Unidad 8 - Organización de la Memoria

1. Almacenamiento de Programas

Como ya vimos, el ensamblador traduce el programa de lenguaje ensamblador a código de máquina

- Asigna una dirección a cada instrucción y busca los símbolos (etiquetas y nombres de variables).
- Genera el *código de máquina* y la *tabla de símbolos*.
  El código de máquina y la tabla de símbolos se almacenan en el **archivo objeto**, que puede ser ejecutado por el procesador una vez cargado en memoria. Luego el enlazador toma los archivos objeto, los recursos necesarios (bibliotecas) y enlaza el código objeto con las bibliotecas que necesita, esto produce finalmente un **fichero ejecutable**. Este enlace se hace en tiempo de ejecución para programas enlazados dinámicamente.

Para **cargar y ejecutar** un programa, el sistema operativo lee el *archivo ejecutable* del *dispositivo de almacenamiento* y lo carga en la memoria de la computadora,

- En **procesadores convencionales**, se ubica en la memoria disponible y la gestión queda a cargo del programador.
- En **procesadores avanzados**, se ubica en el contexto de un **proceso**, y la gestión queda a cargo de la unidad de manejo de memoria.
  Como ya vimos, la ubicación de cada segmento de memoria del programa (código, datos, variables y pila) depende del *ISA*.

Para ejecutar un programa, las únicas formas de almacenamiento a las que puede acceder el procesador son los **registros** y la **memoria principal** (no lo dice, pero es básicamente la RAM). Los datos guardados en otros dispositivos de memoria son intermediados por la memoria principal o su acceso es gestionado por la unidad de manejo de memoria (MMU, que ya hablaremos mas tarde).

La unidad de memoria únicamente "ve"

- Direcciones de lectura.
- Requerimientos de escritura (WriteEnable - WE), y sus correspondientes direcciones y datos.

1.1. Jerarquía de Almacenamiento

Para mejorar el desempeño de la computadora (que depende tanto del desempeño del procesador como de su sistema de memoria), el sistema de almacenamiento se organiza en una jerarquía según **velocidad**, **costo**, **volatilidad**. De este modo, definimos cuatro niveles de almacenamiento:

- Interno: son los registros del procesador, el almacenamiento más rápido, aunque pequeño.
- Principal: memoria del sistema y memorias cache, circuitos integrados para uso inmediato.
- Masivo on-line (secundario): disco duro, de estado sólido y flash USB. El procesador necesita de canales de I/O para accederlos y transferir datos. Estos retienen datos al cortar la alimentación.
- Masivo off-line (terciario): unidades de almacenamiento masivo extraíble (cintas magnéticas, discos ópticos). Archivan información a la que rara vez se accede y/o transfieren cantidades grandes de datos.

2. Direcciones Lógicas y Físicas

Una dirección de memoria identifica de forma única una posición en la memoria.

Como acabamos de repasar, un programa pasa a través de diferentes pasos (compilación, enlazado y carga) antes de ser ejecutado. A lo largo de estas etapas, las direcciones son representadas de diferentes maneras.

- El *código fuente* (lo que escribimos) típicamente utiliza direcciones simbólicas (los nombres de las variables, por ejemplo).
- El *compilador* típicamente relaciona la dirección simbólica con una **dirección relocalizable**.
- El *enlazador* y/o *cargador* (no se que es esto), relacionan la *dirección relocalizable* con una **dirección física** (o absoluta).

Esta relación entre las direcciones simbólicas de las instrucciones con las direcciones en memoria se pueden calcular durante:

- La *compilación*, si se conoce la posición en memoria donde se ubicará el programa.
- La *carga*, si no se conoce la dirección. El compilador genera una dirección relocalizable cuyo **inicio** (dirección real de referencia ponele) cambia cada vez que se carga.
- La *ejecución*, si el proceso puede moverse durante la ejecución desde un lugar de memoria a otro.

Entonces, vemos que hay dos tipos de direcciones:

- **lógicas**: es la dirección vista por el usuario (programa) y se utiliza como referencia para acceder a la dirección física. Es el único tipo de dirección con las que trabaja el programa.
- **físicas**: es una posición en la unidad de memoria del sistema.

Y estas a su vez definen dos espacios de direcciones: **espacio de direcciones virtuales** (conjunto de direcciones a las que puede referenciar el programa) y **espacio de direcciones físicas** (conjunto de direcciones físicas a las que accede el programa).

