Arquitecturas RISC

v.2014

William Stallings, *Organización y Arquitectura de Computadores, Capítulo 12: Computadores de repertorio reducido de instrucciones.*

John Hennessy – David Patterson, *Arquitectura de Computadores – Un enfoque cuantitativo 1ª ed, Capítulos 1 a 6 y Apéndice E: Visión general de las arquitecturas RISC (3ª ed Ap C,* ***4ª ed Ap J****).*

Avances importantes en microprocesadores desde 1950

- **Unidad de control microprogramada y el concepto de familia** (1964): compatibilidad ISA con diferentes implementaciones [IBM 360].
- **Memoria cache** (1968) [IBM 360].
- **Segmentación del cauce de instrucciones** (pipeline).
- **Arquitecturas RISC (1985)**: involucra la mayoría de los aspectos importantes relacionados con la arquitectura y la organización.

No existe en la actualidad una tecnología
que haga prever un impacto semejante.
Múltiples procesadores?

Arquitecturas RISC

La brecha semántica

- El costo del hardware baja (evolución tecnológica) y el costo del software sube (poco confiable, difícil de mantener).
- Como una solución aparecen los lenguajes de programación de alto nivel (HLL): mayor nivel de abstracción (programación estructurada vs. OO, más concreto y más poderoso), pero aumenta la brecha semántica.
- Cuál es la forma más eficiente de generar lenguaje máquina?
  - Aproximación CISC: ISAs complejas, con muchas instrucciones y modos de direccionamiento, incluyendo instrucciones de HLL. Compiladores simples.
  - Aproximación RISC: simplificar el set de instrucciones adaptándolo a los requerimientos REALES de los programas (estudios). Cambio de estrategia.

Arquitecturas RISC

Principales características

- Arquitectura de carga/almacenamiento, con pocos modos de direccionamiento.
- Un gran número de registros de uso general, cuya utilización se optimiza en el compilador.
- Repertorio de instrucciones limitado, sencillo y con formato fijo
- Especial énfasis en la segmentación y unidad de control cableada.

Arquitecturas RISC

Algunas arquitecturas

	Complex Instruction Set (CISC) Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2–6	2–57	1–11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40–520	32	32	40–520	32
Control memory size (kbits)	420	480	246	—	—	—	—	—
Cache size (kbytes)	64	64	8	32	128	16–32	32	64

Arquitecturas RISC

Orígenes (1980-1990)

- John L. Hennessy (Stanford) -> arquitectura MIPS (Microprocessor without Interlocking Pipeline Stages)
  - MIPS Technology Inc. Diseñadores de los procesadores RXXXX; fabricados por Nec, Toshiba y SGI; utilizados en PlayStationII, Nintendo64, CISCO, SGI.
- David A. Patterson (Berkeley) -> arquitecturas RISC I y II
  - Conduce al diseño de la arquitectura SPARC fabricada por Sun Microsystems. Workstations.
- Juntos -> arquitectura DLX (H-P)
  - Base para éstas y otras arquitecturas (Alpha de Dec, 88000 de Motorola, 29000 de AMD o i960 de Intel). Pocas diferencias!

Evaluación dinámica de los programas en tiempo de ejecución

Qué hacen los programas escritos en HLL la mayor parte del tiempo? [Patterson, 1982]

- **Frecuencia dinámica de instrucciones**:
  - Movimiento de datos 43%
  - Control de flujo 23%
  - Operaciones aritméticas y lógicas 20%
  - Comparaciones 13%
  - Otras 10%

Del recuento de instrucciones y referencias a memoria resulta un gran peso de las **llamadas a funciones** de los lenguajes estructurados (salvaguarda del estado y pasaje de parámetros).

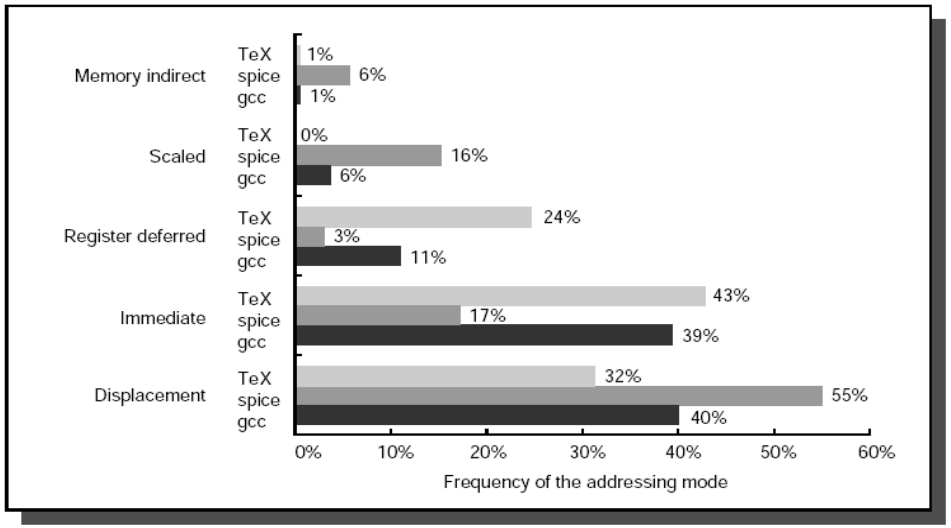
Arquitecturas RISC

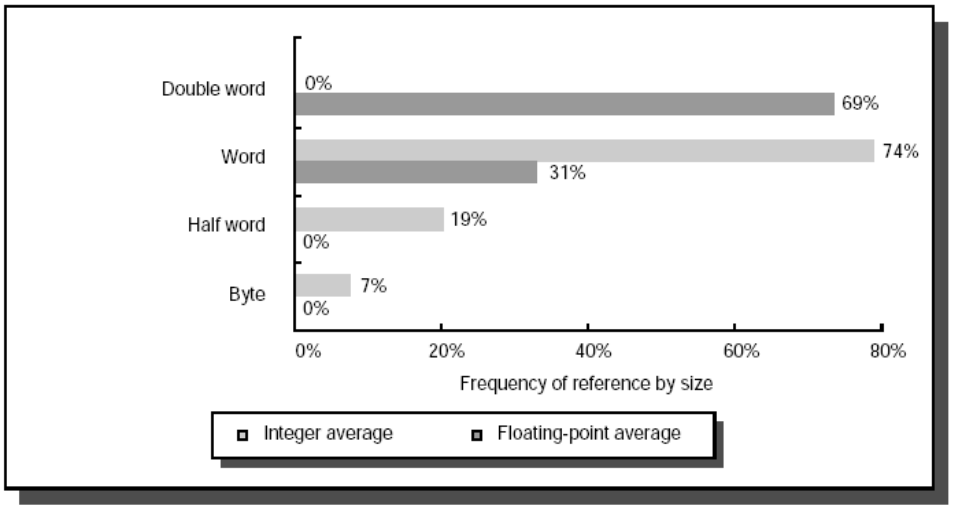
Evaluación dinámica (cont)

- **Tipos de operandos:** 60% variables escalares (enteros, reales, char, etc.), de las cuales el 80% son locales.
- **Llamadas a procedimientos:** gran cantidad de accesos a memoria. El 98% usa menos de seis argumentos, de los cuales el 92% usa menos de seis variables escalares locales. Pocos niveles de anidamiento (menor que seis). Concepto: referencia a operandos localizada. 6/6/6
- **Modos de direccionamiento:** solo el 18% de las instrucciones utiliza modos sofisticados. La mayoría utiliza modos que se resuelven en un ciclo (registro, desplazamiento).

