COMPUTER ORGANIZATION AND ARCHITECTURE (COA)

EET 2211

4TH SEMESTER – CSE & CSIT

CHAPTER 15, LECTURE 31

By Ms. Arya Tripathy

REDUCED INSTRUCTION SET COMPUTERS

TOPICS TO BE COVERED

- ✓ Instruction Execution Characteristics
- ✓ The Use of a Large Register File
- ✓ Compiler-Based Register Optimization
- ✓ Reduced Instruction Set Architecture
- ✓ RISC Pipelining
- ✓ MIPS R4000
- ✓ SPARC
- ✓ RISC versus CISC Controversy

Contd.

- **15.4 Reduced Instruction Set Architecture**

Why CISC

Characteristics of Reduced Instruction Set Architectures

CISC versus RISC Characteristics

- **15.5 RISC Pipelining**

Pipelining with Regular Instructions

Optimization of Pipelining

- **15.6 MIPS R4000**

Instruction Set

Instruction Pipeline

- **15.7 SPARC**

SPARC Register Set

Instruction Set

Instruction Format

- **15.8 RISC versus CISC Controversy**

LEARNING OBJECTIVES

✓Provide an overview research results on instruction execution characteristics that motivated the development of the RISC approach.

✓Summarize the key characteristics of RISC machines.

✓Understand the design and performance implications of using a large register file.

✓Understand the use of compiler-based register optimization to improve performance.

✓Discuss the implication of a RISC architecture for pipeline design and performance.

✓List and explain key approaches to pipeline optimization on a RISC machine.

REDUCED INSTRUCTION SET ARCHITECTURE

WHY CISC

MOTIVATION

- to simplify compilers
- to improve performance

COMPILER SIMPLIFICATION?

-Disputed

-Complex machine instructions harder to exploit

-optimization much more difficult

ADVANTAGES OF SMALLER PROGRAMS

- the program takes up less memory
- memory today being so inexpensive

SMALLER PROGRAM IMPROVE PERFORMANCE IN THREE WAYS

- 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).

	[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	

Code Size Relative to RISC I

FASTER PROGRAMS?

- bias toward the use of simpler instructions
- entire control unit must be made more complex
- the microprogram control store must be made larger, to accommodate a richer instruction set.
- Thus the execution time of the simple instructions Increase

It is far from clear that complex instruction sets is Appropriate solution leads to opposite path

CHARACTERISTICS OF REDUCED INSTRUCTION SET ARCHITECTURES

- One instruction per cycle (a machine cycle is defined as the time taken to fetch two operands from registers, perform an ALU operation and store the result in a register)
- Register-to-register operations
- Simple addressing modes
- Simple instruction formats

CHARACTERISTICS OF REDUCED INSTRUCTION SET ARCHITECTURES

- One instruction per cycle
- Register-to-register operations
- Simple addressing modes
- Simple instruction formats
- More effective optimizing compilers
- A control unit built specifically for those instructions and using little or no microcode could execute
- Instruction pipelining
- More responsive to interrupts

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 = 60, D = 0, M = 60

(b) A ← B + C; B ← A + C; D ← D − B

I = number of bytes occupied by executed instructions
D = number of bytes occupied by data
M = total memory traffic = I + D

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

CISC vs. RISC CHARACTERISTICS

1. A single instruction size.
2. That size is typically 4 bytes.
3. A small number of data addressing modes
4. No indirect addressing that requires to make one memory access to get the address of another operand in memory.
5. No operations that combine load/store with arithmetic (e.g., add from memory, add to memory).
6. No more than one memory-addressed operand per instruction.
7. Does not support arbitrary alignment of data for load/store operations.
8. Maximum number of uses of the memory management unit (MMU) for a data address in an instruction.
9. Number of bits for integer register specifier equal to five or more.
10. Number of bits for floating-point register specifier equal to four or more.

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]

RISC PIPELINING

RISC Pipelining

Pipelining with Regular Instructions

- to enhance performance

An instruction cycle has the following two stages

I: Instruction fetch.

E: Execute. Performs an ALU operation with register input and output.

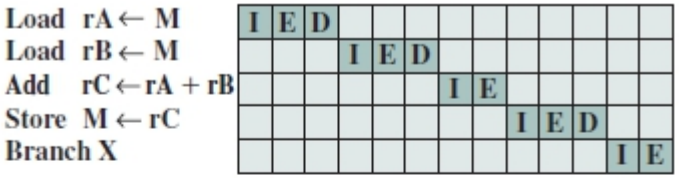
For load and store operations, three stages are required

I: Instruction fetch.

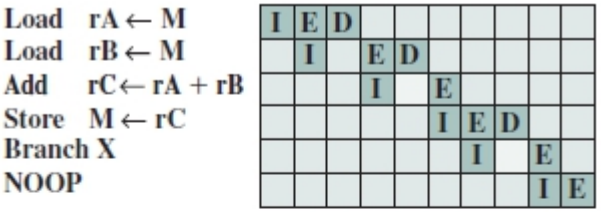
E: Execute. Calculates memory address.

D: Memory. Register-to-memory or memory-to-register operation.

