Procesadores RISC

Contenido

Introducción

A partir del desarrollo de la computadora con programa almacenado en 1950, han habido pocas innovaciones en la arquitectura de las computadoras. Los principales avances son:

1. *Concepto de familia*. Separa la arquitectura de una máquina de su implementación.
2. *Unidad de control microprogramada*. Facilita la tarea de diseñar e implementar la unidad de control y da soporte al concepto de familia.
3. *Memoria cache*. Mejoró las prestaciones de manera sustancial.
4. *Múltiples procesadores*.

5. *RISC (“Reduced Instruction Set Computer”)*. Los elementos clave que han definido y diseñado a la mayoría de los sistemas RISC son:
   - Conjunto de instrucciones limitado y sencillo.
   - Un gran número de registros de uso general, o el uso de tecnología de compiladores para optimizar el uso de los registros.
   - Un énfasis en la optimización de la segmentación de instrucciones.

Características de la Ejecución de Instrucciones

Una de las partes relacionadas con las computadoras que ha evolucionado de manera más visible es el de los lenguajes de programación. Conforme el costo del hardware ha disminuido, el costo relativo del software ha ido creciendo, ya sea por escasez de programadores, o bien, por la falta de fiabilidad de los programas. La respuesta a lo anterior, por parte de investigadores y de la industria es desarrollar lenguajes de programación de alto nivel más potentes y complejos (HLL "high-level languages") que permitan al programador expresar los algoritmos de manera más concisa.

Lo anterior ocasionó otro problema, conocido como el *salto semántico*, es decir, la diferencia entre las operaciones que proporcionan los HLLs y las que proporciona la arquitectura de la computadora. Las consecuencias son: ineficiencia de la ejecución, tamaño excesivo del programa en lenguaje máquina y complejidad de los compiladores. Para lo anterior, los diseñadores propusieron arquitecturas que incluyen grandes conjuntos de instrucciones, docenas de modos de direccionamiento, y varias sentencias HLL implementadas en hardware. Las instrucciones están pensadas para:

1. Facilitar el trabajo del escritor de compiladores.
2. Mejorar la eficiencia de la ejecución.
3. Dar soporte a HLLs aún más complejos y sofisticados.

Los aspectos cuyo cálculo tiene interés para conocer las características de ejecución de las instrucciones, son:

1. *Operaciones realizadas*.
2. *Operandos usados*.
3. *Secuenciamiento de la ejecución*.

Operaciones

En varios estudios donde se ha analizado el comportamiento de los programas escritos en HLLs, se tiene que las sentencias de asignación predominan y también las sentencias condicionales (IF, LOOP), estas últimas se implementan en lenguaje máquina con alguna instrucción del tipo comparar y saltar.

Para el diseñador del conjunto de instrucciones máquina le indica que debe ser implementado de una forma “óptima”. Lo anterior no indica que sentencias consumen más tiempo de ejecución. Para determinar el número medio de instrucciones máquina y referencias a memoria por cada tipo de sentencia, se compilaron en VAX, PDP-11 y Motorola 68000 los programas de Patterson.

	Frecuencia dinámica		*Instrucciones máquina ponderadas*		*Referencias a memoria ponderadas*	
	Pascal	*C*	*Pascal*	*C*	*Pascal*	*C*
ASSIGN	45	38	13	13	14	15
LOOP	5	3	42	32	33	26
CALL	15	12	31	33	44	45
IF	29	43	11	21	7	13
GOTO	–	3	–	–	–	–
OTRAS	6	1	3	1	2	1

Operandos

De acuerdo al estudio de Patterson, se muestra que la mayoría de las referencias se hacen a variables escalares simples, donde más del 80% de los datos escalares son variables locales al procedimiento. De acuerdo a estos estudios y otros se refleja la importancia de una arquitectura que preste a un rápido acceso a operandos, es decir, es un candidato fundamental a optimizar.

Llamadas a procedimientos

De acuerdo a la tabla mostrada anteriormente las llamadas y retornos a procedimientos son las operaciones que consumen más tiempo en programas HLL compilados, por lo que es conveniente implementar estas operaciones eficientemente, los aspectos que deben considerarse son: el número de parámetros y variables y la profundidad de anidamiento. El estudio de Tanenbaum indica que el 98% de los procedimientos llamados dinámicamente se les pasaban menos de seis argumentos, y que el 92% de ellos usaba menos de seis variables escalares locales. El equipo RISC de Berkeley presentó resultados parecidos y además encontraron que es poco común tener una larga secuencia ininterrumpida de llamadas procedimientos seguida por la correspondiente secuencia de retornos. Con lo anterior se confirma la conclusión de que las referencias a operandos están muy localizadas.

