Complex instruction set computing (CISC)

En la arquitectura computacional, CISC (del inglés *complex instruction set computer*) es un modelo de arquitectura de computadora. Los microprocesadores CISC tienen un conjunto de instrucciones que se caracteriza por ser muy amplio y permitir operaciones complejas entre operandos situados en la memoria o en los registros internos, en contraposición a la arquitectura RISC.

Este tipo de arquitectura dificulta el paralelismo entre instrucciones, por lo que, en la actualidad, la mayoría de los sistemas CISC de alto rendimiento implementan un sistema que convierte dichas instrucciones complejas en varias instrucciones simples del tipo RISC, llamadas generalmente microinstrucciones.

Los CISC pertenecen a la primera corriente de construcción de procesadores, antes del desarrollo de los RISC. Ejemplos de ellos son: Motorola 68000, Zilog Z80 y toda la familia Intel x86 usada en la mayoría de las computadoras personales actuales.

Hay que hacer notar, sin embargo que la utilización del término CISC comenzó tras la aparición de los procesadores RISC como nomenclatura despectiva por parte de los defensores/creadores de éstos últimos.

RISC

En la arquitectura computacional, **RISC** (del inglés *reduced instruction set computer*, Computación de Juego de Instrucciones Reducidas) es un tipo de diseño de CPU generalmente utilizado en microprocesadores o microcontroladores con las siguientes características fundamentales:

1. Instrucciones de tamaño fijo y presentadas en un reducido número de formatos.
2. Sólo las instrucciones de carga y almacenamiento acceden a la memoria de datos.

Además estos procesadores suelen disponer de muchos registros de propósito general.

El objetivo de diseñar máquinas con esta arquitectura es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria. Las máquinas RISC protagonizan la tendencia actual de construcción de microprocesadores. PowerPC,DEC Alpha, MIPS, ARM, SPARC... son ejemplos de algunos de ellos.

RISC es una filosofía de diseño de CPU para computadora que está a favor de conjuntos de instrucciones pequeñas y simples que toman menor tiempo para ejecutarse. El tipo de procesador más comúnmente utilizado en equipos de escritorio, el x86, está basado en CISC en lugar de RISC, aunque las versiones más nuevas traducen instrucciones basadas en CISC *x86* a instrucciones más simples basadas en RISC para uso interno antes de su ejecución.

La idea fue inspirada por el hecho de que muchas de las características que eran incluidas en los diseños tradicionales de CPU para aumentar la velocidad estaban siendo ignoradas por los programas que eran ejecutados en ellas. Además, la velocidad del procesador en relación con la memoria de la computadora que accedía era cada vez más alta. Esto conllevó la aparición de numerosas técnicas para reducir el procesamiento dentro del CPU, así como de reducir el número total de accesos a memoria.

Filosofía de diseño RISC

A finales de los setenta, investigaciones en IBM (y otros proyectos similares en otros lugares), demostraron que la mayoría de esos modos de direccionamiento *ortogonal* eran ignorados por la mayoría de los programas. Esto fue un efecto colateral en el incremento en el uso de compiladores para generar los programas, algo opuesto a escribirlos en lenguaje ensamblador. Los compiladores tendían a ser demasiado tontos en términos de las características que usaban, un efecto colateral del intento por hacerlos pequeños. El mercado se estaba moviendo hacia un uso más generalizado de los compiladores, diluyendo aún más la utilidad de los modelos ortogonales.

Otro descubrimiento fue que debido a que esas operaciones eran escasamente utilizadas, de hecho tendían a ser más lentas que un número pequeño de operaciones haciendo lo mismo. Esta paradoja fue un efecto colateral del tiempo que se utilizaba diseñando los CPU, los diseñadores simplemente no tenían tiempo de optimizar cada instrucción posible, y en vez de esto sólo optimizaban las más utilizadas. Un famoso ejemplo de esto era la instrucción `VAX INDEX`, que se ejecutaba más lentamente que un ciclo que implementara el mismo código.

