Capítulo 1

INTRODUCCIÓN

1.1 LA COMPUTADORA COMO ENTE PROGRAMABLE

¿Para qué sirve una computadora personal? Bueno, depende, si tiene gabinete de escritorio, horizontal, puede usarse para colocar algunos adornos sobre él, en el frente se pueden pegar notas autoadheribles con las cosas urgentes por hacer; si el gabinete es de torre, en cambio, puede usarse como soporte para libros o como medio para dividir la superficie de la mesa, delimitando espacios de trabajo diferentes, si se coloca debajo del escritorio puede eventualmente ser usada para levantar los pies. En efecto, los usos de la computadora personal son muchos, eso sin considerar las *lap top*, cada día más usuales, cuyas características las convierten en una excelente mesa portátil.

Ciertamente los descritos serían los únicos usos de la computadora personal de contar sólo con los componentes físicos de ella, el *hardware*. Pero claro, todos sabemos que una computadora personal sirve para miles de cosas: conectarse a Internet, consultar correo electrónico y redes sociales, navegar, jugar, escuchar música y ver películas, procesar texto u hojas de cálculo, hacer compras en línea, en fin. Hoy día no hay prácticamente ninguna actividad humana en el mundo civilizado en la no estén involucradas las computadoras.

La versatilidad de nuestras computadoras actuales proviene del hecho de que son entes programables, saben hacer muchas cosas diferentes porque podemos indicarles cómo deben hacerse esas cosas. Una

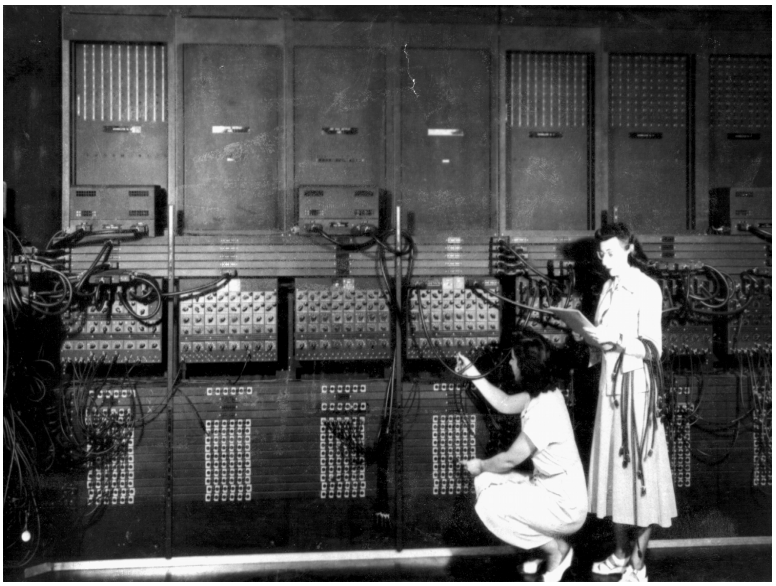

Figura 1.1. Dos programadoras de ENIAC haciendo su labor.

computadora es útil porque ejecuta *software*, programas diferentes para diferentes actividades. Muchas de nuestras comodidades actuales se deben a la sinergia entre el hardware y el software. Seguramente el lector está familiarizado con la programación de computadoras usando lo que se denomina *lenguaje de alto nivel*, como suele hacerse en la actualidad en todo el mundo.

Programar una computadora no siempre ha sido así de sencillo. La primera computadora electrónica de propósito general en la historia fue la famosa ENIAC (*Electronic Numerical Integrator And Computer*) y programarla era una ardua labor. Realmente había que reconfigurar el hardware, desconectando cables y reconectándolos en otros lugares (Fig. 1.1).

