MODULO I: INTRODUCCION

Tema 1: Introducción a la estructura de computadores

Objetivos:

- Describir el contenido general de la asignatura y situarlo en el conjunto de materias que abordan el estudio de un computador digital.
- Conocer las características fundamentales del funcionamiento de un computador con arquitectura von Neumann y analizar las principales mejoras añadidas a dicha arquitectura: interrupciones, memoria caché y memoria virtual.
- Diferenciar la tecnología, la estructura y la arquitectura de un computador, y hacer un recorrido general sobre las mejoras de rendimiento basadas en el paralelismo.
- Introducir los lenguajes de descripción hardware como instrumentos de especificación y simulación.

Contenido:

1. Objetivos de la asignatura

La arquitectura de un computador está constituida por el conjunto de funcionalidades disponibles para un programador que utiliza el lenguaje máquina, básicamente, el repertorio de instrucciones y los elementos de memoria referenciados desde él, es decir, los registros generales y la memoria principal. Las funcionalidades de una arquitectura se pueden conseguir con diferentes organizaciones internas o estructuras, diferenciándose unas de otras fundamentalmente en los parámetros de rendimiento y el coste. Finalmente, la estructura de un computador se puede implementar con diferentes tecnologías, siendo nuevamente el coste y el rendimiento los elementos diferenciales. *Arquitectura*, *estructura* y *tecnología* constituyen, pues, tres niveles de estudio del hardware de un computador.

En esta asignatura abordaremos el estudio de la organización o estructura interna de un computador. Para ello la materia la dividiremos en cuatro módulos. En el primero realizaremos una introducción general a la estructura de computadores. En el segundo estudiaremos la arquitectura del repertorio de instrucciones (*ISA*), arquitectura que define la interfaz hardware-software de la máquina. En el tercero veremos toda la jerarquía de memoria de una máquina y su gestión. Finalmente, en el cuarto estudiaremos las unidades de entrada/salida, los periféricos y los

buses de comunicación. El estudio de la unidad aritmético-lógica y la unidad de control se aborda en la asignatura *Ampliación de Estructura de Computadores*.

Para situar con mayor precisión el objeto de estudio de esta asignatura analizaremos en el apartado siguiente los diferentes niveles de descripción que se suelen contemplar en el estudio de un computador digital.

2. Niveles de descripción de un computador

La estrategia que habitualmente se utiliza para abordar el estudio de los sistemas complejos consiste en especificarlos a diferentes niveles de abstracción. Cada nivel se caracteriza por:

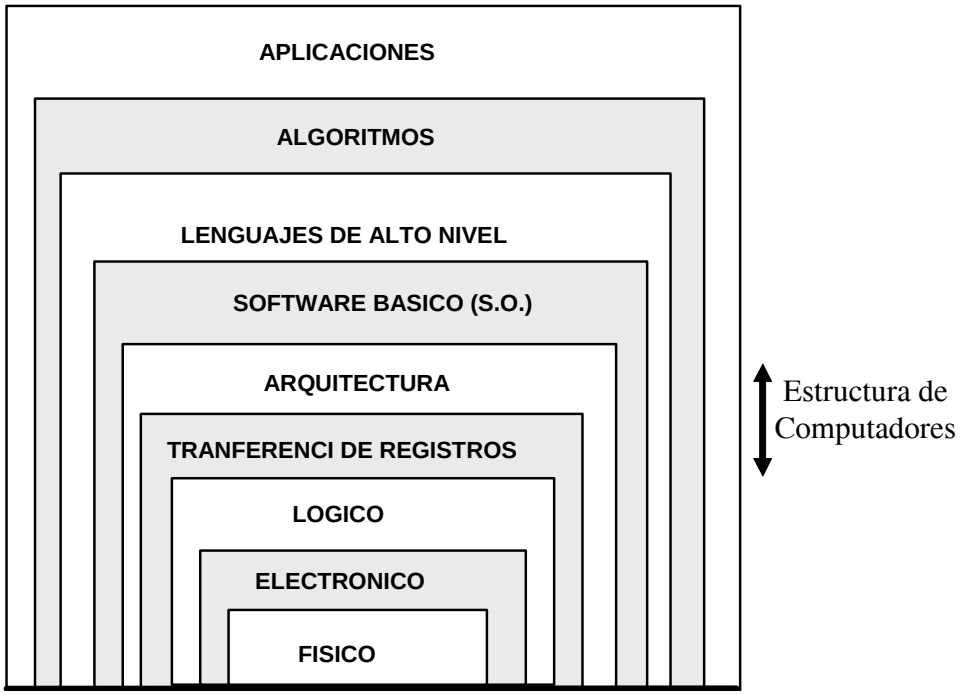
A) Unos elementos de entrada, es decir, disponibles para el diseño en este nivel, y que proceden del nivel inmediato inferior.

B) Unos elementos de salida, es decir, objetivos del diseño en este nivel, y destinados al nivel inmediato superior.

C) Una metodología de análisis y síntesis de los elementos de salida en términos de los de entrada.

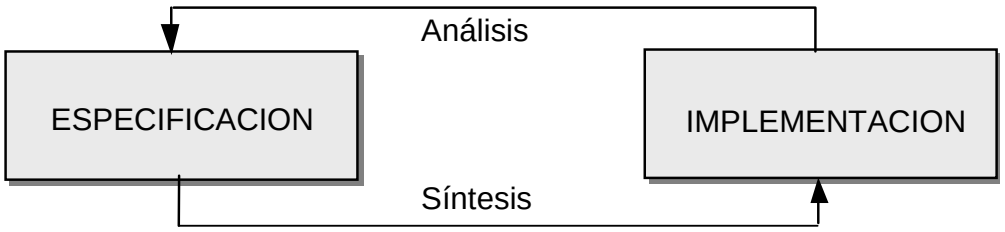
Con esta estrategia, la complejidad del sistema queda dividida, acotada y organizada en las complejidades parciales de cada nivel, dentro de cuyos límites se puede aplicar una metodología propia de estudio.

Al computador digital como sistema artificial complejo se le ha aplicado esta estrategia. En nuestro caso consideraremos los siguientes niveles de abstracción dentro del estudio de un computador digital:

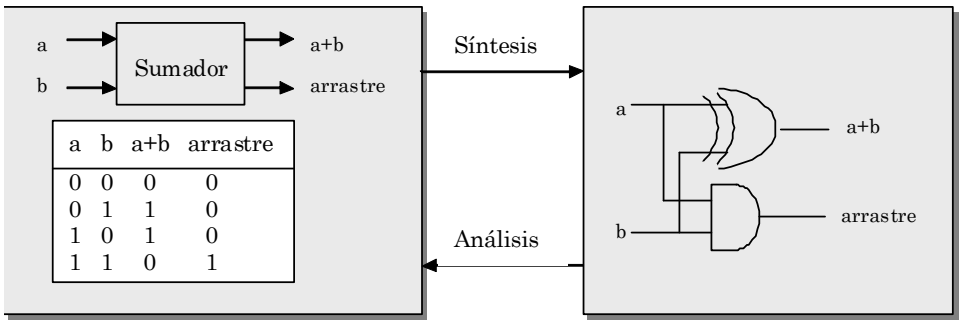


Cada nivel se corresponde con la visión que tiene del sistema un tipo determinado de usuario, y en cada uno podemos considerar dos procesos de estudio diferentes, el *análisis* y la *síntesis*. El análisis parte de la implementación del sistema a un cierto nivel en términos de elementos básicos del nivel inferior, y llega a determinar la función del mismo, es decir, su

especificación. El sentido de la *síntesis* es el opuesto, parte de la especificación de un sistema y obtiene su implementación en función de los elementos básicos del nivel inferior. En la siguiente figura hemos representado gráficamente esta relación:



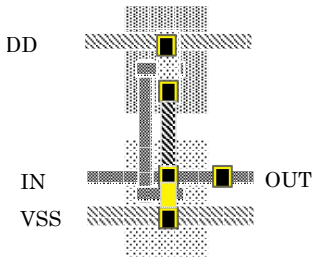
Ejemplo:



Comentaremos en los sub-apartados siguientes cada uno de estos niveles.