Arquitecturas RISC

Evaluación dinámica (cont)





Rank	80x86 instruction	Integer average (% total executed)
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
Total		96%

FIGURE 2.11 The top 10 instructions for the 80x86. These percentages are the average of the same five SPECint92 programs as in Figure 2.7 on page 77.

Arquitecturas RISC

Evaluación dinámica (cont)

Conclusiones de las mediciones:

- Preponderancia de operaciones y modos de direccionamiento simples **-> ISA REDUCIDO**.
- Los operandos son mayormente escalares locales **-> REGISTROS** y **LOAD/STORE**.
- Importancia de llamadas a procedimientos **-> VENTANAS**.

Además es clave un pipeline eficiente -> **ANCHO DE INSTRUCCIONES FIJO** e **INSTRUCCIONES SIMPLES.**

Si resulta...

- **CONTROL CABLEADO**, puede aumentarse $\mathbf{f_{CLOCK}}$ **y por lo tanto mayor RENDIMIENTO**

Principales aspectos RISC

(el repertorio de instrucciones reducido y sus consecuencias en la organización)

- Pocas instrucciones simples
- Arquitectura load/store
- Pocos modos de direccionamiento
- Instrucciones de ancho fijo
- Gran número de registros

1. Pocas instrucciones simples

- El objetivo es que ejecuten, de ser posible, en un único ciclo de máquina (luego de ser captadas y decodificadas, por supuesto). Pipeline de tres etapas para las instrucciones sin referencia a memoria: FI – DI – EI.
- Puedo utilizar control cableado porque son simples.
- La unidad de control es simple, por lo tanto puede funcionar con mayor frecuencia de clock.
- El pipeline es eficiente si las instrucciones son de tiempo de ejecución similar en cada etapa.
- Las instrucciones complejas de los CISC se implementan como una secuencia de operaciones RISC. {**EJEMPLO}**

2. Arquitectura load/store

- Las únicas instrucciones que referencian datos en memoria son las de carga y almacenamiento. Las demás operan solo con registros.
- Los accesos a memoria no pueden hacerse en un ciclo. Cuatro etapas en el pipeline para las instrucciones con referencia a memoria: FI – DI – CA (calcular dirección) – TR (transferir). La instrucción siguiente debe retardarse. Ver ***delayed load***.

- Discusión sobre la eficiencia de los programas resultantes si me interesa el tráfico con memoria y la densidad de código resultante (no confundir con programa simbólico más corto)

8	16	16	16
Add	B	C	A

Memory-to-Memory
I = 56, D = 96, M = 152

8	4	16		
Load	rB	B		
Load	rC	B		
Add	rA	rB	rC	
Store	rA	A		

Register-to-Memory
I = 104, D = 96, M = 200

(a) A ← B + C

8	16	16	16
Add	B	C	A
Add	A	C	B
Sub	B	D	D

Memory-to-Memory
I = 168, D = 288, M = 456

8	4	4	4
Add	rA	rB	rC
Add	rB	rA	rC
Sub	rD	rD	rB

Register-to-Memory
I = 104, D = 96, M = 200

(b) A ← B + C; B ← A + C; D ← D – B

I = Size of executed instructions
D = Size of executed data
M = I + D = Total memory traffic

3. Pocos modos de direcciona-miento

- Además de los modos **registro** e **inmediato**, el principal modo de direccionamiento (y casi siempre el único) es el de **desplazamiento** (offset), con registro de 32 bits más un desplazamiento de 16 bits, solo utilizado para LOAD/STORE.

  MIPS: lw r2,128(r3)

- Se puede utilizar para simular el modo **directo** o **absoluto**, muy utilizado para acceder a datos estáticos, usando r0.
- También se puede simular el modo **indirecto con registro** haciendo el desplazamiento nulo.
- **IMPORTANTE:** Se evitan los modos que necesiten acceder a memoria para componer la dirección de un operando (indirecto con memoria).

4. Instrucciones de ancho fijo y formato uniforme

- Usualmente instrucciones de ancho fijo de 32 bits.
- Esto hace que la captación y decodificación de instrucciones sea simple y rápida. No se necesita esperar a que se conozca el largo de la instrucción actual para empezar a decodificar la próxima.
- El formato uniforme simplifica la decodificación porque el código de operación y el campo de dirección están ubicados en la misma posición para todas las instrucciones.

Ver luego la codificación del repertorio de instrucciones de DLX:

Tipo I (carga y almacenamiento)

Tipo R (registro-registro)

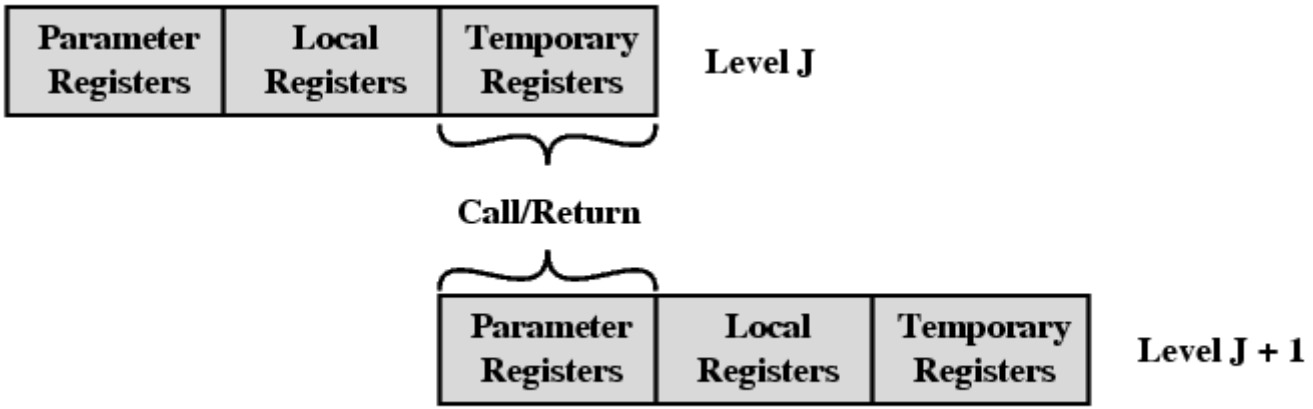
Tipo J (saltos)

5. Gran número de registros

- Las variables y los resultados intermedios pueden ser almacenados en registros evitando repetidos accesos a memoria.
- Los registros pueden utilizarse para el pasaje de parámetros a procedimientos, evitando hacerlo por la pila (memoria). Recordar medidas 6/6/6.
- Puede utilizarse un grupo de registros para cada procedimiento, evitando tener que salvar el estado.
- Es posible implementar muchos registros porque se redujo la complejidad de la unidad de control.
- La arquitectura no limita la utilización de los registros. Son de propósitos generales. Sin embargo el compilador debe organizarse de alguna manera, según su conveniencia (por ejemplo si implementa un stack).