Arquitectura de von Neumann

Conscientes de que esta labor era lenta, delicada y propensa el error, Presper Eckert, John Mauchly, Herman Goldstine y John von Neumann discutieron, acerca de una mejor manera de proveer a una computadora del programa a ejecutar. Concluyeron que el programa debía poder almacenarse en la *memoria* de la máquina, para de allí ser leído e interpretado por la entidad encargada de la ejecución, llamada *unidad de control*. Las instrucciones serían entonces ejecutadas, con datos tomados de la misma memoria, por la *unidad aritmético-lógica* (ALU por sus siglas en inglés) de la máquina. Tanto los datos como el programa debían proveerse a la computadora desde el exterior y los resultados, finalmente debían ser también enviados al exterior, así que se requerían

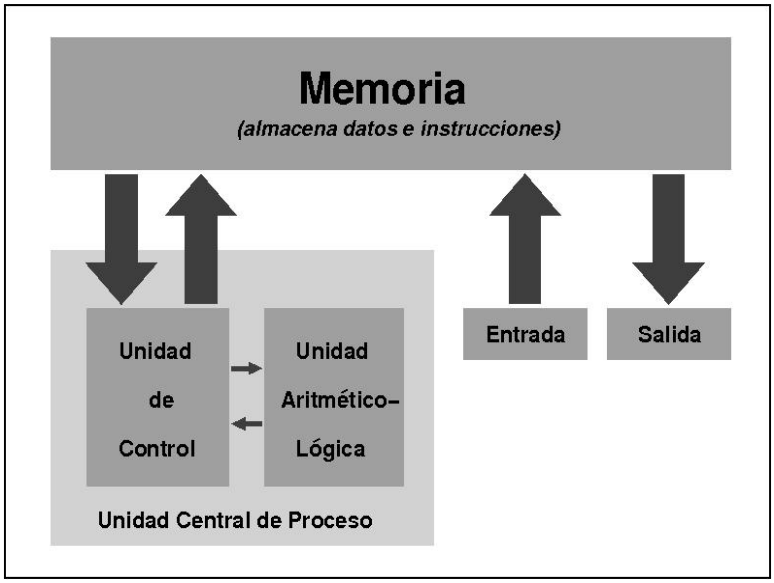

Figura 1.2. Esquema general de la arquitectura de von Neumann.

también de dispositivos de *entrada/salida*. La idea era usar esta alternativa en la siguiente computadora, la sucesora de ENIAC, a la que se denominó EDVAC (*Electronic Discrete Variable Automatic Computer*). Así, von Neumann escribió estos conceptos en un reporte que vió la luz en junio de 1945, llamado *First Draft of a Report on the EDVAC*. En el reporte se describe, esencialmente, lo que hoy llamamos *Arquitectura de von Neumann*, o bien *arquitectura de programa guardado*. En la figura 1.2 se muestra esquemáticamente el concepto.

1.2 BINARIO

A lo largo del tiempo la labor de programación se ha echo cada vez más simple gracias la construcción de toda una infraestructura para ello. En realidad nuestras computadoras no entienden lo que les decimos en C, en Java o en Scheme, las instrucciones que nuestras unidades centrales de proceso pueden interpretar y ejecutar son mucho más simples que las de un lenguaje de alto nivel y además deben ser dadas exclusivamente en el formato que la computadora puede almacenar y decodificar: en binario.

$\{0, 1\}$ **es suficiente**

El lector probablemente haya escuchado antes que "las computadoras trabajan en binario". En efecto, los circuitos electrónicos con los que nuestras computadoras se construyen sólo son capaces de distinguir entre dos niveles de voltaje (0 y 5 volts, generalmente), a los que

podríamos decirles A y B, Juan y Martha, o.. 0 y 1. Esta última es, por mucho, la mejor opción, porque 0 y 1 son los dígitos posibles en un sistema numérico posicional base 2, así que podemos representar cualquier número y por tanto hacer operaciones aritméticas. También podríamos decirles *verdadero* y *falso* y usarlos como los valores de verdad que utilizábamos en lógica proposicional, esto también nos es útil, porque entonces podemos construir funciones lógicas como la conjunción (Y), la disyunción (O), la implicación (Si.. entonces), y en general todas las que se expresan en tablas de verdad. A estas les llamaremos *funciones de conmutación*, las revisaremos más adelante y esencialmente es lo que, junto con las operaciones aritméticas, se lleva a cabo en la unidad aritmético-lógica ya mencionada. En síntesis, una computadora puede hacer todo lo que se requiere de ella trabajando en binario, con dos valores bastan.