2.1. Nivel Físico

En el nivel físico se manipulan como elementos de entrada las formas geométricas que se corresponden con las máscaras de difusión utilizadas en el proceso de fabricación de los circuitos integrados del computador. Determinadas disposiciones de estas formas representan dispositivos electrónicos concretos, tales como transistores, resistencias, etc., que son los elementos de salida del nivel físico. En este nivel se suele utilizar como herramienta de estudio software de manipulación gráfica con restricciones.

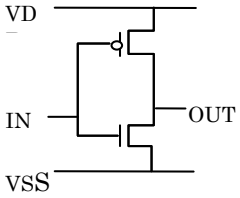


Físico (silicio)

2.2. Nivel Electrónico

En este nivel los elementos de salida, es decir, los biestables y las puertas lógicas, se obtiene a partir de dispositivos electrónicos (resistencias, transistores, etc.) conectados según una determinada topología. Como metodología de estudio (análisis y síntesis) en este nivel se utilizan

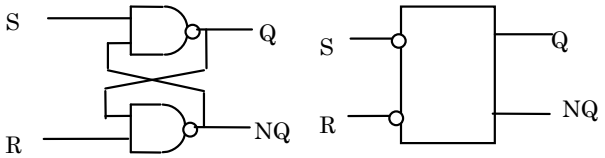
técnicas cuantitativas de análisis en el plano eléctrico-temporal, fundamentalmente ecuaciones algebraicas y diferenciales.



Circuito eléctrico

2.3. Nivel Lógico

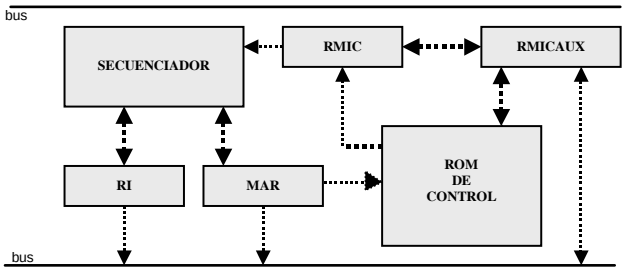
Los elementos de entrada a este nivel son los biestables y las puertas lógicas, y los de salida son módulos combinacionales y secuenciales tales como multiplexores, codificadores, sumadores, registros, contadores, etc. Este nivel de estudio dispone de sus propias técnicas de análisis y síntesis. El comportamiento de un circuito combinacional se representa con una función lógica que admite una expresión algebraica manipulable simbólicamente dentro de un formalismo matemático: el álgebra de conmutación (un álgebra de Boole). Para los circuitos secuenciales se utiliza la teoría de las máquinas de estados finitos.



Circuito lógico

2.4. Transferencia de Registros (RT)

Los elementos de entrada al nivel RT son registros, módulos combinacionales y elementos de interconexión (buses y/o multiplexores). Los primeros mantienen el estado del sistema, los segundos definen las transformaciones elementales del estado, y los terceros permiten el intercambio de información entre los dos primeros. Los elementos de salida son el conjunto de transferencias elementales posibles en la ruta de datos construida con los tres tipos de elementos de entrada. Al contrario de lo que ocurría en los dos niveles anteriores, el nivel RT no dispone de una herramienta propia de análisis y síntesis, aunque sí es posible utilizar los modernos lenguajes de descripción hardware (por ejemplo VHDL) como instrumentos de expresión precisa y estándar a este nivel.



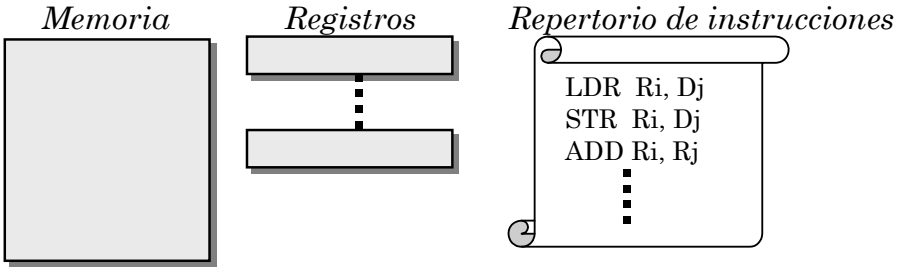
Transferencia de registros

2.5. Arquitectura (lenguaje máquina)

Este es el nivel que separa el hardware del software. Los elementos básicos de entrada son las transferencias y transformaciones posibles de información en la ruta de datos de un computador. Con ellos se construyen las instrucciones máquina y su método de secuenciamiento, es decir, lo que se denomina un *lenguaje máquina*. Lo más significativo de este nivel es que con él comienzan los niveles propiamente simbólicos, es decir, niveles cuyos componentes básicos no son objetos físicos, sino símbolos relacionados por un lenguaje, el instrumento por excelencia del conocimiento simbólico.

El nivel de lenguaje máquina suele ser el primer nivel al que tiene acceso el usuario de un computador, salvo si la máquina es microprogramable, en cuyo caso dispone de un nivel intermedio que permite modificar el repertorio de instrucciones. Normalmente, el usuario no utiliza directamente el lenguaje máquina, sino una representación simbólica del mismo conocida como *lenguaje ensamblador*, que ofrece algunas prestaciones más que el puro lenguaje máquina, como el uso de macros y la definición simbólica de constantes y posiciones de memoria.

Este nivel queda definido por 1) *el repertorio de instrucciones*, con sus formatos, tipos de direccionamiento, modos de secuenciamiento y representación de datos, y 2) *la memoria* y el conjunto de *registros* visibles por el programador. En la actualidad cabe distinguir dos planteamientos diferentes a la hora de definir el nivel máquina de un computador.



En primer lugar está el planteamiento CISC (*Complex Instruction Set Computers*) que define un repertorio de instrucciones bastante complejo y numeroso, con muchos tipos de direccionamiento y muchos modos de control, pretendiendo reducir la distancia semántica que lo separa de los lenguajes de alto nivel, y facilitar así el diseño del compilador. La microprogramación es la técnica de diseño de la unidad de control con la flexibilidad suficiente para facilitar la implementación de las máquinas CISC. Un caso extremo de proximidad al lenguaje de alto nivel lo tenemos en arquitecturas avanzadas que tienen este tipo de lenguaje como su lenguaje máquina.

En segundo lugar tenemos el planteamiento RISC (*Reduced Instruction Set Computer*), que simplifica la complejidad y el número de instrucciones máquina, dejándolo reducido a un conjunto pequeño y rápido que cubre un porcentaje muy elevado del peso computacional de los programas. Los compiladores para este tipo de arquitecturas asumen la responsabilidad de utilizar eficientemente unas instrucciones con poco contenido semántico pero elevada velocidad de ejecución.

Dos aspectos importantes del nivel máquina son también la gestión de memoria y la entrada/salida. Sin embargo, en los computadores actuales estos recursos los utiliza el programa del usuario a través del sistema operativo.

2.6. Software básico (sistema operativo)

El Sistema Operativo (SO) no constituye un nivel del mismo tipo que los demás, por ejemplo, el lenguaje máquina o el lenguaje de alto nivel. En realidad se trata de un gestor de determinados recursos del nivel máquina que por la frecuencia y complejidad de uso resulta más eficiente utilizarlos agrupados en una especie de máquina virtual que es el SO.