Casi al mismo tiempo, las CPU comenzaron a correr a velocidades mayores que las de la memoria con la que se comunicaban. Aún a finales de los setenta, era aparente que esta disparidad continuaría incrementándose al menos durante la siguiente década, para entonces los CPU podrían ser cientos de veces más rápidos que la memoria. Esto significó que los avances para optimizar cualquier modo de direccionamiento serían completamente sobrepasados por las velocidades tan lentas en las que se llevaban a cabo.

Otra parte del diseño RISC llego desde las medidas prácticas de los programas en el mundo real. Andrew Tanenbaum reunió muchos de éstos, demostrando así que la mayoría de los procesadores estaban sobredimensionados. Por ejemplo, él demostró que el 98 % de todas las constantes en un programa podían acomodarse en 13 bits, aun cuando cada diseño de CPU dedicaba algunos múltiplos de 8 bits para almacenarlos, típicamente 8, 16 o 32, una palabra entera. Tomando este hecho en cuenta sugiere que una máquina debería permitir que las constantes fuesen almacenadas en los bits sin utilizar de otras instrucciones, disminuyendo el número de accesos a memoria. En lugar de cargar números desde la memoria o los registros, éstos podrían estar *ahí mismo* para el momento en el que el CPU los necesitara, y por lo tanto el proceso sería mucho más rápido. Sin embargo, esto requería que la instrucción misma fuera muy pequeña, de otra manera no existiría suficiente espacio libre en los 32 bits para mantener constantes de un tamaño razonable.

Fue el pequeño número de modos y órdenes que dio lugar al término *conjunto reducido de instrucciones*. Ésta no es una definición correcta, ya que los diseños RISC cuentan con una vasta cantidad de conjuntos de instrucciones para ellos. La verdadera diferencia es la filosofía para hacer todo en registros y llamar y guardar los datos hacia ellos y en ellos mismos. Ésta es la razón por la que la forma más correcta de denominar este diseño es *cargar-almacenar*. Con el paso del tiempo las técnicas de diseño antiguas se dieron a conocer como *Computadora con Conjunto de Instrucciones Complejo*, CISC por sus siglas en inglés, aunque esto fue solamente para darles un nombre diferente por razones de comparación.

Por esto la filosofía RISC fue crear instrucciones pequeñas, implicando que había pocas, de ahí el nombre *conjunto reducido de instrucciones*. El código fue implementado como series de esas instrucciones simples, en vez de un sola instrucción compleja que diera el mismo resultado. Esto hizo posible tener más espacio dentro de la instrucción para transportar datos, resultando esto en la necesidad de menos registros en la memoria. Al mismo tiempo la interfaz con la memoria era considerablemente simple, permitiendo ser optimizada.

Sin embargo RISC también tenía sus desventajas. Debido a que una serie de instrucciones son necesarias para completar incluso las tareas más sencillas, el número total de instrucciones para la lectura de la memoria es más grande, y por lo tanto lleva más tiempo. Al mismo tiempo no estaba claro dónde habría o no una ganancia neta en el desempeño debido a esta limitación, y hubo una batalla casi continua en el mundo de la prensa y del diseño sobre los conceptos de RISC.

Multitarea

Debido a lo redundante de las microinstrucciones, los sistemas operativos diseñados para estos microprocesadores, contemplaban la capacidad de subdividir un microprocesador en varios, reduciendo el número de instrucciones redundantes por cada instancia del mismo. Con una arquitectura del software optimizada, los entornos visuales desarrollados para estas plataformas, contemplaban la posibilidad de ejecutar varias tareas en un mismo ciclo de reloj. Así mismo, la paginación de la memoria RAM era dinámica y se asignaba una cantidad suficiente a cada instancia, existiendo una especie de 'simbiosis' entre la potencia del microprocesador y la RAM dedicada a cada instancia del mismo.

La multitarea dentro de la arquitectura CISC nunca ha sido real, tal como en los RISC sí lo es. En CISC, el microprocesador en todo su conjunto está diseñado en tantas instrucciones complejas y diferentes, que la subdivisión no es posible, al menos a nivel lógico. Por lo tanto, la multitarea es aparente y por ordenes de prioridad. Cada ciclo de reloj trata de atender a una tarea instanciada en la RAM y pendiente de ser atendida. Con una cola de atención por tarea FIFO para los datos generados por el procesador, y LIFO para las interrupciones de usuario, trataban de dar prioridad a las tareas que el usuario desencadenara en el sistema. La apariencia de multitarea en un CISC tradicional, viene de la mano de los modelos escalares de datos, convirtiendo el flujo en un vector con distintas etapas y creando la tecnología pipeline.

