Arquitectura y Organización de Computadoras II

Computadoras Modernas

MSc. Ing. Ticiano J. Torres Peralta
Ing. Pablo Toledo

2023 – Segundo Cuatrimestre

Universidad Nacional de Tucumán
Facultad de Ciencias Exactas y Tecnología

CISC Vs RISC

Durante los años 70, se puso mucho énfasis en el diseño de instrucciones complejas, posibles gracias a una unidad de control interpretada. Con esta elección de diseño, los diseñadores intentaron cerrar la “brecha semántica” entre lo que podían hacer las máquinas y lo que requerían los lenguajes de programación de alto nivel. A nadie se le ocurrió diseñar una máquina más sencilla.

CISC Vs RISC

Al rededor de los años 80, empezó a surgir un nuevo tipo de arquitectura. Un grupo liderado por David Patterson y Carlo Sequin diseñó un chip de CPU que no utilizaba interpretación (en otras palabras, no estaba microprogramado). Para este concepto surgió el término RISC (Computadora con conjunto de instrucciones reducido).

Los procesadores RISC eran significativamente diferentes a los procesadores CISC. En principio, no tenían que ser compatibles con productos existentes, sus diseñadores eran libres de elegir nuevos conjuntos de instrucciones que maximizaran el rendimiento total del sistema. Si bien el énfasis inicial estaba en instrucciones simples que pudieran ejecutarse rápidamente, pronto se dieron cuenta de que diseñar instrucciones que pudieran emitirse (dispatch) rápidamente era la clave para un buen desempeño. En otras palabras, se ponía menos importancia en cuánto tiempo tomaba realmente una instrucción y mas en cuántas se podían iniciarse por segundo (maximizar el IPC).

CISC Vs RISC

Una de las principales características de estas arquitecturas es el número típicamente pequeño de instrucciones, alrededor de 50 en comparación con las 200 a 300+ en computadoras establecidas como el VAX (1000+ para Intel x86).

Aunque RISC podría tener una ventaja de rendimiento sobre su contraparte CISC, esta arquitectura no ha desplazado a la antigua CISC. ¿Por qué? El mayor problema es la compatibilidad con versiones anteriores y las empresas han invertido miles de millones de dólares en software para la línea Intel. En segundo lugar, Intel ha podido emplear las mismas ideas incluso en una arquitectura CISC. Desde el 486, las CPU Intel contienen un núcleo RISC para interpretar instrucciones simples, mientras interpretan las instrucciones más complicadas de la forma CISC habitual. El resultado neto es un rendimiento general competitivo manteniendo la compatibilidad con versiones anteriores.

CISC Vs RISC

Hoy en día, existen ciertos principios, en gran parte provenientes de los conceptos de las máquinas RISC, que se aceptan como una buena forma de diseñar computadoras. Todo esto supone que no habrá ningún cambio tecnológico importante.

En si, hay un conjunto de principios de diseño, denominados principios de diseño RISC, al que se adhieren la mayoría de las arquitecturas modernas. Por supuesto, las restricciones externas, como la compatibilidad con versiones anteriores, siempre presentan compromisos.

Principios de Diseño RISC

- Todas las instrucciones se ejecutan directamente mediante hardware.
  - Las instrucciones no se interpretan mediante microinstrucciones. Se ejecutan directamente por hardware. La eliminación de un nivel de interpretación proporciona una mayor velocidad para la mayoría de las instrucciones.
- Maximizar la taza en que se emiten las instrucciones.
  - Los ordenadores modernos recurren a muchos trucos para maximizar su rendimiento. Uno de los principales es intentar iniciar tantas instrucciones por segundo como sea posible. El principio sugiere que el paralelismo puede desempeñar un papel importante.
  - Aunque las instrucciones siempre se encuentran en el orden del programa, no siempre se emiten en el orden del programa y es posible que no finalicen en el orden del programa. Por supuesto, para instrucciones interdependientes, la CPU debe asegurarse de que la primera instrucción se ejecute antes de que se ejecute la segunda.