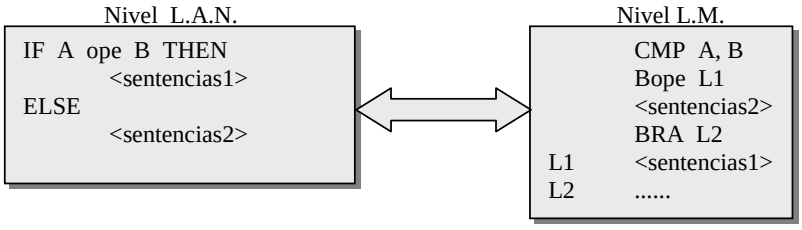
En los primeros computadores las funciones del SO eran escasas, limitadas básicamente a la carga del programa y a la entrada salida. Sin embargo, las competencias de este sistema han ido aumentando con la complejidad y sofisticación de las máquinas modernas, que funcionan en entornos multiusuario y multitarea y que requieren una gestión de todos los recursos de la máquina: la CPU, la jerarquía de memoria, el tratamiento de las excepciones, los mecanismos de protección, etc.

El protagonismo que el SO tiene en un computador actual hace que cuando se diseña la arquitectura de un procesador se tengan muy en cuenta las funciones de este sistema.

2.7. Lenguajes de alto nivel

En este nivel se utilizan lenguajes de programación con una sintaxis y una semántica más complejas que las del lenguaje ensamblador. Estos lenguajes pretenden facilitar el trabajo del programador aportando recursos expresivos más próximos a los problemas que se van a resolver. En este sentido cabe diferenciar dos grandes grupos. Un primer grupo, denominado de *estilo imperativo* como Pascal, Fortran, etc., que presentan una semántica operacional que refleja directamente el funcionamiento de la máquina Von Neumann, es decir, que obligan al programador a expresar la secuencia de órdenes cuya ejecución resuelve el problema en cuestión. El segundo grupo lo formarían los lenguajes de estilo declarativo, como Prolog, Miranda o Lisp (puro), que disponen de un modelo computacional diferente (aunque equivalente) al de la máquina Von Neumann. En este caso el programador tan solo tiene que declarar las relaciones lógicas o funcionales de los elementos que intervienen en la especificación del problema a resolver.

Este nivel requiere un proceso de traducción al nivel máquina que es realizado por un programa denominado *compilador*. Se trata de un programa que toma como dato de entrada un programa fuente escrito en un lenguaje de alto nivel, y produce como salida un programa objeto escrito en lenguaje máquina con una semántica equivalente (igual significado).



2.8. Algoritmos

En el nivel algorítmico se expresa la resolución de un problema mediante un conjunto de reglas aplicadas de forma sistemática y ordenada, es decir, mediante un algoritmo. Los procedimientos que define un algoritmo son independientes de cualquier lenguaje de programación y de cualquier máquina particular.

2.9. Aplicaciones

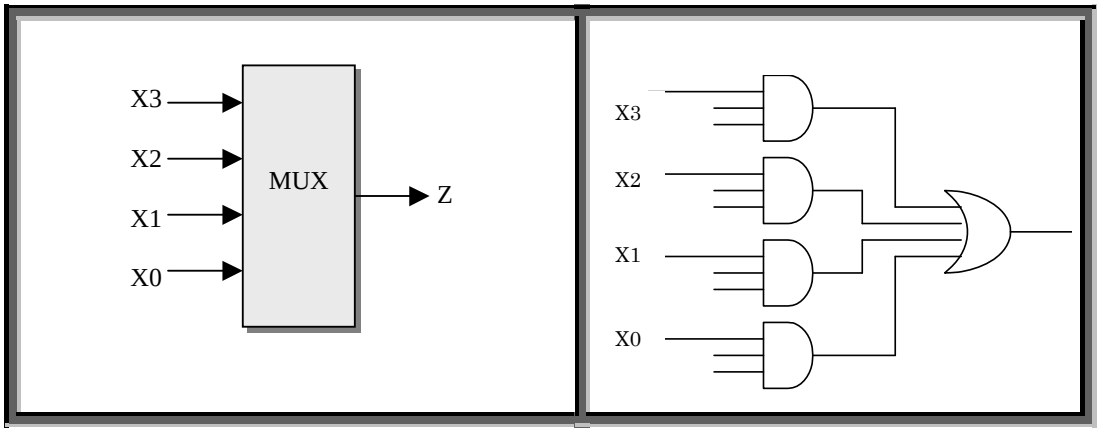
Las aplicaciones se corresponden con dominios de actividad que pueden automatizarse con el uso de un computador digital. Del análisis del dominio se extraen unas especificaciones funcionales que son expresadas mediante algoritmos. Codificados estos algoritmos en un lenguaje de programación y previa compilación, se ejecutan en la máquina.

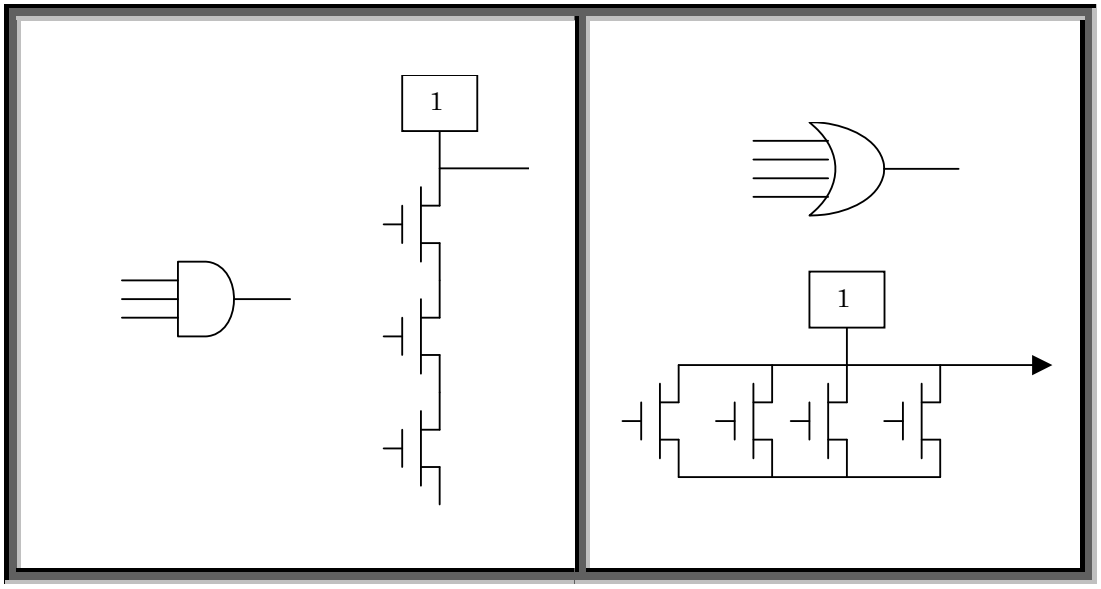
2.10. Inconvenientes de la división entre niveles

El establecimiento de niveles de abstracción en el estudio de un computador hace posible acotar su complejidad al utilizar metodologías de análisis y síntesis propias en cada nivel, permitiendo que dentro de un nivel el usuario pueda abstraerse de lo que ocurre en los demás niveles. Este planteamiento que facilita sin duda el estudio del computador presenta sin embargo

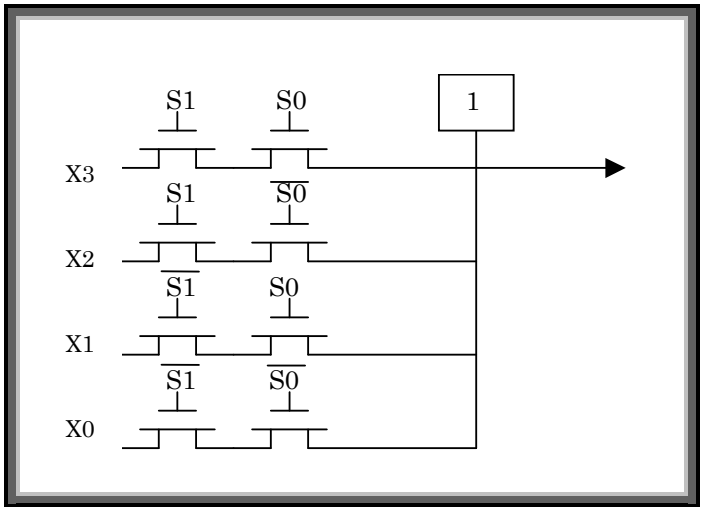
algunas dificultades cuando se contempla el problema de la optimización. En efecto, a la hora de implementar una especificación no sólo deben cumplirse todos los requerimientos funcionales de la misma, además se deben optimizar ciertas funciones de calidad que generalmente tienen que ver con la velocidad (maximización) y el costo (minimización). En efecto, existen ocasiones en las que contemplar tan solo los niveles frontera de un nivel en el que se plantea un problema de diseño puede dar lugar a la imposibilidad de optimizar la implementación. A título de ejemplo citaremos dos casos en los que se presenta esta situación. El primero entre los niveles eléctrico y lógico, y el segundo entre los niveles lenguajes de alto nivel y arquitectura (repertorio de instrucciones)