Los microprocesadores actuales, al ser hibridos, permiten cierta parte de multitarea real. La capa final al usuario es como un CISC tradicional, mientras que las tareas que el usuario deja pendientes, dependiendo del tiempo de inactividad, el sistema traducirá las instrucciones (el software ha de ser compatible con esto) CISC a RISC, pasando la ejecución de la tarea a bajo nivel, en donde los recursos se procesan con la filosofía RISC. Dado que el usuario solo atiende una tarea por su capacidad de atención, el resto de tareas que deja pendientes y que no son compatibles con el modelo de traducción CISC/RISC, pasan a ser atendidas por el tradicional pipeline, o si son tareas de bajo nivel, tal como desfragmentaciones de disco, chequeo de la integridad de la información, formateos, tareas gráficas o tareas de cálculo matemático intenso.

En vez de tratar de subdividir a un solo microprocesador, se incorporó un segundo microprocesador gemelo, indéntico al primero. El inconveniente es que la RAM debía de ser tratada a nivel hardware y los módulos diseñados para plataformas monoprocesador no eran compatibles o con la misma eficiencia, que para las plataformas multiprocesador. Otro inconveniente, era la fragmentación del BYTE de palabra. En un RISC tradicional, se ocupan los

BYTES de la siguiente forma: Si la palabra es de 32 BITS (4 BYTES de palabra de 8 BITS cada una, o dos de 16 o una de 32), dependiendo de la profundidad del dato portado, dentro del mismo BYTE, se incluian partes de otras instrucciones y datos. Ahora, al ser dos microprocesadores distintos, ambos usaban registros independientes, con accesos a la memoria propios (en estas plataformas, la relacion de RAM por procesador es de 1/1). En sus orígenes, las soluciones se parecían a las típicas ñapas de albanil, cada placa base incorporaba una solución solamente homologada por la chip set usada y los drivers que la acompañaban. Si bien la fragmentación siempre ha sido como ese mosquito que zumba en el oido, pero que por pereza permitimos que nos pique, llegó un momento que era imposible evadir el zumbido. Esta época llegó con las plataformas de 64 BITS.

Simple Instruction Set Computing (SISC)

SISC (*Simple Instruction Set Computing*) es un tipo de arquitectura de microprocesadores orientada al procesamiento de tareas en paralelo. Esto se implementa mediante el uso de la tecnología VLSI, que permite a múltiples dispositivos de bajo costo que se utilicen conjuntamente para resolver un problema particular dividido en partes disjuntas. La arquitectura RISC es un subconjunto del SISC, centrada en la velocidad de procesamiento debido a un conjunto de instrucciones reducido.[1]

Microprocesadores SISC (o RISC) nunca han logrado amenazar el amplio dominio de los procesadores CISC en los ordenadores personales, debido a su popularidad y al aumento constante en la capacidad de procesamiento de los mismos.[2] Por lo tanto, el uso de RISC y SISC sigue limitado a necesidades muy específicas de procesamiento, como en los procesadores DSP.

Conjunto de Instrucciones

Un **conjunto de instrucciones** o **repertorio de instrucciones**, **juego de instrucciones o ISA** (del inglés *Instruction Set Architecture*, Arquitectura del Conjunto de Instrucciones) es una especificación que detalla las instrucciones que una CPU de un ordenador puede entender y ejecutar, o el conjunto de todos los comandos implementados por un diseño particular de una CPU. El término describe los aspectos del procesador generalmente visibles a un programador, incluyendo los tipos de datos nativos, las instrucciones, los registros, la arquitectura de memoria y las interrupciones, entre otros aspectos.

Existe principalmente de 3 tipos: CISC (*Complex Instruction Set Computer*), RISC (*Reduced Instruction Set Computer*) y SISC (*Simple Instruction Set Computing*).

La arquitectura del conjunto de instrucciones (ISA) se emplea a veces para distinguir este conjunto de características de la microarquitectura, que son los elementos y técnicas que se emplean para implementar el conjunto de instrucciones. Entre estos elementos se encuentras las microinstrucciones y los sistemas de caché.

