

William Stallings Computer Organization and Architecture 10th Edition



+ Chapter 15

Reduced Instruction Set Computers (RISC)

Table 15.1

Characteristics of Some CISCs, RISCs, and Superscalar Processors

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

Table 15.1

Characteristics of Some CISCs, RISCs, and Superscalar Processors

Characteristic	Superscalar		
	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1993	1996	1996
Number of instructions	225		
Instruction size (bytes)	4	4	4
Addressing modes	2	1	1
Number of general-purpose registers	32	40 - 520	32
Control memory size (kbits)	—	—	—
Cache size (kB)	16-32	32	64

Instruction Execution Characteristics

High-level languages (HLLs)

- Allow the programmer to express algorithms more concisely
- Allow the compiler to take care of details that are not important in the programmer's expression of algorithms
- Often support naturally the use of structured programming and/or object-oriented design

Execution sequencing

- Determines the control and pipeline organization

Semantic gap

- The difference between the operations provided in HLLs and those provided in computer architecture

Operands used

- The types of operands and the frequency of their use determine the memory organization for storing them and the addressing modes for accessing them

Operations performed

- Determine the functions to be performed by the processor and its interaction with memory

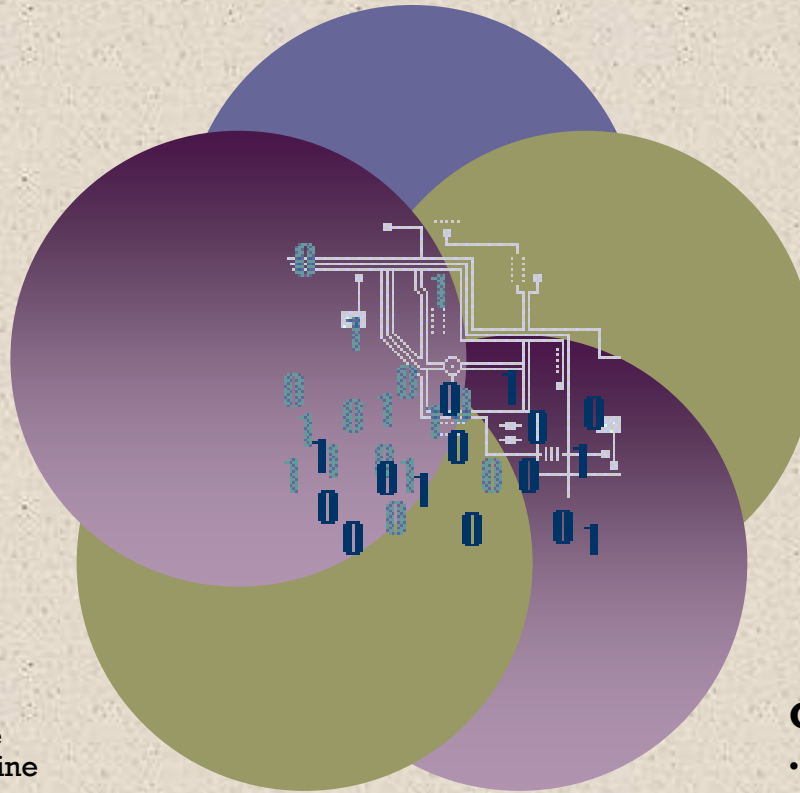




Table 15.2

Weighted Relative Dynamic Frequency of HLL Operations [PATT82a]

	Dynamic Occurrence		Machine-Instruction Weighted		Memory-Reference Weighted	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%



Table 15.3

Dynamic Percentage of Operands

	Pascal	C	Average
Integer Constant	16%	23%	20%
Scalar Variable	58%	53%	55%
Array/Structure	26%	24%	25%



Table 15.4

Procedure Arguments and Local Scalar Variables

Percentage of Executed Procedure Calls With	Compiler, Interpreter, and Typesetter	Small Nonnumeric Programs
>3 arguments	0–7%	0–5%
>5 arguments	0–3%	0%
>8 words of arguments and local scalars	1–20%	0–6%
>12 words of arguments and local scalars	1–6%	0–3%



Implications



- HLLs can best be supported by optimizing performance of the most time-consuming features of typical HLL programs
- Three elements characterize RISC architectures:
 - Use a large number of registers or use a compiler to optimize register usage
 - Careful attention needs to be paid to the design of instruction pipelines
 - Instructions should have predictable costs and be consistent with a high-performance implementation



The Use of a Large Register File

Software Solution

- Requires compiler to allocate registers
- Allocates based on most used variables in a given time
- Requires sophisticated program analysis