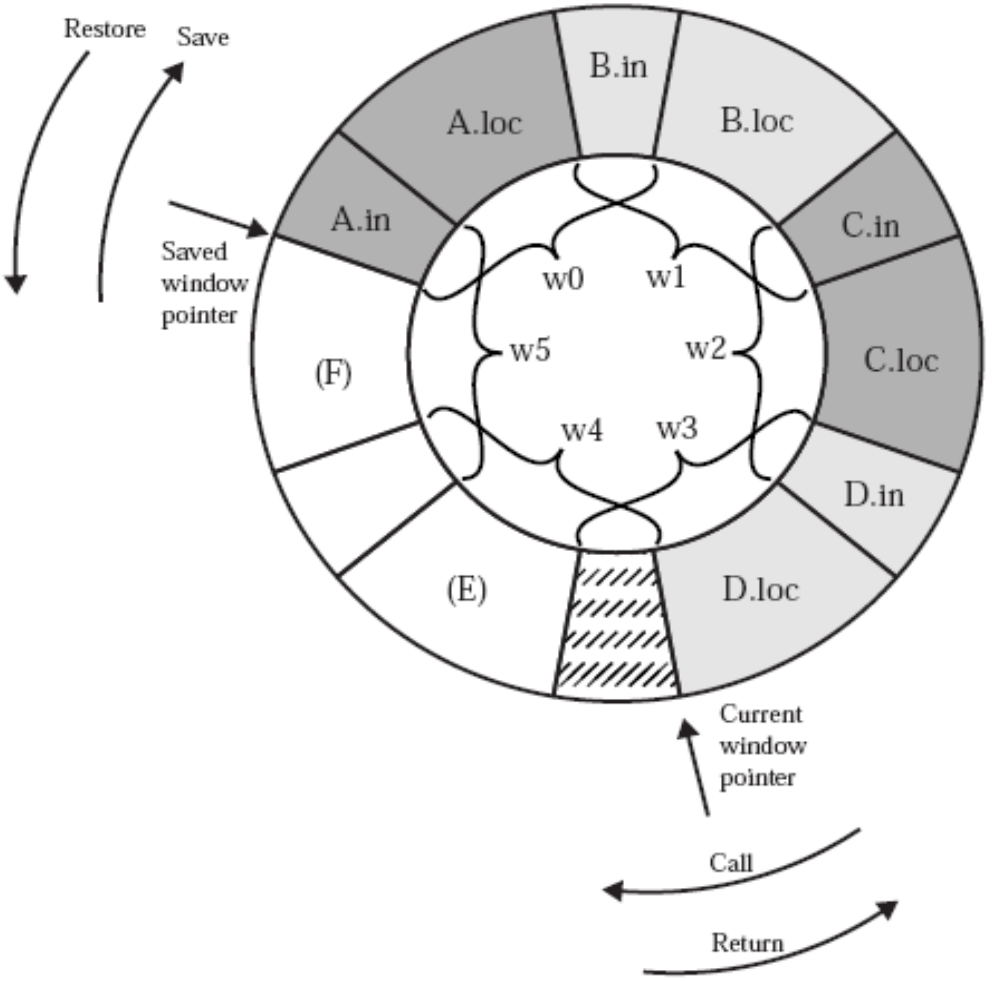
Ejemplo: Implementación de ventanas de registros

- Es una forma de optimizar la utilización de un gran número de registros ante la presencia de llamadas a funciones originadas por la programación estructurada.
- En cada momento sólo es visible una ventana de N registros (del 0 al N-1).
- Se definen tres areas de trabajo de largo fijo superpuestas (in, local, out).
- Los parámetros a procedimientos se pasan sin transferencia real. Sólo es necesario incrementar el puntero de ventana actual (CWP).

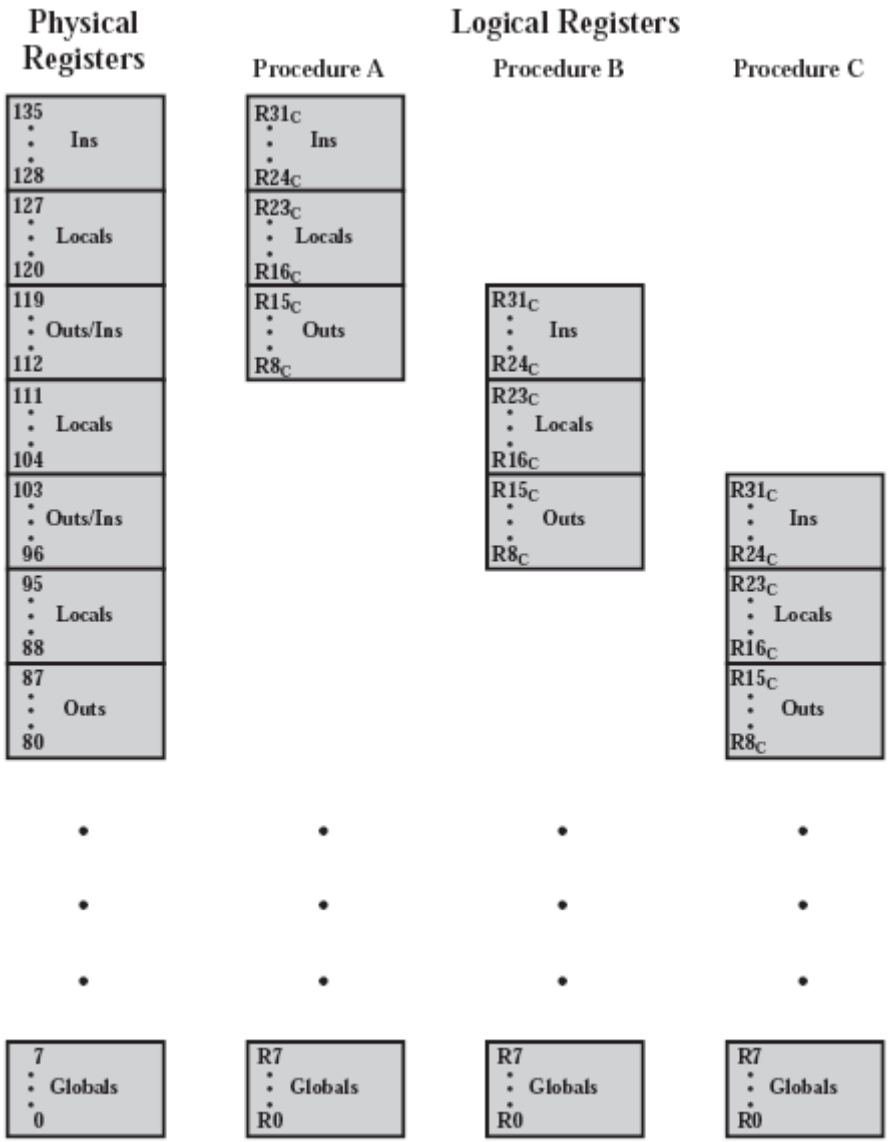


Ventana de registros (cont)

- Una llamada a procedimiento incrementa CWP.
- CWP = SWP genera interrupción, salva a memoria e incrementa SWP.
- Idem al retorno.
- Recordar 6/6/6.
- Con k ventanas k-1 proc.
- GLOBALES.