Procesadores con diferentes diseños internos pueden compartir un conjunto de instrucciones; por ejemplo el Intel Pentium y AMD Athlon implementan versiones casi idénticas del conjunto de instrucciones x86, aunque tienen diseños internos completamente opuestos.

Ciclo de CPU

Un **ciclo de CPU** es un pulso electromagnético que genera el oscilador de cuarzo presente en todo procesador o microprocesador de computadora.

La velocidad de funcionamiento del microprocesador viene determinada por el ritmo de los impulsos de su reloj.

Este reloj u oscilador es un circuito electrónico encargado de emitir a un ritmo constante impulsos eléctricos.

El funcionamiento de este reloj es comparable con un metrónomo con su péndulo que oscila de izquierda a derecha. El intervalo de tiempo que el péndulo tarda en recorrer esa distancia y regresar a su punto inicial se denomina ciclo.

Todos los microprocesadores poseen un oscilador o reloj que, al igual que el metrónomo, marca el número de ciclos por segundo. En principio podría pensarse que a mayor número de ciclos por segundo, mayor velocidad, pero esto es cierto solo cuando se comparan procesadores de diseño similar. Según la arquitectura del procesador (RISC o CISC) y de la tecnología empleada se requerirán más o menos ciclos para la ejecución de una instrucción, o incluso más de una instrucción por ciclo. Por ello la velocidad de procesadores un parámetro para comparar prestaciones entre procesadores similares, pero no decisivo, pues en la velocidad del procesador influyen otros parámetros como la memoria caché, etc.

Frecuencia de reloj

La mayoría de los CPU, y de hecho, la mayoría de los dispositivos de lógica secuencial, son de naturaleza síncrona.[8] Es decir, están diseñados y operan en función de una señal de sincronización. Esta señal, conocida como **señal de reloj**, usualmente toma la forma de una onda cuadrada periódica. Calculando el tiempo máximo en que las señales eléctricas pueden moverse en las varias bifurcaciones de los muchos circuitos de un CPU, los diseñadores pueden seleccionar un período apropiado para la señal del reloj.

Este período debe ser más largo que la cantidad de tiempo que toma a una señal moverse, o propagarse en el peor de los casos. Al fijar el período del reloj a un valor bastante mayor sobre el retardo de la propagación del peor caso, es posible diseñar todo el CPU y la manera que mueve los datos alrededor de los "bordes" de la subida y bajada de la señal del reloj. Esto tiene la ventaja de simplificar el CPU significativamente, tanto en una perspectiva de diseño, como en una perspectiva de cantidad de componentes. Sin embargo, esto también tiene la desventaja que todo el CPU debe esperar por sus elementos más lentos, aún cuando algunas unidades de la misma son mucho más rápidas. Esta limitación ha sido compensada en gran parte por varios métodos de aumentar el paralelismo del CPU (ver abajo).

Sin embargo, las mejoras arquitectónicas por sí solas, no solucionan todas las desventajas de CPUs globalmente síncronas. Por ejemplo, una señal de reloj está sujeta a los retardos de cualquier otra señal eléctrica. Velocidades de reloj más altas en CPUs cada vez más complejas hacen más difícil de mantener la señal del reloj en fase (sincronizada) a través de toda la unidad. Esto ha conducido que muchos CPU modernos requieran que se les proporcione múltiples señales de reloj idénticas, para evitar retardar una sola señal lo suficiente como para hacer al CPU funcionar incorrectamente. Otro importante problema cuando la velocidad del reloj aumenta dramáticamente, es la cantidad de calor que es disipado por el CPU. La señal del reloj cambia constantemente, provocando la conmutación de muchos componentes (cambio de estado) sin importar si están siendo usados en ese momento. En general, un componente que está cambiando de estado, usa más energía que un elemento en un estado estático. Por lo tanto, a medida que la velocidad del reloj aumenta, así lo hace también la disipación de calor, causando que el CPU requiera soluciones de enfriamiento más efectivas.