Principios de Diseño RISC

- Las instrucciones deben ser fáciles de decodificar.
  - Un límite crítico en la tasa de emisión de instrucciones es decodificar instrucciones individuales para determinar qué recursos necesitan. Para ayudar a esto, se pueden hacer instrucciones regulares o de longitud fija, con una pequeña cantidad de campos. Cuantos menos formatos diferentes haya para las instrucciones, mejor.
- Sólo instrucciones de cargas y almacenamientos deben hacer referencia a la memoria.
  - Una de las formas más sencillas de dividir las operaciones en pasos separados es exigir que los operandos de la mayoría de las instrucciones provengan de los registros de la CPU y los resultados vuelvan a ellos. El paso para mover operandos de la memoria a los registros se puede realizar en instrucciones separadas. Dado que el acceso a la memoria puede llevar mucho tiempo y el retraso es impredecible (depende en que nivel de la jerarquía de memoria se encuentra el dato), es mejor superponer estas instrucciones con otras instrucciones. Esta observación significa que sólo las instrucciones LOAD y STORE deben hacer referencia a la memoria. Todos los demás deberían operar sólo en registros.

Principios de Diseño RISC

- Hay que proporcionar muchos registros.
  - Dado que el acceso a la memoria es relativamente lento, es necesario proporcionar muchos registros (al menos 32). De esta manera, una vez que se busque una palabra, se la puede mantener en un registro hasta que ya no sea absolutamente necesaria. No es deseable quedarse sin registros y tener que volver a vaciarlos a memoria para luego volver a cargarlos.
- Aprovecha el paralelismo. Podemos definir tres niveles diferentes donde un arquitecto puede aprovechar el paralelismo:

1. Paralelismo a nivel del sistema: para mejorar el rendimiento de una máquina, se pueden utilizar varios procesadores y varios discos. La carga de trabajo se puede distribuir entre los procesadores y los discos, lo que da como resultado un rendimiento mejorado. Poder ampliar estos recursos se llama escalabilidad y es un activo valioso para las computadoras.

Principios de Diseño RISC

2. Paralelismo a nivel de instrucción: a nivel de procesadores individuales, la forma más sencilla de lograrlo es mediante segmentación. La idea básica detrás de la segmentación es superponer la ejecución de instrucciones para reducir el tiempo total para completar una secuencia de instrucciones. El echo más importante que permite que una segmentación funcione es que no todas las instrucciones dependen de su predecesor inmediato.

3. Paralelismo a nivel de datos: este tipo de paralelismo se explota a nivel de diseño digital. Por ejemplo, las cachés set-assosiative utilizan varios bancos de memoria que normalmente se buscan en paralelo; las ALU modernas utilizan carry-lookahead, que utiliza el paralelismo para acelerar el proceso de cálculo de la suma; distribuir datos en muchos discos o memoria para lecturas y escrituras paralelas.

