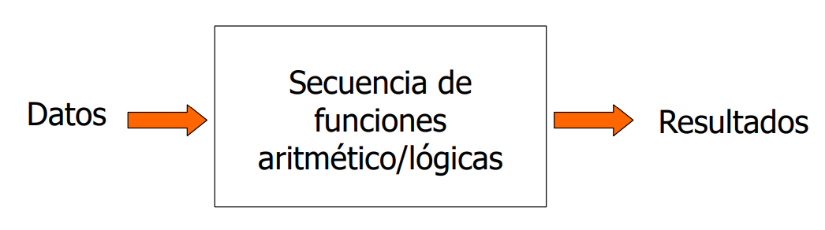
**Clase 1 – Repaso OC**Concepto de “programa”

Se puede considerar como antecedente más cercano de los Sistemas de Cómputo a las máquinas de calcular.  
Las primitivas máquinas de calcular eran de tipo mecánico, capaces de realizar una sola operación (por ejemplo suma o resta) a la vez. Si el cálculo a resolver requería un solo tipo de operación por ejemplo sumas o restas repetitivas) solo era necesarioingresar los datos de cada operación.

Si, por el contrario, también se cambiaba la operación, se requería además modificar la máquina.

En ambos casos, era necesaria la intervención de una persona (“operador”) para hacer esas tareas.

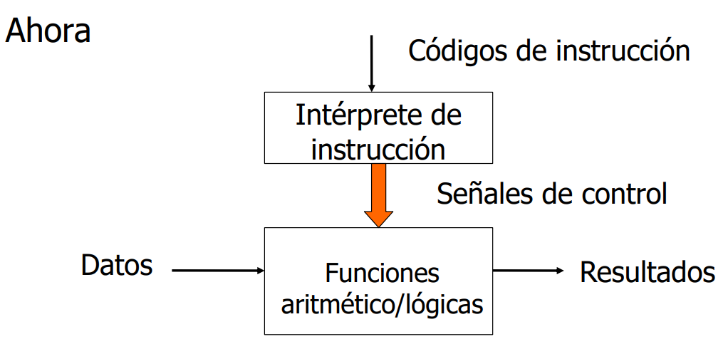
Conceptualmente, esta forma de operación se puede asimilar al siguiente modelo:

Este modelo en el que un “operador” tiene que actuar (modificar) sobre la máquina (es decir la “Unidad de Cálculo”) de acuerdo a la tarea a ejecutar, se puede considerar como un “modelo de Programación en hardware”, porque cuando se cambia la tarea, se requiere cambiar el medio físico que la ejecuta.

En un Sistema de cómputo, la secuencia de tareas está almacenada en la máquina como instrucciones.

Las instrucciones son interpretadas por la propia máquina en una unidad especial (Unidad de Control CU o Intérprete de instrucciones por hardware) y son son convertidas en señales que comandan: la entrada de los datos, el tipo de operación, y la salida de los resultados. La Unidad de cálculo es universal y capaz de resolver distintos tipos de operaciones.

El tipo de operación a resolver lo obtiene desde el Intérprete de instrucciones apartir de la instrucción leída.  
El modelo de operación cambia de la siguiente manera:



Este nuevo modelo, en el que las operaciones se definen por medio de instrucciones almacenadas en memoria, se puede considerar como un “modelo de Programación en software”, porque el cambio de la tarea, solo requiere modificar la secuencia de instrucciones.

En el modelo de Programación en software, la información requerida para ejecutar cada paso de la tarea es proporcionada por la secuencia de instrucciones a ejecutar. Cada instrucción es convertida en señales que son enviadas a las unidades funcionales.

Cambiar la respuesta del Sistema implica únicamente modificar la secuencia de instrucciones (“programa”), que es fácilmente modificable (el “software”) debido a que está almacenada en una memoria dentro de la propia máquina.

Por otra parte, las unidades funcionales (el “hardware”) no deben modificarse, su respuesta se modifica desde el programa (“software”).  
  
Cada instrucción contiene la información necesaria para resolver esa acción.

Se requiere de un “intérprete por hardware” (la Unidad de control) que convierta la instrucción (en realidad decodifica) en el conjunto de señales requeridos por el hardware.

Básicamente, un programa es una secuencia de instrucciones que realiza una tarea determinada. El programa puede ser modificado de una manera relativamente fácil (mucho más fácil que modificar el hardware). No hay que cambiar el hardware, es siempre el mismo. Cambia su comportamiento (y los resultados que produce) de acuerdo a la secuencia de acciones que ejecuta.

En definitiva, un Programa es una secuencia de acciones almacenadas en el sistema de

cómputo.

Cada acción es una instrucción (de máquina).

Cada instrucción es convertida en señales.

Diferentes señales de control se necesitan para cada operación.

La Unidad de Control (CU) obtiene toda la información necesaria para resolver (ejecutar) la acción definida en la instrucción.

Modelo de von Neumann – IAS

La máquina se conoció como IAS, y fué desarrollada por John von Neumann en la Universidad de Princeton (NJ). El Proyecto comenzó en 1942 y los últimos ajustes se completaron entre 1951 y 1952.

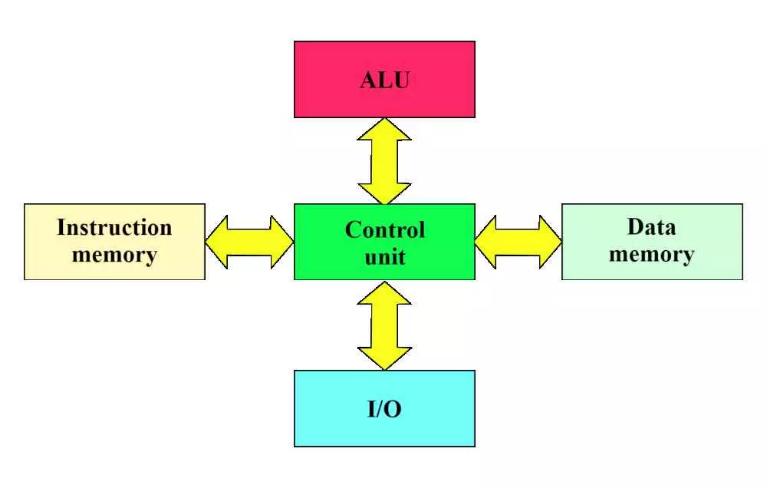
Una de las novedades más importantes de la IAS era que operaba en binario (en lugar de decimal). Tenía una memoria única (para instrucciones y datos), con 4096 palabras de 40 bits.

El repertorio de instrucciornes estaba compuesto por 21 instrucciones de 20 bits (incluyendo transferencia de datos, saltos, aritméticas, cálculo de dirección)

El banco de registros de almacenamiento en la CPU estaba compuesto por:

* Registro Buffer de Memoria (MBR)
* Registro de Direcciones de Memoria (MAR)
* Registros de Instrucción y Buffer de Instrucción
* Registro Contador de Programa (Program Counter)
* Registros Acumulador y Multiplicador/Cociente

El modelo de von Neumann de la IAS estaba basado en 5 Unidades funcionales.



Las 5 Unidades funcionales eran:

* Unidad de Control (UC)
* Unidad Aritmético-Lógica (ALU)
* Unidad de Memoria
* Unidad de Entrada (de datos)
* Unidad de Salida (de datos)

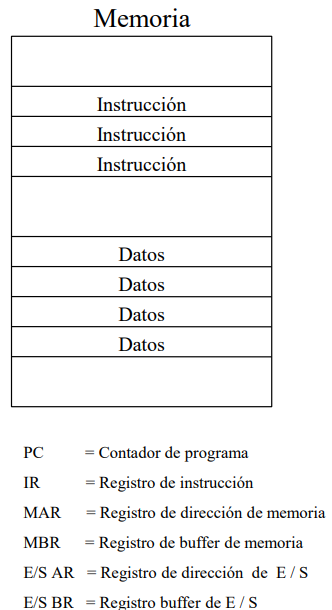
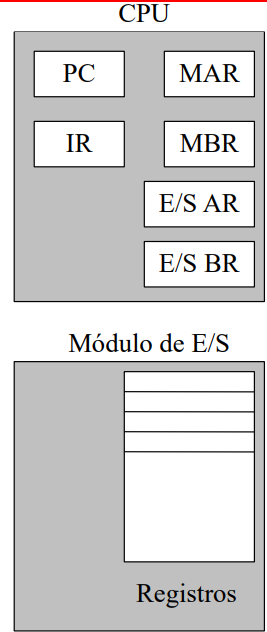
Este modelo de Arquitectura se puede simplificar en un modelo basado en 3 Unidades funcionales, modelo que perdura actualmente en muchas máquinas de propósito general.

La UNIDAD CENTRAL DE PROCESAMIENTO (CPU) constituída por:

* Unidad de Control (UC)
* Unidad Aritmético-Lógica (ALU)
* Registros.

La UNIDAD DE ENTRADA/SALIDA (E/S) para introducir y extraer los datos e instrucciones.

La UNIDAD DE MEMORIA para almacenar temporalmente datos e instrucciones.



Lenguaje de máquina o absoluto

El lenguaje que la máquina es capaz de “entender” y ejecutar se conoce como “lenguaje absoluto o de máquina”, y es del tipo binario.

Las instrucciones están codificadas mediante un conjunto de bits almacenados en palabras de memoria (1 palabra o más).

La representación de las instrucciones en memoria es puramente binaria.

Lenguaje simbólico (Assembly)

Dado que es prácticamente imposible para un programador tratar con las representaciones binarias de las instrucciones de máquina, se usan otras representaciones más“amigables”.

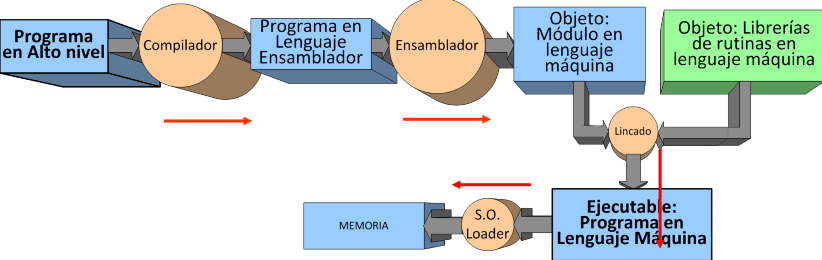
Estas representaciones tienen distintos grados o niveles de abstracción. En el nivel siguiente al nivel de lenguaje de máquina, en cuanto a nivel de abstracción, está la representación simbólica (Lenguaje Assembly).

En la representación simbólica, los campos que forman la instrucción se pueden representar con textos o números.

En el lenguaje Assembly, los diferentes campos de la instrucción se pueden reemplazar por números o símbolos más fáciles de usar que el binario absoluto.

Por ejemplo, el campo binario de la instrucción que contiene la información del tipo de instrucción (campo de código de operación COP) puede ser reemplazado, en el lenguaje Assembly por mnemónicos más sencillos de memorizar:

* ADD (significa código de operación de) adición (suma)
* SUB sustracción (resta)
* MOV movimiento de datos
* AND, OR, XOR operaciones lógicas

El proceso de transformación (compilación) de un programa escrito en lenguaje de alto nivel (HLL) hasta el lenguaje de máquina típicamente es el siguiente.  


Elementos de una instrucción de máquina

Una instrucción de máquina es un código binario compuesto por una cantidad determinada de bits.

Los bits son usados por la Unidad de Control (CU) para continuar y completar el proceso de ejecución de la instrucción. Para ello, el código debe contener suficiente “información” para que la Unidad de Control pueda resolverla.

Las instrucciones tienen distinto grado de complejidad. Van desde las que no hacen nada, hasta las que operan sobre múltiples datos.

En general las instrucciones más complejas requieren de múltiples pasos o acciones. Las más sencillas se resuelven en pocos pasos.

Un ejemplo de instrucción “relativamente compleja” es la que busca 2 operandos y produce un resultado. Para esta instrucción la CPU necesita definir:

1: tipo de operación

2: Lugar del 1er operando (donde está)

3: Lugar del 2do operando (donde está)

4: Lugar del resultado (donde guardarlo)

5: donde está la próxima instrucción

Dado que hay diferentes datos que proveer, la instrucción está organizada en conjuntos de bits, comúnmente denominados campos, que proveerán esos datos.

La cantidad de bits de un campo dependen de la cantidad de información que debe proveer. Por ejemplo, si un procesador dispone de 256 tipos de instrucciones, y un campo se usa para especificar el tipo de instrucción (conocido comúnmente como campo de Código de Operación COP), entonces ese campo requiere de al menos 8 bits.

Los campos que típicamente pueden estar incluidos en una instrucción son:

* 1.- Campo de Identificación de la instrucción (Código de operación): Especifica la operación a realizar (ej. sumar).
* 2.- Campos de Referencias a operandos: Establece la información de referencia a el o los operandos fuentes (si es que se requieren). La información de referencia puede ser de distinto tipo (de acuerdo al modo de direccionamiento).

La operación puede involucrar uno ó más operando fuente (o de entrada). Por ejemplo en una suma se requiere especificar 2 operandos.

* 3.- Campo de Referencia del resultado: Establece dónde almacenar el resultado. La operación puede involucrar uno o más resultados (de salida). Por ejemplo, en una suma se produce 1 resultado solamente.
* 4.- Campo de Referencia de la siguiente instrucción. Provee a la CPU con información para determinar donde buscar la siguiente instrucción después de la ejecución de la instrucción anterior. En la mayoría de los casos, la próxima instrucción se ubica a continuación de la instrucción actual, es decir, en la siguiente posición de memoria. Esta situación se conoce como acceso

secuencial a instrucciones consecutivas de memoria.

Taxonomía basada en formato de instrucción

Una forma de clasificación de las máquinas está basado en el formato de instrucción. La clasificación identifica la cantidad de campos que contienen direcciones de memoria.

Suponiendo el caso de una instrucción de suma de 2 operandos en memoria, y resultado almacenado en memoria, se necesitan:

* 2 campos de direcciones para hacer referencia a los operandos.
* 1 campo de dirección para hacer referencia a donde almacenar el resultado.
* 1 campo de dirección de referencia de la ubicación de la próxima instrucción.

En base a lo anterior, se pueden tener 5 tipos de maquinas básicos:

Máquina de 4 direcciones:

COP DirRes DirOP1 DirOp2 DirProxIns

Máquina de 3 direcciones:

COP DirRes DirOp1 DirOp2

Máquina de 2 direcciones:

COP DirOp1 DirOp2

Máquina de 1 dirección:

COP DirOp1

Maquina de 0 direcciones:

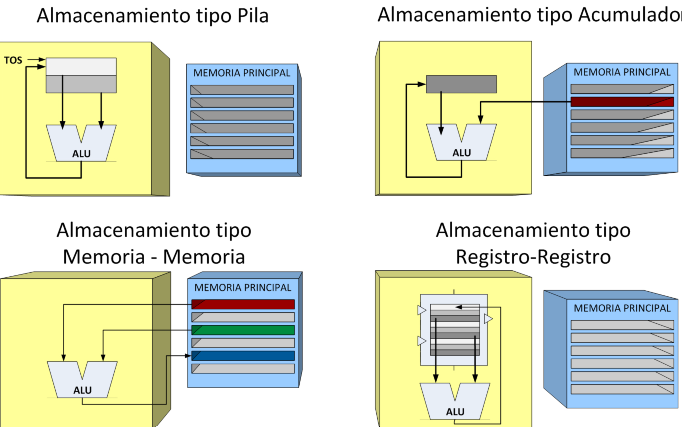
COP

Taxonomía basada en los tipos de almacenamientos de datos

Otra forma de clasificación de las máquinas está basada en los tipos de almacenamiento de los datos. Los datos pueden estar almacenados en 3 lugares físicos:

* Memoria: cache, principal, virtual. En todos los casos se requiere para referenciar el dato, una dirección de memoria.
* Registro de la CPU: la referencia es un número de registro
* Dispositivo de E/S: la referencia es un número de registro de E/S

Suponiendo una instrucción con referencias a 2 operandos y 1 resultado, podemos tener 4 tipos de máquinas básicas.



Decisiones en el diseño del conjunto de instrucciones

Cuando se diseña la arquitectura de un procesador son muchos los aspectos que se tienen en cuenta. Algunos de los más relevantes son:

1) Formato de instrucción

2) Tipo de datos

3) Repertorio de instrucciones

4) Cantidad y tipo de registros

5) Modos de direccionamiento

**1) Formato de instrucción**

Se refiere a la estructura que tendrán las instrucciones. Puede ser fijo o variable y hay que tener en cuenta la cantidad de campos

* Formato fijo: todas las instrucciones tienen el mismo tamaño (bits). La captación y ejecución es más sencilla pero aumenta la cantidad de instrucciones de los programas.
* Formato variable: la longitud de las instrucciones varía con el tipo de instrucción y modo de direccionamiento. La captación y ejecución es más compleja pero disminuye la cantidad de instrucciones de los programas.

A mayor cantidad de referencias de direcciones, se tienen instrucciones más complejas y largas, menos instrucciones por programa, y captación y ejecución de las instrucciones en forma más lenta.

A menor cantidad de referencias de direcciones, se tiene instrucciones más simples y cortas, más instrucciones por programa, captación y ejecución de las instrucciones más rápida.

**2) Tipos de datos**

Una decisión relevante, referida a las características del procesador y del repertorio de instrucciones, es la cantidad y tipo de datos que puede soportar.  
Los tipos de datos más comunes son:

* Numérico: Con/sin signo, punto fijo / punto flotante, BCD, etc.
* Caracter: ASCII, EBCDIC, etc.
* Datos lógicos: manipulación de bits

**3) Repertorio de instrucciones**

En el diseño del repertorio de instrucciones, se deben considerar, de manera prioritaria:

* La cantidad de instrucciones (operaciones)
* Tipos de instrucciones
* Características de las instrucciones (sencillas, complejas, etc.)

**4) Administración y uso de Registros**

Dado que los registros disponibles en un procesador son un recurso escaso y tienen un fuerte impacto en la velocidad de ejecución de los programas, deben ser diseñados en forma sumamente cuidadosa, en particular:

* Tamaño y cantidad de registros
* Tipos de registros: como va a ser el uso y a que instrucciones se van a aplicar.

**5) Modos de direccionamiento**

Un aspecto importante en el diseño del repertorio de instrucciones es el referido a los modos de direccionamiento. Las decisiones de diseño de los modos de direccionamiento se orientan a:

* Cantidad de modos de direccionamiento
* Tipos de modos de direccionamiento

Los diseños de las arquitecturas de procesadores se han orientado hacia 2 estrategias:

1. Aumentar la complejidad y prestaciones del procesador
2. Simplificar y optimizar el Procesadores

El primer camino conduce a procesadores identificados genéricamente como tipo CISC (Computadores de Conjunto Complejo de Instrucciones)

El segundo corresponde a las familias de procesadores RISC (Computadores de Repertorio de Instrucciones Reducido)

Los procesadores CISC básicamente tienen: Repertorio de instrucciones muy amplio, muchos modos de direccionamiento y formato de instrucción variable.

Los procesadores RISC se caracterizan por tener: Repertorio de instrucciones simplificado, pocos modos de direccionamiento y formato de instrucción fijo.

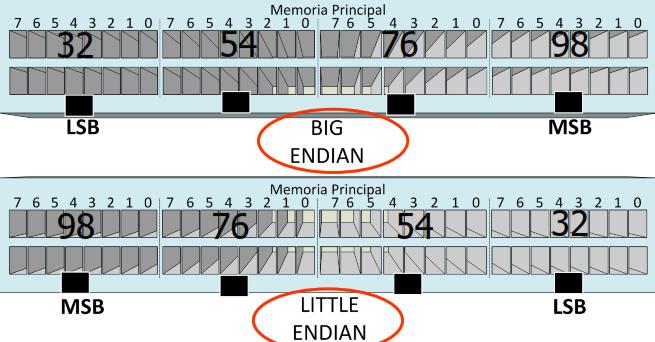
Técnicas de almacenamiento

Se refieren a la forma en que los datos se guardan en memoria, particularmente en los casos en que ocupan más de una palabra de memoria.

Ejemplo: Supongamos que tenemos el siguiente dato de 32 bits (4 bytes), 98765432H (32 bits) y supongamos tener una memoria organizada en palabras de 8 bits (1 byte por palabra). Existen 2 formas de almacenar el dato (en múltiples palabras de memoria) de acuerdo al orden en que se almacenan las diferentes partes del dato:

1. **Big-endian:** el byte más significativo se almacena en la dirección con valor numérico más bajo (es decir en la dirección mas baja)
2. Little-endian: el byte mas significativo se almacena en la dirección con valor numérico más alto (es decir, en la dirección más alta).