Un método de tratar la conmutación de componentes innecesarios se llama el clock gating, que implica apagar la señal del reloj a los componentes innecesarios, efectivamente desactivándolos. Sin embargo, esto es frecuentemente considerado como difícil de implementar y por lo tanto no ve uso común afuera de diseños de muy baja potencia.[9] [10] Otro método de tratar algunos de los problemas de una señal global de reloj es la completa remoción de la misma. Mientras que quitar la señal global del reloj hace, de muchas maneras, considerablemente más complejo el proceso del diseño, en comparación con diseños síncronos similares, los diseños asincrónicos (o sin reloj) tienen marcadas ventajas en el consumo de energía y la disipación de calor. Aunque algo infrecuente, CPUs completas se han construido sin utilizar una señal global de reloj. Dos notables ejemplos de esto son el AMULET, que implementa la arquitectura del ARM, y el MiniMIPS, compatible con el MIPS R3000. En lugar de remover totalmente la señal del reloj, algunos diseños de CPU permiten a ciertas unidades del dispositivo ser asincrónicas, como por ejemplo, usar ALUs asincrónicas en conjunción con pipelining superescalar para alcanzar algunas

ganancias en el desempeño aritmético. Mientras que no está completamente claro si los diseños totalmente asincrónicos pueden desempeñarse a un nivel comparable o mejor que sus contrapartes síncronas, es evidente que por lo menos sobresalen en las más simples operaciones matemáticas. Esto, combinado con sus excelentes características de consumo de energía y disipación de calor, los hace muy adecuados para sistemas embebidos.

Paralelismo

Modelo de un CPU subescalar. Note que toma quince ciclos para terminar tres instrucciones.

La descripción de la operación básica de un CPU ofrecida en la sección anterior describe la forma más simple que puede tomar un CPU. Este tipo de CPU, usualmente referido como **subescalar**, opera sobre y ejecuta una sola instrucción con una o dos piezas de datos a la vez.

Este proceso da lugar a una ineficacia inherente en CPUs subescalares. Puesto que solamente una instrucción es ejecutada a la vez, todo el CPU debe esperar que esa instrucción se complete antes de proceder a la siguiente instrucción. Como resultado, el CPU subescalar queda "paralizado" en instrucciones que toman más de un ciclo de reloj para completar su ejecución. Incluso la adición de una segunda unidad de ejecución (ver abajo) no mejora mucho el desempeño. En lugar de un camino quedando congelado, ahora dos caminos se paralizan y aumenta el número de transistores no usados. Este diseño, en donde los recursos de ejecución del CPU pueden operar con solamente una instrucción a la vez, solo puede, posiblemente, alcanzar el desempeño **escalar** (una instrucción por ciclo de reloj). Sin embargo, el desempeño casi siempre es subescalar (menos de una instrucción por ciclo).

Las tentativas de alcanzar un desempeño escalar y mejor, han resultado en una variedad de metodologías de diseño que hacen comportarse al CPU menos linealmente y más en paralelo. Cuando se refiere al paralelismo en los CPU, generalmente son usados dos términos para clasificar estas técnicas de diseño.

- El paralelismo a nivel de instrucción, en inglés Instruction Level Parallelism (ILP), busca aumentar la tasa en la cual las instrucciones son ejecutadas dentro de un CPU, es decir, aumentar la utilización de los recursos de ejecución en la pastilla
- El paralelismo a nivel de hilo de ejecución, en inglés Thread Level Parallelism (TLP), que se propone incrementar el número de hilos (efectivamente programas individuales) que un CPU pueda ejecutar simultáneamente.

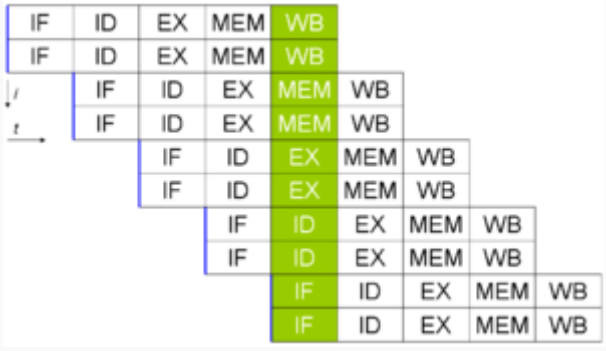
Cada metodología se diferencia tanto en las maneras en las que están implementadas, como en la efectividad relativa que producen en el aumento del desempeño del CPU para una aplicación.