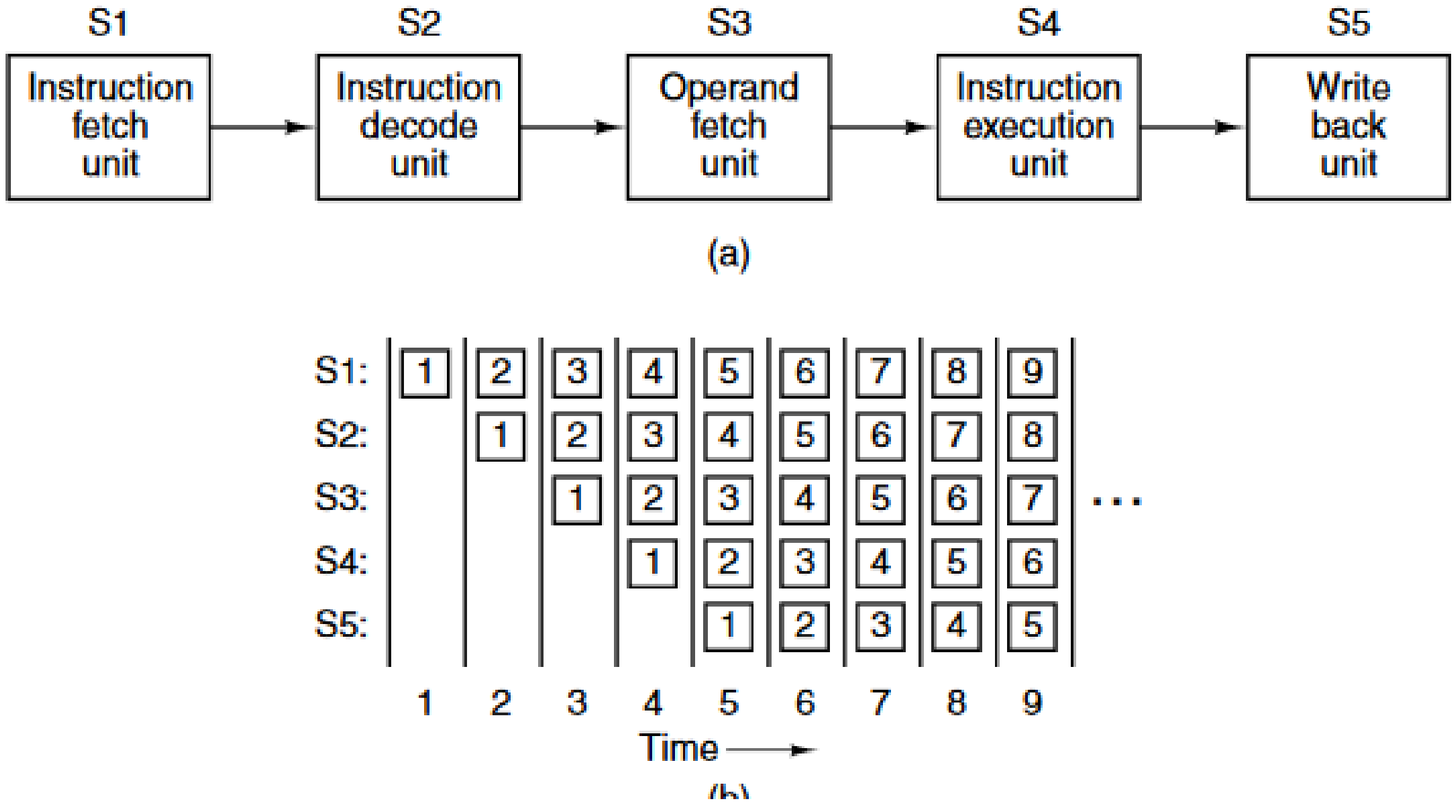
Paralelismo a Nivel Instrucciones

Sabemos que la búsqueda de instrucciones de la memoria es un cuello de botella importante en la velocidad de su ejecución. Durante muchos años, las CPU tenían la capacidad de buscar instrucciones de la memoria por adelantado, para que estuvieran ahí cuando fueran necesarias. Estos se almacenaban en un conjunto especial de registros llamado búfer de prefetch. De esta manera, cuando se necesitaba una instrucción, generalmente se podía tomar del búfer en lugar de esperar desde una lectura de memoria.

Esto dividió la ejecución de instrucciones en dos partes importantes, búsqueda y ejecución. El concepto de segmentación lleva esta estrategia mucho más allá. En lugar de dividirse en dos partes, se puede dividir en muchas partes. Cada parte maneja una pieza de hardware dedicada, que se puede ejecutar en paralelo.

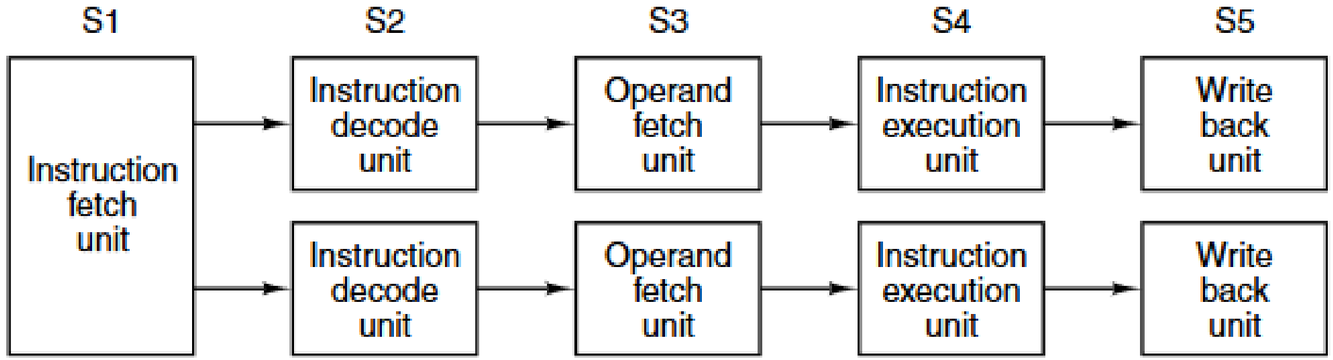
La segmentación es una compensación entre latencia y ancho de banda.