$\{0,1\}$ es más confiable

Sin embargo no siempre las computadoras han trabajado en binario, ENIAC era decimal. Pero desde 1962 ya no se han hecho más (aparentemente la última fue la UNIVAC III, hecha por la compañía que originalmente formaron Eckert y Mauchly, los diseñadores de ENIAC). Para explicar esto es conveniente un ejemplo: imagine el lector dos dianas de tiro al blanco, ambas circulares y del mismo tamaño; supóngase que una de ellas, a la que llamaremos A, está dividida en 10 bandas concéntricas y la otra (B) en sólo dos. Si ahora se pone a una persona con un dardo, se le pide que diga en voz alta a cuál banda pretende darle y se le indica que dispare, ¿con cuál de las dianas tiene mayor probabilidad de atinar? Bueno, ciertamente en la diana de sólo dos bandas, esta tolera un mayor rango de error, la precisión del tirador puede ser menor que en el caso de la de 10. De igual forma nuestros dispositivos electrónicos toleran un mayor rango de variación en voltajes, intensidades o cargas sí sólo se consideran dos valores, son más confiables si trabajan en binario.

Así que el binario no sólo resulta ser suficiente, sino también conveniente.

Podemos pensar entonces que en la memoria de nuestras computadoras se almacenan datos e instrucciones en binario, y esas cadenas de ceros y unos se transfieren al procesador central[1] para ser interpretadas como instrucciones o para ser considerados como los operandos con los que se deben hacer las operaciones aritméticas y lógicas. En nuestras computadoras actuales los datos e instrucciones se almacenan

[1] Al procesador central le llamaremos también, indistintamente, procesador o CPU (por las siglas en inglés de *Central Processing Unit*.

temporalmente dentro del procesador en lo que se denominan *registros*. Cuando se dice que un procesador es de 32 o de 64 bits esta nomenclatura se refiere al tamaño de almacenamiento que tienen los registros en los que se guardan los operandos de su ALU.

1.3 LENGUAJES Y NIVELES

Lenguaje de máquina

Cuando hacemos un programa en lenguaje C, éste tiene que recorrer un camino para convertirse en algo que la unidad central de proceso pueda ejecutar. El programa debe ser traducido a instrucciones de *lenguaje de máquina*. Estas, por lo antes dicho, pueden pensarse como secuencias de ceros y unos. Una instrucción podría ser algo como:

`10100101010001001010011101O`

A lo mejor los primeros cinco bits (`10100` = 20) indican que hay que hacer la operación número 20 del catálogo de operaciones de la ALU, por ejemplo una suma. Los siguientes seis (`101010` = 42) indican que uno de los operandos está almacenado en el registro 42 y que es allí donde debe guardarse el resultado de la suma. Los restantes 16 bits (`0010010100111010` = 9530) podrían representar el otro operando, el número 9530. Así que la operación podría escribirse cómo:

ADD R20, 9530

Donde se ha usado la palabra `ADD` para decir "suma".

Lenguaje ensamblador

Podríamos, de hecho, si tuviéramos la paciencia suficiente, traducir un programa en lenguaje de máquina almacenado en memoria usando palabras y códigos como los anteriores, a los que se les llama *mnemónicos*. Expresar un programa así es lo que se denomina programar en *lenguaje ensamblador*. Esta era un práctica usual hasta los años 70 y siguió siéndolo en el contexto de las computadoras personales hasta la del 80. Al programa que debe traducir de lenguaje ensamblador a lenguaje de máquina se le denomina, precisamente, *ensamblador*. La labor de traducción de un ensamblador es simple, las instrucciones en lenguaje ensamblador corresponden, una a una, con las de lenguaje de máquina.

Lenguajes de alto nivel

Para traducir de un lenguaje como C a lenguaje de máquina las tareas son muchas y bastante más complejas. Se requiere generar un equivalente semántico del programa en C en lenguaje de máquina. Cada instrucción en C corresponde a varias instrucciones de máquina, es por eso que se les denomina *lenguajes de alto nivel*, el término se refiere al nivel semántico de las instrucciones.

