# 1

# CONCEPTOS FUNDAMENTALES

## 1.1. Origen de los computadores

La necesidad de procedimientos de cálculo automático surge de las **limitaciones** que tiene el cálculo manual cuando el volumen de cálculos es apreciable, estas limitaciones son:

- La **velocidad de cálculo de un operador humano es limitada y muy reducida**. Una suma o una multiplicación puede llevar un tiempo del orden de varios segundos o minutos. Hay ciertos problemas que requieren miles de estas operaciones; estos problemas no pueden ser resueltos por un operador humano en un tiempo razonable.
- El operador humano es muy propenso a cometer **errores** en operaciones complejas, esto hace que tengan que tomarse ciertas medidas para evitarlos, lo que supone una pérdida de tiempo. Si el cálculo se realiza con algún tipo de máquina, estos errores pueden evitarse, ya que la máquina no estará afectada por las causas que, ordinariamente, provocan los errores humanos tales como distracciones, fatiga, etc.

Estas causas han hecho que el hombre, desde la antigüedad, haya tratado de buscar formas de automatizar los cálculos. Precisamente la palabra **Informática** se origina de la fusión de las palabras **Información** y **Automática**, y la podríamos definir como *el conjunto de técnicas tendentes a facilitar el tratamiento automático de la información.*

## 1.2. Breve reseña histórica

En los párrafos siguientes veremos, de manera breve, las etapas por las que ha pasado la evolución de las máquinas automáticas de calculo. Para comenzar, podemos mencionar que la palabra **cálculo** proviene del latín *cálculus-i* que significa piedra. Esto es así porque, en la antigüedad, las operaciones aritméticas se realizaban cambiando de posición pequeños guijarros colocados sobre el suelo.

### 1.2.1. La era mecánica de los computadores

Podríamos decir que las máquinas mecánicas de calcular constituyen la "era arcaica"o generación 0 de los computadores. Una evolución de estas máquinas son las registradoras mecánicas que aún existen en la actualidad. Otro elemento de cálculo mecánico que se utilizó hasta hace pocos años fue la **regla de cálculo**, que se basa en el cálculo logarítmico, y cuyo origen son los círculos de proporción de *Neper*. Ingenios clásicos de esa etapa fueron la máquina de *Pascal*, que podía realizar sumar, restas y, posteriormente, multiplicaciones y divisiones, y las dos máquinas de *Charles Babbage*: la máquina de diferencias y la **máquina analítica**. Esta última fue la precursora de los computadores actuales.

La fase final de la era mecánica de la Informática la constituyen los computadores electromecánicos, basados en lógica de relés, desarrollados principalmente en Alemania en la década de los 30.

### 1.2.2. La era electrónica de los computadores

Los computadores basados en elementos mecánicos plantean ciertos problemas:

- La **velocidad de trabajo está limitada** por la inercia de las partes móviles.
- La **transmisión de la información por medios mecánicos** (engranajes, palancas, etc.) **es poco fiable** y difícilmente manejable.

Los computadores electrónicos salvan estos inconvenientes ya que carecen de partes móviles y la velocidad de transmisión de la información por métodos eléctricos no es comparable a la de ningún elemento mecánico.

El primer elemento electrónico usado para calcular fue la válvula de vacío y, probablemente, el primer computador electrónico de uso general fue el E.N.I.A.C. (*Electronic Numerical Integrator and Calculator*) construido en la Universidad de Pennsylvania (1943-46). El primer computador de **programa almacenado** fue el E.D.V.A.C. (*Electronic Discrete Variable Computer*, 1945-51) basado en la idea de **John Von Neumann**, que también participó en el proyecto E.N.I.A.C. de que el programa debe almacenarse en la misma memoria que los datos.

### 1.2.3. Generaciones de ordenadores

En la evolución de las máquinas para el tratamiento automático de la información pueden distinguirse una serie de hitos que marcan la diferencia entre las denominadas **generaciones de ordenadores**. Las generaciones habidas hasta la actualidad han sido:

**1ª generación**: (1946-1955) Computadores basados en **válvulas de vacío** que se programaban en **lenguaje máquina** o en lenguaje ensamblador.

**2ª generación**: (1953-1964) Computadores de **transistores**. Evolucionan los **modos de direccionamiento** y surgen los **lenguajes de alto nivel**.

**3ª generación**: (1964-1974) Computadores basados en **circuitos integrados** y con la posibilidad de trabajar en **tiempo compartido**.

**4ª generación**: (1974-) Computadores que integran toda la CPU en un solo circuito integrado (**microprocesadores**). Comienzan a proliferar las **redes de computadores**.

### 1.2.4. Tendencias actuales

Las tendencias actuales de la Informática, en cuanto a las arquitecturas, tienden a aumentar la velocidad de procesamiento de los computadores por dos caminos:

**Computadores con conjunto reducido de instrucciones** (RISC, *reduced instruction set computer*): en estadísticas sobre el uso de las instrucciones en computadores actuales se ha visto que un porcentaje muy alto de instrucciones se usan muy poco. Ello lleva como efecto que un conjunto de instrucciones muy amplio puede retardar innecesariamente las instrucciones que más se usan; esto se debe a que un juego de instrucciones amplio requiere el uso de control microprogramado, mientras que la tecnología actual permite, para conjuntos de instrucciones reducidos, el uso de unidades de control cableadas más rápidas. Por otra parte, las instrucciones más simples son más fáciles de ejecutar en cadena, con la técnica de **segmentación** que se describe a continuación.

**Proceso paralelo**: El proceso paralelo trata de hacer funcionar a la vez a varias unidades de proceso. Existen muchas formas de proceso paralelo siendo una de las más frecuentes la **segmentación** que consiste en procesar a la vez varias instrucciones aunque cada una de ellas en diferente fase, de forma similar a una fabricación en cadena. Otras formas de proceso paralelo requieren múltiples procesadores. En este caso, para resolver cierto tipo de problemas, será necesario dividir el problema en partes para resolver cada una de ellas en un procesador. También es posible repartir los diversos procesos en diferentes procesadores lo que se conoce como **multiproceso simétrico**, que es mucho más sencillo que lo anterior ya que no hay que dividir el problema en partes. Otra posibilidad es el **procesamiento multihilo** (*multithread processing*) en que un mismo proceso se desarrolla en múltiples hilos que pueden ser tratados por diferentes procesadores.

Por otra parte, la Electrónica está evolucionando de forma paralela para conseguir dispositivos electrónicos cada vez más rápidos. En esta línea existen procesadores que funcionan a varios Gigaciclos.

## 1.3. Requisitos mínimos para la computación

Para saber cuáles son las necesidades mínimas para un proceso automático de cálculo analizaremos el proceso seguido por un proceso de cálculo manual. El proceso, si es algo complicado, requerirá de un papel que realizará las funciones de **almacenamiento**. La información contenida en ese papel puede ser de dos tipos:

**Instrucciones**, es decir, la secuencia de operaciones que hay que efectuar para resolver el problema (receta).

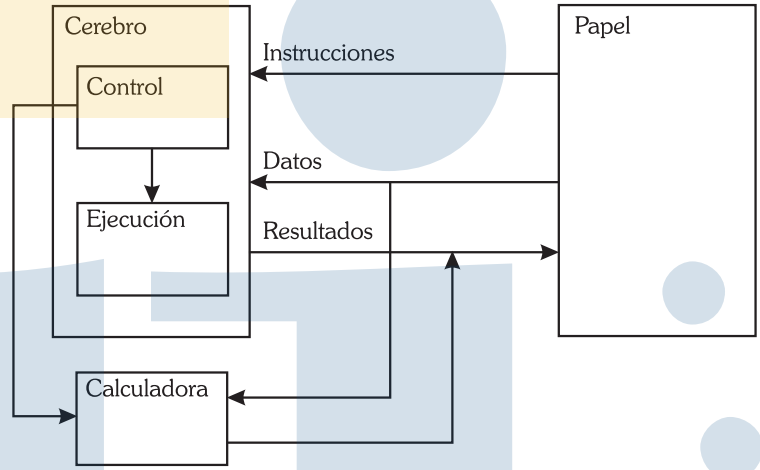


**Fig. 1.1**. Proceso de cálculo manual.

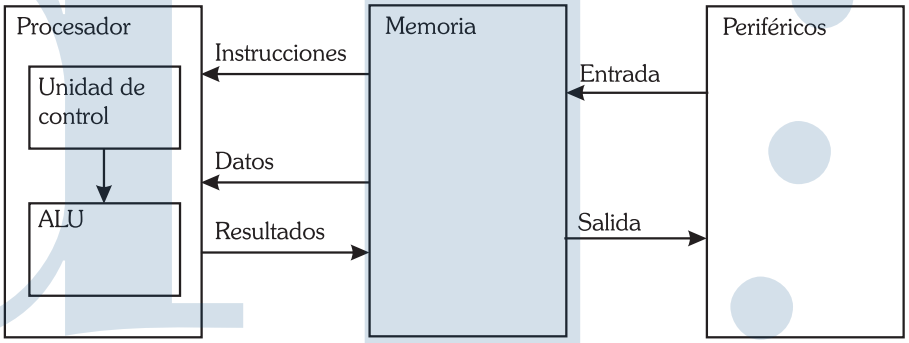


**Fig. 1.2**. Proceso de cálculo automático.

**Datos**: estos datos pueden ser los datos de partida para resolver el problema o también resultados de los cálculos intermedios.

El papel también servirá para escribir el resultado y comunicarlo a otras personas.

Todo este proceso (figura 1.1) tiene lugar en el cerebro que, por un lado, **controla todo el proceso** y, por otro, **ejecuta las operaciones** propias del cálculo. Esta última función puede realizarse mediante una calculadora, esto es así debido a que las funciones de ejecución son puramente mecánicas.

Los componentes de un proceso de cálculo automático son similares al proceso anterior (figura 1.2). La **memoria** corresponde al papel utilizado por el operador humano. La unidad de control hace una de las funciones del cerebro: **controlar el proceso**, es decir, **interpretar las instrucciones**, y la **unidad aritmética y lógica** (**ALU:** ***arithmetic and logic unit***) realiza el otro: **ejecutar las operaciones**. Por otra parte, también será necesaria una forma de **comunicación con el exterior** que intercambie información con los usuarios, tanto los datos de entrada como los resultados: esta es la función de los **equipos de entrada y salida o periféricos** (en el ejemplo anterior, esta función también la realizaba el papel). La función de comunicación es imprescindible y sin ella el computador no tendría ninguna utilidad, ya que nadie podría conocer

los resultados de los cálculos que la máquina hiciera. Por otro lado, también serán necesarios unos medios de comunicación entre las diferentes partes anteriores.

Como consecuencia de lo anterior, podemos decir que cualquier calculador, humano o artificial, necesita, como mínimo, los siguientes componentes:

- Un **procesador** capaz de interpretar y ejecutar instrucciones.
- Una **memoria** para almacenar instrucciones y datos.
- Unos medios de **comunicación** con el exterior.
- Unos medios de **intercomunicación** entre los elementos anteriores

## 1.4. Máquinas de Turing

Podemos interpretar un **proceso de computación** como la *evaluación de alguna aplicación* $f(X)$*, donde* $X$ *es un dato de entrada y* $Z = f(X)$ *es el resultado del proceso*. En este intento de definición, $X$, $Z$ y $f$ pueden tener las interpretaciones más variadas: $X$ y $Z$ pueden tener la naturaleza más diversa: pueden ser números, palabras, ficheros, etc. y la función $f$ puede ser cualquier tipo de proceso: un cálculo numérico, la actualización de un fichero, etc.

Para evaluar la función $f(X)$ en una máquina concreta, tenemos que expresar la función $f$ como una composición de $n$ aplicaciones

$$f = f_n \circ f_{n-1} \circ \ldots \circ f_2 \circ f_1$$

de tal forma que las funciones $f_i$ pueden ser especificadas mediante el conjunto de instrucciones de la máquina, es decir, el conjunto de operaciones elementales que la máquina puede realizar. La secuencia de funciones

$$f_1, f_2, \ldots, f_{n-1}, f_n$$

puede interpretarse como el **programa** para evaluar la función $f(X)$ y la secuencia de operaciones elementales

$$Y_1 = f_1(X), Y_2 = f_2(Y_1), ..., Y_{n-1} = f_{n-1}(Y_{n-2}), Z = f_n(Y_{n-1})$$

puede tomarse como la **definición formal del proceso de computación** $f(X)$.