Ventana de registros del procesador SPARC



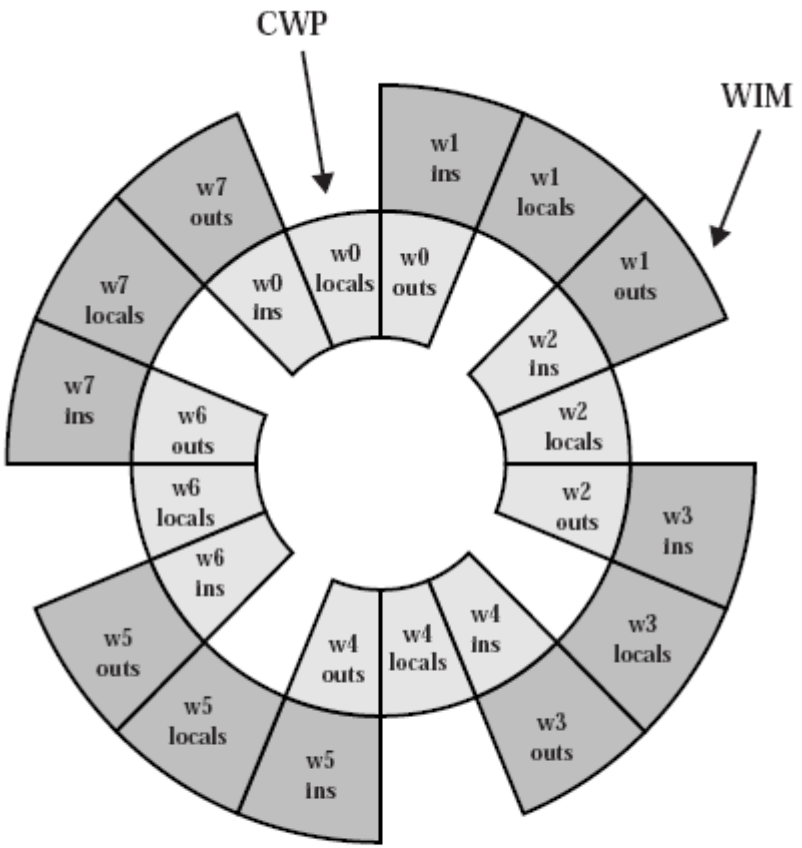
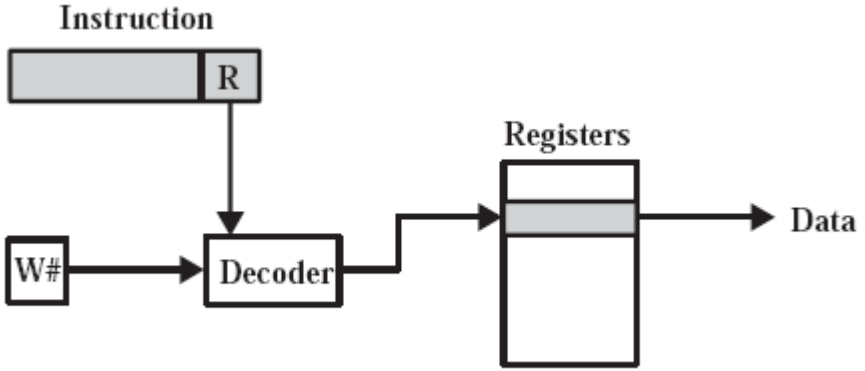


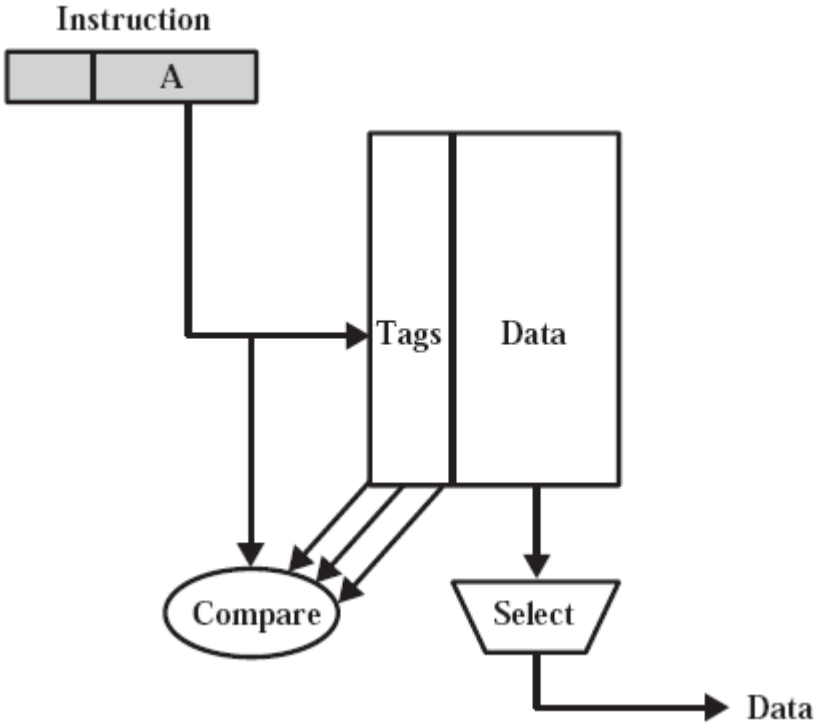
Figure 12.12 Eight Register Windows Forming a Circular Stack in SPARC

Discusión

Banco de registros vs. cache de datos

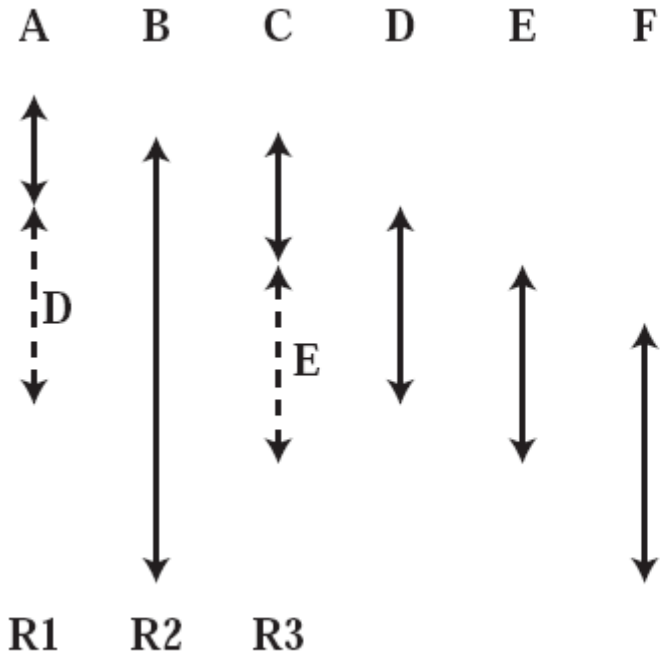
Disponibilidad, número de bits necesarios, velocidad.

