Arquitectura de Computadores

8. Arquitecturas RISC

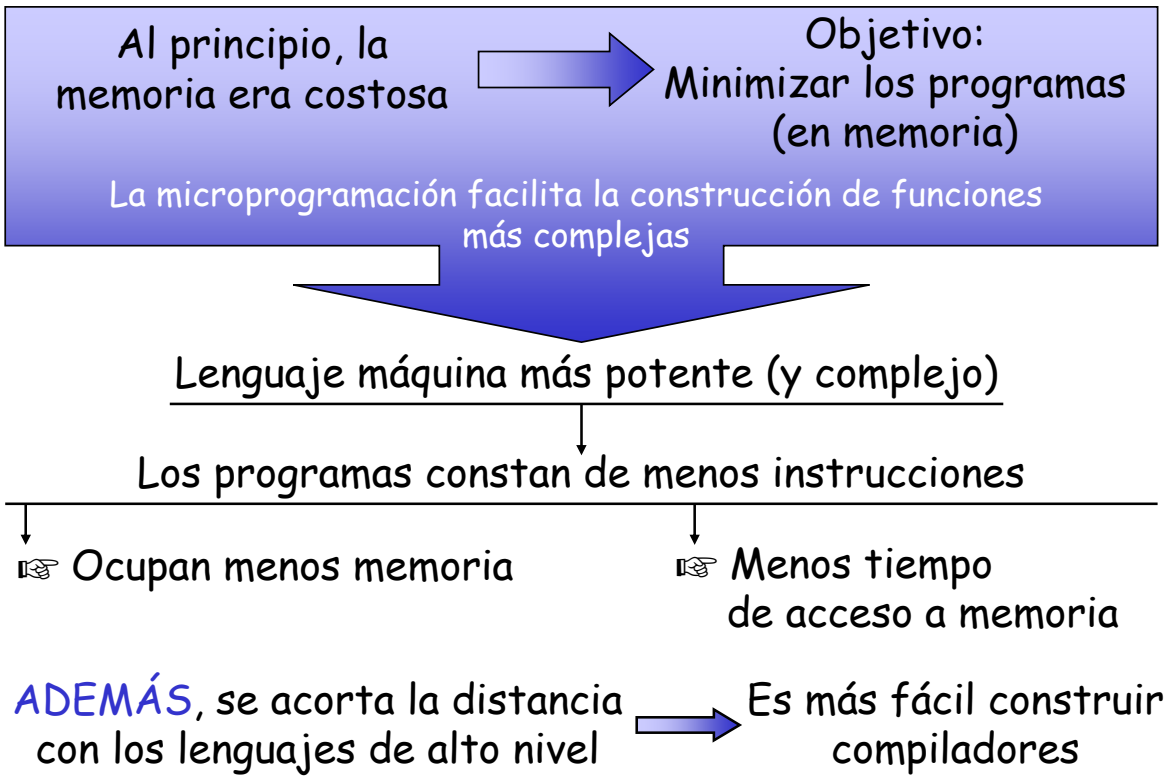
1. Evolución de los CISC
2. Análisis de los Programas
3. Principios y Características de las Arquitecturas RISC

La evolución de la arquitectura de computadores ha mostrado una tendencia desde sus principios hacia una complejidad creciente del lenguaje ofrecido. Sin embargo, en 1975, los arquitectos de computadores de IBM cuestionaron, con la construcción del 801, que la creciente complejidad fuera el camino más apropiado para conseguir una mejor relación coste/rendimiento. Ya que IBM no aireó demasiado esta nueva filosofía, no tuvo gran repercusión hasta que a principios de los años 80 Hennessy y Patterson (en Berkeley y Stanford) retomaron la idea y dieron lugar al movimiento **RISC** (*Reduced Instruction Set Computer*) en oposición a los tradicionales ordenadores con complejos juegos de instrucciones (**CISC**, *Complex Instruction Set Computer*).

Para entender los principios de las arquitecturas RISC, primero veremos la explicación de la complejidad que han ido adquiriendo las arquitecturas. Después **analizaremos cómo se realiza la ejecución de las instrucciones** en el procesador, considerando cada instrucción por separado y también mediante estudios estadísticos de los tipos de instrucciones que ejecutan los programas. Este análisis de la ejecución de los programas es lo que nos llevará a entender cómo se puede **mejorar la relación coste/rendimiento** ofreciendo juegos de instrucciones simples y sencillas, en lugar de incrementar la complejidad de las instrucciones (y el precio de los procesadores) según la idea tradicional.

Lo que aquí mostramos es una mera introducción a la arquitectura RISC, con la única intención de mostrar que el enfoque CISC no es la única posibilidad, y que la alternativa RISC es más que razonable. Para un estudio más detallado del análisis de la ejecución de las instrucciones y de los programas puede consultarse alguno de los textos especializados citados en la bibliografía, tales como el de Hennessy y Patterson o el de Kane.

Evolución de los CISC



Al principio de la era electrónica de los ordenadores, la memoria era un recurso muy costoso. Esto dio pie a la idea de que la mejor arquitectura de un ordenador era la que permitía minimizar los programas (en términos de memoria ocupada). Otros factores que también se utilizaban para medir la calidad de una arquitectura era el número de bits por instrucción y la cantidad de bits de instrucciones y de datos que se alimentaban durante la ejecución de un programa.

Debido a la introducción de la microprogramación (que permitió que los complejos circuitos lógicos de la unidad de control se reemplazaran por una memoria con un microprograma), se volvió mucho más barato sustituir funciones que antes se realizaban mediante una serie de instrucciones, por instrucciones más complejas que entendía y ejecutaba el microprograma. Al tener un programa menos instrucciones se conseguían dos cosas: menos espacio de memoria y menos instrucciones que alimentar desde la lenta memoria, luego se aumentaba la velocidad de ejecución. No obstante, a medida que se incorporaban más y más funciones al microcódigo, el tamaño del microprograma fue creciendo.

