Universidad de Costa Rica

Escuela de Ciencias de la Computación e Informática

CI-1220 Ensambladores y Microprocesadores

Arquitectura RISC

Autores:
Emmanuel Arias
Jimena Machado
Mauricio Mangel
María Azofeifa

Profesor:
Ricardo Villalón

27 de noviembre, 2014

Introducción

En este resumen vamos a hablar acerca de la arquitectura de procesadores RISC. Primero desarrollaremos un poco la historia de como nació frente a una arquitectura ya establecida. Con esto empezamos a ver el contraste entre ambas tecnología que ampliaremos en el transcurso del documento. Hablaremos también de los detalles técnicos de la arquitectura así como la comparación de ventajas y desventajas de las arquitecturas RISC y CISC. Por último proponemos un pequeño ejemplo donde compararemos un mismo programa escrito en lenguaje ensamblador para CISC y otro para RISC.

Historia de RISC

Al tratar el tema de la arquitectura RISC es necesario hablar de CISC, por lo que estaremos haciendo comparaciones entre ambos a través del resumen. Las siglas CISC que se refieren a la arquitectura de computadoras con conjunto de instrucciones complejas y RISC la contraposición, cuyas siglas se refieren a un conjunto de instrucciones simples.

Al inicio la memoria era muy cara al igual que el acceso a ella (aún lo es) (Aletan S., 1992), debido a esto CISC se encargaba de usar la dado que era barato sustituir funciones, las cuales antes se realizaban mediante una serie de instrucciones complejas que entendían y ejecutaban el microprograma. Pero los microprogramas empezaron a crecer. Entonces surge el miedo de que los diseños existentes alcanzaran los límites. Por lo tanto empezaron a surgir ideas como que un tamaño más pequeño podría dar un mejor rendimiento, al operar a más altas velocidades de reloj, la canalización (leer la instrucción que sigue justo cuando termine la última) y ejecutar en paralelo varios procesos. Estas ideas influenciaron la creación de RISC, el cual era apto para ellas, por su simple lógica.

El proyecto RISC comenzó en 1975 en el Centro de Investigación Watson de International Business Machines Corp TJ bajo la nombre de la 801. Este proyecto no era conocido por el mundo exterior de IBM hasta principios 1980. Surge como una alternativa a la arquitectura CISC. La idea de de esta nueva arquitectura era generar código eficiente con instrucciones más simples. Tanto en software (instrucciones) como en hardware (registros de memoria). John Cocke un investigador que en la época que trabajaba para IBM es considerado el padre de la arquitectura RISC, ya que el diseño se vio influenciado por sus ideas. Otros dos proyectos con objetivos similares se iniciaron en el principios de 1980 en la Universidad de California en Berkeley y la Universidad de Stanford (Oklobdzija V., 1999).

Debido a que CISC existía desde las más tempranas computadoras, la llegada de RISC causó estragos.

> "Los defensores de RISC afirmaron que podían obtener más potencia de cálculo para una cantidad dada de hardware a través de una combinación de diseño conjunto de instrucciones simplificado, avanzado

> tecnología de compilación y ejecución pipeline del procesador. Los de CISC afirmaban que podían tener más poder computacional por una cantidad de hardware a través de una combinación de diseño conjunto de instrucciones simplificado, por lo que sus máquinas podrían lograr un mayor rendimiento general" (Bryant & O'Hallaron).

Pero con el tiempo se dieron a conocer las deficiencias, o que sus características planteadas inicialmente no eran tan exactas a sus afirmaciones. RISC introdujo más instrucciones, muchas de las cuales llevan varios ciclos para ejecutar. Hoy tienen cientos de instrucciones en su repertorio, lo cual no concuerda tanto con su nombre. Aún así, el núcleo del diseño es un conjunto de instrucciones que se adapta bien a la ejecución en una máquina pipeline (técnica de implementación donde múltiples instrucciones se solapan en ejecución).