Principios de Diseño RISC



Arquitecturas Superescalares

Si un pipeline es bueno, ¡seguramente dos pipeline serán mejores!

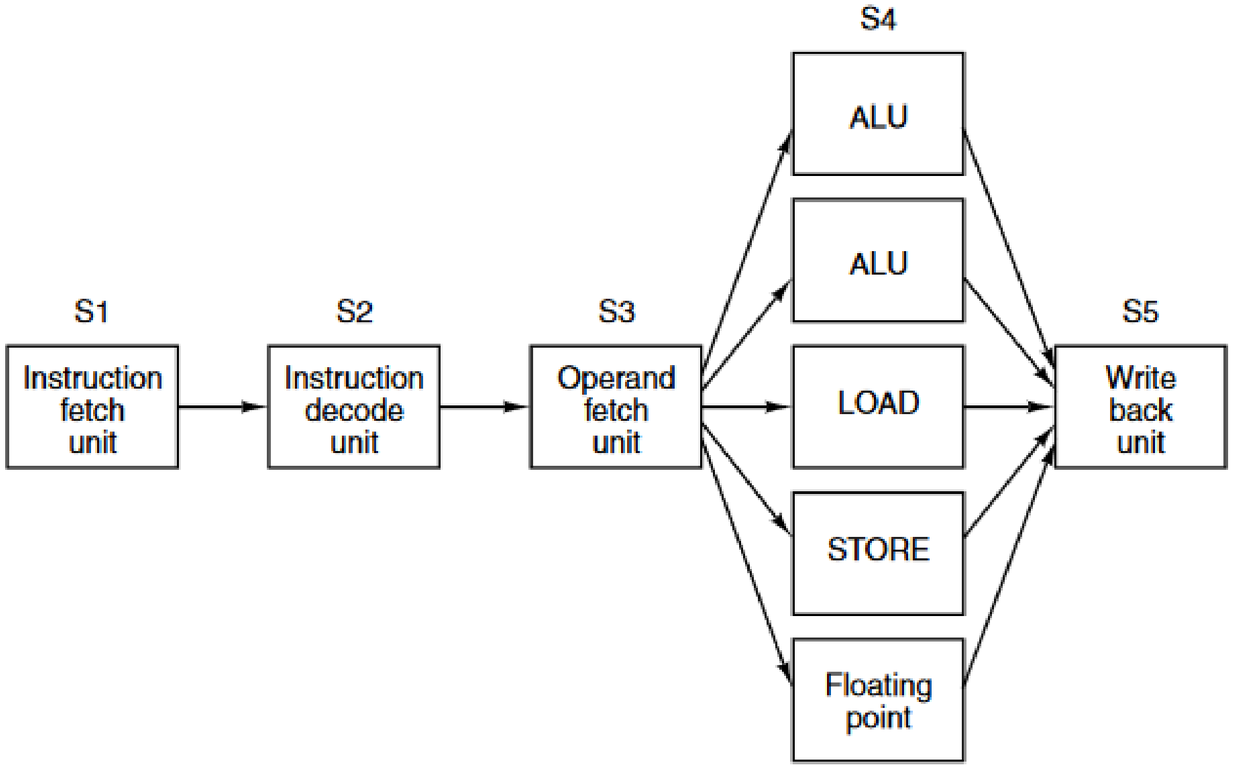


En este ejemplo, una única IFU (Instruction Fetch Unit) reúne pares de instrucciones y coloca cada una en su propia pipeline, cada una con su propia ALU para operación en paralelo. Por supuesto, cada instrucción no debe entrar en conflicto con el uso de recursos ni depender mucho de la otra.