ILP: Entubado de instrucción y arquitectura superescalar

Tubería básica de cinco etapas. En el mejor de los casos, esta tubería puede sostener un ratio de completado de una instrucción por ciclo.

Uno de los más simples métodos usados para lograr incrementar el paralelismo es comenzar los primeros pasos de leer y decodificar la instrucción antes de que la instrucción anterior haya terminado de ejecutarse. Ésta es la forma más simple de una técnica conocida como *Instruction Pipelining* (entubado de instrucción), y es utilizada en casi todos los CPU de propósito general modernos. Al dividir la ruta de ejecución en etapas discretas, la tubería permite que más de una instrucción sea ejecutada en cualquier tiempo. Esta separación puede ser comparada a una línea de ensamblaje, en la cual una instrucción es hecha más completa en cada etapa hasta que sale de la tubería de ejecución y es retirada.

Sin embargo, la tubería introduce la posibilidad de una situación donde es necesario terminar el resultado de la operación anterior para completar la operación siguiente; una condición llamada a menudo como conflicto de dependencia de datos. Para hacer frente a esto, debe ser tomado un cuidado adicional para comprobar estas clases de condiciones, y si esto ocurre, se debe retrasar una porción de la tubería de instrucción. Naturalmente, lograr esto requiere circuitería adicional, los procesadores entubados son más complejos que los subescalares, pero no mucho. Un procesador entubado puede llegar a ser casi completamente escalar, solamente inhibido por las abruptas paradas de la tubería (una instrucción durando más de un ciclo de reloj en una etapa).



Tubería superescalar simple. Al leer y despachar dos instrucciones a la vez, un máximo de dos instrucciones por ciclo pueden ser completadas.

Una mejora adicional sobre la idea del entubado de instrucción (instruction pipelining) condujo al desarrollo de un método que disminuye incluso más el tiempo ocioso de los componentes del CPU. Diseños que se dice que son superescalares incluyen una larga tubería de instrucción y múltiples unidades de ejecución idénticas. En una tubería superescalar, múltiples instrucciones son leídas y pasadas a un despachador, que decide si las instrucciones se pueden o no ejecutar en paralelo (simultáneamente). De ser así, son despachadas a las unidades de ejecución disponibles, dando por resultado la capacidad para que varias instrucciones sean ejecutadas simultáneamente. En general, cuanto más instrucciones un CPU superescalar es capaz de despachar simultáneamente a las unidades de ejecución en espera, más instrucciones serán completadas en un ciclo dado.

La mayor parte de la dificultad en el diseño de una arquitectura superescalar de CPU descansa en crear un despachador eficaz. El despachador necesita poder determinar rápida y correctamente si las instrucciones pueden ejecutarse en paralelo, tan bien como despacharlas de una manera que mantenga ocupadas tantas unidades de ejecución como sea posible. Esto requiere que la tubería de instrucción sea llenada tan a menudo como sea posible y se incrementa la necesidad, en las arquitecturas superescalares, de cantidades significativas de caché de CPU. Esto también crea técnicas para evitar peligros como la predicción de bifurcación, ejecución especulativa, y la ejecución fuera de orden, cruciales para mantener altos niveles de desempeño.

- La predicción de bifurcación procura predecir qué rama (o trayectoria) tomará una instrucción condicional, el CPU puede minimizar el número de tiempos que toda la tubería debe esperar hasta que sea completada una instrucción condicional.
- La ejecución especulativa frecuentemente proporciona aumentos modestos del desempeño al ejecutar las porciones de código que pueden o no ser necesarias después de que una operación condicional termine.
- La ejecución fuera de orden cambia en algún grado el orden en el cual son ejecutadas las instrucciones para reducir retardos debido a las dependencias de los datos.