La función $f$ se podrá descomponer en formas distintas según el nivel de complejidad de las funciones elementales $f_i$. Esto nos da una idea de que un mismo proceso de computación puede ser especificado en diferentes niveles de programación que corresponden a estas diferentes descomposiciones de la función $f$ (**niveles del computador**). Si un proceso $f$ se descompone en un grupo de funciones $f_i$ con un número de funciones elementales, $n$, diremos que la especificación es de **bajo nivel** si el número de funciones $n$ es elevado y diremos que es de **alto nivel** si $n$ es reducido.

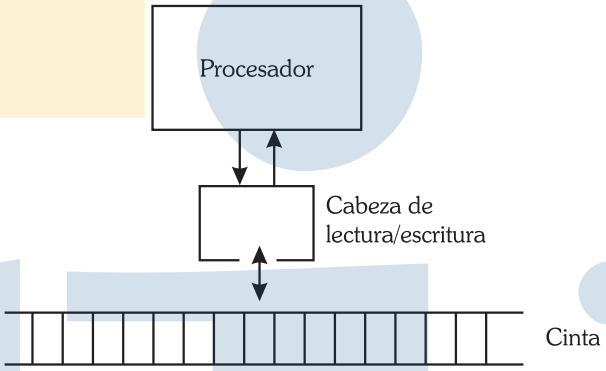


**Fig. 1.3**. Máquina de Turing.

Una cuestión que nos podemos plantear es: ¿Qué cálculos, o procesos de computación en el sentido de la definición anterior, no es razonable que una máquina pueda realizar? Un modelo mental para dar respuesta a este tipo de preguntas fue propuesto por Alan Turing (1936). Este modelo se llama **máquina de Turing** y en la figura 1.3 se muestran sus componentes:

**La memoria** es una cinta sin límites dividida en celdas. Cada una de las celdas puede contener un símbolo de un conjunto finito de ellos, a este conjunto se le denomina **alfabeto de la máquina**.

**El procesador** es una máquina digital con un número finito de configuraciones internas o **estados**.

**La cabeza de lectura/escritura**, que es capaz de leer el contenido de una de las celdas de la cinta, cambiarlo y mover la cinta a ambos lados en una posición a partir de la actual; resumiendo, cuatro operaciones: leer, escribir, mover a la izquierda y mover a la derecha.

La máquina tiene diferentes instrucciones especificadas en el siguiente formato:

$$s_h \quad t_i \quad o_j \quad s_k$$

Esto significa que, si el procesador se encuentra en el estado $s_h$ y la cabeza lectora lee de la cinta el símbolo $t_i$, el procesador ejecuta una operación $o_j$ y cambia el estado del procesador a $s_k$. La operación $o_j$ puede ser una de las siguientes:

- $o_j = t_j \quad \rightarrow$ Esta operación sustituye el símbolo leído $t_i$ por el $t_j$.
- $o_j = \mathrm{R} \quad \rightarrow$ Mueve la cinta una posición a la derecha.
- $o_j = \mathrm{L} \quad \rightarrow$ Mueve la cinta una posición a la izquierda.
- $o_j = \mathrm{H} \quad \rightarrow$ Detiene el proceso.

La determinación de $o_j$ y $s_k$ en función de $s_h$ y $t_i$ se obtiene a partir de la denominada matriz funcional de la máquina. Un esquema el funcionamiento de la máquina de Turing puede verse en la figura 1.4.

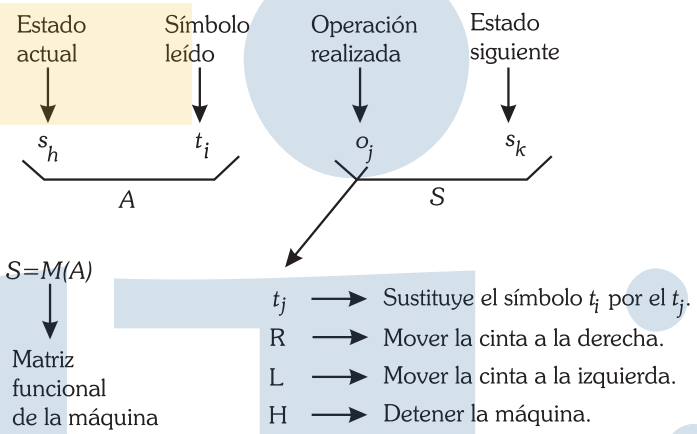


**Fig. 1.4**. Funcionamiento de la máquina de Turing.

| Estado actual | Símbolo leído | Operación | Estado siguiente | Comentarios |
|---|---|---|---|---|
| $s_0$ | b | R | $s_0$ | Buscando el primer 1 |
| $s_0$ | 1 | R | $s_1$ | 1 encontrado |
| $s_1$ | 1 | R | $s_1$ | Buscando blanco de separación |
| $s_1$ | b | 1 | $s_2$ | Blanco encontrado, se cambia por 1 |
| $s_2$ | 1 | L | $s_2$ | Buscando el blanco de la izquierda |
| $s_2$ | b | R | $s_3$ | Hallado blanco ⇒ cambiar el siguiente 1 |
| $s_3$ | 1 | b | $s_4$ | Se cambia el primer 1 por blanco |
| $s_4$ | b | H | $s_0$ | Fin del proceso |

**Fig. 1.5**. Programa de una máquina de Turing para sumar dos números.

Para efectuar un cálculo $Z = f(X)$, la máquina de Turing opera como sigue: el **dato de entrada** $X$ **tiene que codificarse** de forma que pueda almacenarse en la cinta, entonces se **arranca el proceso** causando la secuencia de operaciones $f_1, \ldots, f_n$, cuando concluya la $n$-sima operación, la máquina **habrá escrito en la cinta la codificación del resultado** $Z = f(X)$.

Las máquinas de Turing constituyen un ejemplo de los denominados **autómatas**.

**Ejemplo 1.1:** ***Máquina de Turing para sumar dos números***

*Siguiendo el esquema que se acaba de explicar, lo primero será codificar el dato de entrada $(X)$ que en este caso estará formado por los dos sumandos. La forma más sencilla de codificar un número $n$ es repitiendo un símbolo $n$ veces seguidas. Vamos a suponer que los símbolos del alfabeto de nuestra máquina sólo son dos: el 1 y el blanco (b); según esto, la forma de codificar un número $n$ sería escribir en la cinta $n$ 1's seguidos. Codificaremos en nuestra cinta de esta forma los dos sumandos separados por un blanco:*

$$\ldots bbb11\ldots(n_1)\ldots11b11\ldots(n_2)\ldots1bbb\ldots$$

*Una forma de obtener la suma de $n_1$ y $n_2$ es cambiar el blanco de separación intermedio por un 1 y uno de los 1's extremos por un blanco (por ejemplo el de la izquierda). Esta operación puede realizarse con una máquina de Turing de 5 estados con el programa de la figura 1.5, según el formato anterior, donde se supone la cabeza de lectura/escritura situada inicialmente a la izquierda de ambos datos.*

○

## 1.5. Limitaciones de los computadores

Los computadores tienen muchas limitaciones, siendo las más importantes las siguientes:

**Problemas irresolubles**: Vamos a tratar de definir, en términos de máquinas de Turing, el concepto de función computable. Se dice que una función $f$ es computable, si $f(X)$ puede ser evaluada para cualquier valor de $X$ en un número finito de pasos por una máquina de Turing. Existe toda una teoría para determinar si una función es o no computable, esta teoría se denomina **Teoría de la computabilidad**.

**Problemas no finitos**: las máquinas de Turing tienen una suposición no realizable: la cinta es infinita y, por tanto, puede tener infinito número de estados. Desde este punto de vista las máquinas de Turing pueden considerarse computadores de infinitos estados. Los computadores reales son máquinas con un número finito de estados puesto que **su memoria es limitada**. Existen ciertos problemas que una máquina de Turing puede resolver y un computador de estados finitos no. Por ejemplo una máquina de estados finitos no puede multiplicar exactamente dos números binarios de una longitud arbitraria, sin embargo, una máquina de Turing podría hacerlo. Esta limitación sólo tiene relevancia para algunos problemas matemáticos porque actualmente la memoria de los computadores es finita pero muy grande.

**Limitaciones de velocidad**: La velocidad de los computadores ha ido aumentando con el tiempo según las evoluciones de la Tecnología. En cualquier caso, existen unos **límites físicos** para el aumento de la velocidad de las máquinas:

- Por una parte existe una cota superior de velocidad que es la velocidad de la luz, que también es la velocidad de transmisión de los campos eléctricos. Este límite hace, por ejemplo, que el tiempo de retardo a través de un conductor de 3 cm. sea

$$\frac{3 * 10^{-2}\ \mathrm{m}}{3 * 10^{8}\ \mathrm{m/sg}} = 10^{-10}\ \mathrm{sg} = 10^{-1}\ \mathrm{nsg}$$

Lo que correspondería a una frecuencia de

$$\frac{1}{10^{-10}\ \mathrm{sg}} = 10\ \mathrm{GHz}$$

Esto también nos da idea de que los dispositivos podrán ser más rápidos cuanto más pequeños sean. Existe otra limitación asociada a este hecho: los computadores más

rápidos se calientan más y, por ello, necesitan más superficie para disipar el calor; si son muy pequeños, no tendrán superficie suficiente para ello y ese calor acumulado en su interior podrá llegar a destruir algunos de sus componentes.

- Los computadores actuales son electrónicos, esto hace que la información tenga que fluir a través de conductores eléctricos que tienen una pequeña capacidad, por ello, el tiempo mínimo de transmisión de una información a través de un conductor es el tiempo que esa capacidad tarde en cargarse. Aquí incidimos en la misma idea anterior, cuanto más pequeños sean los conductores, menor será su capacidad y menos tiempo tardarán en cargarse. Para obviar este inconveniente, actualmente se está trabajando en **computadores ópticos** en los que este problema no existe.

La velocidad con que operan los computadores se mide por el número de operaciones básicas que pueden realizar por unidad de tiempo. Una unidad para medir esta velocidad es el **MIPS** (1 millón de instrucciones/sg.), sin embargo esta unidad tiene el inconveniente de que no es homogénea porque las instrucciones de un procesador pueden ser mucho más potentes que las de otro. Una unidad algo más fiable es el **MFLOPS** (1 millón de instrucciones de punto flotante/sg.) que, aunque en menor medida, también adolece del mismo defecto. Otra unidad que se emplea mucho es el **VUP** (VAX *process unit*) que es la velocidad del computador en cuestión tomando como unidad la del VAX-11/780, pero aquí también existe un inconveniente: ¿Qué tipo de programas usamos para comparar ambas máquinas? Este problema, aparentemente sencillo, no tiene solución fácil ya que programas diferentes pueden causar resultados muy dispares. Existen varios tipos de programas para evaluar el rendimiento de un computador (Hennessy & Patterson, 2003):

**Aplicaciones reales**: Se harían las pruebas con los programas que el usuario vaya a ejecutar en ese computador. Si el uso no va a ser concreto se pueden utilizar programas variados de uso frecuente: procesadores de textos, compiladores, programas de CAD, etc.

**Aplicaciones modificadas**: Se extraería de las aplicaciones reales la parte que se pretende medir, prescindiendo de lo demás. Por ejemplo, si sólo se quiere medir la calidad del procesador, se quitarían las partes del programa dedicadas a entrada/salida, etc.

**Núcleos**: Se toman partes representativas de programas reales y se unen en un programa que sólo se usa para evaluar el rendimiento. Un programa de prueba muy conocido de este tipo es el llamado *test de Linpack*.

***Benchmarks***[1] **reducidos**, que son pequeños programas de prueba cuyo resultado es conocido antes de su ejecución. Ejemplos muy tradicionales de este grupo de programas son la inversión de una matriz y el algoritmo de clasificación rápida (*Quicksort*).

***Benchmarks* sintéticos**: Estos programas son una evolución de los núcleos e intentan recoger en un programa de prueba las operaciones más utilizadas en programas reales de índole muy diversa, promediadas en función de su uso en esas aplicaciones reales. Programas de este tipo muy usados son el *Whetstone* y el *Dhrystone*.