Aumentar de a cuatro desde allí podría ser un posible siguiente paso, pero hacerlo duplica demasiado hardware. En cambio, se utiliza un enfoque diferente en las CPU de gama alta. La idea básica es tener una única pipeline pero darle múltiples unidades funcionales.

Arquitecturas Superescalares

Intel Core tiene una estructura similar a:



Arquitecturas Superescalares

El término Arquitecturas Superescalares ha evolucionado con el tiempo y ahora se refiere a procesadores que pueden emitir (dispatch) múltiples instrucciones por ciclo de reloj. Tienen múltiples unidades funcionales a las que entregar estas instrucciones. Generalmente tienen una sola pipeline.

La idea básica es que la tasa de emisión es mucho mayor que la tasa de ejecución. La suposición obvia es que la etapa S3 puede emitir instrucciones mucho más rápido que las etapas S4 pueden completarlas.

Paralelismo a Nivel Sistema

Existe un límite en la velocidad que se le puede dar un solo CPU. Estos incluyen aspectos físicos como la velocidad de la luz (influye en las latencias de los circuitos lógicos), y chips más rápidos producen más calor y la disipación es un gran problema. De hecho, los problemas con la disipación de calor es la razón principal por la que la velocidad del reloj de la CPU se ha estancado en la última década.

El paralelismo a nivel de instrucción ayuda, pero puede ganar alrededor de un factor de 5 o 10 en la mejora del rendimiento. Para obtener mayores factores, la única forma es diseñar computadoras con múltiples CPU.

Paralelismo a Nivel Datos

Un número sustancial de problemas en los dominios computacionales involucran bucles, arreglos, matrices u otras estructuras muy regulares. A menudo, el mismo cálculo se realiza repetidamente en muchos conjuntos de datos diferentes.

Se han utilizado dos soluciones principales para abordar este tipo de problemas y aprovechar del paralelismo a nivel datos:

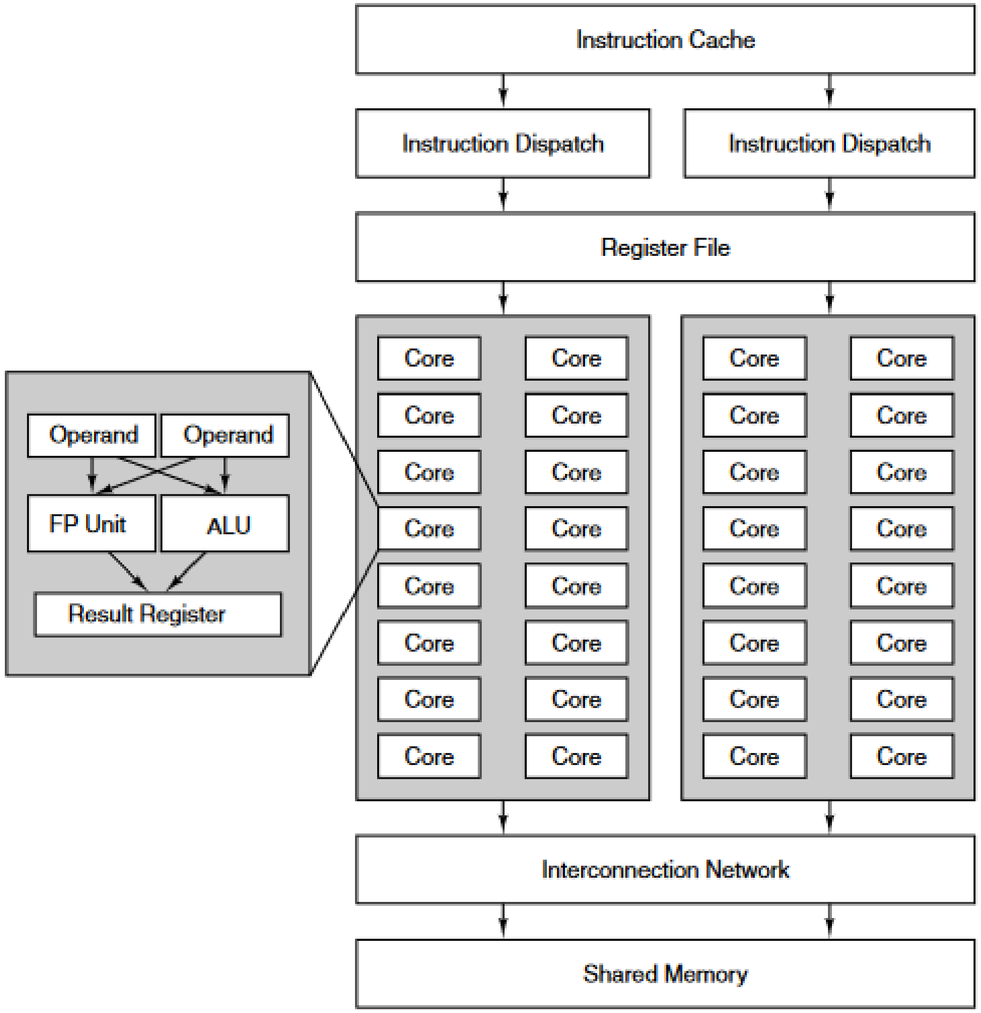
- SIMD
- Procesadores Vectoriales.

