Tema 05: Diseño de un ISA

Arquitectura de Computadoras

Ing. Nicolás Majorel Padilla (npadilla@herrera.unt.edu.ar)

http://microprocesadores.unt.edu.ar/arqcom/

Temas que veremos

- Características requeridas de un ISA.
- Decisiones de Diseño:
  - Tipo de ISA.
  - Instrucciones y Modos de Direccionamiento a incluir.
  - Formato de Instrucción fijo o variable.
  - Manejo de constantes.
  - Tipos de datos a soportar.
  - Evaluación de condiciones e Instrucciones de Salto.
  - 32 vs 64 bits.
- Métricas importantes.
- Principios de Diseño.

Lectura recomendada

- Computer Organization and Design, RISC-V Edition (2da ed, 2021)
  - Sección 2.2: *Operations of the Computer Hardware*
  - Sección 2.3: *Operands of the Computer Hardware*
  - Sección 2.22: *Fallacies and Pitfalls*
  - Sección 2.23: *Concluding Remarks*
- Guía Práctica de RISC-V (1ra ed)
  - Capítulo 1: *¿Por qué RISC-V?*

Repaso de Performance

- **t = CI * CPI * T**
- El ISA influye en CI y en CPI
  - Un buen diseño de ISA es simple y ordenado.
  - Pocos formatos, consistentes.
- Performance se mejora con Paralelismo y Pipelining.
  - Un buen diseño de ISA tiene en cuenta estos factores de antemano.
  - Instrucciones similares (tareas equilibradas).

Características requeridas de un ISA

- Debe ser completo.
  - Se probó que con una sola instrucción alcanza ☺
    - **movfuscator**, compilador de C para x86 que corre DOOM, utilizando únicamente instrucciones **mov**.
- Debe ser eficiente.
  - Instrucciones más usadas deben ser rápidas (Amdahl).
- Debe ser regular.
  - Operaciones y modos de direccionamiento esperados.
  - Formatos uniformes, fijos de ser posible**.**
  - **La simplicidad favorece la regularidad.**
- Debe hacer buen uso de memoria.
  - Manejar datos e instrucciones múltiplos de 1 byte.

Comparación de distintos ISAs

- ISA con operandos en Memoria:
  - ADD A, B, C M(A) ← M(B) + M(C)
  - Generan un cuello de botella en el acceso a memoria.
  - Instrucciones muy largas.
- ISA con operandos en Registros y Memoria:
  - ADD r1, A R(r1) ← R(r1) + M(A)
  - Fácil codificación, menor longitud de programa.
  - Uno de los operandos fuente se sobreescribe.
  - Instrucciones de longitud y duración diferente, según el operando.
- ISAs con operandos en Registros:
  - ADD r1, r2, r3 R(r1) ← R(r2) + R(r3)
  - Codificación simple. Facilita diseño de Hardware.
  - Necesitan más instrucciones para un mismo programa.

Comparación de distintos ISAs

- **Los ISA basados en registros se imponen.**
  - Principalmente, porque son más rápidos que memoria.
  - **Principio de diseño: cuánto más pequeño, más rápido.**
  - Además, porque son más fáciles de optimizar por el compilador.
    - Ej: (A * B) – (C * D) – (E * F).
- **Arquitectura dominante: tipo Load-Store.**
  - Únicamente las instrucciones de tipo Load y de tipo Store pueden acceder a Memoria.
  - Ideal para implementar en un Pipeline.
- En el fondo es una cuestión tecnológica.
  - *¿Qué pasaría si la memoria principal se vuelve tan rápida como los registros?*

¿Cuáles instrucciones incluir en el ISA?

- Si una operación es lenta.
  - Y obliga a que las demás sean lentas.
  - **No incluirla**.
- ¿Y si es necesaria para que el set sea completo?
  - Hacer una **pseudoinstrucción** que la reemplace.
    - Y que el ensamblador la expanda.
- Lo mejor es medir, en varios programas, cuáles son realmente las instrucciones más usadas.
  - Siempre considerando frecuencia dinámica.

¿Cuáles instrucciones incluir en el ISA?