1) Niveles eléctrico <--> lógico Si nos planteamos el diseño de un multiplexor con conmutadores bidireccionales (tecnología NMOS estática) respetando los niveles de diseño, obtendríamos en primer lugar el esquema lógico (con puertas) del multiplexor, y después expresaríamos cada puerta lógica en términos de los conmutadores bidireccionales.





El diseño resultante es más costoso (mayor número de conmutadores) que el que podemos obtener si planteamos el diseño directamente con conmutadores:



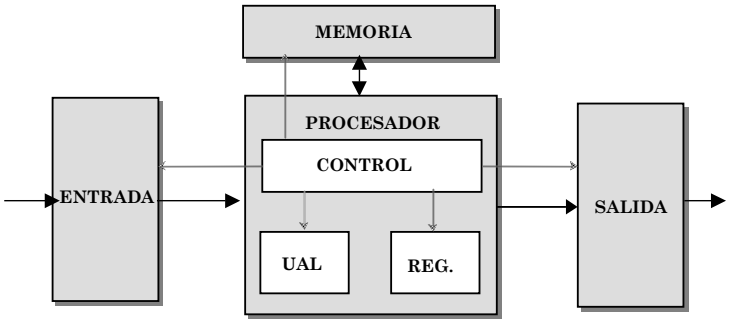
2) Niveles lenguaje de alto nivel <--> arquitectura

En este caso un compilador, para optimizar el código máquina que genera (mayor velocidad), tiene en cuenta no sólo la arquitectura (repertorio de instrucciones) sino la forma en que se ejecutan estas instrucciones (estructura) en la ruta de datos de la máquina. Este hecho puede dar lugar a que el orden de las instrucciones máquina generadas no sea el orden lógico que utilizaría un programador de lenguaje máquina que ignorase los detalles estructurales de la arquitectura. Esta situación viene producida fundamentalmente por la segmentación y paralelización de las instrucciones dentro de la máquina y se estudiará con detalle en las asignaturas de *Ampliación de Estructura de Computadores* y *Arquitectura e Ingeniería de Computadores*.

3. Estructura básica de un computador convencional

3.1. Arquitectura *von Neumann*

La estructura básica de un computador actual sigue siendo la original de von Neumann, una máquina secuencial que ejecuta datos escalares y que hemos representado en la siguiente figura:



La **memoria** almacena las instrucciones del programa, los datos iniciales, los resultados parciales y los finales. Se accede de forma directa (RAM) a cualquier posición para realizar operaciones de lectura o escritura.

El **procesador** es la unidad encargada de leer y ejecutar las instrucciones. Para ello dispone de una *ruta de datos* constituida por un conjunto de registros (REG.), una unidad aritmético-lógica (UAL), y unos buses de comunicación; y una *unidad de control*, que es la encargada de generar las señales que gobiernan todos los dispositivos.

La **entrada** y **salida** constituyen la unidad para la transferencia de información con el mundo exterior. Funcionalmente la máquina tiene las siguientes características:

1) Organización lineal de la memoria

2) Palabra de longitud fija.

3) Espacio único de direcciones.

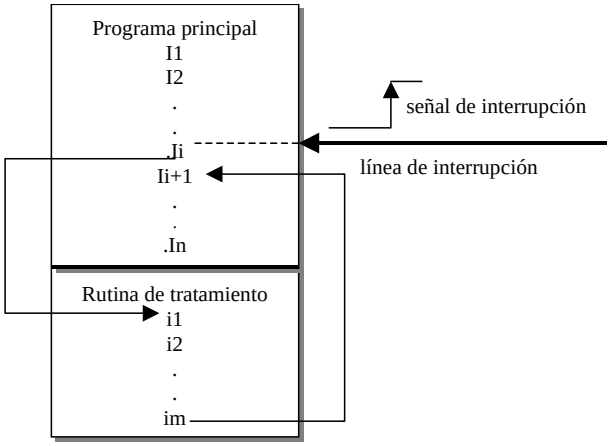
4) Memoria única para datos e instrucciones sin diferenciar entre ambos.

5) Ejecución secuencial de las instrucciones salvo las de ruptura de secuencia

A esta organización básica de von Neumann se han ido incorporando algunas aportaciones significativas entre las que destacaremos las siguientes: Sistema de interrupciones, Sistema de memoria caché y Sistema de memoria virtual.

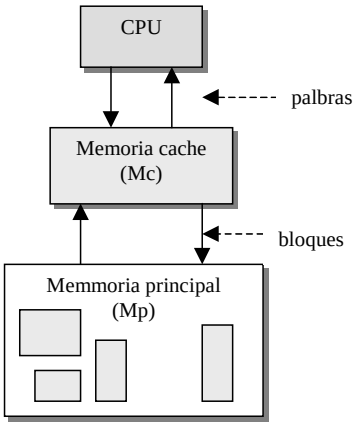
3.2. Sistema de interrupciones

El Sistema de interrupciones permite la interrupción de un programa en ejecución producida por una señal externa a la máquina. El sistema de interrupciones permite una mejor sincronización de la Entrad/Salida con el exterior y la posibilidad de compartir la CPU por más de un programa



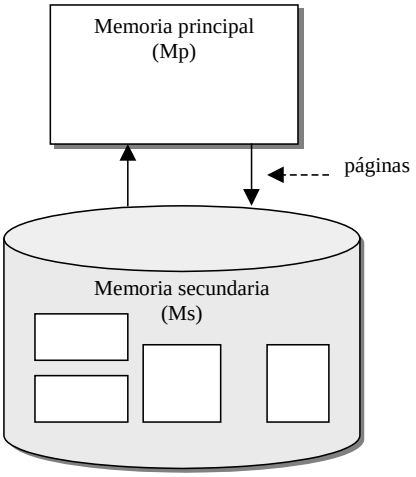
3.3. Sistema de memoria caché

El Sistema de memoria caché permite disminuir el tiempo de acceso a la memoria principal (Mp) ubicando una memoria de menor tamaño y mayor velocidad (memoria caché, Mc) entre la CPU y Mp. El sistema explota la localidad de referencia de los programas haciendo que Mc contenga en cada momento los bloques de Mp más referenciados, y evitando así que la CPU tenga que acceder a Mp:



3.4. Sistema de memoria virtual

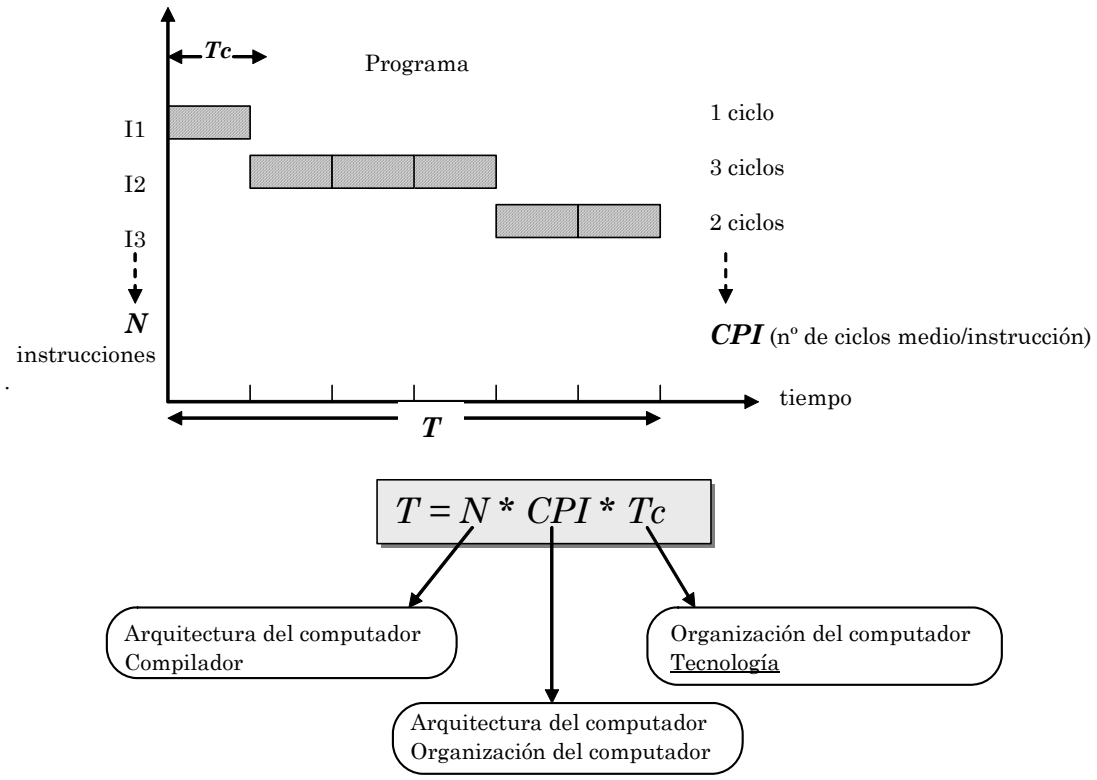
El Sistema de memoria virtual permite la ejecución de programas cuyo tamaño supere el de la Mp. Para ello el sistema mantiene en Mp, de forma transparente para el programador, tan sólo el conjunto de páginas activas (con mayor probabilidad de ser referenciadas) del programa en ejecución. Las restantes páginas que completan el programa residen en la memoria secundaria, hasta que son referenciadas, en cuyo caso el sistema las lleva a Mp:



4. Evolución histórica: tecnología, estructura y arquitectura

La velocidad de procesamiento de información de un computador está determinada básicamente por tres elementos: arquitectura, organización (o estructura) y tecnología.

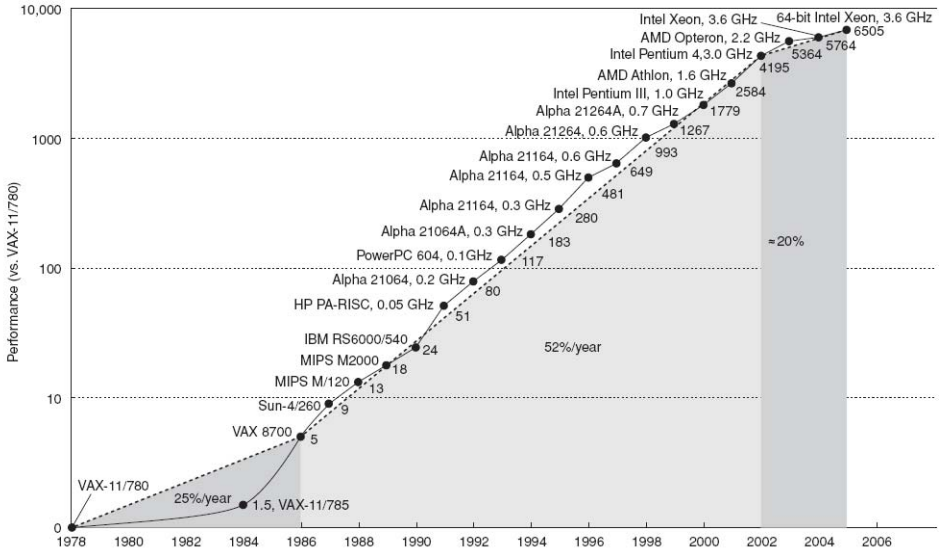
Podemos analizar la influencia de estos elementos en la velocidad de procesamiento de información de un computador teniendo en cuenta que el tiempo T de ejecución de un programa se puede expresar como el producto de tres factores: el número de instrucciones del programa (N), el número medio de ciclos por instrucción (CPI), y el tiempo de ciclo (Tc), como hemos representado en la siguiente figura:



T = tiempo de ejecución del programa
N = número de instrucciones del programa
CPI = número medio de ciclos por instrucción
Tc = tiempo de ciclo

Es decir, mientras la arquitectura influye a través del número medio de ciclos por instrucción y del número total de instrucciones, la organización lo hace a través de éste último y el tiempo de ciclo, mientras que la tecnología lo hace casi exclusivamente a través del tiempo de ciclo (o su inversa, la frecuencia de reloj del procesador). El tiempo de ciclo viene determinado por los tres niveles tecnológicos: físico, electrónico, lógico y transferencia de registros.

La tecnología ha experimentado una transformación continua durante las últimas décadas. Desde la aparición del primer computador comercial, la industria informática ha pasado por cuatro generaciones de computadores, diferenciadas básicamente por la tecnología de los componentes básicos. Los relés y las válvulas de vacío de 1940 a 1950, los diodos y transistores discretos de 1950 a 1960, los circuitos integrados de pequeña y media escala de integración (SSI/MSI) de 1960 a 1970, y los circuitos integrados de alta y muy alta escala de integración (LSI y VLSI) desde 1970 en adelante. La disminución del tiempo de conmutación de los componentes electrónicos ha repercutido directamente en el aumento de velocidad de los computadores. También el aumento de la capacidad de integración y de encapsulado han repercutido en la misma dirección. Los cambios tecnológicos alteran constantemente las relaciones de compromiso tecnología-organización-arquitectura, obligando a la reconsideración de viejas ideas ante un nuevo avance tecnológico. En la siguiente figura se muestra el crecimiento del rendimiento de los procesadores desde 1978 medido en SPECint



4.1. Aumento de rendimiento a través de la organización y arquitectura

A lo largo de las últimas cuatro décadas la organización y arquitectura de computadores ha experimentado un desarrollo gradual, no traumático, que ha ido decantando aquellas características responsables de las mejoras de rendimiento.

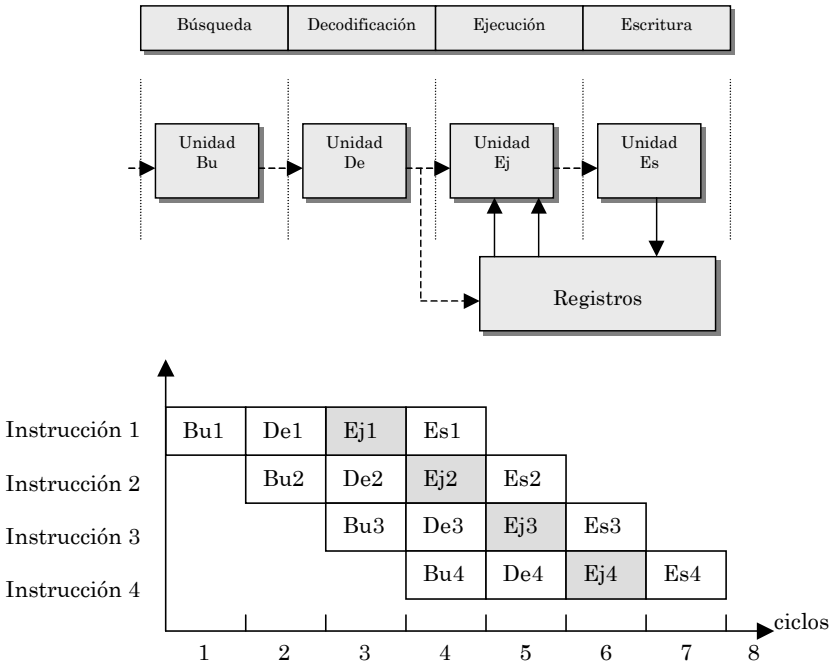
4.1.1. Paralelismo y Segmentación

Las organizaciones y arquitecturas paralelas consiguen que en ciertos instantes de tiempo el computador procese simultáneamente más de una operación básica. La simultaneidad temporal se consigue fundamentalmente con dos técnicas: el paralelismo y la segmentación. La primera ejecuta simultáneamente varias operaciones independientes replicando el número de operadores hardware. La segunda descompone el operador y la operación correspondiente en etapas secuenciales y autónomas, de manera que simultáneamente se puedan ejecutar etapas diferentes de varias operaciones. Ambas técnicas se consideran como dos formas del paralelismo: el paralelismo espacial o replicación la primera, y el paralelismo temporal la segunda.