Consecuencias

Una arquitectura de conjunto de instrucciones cercano al de los HLLs no es la estrategia de diseño más efectiva. Se puede ofrecer mejor soporte optimizando las prestaciones de las características de los programas típicos en HLL que más tiempo consumen.

Con el trabajo de varios investigadores se tienen tres elementos que, por lo general, caracterizan las arquitecturas RISC.

- El uso de un gran número de registros para optimizar las referencias a operandos, donde se busca reducir las referencias a memoria a costa de más referencias a registros.

- El diseño de los cauces de instrucciones, debido a instrucciones de bifurcación condicional y de llamada a procedimientos. Un cauce de instrucciones sencillo será ineficiente ya que una gran proporción de instrucciones que se precaptan, nunca llegan a ejecutarse.
- Conjunto de instrucciones simplificado (reducido).

Uso de un Gran Fichero de Registro

Hay una gran proporción de sentencias de asignación en programas HLL, siendo muchas de éstas de la forma $A = B$. Se tiene un número significativo de accesos a operandos y la mayoría de los accesos se hacen a datos escalares locales que justifica un almacenamiento en registros.

El almacenamiento en registros es el más rápido disponible. El fichero de registros es pequeño y esta en el mismo chip que la ALU y la unidad de control, y emplea direcciones cortas. Se necesita una estrategia que permita que los operandos que se usan con mayor frecuencia se encuentre en registros y se minimicen las operaciones registro-memoria.

Las aproximaciones básicas posibles son:

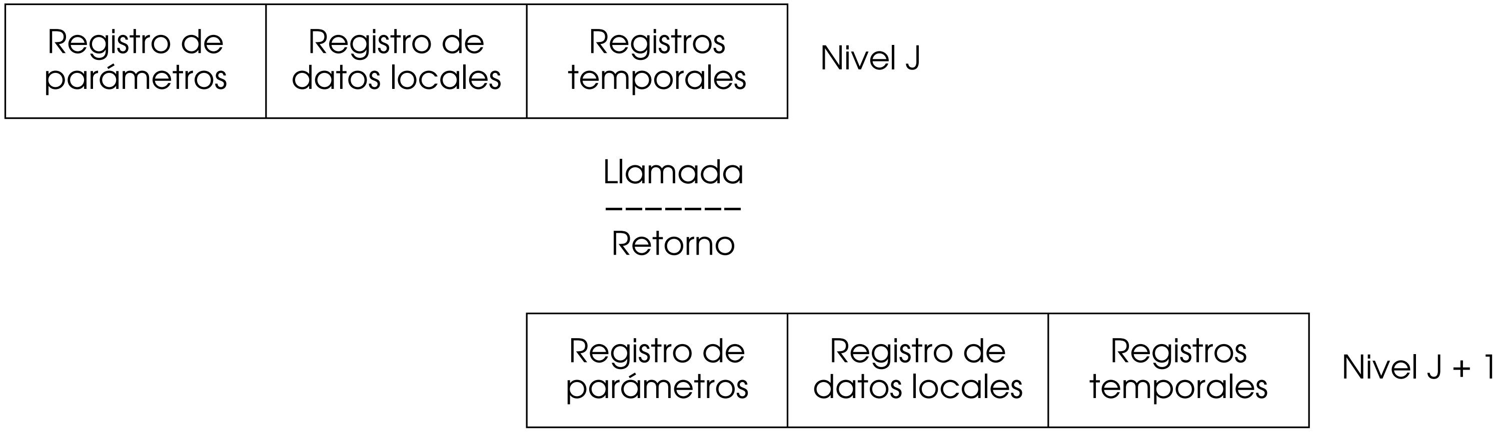
- Software. Confiar al compilador la maximización del uso de los registros, donde intentará asignar registros a las variables que se usen más. Se requiere el uso de algoritmos sofisticados para el análisis del programa.
- Hardware. Usar más registros.

Ventana de registros

Se deriva, que un uso amplio de registros debe reducir la necesidad de acceder a la memoria. Para los datos escalares locales, que son los más usados en las referencias, se deben almacenar en registros, reservando tal vez varios registros para variables globales. Un problema es la definición del *local* ya que varía con cada llamada y retorno de procedimiento.