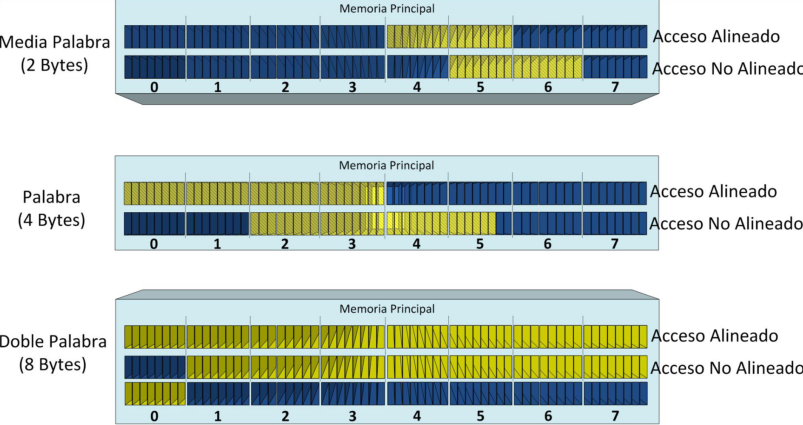
Notar que las 2 formas de almacenamiento son incompatibles entre sí.

03 02 01 00

El otro aspecto relacionado con el almacenamiento de los datos, es el referido a las direcciones de memoria donde se guardan los datos, particularmente en los casos en que ocupan más de una palabra de memoria.

Existen 2 formas de almacenar los datos múltiples en memoria:

* Alineados: si un dato de n bytes empieza en una dirección de memoria múltiplo de 2^m, donde m=log base2 de n, con n>1. En otras palabras, los m bits menos significativos son 0.
* No alineados: si no cumple la condición anterior.

En la figura siguiente se muestran las 2 técnicas para los casos de datos de 2, 4 y 8 bytes.

En los ejemplos anteriores se puede apreciar que:

Si el dato es de 2 bytes, el almacenamiento alineado ocurre cuando se almacena a partir de una dirección múltiplo de 2.

Si el dato es de 4 bytes, el almacenamiento alineado ocurre cuando se almacena a partir de una dirección múltiplo de 4.

Si el dato es de 8 bytes, el almacenamiento alineado ocurre cuando se almacena a partir de una dirección múltiplo de 8.

Algunos procesadores permiten únicamente acceso alineado. Otros procesadores admiten ambos tipos de accesos, alineados y no alienados. Otros, solo alineados. En general, los accesos no alineados son más lentos que los accesos alineados.

Repertorio de instrucciones

Los tipos de instrucciones más comunes que se pueden encontrar en el repertorio de instrucciones de un procesado pertenecen a alguna de las siguientes categorías:

* Transferencias de datos
* Aritméticas
* Lógicas y de rotación y desplazamiento
* Conversión
* Entrada/Salida
* Control de flujo
* Control del sistema

Instrucciones de Transferencia de datos

Son las más típicas. En general se require especificar:

* Ubicación del operando fuente (usando alguno de los modos de direccionamiento)
* Ubicación del operando destino (usando aguno de los modos de direccionamiento)
* Tamaño de los datos a ser transferidos

**Instrucciones Aritméticas**

Son también muy communes. Las instrucciones típicas provistas en un procesador convencional son: Add, Substract, Multiply y Divide. Un aspecto importante de estas instrucciones es el tipo de dato que puede manejar: números enteros sin/con signo, números en punto flotante, etc.

**Instrucciones Lógicas y de rotación y desplazamiento**

Las instrucciones lógicas realizan operaciones que manipulan bits individualmente o en conjunto.  
Las más comunes son las funciones de tipo booleana, es decir: AND, OR, XOR, NOT.  
Las instrucciones de rotación y desplazamiento son, por ejemplo: Rotate (rota a izq/der), Rotate through carry (rota a través del carry a izq/der), Shift (desplaza a izq/der).  
  
**Instrucciones de conversión de tipo**

Son instrucciones poco comunes orientadas a cambiar formatos de datos.

Algunos ejemplos: Conversión de binario a decimal, Conversión de EBCDIC a ASCII

**Instrucciones de ENTRADA/SALIDA**

Algunos procesadores tienen instrucciones específicas de Entrada/Salida. Los que no tienen este tipo de instrucciones, usan las mismas instrucciones de movimiento de datos con la memoria, para realizar las transferencias de E/S.  
Las instrucciones de E/S típicamente son pocas, con acciones específicas. Por ejemplo: IN, OUT

SI disponen de instrucciones específicas de E/S, se requiere disponer, asimismo, de un espacio de direccionamiento separado para E/S y Memoria (típicamente en procesadores de la familia Intel 80x86, por ejemplo).

**Instrucciones de control (de flujo de programa)**

Son instrucciones que pueden modificar el valor contenido en el registro PC. Las instrucciones más comunes de este tipo son:

* Salto Incondicional: JMP destino
* Salto Condicional: JZ destino
* Salto con retorno (llamada a subrutina): CALL subrut
* Retorno de subrutina: RET
* Interrupción por software

**Instrucciones de control de Sistema (control de la CPU)**

Modifican algún estado de operación de la CPU. Las instrucciones más comunes de este tipo son:

* Habilitación o inhibición de interrupciones
* Modo de operación: supervisor/usuario
* Halt: suspensión de actividades

Formato de instrucción fijo

En los formatos de instrucción de longitud fija, el tamaño de la instrucción es el mismo para todas las instrucciones. Por ejemplo, el formato de instrucción de un procesador tipo RISC es el siguiente (solo 3 tipos de instrucciones todas de 32 bits).



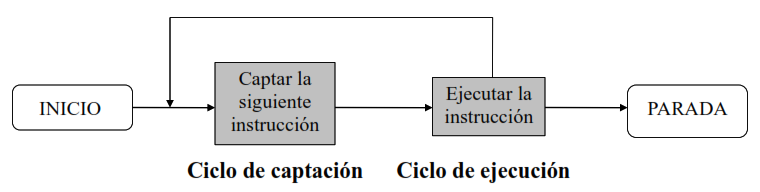
Modos de direccionamiento

Los modos de direccionamiento son las diferentes formas de especificar el lugar donde reside un operando, donde se va a guardar el resultado, o donde buscar la próxima instrucción a ejecutar.  
En general, los procesadores disponen de una variedad más o menos amplia de modos de direccionamiento. Algunos de los más típicos son:

* Inmediato
* Directo: a memoria (“absoluto”), a Registro
* Indirecto: a memoria (en desuso), con registro
* Base: base, base Indexado
* Relativo al PC
* Pila (o relativo al SP)

Ciclo de instrucción básico

El ciclo de instrucción es la secuencia de acciones que realiza el procesador para resolver las instrucciones. En su forma más sencilla, se puede descomponer en 2 acciones básicas: Captación de la instrucción y Ejecución de la operación.



**Ciclo de captación de la instrucción**

El ciclo de captación, o lectura de la instrucción, requiere las siguientes acciones:

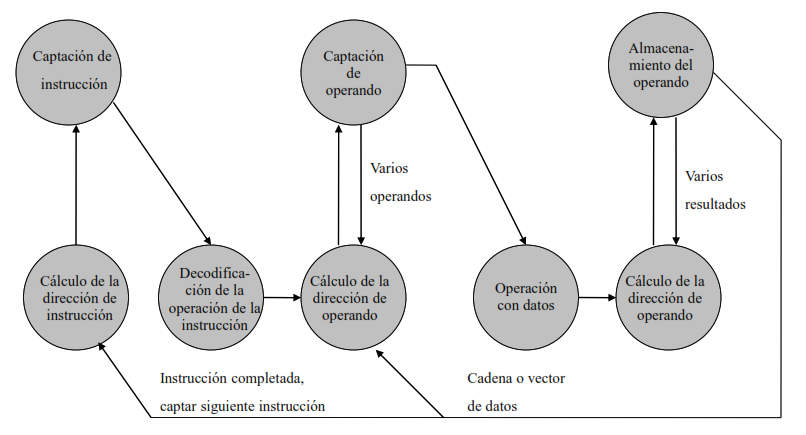
* La dirección de la instrucción que se debe captar se carga en el registro Contador de Programa (PC).
* La UC envía al bus de direcciones el valor almacenado en el PC. La UC lee la instrucción desde la Memoria, y la almacena en un registro temporal denominado Registro de Instrucción (IR).
* El PC se incrementa en 1 (se prepara para apuntar a la próxima instrucción).
* La UC decodifica (“interpreta”) la instrucción captada.

**Ciclo de ejecución de la instrucción**

El ciclo de ejecución puede ser muy variado dependiendo del tipo de instrucción a ejecutar. Algunas de las posibles acciones en este ciclo son:

* Acciones procesador – memoria: por ejemplo transferencia de datos CPU - Memoria.
* Acciones procesador - E/S: por ejemplo, transferencias de datos entre la CPU y un dispositivo de E/S.
* Acciones de procesamiento de datos: operaciones aritméticas o lógicas con los datos.
* Acciones de control: por ejemplo alteración de la secuencia de ejecución (salto, salto a subrutina).
* Acciones combinadas de las anteriores

Ciclo de instrucción detallado



Modelo de Organización del nanoMIPS

Es un modelo simplificado de una familia de máquinas tipo RISC de uso comercial, conocidas como MIPS. Las principales unidades del nanoMIPS son:

* Banco de registros
* ALU de datos
* Contador de programa y cálculo de la dir. de la próxima instruct

PILA

La Pila es una estructura ordenada de datos tipo LIFO (Last In- First Out). El acceso a la estructura se hace únicamente desde el tope (o cabeza) de la estructura y se hace mediante un registro específico denominado Puntero de Pila (SP, Stack Pointer).

Sobre la Pila se pueden realizar 2 operaciones básicas, inversas entre sí:

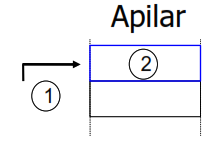
* Almacenar o “apilar”, en Assembly PUSH
* Extraer o “desapilar”, en Assembly POP

Las operaciones de apilado y desapilado requieren 2 acciones que se ejecutan secuencialmente:

1- Movimiento de datos: que pueden ser del tipo Registro a memoria, Memoria a registro o Memoria a memoria

2- Operación del puntero: antes y/o después del movimiento de datos (Stack Pointer – SP). Al haber movimiento de datos y de puntero durante el apilado o desapilado, hay que tener en cuenta dónde apunta el puntero de pila SP y cómo se mueve el S P, porque determina como “crece” la pila (que área de memoria va ocupando)

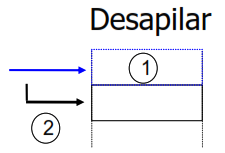
**Manejo de la Pila en procesadores de la familia i80x86**

En los procesadores de la familia Intel, la secuencia del movimiento del dato a la pila y la actualización del SP es la siguiente.

PUSH

* 1° -> decrementar SP
* 2° -> guardar (“apilar”) el dato

Es decir, primero se decrementa el S P, y luego se transfiere el dato al “tope” de la pila (lugar a donde “apunta” el SP).Así, los datos se van almacenando en posiciones decrecientes de memoria. La pila “crece” hacia posiciones de memoria inferiores.

POP

* 1° -> extraer (“desapilar”) el dato
* 2° -> incrementar el SP

Es decir, primero se extrae el dato desde el “tope” de la pila (lugar a donde “apunta” el SP), y luego se incrementa el SP. La operación de POP es exactamente inversa de la de PUSH. El SP queda apuntando a la dirección donde está el último dato guardado (tope de la pila), el dato extraído permanece en la memoria pero no forma parte de la estructura de datos de la pila (copia no válida)

**Manejo de la Pila en procesadores de la familia M68xxx**

Algunos procesadores administran la pila de una manera distinta. Por ejemplo, en los procesadores de la familia Motorola, la secuencia del movimiento del dato a la pila y la actualización del SP es la siguiente.

PUSH:

* 1° -> guardar (“apilar”) el dato
* 2° -> decrementar SP

POP:

* 1° -> incrementar el SP
* 2° -> extraer (“desapilar”) el dato

ahora el SP apunta siempre a la primer posición libre, por encima del tope de la pila.

**Instrucciones para la PILA en Lenguaje Assembly**

Para operar con la Pila, los procesadores disponen de instrucciones de carga (PUSH) y descarga de datos (POP).

Subrutinas

Una subrutina es una sección de código, que recibe el control en un punto de entrada, y lo devuelve en un punto de salida. El objetivo de la subrutina es realizar una tarea definida, para lo cual se le transfiere el control. Una vez finalizada la tarea lo devuelve al programa que la invocó en el punto donde la invocó.

El uso de subrutina tiene muchas ventajas.

* Economía de programa: el código puede ser usado varias veces
* Modularidad: se puede subdividir el programa en unidades pequeñas, más fácilmente verificables.

Una subrutina se invoca mediante una instrucción específica, disponible en el repertorio de instrucciones del procesador.

Por ejemplo, un mnemónico típico usado para invocar una subrutina es la instrucción: CALL

La subrutina termina mediante una instrucción específica de retorno. Por ejemplo: RET

El programa principal invoca la subrutina en la instrucción CALL Nombre (del procedimiento). Al invocarla, le transfiere el control. Es decir, el PC se carga con la dirección de comienzo de la subrutina, y la CPU comienza a ejecutar las instrucciones de la subrutina.

Cuando la subrutina complete su tarea, la última instrucción que ejecuta es la de RET. En ese momento el PC se carga con la dirección de la instrucción siguiente al CALL en el programa principal. De esa manera el programa principal continúa con su tarea en el punto siguiente a la invocación de la subrutina.

Para poder recuperar la dirección de la instrucción siguiente al CALL en el programa principal, esta dirección debe haber sido guardada previamente. El lugar donde se guarda esta dirección es la Pila. Cada vez que se ejecuta una instrucción de CALL, se guarda en la Pila el valor actual del PC. Cuando se ejecuta una instrucción R E T, el PC se carga con la dirección almacenada en el tope de la Pila.

Mediante este mecanismo, al terminar la subrutina, se restablece en el PC la dirección de la instrucción siguiente al CALL.

**Detalles de uso de subrutinas**

Para que una subrutina realice una tarea, requiere de datos. Los datos (argumentos o parámetros) se los proporciona el programa que invoca a la subrutina.

A su vez, la subrutina produce 1 o más resultados. Los resultados deben ser devueltos al programa que invocó la subrutina.

Existen 3 métodos para pasar parámetros a subrutinas.

1. **Vía registros**

Los parámetros se pasan a través de los registros de la CPU.Es un método sencillo, pero limitado por el número deregistros disponibles. **D**ado que se van a modificar los contenidos de los registros,es importante documentar los registros a usar.

1. **Vía memoria**

Los parámetros se transfieren a través de un área definidade memoria (RAM).Difícil de estandarizar, debido a las dificultades en asignarun área de memoria.

1. **Vía pila (stack)**

Los datos se pasan a través de la Pila. Es el método más ampliamente usado porque presenta varias ventajas respecto de los otros métodos.

La principal ventaja es que es independiente de la memoria y registros.

Los registros no tienen que ser modificados en las subrutinas, pero hay que manejar correctamente, porque la Pila va a ser usada por el usuario (en la invocación y en la subrutina) y por el sistema.

Programa principal con llamada a Subrutina

Para invocar una subrutina con pasaje de parámetros vía Pila, en el programa que invoca se hace lo siguiente:

Push Parámetro 1

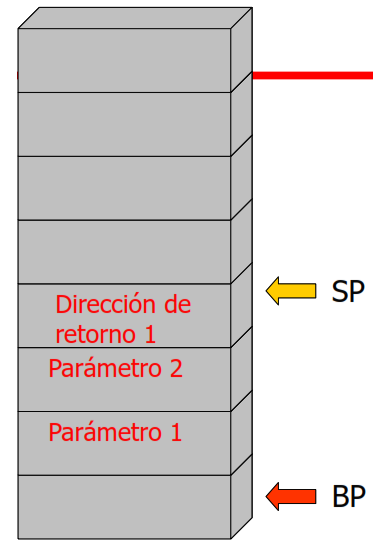
Push Parámetro 2

CALL SUB1 (del procedimiento)

DirRet Prox. Instrucción

Previo a la invocación de la subrutina (es decir previo al CALL) se deben apilar los parámetros requeridos por la subrutina (en el ejemplo, se apilan 2 parámetros).

Tener en cuenta que las 2 instrucciones de Push modifican el SP. El programa principal deberá restablecer a partir de Prox. Instrucción (luego del retorno de la subrutina) el valor correcto del SP (por ejemplo con 2 instrucciones POP).



Para el acceso a la estructura de datos de la Pila se requiere un registro base (dado que no se puede usar el SP), típicamente el BP.

La subrutina no debería modificar los contenidos de los registros al momento de su invocación. Si la subrutina usa un registro lo debería salvaguardar al comienzo, y restaurarlo al final.

**Estructura de la subrutina**

1. Salvar el estado de BP (viejo BP). (BP es BX)

2. Apuntar con BP al tope de la pila (BP=SP)

3. Reservar espacio para datos locales (opcional)

4. Salvar valores de otros registros (opcional)

5. Acceder a parámetros

6. Escribir sentencias a ejecutar cuerpo

7. Restaurar los registros modificados (opcional)

8. Restaurar el valor original de SP y BP

9. Retornar parámetro (opcional)

10. Regresar (correctamente del procedimiento)

1. Salvar el BP (viejo): push BP

2. Inicializa BP: mov BP, SP

Como se modificará el BP, siguiendo el concepto de que los registros no deben ser alterados al finalizar la subrutina, la primera instrucción tiene como objetivo salvaguardar el valor actual del BP.

Una vez hecho esto, la segunda instrucción inicializa el puntero BP para que apunte a la estructura de datos que se formará en la Pila.

SP no puede ser usado para éste propósito porque no es un registro base ó índice(es decir, no hay un direccionamiento base o indexado que use el SP). El valor de SP puede cambiar, pero típicamente BP permanece 'quieto'.

3. Reserva espacio para

variables locales sub SP,2

El tercer paso en la subrutina consiste en reservar espacio para almacenar variables locales (variables temporales usadas por la subrutina y que al finalizar se deben descartar). Eso se puede hacer, por ejemplo moviendo el SP una cantidad determinada de bytes. Por ejemplo, se reservan 2 bytes para variables locales, decrementando el SP en 2.

El sistema puede utilizar al SP sin escribir sobre el área de trabajo (o frame) del procedimiento.

1. Salva otros registros usados en la subrutina

push DI

En el caso que la subrutina necesite usar registros de la CPU, es aconsejable salvarlos antes, para restablecerlos previo a retornar de la subrutina.

En el paso 4 se salva el ó los registros que sean necesarios.

En este ejemplo se salva el registro DI. No necesitan salvarse los registros que no son modificados. Normalmente los registros son salvados después deestablecer el puntero (frame pointer) y reservar el área para los datos locales.

1. Acceder a un parámetro

mov CX, [ BP + cantidad de veces que se incremento el SP ]

En general el acceso a un parámetro se hace sumando un desplazamiento a BP (BP no se modifica), correspondiente a un direccionamiento base con offset.

El offset que se suma al BP se obtiene sumando todos los bytes intercalados entre la posición actua del BP y la posición del parámetro a buscar.

1. Cuerpo de la subrutina

….

….

En el paso 6 se escriben las instrucciones propias de la tarea que desarrolla la subrutina, y que constituyen el cuerpo del procedimiento.

1. Restaurar los registros modificados

pop DI

En el paso 7 se restauran los valores originales en los registros usados durante la subrutina. En el ejemplo, solo se usó el DI (que fué salvado en el paso 4), por lo que solo ese registro es recuperado.

1. Restaurar SP y BP

mov SP,BP  
pop BP

La primer instrucción (mov SP, BP) restablece el valor de SP que pudo haberse movido cuando se reservó espacio para las variables locales (paso 3). Notar que el BP no se movió durante el acceso a la estructura de datos en la Pila.

La segunda instrucción (pop BP), restaura el valor original (el valor “viejo”) que tenía el BP antes de ingresar a la subrutina.

1. Retornar parámetros  
   ….  
   ….

En el paso 9 se actualizan los resultados obtenidos por la subrutina a ser devueltos al programa que la invoca.

1. Retornar  
   ret

El ultimo paso de la subrutina es la instrucción de retorno de subrutina Ret. De esta manera se le devuelve el control al programa principal (que invoca la subrutina) en la instrucción siguiente a la instrucción que invoca a la subrutina (CALL).

Resumiendo, las subrutinas realizan un trabajo determinado. Ese trabajo puede producir un resultado, que debe ser devuelto al programa que la invoca.  
Típicamente la subrutina devuelve 1 o más resultados. El resultado se retorna por las mismas posiciones de memoria donde la subrutina recibió los parámetros.  
En el ejemplo, el resultado debe guardarse en la dirección de memoria con un offset de 6 respecto del valor actual de B P, es decir: BP+6

Notar que en BP+2 está la dirección de retorno que NO debe ser alterada de ninguna manera.

**Anidamiento de subrutinas**

Una subrutina puede invocar a otra subrutina. Esto se conoce como “anidamiento de subrutinas”. Cada subrutina tiene asociado su propio espacio de memoria en la Pila, tal como se vio anteriormente.  
La cantidad de anidamientos posibles de subrutinas depende de varios factores, pero básicamente del espacio de memoria para disponer de la estructura de datos de cada subrutina. Es por eso que la cantidad de anidamiento es muy grande (casi “infinita”)

Dado que la información se guarda en una estructura tipo LIFO, cada vez que se termina una subrutina de un nivel de anidamiento dado, se retorna a la subrutina del nivel anterior. En otras palabras, cuando se termina una subrutina se devuelve el control a la subrutina que la invocó.