Las grandes empresas introdujeron RISC, líneas de procesadores, incluyendo Sun. Los productos ARM. Procesadores RISC han hecho muy bien en el mercado de los procesadores integrados, control de tales sistemas como los teléfonos celulares, los frenos de automóviles, aplicaciones de Internet y supercomputadoras. En estas aplicaciones, el ahorro en el precio y el poder es más importante que mantener la compatibilidad hacia atrás, y RISC lo cumple. Ambos han evolucionado mucho, adoptando mejoras provenientes del contrario y adaptándose a los usos de los ordenadores. Si bien es cierto que en la época de su creación la diferencia era muy amplia "en la actual informática moderna los requisitos son muy diferentes. Los límites en capacidad de almacenamiento son casi inexistentes y los procesadores son capaces de ejecutar millones de instrucciones en un solo segundo" (Espeso P., 2012).

Principios de diseño

La propuesta RISC fue una respuesta a la tecnología cambiante y a la acumulación de conocimientos a partir de los diseños CISC. Para entender esta arquitectura vamos a manejarlo como un concepto para diseñar procesadores. Aunque los procesadores RISC iniciales tenían menos instrucciones en comparación con CISC, la nueva generación de procesadores RISC tiene cientos de instrucciones algunas tan complejas como las CISC. (Dandamudi S., 2005) A pesar de que el objetivo principal no era reducir el número de instrucciones, sino la complejidad, el término se ha pegado.

No existe una definición precisa de lo que constituye un diseño RISC pero aún hay conceptos que se mantienen (Flynn M., 1995):

Operaciones simples El objetivo es diseñar instrucciones simples que cada una se pueda ejecutar en un solo ciclo (tiempo requerido para buscar a 2 operandos de los registros, realizar una operación, y guardar el resultado en un registro), este proceso simplifica el diseño del procesador.

Sistema registro-registro Los procesadores RISC sólo permiten operaciones de cargar / almacenar que permiten el paralelismo para acceder a la memoria (registro-memoria). El resto de las operaciones funciona sobre una base

de registro a registro. Esto hace que el modelo de las instrucciones de asignación se reduzcan a un ciclo. La restricción de operandos a registros también simplifica la unidad de control.

Modos simples de direccionamiento Los diseños RISC permiten pocas formas de direccionar memoria. Con esto los desplazamientos se hacen mucho más simples.

Conjunto amplio de registros Como las instrucciones funcionan de registro a registro se crearon un gran número de estos. Esto ofrece amplias oportunidades para el compilador. Se puede optimizar el programa usando al máximo los registros evitando el uso de memoria para llamados de procedimientos y uso en general de variables.

Longitud fija, formato simple de instrucciones Los diseños RISC utilizan instrucciones de longitud fija y con un formato simple. Los límites de los campos en una instrucción como códigos de operación y fuente de operandos son fijos. Esto permite que la decodificación y la programación de instrucciones sea más eficiente. Todas las arquitecturas RISC de 32 bits normalmente utilizan instrucciones de 32 bits de tamaño. Algunos ejemplos son la SPARC, MIPS, ARM y PowerPC. (Flynn M., 1995)

Procesadores y microprocesadores basados en RISC

Durante el transcurso de esta arquitectura han sido muchos los diseños de procesadores y microprocesadores que han adoptado esta tecnología. Entre los más importantes destacamos:

PowerPC (PPC) Es una arquitectura fundada por la alianza de Apple-IBM-Motorola (AIM). Creado originalmente para computadoras personales, se popularizaron en procesadores de alto rendimiento, usados en las Macintosh de Apple, su uso en consolas de videojuegos y centrado en aplicaciones presentó múltiples usos. (IBM, 2005)

DEC Alpha Originalmente conocido como Alpha AXP, es un conjunto de instrucciones de 63 bits desarrollado por Digital Equipment Corporation (DEC), como reemplazo a la serie VAX. Diseñado especialmente para el cálculo de punto flotante. (Compaq Computer Corporation, 1998)

MIPS (Microporcessor without Interlocked Pipeline Stages) Desarrollados por MIPS Technologies. Los diseños fueron utilizados por la línea de productos informáticos de SGI y actualmente en dispositivos como: Windows CE, routers, gateways residenciales, y consolas de videojuegos como la Sony PlayStation, PlayStation 2 y PlayStation Portable. Debido a que los diseñadores crearon un conjunto de instrucciones claras, que hace programas que ocupan menos espacio y con capacidad multihilo se usa en los cursos sobre arquitectura de computadores en universidades y escuelas técnica. (Patterson & Hennessy, 2004)