Hardware Solution

- More registers
- Thus more variables will be in registers



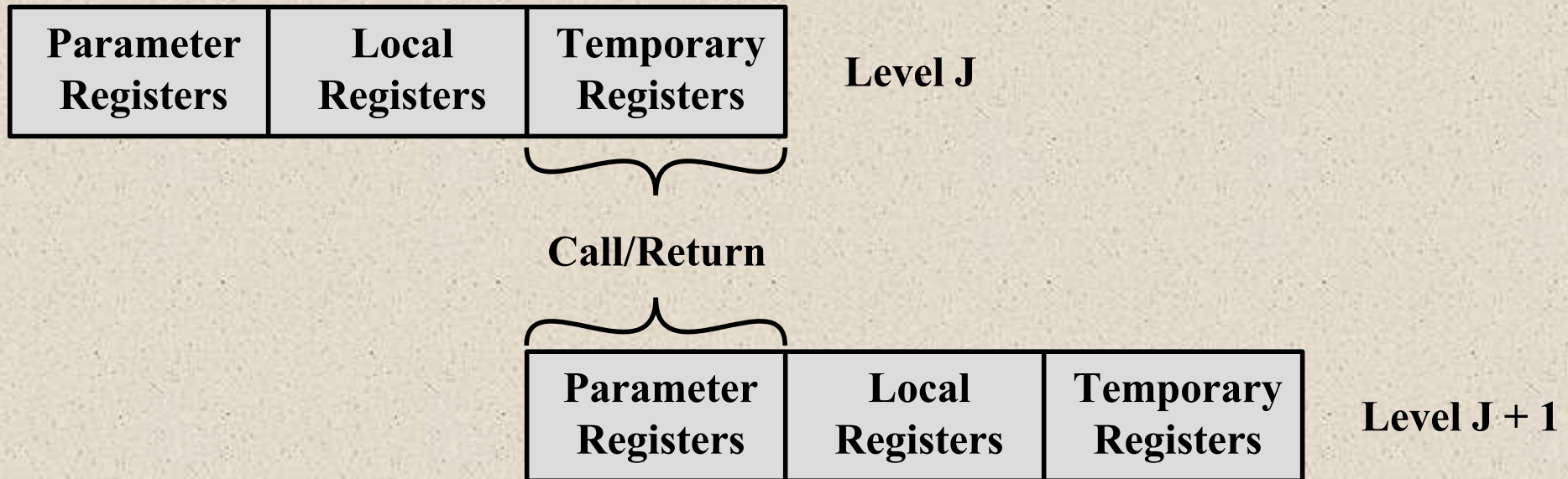


Figure 15.1 Overlapping Register Windows

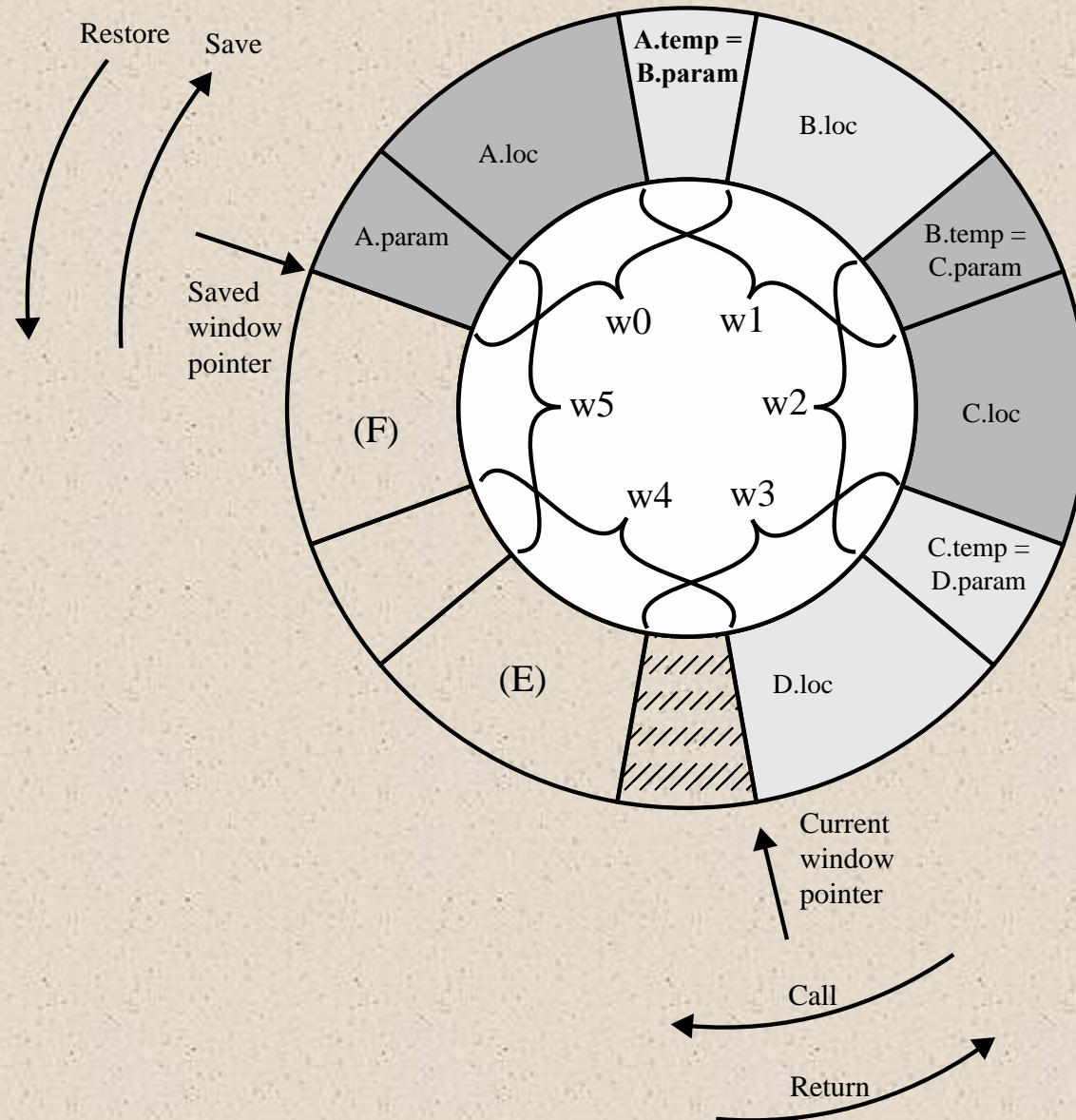


Figure 15.2 Circular-Buffer Organization of Overlapped Windows



Global Variables



- Variables declared as global in an HLL can be assigned memory locations by the compiler and all machine instructions that reference these variables will use memory reference operands
 - However, for frequently accessed global variables this scheme is inefficient
- Alternative is to incorporate a set of global registers in the processor
 - These registers would be fixed in number and available to all procedures
 - A unified numbering scheme can be used to simplify the instruction format
- There is an increased hardware burden to accommodate the split in register addressing
- In addition, the linker must decide which global variables should be assigned to registers