La idea básica de la segmentación estaba ya latente en la propuesta de von Neumann para construir el primer computador de programa almacenado. Al hablar sobre las técnicas de entrada/salida, sugería la conveniencia de disponer un *buffer* que permitiese el solapamiento de la ejecución del programa con las operaciones de E/S, es decir, una forma primaria de procesamiento segmentado.

4.1.2. Procesadores Segmentados

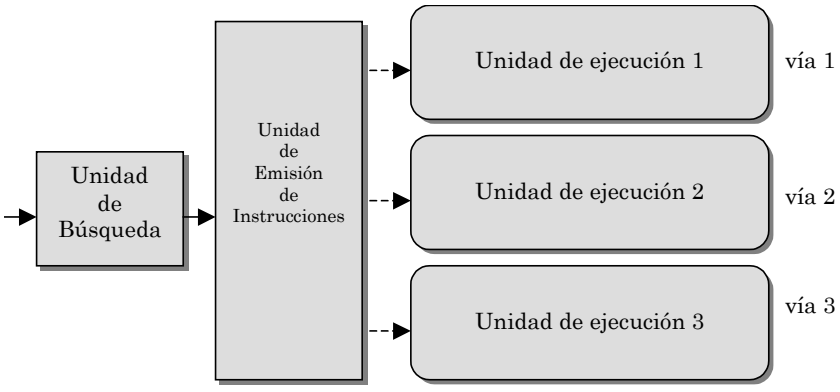
Se trata de arquitecturas monoprocesador que operan con una organización interna en la que se segmenta la ejecución de las instrucciones a fin de iniciar una (y finalizar otra) cada ciclo de operación. Sin embargo este objetivo límite difícilmente llega a conseguirse debido a los riesgos estructurales, las dependencias de datos, las bifurcaciones y las excepciones. Para reducir al máximo el efecto de tales ineficiencias se utilizan técnicas software como la reordenación estática de instrucciones, el renombramiento de registros, y los saltos retardados; y técnicas hardware como el adelantamiento (*forwarding*), la reordenación dinámica de instrucciones y la predicción dinámica de saltos.



4.1.3. Procesadores Superescalares

Un *procesador superescalar* de grado *m* emite *m* instrucciones por ciclo, debiendo ser también *m* el paralelismo a nivel de instrucción para explotarlo completamente. En estos procesadores los recursos para la decodificación y ejecución de instrucciones se incrementan hasta el punto de disponer de *m* cauces segmentados operando concurrentemente, si bien en algunas etapas los cauces pueden compartir algunas unidades funcionales. En general, los conflictos por dependencias de datos, de control y estructurales de los procesadores escalares segmentados siguen existiendo en los superescalares con mayor complejidad.

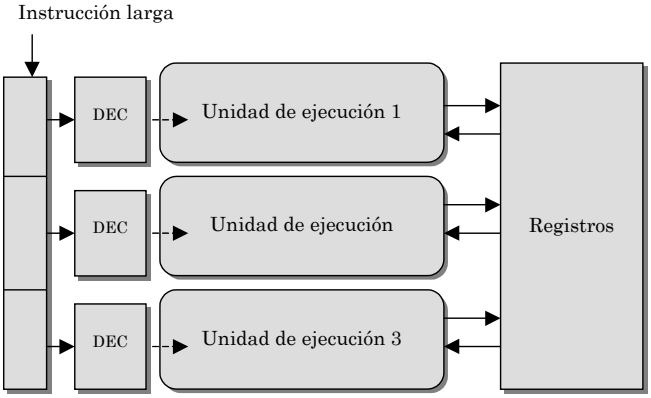
Las máquinas superescalares proporcionan compatibilidad a nivel del código objeto con las máquinas escalares, detectando el paralelismo de las instrucciones en tiempo de ejecución. Normalmente, se dividen las instrucciones máquina en categorías, y como mucho una instrucción de cada categoría puede emitirse simultáneamente.



4.1.4. Procesadores VLIW

En un procesador VLIW (*Very Long Instruction Word*) una única instrucción especifica más de una operación concurrente, reduciéndose el número de instrucciones por programa en

comparación con el caso escalar. Las organizaciones VLIW extendieron y formalizaron el concepto de microcodificación horizontal que se venia utilizando años atrás para el diseño de procesadores de propósito especial dedicados a tareas intensivas en cálculo, tales como el procesamiento de señales digitales. Utilizando técnicas avanzadas de compilación se puede extraer el paralelismo de grano fino de un amplio rango de aplicaciones científicas y de propósito general. En estos procesadores los conflictos por dependencias de datos y estructurales se resuelven antes de la ejecución, y son explícitamente controlados por las instrucciones. El hardware adicional se dedica a caminos de datos paralelos y más unidades funcionales, en lugar de a lógica de control y sincronización.

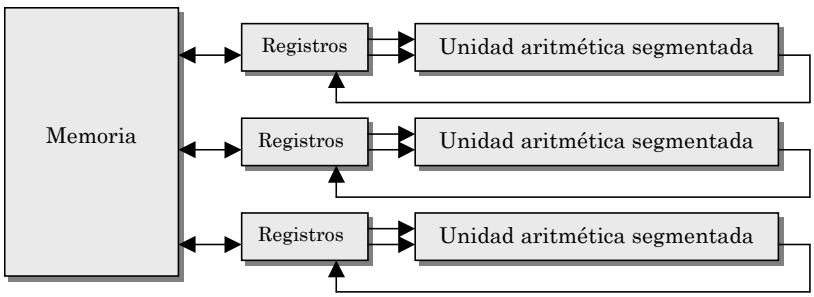


4.1.5. Procesadores Vectoriales

Los procesadores vectoriales disponen de operaciones que trabajan sobre vectores de números. Por ejemplo, una operación vectorial puede sumar dos vectores de 64 elementos en punto flotante dando como resultado otro vector de 64 elementos. La instrucción vectorial es equivalente a un bucle completo en el que se calcula un elemento del resultado en cada iteración, se actualiza el índice y se bifurca al comienzo.

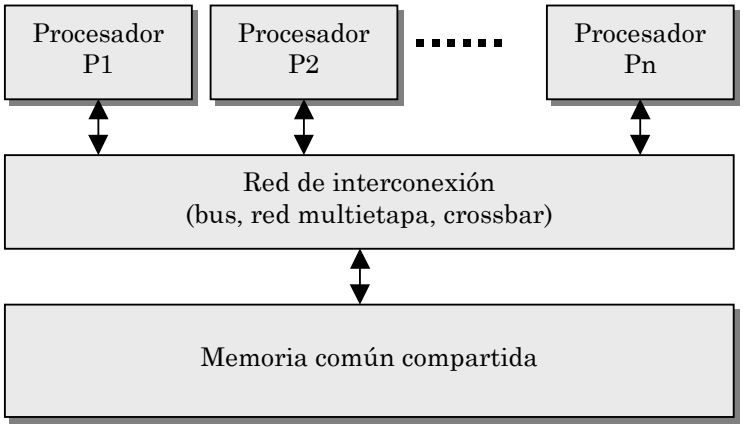
Una simple instrucción vectorial especifica, pues, una gran cantidad de trabajo, por lo que se reduce la anchura de banda necesaria para su lectura en comparación con la de las instrucciones escalares equivalentes. Además, el acceso a los datos tiene un patrón conocido.

