Arquitectura del Computador y Sistemas Operativos

Séptima Clase

Mejoras de Performance

Cuando tratamos de mejorar la performance de un procesador aparecen automáticamente compromisos:

- Velocidad
- Precio
- Confiabilidad
- Consumo de energía

Aumentar la velocidad por lo general complica el diseño, lo que aumenta la cantidad de transistores. Este aumento no solamente aumenta el precio, sino que al haber más piezas aumenta la probabilidad de fallas y el consumo de energía.

Como el precio aumenta más que linealmente con la cantidad de transistores, los dos primeros ítems suelen ser los más importantes a considerar al diseñar.

Mejoras de Performance

Para aumentar la velocidad de ejecución podemos:

- Bajar la cantidad de ciclos por instrucción
- Bajar la complejidad para poder usar un reloj más rápido
- Solapar pasos en la ejecución de instrucciones

Reducción de ciclos por instrucción

Por ejemplo en nuestro procesador de ejemplo cada instrucción llevaba 4 ciclos de reloj, pero en el último no se hacía nada. Se podría modificar la unidad de control para que cada una lleve 3. Es un poco más complejo usar un contador módulo 3 que uno módulo 4, pero es realizable.

Para que sea simple, todas las operaciones llevaban 4 ciclos de reloj, pero había instrucciones como CLR podrían usar 2 (buscar instrucción y borrar registro). Una vez más, que las instrucciones lleven distinta cantidad de ciclos complejiza la unidad de control, pero también es realizable.

Mejoras de Performance

Bajar complejidad y acelerar el CLK

Tomemos la fase 1 de ejecución de nuestro procesador de ejemplo:

- El registro IP tiene la dirección de dónde levantar la siguiente instrucción
- Se habilita el driver tri-state que hace que esa dirección salga al bus de direcciones
- La memoria recibe esa dirección y saca por el bus de datos la siguiente instrucción.
- Se pulsa la señal LOAD del registro IR y éste almacena la instrucción.

Cada una de esas fases lleva tiempo:

- 74LS125 → 15 ns
- HM6116-2 → 120 ns
- 74LS374 → 18 ns

Total 153 ns.

Mejoras de Performance

Bajar complejidad y acelerar el CLK

Si el acceso toma 153 ns, la frecuencia máxima de reloj sería aprox. 6.5MHz

Si cambiamos la memoria HM6116-2 por una más rápida como la R1RP0416DI, las cosas cambian dramáticamente:

- R1RP0416DI → 12 ns

Ahora el tiempo total es 45 ns, lo que permite subir el reloj arriba de los 22 MHz.

Para reducir los tiempos, hay que usar diseños más óptimos, que utilicen menor cantidad de transistores y/o transistores más pequeños y por ende más rápidos

Mejoras de Performance

Solapar pasos en la ejecución de instrucciones

Para entender cómo se puede hacer éso, primero debemos separar cada uno de los pasos de una instrucción genérica. Claramente no todas las instrucciones tienen todos estos pasos, pero al hacerlo resulta evidente que no todos los pasos usan los mismos elementos del procesador.

Instrucción	Obtener Instrucción	Decodificar Instrucción	Calcular Dirección Operandos	Levantar Operandos	Ejecutar Instrucción	Calcular Dirección Resultados	Guardar Resultados
Usa BUS	✔			✔			✔
Usa ALU			✔		✔	✔	

Ni bien lo hacemos, resulta evidente que los recursos están desaprovechados, de hecho en este ejemplo se usan menos del 50% del tiempo.

Mejoras de Performance

Solapar pasos en la ejecución de instrucciones

Ahora consideremos la ejecución simultánea de dos instrucciones

Instrucción 1	Obtener Instrucción	Decodificar Instrucción	Calcular Dirección Operandos	Levantar Operandos	Ejecutar Instrucción	Calcular Dirección Resultados	Guardar Resultados		
Instrucción 2		Obtener Instrucción	Decodificar Instrucción	Calcular Dirección Operandos	Levantar Operandos		Ejecutar Instrucción	Calcular Dirección Resultados	Guardar Resultados
Usa BUS	✔	✔		✔	✔		✔		✔
Usa ALU			✔	✔	✔	✔	✔	✔	

Se ve que ahora la ocupación de los recursos mejoró, pero también que hubo momentos en que dos instrucciones requerían el mismo recurso, y éso obligó a "suspender" una instrucción a la espera de que el mismo estuviera disponible.

