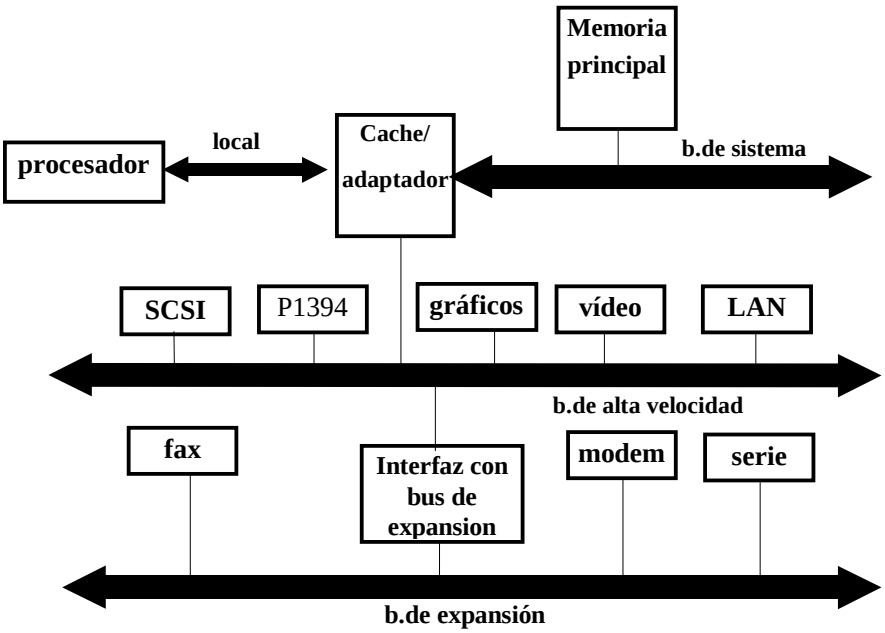
Los componentes de esta jerarquía son los siguientes: Un Bus local que conecta el procesador a la memoria cache. La memoria cache se conecta a través de su controlador con el bus local y el bus de sistema. El uso de cache alivia los accesos del procesador a la memoria principal.

Un Bus de sistema que conecta la memoria principal y al procesador a través de la memoria cache. Como Ventaja tiene que la transferencia de entrada salida no interfiere en la actividad del procesador. Se podrían conectar controladores de E/S directamente al bus de sistema pero esta opción no es eficiente y degrada el rendimiento.

Por último un bus de expansión al que se conectan los controladores de entrada/ salida. La interfaz del bus de expansión con el bus de sistema controla las transferencias entre los dispositivos de entrada /salida y el bus de sistema. De esta forma se conecta gran variedad de dispositivos al sistema al tiempo que se aísla la memoria y el procesador del trafico entre la memoria y los dispositivos de entrada/salida

Como inconveniente tiene que a medida que mejoran las prestaciones de los dispositivos de entrada/salida se degrada la eficiencia de esta estructura MULTIBUS. La Solución está en diseñar estructuras más complejas utilizando buses de alta velocidad, como se ve a continuación.

17.4.2 ARQUITECTURA DE ALTAS PRESTACIONES:



Esta jerarquía añade un cuarto bus a la estructura tradicional. Sigue teniendo el bus **local** que conecta el procesador y la memoria cache. Ésta conecta a través de un adaptador el bus local con el bus de sistema y el de alta velocidad. Al bus de alta velocidad se conectan todos los periféricos de altas prestaciones como las SCSI entrada salida paralela y P1394 (bus diseñado par adispositivos de es de alta capacidad). Los dispositivos de menor velocidad se pueden conectar al bus de expansión

17.5 BUSES ESTÁNDARES

Las razones de la aparición de buses estándar son diversas. En algunos casos se debe a que algunas de las máquinas se hacen tan populares que sus buses se convierten en estándares es el caso IBM PS_AT. En otras ocasiones se reúnen un conjunto de fabricantes para solucionar problemas comunes. Es el caso IPI (Intelligent Peripheral Interface) o del SCSI (Small Computer System Interface). Ambos son buses de entrada /salida. Los buses Nubus, VME , Futurebus, PCI son buses de plano posterior de propósito general diseñados para interconectar procesadores memoria y dispositivos de entrada/salida.