[1] Se denominan *benchmarks* a los programas destinados a medir la velocidad de los computadores.

Actualmente existen *benchmarks* establecidos para medir las prestaciones de los computadores. La mayoría de ellos proceden de SPEC (*Standard Performance Evaluation Corporation*) que dispone de programas de prueba para medir diferentes características de las máquinas. Concretamente el que se está utilizando para evaluar la velocidad de la CPU en la actualidad es SPEC CPU2006.

Para comparar máquinas distintas ante un mismo *benchmark* se puede utilizar la siguiente fórmula que mide su tiempo de ejecución:

$$\text{Tiempo de ejecución} = I * C * \frac{1}{f}$$

donde $I$ es el número de instrucciones que la máquina necesita para ejecutar el *benchmark*, $C$ es el número medio de ciclos de reloj precisos para la ejecución de una instrucción (también llamado CPI) y $f$ es la frecuencia del reloj.

## 1.6. El modelo de Von Neumann

El modelo que marca la estructura de los computadores actuales es el **modelo o arquitectura de Von Neumann**. Los elementos esenciales de este modelo son un **procesador** y una **memoria**:

- La **memoria** es un lugar de almacenamiento donde se guardan **las instrucciones junto con los datos**; esto significa que, en este modelo, una palabra de la memoria puede ser una instrucción o un dato y no hay forma de saberlo si no es por el contexto en que esa palabra se utilice.
- El **procesador** es un intérprete de un juego de instrucciones. Sus funciones son:
  - **Extraer la instrucción** de memoria y **decodificarla**.
  - **Ejecutar** la instrucción que ha sido extraída.
  - **Localizar la siguiente instrucción** para volver al primer paso.

Este proceso continuaría indefinidamente y de esta forma el procesador podrá ejecutar cualquier tipo de programas por grande que fuere su complejidad. Evidentemente el mismo procesador deberá poseer algún tipo de memoria interna con el fin de contener la información acerca de los sucesivos **estados** por los que tiene que pasar para ejecutar cada una de las instrucciones. Esa memoria interna también deberá contener la información acerca de la siguiente instrucción a ejecutar. En la práctica, esa memoria interna, se materializa en los **registros**, que no son otra cosa que memorias de muy poca capacidad, y sin embargo muy rápidas, que forman parte del procesador.

La mayoría de los computadores actuales siguen los mismos patrones de la arquitectura de Von Neumann aunque se haya progresado tanto en los niveles inferiores a este modelo (hardware, tecnologías de circuitos, etc) como en los niveles superiores (lenguajes de alto nivel, sistemas operativos, etc.).

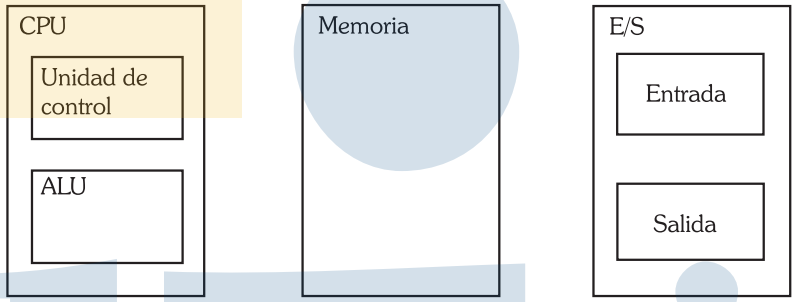


**Fig. 1.6**. Unidades funcionales.

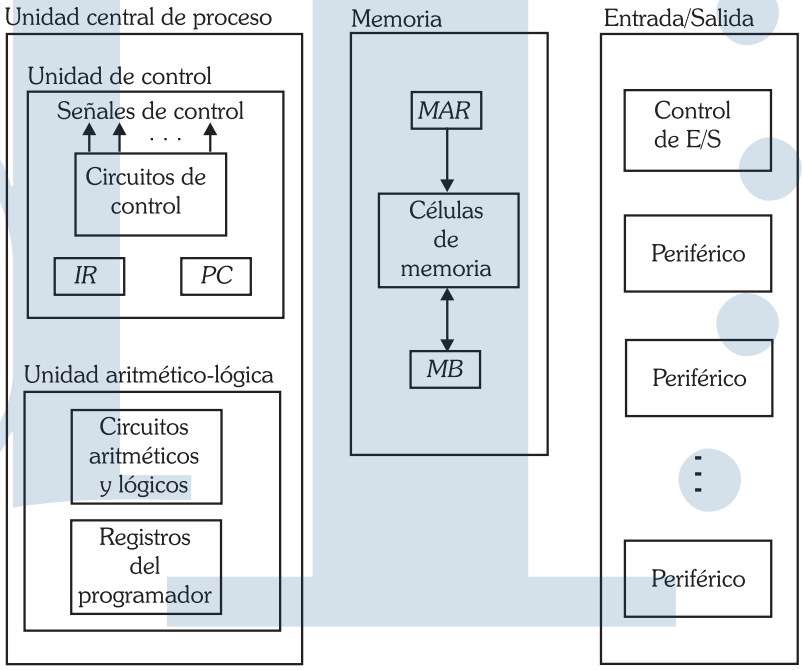


**Fig. 1.7**. Estructura interna de las unidades funcionales.

## 1.7. Unidades funcionales

En líneas generales, los computadores actuales tienen los mismos bloques que los diseñados por Babbage o Von Neumann, a estos bloques en adelante las llamaremos **unidades funcionales** y se muestran en la figura 1.6. La razón de este nombre estriba en que cada una de estas unidades está especializada en la realización de una función. Vamos a especificar con un poco más de detalle la estructura interna de cada una de ellas; esa estructura puede verse en la figura 1.7. Los registros más importantes que se pueden encontrar son:

- El **contador de programa** o **contador de instrucciones** ($PC$, *program counter*): contiene la dirección de la siguiente instrucción a ejecutar.
- El **registro de instrucción** ($IR$, *instruction register*): en este registro se almacena la instrucción que se está ejecutando en cada momento.

```
const
    m = número máximo almacenable en una palabra; {2↑nº de bits-1}
    n = tamaño de la memoria;
    p = número de registros del procesador;
type
    palabra = 0..m;
    dir = 0..n-1;
    memoria = array[0..n-1] of palabra;
    registros = array[0..p-1] of palabra;
procedure interprete (var mem:memoria; var regs:registros; inicio:dir)
var
    pc, dirdato: dir;
    ir, dato: palabra;
    haydato, fin: boolean;
    tipoinstr: integer;
begin
    pc := inicio;
    fin := false;
    while not fin do
    begin
        ir := mem[pc]; pc := pc + 1;
        determinatipo (ir, tipoinstr, haydato);
        if haydato then
        begin
            determinadirdato (dirdato);
            dato := mem[dirdato]
        end
        ejecuta (tipoinstr, dato, mem, regs, pc, fin)
    end
end
```

**Fig. 1.8**. Intérprete para un procesador con la estructura de la figura 1.7 (Pascal).



- Los **registros para uso del programador**, que hemos situado en la ALU, y que pueden tener diferentes estructuras según cuál sea la organización interna del procesador.
- El **registro de dirección de memoria** (*MAR*, *memory address register*): contiene la dirección de memoria donde se va a leer o escribir.
- El registro ***buffer* de memoria** (*MB*, *memory buffer* o *MDR*, *memory data register*): contiene la información leída, en una operación de lectura, o la información que se va a escribir, en el caso de una operación de escritura.

```
#define n tamaño de la memoria
#define p número de registros del procesador
typedef .... palabra; /* char, int, long int, etc.*/
typedef palabra *direccion;
typedef palabra memoria[n];
typedef palabra registros[p];
interprete (memoria mem; direccion inicio; registros regs)
{
    direccion pc, dirdato;
    palabra ir, dato;
    int haydato, fin, tipoinstr;
        pc = inicio;
        fin = 0;
        while (!fin)
        {
            ir = *pc; pc++;
            determinatipo (ir, &tipoinstr, &haydato);
            if (haydato)
            {
                determinadirdato (&dirdato);
                dato = *dirdato;
            }
        ejecuta (tipoinstr, &dato, mem, regs, &pc, &fin);
        }
}
```

**Fig. 1.9**. Intérprete para un procesador con la estructura de la figura 1.7 (C).

## 1.8. Ejecución de las instrucciones

Un programa consta de una secuencia ordenada de instrucciones. El proceso de ejecución de cada una de esas instrucciones pasa por los siguientes **ciclos o fases**:

- El **ciclo o fase de búsqueda**: consiste en la lectura de la instrucción y la búsqueda de sus operandos, si los tiene.
- El **ciclo o fase de ejecución**: consiste en la ejecución de la instrucción y la escritura del resultado.

Cada una de estas fases o ciclos puede descomponerse a su vez en varias etapas más elementales:

**Ciclo de búsqueda**:

- **Extraer de memoria la instrucción** y llevarla al registro de instrucción. Para hacerlo se carga el contador de programa ($PC$) en el registro de dirección de memoria

($MAR$) y el resultado obtenido en el *buffer* de memoria ($MB$) se pasa al registro de instrucción ($IR$).

- **Incrementar el contenido del** $PC$ para que contenga la dirección de la instrucción siguiente.
- **Determinar el tipo de instrucción** que se ha extraído y **determinar el número de sus operandos**.
- **Determinar la localización de los operandos**, si la instrucción los tiene.
- Leer los operandos, si los hay.

**Ciclo de ejecución**:

- **Ejecutar** la instrucción.
- **Almacenar** los resultados en el lugar apropiado.
- **Volver al primer paso** del proceso.

Todos estos pasos pueden resumirse en el procedimiento en Pascal de la figura 1.8 o en la función en C de la figura 1.9.

## 1.9. Clasificaciones de los procesadores

Una clasificación tradicional de los computadores divide a éstos en **maxicomputadores** (o *main frames*), **minicomputadores** y **microcomputadores**. Hace años estos tipos de máquinas tenían diferencias muy claras, por ejemplo, en la longitud de palabra, en la cantidad de memoria, etc. Actualmente, debido a los avances en las tecnologías de circuitos digitales, estas diferencias carecen de significado; hay micros con palabra de 32 bits que pueden direccionar mucha más memoria que una maxi de hace pocos años. Por otra parte, el espectro de los computadores es continuo en cuanto a potencias y posibilidades lo que hace que la clasificación anterior carezca de sentido. Esa clasificación obedece más al uso que se da a la máquina que a las características de la misma: un microcomputador es un computador de uso personal, un mini es un computador para uso de un departamento o pocas personas y una maxi es un computador para toda una gran empresa con un gran número de usuarios. Desde hace años existe un nuevo escalón en esta clasificación que son las estaciones de trabajo (*workstation*). Estas estaciones son máquinas de uso personal pero con una gran potencia de cálculo y muchos recursos, tales como discos de gran tamaño, monitores gráficos de gran resolución, etc. Este escalón se situaría entre los minis y los microcomputadores.

La clasificación en cuanto al uso sigue estando vigente actualmente, aunque se hayan cambiado los nombres: el primer estadio serían los **ordenadores personales** o PC (*personal computers*), anteriormente denominados microcomputadores; luego estarían las **estaciones de trabajo**, mencionadas anteriormente, y el resto serían **servidores**, aunque de tamaños diferentes.

En cuanto a la organización interna de los procesadores, se pueden establecer también, tres grandes grupos:

- Procesadores de **pilas**: basan su funcionamiento en una pila, en cuya cima se dejan los operandos y el procesador deposita sus resultados.

- Procesadores con un solo **acumulador**: estas máquinas tienen un registro preferente, llamado precisamente **acumulador**, donde se deposita obligatoriamente uno de los operandos y queda el resultado de cada operación. Suelen tener otros registros de propósito específico para almacenar direcciones, apuntadores, etc.

- Procesadores con **registros de propósito general**: al contrario que las máquinas del grupo anterior, éstas tienen varios registros en que se pueden depositar datos, direcciones, etc.

Si bien esta clasificación está bien establecida, no hay una frontera muy clara entre estos tipos, especialmente entre los dos últimos.