- El proceso de solapamiento se denomina *pipelining*.
- La suspensión a la espera de un recurso se denomina *stalling*.

Mejoras de Performance

Solapar pasos en la ejecución de instrucciones

Pero el proceso es mucho más complejo de lo que parece. No es sólo cuestión de esperar un recurso, consideremos los siguientes casos:

- La primera instrucción es un salto, y la condición de salto se cumple. La segunda instrucción no debería ejecutarse

- La primera instrucción guarda los datos en el registro B, y la segunda usa el registro B como operando. La segunda instrucción debería demorarse (*stalling*) hasta que la primera haya actualizado el registro B, sino se operaría con un valor inválido.

A pesar de la complejidad, la mejora de performance es tan notoria y la competencia en el mercado por lograr un diferencial de velocidad respecto a la competencia es tan feroz que el pipelining se usa en muchísimos casos.

Mejoras de Performance

Otras formas: La memoria “Caché”

En la historia de las computadoras, las memorias no han aumentado su velocidad al ritmo que lo hicieron los procesadores. Por ende se han buscado soluciones para poder “alimentar” al procesador con el código a la velocidad que éste lo puede procesar.

Existen memorias RAM rápidas, pero son caras y de poca capacidad.

La idea del “*caché*” radica en interponer una memoria rápida entre el procesador y la memoria principal. Ésto no produce beneficio cuando el dato necesario no está en el caché, pero con una política adecuada que mantenga en el mismo los datos más frecuentemente usados, la ganancia en performance es importante.

La política no es trivial, pero generalmente parte de tratar de forma distinta el código y los datos.

En algunos casos se coloca un segundo caché (llamado “*caché nivel 2*” o “caché L2”) entre el caché nivel 1 y la memoria. Este segundo caché es más lento pero más grande que el primero.

Mejoras de Performance

Otras formas: La memoria “Caché”

Hay dos formas usadas para implementar un caché:

- “*Direct-mapped caches*”: Cada posición de memoria puede estar solamente en un lugar del caché. Dado una dirección, la parte baja determina la fila dentro del caché donde *podría* estar, luego el bloque alto se compara con el bloque alto que se guardó en la fila y se define si está o no. Si está la fila contiene el dato.

- “*n-way set-associative cache*”: Dada una dirección, hay n posibles filas donde podría estar.

Cuando un dato se guarda en el caché, la lógica del caché tiene que copiar el dato a la memoria principal. Puede hacerse:

- “*write through*”
- “*write back*”

Mejoras de Performance

Otras formas: Predicción de salto

Ejecutar una instrucción y descartar los resultados ocupa tiempo, viejos procesadores tenían OpCodes que eran “Branch Hints”. Ésto se denomina “Static branch hint”, dado que se deja al compilador la tarea de “sugerirle” al microcódigo si ejecutar la siguiente instrucción a riesgo de tener que descartar el resultado o directamente esperar que la primera termine para ejecutar la segunda.

Hoy no use usan porque los procesadores tienen “Branch Target Buffers” para tomar la decisión ello mismos. Ésto e llama “Dynamic branch hint”.

Mejoras de Performance

Otras formas: Reordenamiento de instrucciones

Si el procesador encuentra que dentro de un bloque de código sin salto hay varias instrucciones, y alguna de ellas es más lenta (por ejemplo una suma de números de punto flotante frente a la suma de enteros), el procesador podría alterar el orden de ejecución de forma que la más lenta se ejecute primero.

A esta práctica se la llama “*hoisting*”, o “*izado*”.

Fin

¿Preguntas?