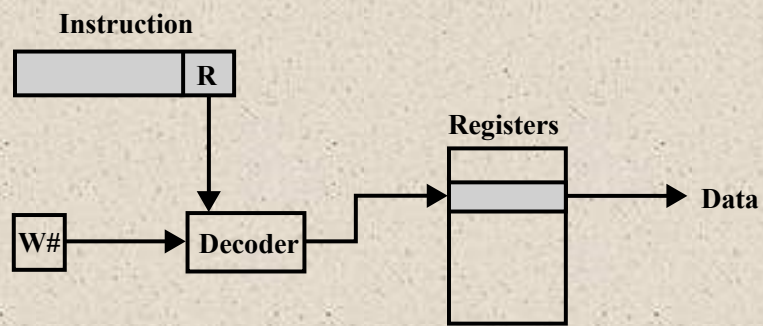


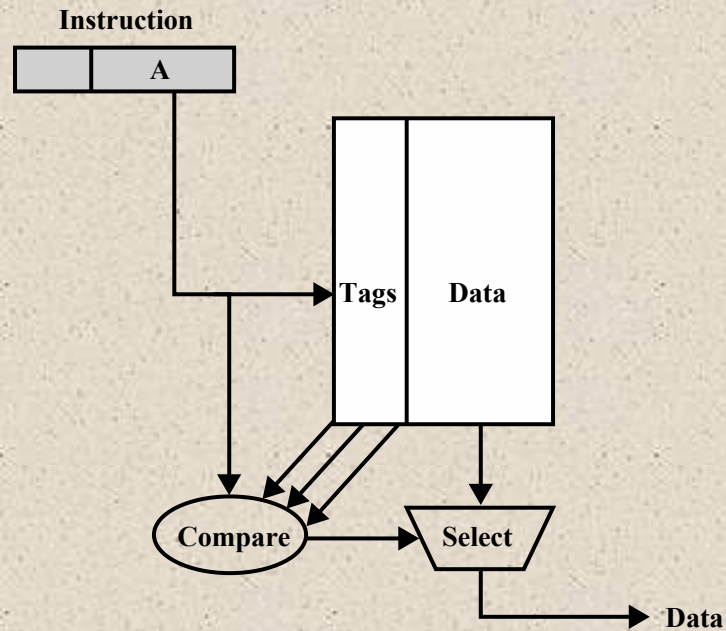
Table 15.5

Characteristics of Large-Register-File and Cache Organizations

Large Register File	Cache
All local scalars	Recently-used local scalars
Individual variables	Blocks of memory
Compiler-assigned global variables	Recently-used global variables
Save/Restore based on procedure nesting depth	Save/Restore based on cache replacement algorithm
Register addressing	Memory addressing
Multiple operands addressed and accessed in one cycle	One operand addressed and accessed per cycle

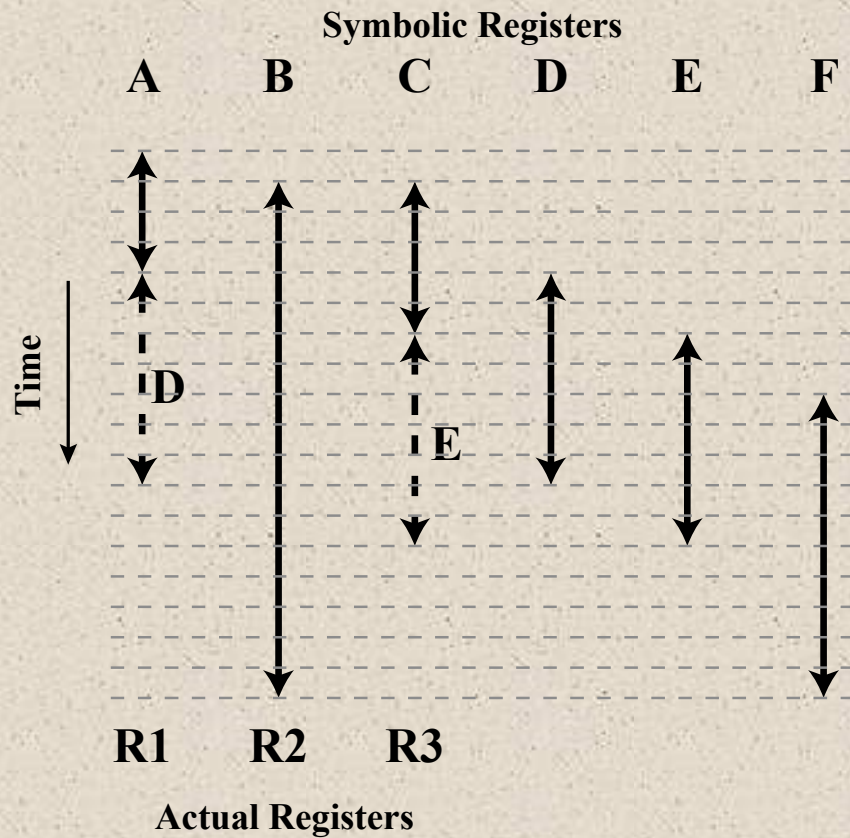


(a) Windows-based register file

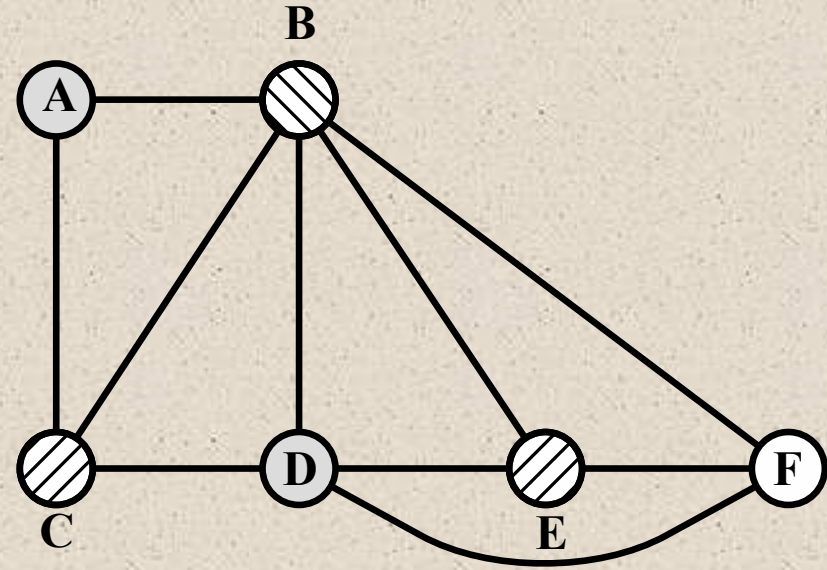


(b) Cache

Figure 15.3 Referencing a Scalar



(a) Time sequence of active use of registers



(b) Register interference graph

Figure 15.4 Graph Coloring Approach



Why CISC ?