Otro argumento para implementar funciones complejas en microcódigo era que estas funciones acortaban la gran distancia semántica que había entre el lenguaje máquina y los lenguajes de alto nivel, ofreciendo un mejor soporte para los compiladores de estos lenguajes. Si la sentencia de un lenguaje de alto nivel se podía traducir a una única (o unas pocas) instrucciones máquina, la construcción de los compiladores sería mucho más sencilla.

...Evolución de los CISC

PERO:

- ✓ El microprograma crece (y se complica)
- ✓ Crecimiento por compatibilidad de familias
- ✓ Muchas instrucciones
  - - Muchos formatos
  - - Instrucciones de longitud variable

Este enriquecimiento del juego de instrucciones fue parejo con un incremento igualmente rápido de los modos de direccionamiento. (Por ejemplo, el MC68000 tenía 12 modos y en el MC68020 creció hasta 18). También, la necesidad de mantener la compatibilidad con los procesadores anteriores de la misma familia hizo que los constructores de ordenadores fueran incrementando los juegos de instrucciones más por compatibilidad que por motivos puramente técnicos.

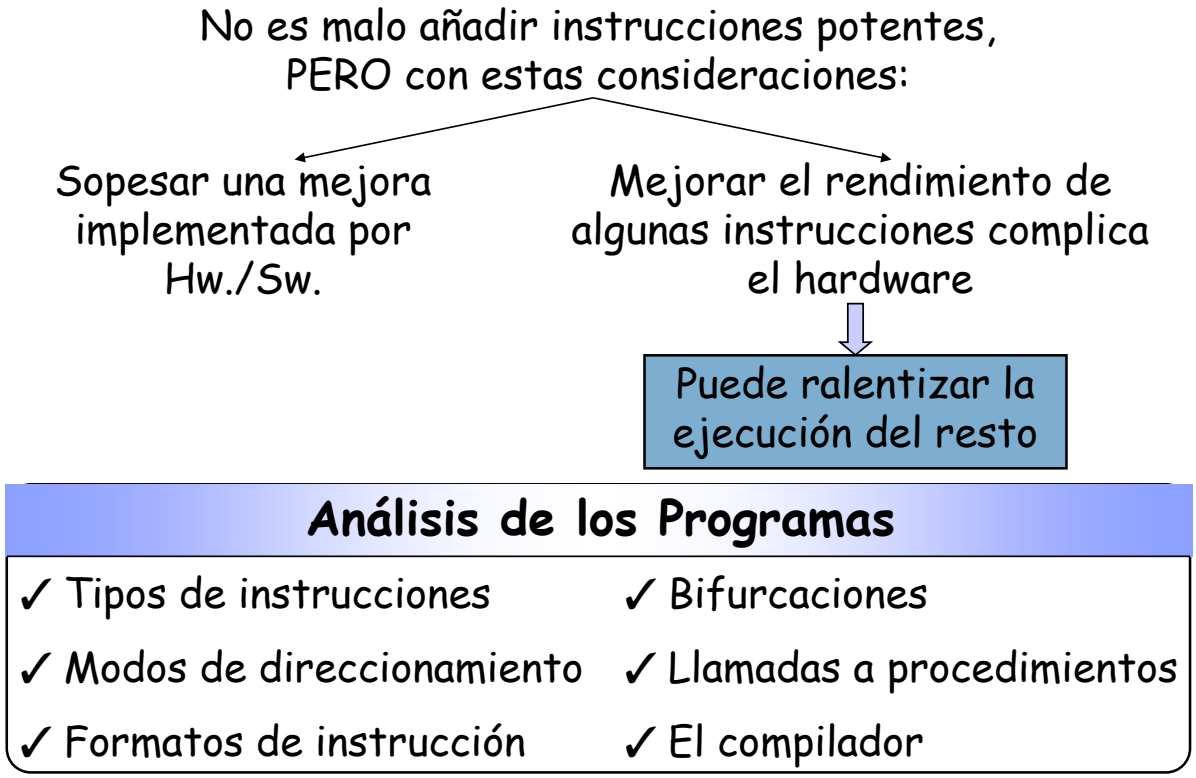
La búsqueda de un código más denso y con instrucciones más potentes hizo necesario utilizar formatos de instrucción con longitud variable, es decir, juegos de instrucciones en los que el código de operación y los campos que especifican los operandos tienen longitudes variables, aunque esto ocasionara que algunas instrucciones estuvieran ubicadas en direcciones de memoria que no fueran múltiplos de palabra.

Análisis de los Programas

Para un *benchmark* dado (programa de medición del rendimiento), el tiempo total empleado en su ejecución es **I x C x Tc** (nº de instrucciones x nº ciclos por instrucción x tiempo o duración de cada ciclo). El rendimiento se puede mejorar reduciendo cualquiera de estos tres factores

El propósito de las arquitecturas tradicionales CISC es minimizar esta expresión haciendo más pequeño el número de instrucciones a base de instrucciones más potentes y complejas. Los detractores de estas arquitecturas piensan que es preferible reducir el número de ciclos por instrucción mediante instrucciones más simples, pero claro, esto tiende a incrementar el número de instrucciones por programa.

...Análisis de los Programas



En principio, no tendría ningún inconveniente el que una arquitectura, partiendo de un juego de instrucciones sencillas lo fuera incrementando con instrucciones potentes y complejas, pero al diseñar la CPU y su juego de instrucciones se deben tener en cuenta estas dos consideraciones:

- Se debe sopesar la mejora del rendimiento obtenida cuando cierta funcionalidad se implementa directamente mediante hardware o cuando se obtiene a través del software.
- Al complicar el hardware para mejorar el rendimiento de algunas instrucciones puede ralentizarse la ejecución del resto.

Se debe analizar el comportamiento de los programas para saber qué genera mayor beneficio, minimizar el número de instrucciones o minimizar el número de ciclos por instrucción y la duración de los ciclos.