****

En el ejemplo que sigue se muestra un programa (principal) PPIO que comienza en la dirección 1000. El PPIO invoca una subrutina SUBR1 (nivel de anidamiento 1) que comienza en la dirección 3000H.

La subrutina SUBR1 invoca a subrutina SUBR2 (nivel de anidamiento 2) que comienza en la dirección 4000H.

Cuando termina SUBR2, de forma correcta, retorna a la tarea que la invocó, es decir ejecuta en SUBR1 la instrucción siguiente del CALL SUBR2 (es decir ejecutará POP BX).

Cuando termina SUBR1, retorna a PPIO, a la instrucción siguiente del CALL SUBR1 (es decir ejecutará POP BX).

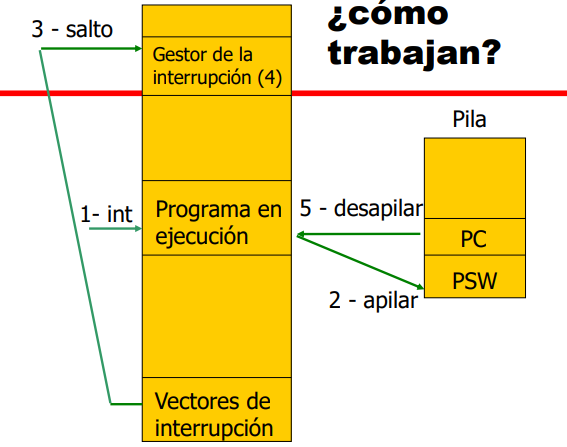
**Clase 2 – Interrupciones**

Una interrupción es un mecanismo que permite alterar ese proceso de “ejecución normal” de la CPU. Este mecanismo permite que la CPU suspenda la tarea que está haciendo y responda a una solicitud de atención para resolver (ejecutar) otra tarea (servicio de la interrupción).

Una vez completado el servicio de la interrupción, el procesador retoma la tarea suspendida, en el punto donde se detuvo (de una manera similar al llamado a subrutina).

De acuerdo al concepto de interrupción, el procesador conmuta de una tarea en ejecución a otra a ejecutarse por efecto de la presencia de un evento (la interrupción). Por lo tanto se requiere de 3 acciones:

1) Detener (suspender) la tarea que está ejecutando el procesador (suspender, no terminar ni abortar).  
2) Bifurcar (saltar) a otra tarea, asociada a la solicitud de interrupción (comúnmente conocida como servicio interrupción).  
3) Restablecer la tarea suspendida en las condiciones en las que se encontraba en el momento en el que se la detuvo.



La forma en que opera la interrupción se puede ver en la figura anterior.

1.- La CPU recibe, mientras está ejecutando una tarea (el programa en ejecución), un pedido de interrupción.

2.- La CPU salva todo o parte del estado de la CPU correspondiente a la tarea a ser suspendida. Al menos salva el Contador de programa (PC) y el registro de estado (PSW), típicamente en la Pila del sistema . Esto lo hace porque lo necesita para restaurar la tarea suspendida.

3.- La CPU busca, en un área de memoria definida, la dirección de comienzo (el “vector”) del servicio de la interrupción, y comienza a ejecutar dicho servicio.

4.- Cuando la CPU termina el servicio de la interrupción, tiene que retornar al programa interrumpido. Esto se hace mediante unainstrucción especial de Retorno de interrupción (RTI).

5.- La ejecución de la instrucción RTI desapila exactamente lo apilado cuando atendió la interrupción (es decir, todo o parte del estado de la CPU). Como mínimo recupera el PC y el PSW. De esta manera retoma la tarea suspendida (elprograma en ejecución) en el punto en que fue interrumpida.

Características de las interrupciones

El origen de una interrupción es la ocurrencia de un evento que requieren la intervención de la CPU.

Existen 2 tipos de eventos:

* Interno: es decir, debido a la ocurrencia de una situación dentro del Sistema de cómputo. Algunos eventos internos que pueden producir interrupción son: un error asociado a la ejecución de una instrucción, desbordamiento aritmético (“overflow”), división por cero, temporizados propios del sistema, fallo del hardware, error de paridad en la memoria, pérdida de energía.
* Externo: asociado a operaciones de E/S con periféricos. Algunos eventos típicos que pueden generara interrupción son: finalización de una transferencia, error en la transferencia, dispositivo indisponible, etc.

**Interrupciones múltiples**

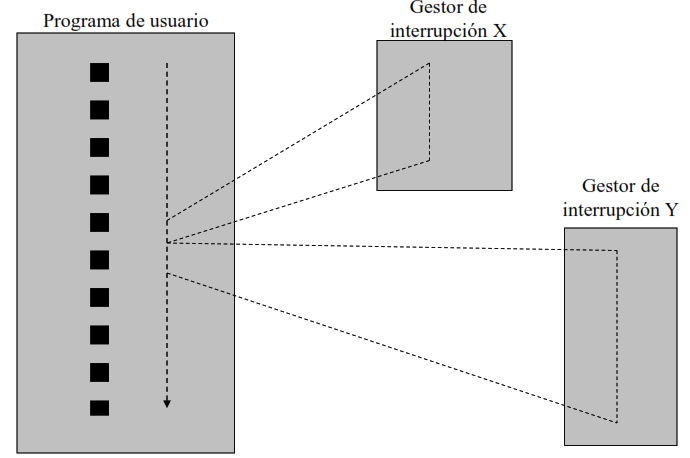
La necesidad de administrar eventos de distinto origen requiere, en la mayor parte de los casos, administrar varias interrupciones. Dado el origen diverso de las interrupciones, hay algunas que son más importantes que otras.

En general, los procesadores son capaces de manejar varias interrupciones de distinta características y orígenes.

Prioridades

Las interrupciones más importantes deben tener mayor “prioridad” que las menos importantes. Cuanto mayor sea su prioridad, mayor es la urgencia para ser atendida, incluso si hay una interrupción en curso.

Si las interrupciones son de igual prioridad, se procesan en el orden en que llegan. Si las interrupciones tienen distinta prioridad, las interrupciones de mayor prioridad pueden interrumpir a las de menor prioridad. La inversa no vale.



**Procesamiento de interrupciones de igual prioridad**

Las interrupciones se atienden según el orden de llegada. Cuando llega una interrupción y es atendida, se inhabilita el resto de las interrupciones de igual o menor nivel de prioridad.

Si llega una nueva interrupción quedará pendiente. El procesador ejecutará el servicio de la interrupción atendida. Al finalizar el servicio se habilitarán nuevamente las interrupciones. Y la interrupción pendiente será atendida.

**Procesamiento de interrupciones de distinta prioridad**

Una interrupción de prioridad más alta puede interrumpir en cualquier momento a una interrupción de prioridad menor. Cuando se ha gestionado la interrupción de prioridad más alta, el procesador vuelve a las interrupciones previas (de menor prioridad).

Terminadas todas las rutinas de gestión de interrupciones se retoma el programa del usuario.

Tipos de interrupciones

**Interrupciones enmascarables y no-enmascarables**

Los sistemas de cómputo pueden responder de manera diferente a los distintos tipos de interrupciones, principalmente asociado a la prioridad que tienen. Típicamente tienen 2 categorías de interrupciones:

* No Enmascarables: son interrupciones que no pueden ser ignoradas, es decir, se atienden indefectiblemente, y están asociadas a eventos críticos, peligrosos o de alta prioridad.
* Enmascarables: pueden ser, eventualmente, “ignoradas”. Para ello el procesador permite realizar algunas acciones que pueden inhibir la atención de la interrupción. Las interrupciones enmascarables generalmente están asociadas a operaciones menos críticas, por ejemplo de E/S.

**Interrupciones por hardware y por software**

Desde el punto de vista de la forma en que se invocan, las interrupciones pueden ser 2 de tipos:

* Por Hardware: son generadas por señales físicas asociadas a eventos externos o internos. De acuerdo al origen de la señal de interrupción, pueden ser: Externas e Internas
* Por Software: son producto de la ejecución de instrucciones con efecto similar a una interrupción por hardware.

**Interrupciones por hardware externas**

Son conocidas como Interrupt request. El origen de estas señales de pedido de interrupción proviene típicamente de dispositivos conectados al subsistema de E/S. Se consideran las “verdaderas” interrupciones porque son aleatorias en relación al proceso en ejecución (es decir, pueden ocurrir en “cualquier instante de tiempo”). El sistema de cómputo tiene que ser capaz de manejar estos eventos externos “no planeados” ó “asincrónicos”. Pueden o no estar relacionadas con el proceso en ejecución en ese momento.

**Interrupciones por hardware internas**

Son conocidas como Trap o excepciones. Son creadas dentro del sistema de cómputo en respuesta a situaciones propias del proceso en ejecución y no vinculadas con operaciones de E/S, por tal motivo no son estrictamente aleatorias.

Algunos eventos que pueden producir un Trap son:

* Condiciones excepcionales: overflow en ALU de punto flotante.
* Fallas de programa: tratar de ejecutar una instrucción no definida.
* Fallas de hardware: error de paridad de memoria.
* Accesos no alineados o a zonas de memoria protegidos

**Interrupciones por software**

Son conocidas como Software interrupt. Son instrucciones explícitas que tienen un efecto similar a una interrupción por hardware.

Como normalmente el SO administra los servicios de las Interrupciones, las Interrupciones por software permiten invocar los servicios del SO asociados a las Interrupciones en otras palabras, son “llamadas” a funciones del SO).

El SO define los lugares donde se cargan los servicios de las Interrupciones. El usuario no conoce, a priori, esos lugares, pero usa los servicios invocándolos a través de las interrupciones que maneja el SO.

Hay sistemas que no permiten hacer una llamada directa a una función del SO, por estar en una zona reservada. Si no se pudieran usar las interrupciones por software como mecanismo de invocación de esos servicios que administra el SO, cuando se necesita administrar una tarea por interrupción, se debería:

Opción 1: escribir el servicio (bastante complicado)

Opción 2: el programa que requiere el servicio debería buscar entre todas las llamadas a funciones del BIOS y SO la que necesita, y reemplazar en el código la dirección de esa función invocada (también muy complicado).

Tratamiento de las interrupciones

**Gestión de las Interrupciones.**

El uso de interrupciones requiere de la gestión ordenada de las siguientes acciones básicas:

1. Detectar el pedido de interrupción

2. Detener la tarea que se estaba ejecutando

3. Salvar el estado de la tarea que se estaba ejecutando

4. Obtener la dirección de comienzo del servicio de la interrupción y bifurcar a dicho servicio

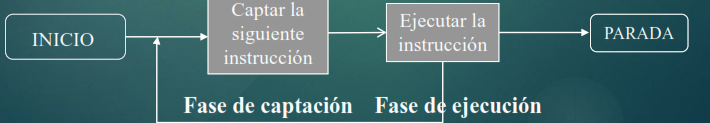
5. Ejecutar el servicio de la interrupción

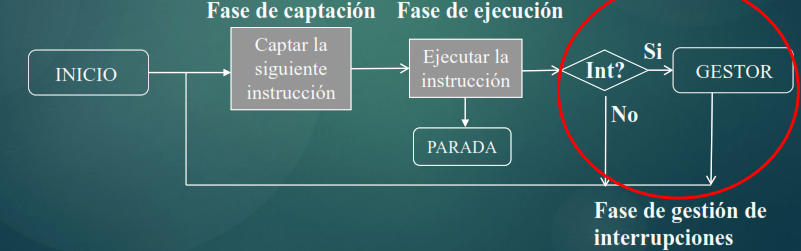
6. Retornar y restaurar el estado en que estaba la tarea interrumpida.

7. Continuar con la ejecución normal de la tarea interrumpida (en el punto en el que se detuvo)

**Detección del pedido de Interrupción.**

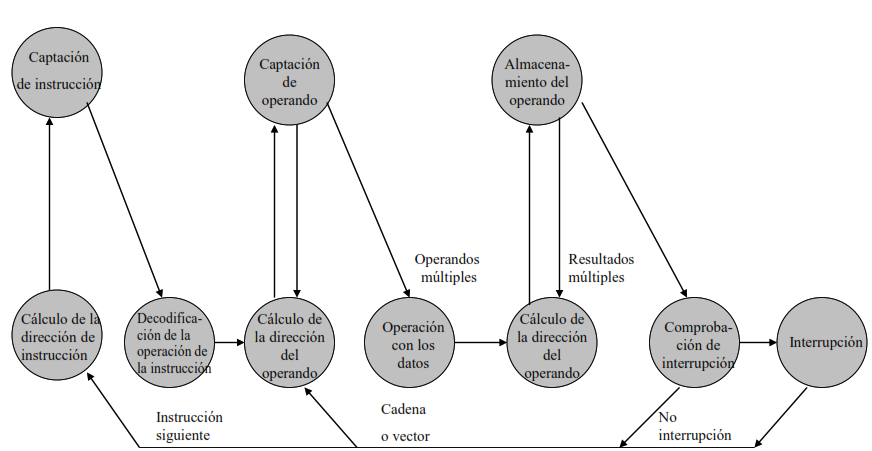
El primer paso en la administración de las interrupciones consiste en detectar el pedido de atención. Para ejecutar esta operación, el procesador examina, en cada ciclo de instrucción, la presencia de interrupciones. De acuerdo a lo visto previamente, el ciclo de instrucción podía interpretarse como un bucle de ejecución interna, infinito, compuesto por 2 fases: búsqueda de la instrucción y ejecución.



Para poder implementar la tarea de detección de interrupciones, en el ciclo de instrucción anterior se agrega, además de las fases de captura y ejecución, la fasede gestión de interrupciones.

La etapa de gestión de interrupciones en el ciclo de instrucción, determina la presencia o ausencia de pedido de interrupciones.

La presencia de un pedido de interrupción se manifiesta mediante una o más señales discretas (bits) comúnmente llamadas bandera (o “Flag”) que la CPU examina.El estado de estos flags asociados a interrupciones están en algún registro (especial) de la CPU. Dependiendo del estado 0 o 1 del flag (es decir, del pedido de interrupción) se tienen 2 posibles caminos.



**Almacenamiento del proceso a ser interrumpido**

Si no hay pedido pendiente (Flag inactivo) se inicia el ciclo de captación de la siguiente instrucción (proceso “normal” de ejecución).

Si hay algún pedido de interrupción pendiente, el procesador guarda en la pila del Sistema, el “estado del proceso”. Existen 2 estrategias de guardado del estado del proceso:

1. Guardar solo la próxima instrucción a ejecutar y algún registro crítico (por ejemplo el registro de estado).
2. Guardar todos los registros del procesador.

El objetivo de esta operación es el de restablecer el estado del procesador al terminar el servicio de la interrupción.

**Bifurcación al servicio de la interrupción**

Obtiene la dirección donde comienza la rutina de la interrupción y carga el PC con este valor, bifurcando de esta manera, al servicio de la interrupción.

Existen varias técnicas para obtener la dirección donde comienza el servicio de la interrupción. En general se dispone de un área de memoria reservada, donde están estas direcciones. Tener en cuenta que son varias direcciones porque el procesador es capaz de atender varias interrupciones, y habrá una dirección por interrupción. Esta área de memoria se llama área de vectores de interrupciones.

**Detección del pedido de múltiples interrupciones**

Dentro del proceso de gestión de las interrupciones, lo primero que tiene que hacer la CPU es detectar el pedido de interrupción. Cuando hay múltiples fuentes de interrupción, hay varias formas para identificar el origen del pedido. Los más comunes son:

Opción 1: 1 señal física de entrada a la CPU por cada Interrupción.

Opción 2: 1 única señal física de entrada a la CPU para todas las interrupciones e identificación por software.

Opción 3: 1 única señal física de entrada a la CPU para todas las interrupciones e identificación por hardware.

**Opción 1:** Se tiene 1 señal física de entrada a la CPU por cada interrupción, es decir que hay múltiples líneas de pedido de interrupción en la CPU. Dado que disponer de líneas en la CPU para interrupción es costoso, la cantidad de señales para ser usadas en interrupciones se acota normalmente a un número reducido (por ejemplo 3 o 4).

Cada dispositivo que puede provocar interrupción tiene una entrada física de interrupción conectada directamente a la CPU. La implementación es bastante sencilla pero esta restringida por la cantidad de líneas disponibles en la CPU.

**Opción 2**: Hay 1 sola entrada física de pedido de interrupción a la que están conectados todos los dispositivos. Para poder identificar la fuente de la solicitud, la CPU debe “preguntar” a cada dispositivo si ha producido el pedido de interrupción. Este método de consulta se conoce como encuesta o polling, y es básicamente un programa que ejecuta la CPU dentro de los servicios de interrupción. Al ser un esquema de detección por software, el tener que consultar uno por uno a todos los dispositivos lo hace relativamente lento e ineficiente.

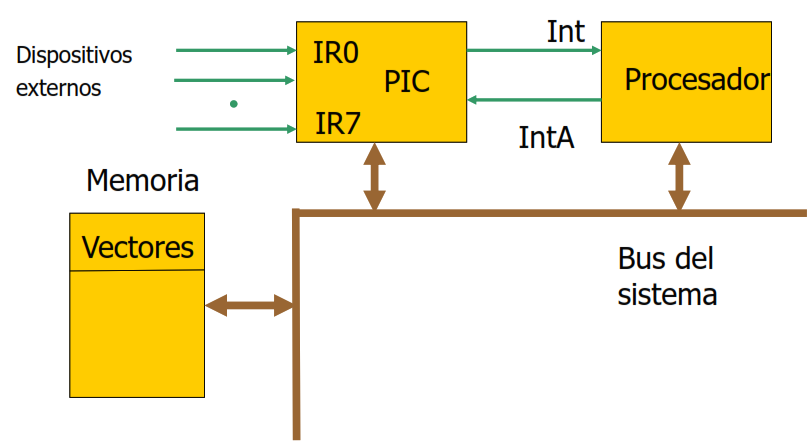
**Opción 3**: Hay 1 sola entrada física de pedido de interrupción a la que están conectados todos los dispositivos. Para poder identificar la fuente de la solicitud, la CPU recibe a continuación, típicamente a través del bus de datos, un número que identifica la fuente de la interrupción (conocido como “vector de la interrupción”).

El vector es provisto por el periférico que generó el pedido, o por algún dispositivo que se ocupe de generar el número dependiendo de la interrupción a ser atendida.

La opción 3 se conoce como Interrupciones vectorizadas.

El escenario que se tiene es el siguiente:  
El procesador tiene una única entrada de pedido de interrupciones.

Hay varios “productores” de interrupciones y un “dispositivo especial” administra las necesidades propias de la interrupción. En la familia Intel, este dispositivo se conoce como Controlador Programable de Interrupciones (‘PIC’). El PIC se encarga, entre otras cosas, de generar el vector, administrar prioridades, habilitar interrupciones, etc.

****

Controlador de interrupciones PIC

**Interrupciones vectorizadas con el PIC**

El PIC recibe los pedidos de interrupción, típicamente de periféricos que piden atención y solicita atención a la CPU con la única señal de pedido de interrupción IntR (Interrupt request)

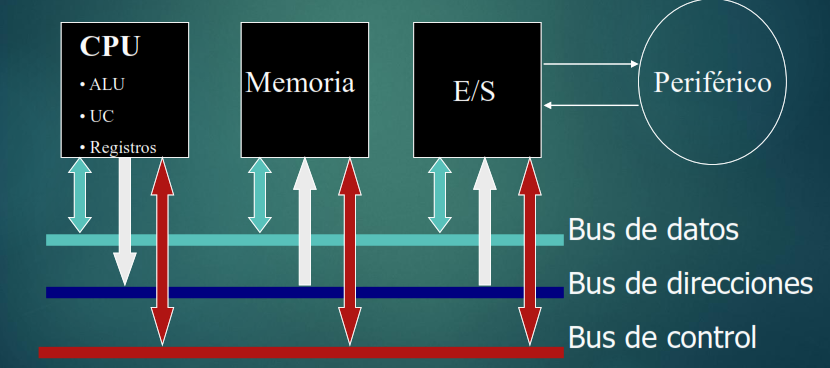
Cuando la CPU está lista para atender la interrupción, le avisa al PIC mediante la señal IntA (Interrupt acknowledge).

El PIC genera en el bus de datos el número de la interrupción (vector) a ser atendida. La CPU lee ese número y busca en la memoria el vector correspondiente al servicio de esa interrupción. Dado que se hace por hardware, es mucho más rápido.

El PIC internamente tiene 3 registros principales:

* El ISR que identifica la interrupción en servicio
* El IRR que contiene los pedidos de interrupción provenientes de los periféricos.
* El IMR que se usa para habilitar/deshabilitar los pedidos de interrupción. Esta funcionalidad se conoce como enmascaramiento de interrupciones.

**Clase 3 – Subsistema E/S**

el modelo de Arquitectura de von Neumann basado en 3 subsistemas: CPU, Memoria y E/S.

comprende los dispositivos que están conectados al bus del sistema y proveen los servicios de transferencia de datos con los Periféricos.