Esta idea de **referenciación** de direcciones físicas por medio de direcciones lógicas es la que nos obliga a pensar en un sistema especializado en **mapear** durante la ejecución las direcciones lógicas con las físicas. Este sistema recibe el nombre de **Unidad de Manejo de Memoria** (Memory Management Unit - MMU).
Ya la veremos en más profundidad, pero las funciones que desempeña son:

- Gestión de acceso a memoria por parte de los programas.
- Automatización de las transferencias de datos entre niveles de jerarquía de memoria.
- Aislación de la ejecución de los programas.
- Ampliación de los espacios de direcciones de la arquitectura, conectando los procesadores con memorias más grandes.

3. Manejo de Memoria

Veremos las técnicas aplicadas a gestionar la memoria, para **asignar dinámicamente** memoria necesaria a los programas según la soliciten y liberarla a medida que no sea necesaria. En general se hace un seguimiento del estado de cada bloque de memoria (libre o asignado) para determinar la asignación de los mismos a los procesos (programas), es decir **cuánta, cuándo y dónde se asigna**.

3.1. Paginación

Es una técnica que divide la memoria física y el espacio de direcciones lógicas en **unidades de tamaño fijo** llamadas *frames* (para direcciones físicas) y *páginas* (para direcciones lógicas), asociando a cada frame una página **de modo que el espacio de direcciones lógicas sea continuo**.
A cada página se le asigna un frame de tamaño similar en la memoria principal. Los frames no tienen que ser contiguos

en la memoria física, por o que cada proceso puede estar disperso en toda la memoria y verse como una colección de frames no-contiguos.

Para realizar un seguimiento de dónde se encuentra cada página dentro del espacio de direcciones físicas y su estado (disponible o no disponible), se utiliza una **tabla de páginas**.
La dirección lógica es dividida en partes:

- Número de página (p): utilizado como índice en la *tabla de páginas*, que contiene la dirección física base del frame asociado a la página.
- Offset de la página (d): combinado con la dirección base, el offset define la dirección física en la memoria.

Para **implementar** esta tabla de página, una opción sería usar un banco de registros (hardware) dedicado a almacenarla:

- Ventajas: traducción rápida.
- Desventajas: cambio de contexto lento, ya que la tabla completa debe ser recargada. El espacio necesario en la tabla es proporcional al espacio de direcciones virtuales y el número de procesor, y el tamaño de la tabla es limitado por el tamaño del banco de registros dedicado.

Por esto, se opta por almacenar las tablas de página en la memoria principal, y por cada tabla contaremos con:

- Registro de base de la tabla de página (PTBR): almacena la dirección física de la base de la tabla.
- Registro de longitud de tabla de página (PTLR): almacena la longitud/tamaño de la tabla de página.
  De esta forma, cambiar la tabla de página consiste en simplemente cambiar estos dos registros.