(Complex Instruction Set Computer)



- There is a trend to richer instruction sets which include a larger and more complex number of instructions
- Two principal reasons for this trend:
 - A desire to simplify compilers
 - A desire to improve performance
- There are two advantages to smaller programs:
 - The program takes up less memory
 - Should improve performance
 - Fewer instructions means fewer instruction bytes to be fetched
 - In a paging environment smaller programs occupy fewer pages, reducing page faults
 - More instructions fit in cache(s)



Table 15.6

Code Size Relative to RISC I

	[PATT82a] 11 C Programs	[KATE83] 12 C Programs	[HEAT84] 5 C Programs
RISC I	1.0	1.0	1.0
VAX-11/780	0.8	0.67	
M68000	0.9		0.9
Z8002	1.2		1.12
PDP-11/70	0.9	0.71	

Characteristics of Reduced Instruction Set Architectures



One machine instruction per machine cycle

- *Machine cycle* --- the time it takes to fetch two operands from registers, perform an ALU operation, and store the result in a register

Register-to-register operations

- Only simple LOAD and STORE operations accessing memory
- This simplifies the instruction set and therefore the control unit

Simple addressing modes

- Simplifies the instruction set and the control unit

Simple instruction formats

- Generally only one or a few formats are used
- Instruction length is fixed and aligned on word boundaries
- Opcode decoding and register operand accessing can occur simultaneously

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	R A	RB	RC	
Store	R A	A		

Register to memory

I = 104, D = 96, M = 200

(a) $A \leftarrow B + C$

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 register

I = 60, D = 0, M = 60

(b) $A \leftarrow B + C$; $B \leftarrow A + C$; $D \leftarrow D - B$

I = number of bytes occupied by executed instructions

D = number of bytes occupied by data

M = total memory traffic = I + D

Figure 15.5 Two Comparisons of Register-to-Register and Memory-to-Memory Approaches

Table 15.7

Characteristics of Some Processors



Processor	Number of instruction sizes	Max instruction size in bytes	Number of addressing modes	Indirect addressing	Load/store combined with arithmetic	Max number of memory operands	Unaligned addressing allowed	Max Number of MMU uses	Number of bits for integer register specifier	Number of bits for FP register specifier
AMD29000	1	4	1	no	no	1	no	1	8	3 ^a
MIPS R2000	1	4	1	no	no	1	no	1	5	4
SPARC	1	4	2	no	no	1	no	1	5	4
MC88000	1	4	3	no	no	1	no	1	5	4
HP PA	1	4	10 ^a	no	no	1	no	1	5	4
IBM RT/PC	2 ^a	4	1	no	no	1	no	1	4 ^a	3 ^a
IBM RS/6000	1	4	4	no	no	1	yes	1	5	5
Intel i860	1	4	4	no	no	1	no	1	5	4
IBM 3090	4	8	2 ^b	no ^b	yes	2	yes	4	4	2
Intel 80486	12	12	15	no ^b	yes	2	yes	4	3	3
NSC 32016	21	21	23	yes	yes	2	yes	4	3	3
MC68040	11	22	44	yes	yes	2	yes	8	4	3
VAX	56	56	22	yes	yes	6	yes	24	4	0
Clipper	4 ^a	8 ^a	9 ^a	no	no	1	0	2	4 ^a	3 ^a
Intel 80960	2 ^a	8 ^a	9 ^a	no	no	1	yes ^a	—	5	3 ^a

a RISC that does not conform to this characteristic.

b CISC that does not conform to this characteristic.

(Table can be found on page 554
in the textbook.)

Load $rA \leftarrow M$
 Load $rB \leftarrow M$
 Add $rC \leftarrow rA + rB$
 Store $M \leftarrow rC$
 Branch X

I	E	D									
			I	E	D						
						I	E				
								I	E	D	
										I	E

(a) Sequential execution

Load $rA \leftarrow M$
 Load $rB \leftarrow M$
 Add $rC \leftarrow rA + rB$
 Store $M \leftarrow rC$
 Branch X
 NOOP

I	E	D									
	I		E	D							
			I		E						
					I	E	D				
					I			E			
								I	E		

(b) Two-stage pipelined timing

Load $rA \leftarrow M$
 Load $rB \leftarrow M$
 NOOP
 Add $rC \leftarrow rA + rB$
 Store $M \leftarrow rC$
 Branch X
 NOOP

I	E	D					
	I	E	D				
		I	E				
			I	E			
				I	E	D	
					I	E	
						I	E

(c) Three-stage pipelined timing

Load $rA \leftarrow M$
 Load $rB \leftarrow M$
 NOOP
 NOOP
 Add $rC \leftarrow rA + rB$
 Store $M \leftarrow rC$
 Branch X
 NOOP
 NOOP

I	E ₁	E ₂	D							
	I	E ₁	E ₂	D						
		I	E ₁	E ₂						
			I	E ₁	E ₂					
				I	E ₁	E ₂				
					I	E ₁	E ₂	D		
						I	E ₁	E ₂		
							I	E ₁	E ₂	
								I	E ₁	E ₂

(d) Four-stage pipelined timing

Figure 15.6 The Effects of Pipelining



Optimization of Pipelining



- Delayed branch
 - Does not take effect until after execution of following instruction
 - This following instruction is the delay slot
- Delayed Load
 - Register to be target is locked by processor
 - Continue execution of instruction stream until register required
 - Idle until load is complete
 - Re-arranging instructions can allow useful work while loading
- Loop Unrolling
 - Replicate body of loop a number of times
 - Iterate loop fewer times
 - Reduces loop overhead
 - Increases instruction parallelism
 - Improved register, data cache, or TLB locality

Table 15.8

Normal And Delayed Branch

Address	Normal Branch		Delayed Branch		Optimized Delayed Branch	
100	LOAD	X, rA	LOAD	X, rA	LOAD	X, rA
101	ADD	1, rA	ADD	1, rA	JUMP	105
102	JUMP	105	JUMP	106	ADD	1, rA
103	ADD	rA, rB	NOOP		ADD	rA, rB
104	SUB	rC, rB	ADD	rA, rB	SUB	rC, rB
105	STORE	rA, Z	SUB	rC, rB	STORE	rA, Z
106			STORE	rA, Z		

Time →

	1	2	3	4	5	6	7	8
100 LOAD X, rA	I	E	D					
101 ADD 1, rA		I		E				
102 JUMP 105				I	E			
103 ADD rA, rB					I	E		
105 STORE rA, Z						I	E	D

(a) Traditional Pipeline


	1	2	3	4	5	6	7	8
100 LOAD X, rA	I	E	D					
101 ADD 1, rA		I		E				
102 JUMP 106				I	E			
103 NOOP					I	E		
106 STORE rA, Z						I	E	D

(b) RISC Pipeline with Inserted NOOP

	1	2	3	4	5	6
100 LOAD X, rA	I	E	D			
101 JUMP 105		I	E			
102 ADD 1, rA			I	E		
105 STORE rA, Z				I	E	D

(c) Reversed Instructions

Figure 15.7 Use of the Delayed Branch



```
do i=2, n-1
    a[i] = a[i] + a[i-1] * a[i+1]
end do
```

(a) original loop

```
do i=2, n-2, 2
    a[i] = a[i] + a[i-1] * a[i+1]
    a[i+1] = a[i+1] + a[i] * a[i+2]
end do

if (mod(n-2,2) = 1) then
    a[n-1] = a[n-1] + a[n-2] * a[n]
end if
```

(b) loop unrolled twice

Figure 15.8 Loop unrolling

MIPS R4000

One of the first commercially available RISC chip sets was developed by MIPS Technology Inc.