La tendencia actual se dirige hacia procesadores con registros de propósito general. La razón para ello es que los registros de la CPU son mucho más rápidos que la memoria y por ello conviene tener suficientes registros para trasladar todas las informaciones que se usen con frecuencia a esos registros. A su vez, las máquinas con registros de propósito general se pueden también clasificar con arreglo al lugar donde residen los operandos de sus instrucciones en tres tipos:

**Registro-registro**, también llamadas máquinas de **carga-almacenamiento**, que se caracterizan por que la mayoría de instrucciones no acceden a memoria y trabajan con operandos que están en los registros. Sólo hay dos clases de instrucciones que acceden a memoria: las instrucciones de **carga**, que transportan datos de la memoria a los registros y se usarán para cargar en los registros los operandos, y las instrucciones de **almacenamiento**, que guardan en memoria el contenido de los registros y por tanto se utilizarán para llevar a la memoria los resultados de las operaciones. Las instrucciones de estos procesadore pueden tener tres operandos: los **operandos fuente** y el **resultado**. Las máquinas RISC mencionadas con anterioridad son de este tipo.

**Registro-memoria**, las instrucciones pueden tener uno de los operandos en memoria pero no los dos. Estas máquinas suelen tener sólo dos operandos por lo que uno de los operandos fuente sirve también para almacenar el resultado, es decir se destruye.

**Memoria-memoria**, estas máquinas permiten que todos sus operandos residan en memoria, aunque pueden también residir en registros, esto hace que sus operaciones sean muy potentes pero también muy lentas. Este tipo de procesadores pueden tener instrucciones de dos o tres operandos.

A lo largo del curso se irán viendo ejemplos y diferencias entre todos estos tipos de máquinas, aunque la mayoría de los ejemplos se refieren a organizaciones de registros de uso general.

## 1.10. Concepto de arquitectura de un computador

El término **arquitectura de un computador** cubre todos los aspectos de su organización. Entre estos aspectos se incluyen las longitudes de palabra, de instrucción (variable o fija), número de direcciones de cada instrucción, conjunto de instrucciones, modos de direccionamiento, etc. Otras cuestiones se refieren a cómo está organizada la memoria y cómo se conectan los

dispositivos de entrada y salida. Como se puede ver, el campo de la Arquitectura de Computadores es extenso y rico ya que cubre la mayoría de los aspectos de su diseño. Bajo el título de **Arquitectura de Computadores** u **Organización de Computadores** se agrupan normalmente ideas semejantes, esto hace que a la última clasificación del párrafo anterior, se le denomine, en ocasiones, **clasificación de las arquitecturas**.

Se puede definir la Arquitectura de Computadores como el *estudio de la estructura, funcionamiento y diseño de computadores*; esto incluye, sobre todo aspectos de hardware, pero también afecta a cuestiones de software de bajo nivel.

## 1.11. Niveles de estudio y descripción de un computador

El estudio de un computador puede realizarse desde diferentes puntos de vista según lo que interese para cada caso. Explicaremos estos niveles de estudio y descripción en orden creciente de nivel (es decir desde el interior la máquina hacia el exterior):

**Descripción geométrica y de componentes**: En este nivel se estudiarían los diferentes dispositivos electrónicos que forman el computador y su disposición geométrica dentro de los circuitos integrados. Este nivel es objeto de estudio de la Electrónica.

**Descripción de circuito electrónico**: Aquí se estudiarían la conjunción de componentes elementales para formar elementos lógicos (puertas, inversores, etc.). Este nivel se sitúa entre la Electrónica analógica y la digital.

**Descripción de circuitos digitales**: Aquí se estudiaría la unión de los diferentes elementos lógicos simples para formar circuitos que realizan funciones específicas (registros, contadores, decodificadores, etc.). Este nivel lo estudia la Electrónica digital.

**Nivel de transferencia de registros (RTL)**: En este nivel se especifican los flujos de información entre los diferentes registros para la ejecución de cada una de las instrucciones. Desde nuestro punto de vista este nivel es uno de los más interesantes.

**Descripción funcional**: Este nivel describe la conducta de entrada y salida del sistema sin entrar en su estructura interna. Es decir, el computador, en este nivel, sería una caja negra que realizaría una serie de operaciones cuya función está bien definida.

## 1.12. Máquinas multinivel

Un computador es un máquina que puede resolver problemas ejecutando una sucesión de instrucciones. Se llama **programa** a una *secuencia de instrucciones que describe cómo realizar alguna tarea*. Los circuitos de un computador pueden reconocer y ejecutar un conjunto muy limitado de instrucciones que, además son muy simples. El conjunto de las instrucciones primitivas de un computador forma un lenguaje que se llama **lenguaje máquina**. Todos los programas que se deseen ejecutar en un computador deben convertirse previamente en una secuencia de instrucciones de este lenguaje. A la gran distancia existente entre el lenguaje empleado por el hombre para comunicarse con la máquina y las instrucciones que los circuitos son

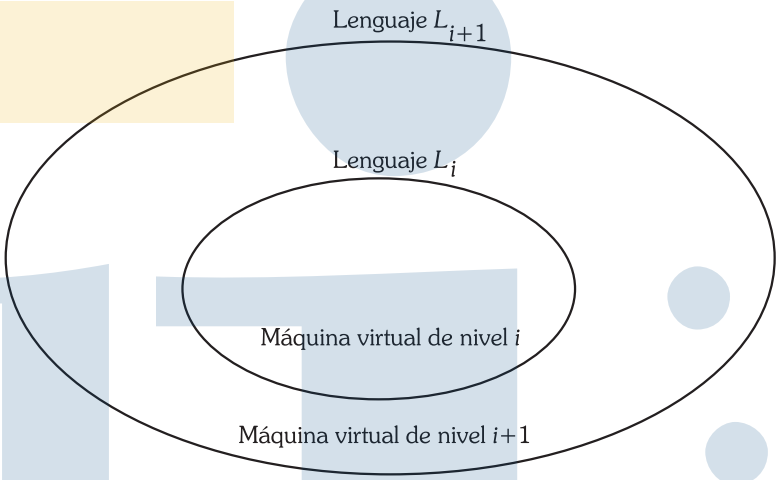


**Fig. 1.10**. Ilustración del concepto de máquina virtual

capaces de ejecutar se le llama **barrera** o **brecha semántica** (*semantic gap*). Debido a que la mayoría de los lenguajes máquina son demasiado elementales, es difícil y tedioso utilizarlos, por eso debe saltarse la barrera semántica y utilizar lenguajes más cercanos al hombre. Dado que la barrera semántica es demasiado amplia y con el fin de que la traducción entre lenguajes no sea demasiado complicada, esta barrera no se salta de una sola vez, sino por etapas. Estas etapas son los llamados **niveles del computador**. Partiendo del lenguaje máquina, llamémosle $L_1$, se hace necesario el diseñar un lenguaje que sea más fácil de utilizar, $L_2$. Se hará necesaria la traducción de las instrucciones del lenguaje $L_2$ al $L_1$. En vez de hablar de traducción, se puede imaginar la existencia de una hipotética máquina, o **máquina virtual** $M_2$ (figura 1.10), siendo la máquina real $M_1$, cuyo lenguaje sea el $L_2$. Los lenguajes $L_1$ y $L_2$ no deben ser muy diferentes para que la traducción sea práctica y rápida. Esto limita al lenguaje $L_2$ y hace necesario el diseño de lenguajes de niveles superiores. Para una máquina actual, el esquema de niveles de máquinas virtuales podría ser el que se expone en la figura 1.11, aunque puede variar de unos modelos a otros. El nivel de lenguaje máquina es el de los programas que es capaz de interpretar un procesador dado; los niveles inferiores a éste normalmente no son accesibles al usuario (programador) de la máquina. El usuario final puede interpretar que el computador es una máquina virtual que entiende instrucciones del nivel más alto (lenguaje de aplicación o lenguaje de alto nivel) cuando realmente esas instrucciones tienen que ser traducidas hasta llegar al nivel ejecutable realmente por el hardware.

La traducción entre dos niveles $L_i$ y $L_{i-1}$ se hará mediante un **compilador** o un **intérprete**. Es interesante, en este punto, señalar la diferencia entre ambos:

- Un compilador toma cada instrucción del lenguaje $L_i$, lo sustituye por un grupo de instrucciones del lenguaje $L_{i-1}$ y continúa.

- Un intérprete toma un elemento del lenguaje $L_i$ y la traduce hasta el nivel más bajo $L_1$ (es decir, la ejecuta) antes de continuar con la siguiente.

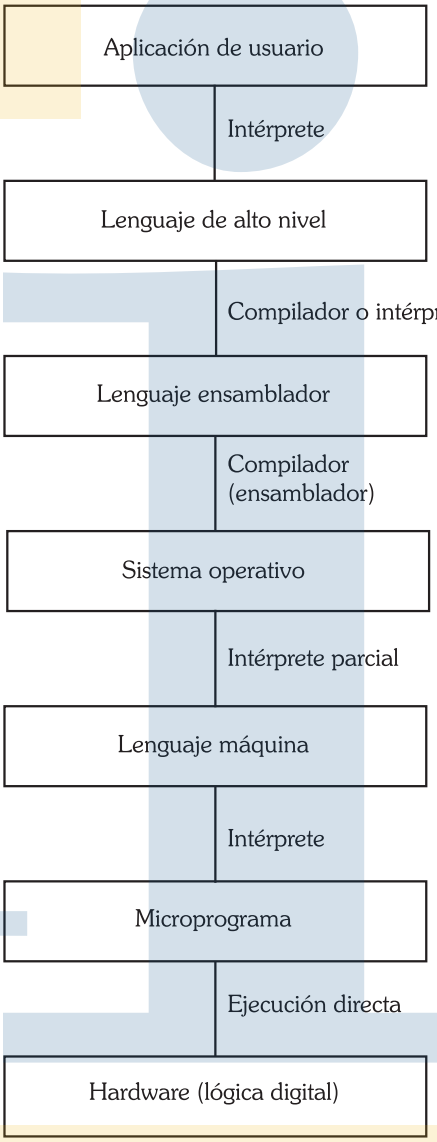


**Fig. 1.11**. Niveles de un computador actual.

## 1.13. Ejemplos de arquitecturas reales

En este apartado veremos las estructuras de registros de los procesadores que se van a estudiar más adelante, tal y como las ve un programador de lenguaje máquina. Se verán ejemplos de todos los tipos de arquitecturas expuestos anteriormente.

### 1.13.1. PDP-11

En la figura 1.12 se muestran los registros del PDP-11 (*Programmed Data Processor*) de DEC (*Digital Equipment Corporation*). Esta máquina dispone de 8 registros de uso general de 16 bits ($R0$ a $R7$), de los cuales $R6$ es el apuntador de pila ($SP$, *stack pointer*) y $R7$ es el

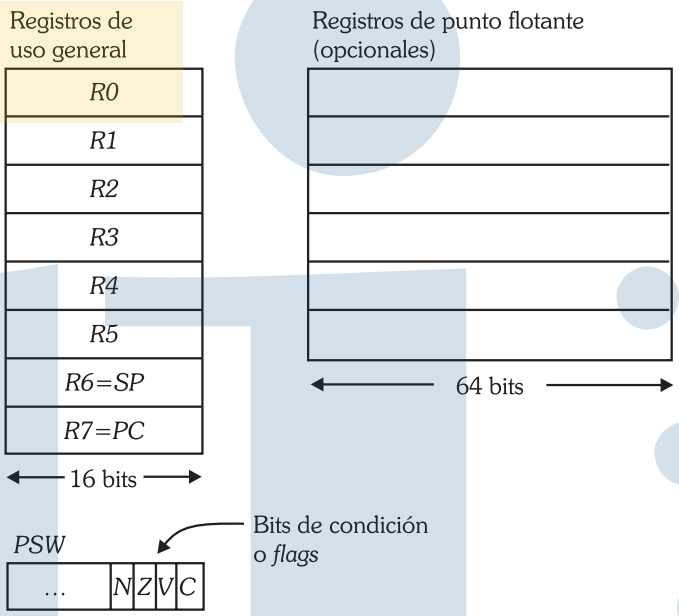


**Fig. 1.12**. Estructura de registros del procesador PDP-11.

contador de programa ($PC$, *program counter*). Como opción, puede tener 6 registros de punto flotante de 64 bits; esta opción amplía el juego de instrucciones del procesador original. El PDP-11 tiene **arquitectura de registros generales** del tipo **memoria-memoria**.