Rango	Modos de direccionamiento	Promedio
1	Load	22%
2	Conditional Branch	20%
3	Compare	16%
4	Store	12%
5	Add	8%
6	And	6%
7	Sub	5%
8	Move Reg. to Reg.	4%
9	Call	1%
10	Return	1%
	Total	96%

- ¡Las 10 instrucciones más usadas cubren el 96% de los casos!
- Y además, ¡son las más simples!

¿Cuáles modos de direccionamiento?

- Vimos que hay muchos (>10 en Thumb2).
- Si medimos su uso en varios programas:

Modos de direccionamiento	Promedio	Mínimo	Máximo
Indexado con Desplazamiento	42%	32%	55%
Inmediato	33%	17%	43%
Registro Indirecto	13%	3%	24%
Escalado	7%	0%	16%
Memoria Indirecta	3%	1%	6%
Otros	2%	0%	3%

- ¡88% de las veces se usan sólo 3 modos!
  - **Hacer más rápido el caso más común.**

¿Qué tipo de formato elegir?

- Los formatos de instrucción pueden ser de longitud fija o variable.
- Cuando los formatos son de longitud fija, todos tienen la misma longitud.
  - Idealmente, con campos similares.
- Los formatos de longitud variable dependen de la cantidad de operandos y de su modo de direccionamiento.
  - Su objetivo es minimizar el tamaño del programa en memoria.
  - x86 tiene instrucciones desde 1 byte hasta 15 bytes.
- *¿Cuál es nuestro objetivo?*

¿Qué tipo de formato elegir?

- Los formatos de longitud fija tienen algunas ventajas:
  - En general, mejor performance.
  - Mayor simplicidad.
  - Mayor regularidad.
- Y también tienen algunas desventajas:
  - Menor densidad de código: se necesitan más instrucciones para hacer la misma tarea.
  - Menor posibilidad de ampliación: en algún momento, se acaban los bits.
- Actualmente, están de moda formatos con dos longitudes fijas.
  - Por ejemplo, instrucciones de 32 y de 16 bits únicamente.
  - Intenta obtener las ventajas de ambos esquemas.

¿Qué tamaño de constantes?

- Nuevamente, midiendo en varios programas:
  - Para Desplazamientos, sólo el 1% supera los 16 bits.
  - Para constantes, el 60% entra en 8 bits, el 80% en 16 bits.
- ¿Y las constantes más grandes?
  - *¿Un formato más ancho?*
  - No, conviene usar combinaciones de instrucciones.
  - **No castigar a todos por casos minoría.**
- La constante ‘0’ es MUY usada.
  - Idea: que un registro contenga siempre un cero, cableado.
  - Un registro menos, pero se gana en eficiencia.

¿Qué tipos de datos soportar?

- Manipulación de bits.
- Enteros y caracteres:
  - 8-bit (byte), 16-bit (halfword), 32-bit (word) , 64-bit (double-word).
  - Los números siempre en complemento a 2.
  - 75% de las veces se usan datos de 32 bits.
- Punto flotante:
  - Precisión simple (float, 32-bit), doble (double, 64-bit) y quádruple (quadword, 128-bit).
  - 70% de las veces se usa precisión doble.

¿Cómo se evalúan las condiciones?

- Mediante códigos de condición (CC).
  - Instrucciones aritméticas setean los CC, las instrucciones de salto los evalúan.
  - Suele requerir menos instrucciones.
  - Pero no todas las instrucciones cambian todos los CC.
  - Dificulta la implementación en pipeline.
    - Porque los CC se leen y se escriben en distintos momentos.
- Sin códigos de condición:
  - Instrucciones que comparan registros, y guardan resultado en otro registro.
  - Saltos que realizan la comparación entre registros.

Consideraciones sobre los saltos

- La mayoría de los destinos de los saltos son muy cercanos a la propia instrucción.
- Por ello, es muy utilizado el modo de direccionamiento relativo a PC.
  - Con al menos 8 bits para desplazamiento.
    - Permite saltar 128 **instrucciones** hacia adelante o hacia atrás.
  - *¿Qué hacemos si queremos saltar más lejos?*
- En programas de enteros, el salto condicional por igual/distinto es por lejos el más usado.
  - Hacer que se ejecute más rápido.