En la siguiente tabla aparecen algunas de las opciones de diseño de un bus indicando la decisión que se debería adoptar en el caso que se deseara un bus de alto rendimiento o de bajo coste:

Opción	alto rendimiento	bajo coste
Anchura de bus	Separadas datos direcciones	Mulitplexa datos direcciones
Anchura de datos	Mas ancho es mas rápido(32)	Mas estrecho mas barato (8)
Tamaño de transferencia	Múltiples palabras menos gasto de bus	Mas simple transferir una palabra
Maestros del bus	Múltiples(arbitraje)	Único
Reloj	Sincrono	Asíncrono

características	**VME**	**nubus**	**Futurebus**	**IPI**	**SCSI**
	plano posterior	plano posterior	plano posterior	ES	ES
	no mux	mux	mux		
	16-32bits	32	32	16	8
nº maestros	múltiple	múltiple	múltiple	único	múltiple
arbitraje	daisy chain	autoseleccion	autoseleccion		autoseleccion
	asincro	sincro	asincro	asincro	las dos

mx nº dispositivos	21	16	20	8	7
longitud	0.5m	0.5m	0.5m	50	25
ancho de banda de una palabra	12,9MB/s	13,2MB/S	15,5	25	5 o 1,5
ancho de bandamultiples palabras	13,6	26,4	20,8	25	1,5mb

Para dar la anchura de banda de un bus conviene indicar también los tiempos de memoria en el cuadro se supone memoria de 150ns

17.6 EJEMPLOS DE BUS: EL PCI Y EL FUTUREBUS

17.6.1 PCI (PERIPHERAL COMPONENT INTERCONNECT)

Es un Bus de plano posterior que utiliza un arbitraje centralizado asíncrono. Fue propuesto en 1990 por INTEL para los sistemas basados en pentium. Posteriormente hicieron las patentes de dominio público y se promueve la sociedad PCI SIG para su estudio desarrollo y promoción . Como consecuencia el PCI ha sido ampliamente adoptado en Computadores personales, estaciones de trabajo y Servidores. Además de bus de plano posterior también se puede utilizar como: bus de E/S y como bus de arquitectura entre planta (alta velocidad).

Fue pensado para dar prestaciones a los sistemas de entrada/salida de alta velocidad como son los adaptadores de pantalla gráfica, controladores de interfaz de red, etc. Su característica más importante es que se ajusta, económicamente, a los requisitos de E/S de los sistemas actuales. Es decir se implementa con pocos circuitos integrados y permite la conexión a otros buses.

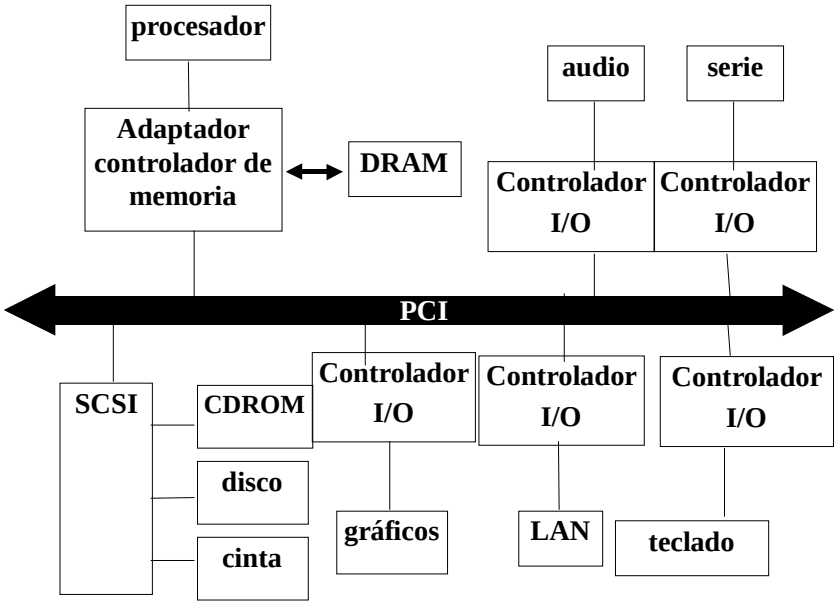
Su línea de datos puede seleccionarse de 32 o de 64 bits, su frecuencia es de 33Mhz y su velocidad de transferencia de 264Mbps o 2.11Gbps