El hecho de considerar al $PC$ y al $SP$ como registros generales hace que estos registros, que en otros procesadores son únicamente de control, se traten como cualquier otro lo que da, como veremos más adelante, más potencia a los modos de direccionamiento. Otro registro muy importante es la palabra de estado del procesador ($PSW$, *Processor status word*) cuyos últimos bits se llaman **códigos** o **bits de condición** (*flags*) e indican algunas propiedades del resultado de la última operación efectuada. Estos bits son: $Z$ (cero), $N$ (negativo), $V$ (desbordamiento u *overflow*) y $C$ (acarreo, llevada o *carry*). Los códigos de condición se usan para tomar las decisiones en las bifurcaciones condicionales como se verá en el capítulo 4.

Una cuestión que hay que mencionar es la organización de la memoria. La información mínima direccionable en la memoria del PDP-11 es el byte, pero se pueden también direccionar palabras (16 bits). La organización de la memoria obliga a que las palabras siempre se direccionen en direcciones pares. Este hecho se llama **alineación**. En cualquier máquina, se dice que un acceso a un objeto de $s$ bytes está alineado en la dirección $A$ si $A \bmod s = 0$, es decir, si los últimos $\log_2 s$ bits de la dirección $A$ son 0 (suponiendo que $s$ sea potencia de 2). En el PDP-11, la alineación afecta especialmente a las instrucciones cuyas longitudes siempre son múltiplos de una palabra y por tanto siempre deben residir en direcciones pares. Esto hace que el contenido del contador de programa siempre sea par y se deba incrementar en 2 cada vez que se lee una palabra de código. Por otra parte, cuando se accede a una palabra, el byte de menor peso se sitúa en la dirección más baja de las dos que ocupa. Esta forma de trabajar se denomina *little-endian*, si los bytes se ordenaran de forma inversa se dice que el ordenador funciona en *big-endian* (Cohen, 1981).

Es necesario comentar que el error más grave en el diseño del PDP-11 fue dimensionar sus

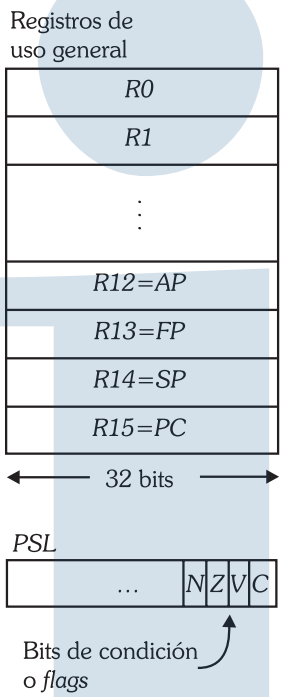


**Fig. 1.13**. Registros de la arquitectura VAX.

direcciones con 16 bits que, aunque en la época eran suficientes, se quedaron cortos en muy poco tiempo.

### 1.13.2. VAX

Esta máquina (originalmente llamada VAX-11, *Virtual Address Extended* PDP-11: PDP-11 extendido con direcciones virtuales), también de DEC, está basada en la anterior pero es más ambiciosa ya que está concebida como una gran computador. Tiene también organización de registros generales pero de 32 bits y en mayor número. También el $PC$ y el $SP$ se consideran registros de uso general con las mismas implicaciones que en el PDP-11. Existen otros registros de control: $R12$ o $AP$ (*argument pointer*, apuntador a los argumentos) y $R13$ o $FP$ (*frame pointer*, apuntador de trama), estos registros se usan en las llamadas a procedimientos. La palabra de estado del procesador pasa a ser una doble palabra ($PSL$: *processor status longword*: doble palabra de estado del procesador) y los últimos cuatro bits tienen el mismo significado que en el PDP-11 (**bits de condición**). Puede manejar diferentes tamaños de datos: byte, palabra (*word*, 2 bytes), doble palabra (*longword*, 4 bytes) y cuádruple palabra (*quadword*, 8 bytes). La mínima información direccionable es el byte pero, si la información direccionada es mayor no hay restricciones en cuanto a su dirección; sin embargo, si se respeta la alineación, los accesos a memoria son más rápidos. En cuanto a la ordenación de los bytes de los tipos datos anteriores, este procesador sigue la filosofía *little-endian* (recuérdese el apartado 1.13.1).

El VAX es muy versátil en cuanto a los tipos de datos que puede manejar: enteros de diferentes tamaños (desde un byte hasta ocho), números en punto flotante en dos tamaños (simple precisión, 4 bytes y doble precisión, 8 bytes), caracteres, números en decimal empaquetado

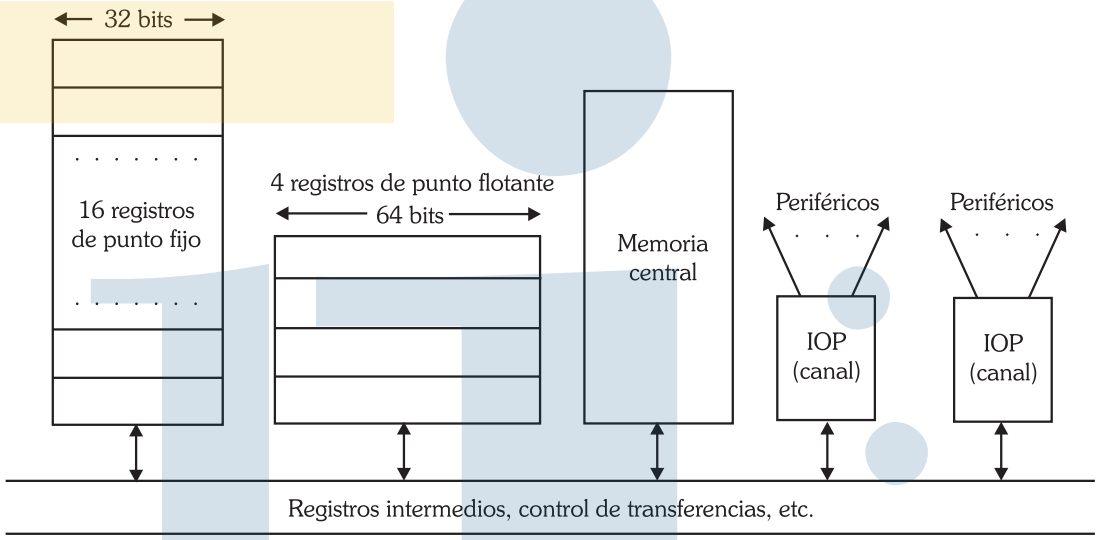


**Fig. 1.14**. Estructura de los computadores de las series IBM-360/370.

(BCD), cadenas y otros de menor importancia. Este procesador es el ejemplo más típico de una **máquina de registros de uso general con arquitectura memoria-memoria**.

La estructura de registros de los procesadores con arquitectura VAX puede verse en la figura 1.13.

Entre las características más importantes de su organización cabe destacar la incorporación de una memoria caché, para aumentar la velocidad de trabajo, y un acelerador de punto flotante opcional cuya misión es incrementar la velocidad de las instrucciones de punto flotante que, a diferencia del PDP-11, son posibles aunque este acelerador no esté presente.

### 1.13.3. IBM-360/370

En la figura 1.14 puede verse la estructura, desde el punto de vista del programador, de los computadores IBM de la serie 360, que más tarde se continuó con la 370 (que en realidad emula a la anterior). Este computador fue el primero en introducir los registros de uso general, tanto en punto fijo (para aritmética entera) como en punto flotante. Otra característica de esta serie de computadores es la gestión de las operaciones de entrada y salida mediante procesadores dedicados denominados **IOP** (*input-output procesors*) o **canales**). Esta máquina es un **computador con registros de uso general** que está a medio camino entre las arquitecturas registro-memoria y memoria-memoria, aunque son pocas las instrucciones que obedecen al segundo esquema.

### 1.13.4. HP-3000

Este computador es un mini de 16 bits cuya principal característica es tener una **arquitectura basada en pila**. La memoria de esta máquina contiene las instrucciones y los datos en segmentos separados. El esquema de los registros apuntadores y su utilización puede verse en la figura 1.15. El área de programa (figura 1.15 (a)) está limitado por los contenidos de los

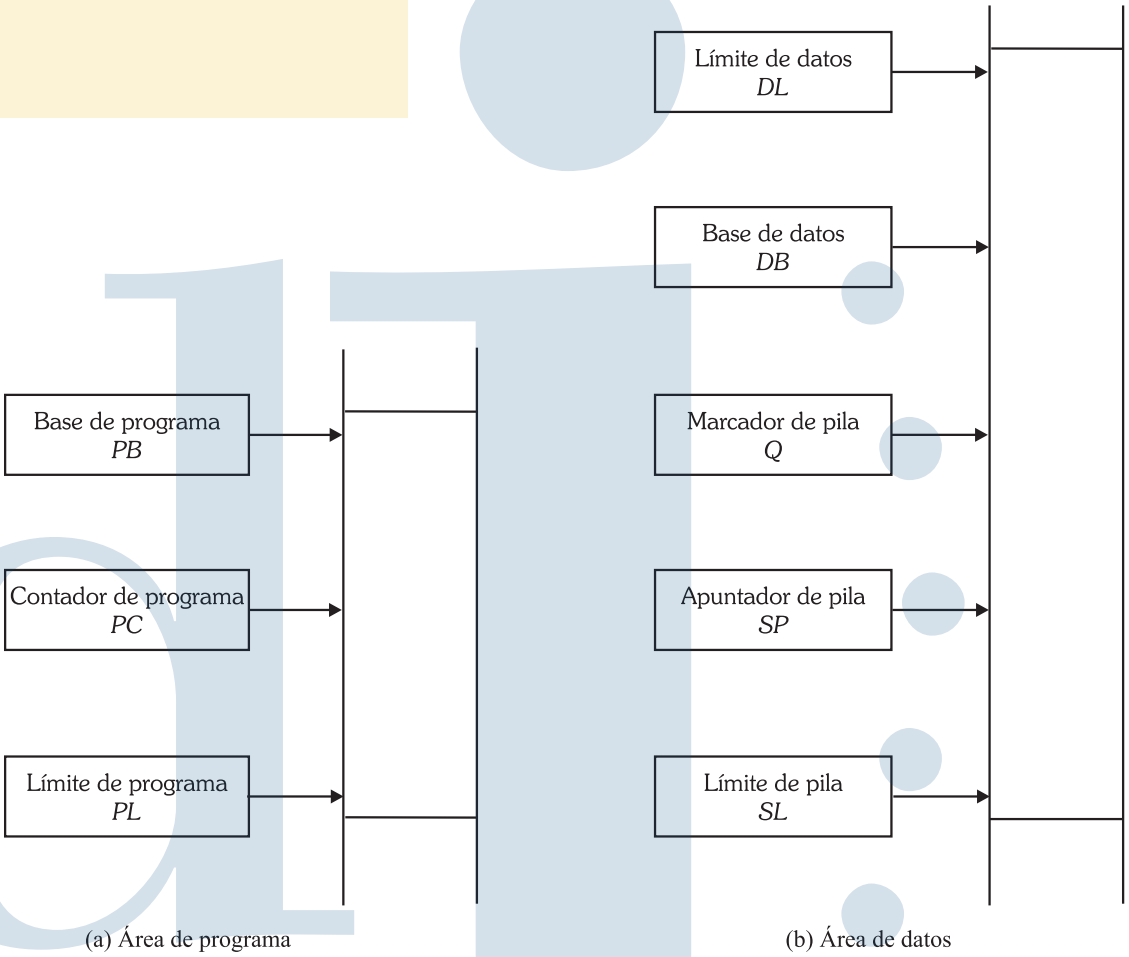


**Fig. 1.15**. Organización de código y datos en la memoria del HP3000.

registros $PB$ (Base de programa) y $PL$ (*Program Limit*: límite de programa). El contador de programa ($PC$) tiene la función convencional de apuntar a la siguiente instrucción a ejecutar. El área de datos (figura 1.15 (b)) se divide, a su vez, en dos partes: el área de datos convencional y el área de pila. La frontera divisoria de ambas zonas está determinada por el contenido del registro $DB$ (apuntador base de datos). Los límites de las áreas los marcan los contenidos de los registros $DL$ (*Data Limit*: límite de datos) y $SL$ (*Stack Limit*: límite de pila). El elemento de cima de pila está señalado, como siempre, por el $SP$ (apuntador de pila). El hardware verifica que la pila crezca siempre dentro de los límites marcados por $DB$ y $SL$. El registro $Q$ (marcador de pila) se emplea para delimitar los datos del procedimiento actual (trama de pila del procedimiento).