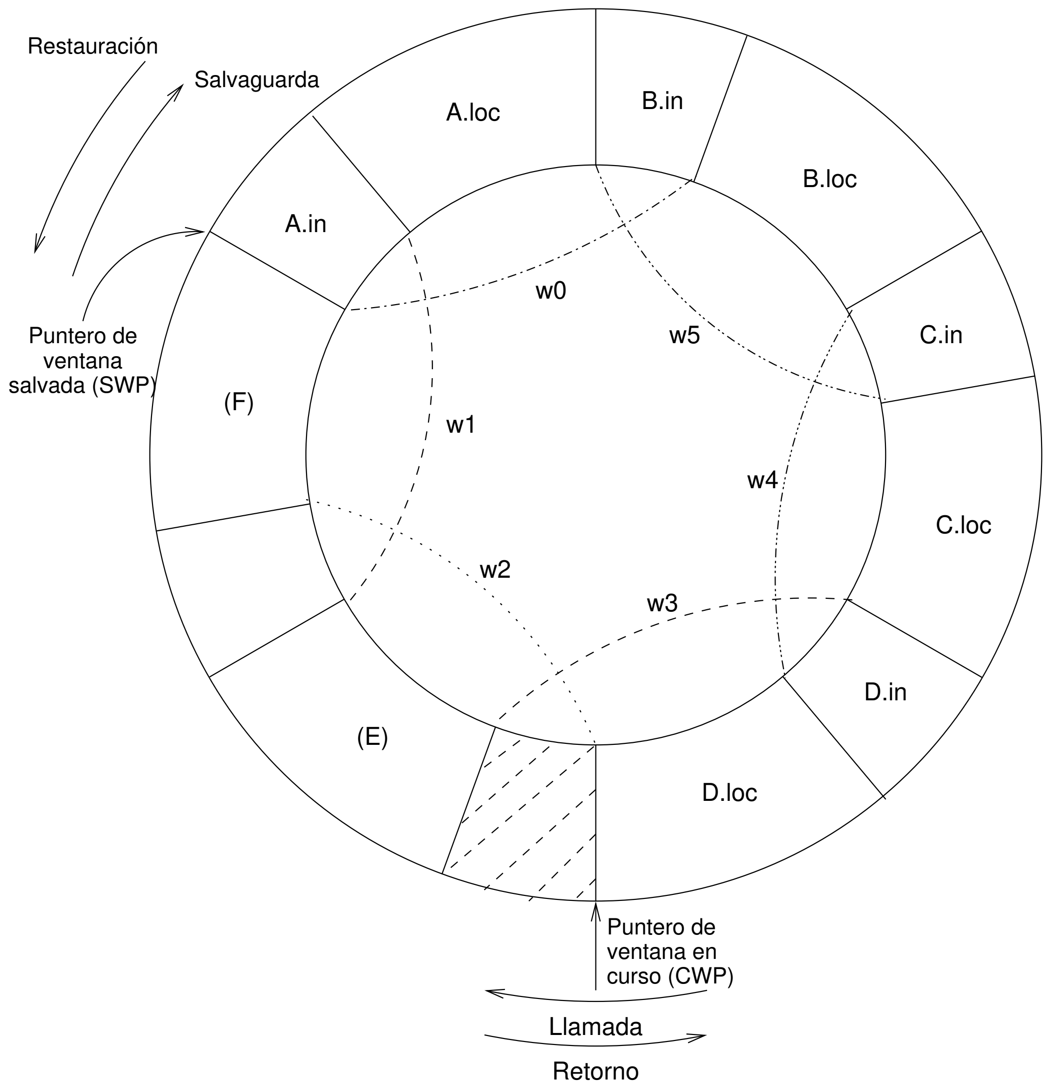
De acuerdo a los estudios comentados previamente en un procedimiento típico se tienen pocos parámetros de llamada y variables locales, y por otra parte, la profundidad de activación de procedimientos es pequeño. Para usar estas propiedades, se emplean múltiples conjuntos pequeños de registros, c/u asignado a un procedimiento distinto. Una llamada a un procedimiento hace que el CPU conmute automáticamente a una ventana distinta de registros, en lugar de guardarlos en memoria. El concepto se muestra en la siguiente figura, donde los registros de parámetros contienen parámetros pasados al procedimiento, los registros locales se usan para variables locales y los registros temporales de un nivel son físicamente los mismos que los registros de parámetros del nivel más bajo adyacente.