Compiladores e intérpretes

Los programas encargados de hacer traducciones de lenguaje de alto nivel a lenguaje de máquina se dividen en dos categorías: los que generan un programa completo semánticamente equivalente al escrito

en lenguaje de alto nivel, llamados *compiladores* y los que traducen una por una las instrucciones de lenguaje de alto nivel, conforme leen traducen y alimentan con eso al procesador, a estos se les denomina *intérpretes*. Los compiladores además de traducir, también optimizan el código varias veces conforme el proceso de traducción se acerca más a lenguaje de maquina.

1.4 RELOJ

Operación en tiempo discreto

Nuestros dispositivos electrónicos funcionan de tal forma que, para tener un estado claro, bien definido, se debe esperar un cierto tiempo suministrando corriente para alcanzarlo. Así que una vez que se ha hecho lo necesario para representar un 1 en una celda de un registro, se debe esperar un tiempo para garantizar que, en efecto, allí se almacena un 1. Esto significa que los procesos de transferencia no son instantáneos, nuestras computadoras no pueden operar de manera, formalmente hablando, contínua. Operan en pasos discretos de tiempo. Por tanto se requiere que haya una entidad encargada de marcar el tiempo, de indicar cuando se puede operar y cuando hay que esperar, algo como el metrónomo que se utiliza en la enseñanza musical. A esto se le denomina *reloj*[2].

Oscilaciones

Todas nuestras computadoras actuales operan a un ritmo establecido por un reloj, cuanto más rápido oscile mayor será el número de operaciones aritmético-lógicas o de transferencia de registros que se pueden hacer por unidad de tiempo. La frecuencia del reloj está, cómo en todo lo que oscila, dada en términos de oscilaciones por segundo (Herz) aunque nuestras computadoras operan a velocidades tan sorprendentes que tenemos que usar múltiplos de ella como los gigaherz (GHz, miles de millones de oscilaciones por segundo). En la tabla 1.1 se listan algunos de los prefijos usuales.

A las oscilaciones del reloj que marca el ritmo de trabajo en la computadora se les llama *tics* o *clocks*.

1.5 LA ECUACIÓN DE DESEMPEÑO DE CPU

Considerando la frecuencia de operación de una CPU es posible calcular el tiempo efectivo de operación del procesador en la ejecución de un programa. Con T denotaremos el tiempo total de ejecución de un

[2]No se debe confundir el término con el de un reloj que indique la hora del día. El reloj al que nos referimos aquí es simplemente un marcador del ritmo, conceptualmente un *tic-tac*.

Prefijo	Valor	Prefijo	Valor
zepto	10^{21}	kilo	10^3 o 2^{10}
atto	10^{-18}	mega	10^6 o 2^{20}
femto	10^{-15}	giga	10^9 o 2^{30}
pico	10^{-12}	tera	10^{12} o 2^{40}
nano	10^{-9}	peta	10^{15} o 2^{50}
micro	10^{-6}	exa	10^{18} o 2^{60}
mili	10^{-3}	zeta	10^{21} o 2^{70}

Tabla 1.1. Prefijos comunes y su valor.

programa, con C el número de ciclos necesarios para ejecutarlo y con D la duración de cada ciclo. Así:

$$T = C \cdot D \tag{1.1}$$

Por ejemplo un programa que tarda en ejecutarse 7 millones de ciclos en una computadora cuyo ciclo dura 2 ns. tarda: $T = 7 \times 10^6 \cdot 2 \times 10^{-9} = 0.014\ s$.

Otra manera de medir el tiempo de CPU es, evidentemente:

$$T = \frac{C}{F} \tag{1.2}$$

donde $F = \frac{1}{D}$ es la frecuencia de operación del reloj, los gigaherz.

Un programa que tarda 17 millones de ciclos en una máquina a 2 GHz, tarda $T = (17 \times 10^6)/(2 \times 10^9) = 0.0085\ s$ de tiempo de CPU.

Pero es difícil saber cuantos ciclos en total toma la ejecución de un programa, así que podemos contar mejor el número de instrucciones (I). Si sabemos el tiempo promedio de ciclos por cada instrucción[3] (R).

$$R = \frac{C}{I} \tag{1.3}$$

Esta es una medida de que tan meritorio es un procesador, que tan baratas, en ciclos de reloj, son sus instrucciones.