Inspired by an experimental system developed at Stanford

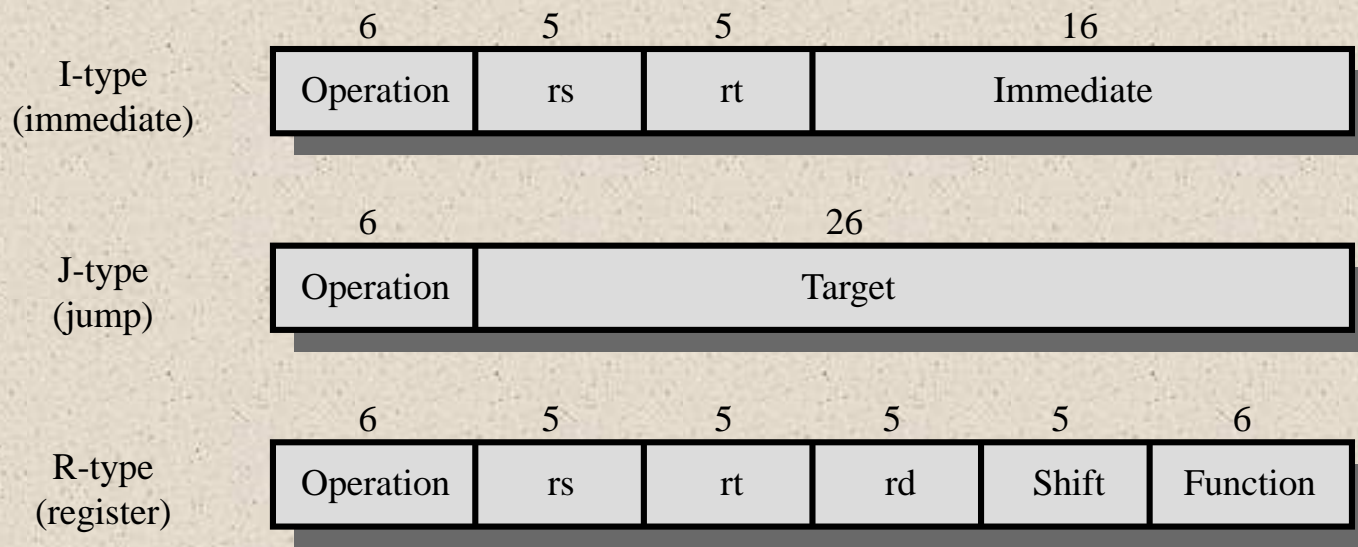
Has substantially the same architecture and instruction set of the earlier MIPS designs (R2000 and R3000)

Uses 64 bits for all internal and external data paths and for addresses, registers, and the ALU

Is partitioned into two sections, one containing the CPU and the other containing a coprocessor for memory management

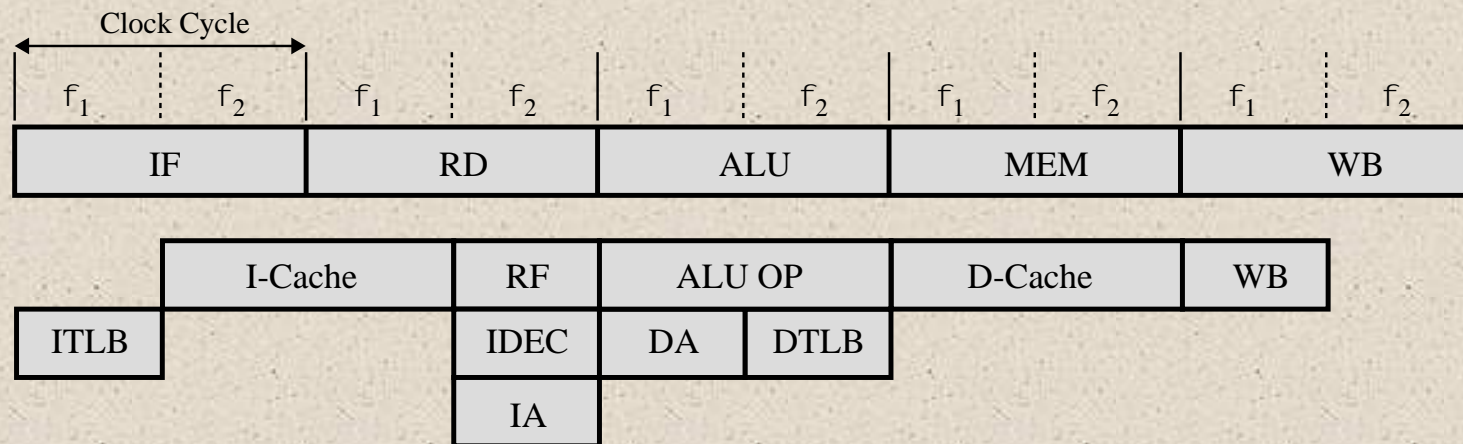
Supports thirty-two 64-bit registers

Provides for up to 128 Kbytes of high-speed cache, half each for instructions and data

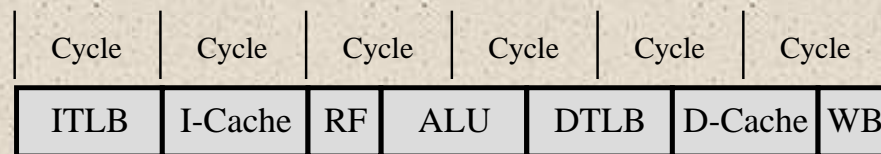


Operation	Operation code
rs	Source register specifier
rt	Source/destination register specifier
Immediate	Immediate, branch, or address displacement
Target	Jump target address
rd	Destination register specifier
Shift	Shift amount
Function	ALU/shift function specifier