Diversos estudios han analizado las características del código producido por los compiladores de los lenguajes de alto nivel sobre arquitecturas CISC, para así poder determinar las características de las instrucciones máquina que realmente se utilizan y se ejecutan. Estos análisis sacan a la luz algunas observaciones sobre ciertos aspectos que merece la pena comentar:

- Tipos de Instrucciones (simples/complejas)
- Modos de direccionamiento
- Formatos de instrucción
- Bifurcaciones
- Llamadas a procedimientos
- El compilador

Veamos, en las siguientes diapositivas, qué tienen de particular estas observaciones.

Tipos de Instrucciones

TIPO DE INSTRUCCIÓN	FREC. DINÁMICA	COMPLEJAS
Movimiento/copia	32,85	
Bifurcación	15,58	
Decrementa y salta (DBcc)	8,37	8,37
Comparación/test	10,95	
Aritméticas/lógicas	16,31	
Suma, resta, AND, OR	11,47	
Multiplicación	0,58	0,58
División	0,13	0,13
Desplazamiento/rotación	3,25	
Extensión de signo	0,88	
Subrutinas	6,14	
Salto a subrutina	1,69	
Retorno	1,69	
Link/Unlink	2,76	2,76
Misceláneas	7,53	
Op. con bits	0,22	0,22
Borrar (*Clear*)	2,69	
Load and push ef. addr.	4,25	
Set condicional	0,37	
TOTAL	**100,00%**	**12,06%**

Distribución porcentual de instrucciones según su frecuencia dinámica

La tabla de la diapositiva muestra la distribución dinámica de instrucciones (las que realmente se ejecutan, no las que figuran en los programas) según un estudio realizado para el MC68000.

Algunas de estas operaciones, teniendo en cuenta su cometido, pueden considerarse complejas, tales como `DBcc`, `MUL`, `DIV`, `LINK`, `UNLINK` y las operaciones con bits, no obstante solamente suponen el 12% de las instrucciones ejecutadas.

Modos de Direccionamiento

DIRECCIONAMIENTO	OP. FUENTE	OP. DESTINO
Ninguno	35,10	17,28
Directo a registro	12,50	49,48
Indirecto a registro	30,71	25,44
(An)	4,08	1,50
(An)+, -(An)	4,16*	13,00*
Despl. + (An)	20,10	9,13
Despl. + (An) + (Xn)	2,37*	1,81*
Absoluto	0,54*	0,00*
Relativo al PC	0,42	
Despl. + (PC)	0,32	
Despl. + (PC) + (Xn)	0,10*	
Inmediato	7,77	
#xxx	6,57*	
Quick	1,20	
Implícito a pila	3,08*	4,93
Total	**90,12%**	**97,13%**
Total Modos Complejos*	**16,82%**	**19,74%**

Distribución porcentual de modos de direccionamiento según su frecuencia dinámica

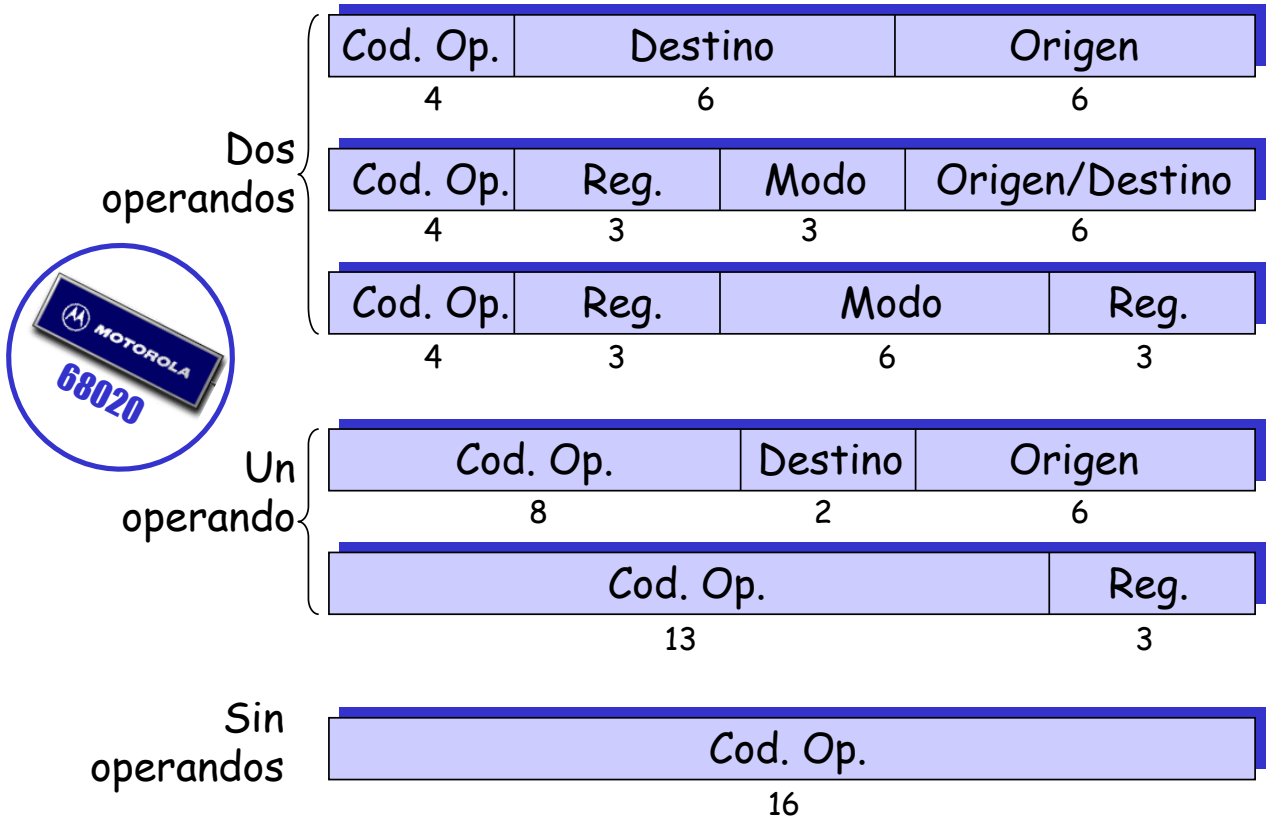
La complejidad de una instrucción también viene determinada por los modos de direccionamiento que puede utilizar para especificar los operandos.