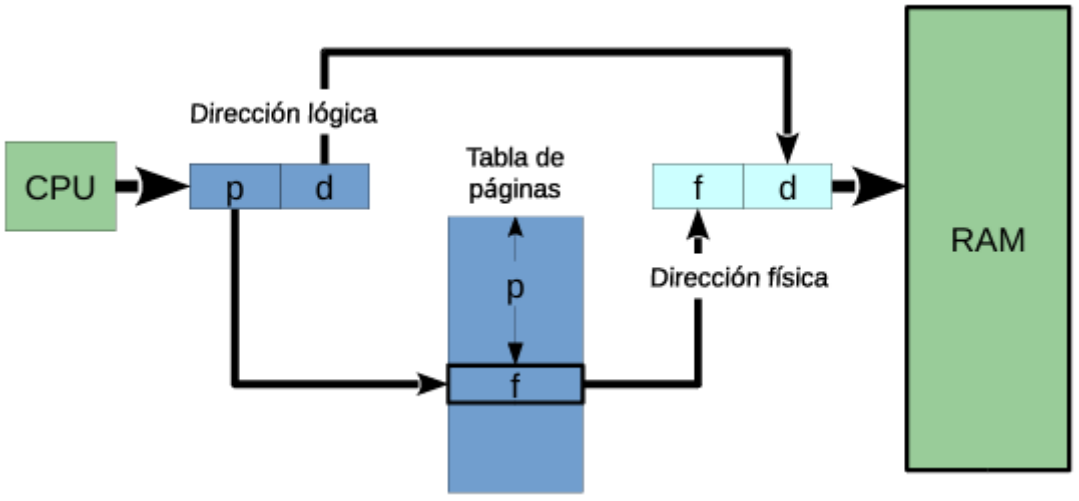


Tabla de Páginas

3.1.1. TLB

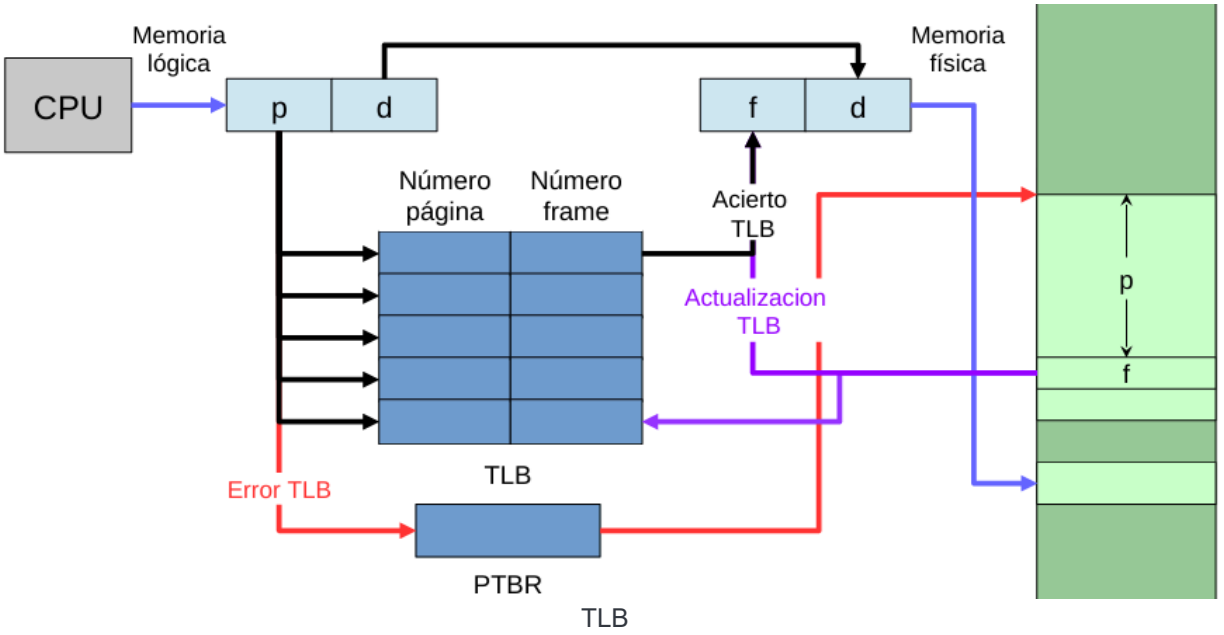
La **Translation Look-aside Buffer** (TLB) es una tabla de rápida lectura basada en memorias asociativas, similar (si no igual) a una cache. Almacena las páginas y frames utilizados con mayor frecuencia. Y aunque tiene una elevada velocidad de traducción, también tiene un tamaño pequeño. En una arquitectura Harvard o Harvard modificada pueden existir separadas una **TLB para datos** (DTLB) y una **TLB para instrucciones** (ITLB). Al igual que las cach'e, las TLB pueden tener múltiples niveles, uno rápido y pequeño totalmente asociativo (L1-TLB) y otro más grande que es más lento (L2-TLB).

3.1.2. Paginación Multiple

Otro problema con el tamaño de las páginas es el de encontrar un espacio de direcciones físicas contiguo lo suficientemente grande para la tabla. Una solución a este problema consiste en subdividir la tabla y "diseminarla" en partes no contiguas, generando un esquema de páginas de múltiples niveles.

3.1.3. Protección de la Memoria

El bit de protección se utiliza para indicar si una página se puede leer, se puede escribir, se puede ejecutar, entre otras restricciones. Asociando estos bits de protección a cada frame en la tabla de páginas se logra la protección de la memoria.



3.2. Segmentación

Es una técnica que divide la memoria en áreas (llamadas **segmentos**) que corresponden a agrupaciones lógicas de información, es decir que la memoria se organiza desde la perspectiva del programador, como una colección de distintos tipos de información sin un orden preestablecido.