Debido a la gran variedad de Periféricos con los que se requiere intercambiar información, el subsistema de E/S tiene que ser lo suficientemente flexible para permitir:

* Trasmisión de diferentes cantidades de datos
* Rango de velocidades de transmisión muy amplio.
* Diferentes formatos de dato y tamaño de palabra.

En general, todos los Periféricos son más lentos que la CPU y la Memoria. Los dispositivos que forman parte del subsistema de E/S permiten descongestionar el trabajo de la CPU.

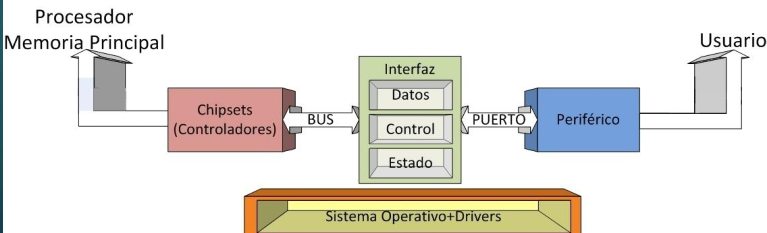
Dispositivos Periféricos:

Existe una variedad muy grande de equipos periféricos. Algunos de los tipos más comunes son:

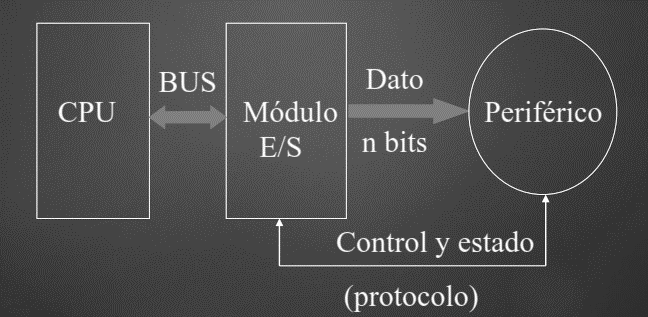
* Comunicación hombre-máquina: monitor/pantalla, mouse, teclado
* Almacenamiento: Disco duro, cd, dvd.
* Impresión: impresora, scanner.
* Comunicación con dispositivos remotos: modem, placa de red.
* Multimedia: microfono, partalnes.
* Automatización y control: sensores, alarmas, adquisición de datos.

Módulo de interfaz de E/S

Son los dispositivos más sencillos para implementar las transferencias de E/S con periféricos. Los puertos E/S conectan la interfaz entre el procesador/memoria y un periferico.  
Son adminsitrados por el SO a través de drivers específicos.



El esquema básico de transferencia de datos usando un Módulo de E/S sencillo es el que se muestra en la figura siguiente.



Se puede observar el Módulo de E/S comunicado con la CPU a través del Bus del Sistema.  
La conexión con el periferico provee dos tipos de informaciones:

1. Datos: Informacion útil a transferir.
2. Control y estado: información que permite realizar la transferencia (en lo posible libre de errores). Por ejemplo: sentido de la transferencia (de entrada, de salida), operación (de lectura, de escritura), estado del periférico (listo, no-listo, en falla).

Puede ejecutar dos tipos de comunicaciones:

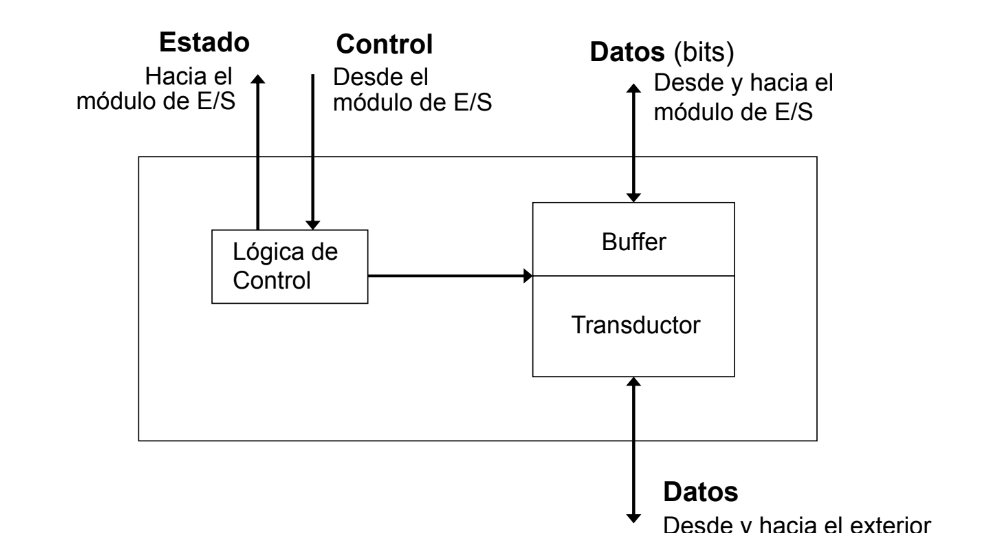
1. Hacia el Periferico:

* Transferir datos con el periferico.
* Controlar y temporizar uno o mas dispositivos externos.
* Almacenar temporalmente datos (‘buffer’)
* Detectar errores.

1. Hacia el Bus ( internas con cpu y memoria):

* Interpretar ordenes que recibe de CPU y transmitirlas al periférico.
* Transmitir datos con la CPU (registros) y Memoria.
* Informar a la CPU del estado del periferico.

Esquema simplificado de un Periferico Conectado:



Se puede apreciar que un periférico tiene dos bloques funcionales que manejan la comunicación con el modulo E/S:

1. Hacia el exterior:

* Sección de manipulación de datos (buffer/ trasductor). Almacena y convierte los datos a intercambiar con el periférico
* Sección de control y estado (lógica de control): recibe y genera señales de control y estado de periféricos.

1. Hacia Modulo E/S:

* Sección de manipulación de datos.
* Sección de manipulación de señales de control y estado

Diagrama

Descripción generada automáticamenteEsquema detallado de un modulo E/S: Lo amarillo es visible al programador

En el esquema anterior se pueden apreciar los bloques funcionales que manejan la comunicación hacia el periférico y hacia el bus del sistema.

Hacia el sistema los recursos que son “visibles” a la CPU (y al programador) son básicamente registros. Existen 2 tipos de registros:

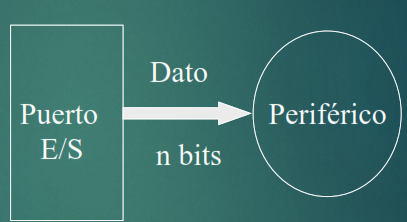
* Registros de datos: que contienen la información útil recibida desde, o transmitida a, el periférico.
* Registros de control y estado: que controlan las características de la transferencia, y almacenan señales de estado de la comunicación (falla, no listo, etc.).

Puertos de E/S

Existen dos tipos Basicos:

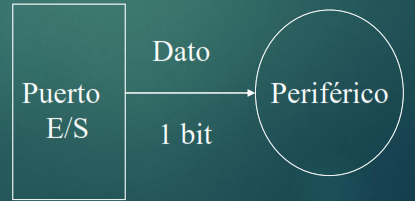
* En Paralelo: hay varias líneas de datos (n) que transfieren n bits simultáneamente entre el puerto de E/S y el periférico. Como por ejemplo las impresoras paralelo y scanner.

Se requiere disponer de una conexión mediante un cable que incluya al menos los n bits de datos, lo que lo hace un método bastante costoso.



* En Serie: hay 1 línea de dato para la transferencia entre el puerto de E/S y el periférico. Ejemplos: impresora serie, red Ethernet, mouse, teclado, etc.

Se requiere disponer de una conexión mediante un cable sencillo. El costo es mucho menor. Los datos deben serializarse (transmitirse de 1 bit por vez, uno a continuación del otro) lo que en teoría sería mucho más lento que transmitir de a n bits simultáneamente, como lo hace el puerto paralelo.



CPU y E/S

Las Puertas de E/S son un tipo particular de dispositivo de E/S. Hay otros dispositivos más complejos que proveen de otras prestaciones, además de la tarea básica de implementar la transferencia de los datos, por ejemplo:

* Ocultar las propiedades particulares del dispositivo periférico a la CPU: temporizados, formatos, electro mecanismos, etc.
* Manejar múltiples dispositivos simultáneamente.
* Controlar varias funciones del dispositivo.

Registros de un Puerto de E/S

Desde el punto de vista de la CPU, una operación de E/S requiere acceder a los registros internos del Módulo de Interfaz de E/S. Los registros pueden ser de lectura y/o escritura.

Dentro de Módulo de E/S hay 2 tipos de registros:

* De DATOS: interviene en la transferencia de entrada o de salida del dato a intercambiar entre el sistema de cómputo y el periférico.

La transferencia del dato entre el sistema de computo y el periférico consiste en:

* + Operaciones de entrada: lectura de un registro de dato (es decir un registro escrito por el periférico y leído por el CPU)
  + Operaciones de salida: escritura de un registro de dato (es decir, un registro escrito por la CPU y leído por el periférico)
* De CONTROL: registros que controlan el funcionamiento del módulo, la transferencia y el periférico.  
  La supervisión y/o control de una transferencia requiera de:
  + Control: adecuar la configuración del módulo para ajustar formatos, sincronizaciones, etc.
  + Estado: registrar el estado operativo del módulo y del periférico. -

Acceso al subsistema E/S

Desde el punto de vista de la CPU, el Subsistema de E/S está compuesto por un conjunto de registros a los que accede para una operación de entrada o de salida.

Existen 2 técnicas de acceso a estos registros:

* Espacio de E/S compartido con memoria.
* Espacio de E/S separada de memoria.

Espacio de E/S compartido con memoria (memory-mapped)

En esta técnica los registros de los dispositivos de E/S y memoria comparten un único espacio de direcciones.

Los registros de la E/S se comportan idéntico a una memoria de lectura/escritura. No hay instrucciones especificas para E/S, se usan las mismas instrucciones de movimiento de datos a memoria. Ej: MOV red-dato, AL.

Espacio de E/S separado de memoria (modelo Intel)

En esta técnica los registros de los dispositivos de E/S y la memoria están en diferentes espacios de direcciones. Dado que el bus de direcciones es compartido por la memoria y el subsistema de E/S, se requieren señales de control adicionales para identificar a donde está accediendo la CPU: a la memoria, o a la E/S. Hay instrucciones específicas de E/S, distintas de las instrucciones de acceso a la memoria.

Cuando se ejecutan estas instrucciones específicas, en el bus de control se identifica el acceso al mapa de direcciones de E/S. Para el resto de las instrucciones en el bus de control se identifica el acceso a la Memoria.

Ejemplos:

**1) Entrada:** IN dest, fuente

Donde dest es AL o AX (8 o 16 bits) y fuente un número de 8 bits sin signo (0 y 255) o DX (número entre 0 y 65535).

**2) Salida:** OUT dest, fuente Donde fuente es AL o AX (8 o 16 bits) y dest un número de 8 bits sin signo (0 y 255) o DX (número entre 0 y 65535).

GESTION DE LA TRANSFERENCIA

Existen tres tipos de estrategias básicas para transferir datos entre el sistema de cómputo y el periférico:

* E/S programada y espera de respuesta
* E/S programada y administrada por interrupción
* E/S con acceso Directo a memoria (DMA)

Las dos primeras requieren intervención directa de la CPU, es decir que la CPU participa en la transferencia de todos los datos a transferir (byte o Word).

E/S Programada y administrada por interrupción

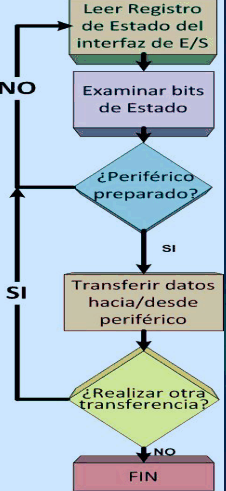
Es decir que la CPU tiene control casi directo sobre la operación de E/S. Entre otras acciones hace:

* Comprueba el estado del dispositivo
* Envía los comandos requeridos
* Realiza la transferencia de todos los datos (de a uno)

En cada dato que es transferido, la CPU espera que el módulo E/S termine la operación, típicamente que el periférico “acepte” el dato. Durante la espera, la CPU permanece ociosa (no deseable).

La secuencia de acciones que ejecuta la CPU es:

1. La CPU verifica el estado de periférico (preparado/no-preparado) leyendo un registro del módulo de interfaz.
2. Examina el estado del periférico chequeando el bit (o bits) que identifican dicho estado.
3. Si el dispositivo no está listo (por ejemplo bit=0), la CPU vuelve al paso 1. Este lazo significa que la CPU “espera” hasta que el periférico se pone en “preparado”, es decir, listo para la transferencia.
4. Cuando el dispositivo está listo, la CPU transfiere 1 dato hacia o desde el módulo de interfaz.
5. Si hay más datos que transferir vuelve al paso 1.
6. Si se completó la transferencia, termina el servicio de E/S.



E/S Programada y administrada por interrupción

Cada vez que el módulo está listo (o completó una transferencia) avisa a la CPU con un pedido de interrupción. Ahora la CPU no tiene que ejecutar el lazo de comprobación del estado del módulo (pasos 1, 2 y 3 de la igura anterior). Solo inicia la transferencia al recibir el pedido de interrupción del periférico.

Durante el tiempo que el periférico no está listo, la CPU no tiene que esperar, puede seguir ejecutando otra tarea.

La secuencia de acciones para una transferencia de entrada de 1 dato (por ejemplo, 1 carácter) administrado por interrupción es:

* La CPU inicia la operación de lectura (entrada) enviando una orden de lectura (READ) al módulo de E/S.
* El módulo de E/S solicita el dato al periférico.
* El periférico busca el dato, mientras la CPU continúa con sus tareas.
* Cuando el módulo E/S tiene el dato enviado por el periférico emite un pedido de interrupción a la CPU.
* La CPU detecta el pedido, interrumpe el proceso, y bifurca al servicio de la interrupción.
* Durante la interrupción, la CPU lee el dato desde el módulo de E/S.
* La CPU retorna a la tarea interrumpida.

Interrupciones en operaciones de E/S

Cuando hay varios dispositivos periféricos, con esta forma de administración de las transferencias de E/S se requiere poder identificar la fuente de interrupción.

Como se vio durante el análisis de Interrupciones, existen varias estrategias distintas para identificar la fuente de la interrupción:

1. Diferentes líneas de interrupción
2. Una sola línea de interrupción y encuesta por software
3. Una sola línea de interrupción con conexión en cadena (Daisy chain) tipo “hard poll” (encuesta por hardware)
4. Una sola línea de interrupción y vectorizado.

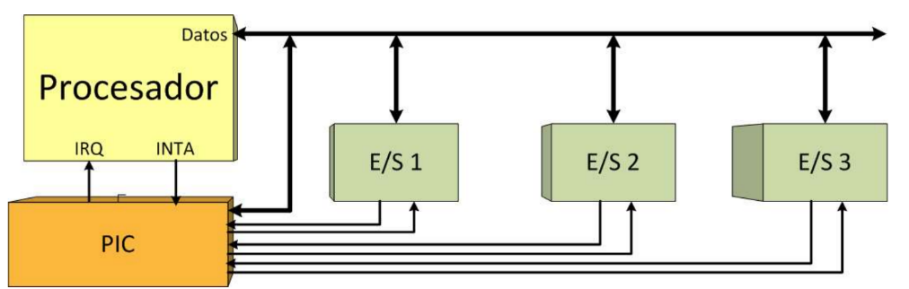
**1) Diferentes líneas de interrupción:** Se dispone de una línea de interrupción por cada dispositivo. Es sencilla de implementar. Hay una limitación en la cantidad de dispositivos a conectar debido a la cantidad restringida de señales de interrupción que puede manejar la CPU.

**2) Una sola línea de interrupción y encuesta por software:** Se dispone de 1 sola línea de interrupción para todos los dispositivos. Cuando ocurre el pedido de interrupción, la CPU tiene que consultar a cada dispositivo (es decir, a cada módulo de E/S) para determinar quien fue el demandante. Este esquema de encuesta por software (también conocido como “polling”) puede resultar sumamente lento.

**3) Una sola línea de interrupción con conexión en cadena:** Se dispone de 1 sola línea de interrupción INTR para todos los dispositivos. La línea de respuesta de la CPU INTA (reconocimiento de interrupción) se conecta encadenadamente a todos los módulos (conexión tipo “margarita” o “Daisy chain”).

Una vez enviada la confirmación de parte de la CPU, el módulo que está más adelante (más próximo a la CPU) en la conexión Daisy chain responderá colocando un vector (palabra), en el bus, que lo identifica. Si hay otros pedidos de interrupción más abajo del que respondió, deberán esperar la terminación del servicio del que respondió.

**4) Una sola línea de interrupción y vectorizado:** Se dispone de 1 sola línea de interrupción INTR para todos los dispositivos. Un controlador dedicado (PIC) provee el vector que identifica la fuente de interrupción. Las líneas de interrupción tienen un orden de prioridad, las líneas con más prioridad pueden interrumpir a las líneas con menor prioridad. Si existe un maestro del bus, solo él puede interrumpir.



En general:

* Las operaciones de E/S administradas por interrupción son más eficientes que las programadas con espera.
* Ambas técnicas requieren la intervención directa de la CPU.
* Al tener que intervenir la CPU en las transferencias de los datos se presentan 2 problemas:
  + La velocidad de transferencia depende de la capacidad de la CPU de atender estas tareas. Aunque la CPU es muy rápida, puede ser que en determinadas circunstancias no sea capaz de administrar varias transferencias simultáneamente.
  + La CPU puede permanecer ocupada mucho tiempo durante la operación, sin poder hacer otras tareas. Además, si el volumen de datos a transferir es grande, el tiempo de ocupación de la CPU crece también.

Para analizar el comportamiento de la CPU en transferencias de E/S, vamos a analizar la respuesta de la CPU en 2 casos:

1) Transferencia hacia un periférico lento (impresora de 20 páginas/minuto)

2) Transferencia hacia un periférico rápido (disco de 10Mb/s)

Para los 2 casos, se analizará la respuesta del sistema para transferencias de E/S manejadas por la CPU con espera de respuesta y administrada por interrupción.

Sobre la CPU

Se tiene un procesador con un reloj de 200 MHz (período del reloj = 5 ns).

El procesador, en promedio, necesita 2 ciclos de reloj por instrucción. Este parámetro se conoce como CPI. En este caso CPI=2. Es decir, que una instrucción tarda, en promedio:

Ciclo de instrucción = 2 x 5 ns = 10 ns = 10^-8

El procesador tiene, por lo tanto, una capacidad de procesamientode:

N° de instrucciones por segundo = 1 / tiempo de ciclo de instrucción

N° instrucciones por segundo = 1/10^-8

N° Instrucciones por segundo = 100 millones de instrucciones por segundo (100 Mips)

**Caso 1: Transferencia a una Impresora**

Se tiene que imprimir (operación de salida) un archivo de 10 Kbytes en una impresora láser de 20 páginas por minuto. Considerando que 1 página tiene (estimado) 3.000 letras (es decir, 3000 caracteres) y 1 carácter = 1 byte

La impresora imprime:

20 ppm = 20 pag/min x 3000 car/pag = 60.000 caracteres por minuto

Es decir, que la velocidad de transferencia de la impresora es de :

Vt = 60000 car./min = 60000 car./60 seg. = 1000 car./s = 1 Kbyte/s

**a) E/S con espera de respuesta**

La CPU entra en un bucle y envía un nuevo byte cada vez que la impresora está preparada para recibirlo

Si la impresora tarda 1 seg en imprimir 1 Kbyte (ya que la tasa de transferencia es de 1 kbyte/seg), necesitará:

Tiempo total de transferencia = 10 seg para 10 Kbytes.

Conclusión:

La CPU está ocupada con la operación de E/S durante 10 seg.

**Observación:**

**Ya que la velocidad de la CPU es de 100MIPs, en ese tiempo la CPU podría haber ejecutado: N°instrucciones en 10 seg. = 1000 millones de instrucciones.**

**b) E/S administrada con interrupciones**

La impresora genera una interrupción cada vez que está preparada para recibir un nuevo byte.

Si la gestión de la interrupción (que se llama ATI) requiere 10 instrucciones (entre las que se incluyen salvar contexto, comprobar estado, transferir byte, restaurar contexto, y retornar), entonces:

Para transferir 10 Kbytes se requiere ejecutar 10.000 veces la ATI.

El tiempo que dura el servicio de la transferencia es, aproximadamente, el requerido por las 10.000 ATI. Recordando que 1 instrucción tiene un tiempo de ejecución de 10^-8 seg

Tiempo total de transferencia = 10.000 ATI = 10.000 x 10 instr =10.000 x 10 x 10^-8 seg = 100.000 x 10^-8seg = 10^-3 seg = 0,001seg

**Conclusión:**

**La CPU está ocupada con la operación de E/S durante 0,001 seg. Es decir, que la E/S por interrupciones reduce en 10.000 veces el tiempo que la CPU está ocupada gestionando la impresora, y por lo tanto, es mucho más eficiente.**

**Caso 2: Transferencia a un disco**

Se tiene que transferir un archivo de memoria a disco de 10 Mbytes considerando que:

El disco posee una velocidad de transferencia de 10 MB/s (1 bytecada 10-^7 seg ó 100 nanoseg) significa que:

Tiempo total de transferencia= 1 segundo.