Dimensiones del ISA

- Los ISAs más comunes son de 32 bits.
  - Este es el tamaño de los operandos enteros, de las Instrucciones, de las direcciones a memoria y de los registros.
- También hay ISAs más pequeños (Thumb).
  - Pero internamente suelen tener muchas cosas de 32 bits.
  - O directamente son una combinación de 16 y 32 bits (Thumb2).
- También hay ISAs de 64 bits, más “recientes”.
  - Expanden todos los tamaños a 64 bits.
  - x86_64 es del 2003, ARMv8 es del 2015.
  - **Principal motivación: direccionamiento a memoria.**
    - *“Solamente hay un error que se puede cometer en diseño de computadoras que es difícil de reponerse: no tener suficientes bits para el manejo y direccionamiento de la memoria”* (Gordon Bell, 1976).
- ¿ISAs de 128 bits? Aún no, pero…

Métricas para diseño de ISA

- En casi todos los casos, vimos la importancia de identificar y medir las variables.
  - Una métrica adecuada provee buenos objetivos de diseño.
  - **No se puede mejorar lo que no se puede medir.**
- Además, los procesadores para laptops / desktops / servidores / celulares / sistemas embebidos son esencialmente distintos.
  - Tienen distintos requerimientos de Performance, de Consumo de Energía y de Área (Costo).
  - Inclusive, en un sistema embebido uno conoce exactamente el código a ejecutar, por lo que idealmente podría seleccionar cuáles características de un ISA incluir.

Métricas para diseño de ISA

- Es importante definir métricas para comparar diferentes ISAs:
  - Tamaño del programa en Memoria.
  - Cantidad de Instrucciones que se ejecutan.
  - Cantidad de accesos/transferencias a memoria.
  - Cantidad de ciclos promedio por instrucción.
  - *¿Cuál es mejor métrica?*

Métricas para diseño de ISA

- **Performance** es la métrica más importante, pero hoy en día también es importante considerar estas otras métricas:
- **Costo:** un ISA pequeño y simple favorece implementaciones pequeñas.
- **Simplicidad:** menos tiempo y costo de implementación, menor documentación, mejor curva de aprendizaje para clientes.
- **Aislamiento de Arquitectura e Implementación:** No favorecer implementaciones particulares en detrimento de implementaciones futuras.
- **Espacio para crecer:** nuevas aplicaciones requieren nuevas instrucciones.
- **Tamaño del programa:** crítico en sistemas embebidos.
- **Facilidad de programar, compilar y linkear:** una buena cantidad de GPR; direccionamiento relativo al PC; duración de las instrucciones.

Principios de Diseño

- La simplicidad favorece la regularidad.
- Hacer más rápido el caso más común.
- Cuanto más pequeño, más rápido.
- No perjudicar a la mayoría por casos minoría.
- Un buen diseño requiere compromisos.

Resumen final

- Diseñar un ISA eficiente, en el sentido de Performance.
  - Pero considerando otras métricas según el tipo de computadora.
- Tener siempre presente los Principios de Diseño.
- Accesos a memoria en múltiplos de byte.
- Formatos de instrucción de longitud fija.
- GPR e ISA tipo Load-Store dominan.
  - Si se usa PF, con sus propios registros separados.
  - Un registro cableado a cero ayuda a una mejor eficiencia.
- Instrucciones: pocas, rápidas y simples.
- Soportar los tipos de datos más comunes.
- Modos de direccionamiento: pocos.
  - Indexado (con desplazamiento de 16 bits como máximo), Inmediato (con constantes de 8 a 16 bits), de Registros Indirecto y Relativo a PC.
- Si se busca performance, conviene no usar CCR.

Agradecimientos

- Las diapositivas de este tema fueron basadas en las realizadas por el Ing. Daniel Cohen.
- A su vez inspiradas en las clases del curso CS152 de la Universidad de Berkeley, California, USA.
  - Realizadas por los Prof. D. A. Patterson, John Lazzaro, Krste Asanovic.