En la diapositiva se muestra la frecuencia dinámica de los modos de direccionamiento utilizados en el MC68020 de Motorola. De los datos de esta tabla se puede observar que los modos complejos de direccionamiento (los marcados con "*") constituyen solamente el 16,82% para especificar un operando fuente, y el 19,74% para indicar el resultado de la operación.

(Obsérvese que las sumas de los porcentajes no dan el 100%. Esto se debe a que no se han contemplado las instrucciones sin ningún operando explícito).

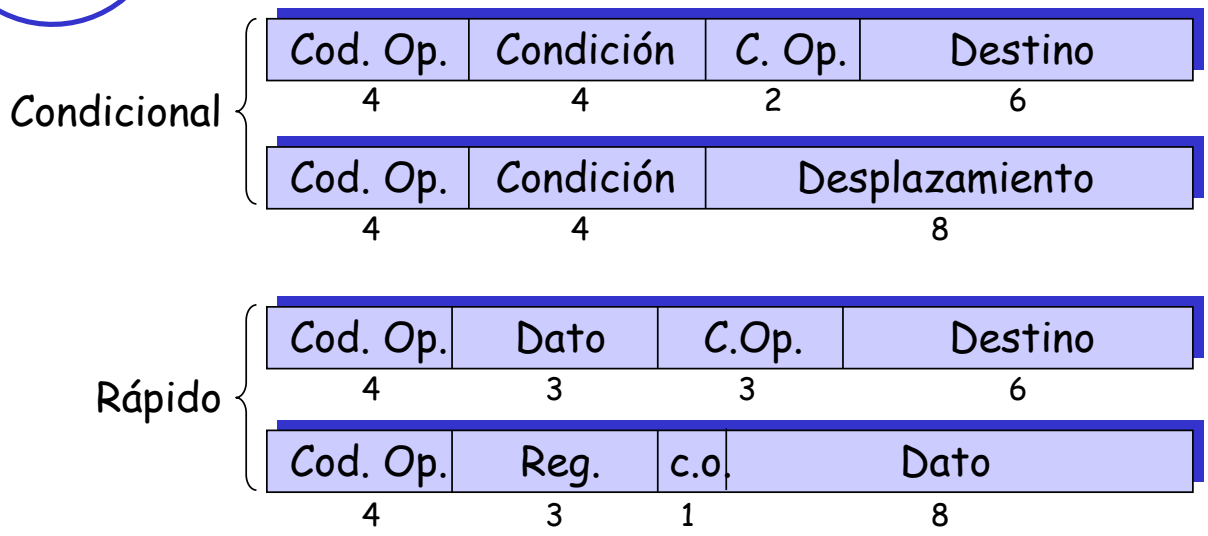
Un direccionamiento se puede entender como complejo cuando se necesita más de un ciclo de reloj para calcular la dirección del operando, por requerir sumas, restas, alguna operación, o la extracción de palabras adicionales.

Formatos de Instrucción



Aquí se muestran algunos de los formatos de instrucción del MC68020. Disponer de muchos formatos de instrucción significa que cada uno de los bits o campos que hay en una instrucción tienen diversas interpretaciones, dependiendo de los valores de otros campos, como por ejemplo del código de operación. Como consecuencia de esto, la decodificación de una instrucción puede requerir más tiempo, pues se requiere una lógica más compleja y, por supuesto, hardware adicional.

Más Formatos



Arquitectura de Computadores Arquitecturas RISC - 9

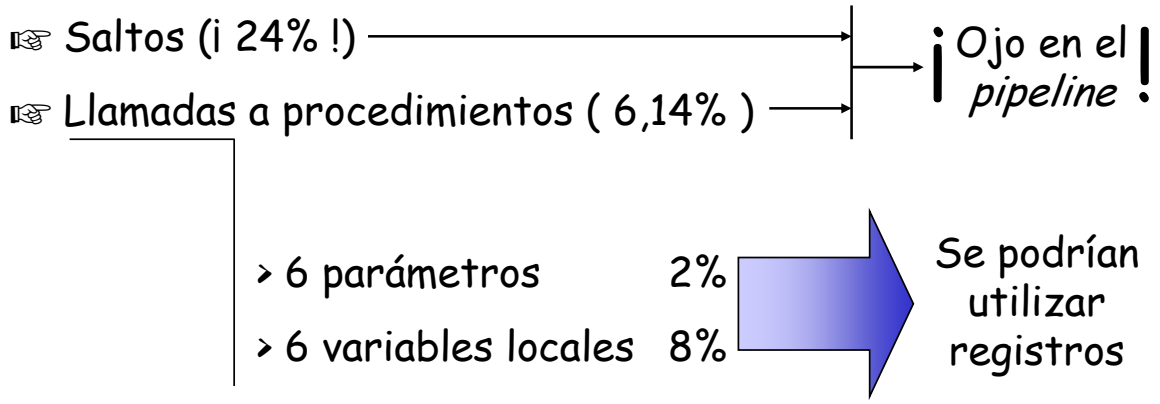
Flujo de Ejecución

TIPO DE INSTRUCCIÓN	FREC. DINÁMICA
Movimiento/copia	32,85
Bifurcación	15,58
Decrementa y salta (DBcc)	8,37
Comparación/test	10,95
Aritméticas/lógicas	16,31
Suma, resta, AND, OR	11,47
Multiplicación	0,58
División	0,13
Desplazamiento/rotación	3,25
Extensión de signo	0,88
Subrutinas	6,14
Salto a subrutina	1,69
Retorno	1,69
Link/Unlink	2,76
Misceláneas	7,53
Op. con bits	0,22
Borrar (*Clear*)	2,69
Load and push ef. addr.	4,25
Set condicional	0,37

En los procesadores actuales, la segmentación o *pipeline* es una de las mejores formas de acelerar el ritmo de ejecución de instrucciones. No obstante, las instrucciones que rompen el flujo secuencial de ejecución afectan muy negativamente al funcionamiento óptimo del *pipeline*. Por esto, vamos a analizar la importancia de este tipo de instrucciones por su porcentaje de uso respecto al total de las instrucciones que se ejecutan normalmente.