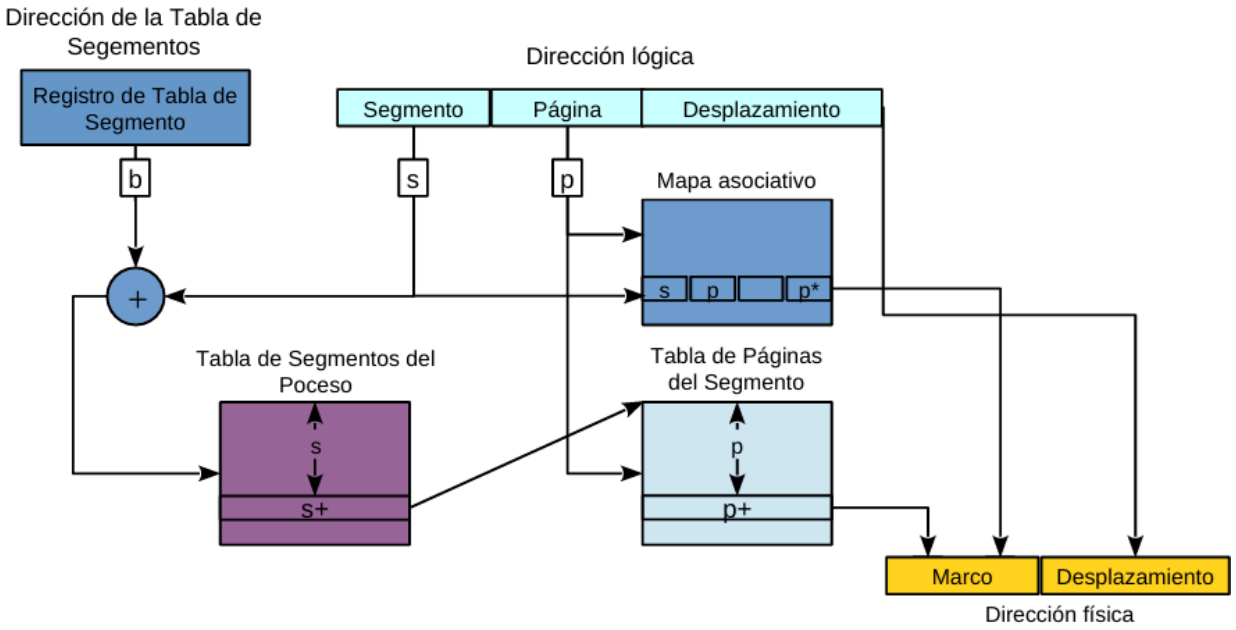
Se organiza el espacio de direcciones lógicas como un conjunto de segmentos, donde cada dirección lógica identifica tanto la etiqueta del segmento como el desplazamiento dentro de él. Estos segmentos son creados automáticamente por el compilador reflejando el programa de entrada. Por ejemplo: un compilador de C podría crear **segmentos de código**, segmento de **variables globales**, **segmento de heap y pilas**, además de las pilas utilizadas por cada hilo y las bibliotecas.

Similar a cómo sucede para páginas, contaremos con una Tabla de Segmentos, donde cada entrada contendrá los siguientes campos del segmento:

- Dirección física base del segmento.
- Tamaño del segmento.
  A su vez, cada tabla de segmento se define por un Registro Base de Tabla (STBR) que apunta a la dirección física base de la tabla, y por un Registro de Tamaño de Tabla (STLR), que indica el número de segmentos válidos de la tabla.

3.2.1. Fragmentación

La desventaja de la segmentación es que sugre de **fragmentación** y por esto necesita ser *compactado*.



Paginacion y segmentacion

3.3. Virtualización

Es una técnica que abstrae las direcciones lógicas de los recursos realmente disponibles, aumentando el tamaño del espacio de direcciones lógicas más allá del de las direcciones físicas disponibles. Esto lo hace mediante intercambios con el almacenamiento secundario.

El sistema puede intercambiar temporalmente un proceso entre la memoria principal y secundaria de acuerdo a los recursos disponibles, manteniendo una **cola de procesos** que están siendo ejecutados que tengan una imagen en la memoria secundaria.
La desventaja más obvia es que la velocidad del almacenamiento secundario crea un cuello de botella.

4. MMU

La **Unidad de Manejo de Memoria** (Memory Management Unit - MMU) es el dispositivo responsable de manejar los accesos a la memoria desde el procesador.
Entre sus funciones se encuentran:

- Traducción de direcciones.
- Protección de la Memoria.
- Control de Caché.
- Conmutación de Bancos de Memoria (en arquitecturas más simples).

Por ejemplo: el sistema operativo puede usar la MMU para impedir el acceso a la memoria por un programa.