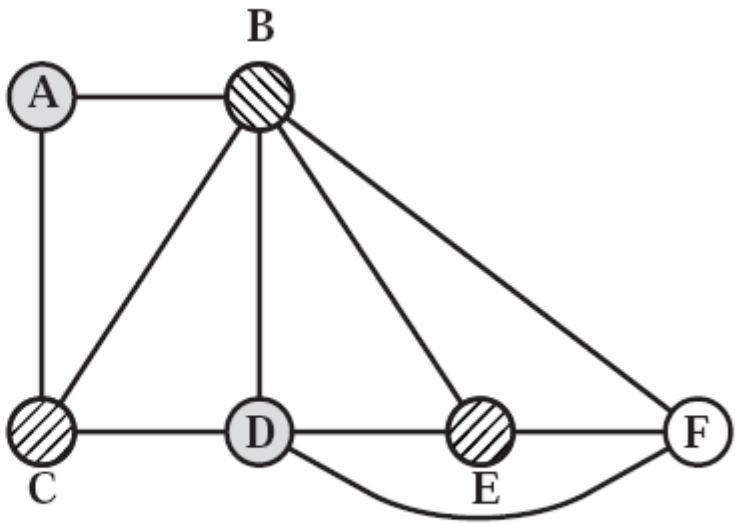




Optimización de registros basada en el compilador

- Los HLL no utilizan registros. El compilador utiliza registros virtuales y trata de adaptarlos a los registros disponibles.
- Compromiso entre disponibilidad de registros y sofisticación del compilador.
- Coloreado de grafos:





Carga retardada (delayed load)

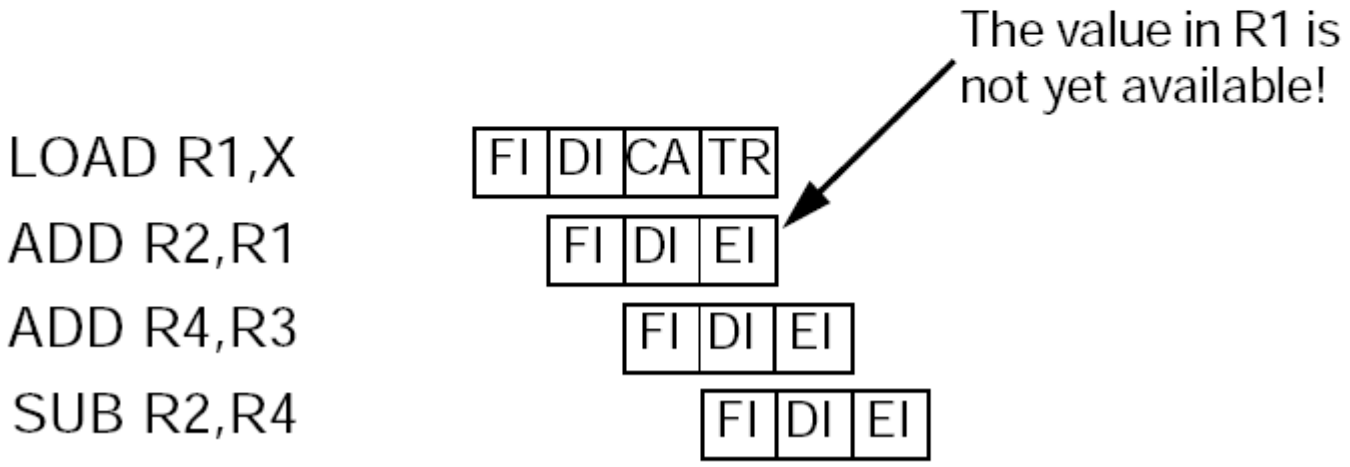
- La instrucción posterior a un LOAD debe suspenderse durante un ciclo si necesita el operando.

- La técnica de **carga retardada** (similar al salto retardado) es muy utilizada en RISC por su gran eficiencia.
- Consiste en ejecutar la instrucción siguiente sin detención. El operando no está disponible. El compilador (o el programador ASM) es responsable de las consecuencias.

Carga retardada (cont)

- En esas condiciones, el programa presentado es incorrecto (RAW hazard).

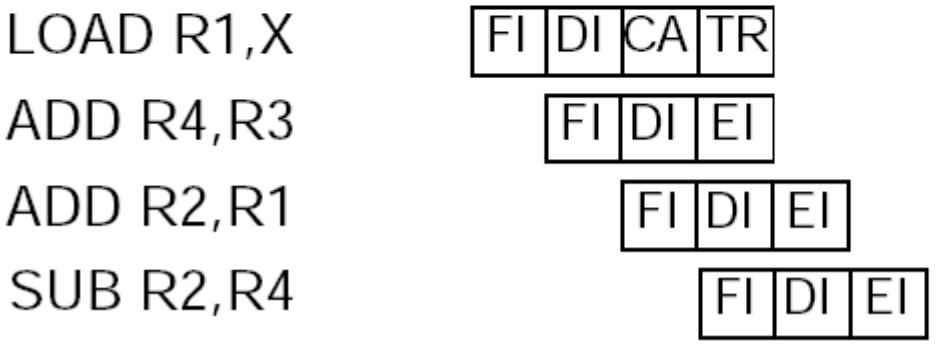
LOAD	R1,X	loads from address X into R1
ADD	**R2,R1**	**R2 ← R2 + R1**
ADD	R4,R3	R4 ← R4 + R3
SUB	R2,R4	R2 ← R2 - R4



Carga retardada (cont)

- La siguiente secuencia es correcta. Depende del compilador el encontrar la instrucción que pueda colocarse a continuación del LOAD, sin afectar el resultado del programa. Estadística: entre el 80-90% de los casos es posible.

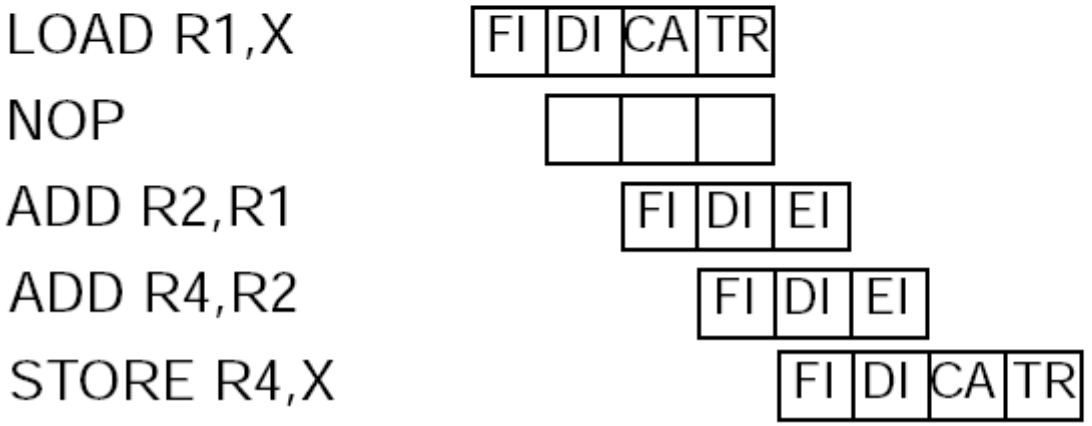
LOAD	R1,X	loads from address X into R1
ADD	R4,R3	R4 ← R4 + R3
ADD	**R2,R1**	**R2 ← R2 + R1**
SUB	R2,R4	R2 ← R2 - R4



Carga retardada (cont)

- En el caso de no conseguirla, el compilador debe insertar un NOP. El cauce no se detiene, pero se pierde una instrucción.

```
LOAD   R1,X      loads from address X into R1
NOP
ADD    R2,R1     R2 ← R2 + R1
ADD    R4,R2     R4 ← R4 + R2
STORE  R4,X      stores from R4 to address X
```



Ventajas RISC (según Patterson)

- **Oblea más pequeña**. Un diseño entra antes en una determinada tecnología de integrados. Al evolucionar la tecnología permite incorporar cache, fp, etc. Menor consumo de potencia.
- **Tiempo de desarrollo más corto**. Por lo tanto es más barato y mejor sintonizado con la tecnología.
- **Mayor performance** (discutible). Argumento: mas simple implica más rápido. Hicieron uno simple y comprobaron que al agregar instrucciones complejas desmejoraba.