Ventanas de registros solapadas



Las ventanas de registros contienen unas pocas activaciones de procedimientos y las más recientes, las más antiguas han de guardarse en memoria y restaurarse más tarde, por lo que la organización es un buffer circular de ventanas solapadas. La organización se muestra en la siguiente figura, con el puntero de ventana en curso (“Current-Window Pointer”,CWP) y el puntero de ventana salvada (“Saved-Window Pointer”,SWP).

Buffer circular con 6 ventanas solapadas



Variables Globales

El esquema de ventanas no trata el almacenamiento de variables globales, se sugieren dos opciones:

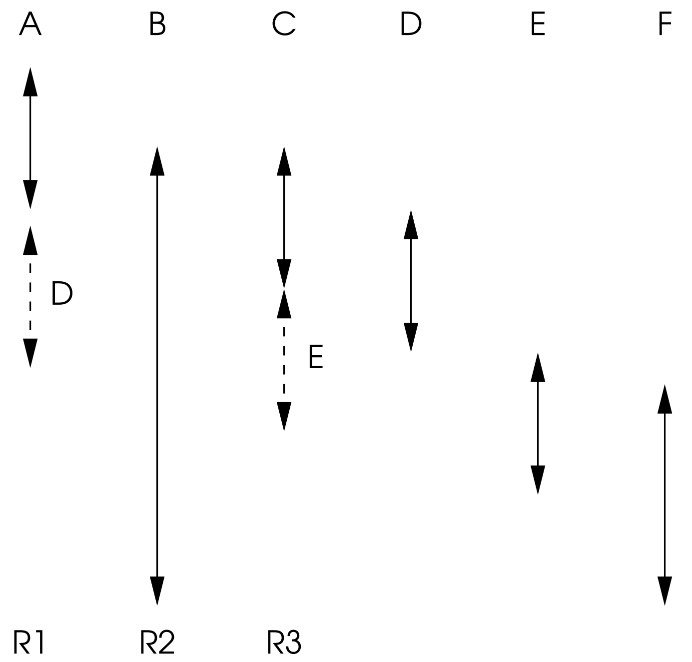
- Que el compilador asigne posiciones de memoria y sean referenciadas, por lo tanto, en memoria, lo cual es sencillo, pero si se accede frecuentemente es ineficiente.
- Usar un conjunto de registros globales donde el compilador ha de decidir que variables globales deben ser asignadas a registros.

Optimización de Registros

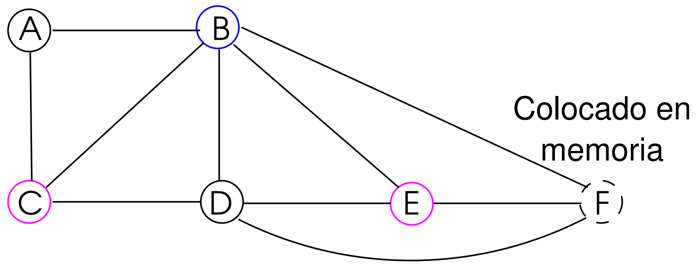
El objetivo del compilador es mantener en registros los operandos necesarios para tantos cálculos como sea posible, y minimizar las operaciones de carga-almacenamiento. Se usa la siguiente aproximación, cada cantidad candidata a residir en un registro se asigna a un registro simbólico. Después el número ilimitado de registros simbólicos es asignado a un número fijo de registros reales. Los registros simbólicos cuyo uso no se solapa pueden compartir el mismo registro real. Si hay más cantidades que registros, algunas de las cantidades se asignan a posiciones de memoria y otras a registros. La técnica usada en los compiladores para RISC es el coloreado de grafos.

El problema es, dado un grafo asignar colores a los nodos de manera que nodos adyacentes tengan colores diferentes y se minimice el número de colores distintos. Se adapta una vez que el programa se ha analizado para construir un grafo de interferencias entre registros. Los nodos son los registros simbólicos. Si dos registros simbólicos están “vivos” se unen por un arco para representar su interferencia. Se intenta colorear el grafo con n colores, donde n es el número de registro. Los nodos que no se pueden colorear se colocan en memoria.