En el caso donde una porción del CPU es superescalar y una parte no lo es, la parte que no es superescalar sufre en el desempeño debido a las paradas de horario. El Intel Pentium original (P5) tenía dos ALUs superescalares que podían aceptar, cada una, una instrucción por ciclo de reloj, pero su FPU no podía aceptar una instrucción por ciclo de reloj. Así el P5 era superescalar en la parte de números enteros pero no era superescalar de números de coma (o punto [decimal]) flotante. El sucesor a la arquitectura del Pentium de Intel, el P6, agregó capacidades superescalares a sus funciones de coma flotante, y por lo tanto produjo un significativo aumento en el desempeño de este tipo de instrucciones.

El entubado simple y el diseño superescalar aumentan el ILP de un CPU al permitir a un solo procesador completar la ejecución de instrucciones en ratios que sobrepasan una instrucción por ciclo (**IPC**).[12] La mayoría de los modernos diseños de CPU son por lo menos algo superescalares, y en la última década, casi todos los diseños de CPU de propósito general son superescalares. En los últimos años algo del énfasis en el diseño de computadores de alto ILP se ha movido del hardware del CPU hacia su interface de software, o ISA. La estrategia de la muy larga palabra de instrucción, very long instruction word (VLIW), causa a algún ILP a ser implícito directamente por el software, reduciendo la cantidad de trabajo que el CPU debe realizar para darle un empuje significativo al ILP y por lo tanto reducir la complejidad del diseño.

TLP: ejecución simultánea de hilos

Otra estrategia comúnmente usada para aumentar el paralelismo de los CPU es incluir la habilidad de correr múltiples hilos (programas) al mismo tiempo. En general, CPUs con alto TLP han estado en uso por mucho más tiempo que los de alto ILP. Muchos de los diseños en los que Seymour Cray fue pionero durante el final de los años 1970 y los años 1980 se concentraron en el TLP como su método primario de facilitar enormes capacidades de computación (para su tiempo). De hecho, el TLP, en la forma de mejoras en múltiples hilos de ejecución, estuvo en uso tan temprano como desde los años 1950. En el contexto de diseño de procesadores individuales, las dos metodologías principales usadas para lograr el TLP son, multiprocesamiento a nivel de chip, en inglés chip-level multiprocessing (CMP), y el multihilado simultáneo, en inglés simultaneous multithreading (SMT). En un alto nivel, es muy común construir computadores con múltiples CPU totalmente independientes en arreglos como multiprocesamiento simétrico (symmetric multiprocessing (SMP)) y acceso de memoria no uniforme (Non-Uniform Memory Access (NUMA)). Aunque son usados medios muy diferentes, todas estas técnicas logran la misma meta: incrementar el número de hilos que el CPU(s) puede correr en paralelo.

Los métodos de paralelismo CMP y de SMP son similares uno del otro y lo más directo. Éstos implican algo más conceptual que la utilización de dos o más CPU completos y CPU independientes. En el caso del CMP, múltiples "núcleos" de procesador son incluidos en el mismo paquete, a veces en el mismo circuito integrado. Por otra parte, el SMP incluye múltiples paquetes independientes. NUMA es algo similar al SMP pero usa un modelo de acceso a memoria no uniforme. Esto es importante para los computadores con muchos CPU porque el tiempo de acceso a la memoria, de cada procesador, es agotado rápidamente con el modelo de memoria compartido del SMP, resultando en un significativo retraso debido a los CPU esperando por la memoria. Por lo tanto, NUMA es considerado un modelo mucho más escalable, permitiendo con éxito que en un computador sean usados muchos más CPU que los que pueda soportar de una manera factible el SMP. El SMT se diferencia en algo de otras mejoras de TLP en que el primero procura duplicar tan pocas porciones del CPU como sea posible. Mientras es considerada una estrategia TLP, su implementación realmente se asemeja más a un diseño superescalar, y de hecho es frecuentemente usado en microprocesadores superescalares, como el POWER5 de IBM. En lugar de duplicar todo el CPU, los diseños SMT solamente duplican las piezas necesarias para lectura, decodificación, y despacho de instrucciones, así como cosas como los registros de propósito general. Esto permite a un CPU SMT mantener sus unidades de ejecución ocupadas más frecuentemente al proporcionarles las instrucciones desde dos diferentes hilos de software. Una vez más esto es muy similar al método superescalar del ILP, pero ejecuta simultáneamente instrucciones de múltiples hilos en lugar de ejecutar concurrentemente múltiples instrucciones del mismo hilo.