Contribuciones RISC a la historia

- Segmentación del cauce de instrucciones. Es la forma más simple de concurrencia, con la cual se pueden obtener aceleraciones de 2 o 3. Esta técnica es de implementación eficiente en una arquitectura RISC.
- Ciclo de clock corto y ejecución en un ciclo. Permite al microprocesador explotar a fondo la tecnología de memorias. Nota: en la época del diseño de los CISC las memorias eran lentas, pero también los microprocesadores se hicieron más lentos al pasar de TTL a MOS por cuestiones de consumo.

Hoy es posible implementar un CISC de la década del 80 con las siguientes características: Hard-Wired, Single-Cycle, Pipelined (ejemplo ADuC8051)

Desventajas RISC

- Menor densidad de código (discutible). Importante si la disponibilidad de memoria es crítica. Ocasiona un menor rendimiento del caché de instrucciones. ARM Thumb.
- No ejecutan código x86. Puede utilizarse simulación por software, pero no puedo fabricar una IBM PC compatible.

RISC vs. CISC

La comparación es difícil por varios motivos:

- No existe una pareja de máquinas RISC y CISC comparables en costo del ciclo de vida, nivel de tecnología, complejidad de hardware, sofisticación del compilador, soporte para el sistema operativo, etc.
- No existen los benchmarks definitivos.
- Es difícil separar los efectos del hardware de los efectos debidos a la habilidad en el diseño del compilador.
- En los últimos años, dado el aumento de la densidad de integración, ha habido una convergencia de las dos tecnologías.

RISC

Any computer announced after 1985.

Steven Praybylski (MIPS Technology)

Comparación

RISC	CISC
Multiple register sets, often consisting of more than 256 registers	Single register set, typically 6 to 16 registers total
Three register operands allowed per instruction (e.g., `add R1, R2, R3`)	One or two register operands allowed per instruction (e.g., `add R1, R2`)
Parameter passing through efficient on-chip register windows	Parameter passing through inefficient off-chip memory
Single-cycle instructions (except for `load` and `store`)	Multiple-cycle instructions
Hardwired control	Microprogrammed control
Highly pipelined	Less pipelined
Simple instructions that are few in number	Many complex instructions
Fixed length instructions	Variable length instructions
Complexity in compiler	Complexity in microcode
Only load and store instructions can access memory	Many instructions can access memory
Few addressing modes	Many addressing modes

Comparación (cont)

		Inst.	Ancho	Formato	Modos	Reg.
CISC	*VAX 11/780*	*303*	*2a57*	*Variable*	*22*	*16*
CISC	*Pentium*	*235*	*1a11*	*Variable*	*11*	*8*
RISC	*PowerPC*	*206*	*4*	*Fijo*	*2*	*32*
RISC	*Sun SPARC*	*52*	*4*	*Fijo*	*2*	*520*

Comparación (cont)

	IBM 360/370	Intel 8086	Motorola 68000	DEC VAX
Date announced	1964/1970	1978	1980	1977
Instruction size(s) (bits)	16, 32, 48	8, 16, 24, 32, 40, 48	16, 32, 48, 64, 80	8, 16, 24, 32,..., 432
Addressing (size, model)	24 bits, flat/ 31 bits, flat	4 + 16 bits, segmented	24 bits, flat	32 bits, flat
Data aligned?	Yes 360/ No 370	No	16-bit aligned	No
Data addressing modes	2/3	5	9	= 14
Protection	Page	None	Optional	Page
Page size	2 KB & 4 KB	—	0.25 to 32 KB	0.5 KB
I/O	Opcode	Opcode	Memory mapped	Memory mapped
Integer registers (size, model, number)	16 GPR × 32 bits	8 dedicated data × 16 bits	8 data and 8 address × 32 bits	15 GPR × 32 bits
Separate floating-point registers	4 × 64 bits	Optional: 8 × 80 bits	Optional: 8 × 80 bits	0
Floating-point format	IBM (floating hexadecimal)	IEEE 754 single, double, extended	IEEE 754 single, double, extended	DEC

Cuatro arquitecturas de los 70

Comparación (cont)

	Alpha	MIPS I	PA-RISC 1.1	PowerPC	SPARC v.8
Date announced	1992	1986	1986	1993	1987
Instruction size (bits)	32	32	32	32	32
Address space (size, model)	64 bits, flat	32 bits, flat	48 bits, segmented	32 bits, flat	32 bits, flat
Data alignment	Aligned	Aligned	Aligned	Unaligned	Aligned
Data addressing modes	1	1	5	4	2
Protection	Page	Page	Page	Page	Page
Minimum page size	8 KB	4 KB	4 KB	4 KB	8 KB
I/O	Memory mapped	Memory mapped	Memory mapped	Memory mapped	Memory mapped
Integer registers (number, model, size)	31 GPR × 64 bits	31 GPR × 32 bits	31 GPR × 32 bits	32 GPR × 32 bits	31 GPR × 32 bits
Separate floating-point registers	31 × 32 or 31 × 64 bits	16 × 32 or 16 × 64 bits	56 × 32 or 28 × 64 bits	32 × 32 or 32 × 64 bits	32 × 32 or 32 × 64 bits
Floating-point format	IEEE 754 single, double	IEEE 754 single, double	IEEE 754 single, double	IEEE 754 single, double	IEEE 754 single, double

Cinco arquitecturas RISC

Comparación (cont)

	ARM	Thumb	SuperH	M32R	MIPS16
Date announced	1985	1995	1992	1997	1996
Instruction size (bits)	32	16	16	16/32	16/32
Address space (size, model)	32 bits, flat	32 bits, flat	32 bits, flat	32 bits, flat	32/64 bits, flat
Data alignment	Aligned	Aligned	Aligned	Aligned	Aligned
Data addressing modes	6	6	4	3	2
Integer registers (number, model, size)	15 GPR x 32 bits	8 GPR + SP, LR x 32 bits	16 GPR x 32 bits	16 GPR x 32 bits	8 GPR + SP, RA x 32/64 bits
I/O	Memory mapped	Memory mapped	Memory mapped	Memory mapped	Memory mapped

Cinco arquitecturas RISC embedded

Laboratorio: MIPS64

Claves de diseño

ISA simple de carga/almacenamiento (RISC)

Optimización de la segmentación (incluyendo instrucciones de largo fijo)

Eficiencia como 'target' de compiladores (registros)

Registros

32 GPR de 64 bits: R0(siempre cero),R1,R2,...R31(return address)

32 FPR de 64 bits: F0,F1,...F31 (simple precisión, medio vacío)

Tipos de datos

Enteros de 8, 16, 32 y 64 bits

Punto flotante IEEE 754 de simple y doble precisión

Modos de direccionamiento

Inmediato

Desplazamiento: d_{16b}(Rn) {si d=0 *register indirect*, si uso R0 *absoluto*}

ISA MIPS64

Formato de instrucciones

Todas de 32 bits

6 bits de código de operación

32 registros (5 bits)

Tres tipos de instrucciones: I, R y J

Operaciones

Load/Store (L, S y M)

ALU (ADD, SUB, MUL, DIV, OR, AND..)