Una de las pegas que se puede argumentar a la organización de pila es que la velocidad está limitada por los múltiples accesos a memoria. Este inconveniente puede paliarse haciendo que los datos más altos de la pila radiquen en registros del procesador. Existen 4 registros para almacenar de 0 a 4 elementos de la pila, por ello se utilizan dos registros de direccionamiento más: el $SR$ (*Stack in register*), de 3 bits, que indica cuántos registros se están utilizando, y el $SM$ (*stack in memory*) que indica cuál es el último elemento de la pila en memoria de manera

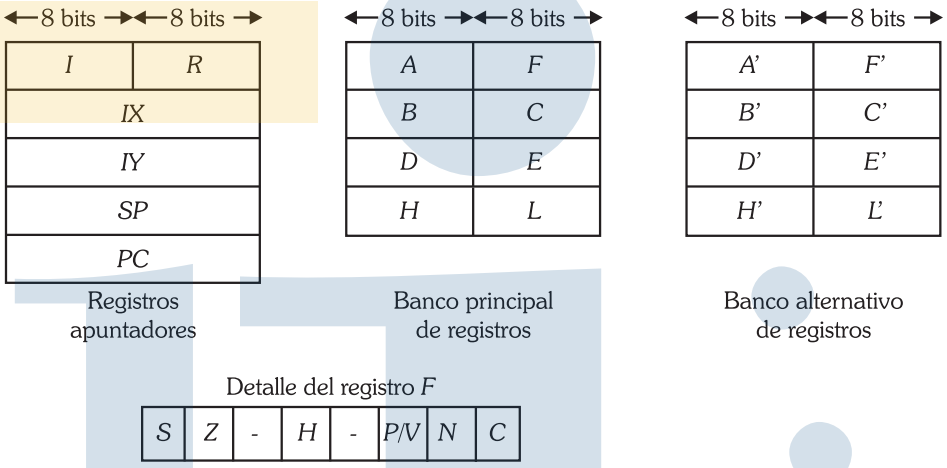


**Fig. 1.16**. Registros del microprocesador Z-80.

que siempre se cumple que

$$SP = SM + SR$$

En cualquier caso, el programador no tiene que tener ninguna consciencia de la existencia de estos registros que contienen las últimas posiciones de la pila.

### 1.13.5. Z-80

El Z-80 de Zilog es un microprocesador de 8 bits aunque tiene instrucciones que manejan números de 16 bits (en forma *little-endian*, recuérdese el apartado 1.13.1). Su arquitectura es compatible en software con el Intel 8080 y tiene sus instrucciones como un subconjunto. Esta compatibilidad le ha condicionado mucho, aunque el Z-80 ha mejorado al 8080 en muchas cuestiones como tecnología, velocidad, nuevos modos de direccionamiento, juego de instrucciones más amplio, banco de registros más completo, etc.

El Z-80 es una máquina que está **entre las arquitecturas de acumulador y de registros generales**. Si se considera en este último grupo, la situaríamos en el tipo **registro-memoria**. La palabra del Z-80 es de 8 bits aunque puede direccionar datos de 16 sin limitaciones de alineación. La estructura de registros del Z-80 se muestra en la figura 1.16. Se observa que hay dos bancos de registros de datos y un juego de registros especiales (normalmente para direccionamiento). El fin de los dos bancos de registros es ganar tiempo ante la presencia de interrupciones en que se puede conmutar al banco alternativo. Entre los registros cabe citar: $A$, que es el acumulador; $B$, $C$, $D$, $E$, $H$ y $L$ son registros generales de 8 bits que también pueden tomarse por parejas de 16 bits ($BC$, $DE$ y $HL$); $IX$ e $IY$, son los registros índices; $I$ que almacena la parte alta de la dirección de comienzo de la tabla de vectores de interrupción (ver apartado 4.4.6), $R$ es el registro que almacena el bloque de memoria que se va a refrescar y $F$ que es el registro de **señalizadores** (*flags*) que contiene los bits de condición. Cabe destacar que a los típicos bits $N$, $Z$, $V$ y $C$, el Z-80 añade dos más: $P$, bit de **paridad**, que indica si el número de unos del resultado ha sido par o impar, y $H$, **semillevada** (*half-carry*) que indica si ha habido llevada de la mitad baja a la mitad alta del resultado.

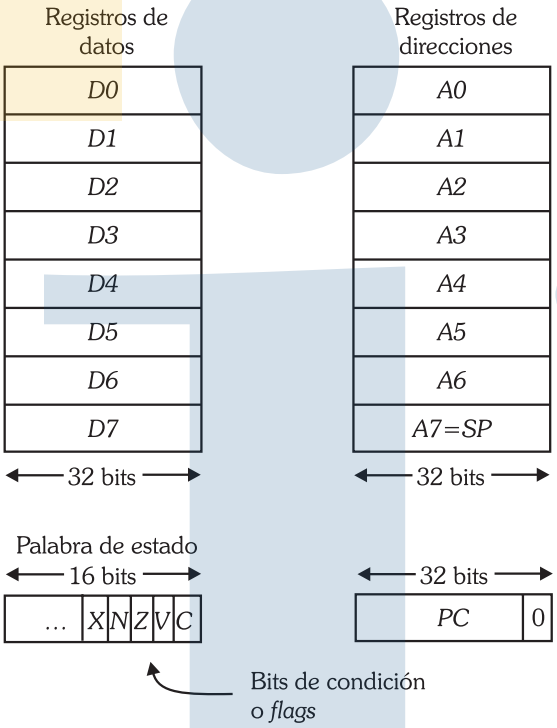


**Fig. 1.17**. Registros del procesador MC68000.

**Tabla 1.1**. Comparación de la familia de procesadores de Motorola MC680xx.

| Nombre | Año | Bus de datos | Espacio de direcciones | Comentarios |
|---|---|---|---|---|
| 68000 | 1979 | 16 | 16 M | Primer miembro de la familia |
| 68008 | 1982 | 8 | 4 M | Bus de datos de 8 bits |
| 68010 | 1983 | 16 | 16 M | Memoria virtual |
| 68012 | 1983 | 16 | 2 G | Mayor espacio direccionable |
| 68020 | 1984 | 32 | 4 G | CPU de 32 bits |
| 68030 | 1987 | 32 | 4 G | Incorpora manejo de memoria |
| 68040 | 1989 | 32 | 4 G | Coprocesador y memoria caché |
| 68060 | 1994 | 32 | 4 G | 2 procesadores segmentados |

### 1.13.6. MC68000 y derivados

El MC68000 de Motorola es un microprocesador muy potente con **organización de registros generales del tipo memoria-memoria**. Los registros están divididos en dos grupos: **registros de datos y de direcciones**, todos de 32 bits. Como se puede ver en la figura 1.17, el MC68000 tiene 8 registros de datos ($D0$-$D7$) y 8 de direcciones ($A0$-$A7$) de los cuales $A7$ es el apuntador de pila $SP$. Este registro, según el procesador esté en modo supervisor o en modo usuario puede representar dos registros diferentes: el $SSP$ (*Supervisor Stack Pointer*, apuntador de pila de supervisor) o el $USP$ (*User Stack Pointer*, Apuntador de pila de usuario). El registro

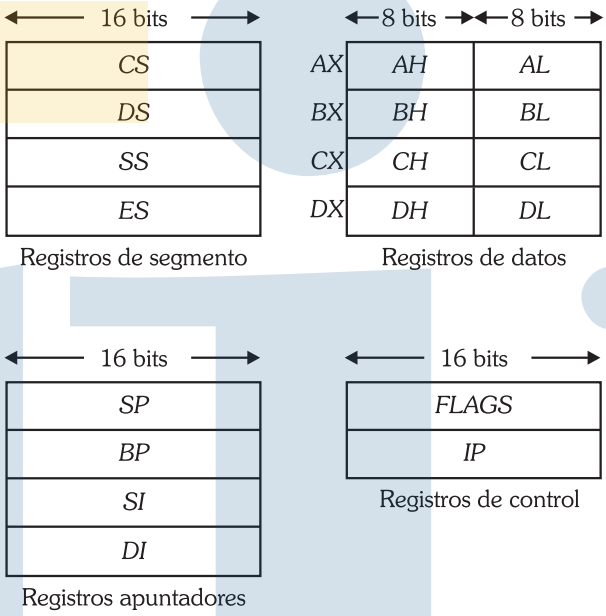


**Fig. 1.18**. Registros del microprocesador Intel 8086.

de estado tiene 16 bits entre los que se encuentran los bits de condición que además de los bits ya mencionados $N$, $Z$, $V$ y $C$ incluye otro bit más, $X$, denominado **bit de extensión** que sirve para encadenar operaciones sobre datos de mayor longitud de los soportados por la máquina. Aunque todos los registros tienen 32 bits, el bus de datos sólo tiene 16 y el bus de direcciones 23 ($A_{23}$-$A_1$), el bit de dirección $A_0$ se manda a la memoria mediante dos líneas de control que indican si se quiere acceder al byte bajo, al alto o a ambos (acceso a palabras de 16 bits). El byte es la información mínima direccionable, sin embargo, también se pueden direccionar palabras aunque sólo en direcciones pares (**alineación obligatoria**). Los tamaños de datos con que esta máquina puede trabajar son **byte**, **palabra** (2 bytes) y **doble palabra** (4 bytes), estos datos se organizan en memoria según la pauta *big-endian*.

Los principales procesadores derivados del MC68000 son: el MC68020, cuya arquitectura es como la del MC68000 pero con buses de 32 bits; el MC68030, que es similar al anterior pero con una unidad de manejo de memoria para facilitar el trabajo con memoria virtual, el MC68040, que es igual que el MC68030 pero con una memoria caché dentro de la pastilla y un coprocesador de punto flotante y, por último, el MC68060, que incorpora dos procesadores segmentados formando lo que se denomina un **procesador superescalar**. La tabla 1.1 muestra la evolución de esta familia de microprocesadores.

### 1.13.7. i-8086 y derivados

El procesador 8086 de Intel dispone de **cuatro tipos de registros** (figura 1.18): registros de **datos**, de **segmento**, **apuntadores**, y de **control**. Este procesador puede ser considerado como una **máquina de registros generales con arquitectura registro-memoria**.