Desde el punto de vista arquitectónico son procesadores segmentados con instrucciones máquina vectoriales. Al no existir dependencias entre operaciones de una instrucción vectorial, se explota eficientemente la segmentación en las unidades aritméticas. Pero para conseguir el rendimiento máximo de estas arquitecturas hay que alimentar a las unidades funcionales segmentadas con nuevos datos en cada ciclo de reloj, lo que requiere un gran ancho de banda con la memoria principal.



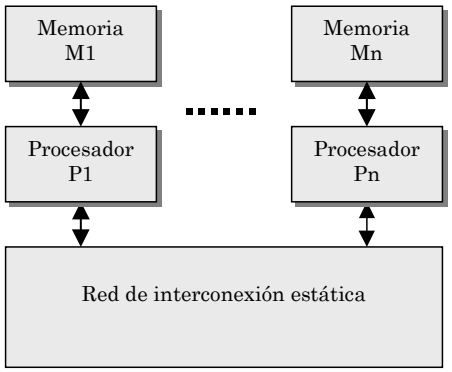
4.1.6. Multiprocesadores de Memoria Compartida

Se trata de arquitecturas compuestas por un conjunto de procesadores que acceden a una única memoria común a través de una red de interconexión. Utilizan memorias caché locales para las que hay que resolver el problema de su coherencia con respecto a la memoria principal y entre sí. Las soluciones que se adoptan tienen un mayor o menor soporte hardware y dependen mucho de la red de interconexión. Otro problema que plantean estas arquitecturas es el de la sincronización de los diferentes procesadores cuando participan de una tarea común. Para ello es necesario disponer del soporte hardware en forma de instrucciones máquina del tipo TEST&SET, TEST&AND, etc. que permiten implementar secciones críticas y otros mecanismos de sincronización a nivel del sistema operativo.



4.1.7. Multicomputadores

Son multiprocesadores de memoria distribuida donde cada procesador tiene un espacio privado de direcciones. Se comunican y sincronizan mediante paso de mensajes a través de una red de interconexión. Las topologías de red más utilizadas son la malla y el hipercubo.



5. Lenguajes de descripción hardware.

Los lenguajes de descripción hardware son lenguajes de alto nivel con una sintaxis similar a los de programación (C, ADA, Pascal, Modula, etc.) y una semántica que permite el modelado y simulación de los dispositivos hardware a diferentes niveles de abstracción. Los primeros lenguajes de este tipo sólo pretendían servir de vehículo de comunicación del diseño. Se trataba de formalismos de especificación de los dispositivos hardware desarrollados por instituciones universitarias o por industrias electrónicas que alcanzaron escasa difusión. Pero los actuales lenguajes han adquirido un alto grado de estandarización y han adoptado los nuevos conceptos de la ingeniería software, permitiendo la verificación de la especificación del diseño mediante

simulación. Se utilizan como vehículo de entrada a muchas herramientas de diseño automático. Revisaremos brevemente algunos de estos lenguajes

5.1. Lenguajes precursores

5.1.1. CDL (Computer Design Language)

Fue desarrollado por Yaohan Chu a comienzo de los años 60 bajo el principio de separación de la componente lógica y electrónica de un computador digital. CDL refleja directamente el hardware y sus operaciones, es decir, existe una correspondencia uno-a-uno entre los objetos y operaciones hardware (registros, RAMs, relojes, suma, cuenta, etc.) y las construcciones del lenguaje. La primera versión del simulador CDL para IBM 7090 estuvo disponible en 1968, y la versión tercera para Univac pocos años más tarde. Se utilizó en universidades y en la industria del radar y aeronáutica.

5.1.2. DDL (Digital systems Design Language)

Se desarrolló a mediados de los 60 en la Universidad de Wisconsin con varios objetivos: precisión y concisión para facilitar la especificación de los diseños, potencia suficiente para modelar sistemas complejos, independencia respecto a cualquier tecnología o procedimiento de diseño, capacidad de especificación a diferentes niveles de abstracción y, finalmente, una sintaxis y una semántica que permitieran la documentación jerárquica del diseño.

5.1.3. AHPL (A Hardware Programming Language)

Fue propuesto por F.J. Hill y G.R. Peterson unos meses más tarde que el CDL y DDL, y apareció publicado por primera vez en 1973 en la edición original de *Digital Systems: Hardware Organization and Design*. Los autores concibieron AHPL como un lenguaje de síntesis: todo dispositivo síncrono que pudiese ser implementado en hardware debía ser expresable en AHPL de manera tal que se pudiese traducir a una realización física siguiendo un conjunto simple de reglas. En opinión de uno de sus autores, F.J. Hill, que participó como miembro del grupo de trabajo que formuló las especificaciones originales para VHDL, la existencia de AHPL favoreció la incorporación de mecanismos para permitir el proceso de síntesis en VHDL.

5.1.4. ISPS (Instruction Set Processor Specifications)

Con este lenguaje se dio un paso importante hacia la formalización del proceso de diseño a niveles de comportamiento. Además de la simulación y la síntesis, ISPS se utilizó en la generación de software, la verificación de programas y la evaluación de arquitecturas. ISPS favoreció los aspectos de comportamiento sobre los estructurales pero sin eliminarlos completamente.

5.1.5. TI-HDL (Texas Instruments-HDL)

Es un lenguaje de descripción jerárquica del diseño, estructurado en bloques y basado en texto, que se ha utilizado principalmente en el diseño de circuitos integrados. Procede de antiguos lenguajes usados allá por 1968, concretamente el TIBSD (*TI Boolean System Description*), desarrollado como lenguaje de entrada de datos para un sistema CAD de circuitos impresos, y Fusim (*Functional Simulator*), utilizado para describir modelos de alto nivel de microprocesadores y para generar prototipos de patrones de tests.

5.2. Lenguajes actuales

5.2.1. Verilog

Es un lenguaje de descripción hardware diseñado por la compañía Cadence Design Systems Inc., que se ha venido utilizando como lenguaje del simulador digital Cadence. El uso de Verilog está promovido por la Open Verilog International (OVI), que publicó en octubre del 91 la primera versión del Hardware Description Language Reference Manual. En Verilog la unidad de diseño fundamental es el módulo, que describe un componente hardware con su interfaz y contenido. Desde un punto de vista funcional, un módulo Verilog contiene la información de una entidad y su correspondiente arquitectura VHDL. Verilog no proporciona compilación independiente de