Control (B y J)

FP (ADD.D, ADD.S, SUB.D, MUL.D...)

Modos de direccionamiento

Inmediato

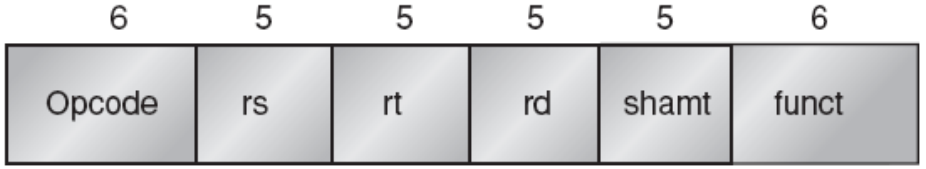
Indirecto via registro con offset

I-type instruction

Encodes: Loads and stores of bytes, half words, words, double words. All immediates (rt ← rs op immediate)

Conditional branch instructions (rs is register, rd unused)
Jump register, jump and link register
(rd = 0, rs = destination, immediate = 0)

R-type instruction



Register-register ALU operations: rd ← rs funct rt
Function encodes the data path operation: Add, Sub, . . .
Read/write special registers and moves

J-type instruction

Jump and jump and link
Trap and return from exception

Instruction type/opcode	Instruction meaning
Data transfers	*Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR*
LB,LBU,SB	Load byte, load byte unsigned, store byte (to/from integer registers)
LH,LHU,SH	Load half word, load half word unsigned, store half word (to/from integer registers)
LW,LWU,SW	Load word, load word unsigned, store word (to/from integer registers)
LD,SD	Load double word, store double word (to/from integer registers)
L.S,L.D,S.S,S.D	Load SP float, load DP float, store SP float, store DP float
MFC0,MTC0	Copy from/to GPR to/from a special register
MOV.S,MOV.D	Copy one SP or DP FP register to another FP register
MFC1,MTC1	Copy 32 bits to/from FP registers from/to integer registers
Arithmetic/logical	*Operations on integer or logical data in GPRs; signed arithmetic trap on overflow*
DADD,DADDI,DADDU,DADDIU	Add, add immediate (all immediates are 16 bits); signed and unsigned
DSUB,DSUBU	Subtract; signed and unsigned
DMUL,DMULU,DDIV, DDIVU,MADD	Multiply and divide, signed and unsigned; multiply-add; all operations take and yield 64-bit values
AND,ANDI	And, and immediate
OR,ORI,XOR,XORI	Or, or immediate, exclusive or, exclusive or immediate
LUI	Load upper immediate; loads bits 32 to 47 of register with immediate, then sign-extends
DSLL,DSRL,DSRA,DSLLV, DSRLV,DSRAV	Shifts: both immediate (DS__) and variable form (DS__V); shifts are shift left logical, right logical, right arithmetic
SLT,SLTI,SLTU,SLTIU	Set less than, set less than immediate; signed and unsigned
Control	*Conditional branches and jumps; PC-relative or through register*
BEQZ,BNEZ	Branch GPRs equal/not equal to zero; 16-bit offset from PC + 4
BEQ,BNE	Branch GPR equal/not equal; 16-bit offset from PC + 4
BC1T,BC1F	Test comparison bit in the FP status register and branch; 16-bit offset from PC + 4
MOVN,MOVZ	Copy GPR to another GPR if third GPR is negative, zero
J,JR	Jumps: 26-bit offset from PC + 4 (J) or target in register (JR)
JAL,JALR	Jump and link: save PC + 4 in R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address
ERET	Return to user code from an exception; restore user mode
Floating point	*FP operations on DP and SP formats*
ADD.D,ADD.S,ADD.PS	Add DP, SP numbers, and pairs of SP numbers
SUB.D,SUB.S,SUB.PS	Subtract DP, SP numbers, and pairs of SP numbers
MUL.D,MUL.S,MUL.PS	Multiply DP, SP floating point, and pairs of SP numbers
MADD.D,MADD.S,MADD.PS	Multiply-add DP, SP numbers and pairs of SP numbers
DIV.D,DIV.S,DIV.PS	Divide DP, SP floating point, and pairs of SP numbers
CVT._._	Convert instructions: CVT.x.y converts from type x to type y, where x and y are L (64-bit integer), W (32-bit integer), D (DP), or S (SP). Both operands are FPRs.
C.__.D,C.__.S	DP and SP compares: "__" = LT,GT,LE,GE,EQ,NE; sets bit in FP status register

IMPORTANTE

DLX, MIPS y MIPS64 son ISAs (idem IA-32)

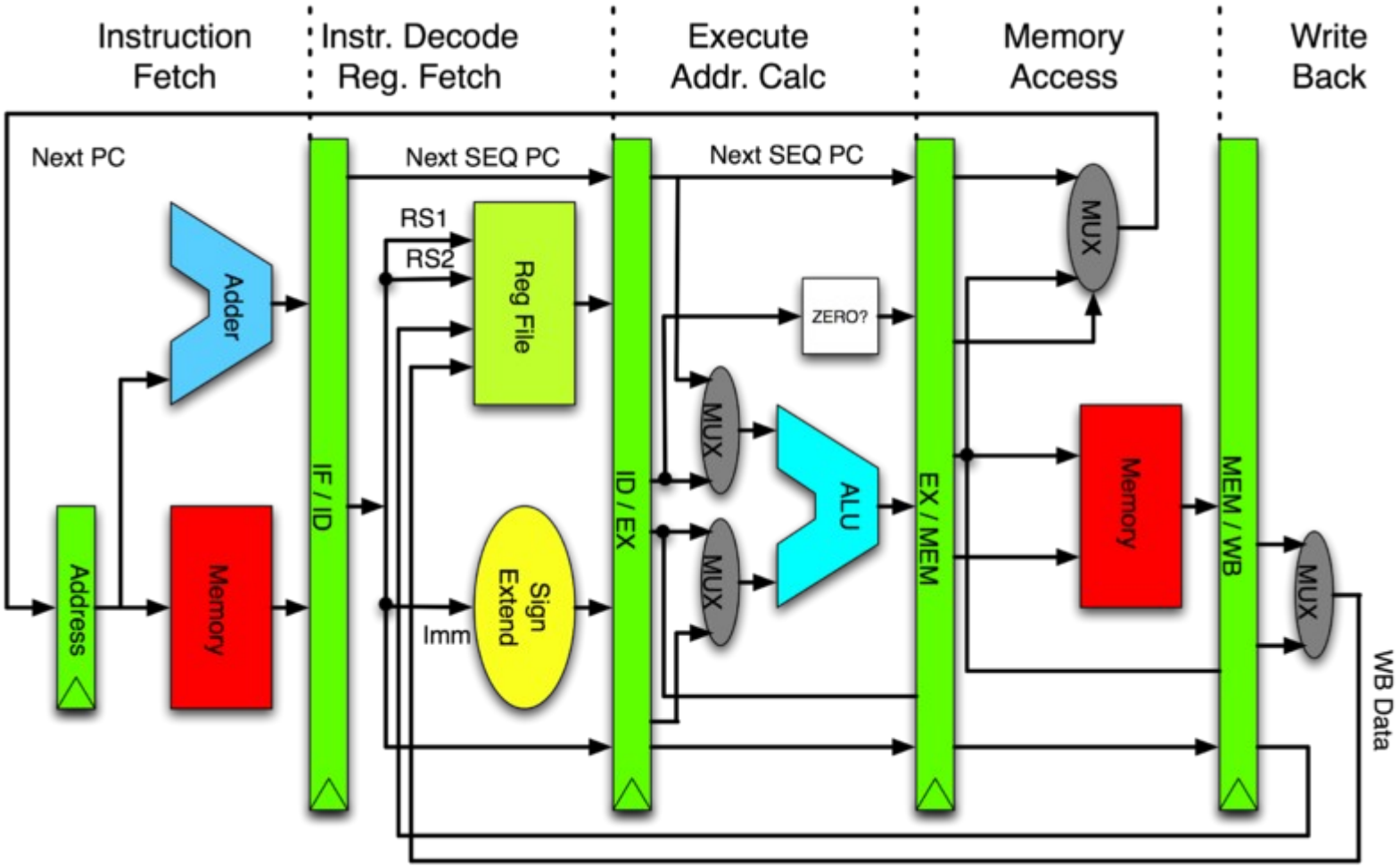
WinDLX, WinMIPS64, Sparc y R10000 son implementaciones (idem Pentium o Athlon)