Los registros de datos son de 16 bits y cada uno de ellos puede separarse en dos registros de 8 bits. Sus nombres son $AX$ ($AH$ y $AL$), $BX$ ($BH$ y $BL$), $CX$ ($CH$ y $CL$) y $DX$ ($DH$ y

**Tabla 1.2**. Evolución de los derivados del 8086 de Intel.

| Nombre | Año | Bus de datos | Espacio de direcciones | | Comentarios |
|---|---|---|---|---|---|
| | | | Real | Virtual | |
| 8086 | 1978 | 16 | 1 M | - | Primer microprocesador de 16 bits |
| 8088 | 1980 | 8 | 1 M | - | Bus de 8 bits |
| 80186 | 1982 | 16 | 1 M | - | 8086 + gestión de E/S |
| 80188 | 1982 | 8 | 1 M | - | 8088 + gestión de E/S |
| 80286 | 1982 | 16 | 16 M | 1 G | 8086 + gestión de memoria |
| 80386 | 1985 | 32 | 4 G | 64 T | Registros de 32 bits |
| 80386SX | 1988 | 16 | 4 G | 64 T | 80386 con bus de 16 bits |
| 80386SL | 1990 | 16 | 4 G | 64 T | 80386SX de bajo consumo |
| 80486 | 1989 | 32 | 4 G | 64 T | 80386 + coprocesador y caché |
| 80486SX | 1991 | 32 | 4 G | 64 T | 80486 sin coprocesador |
| Pentium | 1993 | 32 | 4 G | 64 T | Proceso paralelo y núcleo RISC |
| Pentium Pro | 1995 | 32 | 64 G | 64 T | Predicción de bifurcaciones |
| Pentium II | 1997 | 32 | 64 G | 64 T | Incorporación de instrucciones MMX |
| Celeron | 1998 | 32 | 64 G | 64 T | Pentium II de bajas prestaciones |
| Xeon | 1998 | 32 | 64 G | 64 T | Pentium II para servidores |
| Pentium III | 1999 | 32 | 64 G | 64 T | Nuevas instrucciones de punto flotante |
| Pentium 4 | 2000 | 32 | 64 G | 64 T | Nuevas instrucciones multimedia (SSE) |
| Itanium | 2001 | 64 | 64 G | 64 T | Nueva arquitectura de 64 bits |
| Itanium 2 | 2002 | 64 | 64 G | 64 T | Mayor profundidad de segmentación |

$DL$). Los registros apuntadores son $SI$ (índice fuente), $DI$ (índice destino), $SP$ (apuntador de pila) y $BP$ (apuntador de base). Los registros de segmento se usan porque el 8086 direcciona la memoria por segmentación, es decir, cada acceso a memoria se identifica por números: la dirección de comienzo de segmento ($S$) y el desplazamiento dentro del segmento ($d$). Los registros de segmento contienen los 16 bits más significativos de la dirección de comienzo de segmento (que tiene 20 bits). Los últimos 4 bits se suponen 0 porque la dirección de inicio de segmento debe ser múltiplo de 16. Para representar una dirección de la memoria del 8086 se escribe $S : d$ y la forma de calcular la dirección efectiva a partir de esta información es:

$$\text{Dirección efectiva} = 16\,S + d$$

La dirección de comienzo de segmento, $S$, radica en uno de los registros de segmento que son $CS$ (segmento de código), $DS$ (segmento de datos), $SS$ (segmento de pila) y $ES$ (segmento extra). Esta forma de segmentación limita la memoria direccionable por este procesador a 1 Mb. ya que las direcciones efectivas tienen 20 bits. Los registros de control también tienen 16 bits y son el registro de estado ($FLAGS$), con los flags de condición, y el apuntador de instrucciones ($IP$, *instruction pointer*) que hace las funciones de contador de programa referido al segmento de código, de forma que la dirección de la siguiente instrucción a ejecutar viene dada por:

$$\text{Dirección siguiente instrucción} = 16\,CS + IP$$

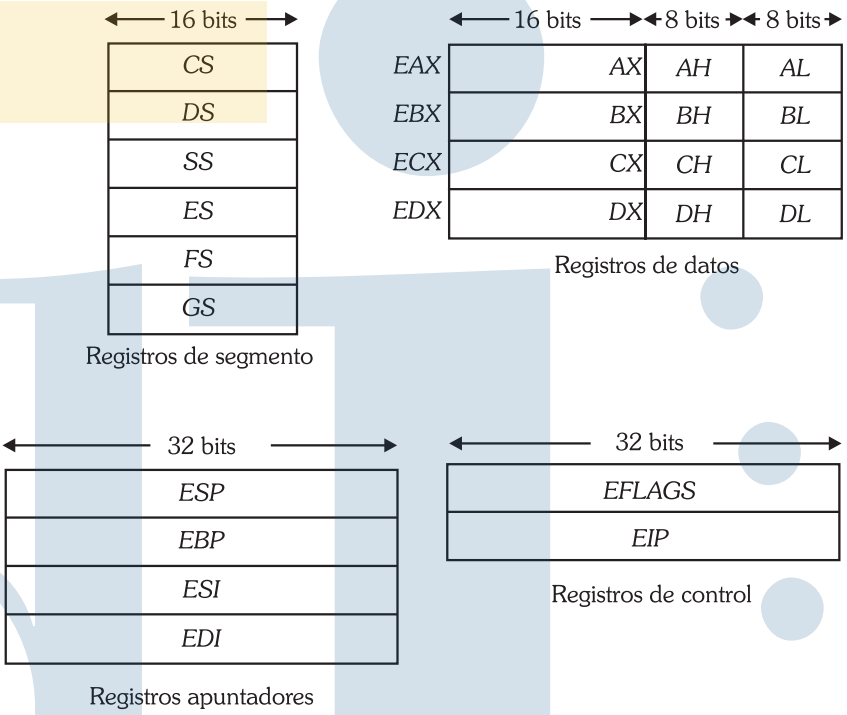


**Fig. 1.19**. Registros de los procesadores Intel 80386 y 80486.

El 8086 puede trabajar con datos de 8 y 16 bits sin restricciones de alineación pero la máquina es más eficiente si las palabras se encuentran en dirección par. La forma de almacenar los bytes es *little-endian*.

Entre los derivados del 8086 se encuentran el 8088, que es internamente igual al 8086 pero con el bus de datos externo de 8 bits, el 80286 que es una versión del 8086 que tiene incrementada su memoria direccionable a 16 Mb., el 80386 que tiene los registros y el bus de 32 bits, el 80386SX, que es una versión del 80386 con bus de 16 bits compatible con el del 80286, y el 80486 que es una versión más rápida del 80386 que incluye dentro del mismo chip el coprocesador matemático y una memoria caché. El último elemento de este grupo de procesadores, el Pentium posee muchos elementos de los grandes computadores como el proceso paralelo, arquitectura RISC, etc. Un miembro interesante de esta familia de procesadores es el 80386SL que es una versión del 80386SX pensada para computadores portátiles y que, por tanto, tiene bajo consumo para prolongar la vida de las baterías. Una característica notable de este procesador es que dispone de un chip de apoyo denominado 82360SL que integra gran parte de los circuitos exteriores necesarios en una placa base de un computador personal. Otra característica importante de los chips SL es que disponen de un modo de funcionamiento en que, con el reloj desactivado, mantienen su estado sin apenas consumo y pueden volver a la tarea anterior sin perder datos. En la tabla 1.2 se muestra la evolución de los microprocesadores de esta familia.

Los procesadores 80386 y 80486 tienen registros de 32 bits denominados $EAX$, $EBX$, $ECX$, $EDX$, $ESP$, $EBP$, $ESI$, $EDI$, $EIP$ y $EFLAGS$ cuya parte baja corresponde a los registros habituales del 8086. La estructura de estas máquinas puede verse en la figura 1.19. Otra característica importante de los procesadores 80386 y 80486 es la posibilidad de funcionamiento en tres modos diferentes: el modo real, compatible totalmente con el 8086, el modo

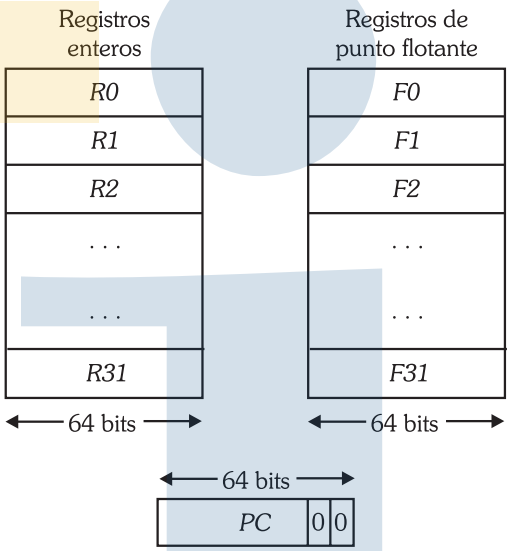


**Fig. 1.20**. Estructura de registros de los procesadores con arquitectura Alpha.

protegido, en que tiene una gran capacidad de direccionamiento y realiza muchas funciones para facilitar el trabajo multitarea y, por último, el modo virtual, en que es compatible con el 8086 pero trabajando en multitarea de forma que cada proceso ve un 8086 para él solo. Este modo en realidad es un submodo del modo protegido de forma que, estando un proceso en modo virtual, puede funcionar de forma compatible con el 8086. El modo virtual se caracteriza porque las funciones de E/S, que trabajando en multitarea pueden ocasionar problemas de interbloqueos, provocan llamadas al sistema operativo con lo que éste las puede controlar para que esos bloqueos no se produzcan.

## 1.13.8. Arquitectura Alpha

La arquitectura Alpha es el último diseño de DEC cuya primera realidad fue el procesador 21064 (Sites, 1992) al que posteriormente siguieron los procesadores 21164, 21264 y 21364. Su característica principal es el manejo de registros de 64 bits. Este procesador se basa en la filosofía RISC con procesamiento segmentado y está diseñado para poder funcionar en paralelo con otros procesadores iguales. El procesador está pensado para trabajar con datos de 64 bits (*quadword* o **cuádruple palabra**) aunque también puede trabajar con datos de 32 bits (*longword* o **doble palabra**), los datos más pequeños *word*, (**palabra**, 16 bits) y **byte** (8 bits) sólo son manejados por instrucciones especiales. También opera de forma nativa con varios formatos de punto flotante. Esta arquitectura pertenece al grupo de máquinas con **registros de uso general con arquitectura registro-registro** o de **carga-almacenamiento**. Este último hecho hace que la mayoría de las instrucciones sólo opere sobre registros, dejando los accesos a memoria para instrucciones específicas. Los datos no tienen restricciones de alineación pero el procesador trabaja de forma más eficiente si se respeta la alineación natural. No obstante, las instrucciones deben estar alineadas a doble palabra porque todas las instrucciones tienen esa longitud. Esa es la razón de que los dos últimos bits del $PC$ sean 0. La estructura de registros de esta máquina se muestra en la figura 1.20. Es interesante resaltar que los registros $R31$ y $F31$, si se utilizan

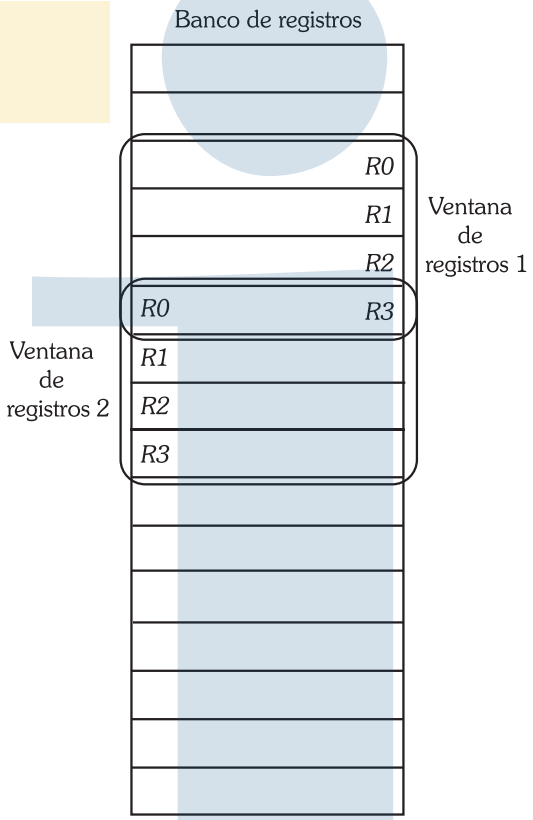


**Fig. 1.21**. Concepto de ventana solapada de registros.

como registros fuente de transferencias, adoptan el valor 0. Estos registros no deben usarse como operandos de destino. Otra observación interesante que se desprende de la figura 1.20 es que el registro de estado del procesador, con los bits señalizadores, y el apuntador de pila no están accesibles al programador, sin embargo, los sistemas operativos más usuales utilizan como apuntador de pila el registro entero $R30$.

Una innovación de esta arquitectura es la incorporación del denominado código PAL (*Privileged Architecture Library*) que es un conjunto de funciones al que se puede llamar mediante una instrucción específica (CALL_PAL) y que depende del sistema operativo utilizado. Esta biblioteca de funciones reside en una memoria de sólo lectura.