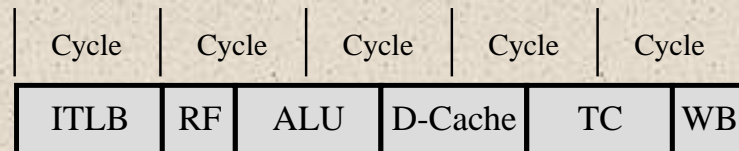
Figure 15.9 MIPS Instruction Formats



(a) Detailed R3000 pipeline



(b) Modified R3000 pipeline with reduced latencies



(c) Optimized R3000 pipeline with parallel TLB and cache accesses

- IF = Instruction fetch
- RD = Read
- MEM = Memory access
- WB = Write back to register file
- I-Cache = Instruction cache access
- RF = Fetch operand from register
- D-Cache = Data cache access
- ITLB = Instruction address translation
- IDEC = Instruction decode
- IA = Compute instruction address
- DA = Calculate data virtual address
- DTLB = Data address translation
- TC = Data cache tag check

Figure 15.10 Enhancing the R3000 Pipeline

Table 15.9

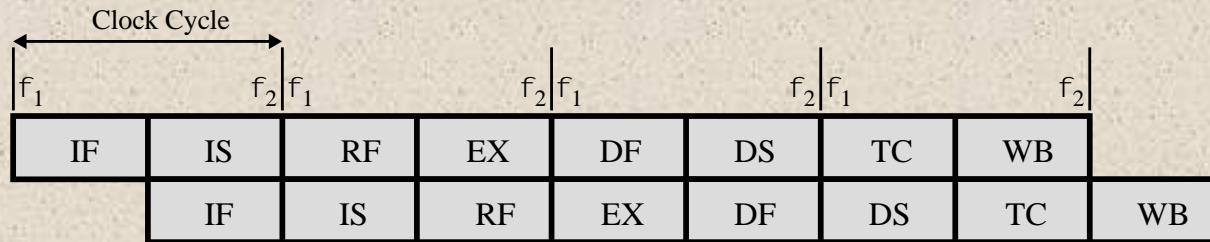
R3000 Pipeline Stages



Pipeline Stage	Phase	Function
IF	f1	Using the TLB, translate an instruction virtual address to a physical address (after a branching decision).
IF	f2	Send the physical address to the instruction address.
RD	f1	Return instruction from instruction cache. Compare tags and validity of fetched instruction.
RD	f2	Decode instruction. Read register file. If branch, calculate branch target address.
ALU	f1 + f2	If register-to-register operation, the arithmetic or logical operation is performed.
ALU	f1	If a branch, decide whether the branch is to be taken or not. If a memory reference (load or store), calculate data virtual address.
ALU	f2	If a memory reference, translate data virtual address to physical using TLB.
MEM	f1	If a memory reference, send physical address to data cache.
MEM	f2	If a memory reference, return data from data cache, and check tags.
WB	f1	Write to register file.



(a) Superpipelined implementation of the optimized R3000 pipeline



(b) R4000 pipeline

IF = Instruction fetch first half
 IS = Instruction fetch second half
 RF = Fetch operands from register
 EX = Instruction execute
 IC = Instruction cache

DC = Data cache
 DF = Data cache first half
 DS = Data cache second half
 TC = Tag check
 WB = Write back to register file

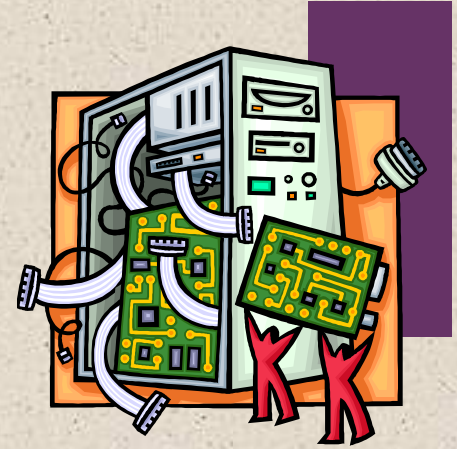
Figure 15.11 Theoretical R3000 and Actual R4000 Superpipelines

R4000 Pipeline Stages

- Instruction fetch first half
 - Virtual address is presented to the instruction cache and the translation lookaside buffer
- Instruction fetch second half
 - Instruction cache outputs the instruction and the TLB generates the physical address
- Register file
 - One of three activities can occur:
 - Instruction is decoded and check made for interlock conditions
 - Instruction cache tag check is made
 - Operands are fetched from the register file
- Tag check
 - Cache tag checks are performed for loads and stores
- Instruction execute
 - One of three activities can occur:
 - If register-to-register operation the ALU performs the operation
 - If a load or store the data virtual address is calculated
 - If branch the branch target virtual address is calculated and branch operations checked
- Data cache first
 - Virtual address is presented to the data cache and TLB
- Data cache second
 - The TLB generates the physical address and the data cache outputs the data
- Write back
 - Instruction result is written back to register file

+ SPARC

Scalable Processor Architecture



- Architecture defined by Sun Microsystems
- Sun licenses the architecture to other vendors to produce SPARC-compatible machines
- Inspired by the Berkeley RISC 1 machine, and its instruction set and register organization is based closely on the Berkeley RISC model