Técnica del coloreado de grafos



Secuencia temporal de uso activo de registros simbólicos



Grafo de interferencias entre registros

Arquitectura de Conjunto de instrucciones reducido

La tendencia hacia conjuntos de instrucciones más ricos, con un número mayor y más complejo se debe al deseo de simplificar los compiladores y mejorar las prestaciones (programas más pequeños y rápidos), con lo que se da mejor soporte a los HLL. Los programas más pequeños ocupan menos memoria, y se mejoran las prestaciones porque se captan menos bytes de instrucción y en un entorno paginado se ocupan menos páginas, reduciendo la falta de página.

Un programa para el CISC, expresado en lenguaje máquina simbólico puede ser *más corto*, pero el número de bits de memoria que ocupa no tiene por que ser *más pequeño*. Ya que los compiladores en los CISC tienden a elegir las instrucciones más sencillas, y los códigos de operación son más largos por haber varias instrucciones. Por otra parte, para tener un conjunto de instrucciones más rico, la unidad de control debe hacerse más compleja o el almacenamiento del microprograma debe hacerse más grande, con lo que aumenta el tiempo de ejecución de las instrucciones simples.

Características comunes

1. *Una instrucción por ciclo*. Un *ciclo máquina* es el tiempo que se tarda en captar dos operandos de registros, realizar una operación de la ALU, y almacenar el resultado en un registro. Con instrucciones sencillas hay poca necesidad de microcódigo y pueden estar cableadas.

2. *Operaciones registro a registro*. La mayoría de las operaciones deben ser del tipo citado, con lo que se simplifica el conjunto de instrucciones y por tanto la unidad de control.

3. *Modos de direccionamiento sencillos*. Los más complejos se pueden hacer por software.

4. *Formatos de instrucción sencillos*. Se usa uno sólo o unos pocos. La longitud de instrucción es fija. Las posiciones de los campos son fijas, con lo que la decodificación del código de operación y el acceso a los operandos se puede hacer simultáneamente, también ayudan as simplificar la unidad de control

Comparación registro a registro y memoria a memoria

8	16	16	16
Add	B	C	A

Memoria a memoria
I=56, D=96, M=152

8	4	16	
Load	rB	B	
Load	rC	C	
Add	rA	rB	rC
Store	rA	A	

Registro a registro
I=104, D=96,M=200

A <– B+C

I=Tamaño total de las instrucciones ejecutadas

D=Tamaño total de los datos en esas instrucciones

M=I+D=Tráfico total con memoria

8	16	16	16
Add	B	C	A
Add	A	C	B
Sub	B	D	D

Memoria a memoria
I=168, D=288, M=456

8	4	4	4
Add	rA	rB	rC
Add	rB	rA	rC
Sub	rD	rD	rB

Registro a registro
I=60, D=0,M=60

A <– B+C; B <– A+C; D <– D–B

Las ventajas potenciales de la aproximación RISC se pueden separar en prestaciones e implementación VLSI. Para las prestaciones se pueden desarrollar compiladores optimizadores más eficaces. Se tienen una unidad de control más rápido que un CISC ya que usa poco o ningún código para las instrucciones que son sencillas. La segmentación de las instrucciones, se puede aplicar mucho más eficazmente con un conjunto de instrucciones reducido. Deben ser más sensibles a las interrupciones.

Características CISC vs. RISC

1. Un único tamaño de instrucción.
2. Tamaño típico de 4 bytes.
3. Pequeño número de modos de direccionamiento, menor que 5.
4. No se usa direccionamiento indirecto.
5. No hay operaciones que combinen carga/almacenamiento con cálculos aritméticos.
6. No se direcciona más de un operando de memoria por instrucción.
7. Las operaciones de carga/almacenamiento no soportan un alineamiento de datos arbitrario.
8. El número de bits de un campo designador de registro entero es de 5 o más.
9. El número de bits de un campo designador de registro de punto flotante es de 4 o más.

Segmentación en RISC

La mayoría de las instrucciones son del tipo registro a registro. El ciclo de instrucción tiene las siguientes dos fases:

- **I**: Captación de instrucción.
- **E**: Ejecución. Realiza una operación de la ALU con registros como entrada y salida.

Las operaciones de carga y almacenamiento requieren tres fases:

- **I**: Captación de instrucción.
- **E**: Ejecución. Calcula una dirección de memoria.
- **D**: Memoria. Operación registro a memoria o memoria a registro.