**a) E/S con espera de respuesta**

La CPU entra en un bucle y envía un nuevo byte cada vez que el disco está preparado para recibirlo. Si el disco transfiere 10 Mbytes/seg, necesita 1 seg para recibir un archivo de 10 Mbytes.

Tiempo total de transferencia = 1 seg para 10 Mbytes.

**Conclusión:**

**La CPU está ocupada con la operación de E/S durante 1 seg.**

**Ya que la velocidad de la CPU es de 100MIPs, en ese tiempo la CPU podría haber ejecutado: N°instrucciones en 1 seg. = 100 millones de instrucciones**

**b) E/S administrada con interrupciones**

El disco genera una interrupción cada vez que está preparado para recibir un nuevo byte. Si la gestión de la interrupción (que se llama ATI) requiere 10 instrucciones (entre las que se incluyen salvar contexto, comprobar estado, transferir byte, restaurar contexto, y retornar), entonces, para transferir 10 Mbytes tenemos que ejecutar 10^7 veces la ATI

El tiempo que dura el servicio de la transferencia es, aproximadamente, el requerido por las 10.000.000 ATI. Recordando que 1 instrucción tiene un tiempo de ejecución de 10^-8 seg

Tiempo total de transferencia = 10.000.000 ATI = 10^7 x 10 instr = 10^7 x10 x 10^-8 seg = 1 seg

**Conclusión:**

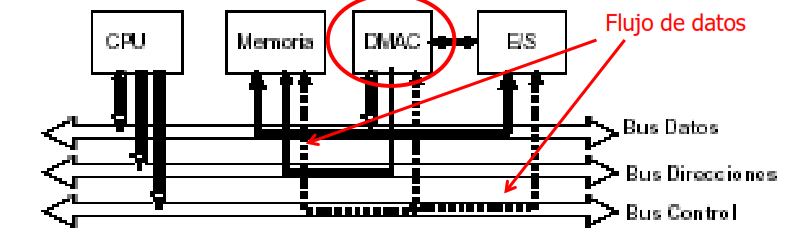
**La CPU está ocupada con la operación de E/S durante 1 seg. Es decir, que no hay diferencia entre las 2 técnicas. En ambas la CPU está ocupada 1 segundo el 100% del tiempo.**

**Si la velocidad del dispositivo fuera mayor, la CPU no podría hacerla.**

Acceso directo a memoria (DMA)

El Acceso Directo a Memoria (DMA) es una técnica de transferencia de datos entre periférico y Memoria sin intervención directa de la CPU. Comúnmente llevada a cabo por un “Controlador de DMA” (o DMAC) específico encargado de llevar a cabo la transferencia.

Físicamente está ubicado como se muestra a continuación.



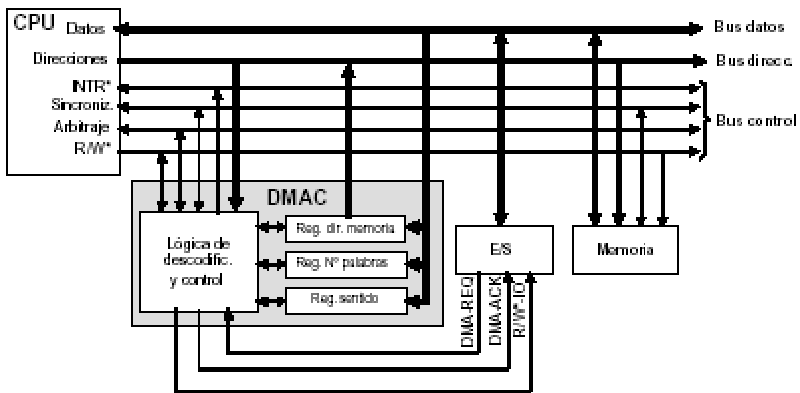
Dado que la transferencia por DMA requiere el uso del Bus, tanto el DMAC como la CPU pueden tomarlo. El DMAC y la CPU “compiten” por el uso del Bus.}

Cuando el DMAC toma el Bus, actúa como “master” durante la transferencia por DMA, y debe ser capaz de:

* Solicitar el uso del bus mediante las señales y la lógica de arbitraje necesarias
* Especificar la dirección de memoria sobre la que se realiza la transferencia
* Generar las señales de control del bus
* Especificar el tipo de operación (lectura/escritura)
* Generar las señales de sincronización de la transferencia

Cuando la CPU entrega el bus al DMAC (8237 DMA chip en la figura), se desconecta lógicamente del mismo, y es el DMAC el que toma el control del bus.

En la figura siguiente se muestra un detalle de la forma en que se conecta al Bus del sistema y de su estructura interna.



El proceso de transferencia requiere realizar una serie de acciones o fases relativamente complejas. Las principales son:

1.- Fase de Inicialización

2.- Fase de ejecución de la transferencia

3.- Fase de finalización y análisis de la transferencia

**1.- Fase de inicialización**

En la fase de inicialización la CPU debe configurar el módulo de E/S y el DMAC con los parámetros de la transferencia.

Inicialización interfaz de E/S:

* Tipo de transferencia (lectura/escritura)
* Configuración del periférico
* Otra información de control para el periférico (por ejemplo, si es un disco se especifica el número de pista, sector, etc.)

Inicialización DMAC

* Nº de bytes o palabras a transferir
* Tipo de transferencia (lectura/escritura)
* Dirección de memoria inicial para la transferencia
* Otra información la transferencia.

**2.- Fase de ejecución de la transferencia**

Cuando el periférico está listo, pide al DMAC iniciar la transferencia mediante una señal física.

Cuando el DMAC recibe el pedido del periférico, pide el control del bus mediante alguna señal especial a la CPU. La CPU típicamente dispone de algunas señales destinadas a implementar las transferencias por DMA.

Cuando reconoce el pedido de DMA, la CPU entrega (libera) el bus y se “desconecta” (ya no controla el bus) lógicamente del mismo.

La CPU avisa al DMAC que liberó el bus mediante otra señal especial.

Al liberar la CPU el bus, el DMAC toma el control del bus y ejecuta la transferencia hasta terminarla.

El DMAC avisa al periférico que puede iniciar la transferencia. El periférico comienza a transferir los datos, a través de bus, con la memoria, de a uno por vez.

La transferencia implica que:

Bus master: DMAC + Periférico - Bus slave: Memoria

Después de la transferencia de cada palabra se actualizan los registros del DMAC:

* Nº de bytes faltantes (o cuenta de los que se transfirieron)
* Próxima dirección de memoria a donde guardar el dato (anterior o posterior de la corriente)

Cuando el número de bytes faltantes es igual a 0 significa que transfirió todos los datos y terminó la transferencia.

**3.- Finalización de la transferencia**

Una vez que termina la fase 2, el DMAC libera el bus y le avisa a la CPU por medio de una señal física y la CPU retoma el control del bus.

El DMAC suele activar, además, una señal de interrupción para indicar a la CPU la finalización de la operación de E/S solicitada.

La CPU, mediante la interrupción, verifica el resultado de la transferencia vía los registros internos del DMAC. Algunos resultados a verificar son:

* Transferencia OK/fallida?
* Errores? Tipo de errores?
* Estado periférico

**Ventajas e inconvenientes del DMA**

La principal ventaja es la eficiencia, dado que la CPU se libera de tener que controlar la transferencia de los datos. Solo prepara la transmisión, y verifica el resultado de lamisma.

La principal desventaja se origina en el uso del bus. Como las transferencias por DMA pueden tener mayor prioridad que la CPU, se puede degradar el rendimiento de la CPU si el DMAC hace uso intensivo del bus.

Sin embargo, no necesariamente la CPU necesite todo el tiempoel bus porque:

* En el caso de Computadoras con memoria caché: La mayor parte del tiempo, la CPU lee instrucciones de la cache, por lo que no necesita usar el bus de memoria. El DMAC puede aprovechar estos intervalos en los que la CPU está leyendo instrucciones de la cache (y por tanto no usa el bus de memoria) para realizar las transferencias.
* En el caso de computadores sin cache: El procesador no utiliza el bus en todas las fases de la ejecución de una instrucción. El DMAC puede aprovechar las fases de ejecución de una instrucción en las que la CPU no utiliza el bus.

**Técnicas de transferencia por DMA**

Hay varias formas distintas de implementar las transferencias por DMA. Vamos a ver 2:

* Por ráfagas (burst)
* Por robo de ciclo (cycle-stealing)

**Método de transferencia de DMA por ráfaga**

Es la vista hasta ahora. El DMAC solicita el control del bus ala CPU.Cuando la CPU concede el bus, el DMAC no lo libera hastahaber finalizado la transferencia de todo el bloque de datos completo.

VENTAJAS: La transferencia se realiza de forma muy rápida, limitada por la velocidad del periférico.

DESVENTAJAS: Durante el tiempo que dura la transferencia la CPU no puede utilizar el bus con memoria, lo que puede degradar el rendimiento del sistema.

**Método de transferencia de DMA por robo de ciclo**

El DMAC solicita el control del bus a la CPU.Cuando la CPU concede el bus al DMAC, se realiza latransferencia de una única palabra y después el DMAClibera el bus.

El DMAC solicita el control del bus tantas veces como sea necesario hasta finalizar la transferencia del bloque completo. El uso del bus se reparte entre la CPU y el DMAC.

VENTAJAS: No se degrada tanto el rendimiento del sistema y de la CPU.

DESVENTAJAS: La transferencia puede tardar un poco más de tiempo.  
  
Notar que dado que la toma y liberación del bus por parte de la CPU no es una interrupción, es decir que el procesador no debe guardar el contexto (es decir, no está interrumpiendo su tarea).  
Si bien el trabajo de la CPU es más lento (que si no estuviera presente la transferencia por DMA), no será tanto como si ella tuviera que estar desconectada del bus todo el tiempo.

En general, para transferencia de E/S de múltiples palabras, la técnica por robo de ciclo es la más eficiente, ya que permite implementar la transferencia por DMA al mismo tiempo que la CPU continúa trabajando en su tarea.  
  
Canales de E/S  
Las transferencias de E/S se pueden dividir, en función de la capacidad para interactuar con los periféricos, en varios niveles:

Nivel 1: CPU + Módulo de Interfaz de E/S o controlador: la CPU controla directamente los periféricos e interfaz, y administra la transferencia por programa (con espera).

Nivel 2: CPU + Módulo de Interfaz E/S o controlador con interrupción: la CPU controla directamente los periféricos y administra la transferencia con programa e interrupciones.

Nivel 3: DMA (DMAC + Módulo de E/S): la CPU no interviene directamente, solo prepara y supervisa la transferencia.

Nivel 4: Canal de E/S básico (Procesador básico + módulo de E/S): la CPU interviene mínimamente.

Nivel 5: Canal de E/S inteligente (Procesador inteligente + módulo de E/S): la CPU no interviene, excepto situaciones especiales.  
  
En el nivel más alto de la escala de transferencias de E/S están los canales de E/S, que representan una extensión al concepto de DMA.

Los canales de E/S tienen la habilidad de ejecutar programas de servicios de E/S, lo que les permite tener un completo control de la transferencia de datos. La CPU no ejecuta las instrucciones de E/S, las realiza el procesador incluido en el canal. El programa que ejecuta el procesador interno del canal está almacenado en la memoria principal.

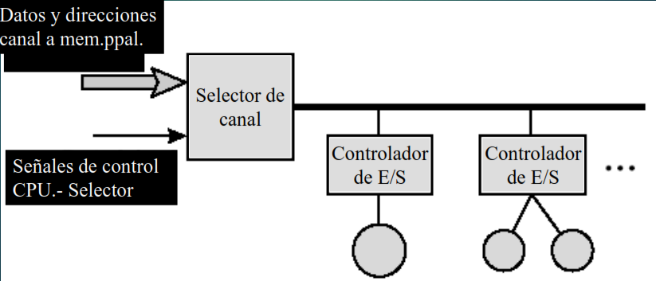
La CPU solamente interviene para iniciar la transferencia, y dar la orden de ejecutar el programa de E/S que está en memoria. El programa de servicio de E/S especifica dispositivos, áreas de memoria a usar, prioridades y acciones ante errores.  
  
Hay 2 tipos básicos de canales de E/S:

* Selector
* Multiplexor

**Canal selector de E/S**

El canal controla varios dispositivos de alta velocidad, de a uno por vez. El canal selecciona un dispositivo y efectúa la transferencia sobre el dispositivo seleccionado.

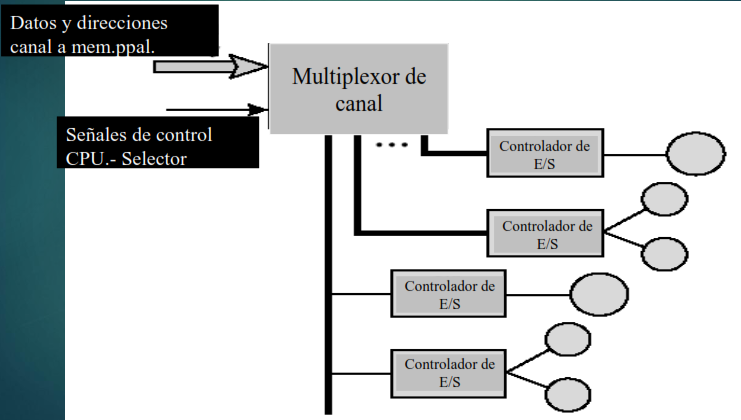
Cada dispositivo tiene asociado un controlador o módulo deE/S que lo maneja. Por lo tanto el canal de E/S ocupa el lugar de la CPU en el control del módulo de E/S.



**Canal Multiplexor de E/S**

El canal controla varios dispositivos de alta velocidad, incluso simultáneamente. El canal mutiplexa la atención entre los dispositivos seleccionados. El multiplexado puede ser:

* Multiplexor de bytes: acepta y transmite de a caracteres.
* Multiplexor de bloques: intercala bloques de datos desde distintos dispositivos.



**Clase 4 – Subsistemas de Memoria**

En el diseño de la Memoria existe un compromiso entre:

* Capacidad
* Velocidad
* Costo

Una Memoria ideal sería aquella que:

Es infinitamente grande, el tiempo de acceso sumamente pequeño (casi 0) y el costo relativamente bajo

Jerarquía de memoria

En una computadora típica hay distintos tipos de memorias, desde las rápidas y caras (Ej.: registros) hasta las lentas y baratas (Ej.: discos). En las computadoras actuales los diferentes tipos de memorias actúan coordinadamente y no separadas. Esa interacción permite un comportamiento global equivalente al que tendría con una memoria única, grande y rápida.

La forma en que se organizan coordinadamente los distintos tipos de memoria se conoce con el nombre de Jerarquía de Memoria. La Jerarquía de memoria se puede pensar como una pirámide de múltiples capas o niveles, de diferentes tamaños y velocidades.

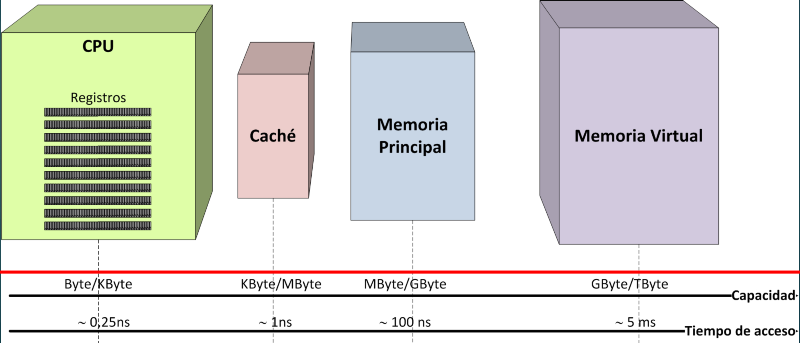
Una Jerarquía de Memoria es, un método de administración del almacenamiento de la información (Memoria) estructurado en varios niveles ubicados físicamente en distintos lugares, con tecnologías, costos, tamaños y velocidades distintas.

De esta manera los programadores “creen” disponer de cantidades casi “ilimitadas” de memoria a un costo accesible, y con velocidades cercanas a las de una memoria “ultrarápida”.

**Niveles principales de la Jerarquía de Memoria:**

* Registros
* Memoria cache (RAM de muy alta velocidad)
* Memoria principal (RAM de alta velocidad)
* Memoria virtual o secundaria (medios de almacenamiento magnético/óptico)

A medida que nos alejamos de la CPU, los niveles son más grandes, más lentos y más baratos que los niveles previos (o superiores) en la jerarquía, de ahí la forma de pirámide.



Los principales objetivos de una Jerarquía de memoria son:

* Maximizar tamaño: idealmente disponer de una “capacidad ilimitada”, equiparada al tamaño del nivel más grande.
* Optimizar velocidad: simular que se dispone de un banco de memoria “ultrarápida”, próximo a la velocidad del nivel más rápido
* Minimizar el costo total: implementar una memoria a un costo cercano al del nivel más lento.

Para que la memoria se comporte como una jerarquía (integrada) debe cumplir las siguientes propiedades:

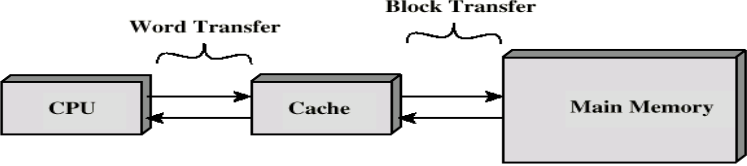
* Inclusión: los datos almacenados en un nivel han de estar también almacenados en los niveles inferiores a él.
* Coherencia: las copias de la misma información en los distintos niveles deben contener los mismos valores.

El manejo de la jerarquía de memoria es administrado por:

* Nivel registros: el compilador. Se puede decir que el programador no interviene en la administración de este recurso porque en los lenguajes de programación no son visibles (con algunas excepciones)
* Cache: la administración se hace por hardware
* Memoria principal: la administración la pueden hacer: Hardware, Sistema operativo o Programador (archivos)

Memoria Cache

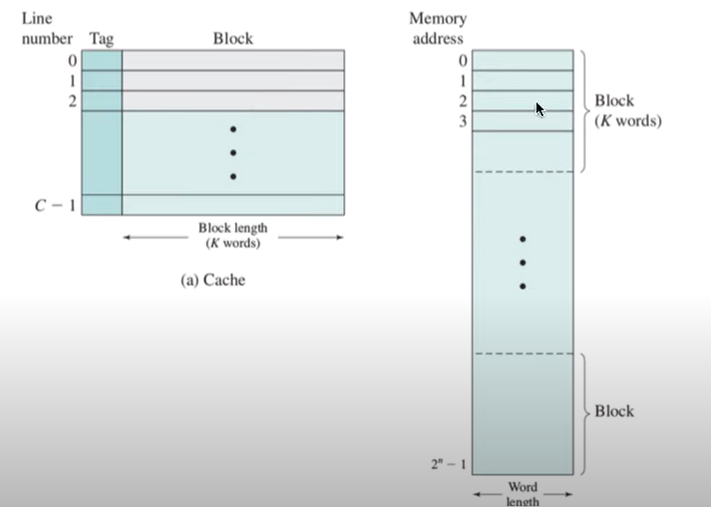
La memoria Caché es una memoria pequeña y muy rápida, que se ubica entre la memoria principal y la CPU. Puede localizarse en un chip separado o dentro de la CPU, o en ambos lugares y contiene algunos sectores (bloques) de la MP.



La información contenida en la Caché se organiza en bloques (también llamadas ranuras) de longitud fija (Ej.: 8 bytes, 16 bytes, 32 bytes, etc.).

En los bloques de la Caché se copian algunos bloques (de idéntico tamaño) de la Memoria principal. La cantidad de bloques copiados depende del tamaño de la memoria Caché y del bloque.

Cada ranura tiene asociada una etiqueta para identificar el bloque de la Memoria que tiene copiado. El conjunto de etiquetas forma el directorio de la cache.



**Funcionamiento:**

Cuando la CPU necesita un dato, genera 1 dirección de memoria. La cache “intercepta” esa dirección y determina si tiene ese dato. Pueden ocurrir 2 situaciones:

ACIERTO: si lo tiene, se lo envía a la CPU (a la velocidad de la cache)

FALLO: Si no lo tiene, se trae el bloque que contiene esa dirección desde la memoria principal, y la cache entrega el dato requerido a la CPU.



Si el dato que busca la CPU está en la cache, la velocidad del acceso depende del tiempo de acceso de la cache (relativamente muy corto).

Si el dato no está en la cache, la velocidad del acceso depende del tiempo de acceso de la memoria principal (relativamente largo).

Así, la eficiencia del uso de la cache depende de la cantidad de veces que “acierta” a la cache. La cantidad de veces que se acierta (la “tasa de aciertos”) no necesariamente tiene que ser proporcional al tamaño de la caché, que es miles de veces mas chico que el de la Memoria principal. La razón de esto tiene que ver con el comportamiento de los programas.