En la tabla superior se puede ver la frecuencia dinámica de ejecución de los diferentes tipos de instrucciones. Pasemos a la página siguiente para comentar la frecuencia con que se ejecutan las instrucciones que rompen el flujo de control secuencial, como son las de bifurcación y las de "decrementa y salta" (DBcc).

...Flujo de Ejecución



Hay dos aspectos relacionados con el control del flujo de ejecución:

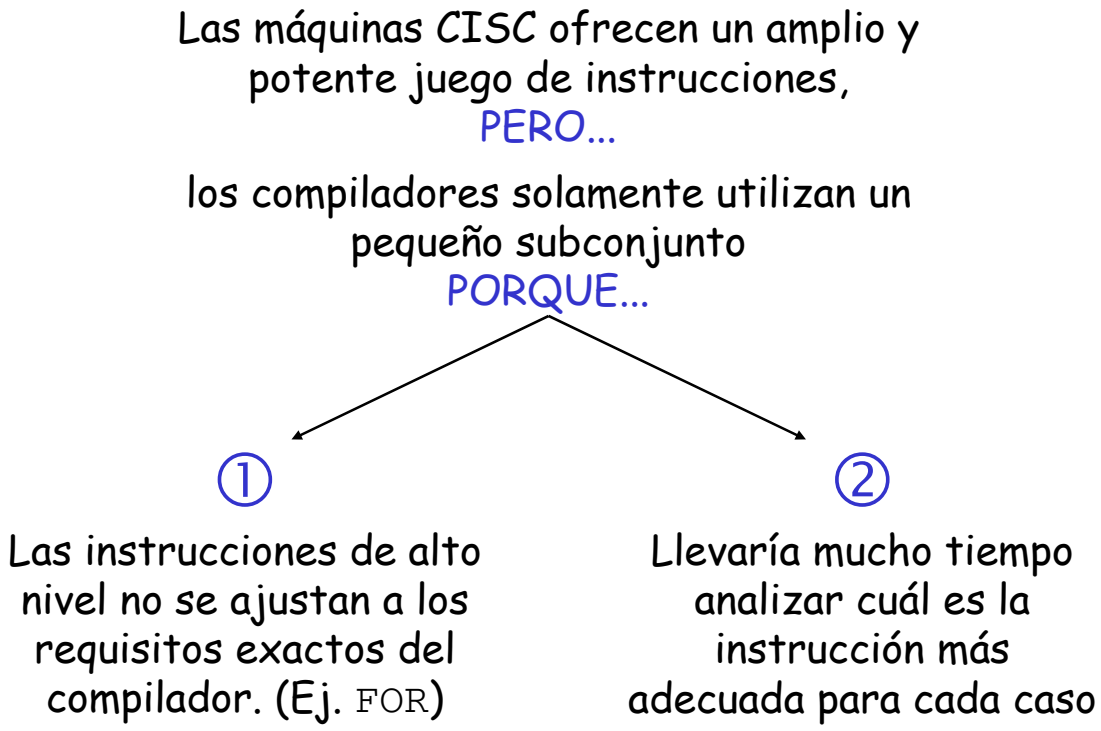
- Saltos para bifurcaciones y control de bucles
- Llamadas a procedimientos

Saltos. Como podemos ver en la tabla de la distribución dinámica, las instrucciones de salto suponen un 24% de las instrucciones ejecutadas (junto con las de tipo `DBcc`), el segundo tipo de instrucciones más ejecutadas. Esto quiere decir que en las máquinas segmentadas (con *pipeline*) se debe prestar especial atención al tratamiento de las instrucciones de salto, pues como ya sabemos pueden producir una detención o vaciado del *pipeline*.

Llamadas a procedimientos. En un estudio realizado por Tanenbaum en 1978 se observó que solamente en el 2% de las llamadas a procedimientos se pasaban más de seis parámetros, y que solamente el 8% de los procedimientos tenían más de seis variables locales escalares. Esto quiere decir que el tamaño del registro de activación o trama de pila utilizada en las llamadas a procedimientos es bastante pequeño, luego se podría pensar en utilizar registros generales, en lugar de la pila, para pasar los parámetros y para contener a las variables locales de las rutinas.

(La “trama de pila” o “registro de activación” es la porción de la pila que se suele utilizar en la llamada a un procedimiento para pasar los parámetros, la dirección de retorno y la reserva de espacio para las variables locales del procedimiento llamado).

El Compilador

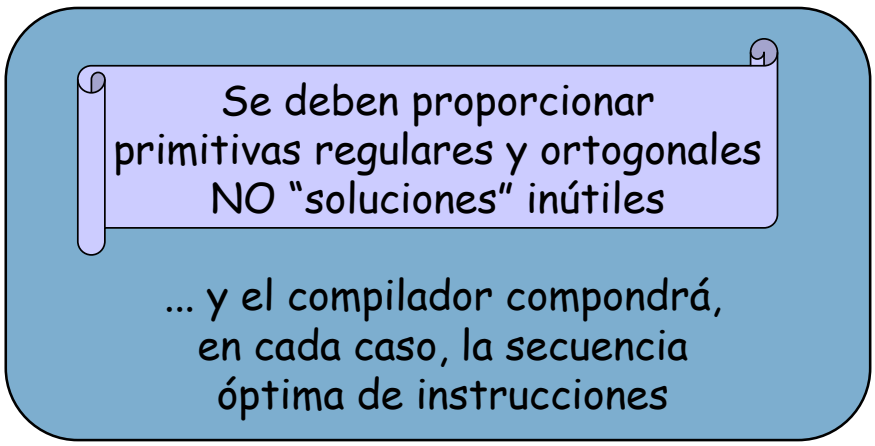


Una de las principales motivaciones de las arquitecturas CISC es proporcionar el más variado tipo de instrucciones, incluso de alto nivel, para facilitar la construcción de compiladores para los lenguajes de alto nivel.