ARM Es una arquitectura de 32 bits desarrollada por British ARM Holdings. Diseñados con menos transistores lo que reduce los costos, el calor y el

uso de energía. Útiles para dispositivos portátiles incluyendo teléfonos inteligentes, computadoras portátiles, tabletas etc. Los fabrican Apple, AppliedMicro, Atmel, Broadcom, Nvidia, NXP, Qualcomm, Samsung Electronics, STMicroelectronics, entre otros. A nivel mundial ARM es el conjunto de instrucciones arquitectura más utilizado en términos de cantidad producida. (Yiu J., 2013)

Diferencias entre CISC y RISC

CISC (Complex instruction set computing) fue desarrollado basado en las limitaciones de la época. Dado que se programaba principalmente en lenguaje de ensamblador, tener instrucciones más complejas facilitaba esto. También intentaba tener todos los tipos de direccionamiento posible. Además, tomaba en cuenta el costo muy alto y la velocidad muy limitada de la memoria, por lo que esto ayudaba a limitar la cantidad de veces que era necesario accesar la memoria. De esta forma, al tener instrucciones más complejas implementadas con el hardware, los programas se hacían más cortos y tomaban menos espacio.

Las limitaciones que se enfrentaban significaban que por un momento CISC era la única opción razonable, por lo cual este nombre no apareció hasta después. Fue cuando surgió RISC (Reduced instruction set computing) facilitado por los avances que hacían la memoria más rápida y barata que se decidió este nombre y se creó esta división. RISC se desarrolló con las limitaciones de CISC en mente, por ejemplo:

- Los compiladores generalmente ignoran instrucciones más complejas, por lo que no llegan a tener mucha utilidad.
- Las estructuras de datos más complejas no se usan frecuentemente, y se pueden crear usando estructuras simples cuando son necesarias.
- Las instrucciones de longitud y tiempo de ejecución variable hace que decodificar y planificar sea ineficiente.

RISC	CISC
⋆ Busca menor cantidad de ciclos de reloj por instrucción ⋆ Instrucciones simples para tomar solo un ciclo de reloj ⋆ Instrucciones de longitud definida ⋆ Hay instrucciones independientes para accesar la memoria y mover los datos guardados ahí a registros ⋆ Más espacio físico dedicado hacia un mayor número de registros ⋆ Cantidad limitada de tipos de direccionamiento ⋆ Duración de instrucciones facilita pipelining	⋆ Busca menor cantidad de instrucciones por programa ⋆ Instrucciones más complejas que toman varios ciclos ⋆ Instrucciones de longitud variable ⋆ Se incorpora el acceso a la memoria dentro de otras instrucciones ⋆ Más espacio físico dedicado a guardar las instrucciones más complejas ⋆ Solía ser predominante por límites de memoria ⋆ Facilitar el trabajo del compilador ⋆ Intenta cubrir muchos tipos de direccionamiento

Table 1: Contraste entre RISC y CISC

A pesar de todo esto, se ve en casos recientes que la distinción entre RISC y CISC no implica diferencias importantes en eficiencia. Estas son causadas por otros aspectos.

Lenguaje ensamblador Cool

Pare tener un ejemplo claro de las características de un lenguaje ensamblador para un procesador basado en un conjunto de instrucciones RISC proponemos hacer una comparación de código. Por un lado vamos a utilizar el lenguaje ensamblador que hemos venido trabajando en el curso. Para obtener el código ensamblador RISC vamos a usar el lenguaje Cool que nos dará la opción de generar el código de ensamblador Cool del programa.

El lenguaje de programación Cool, *Classroom Object-Oriented Language* fue creado para utilizarse en un curso de la Universidad de Virginia. Desarrollado por Alex Aiken para ser una herramienta simple pero poderosa a la vez para iniciar a los estudiantes en programación. Cool es un subconjunto de Java que contiene todos los elementos característicos del paradigma orientado a objetos. Escogimos este lenguaje porque nos permite generar el código ensamblador del programa de una manera similar a la de gcc.