Discusión: Utilidad real de los simuladores

No se utilizan para probar programas de aplicación, pues nadie programa RISCs en assembler (contrario a la filosofía de diseño).

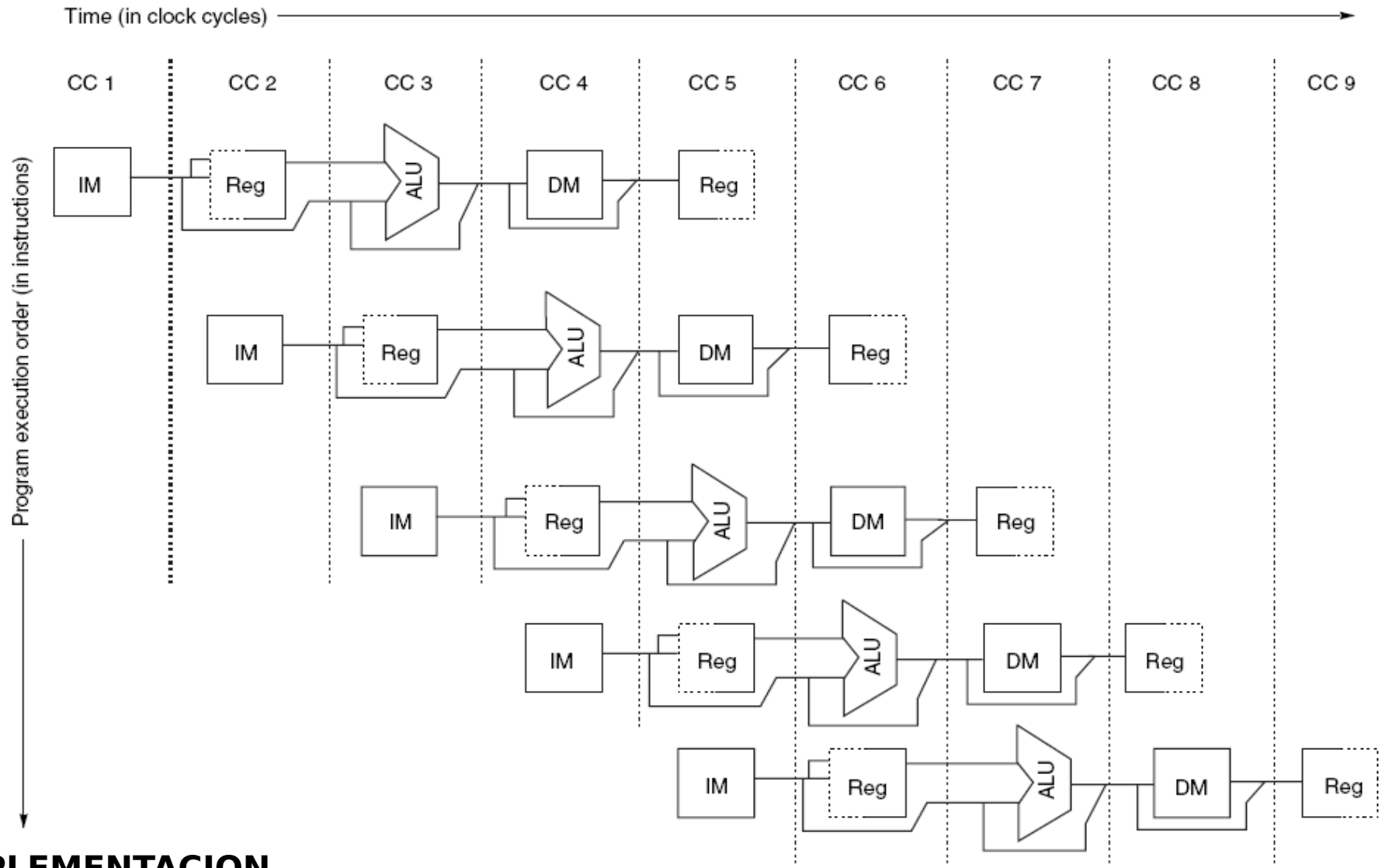
Son muy útiles para verificar diferentes técnicas de compilación.

Segmentación típica



1. **FI**: Captar la instrucción e incrementar PC
2. **ID**: Decodificar instrucción y captar registros – *(Terminan saltos: 2c)*
3. **EX**: Tres opciones: calcular dirección efectiva, ALU R-R o ALU R-Inmediato
4. **MEM**: Acceso a memoria (solo carga o almacenamiento) – *(Termina store: 4c)*
5. **WB**: Escribir registro de salida – *(Termina el resto: 5c)*

4c < CPI < 5c



IMPLEMENTACION

- Puede accederse simultáneamente a la memoria de instrucciones y de datos?
- El acceso a memoria de datos se realiza en un único ciclo? (double?)
- El banco de registros puede accederse simultáneamente para escritura y para lectura?
- Forwarding, salto retardado, carga retardada, etc.
- Operaciones en punto flotante

Tema de investigación

- Low power design (ver ARM System-On-Chip Architecture PDF).
- RISC en FPGA (ver artículos PDF). Ejemplo: MIPS R3000 (32 bits) en un Xilinx Virtex XCV300 (FPGA 300K gates, 64K RAM, 200 MHz).
- Familia de microcontroladores PIC (*Microchip Technology Inc.*) Es RISC?