**Principios en los que se basa la memoria caché**

Los programas tienen un comportamiento basado en 2 principios (que son de carácter empírico).

Principio de localidad temporal de las referencias:

Es altamente probable que los elementos de memoria referenciados recientemente (datos o instrucciones) vuelvan a ser referenciados en el corto tiempo.

Principio de localidad espacial de las referencias:

Es altamente probable que los próximos elementos de memoria referenciados estén en las proximidades de los últimos referenciados.

**Acceso a memoria en un procesador segmentado**

En un procesador con segmentación del cauce, se dispone de un ciclo de reloj para el acceso a memoria. Ese tiempo debe ser suficiente para el acceso a la memoria caché. En caso de fallo el acceso a la memoria principal requiere de varios ciclos extra.

Tal como se vio, cuando la CPU busca un dato se pueden dar 2 situaciones:

Acierto (se conoce como “hit”): se encuentra en la caché el dato solicitado

Fallo (se conoce como “miss”): no se encuentra en la caché el dato solicitado

Cuando ocurre un fallo, el bloque que contiene la palabra accedida se copia de la memoria principal a una línea de caché.

Los fallos de caché se gestionan mediante hardware y causan que el procesador se detenga hasta que el dato esté disponible. Esta acción requiere un tiempo determinado.

El tiempo para servir un fallo depende de la latencia y ancho de banda de la memoria principal.

La latencia es el tiempo necesario para completar un acceso a memoria (depende de la memoria).

El ancho de banda es la velocidad a la cual se puede transferir el dato, es decir, es la cantidad de información por unidad de tiempo que puede transferirse desde/hacia la memoria (depende de la velocidad del bus).

El tiempo de acceso promedio de la CPU es el promedio del tiempo que tarda en obtener los datos buscados en memoria, compuestos por accesos a la caché y accesos a la memoria principal.

t CPU = (1-TF) t accesoMC + TF x PF

TF es la “tasa de fallos”, que se obtiene como:

TF=Tasa de Fallos = número de fallos / número total de accesos

PF es la “penalización por fallo”, es decir el tiempo “gastado” enacceder a la Memoria principal (taccesoMP)

Para mejorar las prestaciones hay que reducir el t accesoMC, TF y PF.

**Consideraciones sobre la cache**

Para el diseño de la caché hay que tener en cuenta varias consideraciones.

1. Tamaño de la memoria caché: Debe ser suficientemente grande para contener la mayor cantidad posible de información. Pero no demasiado grande porque el tamaño tiene impacto en la velocidad (es decir, en el taccesoMC) y en el costo.

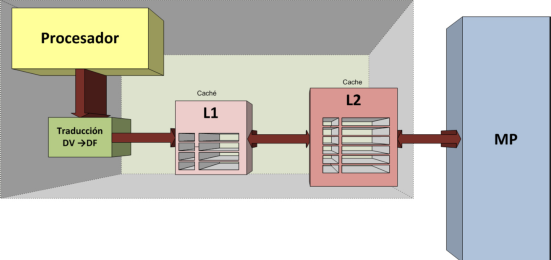
2.Tamaño del bloque o ranura: El tamaño del bloque es muy importante en la tasa de aciertos. El bloque o ranura debe ser suficientemente grande para aprovechar al máximo las referencias cercanas. Al aumentar el tamaño mejora la tasa de aciertos, hasta un cierto tamaño del bloque. Después de eso, aumentar el tamaño no mejora la tasa de aciertos.

Por otra parte, al aumentar el tamaño del bloque hay menos bloques de la memoria principal en la caché, lo que tiende a aumentar la tasa de fallos, y la penalización por fallos porque son más palabras a transferir entre la memoria principal y la caché.

Existe un valor óptimo para el tamaño del bloque.

3.Costo: El costo crece fuertemente con el tamaño de la memoria. Y este costo es significativamente grande cuando la caché está incluida en el chip que contiene el procesador.

4.Niveles de la caché: La memoria caché pude ser una sola (1 nivel) o estar dividida en varias unidades (multiples niveles). Los múltiples niveles de caché por lo general tienen distintos tamaños. En la figura siguiente, la caché está dividida en 2 niveles, el L1 muy pequeño y ultrarápido, y el L2 un poco más grande y lento.



5. Separación de la caché de instrucciones y operandos: El mecanismo de acceso a las instrucciones es distinto al del acceso de los datos. Es por ello que las estrategias para obtener una alta tasa de aciertos es distinta en un caso que en el otro. Teniendo en cuenta esto, es posible mejorar la tasa deaciertos general si la caché se divide en una caché de instrucciones y una de datos, con distintas caracteríticas.

Por ejemplo, ambas caches pueden tener distintos tamaños y políticas de acceso y reemplazo.

**Diseño de la caché**

En el diseño de la caché se deben definir:

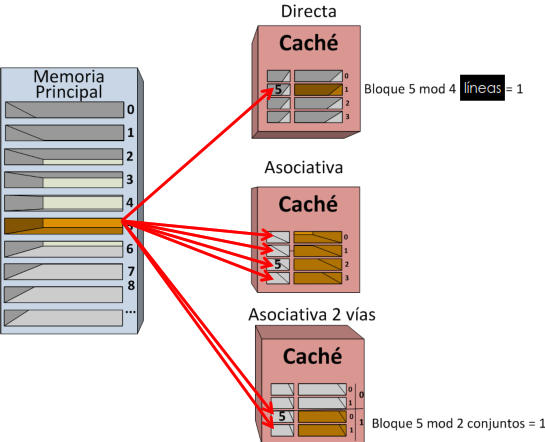
* Organización: tamaño de la caché, cantidad y tamaño de de bloques
* Política de asignación: es decir, el tipo de función de correspondencia entre los bloques de la Memoria Principal y los bloques o ranuras de la Caché.
* Política de reemplazo: se refiere a los algoritmos para reemplazar bloques en la memoria Caché.
* Política de escritura: son los mecanismos de escritura en Memoria Principal.

**Políticas de asignación**

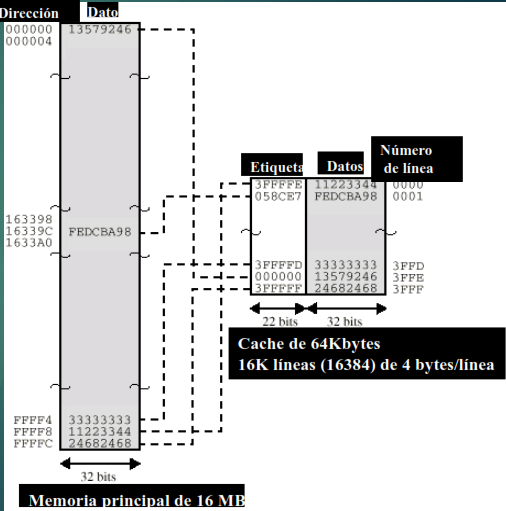
Las políticas de asignación son las funciones (de mapeo) que definen la forma en que se van a asignar los bloques de la Memoria Principal en la Memoria Caché.

Las políticas más empleadas son:

* Correspondencia totalmente asociativa: Un bloque puede almacenarse libremente en cualquier lugar de la caché.
* Correspondencia directa: Un bloque sólo puede estar almacenado en un lugar fijo de la caché.
* Correspondencia asociativa por conjuntos: Un bloque puede almacenarse en un conjunto restringido de lugares en la caché.

****

**Política de asignación Asociativa**

En la imagen de al lado se

muestra un ejemplo de

asignación completamente

asociativa de una memoria

de 16MB y caché de 64KB,

con bloques de 4 bytes. De

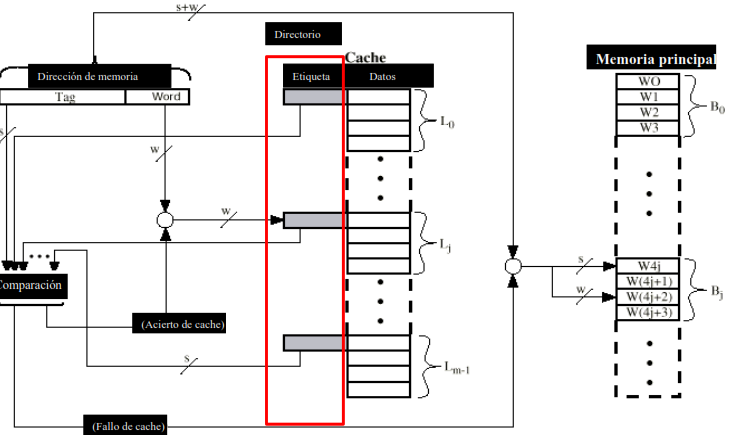
los 24 bits que forman la

dirección, los 22 bits más

significativos se usan para

identificar el bloque, y 2 para

la palabra dentro del bloque

En general, el esquema de operación en un esquema de asignación asociativa, se muestra en la figura siguiente.

En la figura anterior, se puede apreciar que:

La dirección de memoria está compuesta de 2 campos

* Tag: es el número de bloque en la memoria principal.
* Word: es el número de palabra dentro del bloque

El tag es comparado simultáneamente con todas las etiquetas de la caché, que identifican qué bloques de la MP están asignados a la caché. Las etiquetas de la memoria caché forman el “directorio” de la caché.

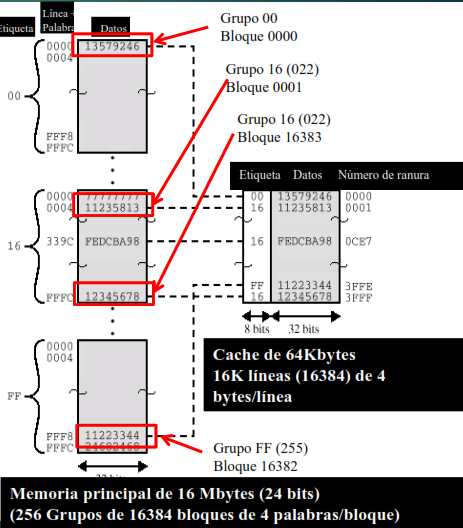
Si la comparación da un acierto, el dato se busca en la caché. Si la comparación da una falla, el dato se trae de la memoria principal.

**Política de asignación Asociativa - Conclusiones**

Un bloque de memoria principal puede colocarse en cualquier línea de la cache. La etiqueta identifica unívocamente un bloque de memoria.  
Todas las etiquetas de las líneas se examinan para buscar una coincidencia. Para esa búsqueda se requiere una memoria del tipo asociativa (es decir, una memoria de acceso por contenido CAM) para implementar el directorio.

La implementación del directorio de la caché es compleja y costosa.  
Esta política permite una libertad absoluta para la asignación y reemplazo de los bloques.

**Política de asignación Directa**

En la imagen de al lado se

muestra un ejemplo de

asignación directa de una

memoria de 16MB y caché de

64KB, con bloques de 4 bytes. De

los 24 bits que forman la

dirección, los 8 más significativos

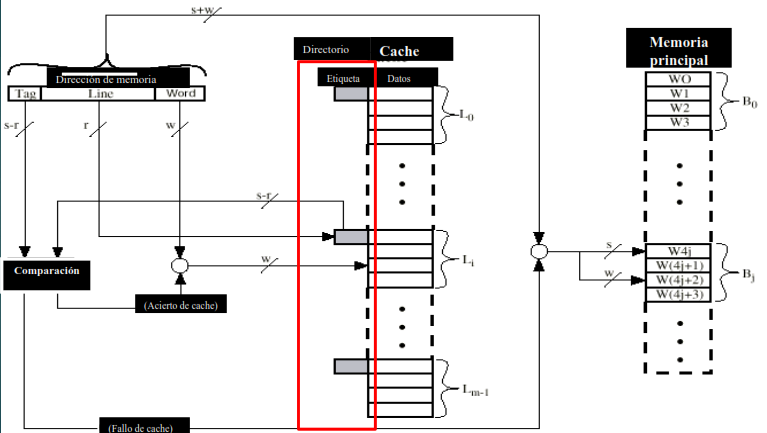
son el N° de grupo, los siguientes

14 el bloque dentro del grupo y

los 2 restantes la palabra dentro

del bloque.

En general, el esquema de operación en un esquema de asignación directa, se muestra en la figura siguiente.



En la figura anterior, se puede apreciar que La dirección de memoria está compuesta de 3 campos:

* Tag: es el número de grupo en la memoria principal
* Line: en el número de bloque dentro del grupo
* Word: es el número de palabra dentro del bloque

El tag es comparado con la etiqueta de la caché correspondiente al número de línea. Hay tantas líneas en la caché como bloques por grupo en la memoria principal. Las etiquetas de la memoria caché forman el “directorio” de la caché.  
Si la comparación da un acierto, el dato se busca en la caché. Si la comparación da una falla, el dato se trae de la memoria principal.

**Política de asignación Directa - Conclusiones**

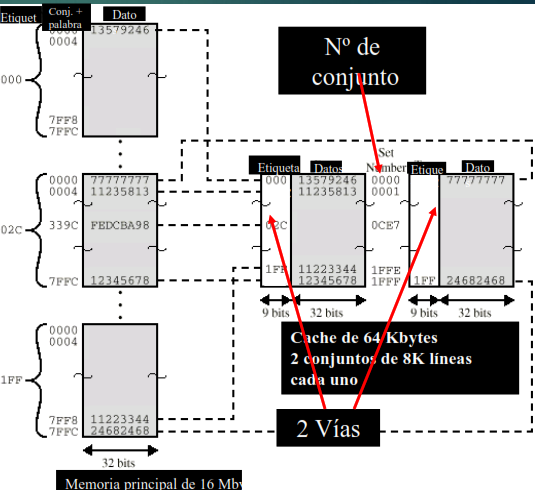
Un bloque de memoria principal puede colocarse en una única línea de la cache.

Nº línea caché = Nº bloque ref. “en módulo” Nº líneas caché

La etiqueta solo contiene el número de grupo de la memoria principal asignado a esa línea de la caché.

Esta política de asignación es muy simple de implementar y poco costosa. Tiene un rendimiento aceptable, aunque a veces puede ser malo. Por ejemplo, si un programa accede a dos bloques que se corresponden a la misma línea (diferentes bloques de memoria principal) de forma repetida, las pérdidas de cache (desaciertos) serán muy grandes.

**Política de asignación Asociativa por conjuntos**

En la imagen de al lado se

muestra un ejemplo de

asignación asociativa por

conjuntos de 2 vías de una

memoria de 16MB y caché

de 64KB, con bloques de 4

bytes. De los 24 bits que

forman la dirección, los 9

más significativos son el N°

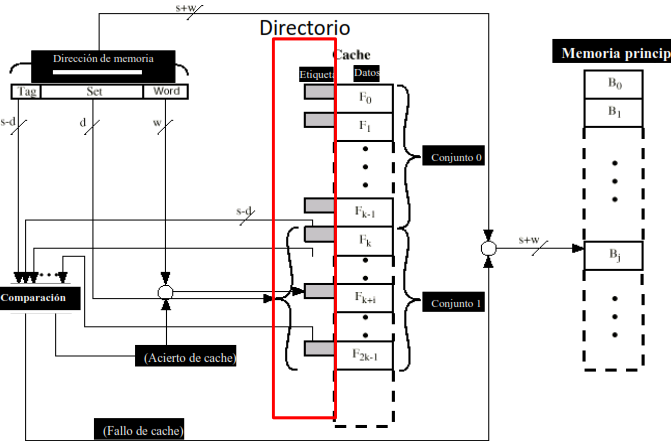
de grupo, los siguientes 13

el conjunto dentro del

grupo, y los 2 restantes la

palabra dentro del bloque

En general, el esquema de operación en un esquema de asignación asociativa por conjuntos, se muestra en la figura siguiente.



**Política de asignación Asociativa por conjuntos - Conclusiones**

Un bloque de memoria principal puede colocarse en bloques determinados de la cache. La cache se divide en un número de conjuntos N (N vías, con N=2, 4, 8 ... etc.)

Cada conjunto contiene un número de líneas o ranuras. Un bloque determinado corresponderá a alguna línea o ranura de un conjunto determinado.

En general, la función asociativa por conjuntos combina lo mejor de las otras correspondencias (asociativa y directa).

**Políticas de reemplazo**

Cuando hay un fallo de acceso a la memoria cache, se debe traer un bloque desde la Memoria principal y almacenar un bloque existente en la memoria cache.

El lugar a donde va a ser ubicado el bloque a traer desde la memoria principal requiere reemplazar un bloque existente. Las diferentes estrategias de reemplazo de los bloques se conocen como Políticas de reemplazo de bloque.

**En Correspondencia directa**

En la función de mapeado directo, la asignación de bloques de la memoria principal a bloques de la cache es fija, no hay elección. Sólo hay una posible línea para cada bloque.

Por lo tanto si se requiere traer un nuevo bloque desde la memoria principal, indefectiblemente reemplazará el que está usando esa ranura actualmente.

**En Correspondencia asociativa y asociativa por conjuntos**

Para las asignaciones asociativa y asociativa por conjuntos, hay varios algoritmos de reemplazo:

1. LRU

2. FIFO

3. LFU

4. Aleatorio

Todos los algoritmos deben implementarse en hardware (por razones de velocidad).

**1.- Menos usado recientemente (LRU)**

Se reemplaza el bloque que lleva más tiempo sin utilizarse, Requiere controles de tiempos, Aprovecha la localidad temporal, Válido en Asociativas por conjunto, donde por cada ranurase agrega un bit de USE para identificar el usadorecientemente.

**2.- Primero en entrar - primero en salir (FIFO).**

Se reemplaza el bloque que entró antes en la caché. Requiere controles de acceso para identificar el orden en que ingresaron.

**3.- Menos frecuentemente usado (LFU)**

Se sustituye aquella línea que ha experimentado menos referencias. Requiere controles de uso.

**4.- Aleatoria:** Se sustituye una línea al azar.

**Políticas de escritura**

Cuando la CPU tiene que almacenar (“escribir”) un resultado en la memoria puede hacerlo tanto en un acierto (la dirección donde se va a guardar el dato está en un bloque de la cache) como en un fallo (la dirección donde se va a guardar el dato NO está en un bloque de la cache).

En cualquiera de las 2 situaciones, se debe evitar inconsistencia de información entre las memorias principal y cache, durante los procesos de escrituras. Es decir que, aún escribiéndose el dato en la cache, el correspondiente bloque de la memoria principal debe será actualizado en algún momento.

Además, a veces un módulo E/S puede tener acceso directo a la memoria principal y requerir información que fué modificada en la cache. En arquitecturas complejas (procesadores paralelos) múltiples CPU pueden tener caches individuales.

Las políticas de escritura son distintas en aciertos que en fallos. Cuando la dirección de memoria a donde tiene que escribir un dato está en la memoria caché, se usan las políticas de escritura en acierto.

Cuando la dirección de memoria a donde tiene que escribir un dato no está en la memoria caché, se usan las políticas de escritura en fallo.

**Políticas de escritura en aciertos**

Hay 2 estrategias:

* Write-through (Escritura inmediata): Se actualizan simultáneamente la posición de la caché y de la memoria principal. Aumenta el tráfico con la memoria principal y Puede haber retrasos durante múltiples escrituras.
* Write-back (Post-escritura).: La información sólo se actualiza en la caché y se escribe la memoria cuando se reemplaza el bloque. El bloque requiere de un bit de “sucio” para indicar cuando se lo escribió.

Como la memoria principal se actualiza en el reemplazo, puede contener información errónea en algún momento.

**Políticas de escritura en fallos**

Hay 2 estrategias:

* No-write allocate: Se escribe directamente en la memoria principal. La caché se usa solo en las lecturas. El bloque no se lleva a la memoria caché mientras no se lo tenga que leer. Habitual con write-through.
* Write allocate: El bloque requerido primero se copia en la cache, y luego se escribe (en la cache). Habitual con write-back.

**Clase ¿? – Segmentación del cauce (namoMips)**

Introducción al nanoMIPSEl nanoMIPS tiene estas características principales:

* Palabra de memoria de 32 bits
* Espacio de direcciones de 64 bits (virtual)
* 32 (31 efectivos) Registros de propósito general de 64 bits (R0..R31). El registro R0 es 0 (es decir, si referenciamos al registro R0 nos devuelve 0).
* Como son 32 registros, se requieren 5 bits para identificarlo.

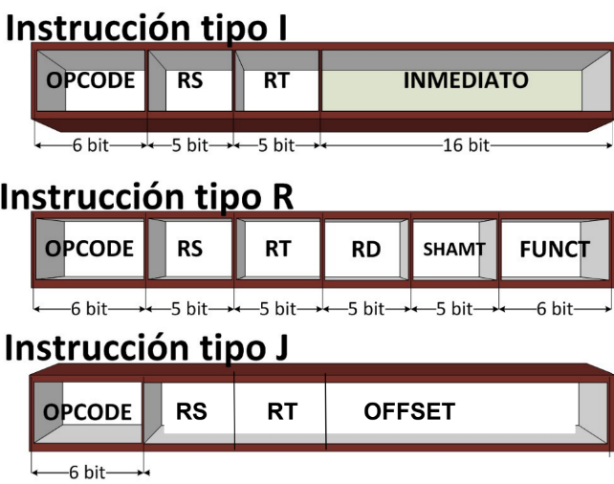
**Formato de instrucción**

El nanoMIPS dispone de 3 tipos de Instrucciones:

* Tipo I: instrucciones de acceso a memoria, únicamente las de carga y almacenamiento (“LOAD” y “STORE”).
* Tipo R: instrucciones aritméticas y lógicas, únicamente registro a registro (no admite operaciones con memoria).
* Tipo J: instrucciones de salto

El acceso a la memoria se hace únicamente con las instrucciones tipo I. Por esta razón a este tipo de máquinas se las identifica como máquina tipo “LOAD/STORE”.