Esta pensado para soportar cierta variedad de configuraciones basadas en microprocesadores

ν **ESTRUCTURA TRADICIONAL**

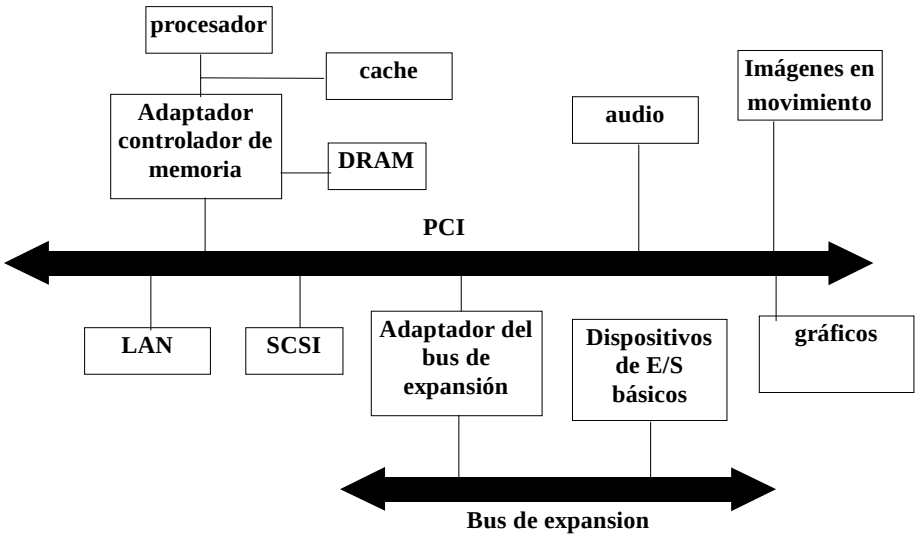
- Se usa como bus de expansión colgando de PCI todos los periféricos del sistema

Estructura usada en el APPLE Macintosh serie 7200



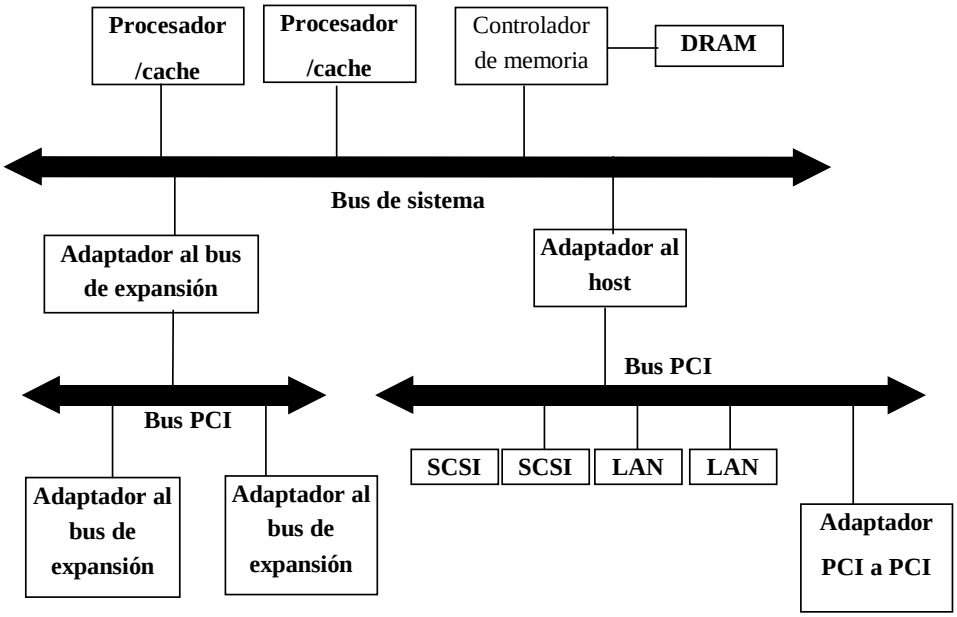
ν **BUS DE ALTA VELOCIDAD**

Solo se conectan al PCI los periféricos de alta velocidad



El adaptador actúa como un buffer de datos puesto que las velocidades de los dispositivos de entrada salida y del procesador pueden diferir

ν **SISTEMA MULTIPROCESADOR:**



En esta configuración al bus de sistema solo se conectan los módulos procesador, cache, la memoria principal y los adaptadores a PCI.. El uso de adaptadores mantiene al PCI independiente de los procesadores. El PCI se utiliza como bus de expansión y como bus de alta velocidad.