Los estudios estadísticos también han observado que los compiladores solamente **utilizan un pequeño subconjunto** de todas las instrucciones ofrecidas por el procesador, y esto es por varias razones. Una es que aunque muchas instrucciones se parecen a sentencias de alto nivel, **su semántica es ligeramente diferente**, por lo que no pueden utilizarse en todos los casos, o solamente para algún lenguaje concreto. Un ejemplo es la sentencia `FOR`: en algunos lenguajes el valor final de la variable de control es indefinido, en otros es igual al último valor que tuvo, y en otros es mayor que el último valor. No es posible equipar a un procesador con una instrucción que sea capaz de satisfacer a cualquier lenguaje. Por ello, los compiladores suelen utilizar una secuencia de instrucciones simples con las que consiguen fácilmente la traducción de la sentencia de alto nivel.

Otro motivo por el que los compiladores no utilizan el juego completo de instrucciones es que **llevaría mucho tiempo analizar en cada sentencia cuál es la instrucción o secuencia de instrucciones más rápida o más corta**.

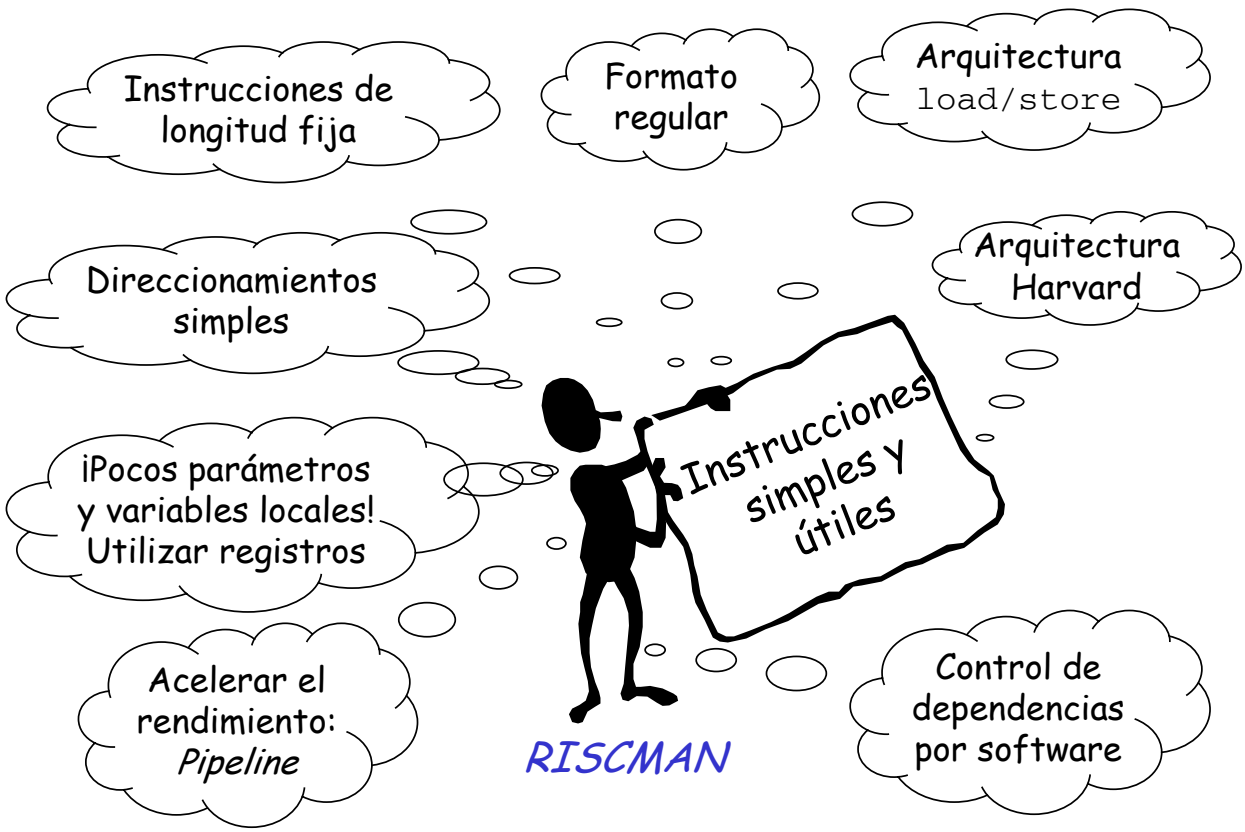
Para favorecer la tarea del compilador se debe ofrecer un juego de instrucciones que sea regular y ortogonal. Debe proporcionar “primitivas” en lugar de “soluciones”, para darle la oportunidad al compilador de componer, en cada caso, la secuencia óptima de instrucciones, en lugar de no ofrecer otra alternativa que un ineficiente juego de complejas instrucciones que no se adaptan exactamente a los requisitos del lenguaje de alto nivel.



Según lo que acabamos de comentar en la página anterior, se prefiere ofrecer una interfaz regular al compilador en lugar de una interfaz aparentemente "amigable". En el peor caso, es preferible que la compilación de un programa sea más costosa a que todas las ejecuciones del programa compilado sean más lentas.

Un ejemplo de esto último lo tenemos en la instrucción `INDEX` del VAX-11. Esta instrucción calcula la dirección de un elemento de una matriz al mismo tiempo que comprueba que el índice del elemento respeta los límites de la matriz. Patterson comprobó en 1980 que a base de una secuencia de instrucciones simples (comparar, bifurcar, sumar, etc.) conseguía una mejora en velocidad del 45% sobre la instrucción original `INDEX`.