El formato es muy regular, porque: Las instrucciones tienen todas el mismo tamaño (32 bits). Los campos con referencias son siempre los mismos.



**Modos de direccionamiento**

El nanoMIPS dispone formalmente de 2 modos de direccionamiento, pero si se consideran 2 casos especiales, se obtienen 2 modos más (4 en total).

* Inmediato: dato de 16 bits.
* Indirecto con desplazamiento: suma de registro más un desplazamiento (distinto de 0).
* Indirecto vía registro: si el desplazamiento es 0 el direccionamiento es indirecto vía registro.
* Directo absoluto: Si el desplazamiento es vía registro R0 (que vale 0), entonces se obtiene directo absoluto.

**Repertorio de instrucciones**

Para analizar el comportamiento de este procesador, se va a considerar un repertorio básico de 8 instrucciones: 2 de movimiento de datos con memoria, 5 aritméticas y logicas, y 1 de salto condicional.



Instrucciones tipo I: Load/store

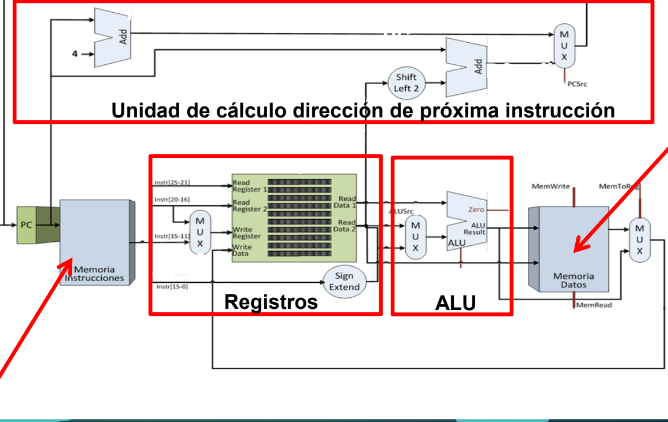
* RS: registro base
* RT: Registro de fuente o destino de la transferencia
* Desp: offset

Instrucciones tipo R: Aritmético/lógicas

* RS: operando 1
* RT: operando 2
* RD: resultado

Instrucciones tipo J: de salto

* RS: operando 1
* RT: operando 2
* Destino: es un desplazamiento respecto del PC (16 bits)

**Ruta de datos e instrucciones**

La imagen anterior muestra la ruta de los datos e instrucciones del nanoMips. Se pueden identificar:

Banco de registros:

* Está compuesto por 32 registros (el R0=0)
* Se pueden leer 2 registros al mismo tiempo (doble entrada)
* Se puede escribir en 1 solo registro

ALU:

* Es una unidad de cálculo típica de 2 entradas y 1 salida
* En la entrada se pueden seleccionar diferentes fuentes de datos

Unidad de cálculo de dirección de próxima instrucción:

* Calcula el próximo valor con el que se carga el PC
* Puede ser el de la instrucción consecutiva (PC+4), o sumando un desplazamiento (instrucción de salto con desplazamiento).

También se puede apreciar que existen 2 bancos de memoria:

Memoria de instrucciones:

* Solo contiene el programa a ejecutar
* Se accede únicamente para lectura, en la fase de búsqueda de la instrucción, y accedida solo a través del PC.

Memoria de datos:

* Contiene los datos a leer o escribir
* Solo se accede a través de las instrucciones LOAD y STORE

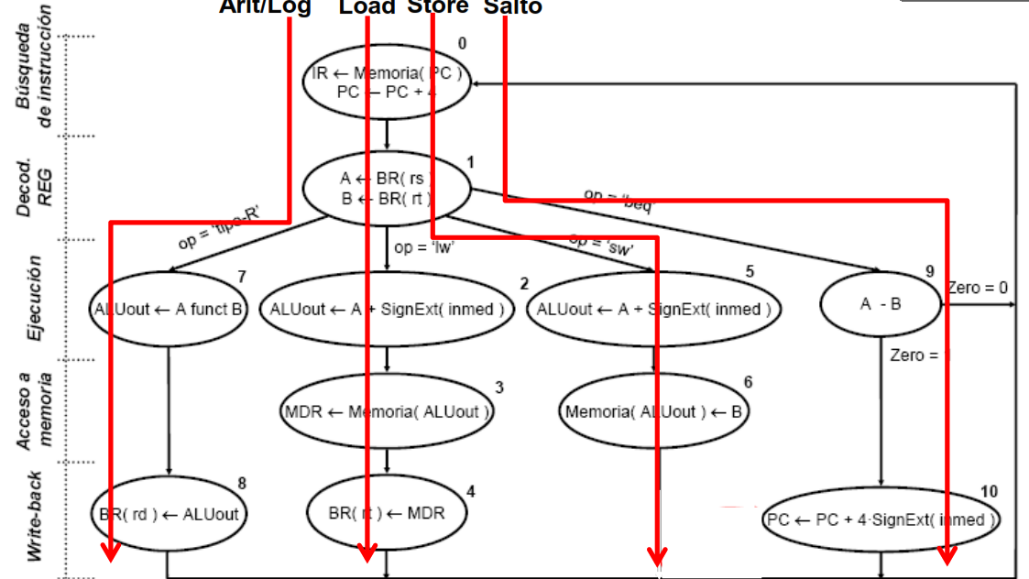
**Ruta de datos, instrucciones y control**

La Unidad de control (CU) es una unidad relativamente sencilla que captura la instrucción y a partir de ella genera las señales de control requeridas por los demás bloques operativos. En particular, la CU necesita el campo del código de operación (OPCODE) para determinar que tareas deberá realizar para completar la instrucción corriente.

El resto de los campos se usan para las identificaciones restantes.

Por ejemplo:

* El campo RS selecciona 1 registro en el banco de registros
* Idem los campos RT y RD (cuando corresponda).
* El resto de los campos se usan para completar las órdenes a dar a las Unidades funcionales restantes (por ejemplo la ALU).

**Diagrama de estados**

Tiene una secuencia de instrucción compuesta por 5 estados.

**Fase 1** - Búsqueda de instrucción (F, Fetch): ocurre en todas las instrucciones. Busca y lee la instrucción en la memoria (de instrucciones), Actualizar el PC (sumar 4 al valor actual del PC): cada instrucción ocupa 4 bytes, por lo que la próxima instrucción consecutiva está en la dirección actual + 4.

**Fase 2** - Decodificación (D, Decode) y acceso a registros: ocurre en todas las instrucciones. Se decodifica la instrucción (del campo CODOP), Se accede al banco de registros

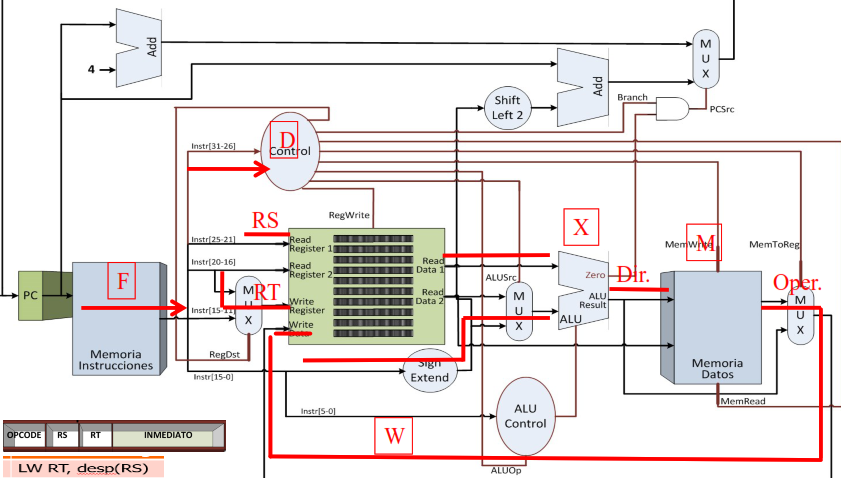
Opcional: Extensión del signo del offset para cálculo de la dirección efectiva.

**Fase 3 -** Ejecución (X, Execute): ocurre en todas las instrucciones. Se ejecuta la operación en la ALU.

**Fase 4** -Acceso a memoria (M, Memory Access), ocurre solo en las instrucciones de LOAD y STORE. Se accede a memoria

**Fase 5 -** Almacenamiento en Registro (W, Writeback): ocurre en las instrucciones que almacenan un dato en un registro

➢ También es en esta fase cuando puede calcular el desplazamiento a sumar al PC en instrucciones de salto.

**Secuencia de estados para una Instrucción LOAD**

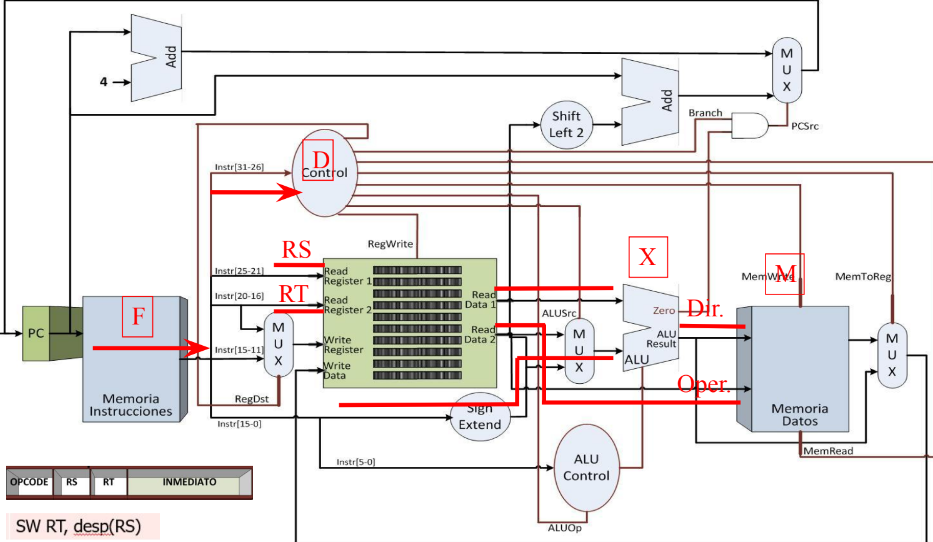
En la imagen anterior se pueden apreciar las 5 fase de ejecución:

**Instrucción LOAD:** la secuencia de fases es F-D-X-M-W(indicado como 0-1-2-3-4)**.**Requiere las 5 fases

**Instrucción STORE:** la secuencia de fases es F-D-X-M (indicado 0-1-5-6). Usa 4 fases, no tiene fase W

**Instrucción Aritmético/lógica:** la secuencia de fases es F-D-X-W (indicado 0-1-7-8)Usa 4 fases, no tiene fase M

**Instrucción de salto:** la secuencia de fases es F-D-X-W (indicado 0-1-9-10). Usa 4 fases, no tiene fase M

**Secuencia de estados para una Instrucción STORE**

En la imagen anterior se puede ver el proceso de ejecución de una instrucción de STORE.

**Fase F**: Lee la instrucción.

Los bits 26 a 31 son leídos por la Unidad de Control.

Los bits 21 a 25 identifican el registro RS (registro puntero)

Los bits 16 a 20 identifican el registro RT (destino o de escritura)

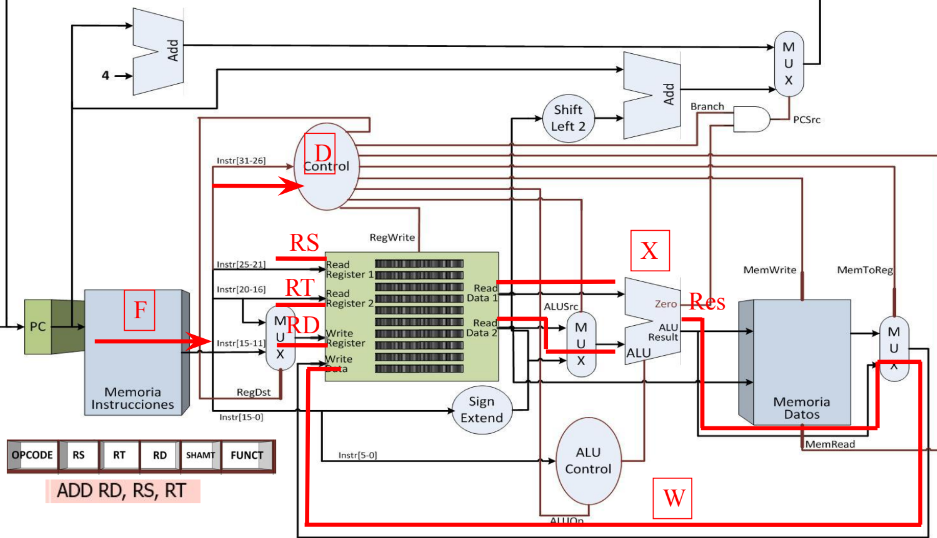
Los bits 0 a 15 son el desplazamiento a sumar al RS

**Fase D:** la UC decodifica la instrucción y se accede al registro RS.

**Fase X:** se calcula la dirección efectiva, donde se va a almacenar el operando cargado en RT, como el valor de RS más el desplazamiento (inmediato).

**Fase M:** con la dirección efectiva, se accede a la memoria de datos, y se carga el dato almacenado en RT.

**Fase W**: no hay.

**Secuencia de estados para una instrucción ADD**

En la imagen anterior se puede ver el proceso de ejecución de una instrucción de ADD.

**Fase F**: Lee la instrucción.

Los bits 26 a 31 son leídos por la Unidad de Control.

Los bits 21 a 25 identifican el registro RS (1er operando)

Los bits 16 a 20 identifican el registro RT (2do operando)

Los bits 11 a 15 identifican el registro de destino RD

**Fase D:** la UC decodifica la instrucción y se accede a los registros RS y RT.

**Fase X**: se suman los contenidos de los registros RS y RT

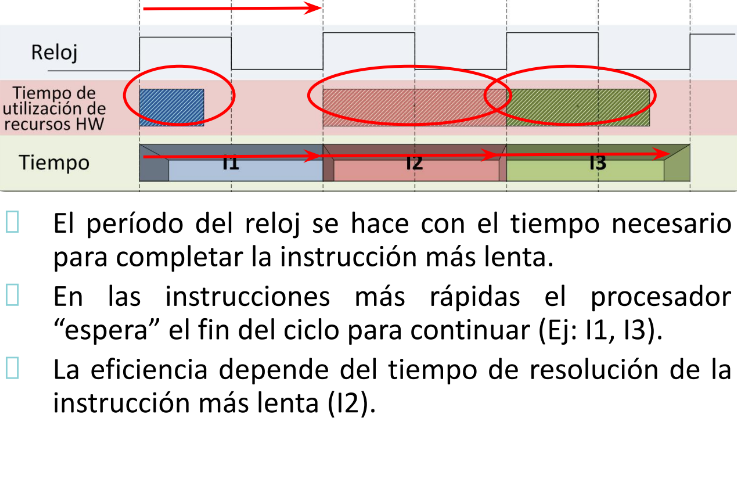
**Fase M**: no hay

Fase W: se escribe el resultado en RD.

**Ciclo de instrucción – Ejecución MONOCICLO**

En las descripciones anteriores, cada vez que el PC envía una dirección a la memoria de instrucciones, se accede a una nueva instrucción. Como no hay registros sincrónicos intercalados en las trayectorias de datos e instrucciones, el dato fluye por las unidades funcionales (registros, ALU, memoria, MUX, etc.) hasta que se complete la ejecución de la instrucción.

No se inicia un nuevo ciclo de instrucción hasta que el PC se carga con un nuevo valor (instrucción consecutiva o la de salto). Si el PC está sincronizado con un reloj (el de la CPU) significa que en cada ciclo de reloj se inicia una nueva instrucción. Este tipo de forma de operación se llama Ejecución MONOCICLO, y se muestra en la imagen siguiente.

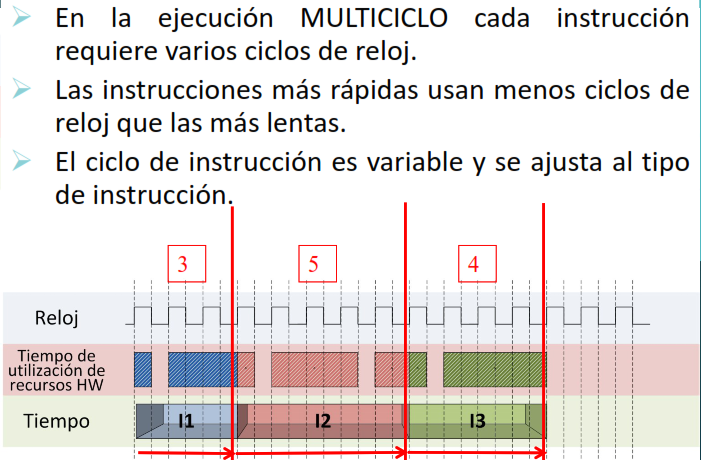


**Ciclo de instrucción – Ejecución MULTICICLO**

En la ejecución MONOCICLO el ciclo de reloj se hace en función de la instrucción “mas larga”, es decir la instrucción de LOAD.

Las instrucciones que son “mas cortas” tienen que esperar un tiempo hasta que se complete el período de reloj. Si la tasa de instrucciones cortas es mucho mayor que la de las largas, se puede perder mucho tiempo inútilmente. Una solución es fijar un ciclo de reloj más pequeño, y disponer que cada instrucción ocupe varios períodos de reloj.

Este tipo de forma de operación se llama Ejecución MULTICICLO, y se muestra en la imagen siguiente.



Segmentación y solapamiento del cauce (Pipelining)

La segmentación del cauce (Pipelining) consiste en descomponer el proceso de ejecución de las instrucciones en fases o etapas. Las fases o etapas son ejecutadas por unidades separadas y capaces de operar simultáneamente. Las instrucciones se van ejecutando a medida que se liberan las unidades y no necesitan esperar la terminación de la previa para comenzar a resolverse.

La segmentación del cauce (pipelining) es una forma particularmente efectiva de organizar el hardware de la CPU para realizar más de una tarea al mismo tiempo. Explota el paralelismo en el flujo secuencial de instrucciones.

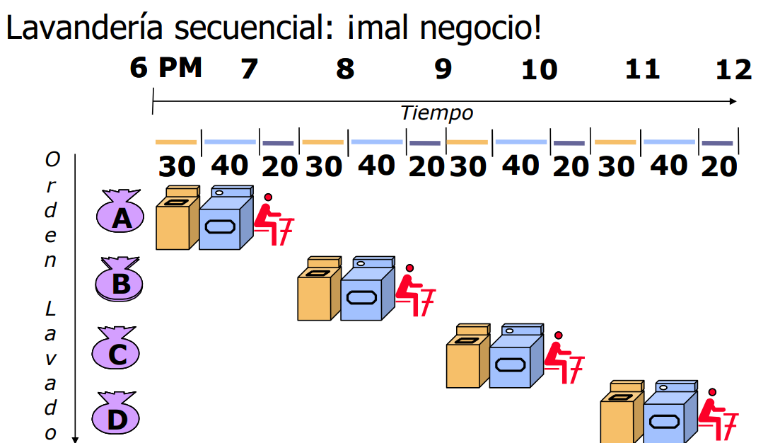
Para entender el concepto de segmentación de una tarea repetitiva se puede analizar un “proceso” de una lavandería que requiere 3 etapas:

➢ Lavado, que requiere 30 minutos

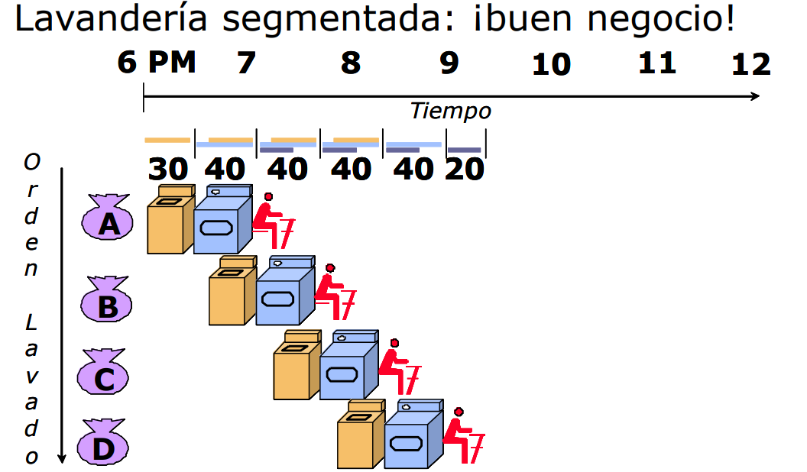
➢ Secado, que requiere 40 minutos

➢ Despacho, que requiere 20 minutos

Las 3 etapas se ejecutan en 3 sectores (estaciones) separados. La primera forma de operar es ejecutar el proceso en forma puramente secuencial, cuando se termina con un encargue recién se empieza con el siguiente.