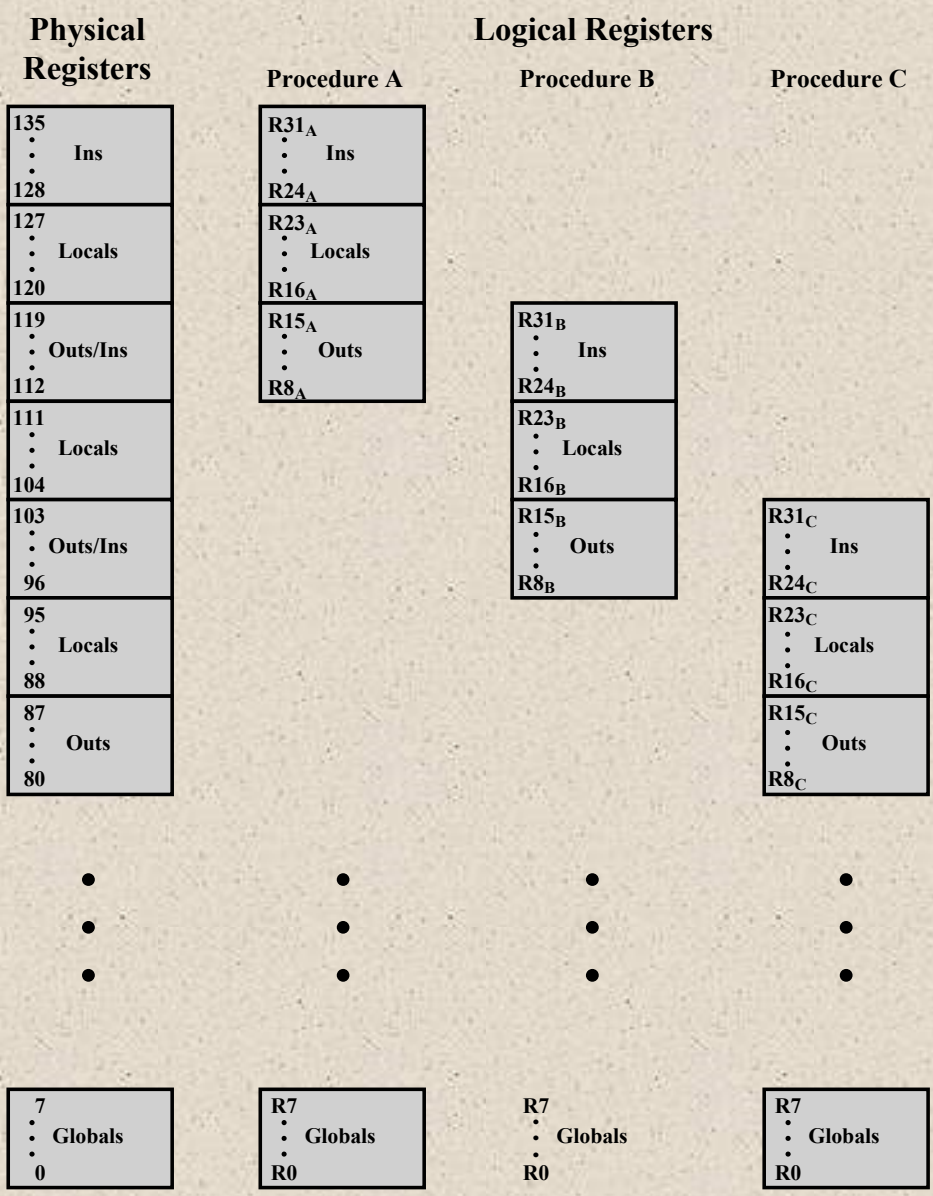


Figure 15.12 SPARC Register Window Layout with Three Procedures

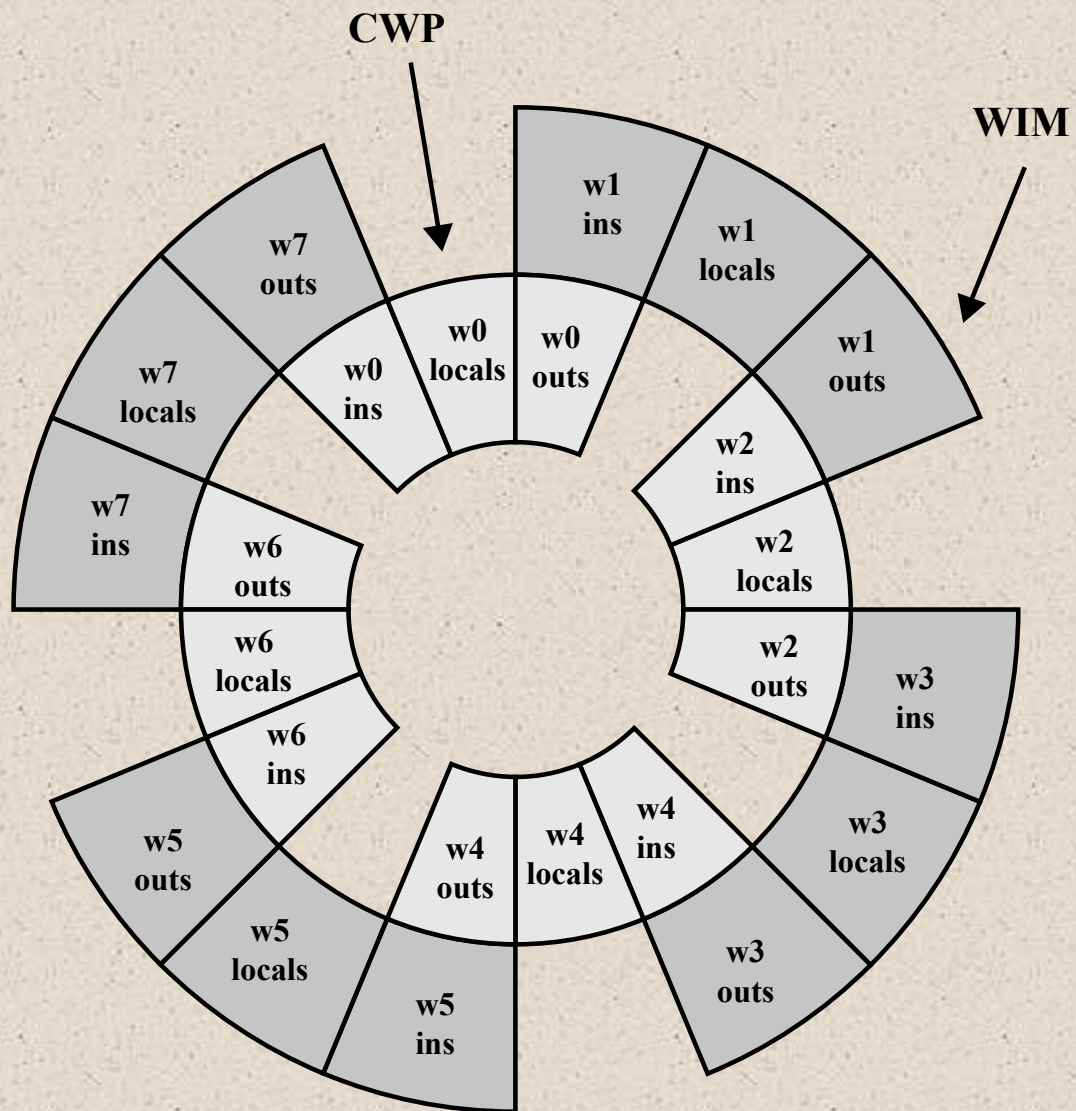


Figure 15.13 Eight Register Windows Forming a Circular Stack in SPARC



Table 15.10

Synthesizing Other Addressing Modes with SPARC Addressing Modes

Instruction Type	Addressing Mode	Algorithm	SPARC Equivalent
Register-to-register	Immediate	operand = A	S2
Load, store	Direct	EA = A	$R_0 + S2$
Register-to-register	Register	EA = R	R_{S1}, R_{S2}
Load, store	Register Indirect	EA = (R)	$R_{S1} + 0$
Load, store	Displacement	EA = (R) + A	$R_{S1} + S2$

S2 = either a register operand or a 13-bit immediate operand

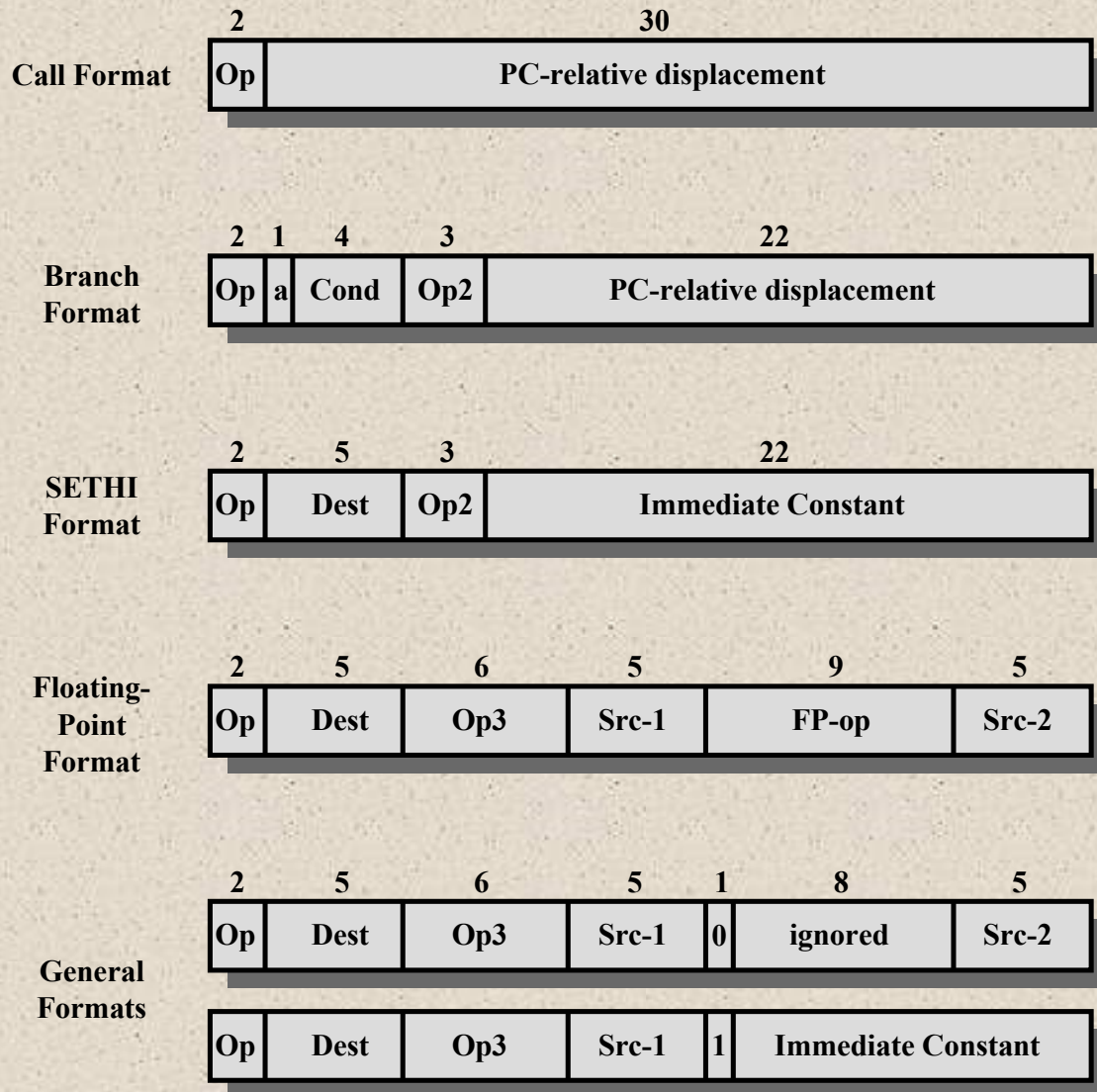


Figure 15.14 SPARC Instruction Formats



RISC versus CISC Controversy



■ Quantitative

- Compare program sizes and execution speeds of programs on RISC and CISC machines that use comparable technology

■ Qualitative

- Examine issues of high level language support and use of VLSI real estate

■ Problems with comparisons:

- No pair of RISC and CISC machines that are comparable in life-cycle cost, level of technology, gate complexity, sophistication of compiler, operating system support, etc.
- No definitive set of test programs exists
- Difficult to separate hardware effects from compiler effects
- Most comparisons done on “toy” rather than commercial products
- Most commercial devices advertised as RISC possess a mixture of RISC and CISC characteristics

+ Summary

Chapter 15

- Instruction execution characteristics
 - Operations
 - Operands
 - Procedure calls
 - Implications
- The use of a large register file
 - Register windows
 - Global variables
 - Large register file versus cache
- Reduced instruction set architecture
 - Characteristics of RISC
 - CISC versus RISC characteristics

Reduced Instruction Set Computers (RISC)

- RISC pipelining
 - Pipelining with regular instructions
 - Optimization of pipelining
- MIPS R4000
 - Instruction set
 - Instruction pipeline
- SPARC
 - SPARC register set
 - Instruction set
 - Instruction format
- Compiler-based register optimization
- RISC versus CISC controversy