De 1.3:

$$C = R \cdot I$$

[3]El término usado en inglés es *clocks per instruction* y frecuentemente se denota con CPI.

en 1.1:

$$T = R \cdot I \cdot D \tag{1.4}$$

o bien en 1.2:

$$T = \frac{R \cdot I}{F} \tag{1.5}$$

Hagamos un análisis de unidades de la expresión 1.4.

$$\frac{ciclos}{instrucción} \times \frac{instrucciones}{programa} \times \frac{segundos}{ciclo} = \frac{segundos}{programa}$$

Los tres factores de desempeño.

En síntesis el desempeño depende de:

1. Frecuencia del reloj.
2. Cantidad de ciclos por cada instrucción.
3. Contador de instrucciones por programa.

y depende en la misma medida de cada uno de los tres elementos, así que una mejora del 10% en alguno de los rubros mencionados genera una mejora del 10% en el desempeño total.

¿De qué dependen estos factores?

Frecuencia. Hardware, tecnología y organización del procesador (calor generado, densidad de componentes).

Ciclos por instrucción. Organización, arquitectura del conjunto de instrucciones.

Instrucciones por programa. Arquitectura del conjunto de instrucciones y eficiencia del compilador.

Algunas veces es útil contar el número de ciclos de reloj como sigue:

$$C = \sum_{i=1}^{n} R_i \cdot I_i$$

i es el índice de la instrucción en el catálogo de instrucciones de la máquina, I_i es el número de veces que la instrucción i se ejecuta en el programa y R_i es el número de ciclos de reloj que tarda en ejecutarse la instrucción i. De donde:

$$T = \left(\sum_{i=1}^{n} R_i \cdot I_i \right) \cdot D \tag{1.6}$$

Esto porque, de acuerdo a la definición de valor esperado de una variable aleatoria (en este caso en número de clocks por instrucción): $E(X) = \sum_{espacio\ muestral} xp(x)$. Como nuestro valor de R es, de hecho, un promedio:

$$R = \sum_{i=1}^{n} R_i \times \frac{I_i}{I}$$

Si sustituimos esto en 1.4 nos da 1.6.

1.6 TECNOLOGÍA Y DISEÑO

En 1975, la desaparecida Digital Equipment Corporation[4], introdujo al mercado su modelo PDP-11/70. Ésta costaba $72,650 dólares, venía con 256KB de memoria RAM (expandibles a 2MB), consumía 6000 Watts, podía hacer unas 300,000 operaciones por segundo y ocupaba al menos, si no se tenían muchos periféricos, el mismo volumen que un refrigerador de 18 pies cúbicos. Hoy día (en 2011) por unos $1500 dólares uno puede adquirir una computadora laptop con 4GB de RAM, capaz de hacer más de una decena de millar de millón de operaciones por segundo, que consume 95 Watts, que pesa menos de dos kilogramos y cabe en un portafolio.

Muchos factores han contribuido a salvar la enorme distancia entre estos dos equipos de cómputo. Primero el avance tecnológico, el hecho de que los transistores con los que se fabrican nuestras computadoras se han podido hacer cada vez más pequeños y de que se hayan podido poner millones de ellos en un sólo circuito integrado (chip). Luego el viraje del mercado, el hecho de que hoy día existen millones de usuarios de computadoras en todo el mundo y cada día se añaden más. También han contribuido, y de manera fundamental, las innovaciones en el diseño de las unidades centrales de proceso (a las que llamaremos también procesadores) siempre en la búsqueda de un mayor desempeño y de obtener la mejor relación costo-beneficio posible.

Cuando en 1971 Intel desarrollo el 4004, el primer microprocesador de la historia, se pudieron poner sus 2300 transistores en una superficie de $12mm^2$. La tecnología de la época permitía hacer una celda de memoria de 10 micras[5]. Hoy día un Intel Xeon de la serie 7500 tiene 2300 millones de transistores en ocho núcleos distribuidos en una superficie de $684mm^2$, la tecnología usada para fabricarlo permite construir celdas

[4]En 1998 DEC pasó a ser parte de *Compaq*, que a su vez fue comprada por Hewlett-Packard en 2002.