Principios RISC



Teniendo en cuenta las observaciones anteriores, se puede considerar que los Principios de las Arquitecturas RISC deberían ser los siguientes:

- Se prefiere solamente instrucciones simples y regulares, pues las complejas, al necesitar una ruta de datos más larga, ralentizan la ejecución del resto de las instrucciones, y además no suelen ser aprovechables por los compiladores.
- Las instrucciones deben ser de longitud fija (y múltiplo de palabra) para acelerar su extracción de memoria.
- Las instrucciones deben ser de formato regular, para facilitar su decodificación.
- Es preferible disponer de modos de direccionamiento simples para no entorpecer la etapa de cálculo de operandos. Cuando sea necesario se pueden realizar accesos complejos a operandos mediante instrucciones de cálculo intermedias.
- Las instrucciones deben operar preferiblemente sobre registros, para acelerar el tiempo de acceso a los operandos. Por esto, las únicas operaciones de acceso a memoria deberían ser las del tipo `load/store`.
- Las llamadas a procedimientos suelen utilizar pocos parámetros, por lo que estos podrían pasarse en registros, que es lo más rápido.
- La rutinas suelen tener pocas variables locales, luego podrían ubicarse en registros en lugar de la pila.
- Los dos puntos anteriores sugieren que las máquinas RISC deberían disponer de un número elevado de registros en la CPU, y cuándo estos sean insuficientes se puede recurrir a la pila.
- Para mejorar el rendimiento, los procesadores deberían utilizar la técnica del *pipeline*, la cual se aprovecharía bastante bien dada la gran regularidad de las instrucciones.
- Dado que hay un número elevado de bifurcaciones y llamadas a procedimientos, debe tenerse muy en cuenta en las máquinas segmentadas.
- Es preferible que las dependencias del *pipeline* las resuelva el compilador, en lugar de complicar el hardware para su detección.
- Para acelerar el acceso a memoria, es aconsejable utilizar una arquitectura Harvard, es decir, memorias separadas para código y para datos. (Esto se materializa con memorias caché separadas).

Aunque no está claro si es o no un principio de diseño RISC, sí se debe comentar que con anterioridad a este tipo de máquinas, los procesadores solían ser microprogramados, mientras que los RISC, al ser más simples, empezaron a tener directamente cableada su lógica interna, con el consiguiente aumento de velocidad.

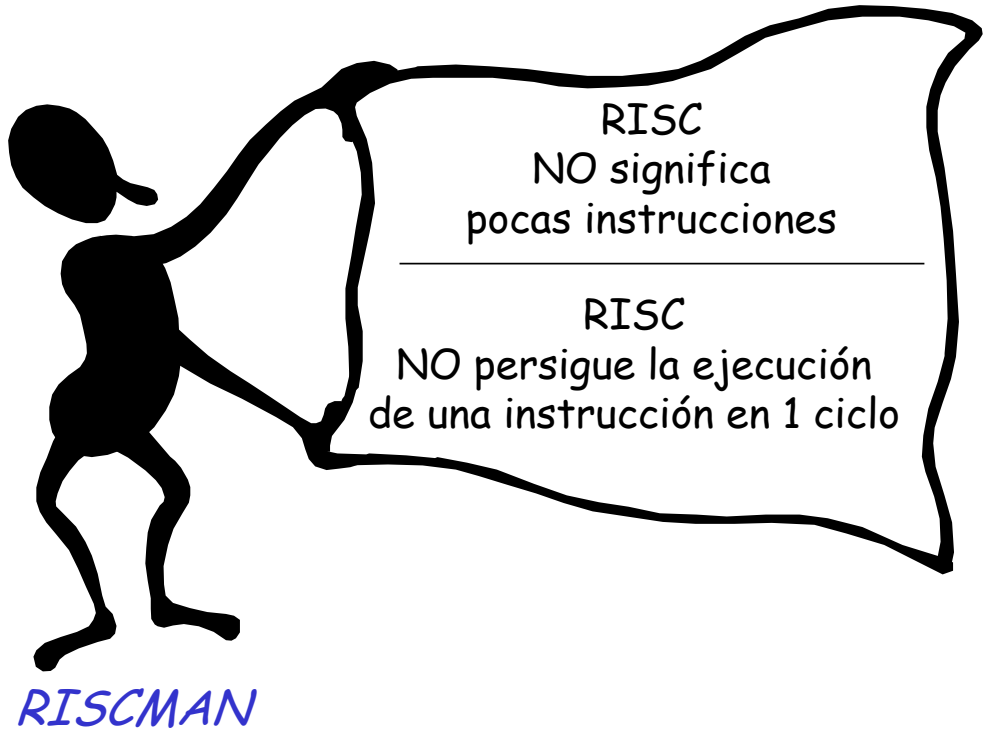
MODELO	Nº INSTRUCCIONES
RISC I	39
MIPS	55
IBM 370/168	208
Xerox Dorado	270
Vax 11/780	303
MC68000	83
MC68020	101
Intel 80486	235
IBM RS/6000	184

¡ PERO ... !

Lo que no caracteriza a las arquitecturas RISC.