Temporización de la ejecución secuencial

Load	A $\leftarrow$ M	I	E	D												
Load	B $\leftarrow$ M				I	E	D									
Add	C $\leftarrow$ A+B							I	E							
Store	M $\leftarrow$ C									I	E	D				
Branch	X												I	E	D	

Tiempo $\longrightarrow$

En la segmentación de dos vías (siguiente figura), en el cual las fases I y E de dos instrucciones diferentes se pueden ejecutar simultáneamente, se ofrece el doble de velocidad de ejecución que un esquema serie. No se consigue la máxima aceleración, ya que se usa una memoria de un único puerto y sólo es posible un acceso a memoria en cada fase, el otro problema es que una instrucción de bifurcación interrrumpe el flujo secuencial. El compilador o ensamblador puede insertar una instrucción NOOP en el flujo de instrucciones.

Temporización con un cauce de dos dos vías

Load A $\leftarrow$ M	I	E	D							
Load B $\leftarrow$ M		I		E	D					
Add C $\leftarrow$ A+B				I		E				
Store M $\leftarrow$ C						I	E	D		
Branch X							I		E	
NOOP									I	E

La segmentación puede mejorarse permitiendo dos accesos a memoria por fase, como se ve en la siguiente figura, con lo que hasta tres instrucciones pueden solaparse. Lo anterior funciona mejor si las tres fases son aproximadamente de la misma duración.

Temporización con un cauce de tres vías

Load	A ← M	I	E	D					
Load	B ← M		I	E	D				
NOOP				I	E				
Add	C ← A+B				I	E			
Store	M ← C					I	E	D	
Branch	X						I	E	
NOOP								I	E

Temporización con un cauce de cuatro vías

La fase E puede ser más larga, ya que requiere una operación de la ALU, pero se puede dividir en dos subfases:

- E_1: Lectura del fichero de registros.
- E_2: Operación de la ALU y escritura en el registro.

Load	A $\leftarrow$ M	I	E_1	E_2	D							
Load	B $\leftarrow$ M		I	E_1	E_2	D						
NOOP					I	E_1	E_2					
Add	C $\leftarrow$ A+B					I	E_1	E_2				
Store	M $\leftarrow$ C						I	E_1	E_2	D		
Branch	X							I	E_1	E_2		
NOOP									I	E_1	E_2	
NOOP										I	E_1	E_2

Optimización de la segmentación

La naturaleza sencilla y regular de las instrucciones RISC permite emplear la segmentación eficientemente. Hay poca variación en la duración de la ejecución de instrucciones, pero las dependencias de datos y bifurcaciones reducen la velocidad de ejecución global. Para compensar lo anterior se usan técnicas de reorganización del código.

En el salto retardado se intercambian las instrucciones, por lo que JUMP se capta antes de la instrucción ADD, pero ADD se capta antes de la ejecución de la instrucción JUMP, con lo que se conserva la semántica original del programa.

Para bifurcaciones condicionales si la condición comprobada por la bifurcación puede alterarse por la instrucción inmediatamente precedente se debe insertar un NOOP.

La carga retardada se usa con las instrucciones LOAD, donde el CPU bloquea el registro destino de la carga, hasta que alcanza una instrucción que necesite ese registro, deteniéndose hasta que la carga finalice. Si el compilador puede reorganizar las instrucciones de forma que se haga el trabajo útil mientras la carga está en el cauce, la eficiencia se aumenta.

Uso del salto retardado

100 Load X,A	I	E	D				
101 Add 1,A		I	E				
102 Jump 106			I	E			
103 NOOP				I	E		
106 Store A,Z					I	E	D

Inserción de NOOP

100 Load X,A	I	E	D			
101 Jump 105		I	E			
102 Add I,A			I	E		
105 Store A,Z				I	E	D

Inversión del orden de las instrucciones

Controversia RISC y CISC

Se puede agrupar en las siguientes categorías para evaluar las ventajas del RISC:

- *Cuantitativa*. Comparar el tamaño de los programas y su velocidad de ejecución.
- *Cualitativa*. Soporte de lenguajes de alto nivel y uso óptimo de los recursos VLSI.

Al intentar compararlos se tienen los siguientes problemas:

- No hay una pareja de máquinas RISC y CISC que sean comparables.
- No existe un conjunto de programas de prueba definitivo.
- Es difícil separar los efectos del hardware respecto a la habilidad en la escritura del compilador.
- Los análisis comparativos con RISC han usado máquinas “de juguete”. Las máquinas comerciales tipo RISC poseen una mezcla de RISC y CISC.