[5]También suele decirse micrómetros, la millonésima parte de un metro o la milésima de un milímetro

Figura 1.3. DEC PDP-11/70 de 1975 (no, no es una máquina expendedora de refrescos) y una laptop modelo 2011. La segunda es unas 30,000 veces mas poderosa y cuesta 2 centésimas partes de lo que costaba la primera.

de memoria de 45nm (nanómetros, o sea 45 millonésimas de milímetro). Durante 2011 seguramente veremos los primeros chips fabricados en 32 y 22nm y para 2015 veremos los de 11nm. Probablemente en 2021 veremos unos de 4nm.

La ley de Moore

Aparentemente el primero en notar y describir esta tendencia en 1965 fue Gordon E. Moore, co-fundador de Intel, por lo que se le ha llamado *la ley de Moore*. En la figura 1.4 se muestra una gráfica del logaritmo del número de transistores por chip contra el tiempo en los principales procesaores de Intel. El hecho de que los datos observados sean bien aproximados por una línea recta, dada la escala logarítmica en el eje de las abscisas, significa que el crecimiento del número de transistores es exponencial. En efecto Moore estimo que se duplicaba cada dos años, algunos más precisos suelen decir que cada 18 meses. El efecto es el mismo, mucha mayor densidad de transistores significa también mucho mayor poder de cómputo; aunque no precisamente en la misma proporción. Cuando se dobla el número de transistores se obtiene, en promedio, un 40% más de poder de cómputo, que de cualquier forma es bastante.

Otros componentes de cómputo siguen aproximadamente leyes similares: la capacidad de los discos duros, el número de pixeles en las cámaras digitales o la capacidad de las memorias flash, por ejemplo.

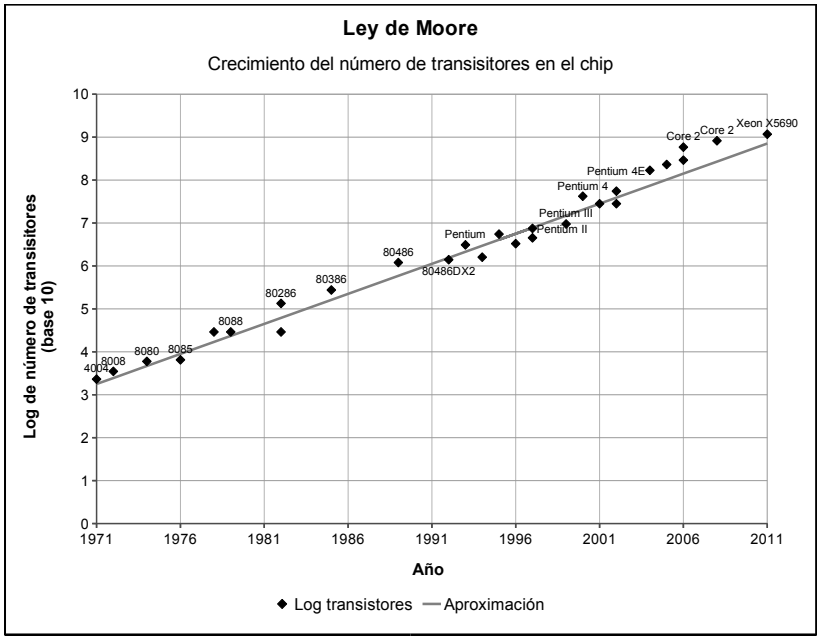

Figura 1.4. Evolución del número de transistores en el chip de un procesador.

La ley de Moore no es gratis, la inversión tecnológica necesaria para producir chips a densidades cada vez más altas también crece, sin embargo se ha mantenido. En buena medida por el principal factor que amortiza el costo de producción del equipo de cómputo: el consumidor.