Paralelismo a Nivel Datos

SIMD (Single Instruction Multiple Data)

- Consta de un gran número de procesadores idénticos que realizan la misma secuencia de instrucciones de diferentes datos.
- Las modernas unidades de procesamiento de gráficos GPUS dependen en gran medida del procesamiento SIMD. La mayoría de los algoritmos son muy regulares, con operaciones repetidas en píxeles, vértices, texturas y bordes.
- GPU Nvidia Fermi. Contiene 16 módulos SM de multiprocesos de flujo (stream processors) SIMD, y cada SM contiene 32 procesadores SIMD.
- Un núcleo de GPU Fermi completamente cargado puede realizar 512 operaciones por ciclo. En comparación con una CPU de cuatro núcleos de uso general, que puede realizar 16.

Paralelismo a Nivel Datos

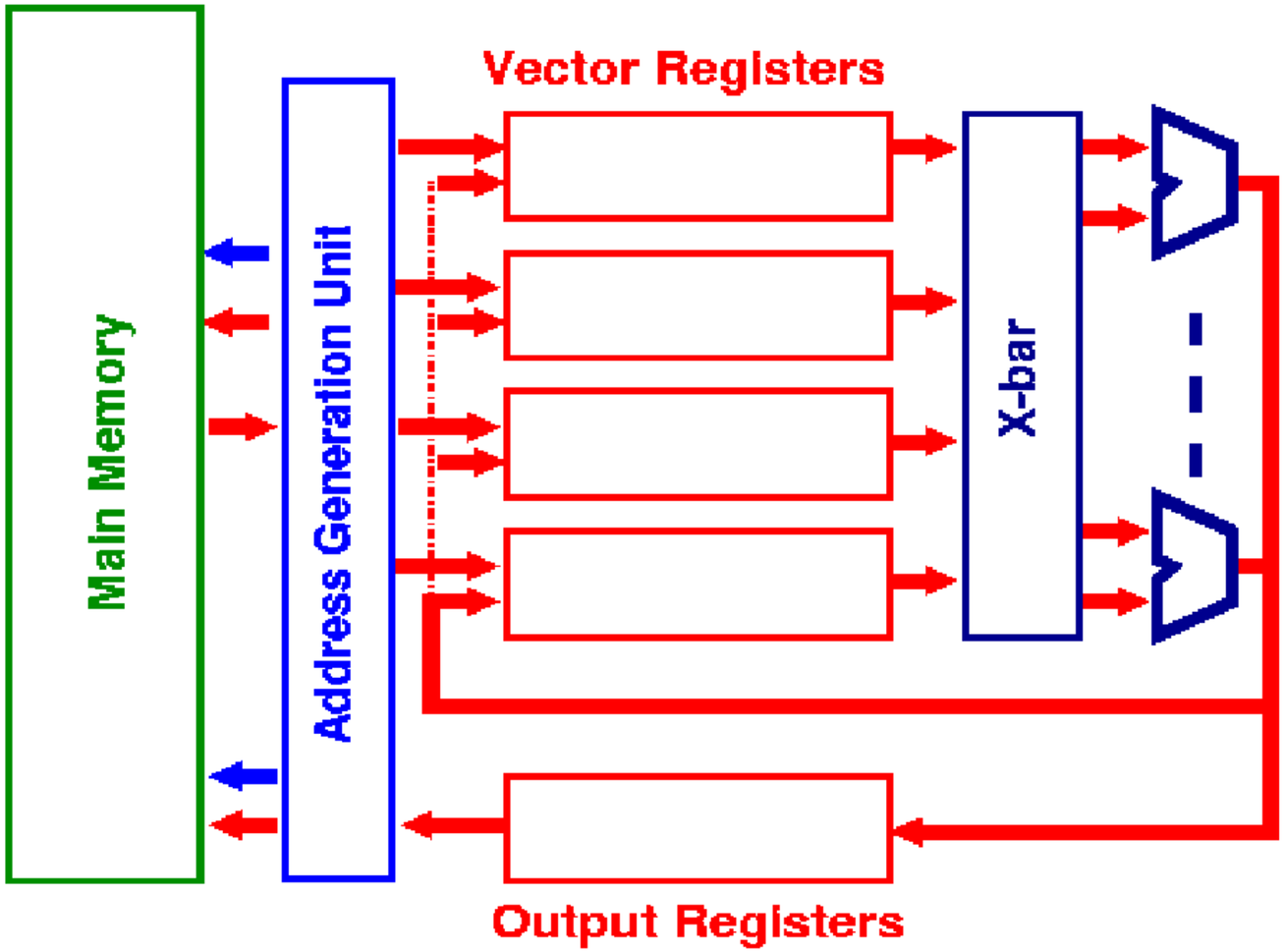


Paralelismo a Nivel Datos

Procesador Vectoriales:

- Es muy parecido a un procesador SIMD. Es muy eficiente para ejecutar una secuencia de operaciones en pares de datos. Pero a diferencia de un SIMD, todas las operaciones se realizan en una única unidad funcional fuertemente segmentada.
- El concepto de un registro vectorial, consiste en un conjunto de registros convencionales que se pueden cargar en una sola instrucción. Luego, una instrucción vectorial realiza una operación en dos de estos registros vectoriales alimentándolos a un sumador segmentado. Y El resultado se guarda en otro registro vectorial.
- SSE y AVX en Intel Core utilizan este modelo de ejecución.

Paralelismo a Nivel Datos



Principio de localidad

Importantes observaciones fundamentales provienen de las propiedades de los programas. Uno de ellos es el principio de localidad:

- Los programas tienden a reutilizar datos e instrucciones que han utilizado recientemente.

Una regla general muy utilizada es que los programas dedican el 90% de su tiempo de ejecución a sólo el 10% del código. Entonces, una implicación de este principio es que podemos predecir con una precisión razonable qué instrucciones y datos utilizará un programa en el futuro cercano en función de sus accesos en el pasado reciente. Se aplicar a los accesos a datos, aunque se aplica más fuertemente a los accesos a instrucciones.

Principio de localidad

Podemos subdividir este principio en dos localidades:

- Localidad temporal: establece que es probable que se acceda a los elementos a los que se accedió recientemente en un futuro próximo.

- Localidad espacial: establece que los elementos cuyas direcciones están cerca unas de otras tienden a ser referenciadas juntos en el tiempo.

Centrarse en el Caso Común

Quizás el principio más generalizado en el diseño de computadoras sea centrarse en el caso común. Al hacer un compromiso de diseño, favorezca el caso frecuente sobre el poco frecuente. Este principio se aplica a la hora de determinar cómo gastar en recursos, ya que el impacto de una mejora es mayor si es que ocurre con frecuencia.

Por ejemplo, la unidad de búsqueda y decodificación de instrucciones del procesador puede usarse con más frecuencia que un multiplicador, así que optimicemos esta unidad primero.

Al aplicar este principio, tenemos que decidir cuál es el caso frecuente y cuánto se puede mejorar el rendimiento haciendo que ese caso sea más rápido.