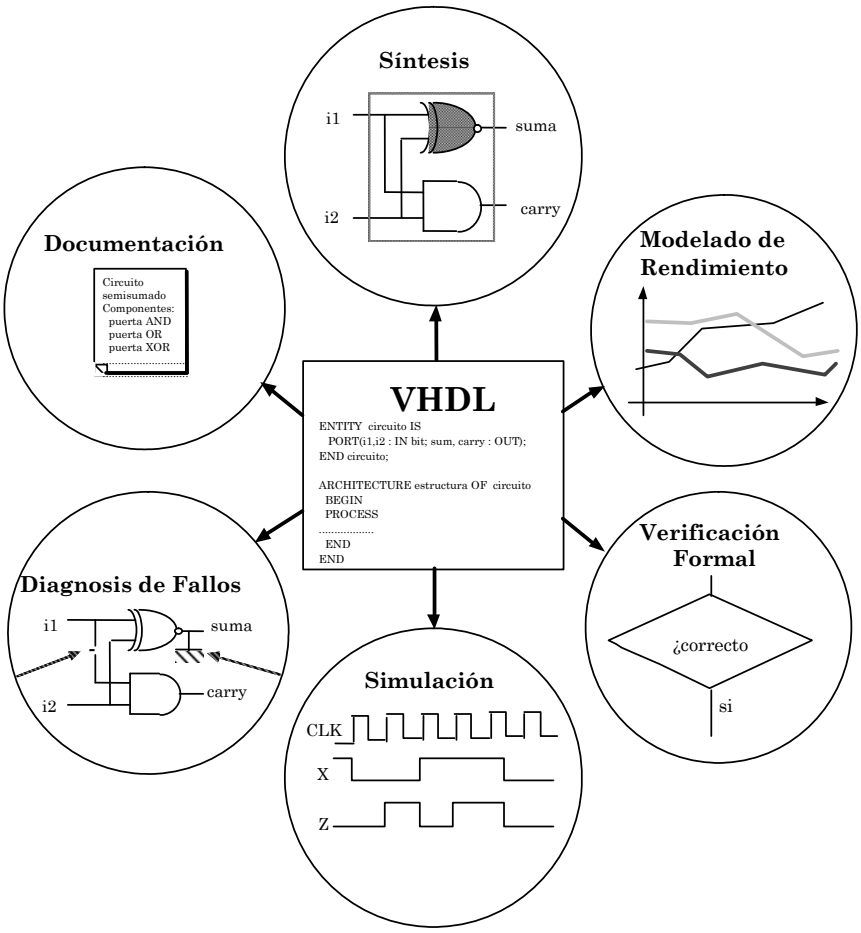
módulos: todos los módulos relacionados con el mismo diseño y simulación deben estar en el mismo archivo.

5.2.2. UDL/I (Unified Design Language for Integrated circuits)

Es un lenguaje de descripción hardware que se viene desarrollando desde 1989 por la Japan Electronic Industry Development Association, dependiente de importantes compañías japonesas tales como NTT. Una de las características de UDL/I es que pretende ser específico para modelar circuitos integrados. La única unidad de diseño existente en UDL/I es la descripción de diseño, que comprende varias sub-entidades denominadas descripciones de módulos. Una descripción de diseño contiene el modelo de un circuito integrado que consta de varias subunidades o módulos, cada uno de los cuales está especificado por una descripción de módulo.

5.2.3. VHDL (VHSIC Hardware Description Language)

Es un lenguaje impulsado por el Departamento de Defensa de los Estados Unidos dentro del programa VHSIC (*Very High Speed Integrated Circuits*) y estandarizado por IEEE Computer Society. Con VHDL se puede estudiar un sistema digital a diferentes niveles de abstracción dentro de un único lenguaje de programación, acelerando considerablemente las diferentes fases diseño y proporcionando un mejor conocimiento del mismo cuando se aborda la fase de implementación física. VHDL es un lenguaje con una semántica orientada a la *simulación*. Por ello, su principal dominio de aplicación es el modelado de dispositivos hardware para comprobar su corrección funcional. Sin embargo, como ilustra la siguiente figura sus áreas de aplicación son cada vez más numerosas, y hoy día se utiliza en la *síntesis automática*, la *diagnosis de fallos,* la *verificación formal,* el *modelado de rendimiento* y la *documentación*



La ***síntesis automática*** tiene como objetivo la generación de dispositivos digitales a partir de una especificación inicial expresada en un lenguaje de descripción hardware. VHDL, al ser un lenguaje con una semántica poco formal, en el que se pueden mezclar diferentes estilos de descripción, no es el más apropiado para utilizar como entrada en las herramientas de síntesis. Sin embargo, dado su *status* de estándar IEEE y su amplia aceptación en la industria electrónica, resulta muy conveniente no romper el ciclo de diseño saliendo de VHDL en la fase de síntesis. Por ello, lo que se realiza habitualmente en este terreno es definir subconjuntos sintetizables de VHDL. En este sentido cabe distinguir dos grandes subconjuntos. Uno que engloba las construcciones secuenciales de VHDL, utilizado para la *síntesis de alto nivel*, y otro que parte de unas construcciones concurrentes de VHDL que permiten descripciones de dispositivos a nivel de transferencia de registros (RTL), utilizado para la *síntesis de bajo nivel.*

La ***diagnosis de fallos*** en circuitos digitales se puede plantear a partir de las descripciones VHDL de los mismos. En efecto, en lugar de identificar la parte de circuito que falla, podemos localizar la parte de descripción VHDL cuya implementación hardware tiene un fallo físico y está causando el fallo de comportamiento observado. En principio esta diagnosis es similar a la que se realiza a nivel de puertas, pero en este caso el nivel de descripción hardware puede ser más alto y consecuentemente el modelo de fallos será diferente.

La ***verificación formal*** de un diseño consiste en probar que para todos los estados iniciales aceptables y para todas las entradas posibles, la implementación del diseño cumple su especificación. Las otras dos alternativas para verificar un diseño, la síntesis automática y la simulación funcional, pueden resultar incompletas. En efecto, los diseños producidos por síntesis automática son correctos por construcción si los componentes primitivos están completamente verificados y si las transformaciones realizadas en el proceso de síntesis son así mismo correctas. Sin embargo, probar la corrección formal de un método de síntesis implica probar la corrección formal del método en sí y del software que lo implementa, tarea que puede resultar impracticable. Por otra parte, la verificación funcional por simulación exhaustiva también es impracticable a partir de una complejidad media en el diseño. La verificación formal requiere un modelo matemático para representar las propiedades bajo estudio y un cálculo para realizar computación simbólica sobre el modelo. La lógica es la rama de las matemáticas más ampliamente utilizada en la verificación formal, incluyendo la lógica de predicados de primer orden, la lógica de orden superior y la lógica temporal. Al no existir una semántica formal para VHDL los sistemas de verificación formal, al igual que los sistemas de síntesis, se limitan a subconjuntos del lenguaje.

Los ***modelos de rendimiento*** constituyen el nivel más alto de abstracción de los sistemas electrónicos. Estos modelos se utilizan principalmente para estudiar la capacidad global de procesamiento de información de un sistema en las primeras etapas del proceso de diseño. El objetivo es identificar las principales unidades funcionales y definir su forma de actuación en la transformación de los datos de entrada en datos de salida. A este nivel de conceptualización el diseñador ve una unidad funcional como algo que realiza una cierta tarea en un cierto tiempo, sin detalles específicos sobre la forma de realizar la tarea. En realidad, puede que estos detalles no se conozcan aún en esta fase del diseño. Debido a esta ocultación de detalles sobre los valores de los datos y sus transformaciones específicas se dice que los datos son no interpretados, y para resaltar este hecho a los modelos de rendimiento se les denomina a veces *modelos no interpretados*. VHDL, al disponer de recursos expresivos con alto nivel de abstracción, facilita la confección de modelos de rendimiento basados en redes de Petri extendidas y colas estocásticas.

Finalmente**, la *documentación*** es una de las tareas más importantes en todo proceso de diseño, y muy particularmente cuando el sistema tiene el grado de complejidad de los circuitos actuales. En estos casos se hace indispensable disponer de lenguajes con una elevada capacidad de abstracción y ampliamente aceptados por los diseñadores, como ocurre en la actualidad con VHDL.

Como muestra la siguiente figura, VHDL dispone de recursos expresivos para cubrir totalmente la descripción y el modelado de dispositivos digitales en los niveles circuito lógico,

transferencia de registros y chip. Los niveles de sistema y circuito eléctrico sólo se cubren parcialmente. Pero lo más interesante del lenguaje en este aspecto es que permite mezclar en una misma descripción diferentes niveles. Esto, unido al modelo de concurrencia que permite la construcción de modelos estructurales en los que los componentes básicos pueden a su vez ser descritos con modelos estructurales, hace posible el establecimiento de una descomposición estructural de la jerarquía de diseño.