La estructura del lenguaje ensamblador Cool es fácil de comprender para alguien familiarizado con el lenguaje de ensamblador para arquitectura CISC que hemos estado utilizando en el curso. Cool posee 11 registros en total más

un conjunto de instrucciones muy pequeño y que podemos mostrar en la Tabla 2.

Como vemos este lenguaje (y en general la arquitectura RISC) comparte instrucciones básicas como operaciones aritméticas, operaciones de cambio de flujo, manejo de pila con push y pop, registro de propósito general y especiales para manejo de los marcos de pila. Sim embargo podemos ver que en el ensamblador Cool las operaciones de mover un dato de registro a memoria y de registro a registro están por aparte. En este lenguaje todos tiene tamaño 32 bits, osea no existen los byte, word, dword etc que conllevan un mayor número de operaciones y circuitería.

Las interrupciones del ensamblador Cool son pocas. Estas son: exit, in_string, in_int, out_int, out_string, String.length, String.concat, String.substr. Los registros que usan son r0, r1 y r2. Como vemos aquí no hay códigos de instrucción como en el ensamblador regular.

Ahora si, teniendo un poco más claro las instrucciones y demás detalles del ensamblador Cool podemos ver código. Primero veamos el ejemplo en C con el que vamos a comparar.

```
#include <stdio.h>
main(){
 printf("%s", "Hola Mundo!");
 return 0;
}
```

Este es el clásico Hola mundo que uno espera como ejemplo básico. En el lenguaje Cool vamos a escribir uno similar:

```
class Main inherits IO {
 main() : Object {
 out_string("Hola Mundo!")
};
};
```

En este la clase Main necesita heredar de IO para poder interactuar con la consola. Veamos ahora sus códigos ensamblador. El del programa en C lo obtenemos usando el programa objdump que viene incluido en las herramientas de GNU. El código obtenido para la sección de main es este:

```
080483b4 <main>:
80483b4:    55    push %ebp
80483b5:    89 e5    mov %esp,%ebp
80483b7:    83 e4 f0    and $0xfffffff0,%esp
80483ba:    83 ec 10    sub $0x10,%esp
80483bd:    c7 04 24 94 84 04 08    movl $0x8048494,(%esp)
80483c4:    e8 27 ff ff ff    call 80482f0 <puts@plt>
80483c9:    b8 00 00 00 00    mov $0x0,%eax
80483ce:    c9    leave
```

```
80483cf:    c3    ret
```

Este código no lo vamos a explicar puesto que suponemos que el lector tiene conocimientos del lenguaje ensamblador. Solo cabe destacar que $0x8048494 apunta a la hilera que se va a imprimir. El código ensamblador del programa en Cool es el siguiente:

```
Main.main:
                mov fp <- sp
                pop r0
                li r2 <- 2
                sub sp <- sp r2
                push ra
Main.main.0:
                jmp Main.main.1
Main.main.1:
                la r1 <- l3
                ld r1 <- r1[3]
                syscall IO.out_string
                mov r1 <- r0
                st fp[0] <- r1
                jmp Main.main.2
Main.main.2:
                ld r1 <- fp[0]
                st fp[0] <- r1
                jmp Main.main.end
Main.main.end:
                pop ra
                li r2 <- 2
                add sp <- sp r2
                return
```

Veamos este ejemplo por partes. El prólogo y el epílogo son muy similares entre ellos. Las operaciones de sumarle y restarle a la pila son iguales a excepción que podemos que no es permitido hacer la operación *sub sp <- sp 2* porque coo ya hemos visto las operaciones son registro-registro. En consecuencia primero debemos cargar el entero a un registro y luego realizar la operación debida. Eso se ve reflejado también en las operaciones de almacenar en la memoria.

En el caso de Cool, el manejo de hileras en consola se hace por medio de una interrupción. Nótese que la clase Main hereda de una clase IO que contiene el método out_string. Uno espera que out_string se vea reflejado en ensamblador como un call a una función pero más bien se maneja a nivel de interrupción.