En la tabla superior, se muestra el número de instrucciones que ofrecían distintos ordenadores de la historia; unos con filosofía RISC (como RISC I, MIPS y el IBM RS/6000); otros, claramente CISC, como el Xerox Dorado, Vax 11, y los procesadores de Intel y Motorola…

…Principios RISC

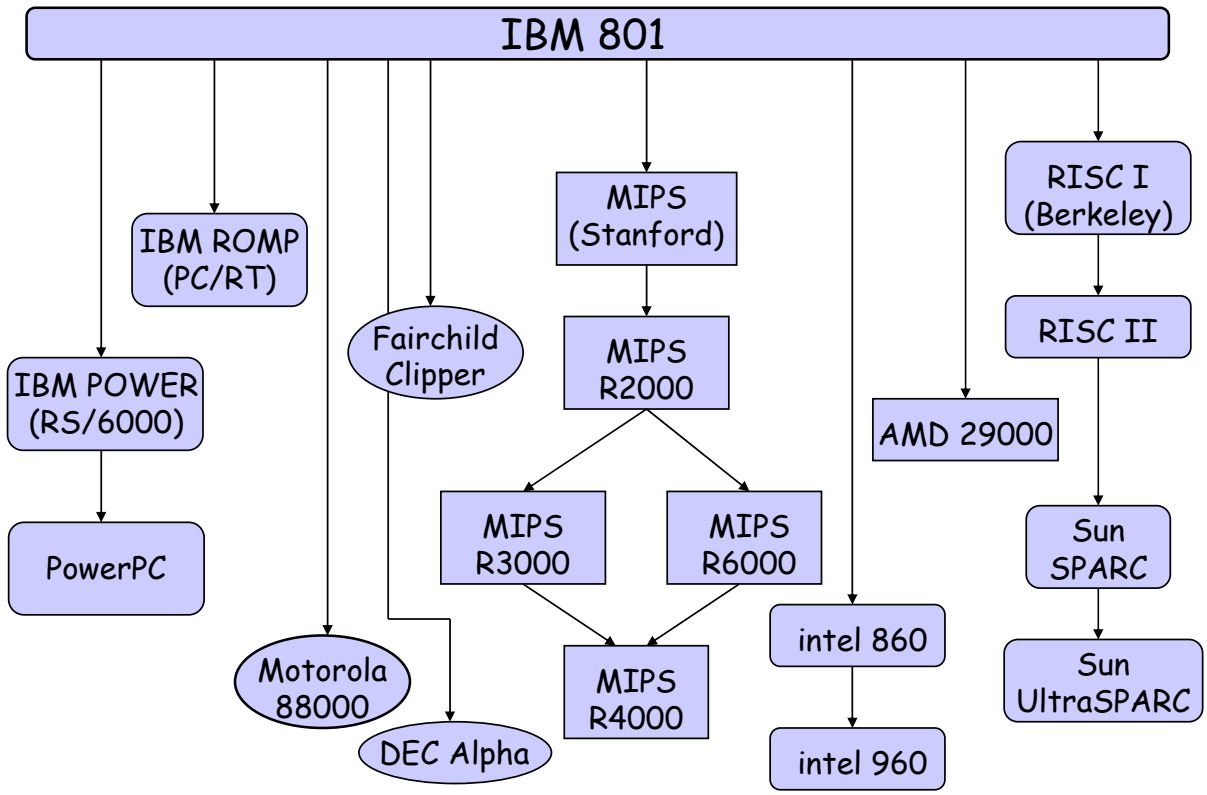


…Como se puede apreciar en la tabla de la diapositiva anterior, en un principio, las arquitecturas RISC ofrecían un juego con bastantes menos instrucciones que las tradicionales.

Por esto, el grupo de Berkeley acuñó el conocido acrónimo RISC (de hecho su máquina fue la RISC I). Pero se debe dejar claro que, actualmente, **la reducción del número de instrucciones no es ningún objetivo de la filosofía RISC**. Fijémonos, por ejemplo en el RS/6000 de IBM; aún siendo RISC, cuenta con 184 instrucciones. Incluso hoy día, conocidos procesadores como el PowerPC, podrían considerarse como CISC si no fuera porque su elevado número de instrucciones está compuesto de instrucciones de longitud fija, simples, regulares y ortogonales.

Tampoco se debe pensar que las arquitecturas RISC persiguen que la duración de una instrucción sea un ciclo de reloj. Lo que realmente persiguen es que mediante una buena organización se consiga aprovechar bien el *pipeline*, de tal manera que sí se consiga aproximarse mucho al rendimiento de una instrucción por ciclo.

Genealogía



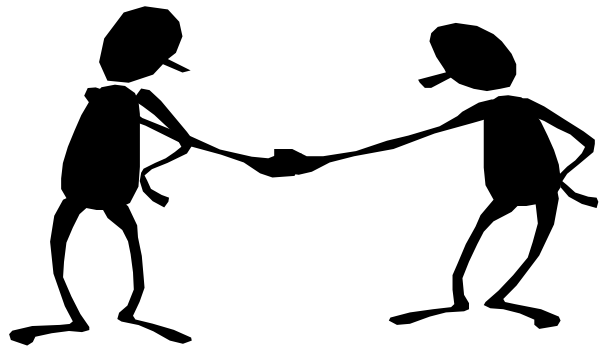
A modo de anécdota, aquí presentamos un árbol genealógico (un poco anticuado) de algunos modelos representativos de las arquitecturas RISC.

No se puede considerar ni exacto en la cronología, ni completo, pero puede ayudar simplemente a presentar algunos procesadores que han seguido este modelo de arquitectura.

Casi todos los constructores de ordenadores tienen sus procesadores RISC. Algunos otros procesadores RISC que no se muestran en el árbol son el Spectrum, de Hewlett-Packard; el Acorn ARM, el Pyramid 90X, y un largo etcétera.

Aunque se dice que la filosofía RISC proviene de las ideas de Seymour Cray (diseñador de los CDC de Control Data y de los Cray, que fueron los ordenadores más rápidos del mundo), el primer ordenador que se construyó con esta arquitectura fue el 801 de IBM, y basados en él han ido naciendo el resto de los procesadores RISC que conocemos.

Los Procesadores de Hoy Día



RISCMAN CISCMAN

Julio 2007

- ✓ Superescalar (factor 2-4)
- ✓ 64 bits de datos
- ✓ Muy segmentados
- ✓ Caché: 32 KB + 2 MB + 24 MB (L1+L2+L3)
- ✓ Coprocesador matemático
- ✓ Muchos registros (Itanium 2: 96 mínimo)

Arquitectura de Computadores Arquitecturas RISC - 18