17.6.1.1 Estructura:

Se puede configurar como de 32 o 64 bits.

Tiene los siguientes grupos funcionales:

- Terminales de sistema, reloj e inicio
- Terminales de direcciones y datos 32 líneas de datos y direcciones multiplexadas en el tiempo

- Terminales de control de la interfaz controlan la temporización de la transferencias y y proporcionan coordinación entre quienes las inician y los destinatarios.
- Terminales de arbitraje. No son líneas compartidas. Cada maestro tiene sus propias líneas. Que conectan directamente al arbitro.
- Terminales para señales de error.
- Además PCI define otras 50 señales opcionales:
- Terminales de interrupción. No son líneas compartidas, cada dispositivo tiene las suyas.
- Terminales de soporte de cache,. Necesarios para permitir memorias cache en el PCI.(permiten el uso de protocolos de coherencia cache de sondeo de bus (snoopy cache)).
- Terminales de extensión de bus a 64 bits.
- Terminales de test .(jtag/boundary scan) se ajustan al estándar ieee149.1

17.6.1.2 Ordenes PCI

La actividad del bus consiste en transferencias entre dos elementos, denominándose maestro al que inicia la transición.

- Reconocimiento de interrupción
- Ciclo especial.- Inicia la difusión de un mensaje a uno ovarios destinos.
- Lectura y escritura es intercambio de datos entre el modulo que inicia la transferencia y y un controlador de entrada salida. Cada dispositivo de entradasalida tiene su propio espacio de direcciones
- Lectura de memoria, lectura de línea de memoria, lectura múltiple de memoria según se lean medio bloque un bloque o varios bloques de cache.
- Escritura de memoria
- Escritura e invalidación de memoria indica que se ha escrito al menos una línea en la cache. (permite el funcionamiento de cache con postescritura)
- Lectura y escritura de configuración. Permite a un dispositivo maestro configurar un dispositivo conectado al PCI. Cada dispositivo PCI puede disponer de hasta 256 registros para configurar dicho dispositivo.
- Ciclo de dirección dual. Indicar que se usan direcciones de 64 bits.

17.6.1.3 Arbitraje

Sistema de arbitraje centralizado asíncrono en el que cada maestro tiene una única señal de petición REQ y cesión GNT del bus. Estas líneas se conectan a un arbitro central y se utiliza un simple intercambio de señales de petición - cesión. El PCI no especifica un algoritmo especial de arbitraje.

17.6.2 FUTUREBUS+

Estándar de bus asincronos de altas prestaciones desarrollado por IEEE. Es Independiente de la tecnología, de procesador y la arquitectura. Utiliza un protocolo básico de transferencia asíncrona aunque puede permitir protocolos sincronizados por la fuente de la transferencia.

No tiene un limite determinado por la tecnología

Está constituido por protocolos de arbitraje paralelos y totalmente distribuidos, que permiten tanto protocolos de conmutación de circuitos como protocolos de transacción partida.

Proporciona soporte para sistemas tolerantes a fallos y de fiabilidad elevada

Ofrece soporte directo para memoria compartida con uso de memorias caches

Proporciona una definición compatible de transporte de mensajes . Tiene anchuras de bus de 32, 64, 128, 256 y usa arbitraje distribuido y centralizado

En definitiva future bus es un bus complejo que lo mismo sirve como bus local que como bus de alta velocidad para servir a periféricos de elevado rendimiento.

> *La diferencia entre PCI y Futurebus es que PCI esta orientada a proporcionar una implementación de bajo costo que requiera mínimo espacio físico mientras que Futurebus busca a proporcionar flexibilidad y y una amplia funcionalidad que se ajuste a las necesidades de una gran variedad de sistemas de altas prestaciones especialmente para sistemas de coste elevado.*