A nivel de hileras podemos ver que en el ensamblador normal la dirección $0x8048494 apunta a una hilera en el programa. En el lenguaje Cool se hace referencia a una etiqueta l3, que luego se direcciona sumándole 4 posiciones.

Esto hace referencia a la etiqueta l5 que a su vez punta a la hilera que se desea imprimir.

Las similitudes entre ambas arquitecturas son mayores en cuanto a la estructura básica. Sin embargo, las instrucciones CISC son más robustas y complejas mientras que en RISC al ser un conjunto de operaciones más pequeño la tarea de programar se vuelve un poco más tediosa. Si se desea ver más ejemplos sobre el lenguaje Cool puede visitar este enlace http://www.cs.virginia.edu/ cs415/cool.html para descargar el compilador de Cool y código de ejemplo. También puede ser útil el manual de referencia del lenguaje ensamblador Cool http://www.cs.virginia.edu/ cs415/cool-manual/node51.html.

References

[1] Aletan S. (1992). *An Overview of RISC Architecture* New York: ACM

[2] Blem E., Menon J., Sankaralingam K. (2013). *Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures* IEEE International Symposium on High Performance Computer Architecture.

[3] Bryant R., O'Hallaron D. (2003). *Computer Systems A Programmer's Perspective* Pearson Education

[4] Dandamudi, S. (2005). *Guide to RISC processors.* USA: Springer

[5] Espeso, P (2012). *CISC frente a RISC, una batalla en blanco y negro.* Xataka

[6] Flynn, M. (1995). *Computer Architecture: Pipelined and Parallel Processor Design.* USA: Jones and Bartlett Publishers

[7] IBM (2005). *PowerPC User Instruction Set Architecture. Book I*

[8] Jain P. (2012). *RISC & CISC architecture* Engineer's Garage.

[9] Oklobdzija V. (1999). *Reduced Instruction Set Computers.* California: Integration Berkeley

[10] Patterson D., Hennessy J. (2004). *Estructura y diseño de computadores.* España: Reverté

[11] Yiu, J. (2013). *The Definitive Guide to the ARM Cortex-M3 and Cortex-M4 Processors.* Newnes

[12] Weimer W. (s.f.). *The Cool Reference Manual* http://www.cs.virginia.edu/ cs415/cool-manual/cool-manual.html

[13] *CS 4610 - Programming Languages - Cool* http://www.cs.virginia.edu/ cs415/cool.html

Tipo	Operación	Descripción
registro	r0	registro de propósito general #0, usualmente se usa como acumulador
registro	r1	registro de propósito general #1
registro	r2	
registro	r3	
registro	r4	
registro	r5	
registro	r6	
registro	r7	
registro	sp	puntero a la pila
registro	fp	puntero marco pila
registro	ra	dirección de retorno
instrucción	li registro <- integer	cargue en registro
instrucción	mov registro <- registro	copia de registro a registro
instrucción	add registro <- registro registro	suma
instrucción	sub registro <- registro registro	resta
instrucción	mul registro <- registro registro	multiplicación
instrucción	div registro <- registro registro	división
instrucción	jmp label	salto incondicional
instrucción	bz registro label	salte si es igual a cero
instrucción	bnz registro label	salte si no es cero
instrucción	beq registro registro label	salte si son iguales
instrucción	blt registro registro label	salte si menor que
instrucción	ble registro registro label	salte si menor o igual que
instrucción	call label	llamado directo a función
instrucción	call registro	llamado indirecto función en registro
instrucción	return	retorno de función
instrucción	push registro	empuje un valor a la pila
instrucción	pop registro	saque un valor de la pila
instrucción	ld registro <- registro [integer]	cargue un dato de memoria
instrucción	st registro [integer] <- registro	almacene un dato en la memoria
instrucción	la registro <- label	cargue dirección en registro
instrucción	alloc registro registro	asiganción memoria
instrucción	syscall name	interrupción para un servicio de sistema operativo

Table 2: Registros e instrucciones del ensablador Cool (Weimwe W., s.f)