CISC vs RISC

Rodrigo Baessa

CISC

Al iniciarse el desarrollo de Microprocesadores, los diseñadores se encontraron con los mismos problemas que con los circuitos digitales de propósito específico:

- Largos tiempos de desarrollo
- Gran costo de errores de diseño/implementación

CISC

La complejidad típicamente se originaba en la lógica de control (*control logic*) que permite que todos los componentes trabajen de la forma correcta.

CISC

La Solución:

Utilizar una memoria (tipo ROM, no volátil) para implementar esta lógica de control. Las entradas de la lógica de control forman la dirección de la memoria y el contenido de la posición de memoria son las salidas de control internas.

Esta memoria es conocida como *microcódigo*.

CISC

Así, se acortan los tiempos de desarrollo y si hay cambios, basta con modificar el microcódigo.

CISC

Al tener esta nueva herramienta, los fabricantes pudieron hacer diseños más complejos, más rápido y a menor costo.

Se buscó entonces, resolver un problema de aquel tiempo: acortar la brecha semántica (*semantic gap*).

CISC

La brecha semántica es la diferencia entre la representación de un problema en un lenguaje de alto nivel (Ej.: C, Java, Scheme) y el lenguaje de máquina (secuencia de 0s y 1s).

Recordemos que en aquel tiempo, una de las tareas más difíciles era la programación de compiladores y sistemas operativos en assembler. Cada arquitectura tenía su(s) sistemas operativos propietario(s).

CISC

Así, buscaron tener un lenguaje de máquina con instrucciones complejas que se parecieran a los comandos y estructuras de lenguajes de alto nivel con modos de direccionamiento que les facilitaran la programación en assembler.

CISC

Así, el set de instrucciones aumentó en cantidad de instrucciones considerablemente.

CISC

Por otro lado, los fabricantes de las primeras arquitecturas de 8 bits que fueron populares comenzaron a sacar al mercado procesadores de 16 bits que, debido a la gran cantidad de software que existía para sus antecesores de 8 bits, debían mantener compatibilidad (*backwards compatibility*).

CISC

Estos procesadores tenían entonces que incluir en su set de instrucciones todas la instrucciones de 8 bits de sus antecesores.

Así fue como se fueron desarrollando las arquitecturas *CISC (Complex Instruction Set Computer)*.

CISC (Resumen)

Objetivo: Ayudar al programador de Assembler.

Estrategia: Reducir la brecha semántica.

Características de CISC:

- Control interno por medio de microcódigo
- Gran cantidad de instrucciones
- Muchos modos de direccionamiento
- Pocos registros
- Casi todas la instrucciones pueden utilizar la memoria como operando
- Largo de instrucciones variable

RISC

A mediados de los ochenta, ya existían sistemas operativos multiplataforma populares (UNIXes). Esto permitía tener compiladores de múltiples arquitecturas (*cross-compilers*). La programación en assembler era poca.

El diseño de compiladores estaba más desarrollado y se giró la atención a la optimización del código generado por estos en busca de la mejora en desempeño.

RISC

Surgió un movimiento, plasmado en un artículo técnico de William Stallings llamado "Reduced Instruction Set Computer".

En este, Stallings proponía y describía un nuevo tipo de arquitectura basado en la simpleza. Proponía reducir drásticamente el número de instrucciones, los modos de direccionamiento y restringir los accesos a memoria a las instrucciones de Load y Store.

Su objetivo era maximizar el desempeño del procesador.

RISC

Estos cambios implicaban que la lógica de control sería más simple y podría ser implementada con compuertas (más rápido que microcódigo), como en los inicios del microprocesador.

Adicionalmente, ya existía software de diseño de circuitos integrados que ayudaba a acortar los tiempos y costos de diseño y de debugging.

RISC

La simplificación permitió implementar ciertas mejoras arquitecturales que hasta ese momento solo se encontraban en los procesadores de mainframes y supercomputadoras, como *caching* y *pipelinening*.

El área de pastilla que ocuparía la lógica de control era menor y permitía incluir otras cosas deseables en la pastilla como caches y *Floating Point Unit (FPU*).

RISC

El iniciar arquitecturas nuevas también permitiría comenzar de cero sin acarrear con instrucciones viejas y características poco deseables para mantener backwards compatibility.

Dado que el acceso a memoria esta restringido a Load y Store, los procesadores RISC deben tener más registros y normalmente implementan una arquitectura de set de instrucciones *3 address machine*.

RISC (Resumen)

Objetivos: Maximizar el desempeño.

Estrategia: Implementar mejoras arquitecturales, lógica de control más rápida, facilitar la implementación de algoritmos optimizantes en los compiladores.

Características de RISC:

- Control interno por medio de compuertas (hardwired)
- Poca cantidad de instrucciones
- Pocos modos de direccionamiento
- Muchos registros
- Accesos a memoria restringidos a Load y Store
- Largo de instrucciones fijo

CISC vs RISC

En la actualidad, las arquitecturas CISC de alto desempeño se han beneficiado de las mismas mejoras arquitecturales que los RISC implementaron por primera vez (caching, pipelinening, etc.).

Debido a falta de compatibilidad y a falta de recursos para investigación y desarrollo, los RISC no han obtenido popularidad en el segmento de desktops y laptops.

Son populares en los segmentos de Servidores de medio y alto desempeño y aplicaciones *embedded*.