Evolución del perfil de usuario

El usuario típico de un sistema de cómputo en la década de los 50 del siglo XX era un tipo de bata blanca con anteojos y un libro de física bajo el brazo, hoy día el usuario típico es cualquier persona. Las cosas comenzaron a cambiar, primero lentamente, cuando las computadoras se incorporaron al sector público y al privado y ya no sólo le pertenecían a la academia o la milicia. Pero el cambio radical ocurrió en los 80, la popularización de la computadora personal trajo consigo un enorme segmento de mercado. Por supuesto un segmento impensable si las aplicaciones de las computadoras permanecieran estáticas, la mayoría de las personas no suelen hacer cálculos de mecánica cuántica en la sala de su casa. Pero hoy casi todo mundo juega, oye música, se comunica o trabaja en su computadora personal usando programas de aplicación muy variados. Un Ferrari es caro porque el costo de producción tiene que amortizarse entre los 20 o 30 clientes que encargan uno al año, un modelo compacto de línea de las principales armadoras es mucho más barato porque el costo se amortiza con los miles de unidades que se

venden. Lo mismo ocurre con el equipo de cómputo: más usuarios, muchos más, significa, mucho menor precio al consumidor final.

Pero ese no fue el único cambio importante durante la década de los 80.

Conjuntos complejos de instrucciones

Desde el surgimiento de los primeros sistemas de cómputo y hasta la década de los 70 la memoria principal de los sistemas de cómputo era de núcleo magnético, un tipo de tecnología costoso y de acceso muy lento. Como los programas requieren estar en memoria para poder ser ejecutados, era necesario hacer programas breves; cuanto más cortos, mejor, para hacer el menor uso de memoria posible. Sin embargo el desarrollo de compiladores no había alcanzado los niveles de optimización que podemos apreciar hoy día, así que los programas buenos debían ser hechos en lenguaje ensamblador por programadores expertos. Para hacer un buen programa se requería conocer a la perfección la arquitectura de la máquina y su conjunto de instrucciones de lenguaje de máquina. Un programador así de especializado cobra mucho, así que los programas buenos eran caros. En respuesta a esto los diseñadores de hardware debían facilitarles la vida a los programadores en la medida de lo posible. Así que optaron por darle soporte en hardware a casi todas las necesidades imaginables de los programadores en ensamblador: si éste requería una instrucción para obtener la raíz cuadrada real de un número, se proporcionaba una instrucción de lenguaje de máquina que lo hacía y, claro está, se diseñaba el hardware necesario para ejecutarla en el procesador. Con el tiempo esto generó enormes conjuntos de instrucciones, el catálogo de lo que el procesador podía hacer en lenguaje de máquina era enorme y además contenía instrucciones muy complejas en su ejecución, extraer una raíz cuadrada no es trivial, al procesador le tomaba decenas de ciclos de ejecución.

Haciendo referencia a la ecuación 1.4, se pretende mejorar el desempeño disminuyendo el número de instrucciones necesarias para construir el programa. Dadas las circunstancias tecnológicas vigentes en los 70 y antes, sólo eso se podía hacer. Cuanto menor la longitud de programa mejor, tanto para el desempeño, como para el costo del software.

A lo largo de la década de los 70 las cosas ya eran diferentes, las memorias de semiconductores comenzaron a usarse más, eran más rápidas y más baratas, empezaron a surgir estándares de facto en la industria, como Unix y el lenguaje de programación C. Esto les concedió libertad a los diseñadores de hardware, el soporte para el programador de ensamblador ya no era fundamental, el hardware podía ser diferente si a fin de cuentas la cara que habría que presentar era la misma: Unix y

un compilador de C. Con este nuevo panorama algunos arquitectos de computadoras se cuestionaron seriamente la premisa de que un programa bueno es un programa corto. Algo que además era mas asequible gracias a las optimizaciones que los compiladores ya hacían en el código que generaban. A lo mejor el programa podía ser más largo siempre y cuando el hardware que lo ejecuta sea más rápido. Si ya no es necesario dar al programador en ensamblador todo lo imaginable, ¿qué es lo que se debe dar?, ¿qué instrucciones debe tener el lenguaje de máquina?, ¿cuáles son las esenciales?