### 1.13.9. Arquitectura SPARC

La arquitectura SPARC (*Scalable Processor Architecture*, arquitectura escalable de procesadores), es ya un ejemplo clásico de **arquitectura RISC** (computadores con conjunto reducido de instrucciones). SPARC es un conjunto de especificaciones que determinan una arquitectura, no es un procesador concreto. Por principio, esta arquitectura es escalable, de forma que pueden definirse versiones sucesivas de la misma con características superiores, siempre salvaguardando la compatibilidad con los programas escritos para las versiones anteriores. Nosotros estudiaremos la versión 9, que es la primera con registros de 64 bits (Weaver & Germond, 1994). En lo referido a los registros enteros de usuario (registros $r$), esta arquitectura provee 8 registros

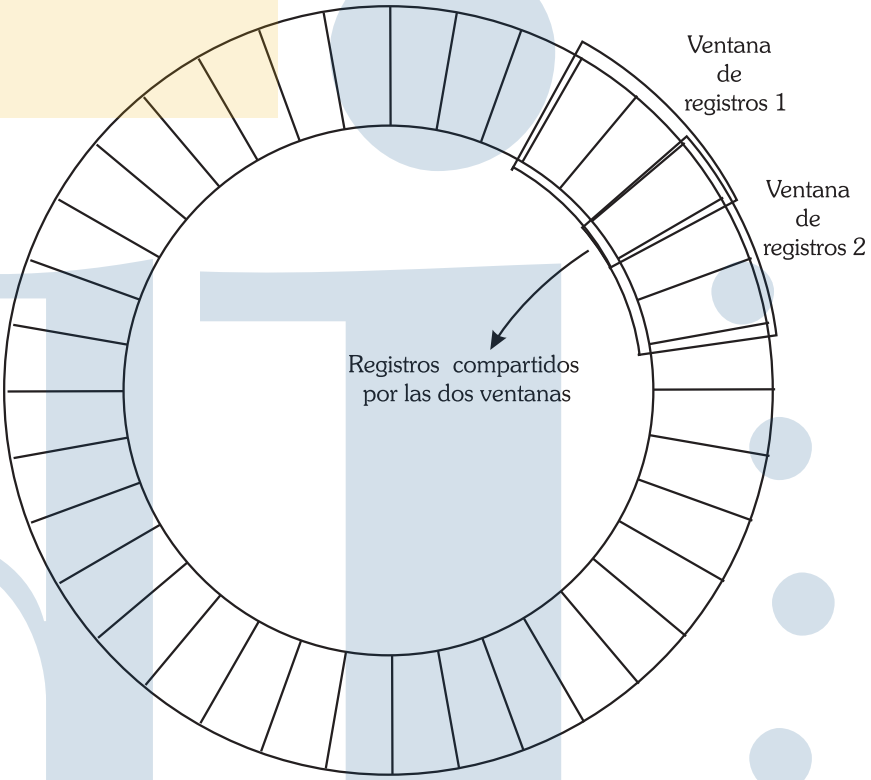


**Fig. 1.22**. Estructura cíclica de las ventanas de registros.

globales, 8 registros globales alternativos, y $N$ grupos de 16 registros, donde $N$ es un número comprendido entre 3 y 32, dependiendo de la implementación. Por ello, el número total de registros $r$ puede oscilar entre 64 y 528. Esta organización de registros está pensada para utilizar ventanas de registros, de forma que cada procedimiento sólo tenga acceso a una de esas ventanas (ver figura 1.21 para aclarar el concepto de ventana de registros). Todas las ventanas comparten los registros globales. Cada ventana tiene un grupo de registros en común con la siguiente, estos registros sirven para pasar parámetros de un procedimiento a otro. El banco de registros tiene una estructura circular (figura 1.22), de forma que, cuando todas las ventanas queden ocupadas, es necesario desalojar la ventana que corresponda al procedimiento más antiguo; posteriormente esa ventana no se restaurará hasta que sea necesaria su utilización por haber retornado todos los procedimientos posteriores. Más detalles sobre las ventanas de registros y los procesadores RISC en general, se estudiarán en el apartado 4.6.

En la figura 1.23 se detalla la organización de las ventanas de registros enteros en esta arquitectura. Algunos de los registros tienen propósitos especiales: el registro $r0 = g0$ (global 0) contiene siempre 0 y se utiliza para inicializar a 0 otros registros, o posiciones de memoria, y para utilizarlos como destino ficticio, por ejemplo, en instrucciones que pretendan cambiar sólo los señalizadores (ver más abajo); por otra parte, el registro $r14 = o6$ se utiliza como apuntador de pila, por lo que también se denomina $sp$; cuando se efectúa una llamada a un procedimiento, el registro $r30 = i6$, que es el $o6$ del procedimiento anterior, es el apuntador de trama de pila, $fp$; por último, la instrucción de llamada a procedimientos (CALL) deposita en $r15 = o7$ su propia dirección, esta información se utiliza para efectuar el retorno del procedimiento. Los conceptos

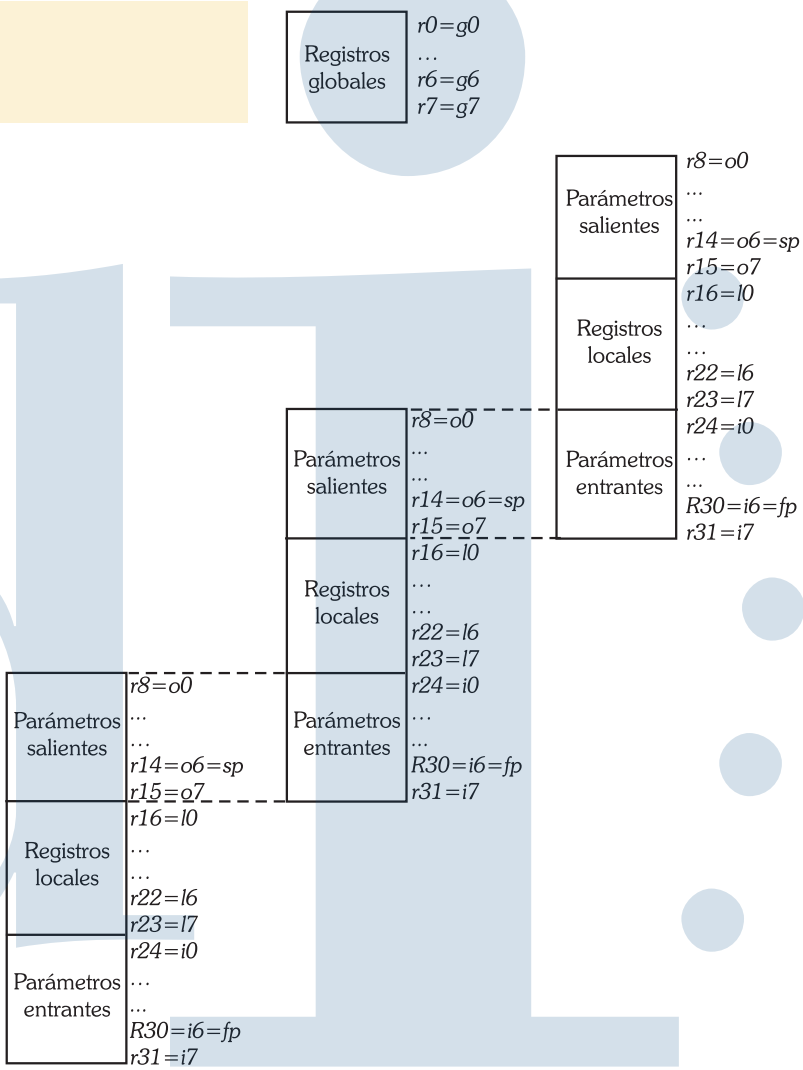


**Fig. 1.23**. Organización de las ventanas de registros en la arquitectura SPARC.

sobre el paso de parámetros mediante ventanas de registros se revisarán en el apartado 4.6.

Esta arquitectura también dispone de 64 registros para números en punto flotante (registros $f$), que son globales a todas las ventanas. Los 32 primeros (de $f0$ a $f31$) pueden almacenar números en punto flotante de simple precisión (32 bits), pero todos pueden agruparse entre ellos para almacenar números en doble precisión (64 bits) o en cuádruple (128 bits), aunque debe respetarse la alineación del número de registro (par para doble precisión y múltiplo de 4 para cuádruple precisión).

Las máquinas SPARC disponen también de diversos registros de control, entre ellos destacan el contador de programa ($PC$), el contador de programa siguiente ($nPC$), que se utiliza para las instrucciones de bifurcación, y el registro de códigos de condición ($CCR$) que contiene los señalizadores habituales ($N$, $Z$, $V$ y $C$), pero en dos grupos: $icc$ (*integer condition codes*), para operaciones de 32 bits y $xcc$ (*extended integer condition codes*), para operaciones de 64 bits, dependiendo del tipo de instrucción se analizarán uno u otro grupo de bits. Para las ope-

raciones de punto flotante existen 4 registros de condición llamados $fcc0$, $fcc1$, $fcc2$ y $fcc3$; cuando se comparan números en punto flotante hay que indicar cuál de ellos se va a utilizar.

Los procesadores con esta arquitectura manejan enteros de 8, 16, 32 y 64 bits, en la nomenclatura SPARC, estos tipo de datos se llaman, respectivamente, **byte**, **semipalabra** (*halfword*), **palabra** (*word* o *singleword*) y **doble palabra** (o **palabra extendida**) (*doubleword*). Como se mencionó anteriormente, también pueden manejar diversos tipos de punto flotante (32, 64 y 128 bits).

Una característica importante de la última versión de la arquitectura SPARC (SPARC V9) es que admite que los datos que ocupen más de un byte se puedan poner en cualquier orden: de mayor a menor peso (*big-endian*) o al revés (*little-endian*). Esto es válido sólo para datos y depende de la situación de un bit del registro de estado PSTATE. Los procesadores que permiten compatibilizar ambos tipos de funcionamiento se dice son *bi-endian*.

Es preciso señalar que, en la arquitectura SPARC, se exige la alineación de las direcciones de memoria de acuerdo al tamaño de los datos a que se accede. Esto afecta particularmente a las instrucciones, que tienen todas 32 bits, por lo que deben residir siempre en direcciones múltiplos de 4.

## Bibliografía y referencias

COHEN, D. 1981. On Holy Wars and a Plea for Peace. *IEEE Computer*, **14**(10), 48–54.

HENNESSY, J.L., & PATTERSON, D.A. 2003. *Computer Architecture. A Quantitative Approach*. 3 edn. Morgan Kaufmann Publishers.

SITES, R.L. 1992. *Alpha Architecture Reference Manual*. Digital Press.

STALLINGS, W. 2006. *Computer Organization and Arquitecture: Designing for Performance*. 7 edn. Pearson-Prentice-Hall. Existe traducción al castellano: Organización y arquitectura de computadores, 7ª edición, Pearson Prentice-Hall, 2006.

TANENBAUM, A.S. 2006. *Structured Computer Organization*. 5 edn. Prentice-Hall International. Existe traducción al castellano de la edición anterior: Organización de computadores: un enfoque estructurado, 4ª edición, Prentice-Hall Hispanoamericana, 2000.

TURING, A.M. 1936. On Computable Numbers with an Aplication to the Entscheidungsproblem. *Pages 230–265 of: Proceedings of the London Mathematical Society*, vol. 42.

WEAVER, D.L., & GERMOND, T. (EDITORES). 1994. *The SPARC Architecture Manual, version 9*. Prentice-Hall International.

# CUESTIONES Y PROBLEMAS

**1.1** Repetir el programa del ejemplo 1.1 para sumar números en una máquina de Turing transformando en blanco el 1 de la derecha en vez del de la izquierda.

**1.2** Repetir el programa del ejemplo 1.1 para sumar números en una máquina de Turing suponiendo que la cabeza está a la derecha de ambos datos.

**1.3** Repetir el programa del ejemplo 1.1 para sumar números en una máquina de Turing con los dos cambios propuestos en los problemas anteriores.

**1.4** Escribir un programa para una máquina de Turing que, dado un número natural, dé como resultado la parte entera de su mitad.

**1.5** Sean dos máquinas con las siguientes características:

Máquina A:

- Frecuencia de reloj: 30 MHz.
- Número medio de ciclos de reloj por instrucción: 2.

Máquina B:

- Frecuencia de reloj: 100 MHz.
- Número medio de ciclos de reloj por instrucción: 5.

Sabiendo que cada instrucción de la máquina B equivale, en promedio, a 1.5 instrucciones de la máquina A, calcular:

a) ¿Qué máquina es más rápida para los programas en que son válidos los parámetros anteriores? ¿En qué proporción?

b) Si se prescindiera de la tecnología empleada en los circuitos: ¿Qué máquina sería más rápida? ¿En qué proporción?

**1.6** Repetir los programas de las figuras 1.8 y 1.9 para un procesador que tenga instrucciones con diferentes números de operandos.