Este modelo secuencial se muestra en la figura siguiente.

Como las unidades funcionales Lavado, Secado y Despacho son unidades distintas, pueden operar en forma simultánea.

Si se superponen (solapan) las tareas de las 3 unidades funcionales es posible obtener una mejora sustancial en el tiempo total de ejecución de los 4 encargues (A, B, C y D). Esa situación se muestra en la figura siguiente.

En el ejemplo anterior se puede observar que cada tarea lleva exactamente el mismo tiempo: 30+40+20 = 90 minutos.

➢ La mejora se obtiene debido al solapamiento de las tareas de las distintas unidades funcionales.

➢ Solapando las tareas A, B, C y D, se consigue realizarlas en 3horas 30 minutos, en lugar de las 6 horas que se tardaba operándolas en forma secuencial.

➢ Se puede aplicar este concepto al proceso repetitivo de ejecutar una secuencia de instrucciones.

Supongamos que se tiene una máquina que resuelve las instrucciones en 3 fases:

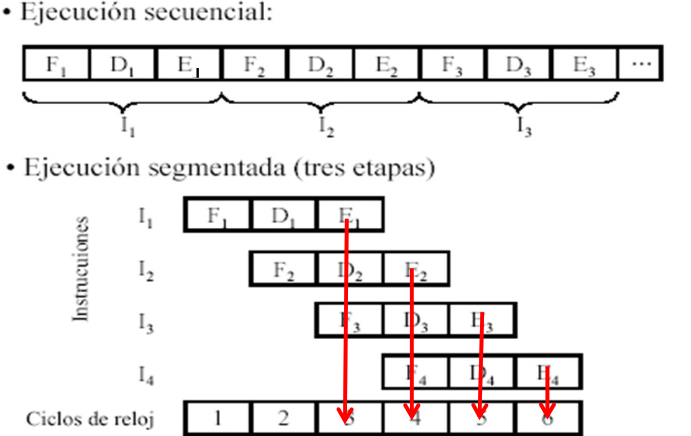
* Búsqueda o Captación (F, Fetch)

1. Acceso a memoria (búsqueda de la instrucción)
2. Incremento del PC

* Decodificación (D, Decode)

1. Decodificación de la instrucción
2. Obtención de los operandos

* Ejecución (E, Execute)

1. Si es procesamiento: ejecución en la ALU
2. Si es acceso a memoria: obtención de la direc efectiva
3. Si es salto: cálculo del destino y decisión de salto (s/n)

En el procesador con ejecución secuencial, se tardan 9 ciclos de reloj para completer 3 instrucciones.

En el procesador segmentado de la figura anterior, de 3 etapas, se tardan 5 ciclos de reloj para completar las mismas 3 instrucciones. En ambos casos, cada instrucción require 3 ciclos de reloj.

Si la máquina tiene 4 fases para resolver una instrucción:

➢ Búsqueda o Captación (F, Fetch)

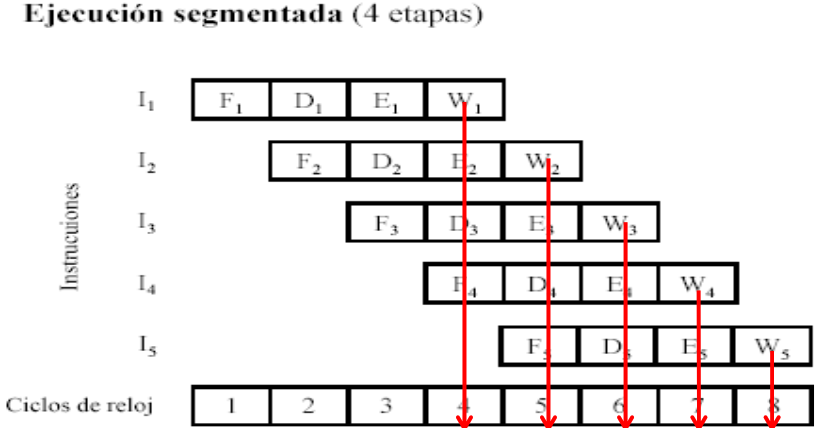
➢ Decodificación (D)

➢ Ejecución (E)

➢ Escritura o actualización (W)

La segmentación queda como se muestra en la figura siguiente.

Ejecución segmentada en una máquina de 4 etapas F, D, E, W, de una secuencia de 5 instrucciones.



En el procesador segmentado de 4 etapas de la figura anterior, se tardan 8 ciclos de reloj para completar 5 instrucciones. En el procesador con ejecución secuencial se tardarían 5x4=20 ciclos de reloj para completer las 5 instrucciones. En ambos casos, cada instrucción require 4 ciclos de reloj.

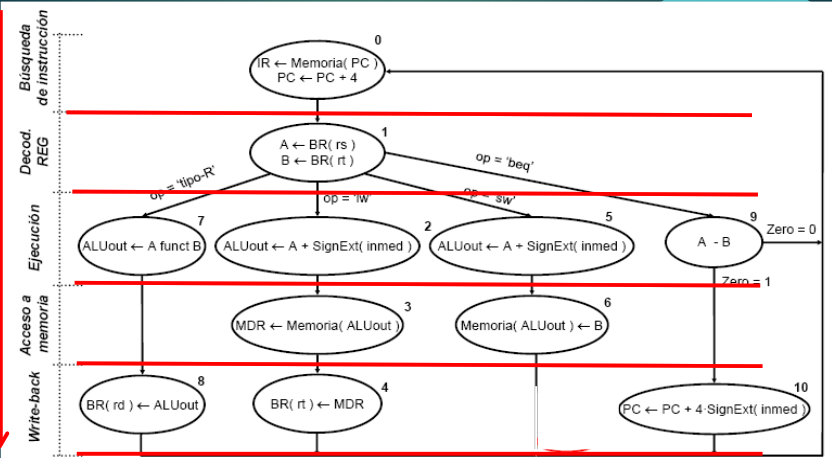
**Conclusiones:**

El comportamiento de un procesador ejecutando una secuencia de instrucciones se comporta de una forma similar a una línea de montaje en una planta de manufactura. Cada instrucción pasa, durante su ejecución, por varias etapas, en cada una se realiza solo una parte del todo.  
La entrada de una nueva instrucción puede hacerse antes de que se terminen las anteriores.  
Por lo tanto, varias instrucciones están siendo manipuladas simultáneamente, cada una en un estado de ejecución distinto.

Dado que el procesador tarda menos tiempo en resolver un conjunto de instrucciones, la segmentación mejora las prestaciones (a nivel de diseño del hardware). Como el programador no interviene en la segmentación y solapamiento de las instrucciones, la segmentación es “invisible” (por ahora) al programador.

Se puede usar tanto en procesadores RISC como CISC. El diseño de procesadores segmentados tiene gran dependencia del repertorio de instrucciones.

Se puede aplicar el concepto de Segmentación en el nanoMIPS. De acuerdo a lo visto, el proceso de ejecución de las instrucciones en el nanoMIPS está compuesto por 5 fases o etapas. En la imagen siguiente se puede apreciar la segmentación de las fases de ejecución de una instrucción.

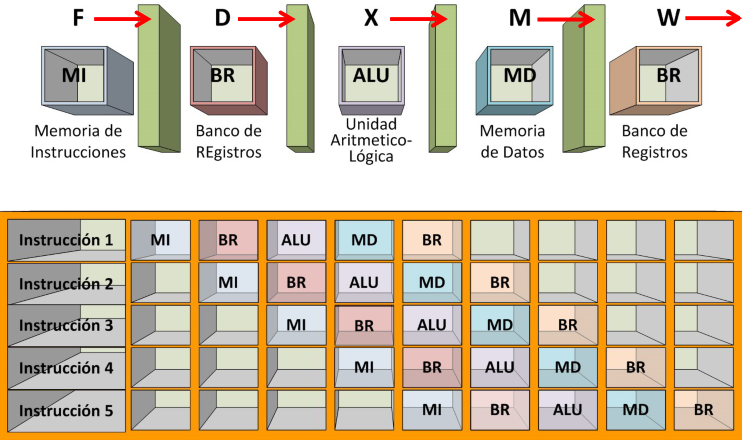


Los 5 segmentos del nanoMIPS son:

F: búsqueda de la insttrucción en la MI  
D: decodificación y acceso a registros en el BR  
X: ejecución de la ALU  
M: acceso a la memoria de datos MD  
W: escritura del resultado en el banco de registros BR

Y los recursos de la máquina son:  
MI: memoria de instrucciones  
BR: banco de registros  
ALU: ALU  
MD: memoria de datos

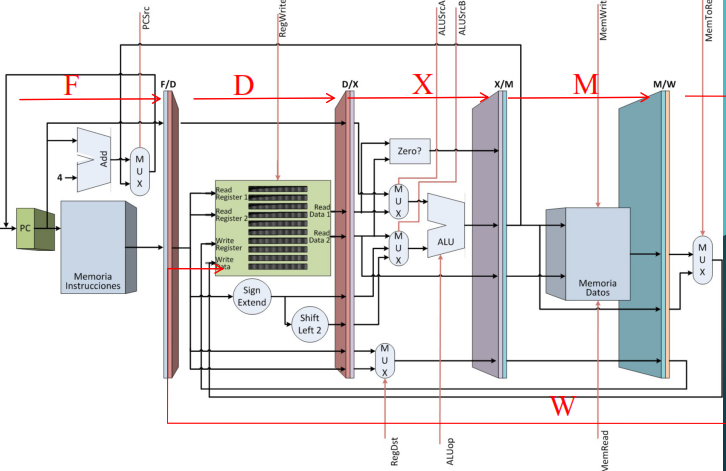
Para poder aplicar el concepto de segmentación en el nanoMIPS, hay que dividir (segmentar) el cauce de los datos en etapas.  
La forma de hacer esto es usando registros sincrónicos entre cada etapa. Los datos avanzan, de etapa a etapa, en cada ciclo de reloj, que es cuando el registro (de separación de la etapa) copia a la salida lo que tiene en su entrada.  
Los segmentos en que queda dividido el cauce de datos, corresponden a las 5 fases de ejecución de la instrucción: F, D, X, M y W.

Este proceso se puede apreciar en la siguiente figura.

Se considera que cada instrucción require 5 ciclos de reloj, uno por cada segmento (F-D-X-M-W). En cada ciclo se incorpora una nueva instrucción, sin haber completado las anteriores.  
La primera instrucción se termina en el período de reloj 5, hasta ese ciclo no se había resuelto ninguna instrucción. A partir del quinto ciclo, se termina 1 instrucción por ciclo.

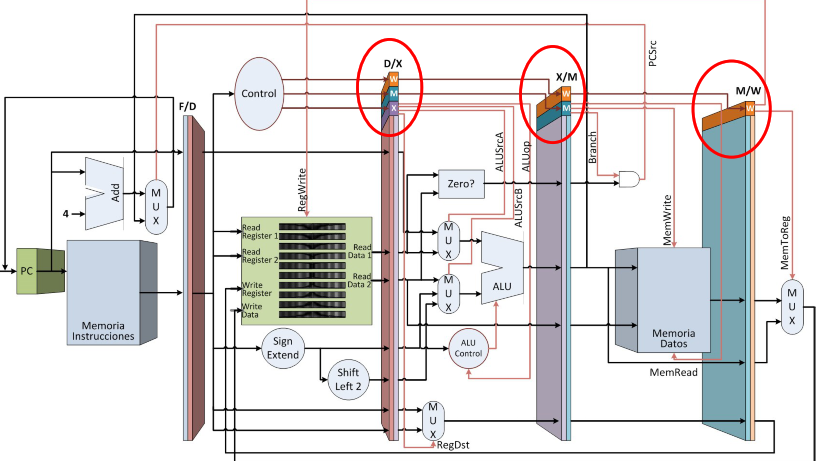
Notar que en el quinto ciclo el procesador está ejecutando 5 instrucciones simultáneamente, pero en distintas fases.

Para implementar la segmentación del cauce se requiere intercalar registros entre cada etapa.  
Los registros (sincrónicos) separan las unidades funcionales, para que puedan operar con distintas instrucciones.  
En cada ciclo de reloj el dato avanza de una unidad funcional a la siguiente en el cauce, hasta completar las fases requeridas por a instrucción.  
El cauce de datos queda segmentado de la siguiente manera.



Así como se segmenta el cauce de los datos, también se debe hacer lo mismo con las señales que controlan las unidades funcionales: MI, BR, ALU, MD, BR.

Eso es así porque cada Unidad funcional está operando con diferentes instrucciones (en diferentes estados de ejecución).   
Por ejemplo, mientras la unidad funcional BR (Banco de registros) está buscando los operandos de la instrucción I1, la unidad funcional MI (Memoria de instrucciones) estará buscando la instrucción I2.   
Las señales de control se segmentan igual que el cauce de datos, usando registros sincrónicos entre cada etapa, como se muestra en la figura siguiente.



Segmentación del cauce de datos y de control.

Análisis de la Segmentación

El máximo rendimiento teórico se obtiene cuando se completa una instrucción en cada ciclo de reloj. En esas condiciones todas las unidades funcionales están trabajando simultáneamente con distintas instrucciones.

Si K es el número de etapas del cauce, entonces:

Vel. procesador segmentado = K x Vel. secuencial

El incremento potencial de la segmentación del cauce es proporcional al número de etapas del cauce.

Notar que no se mejora la velocidad de ejecución de la instrucción, la segmentación incrementa la productividad (throughput), es decir la cantidad de instrucciones resueltas en un período de tiempo determinado.

Para el análisis anterior se han hecho algunas suposiciones:

* Todas las tareas o segmentos duran la misma cantidad de ciclos de reloj (tiempo).
* Todas las instrucciones siempre pasan por todas las mismas etapas.
* Todos las etapas pueden ser manejadas en paralelo (no hay conflictos para usarlas simultáneamente).
* No se consideraron instrucciones de salto.

Para hacer un análisis más preciso se requiere estudiar el comportamiento de la segmentación, pero en condiciones más realistas.

Las primeras correcciones a hacer son:

1. No todas las instrucciones necesitan todas las etapas. Ej: en el nanoMips la instrucción SW RT, inmed(RS) no utiliza W. Ej: en el MSX88: un MOV AX,mem ; no requiere X
2. No todas las etapas pueden ser manejadas en paralelo. Ej: si la memoria no estuviera dividida y fuera una sola, los segmentos F (búsqueda de la instrucción) y M (acceso a memoria) accederían ambos a la memoria (única) de datos e instrucciones.
3. Los programas tienen instrucciones de salto.

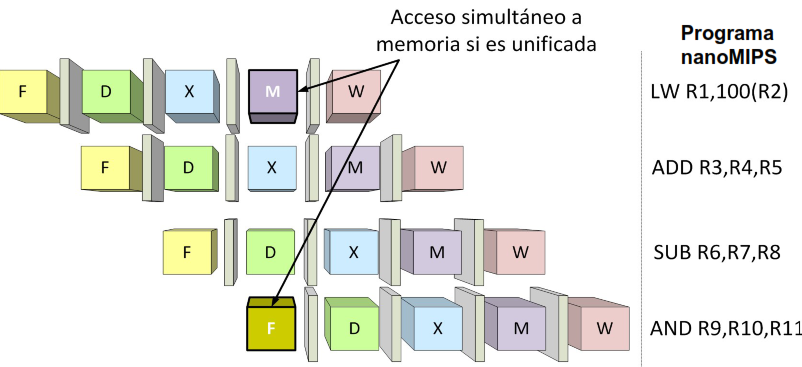
Los conflictos que aparecen si se tienen en cuenta las correcciones anteriores, se denominan riesgos.

Existen 3 tipos de riesgos:

1. Estructurales: son conflictos provocados por el uso de los “recursos”. Los recursos típicamente son: memoria, ALU, registros.
2. Por dependencia de datos: son conflictos originados entre 2 o más instrucciones que comparten un mismo dato. Por ejemplo, una instrucción produce un resultado que lo necesita otra, ambas instrucciones dentro del cauce de datos.
3. Por dependencia de control: son conflictos que ocurren cuando la ejecución de una instrucción depende de cómo se ejecute otra (ej.: un salto y los 2 posibles caminos).

**Ejemplo de conflicto por uso de un recurso: la Memoria**

Si la memoria del nanoMIPS fuera una sola (instrucciones y datos) habría un conflicto cada vez que se solapan los ciclos M y F.



**1) Riesgos Estructurales**

En la imagen anterior se muestra la situación de un conflicto entre las fases F y M si la memoria fuera una sola. Por esta razón, la memoria del nanoMIPS está dividida en MI y MD, de manera de reducir los conflictos por accesos a memoria.

Otra opción para evitar el conflicto por el acceso a un recurso es retardar (retrasar) la ejecución de la tarea los ciclos de reloj necesarios hasta que desaparece el conflicto.

Pero al perder ciclos de reloj se pierde tiempo y baja la performance del procesador. Y además, el retraso puede generar nuevos conflictos que también deben resolverse de alguna manera.

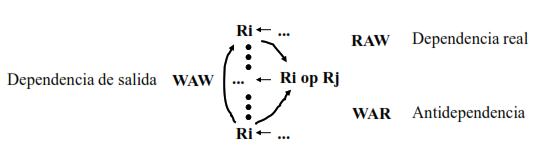
Esta solución se muestra en la figura siguiente.



Se retrasa el inicio de la instrucción AND R9,R10,R11 en 1 ciclo (ciclo de parada) para evitar el conflicto con la instrucción LW R1,100(R2).

**2) Por dependencia de datos:**

Hay riesgos de conflictos por dependencia de datos, cuando un dato es usado en 2 o más segmentos del cauce. Los operandos fuente o destino de una instrucción no están disponibles en el momento en que se necesitan en una etapa determinada del cauce.

Se pueden dar 3 tipos de dependencias:

Los 3 tipos de dependencias de datos son:

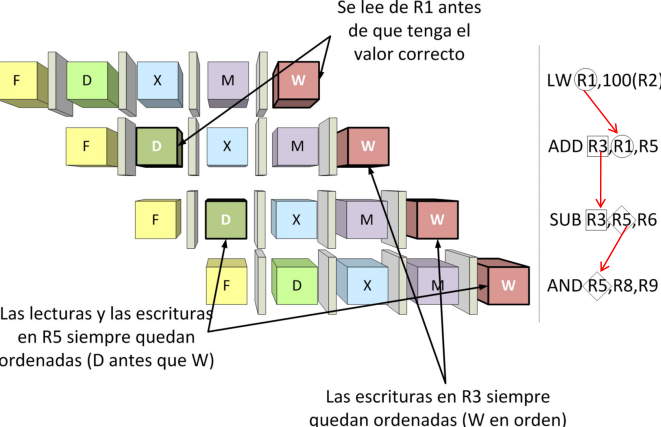
**Lectura después de Escritura (RAW, dependencia verdadera):** Una instrucción escribe un dato que otra lee posteriormente

**Escritura después de Escritura (WAW, dependencia en salida):** una instrucción escribe un dato que otra escribe posteriormente. Sólo ocurre si se permite que las instrucciones se adelanten unas a otras (alteración en la secuencia de ejecución de instrucciones)

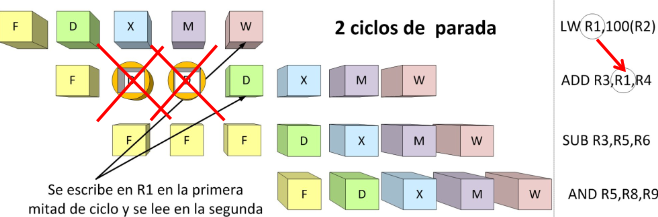
**Escritura después de Lectura (WAR, anti dependencia):** una instrucción lee un dato que otra escribe posteriormente**,** no se puede dar en nuestro cauce simple.

**Ejemplo de conflicto por dependencia de datos**

Se muestra una secuencia de 4 instrucciones con los “3 tipos de dependencias”.



**Ejemplo de resolución del conflicto por dependencia de datos**

Se retrasa la decodificación D de la instrucción ADD R3,R1,R4 3 ciclos para esperar tener el dato disponible de la instrucción LW R1,100(R2).  
Se considera que la escritura W de la instrucción LW R1,100(R2) se hace en el primer medio ciclo y se puede leer por la instrucción ADD R3,R1,R4 en el segundo semiciclo.

**3) Por dependencia de control:**

Son riesgos que pueden ocurrir cuando se va a ejecutar una instrucción de salto condicional. Una instrucción tiene que calcular el nuevo valor que modifica el valor del PC. La próxima instrucción no puede comenzar hasta que no se resuelva el salto.

Una forma de resolver el conflicto es retrasar varios ciclos (3 ciclos) la próxima instrucción hasta que se resuelva el cálculo de la instrucción de salto, como se puede apreciar en la figura siguiente.