RISC Vs. CISC

David Patterson en Berkeley y John Henessy en Stanford, cada uno por su cuenta comenzaron a hacer diseños nuevos, con conjuntos de instrucciones muy pequeños, mínimos, con instrucciones de tamaño fijo, formato fijo y con unas cuantas maneras diferentes de acceder a los datos, demasiado restrictivo para lo que el mundo estaba acostumbrado, pero funcionó y muy bien. La decisión acerca de qué instrucciones y cuales modos de acceder a los datos debían incluirse en el conjunto se tomó con base en extensos y detallados estudios estadísticos acerca de la frecuencia promedio con que eran usados en los programas. La premisa, que después retomaremos con detalle era: hacer mejor lo más frecuente. El resultado fue lo que se denominó RISC (*Reduced Instruction Set Computer*). Por supuesto luego de elegir este nombre, a los diseños con el paradigma previo se les llamó CISC (*Complex Instruction Set Computer*). En la ecuación de desempeño 1.4, el factor en el que se enfoca el diseño CISC es el conteo de instrucciones por programa. En el diseño RISC, en cambio, el factor a mejorar es el número de ciclos por instrucción.

Durante los 80 los diseños RISC se adueñaron del mercado de estaciones de trabajo, computadoras más robustas que una PC, pero menos que un mainframe o una minicomputadora (como la PDP con la que empezamos el capítulo). Luego superaron con creces el desempeño de estas últimas y de hecho las desplazaron ocupando su nicho. Las máquinas más poderosas de los 80 y buena parte de los 90 (sin mencionar supercomputadoras), fueron diseños RISC. Hoy día la mayoría de los procesadores actuales para equipo de cómputo poseen aún muchos de los conceptos RISC subyaciendo en una arquitectura mucho más compleja. Ciertamente la complejidad resurgió, pero de manera mucho más acotada.

Paralelismo a nivel de instrucción

El paradigma RISC hizo crecer el desempeño, aproximadamente un tercio más de lo que se hubiera obtenido sólo gracias a la tecnología de integración de circuitos, añadiendo hardware. La simplificación de las instrucciones y su número mínimo, su formato fijo, hizo posi-

ble, como veremos más adelante, procesar varias instrucciones en simultáneamente, lo que se denomina *paralelismo a nivel de instrucción* o ILP por las siglas en inglés de *Instruction Level Parallelism*.

La ley de Moore se ha mantenido hasta ahora y no se prevé que vaya a dejar de funcionar en un futuro cercano, sin embargo hacer crecer la densidad de nuestros chips trae consigo problemas serios. Miles de millones de transistores, todos trabajando al unísono, todos hacinados en una muy pequeña área. El calor producido por todo esto es tal que hoy en día no es posible ver el chip del procesador de una computadora, está enterrado en enormes placas de aluminio de formas caprichosas diseñadas para disipar calor y bajo varios ventiladores que ayudan en la tarea.

En el rubro del paralelismo a nivel de instrucción, sin embargo, la historia es diferente. Es difícil lograr mayor paralelismo del que se posee actualmente, si el nivel semántico al que se procesa es el de las instrucciones. El procesador a fin de cuentas sólo puede ver, a lo más, unas decenas de instrucciones. Dentro de ese limitado ámbito reacomoda, distribuye, retrasa o adelanta la ejecución de las instrucciones para lograr un máximo nivel de paralelización, para que, en la medida de lo posible, todas las partes del procesador estén siempre trabajando. Es por esto que la tendencia durante las últimas décadas ha sido asignar cada vez mayor responsabilidad al compilador en materia de desempeño. El compilador posee una visión mucho más general y puede, por tanto hacer optimizaciones que al procesador le serian imposibles, donde el procesador tendría que adivinar, por ejemplo, el compilador puede ocasionalmente tener la certeza de lo que ocurrirá.

Paralelismo a nivel de *thread*

Pero desde los primeros años del siglo XXI se ha estado trabajando en un frente diferente para lograr aun mayor desempeño, paralelizando, ya no sólo a nivel de instrucción, sino a un nivel semántico superior: ejecutando tareas completas en paralelo; a lo que se ha llamado *paralelismo a nivel de hilo de ejecución* (TLP, por las siglas en inglés de *Thread Level Parallelism*). Esto, como veremos más adelante implica varios problemas.