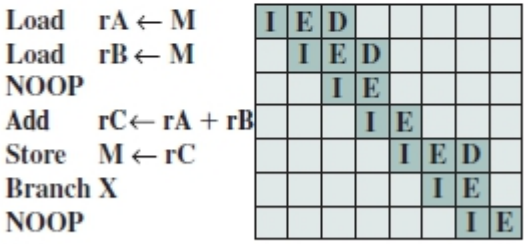
Effects of pipelining



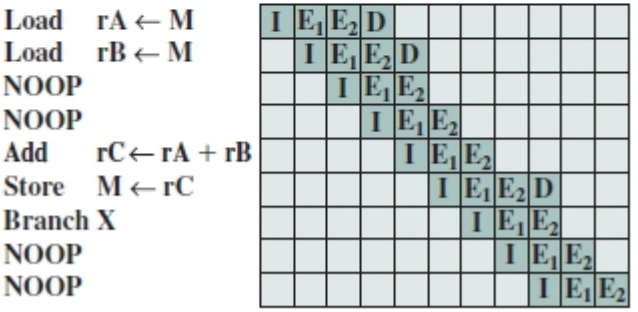
(a) Sequential execution



(b) Two-stage pipelined timing



(c) Three-stage pipelined timing



(d) Four-stage pipelined timing

Optimization of Pipelining

delayed branch- a way of increasing the efficiency of the pipeline

- makes use of a branch that does not take effect until after execution of the following instruction (hence the term *delayed)*.
- The instruction location immediately following the branch is referred to as the *delay slot*.

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		

Use of the Delayed Branch

	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, Ar	I	E	D			
101 JUMP 105		I	E			
102 ADD 1, rA			I	E		
105 STORE rA, Z				I	E	D

(c) Reversed instructions

- ***delayed load-***used on LOAD instructions.
- the register to be the target is locked by the processor
- The processor then continues execution of the instruction stream until it reaches an instruction requiring that register
- idles until the load is complete.
- If the compiler can rearrange instructions , while the load is in the pipeline, efficiency is increased.

- ***loop unrolling***
- Unrolling replicates the body of a loop some number of times called the unrolling factor (*u)*
- *iterates loop fewer times*
- ***Performance can be improved by***
- reducing loop overhead
- increasing instruction parallelism by improving pipeline performance
- improving register, data cache, or TLB locality

loop unrolling Example

```
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) = i) then
    a[n-1] = a[n-1] + a[n-2] * a[n]
end if
```

(b) Loop unrolled twice

MIPS R4000

MIPS R4000

- R4000 uses 64 bits for all internal and external data paths and for a ddresses, registers, and the ALU.
- **advantages**
- A bigger address space—Large enough for an operating system to map more than a terabyte of files directly into virtual memory for easy access
- data such as IEEE double-precision floating-point numbers and character strings ,up to eight characters in a single action

- **MIPS R4000 Architecture**

partitioned into two sections

- CPU
- a coprocessor for memory management.

Intension behind this architecture

- to design a system in which the instruction execution logic was as simple as possible, leaving space available for logic to enhance performance (e.g., the entire memory-management unit).

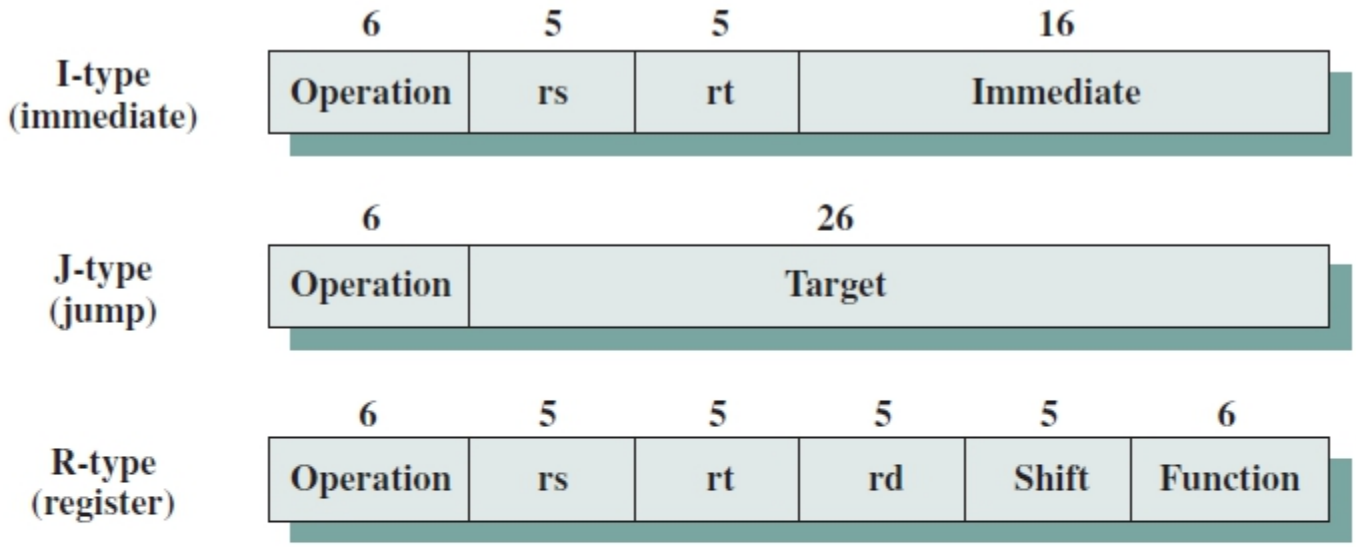
- **Characteristics**
- The processor supports thirty-two 64-bit registers.
- It also provides for up to 128 Kbytes of high-speed cache, half each for instructions and data.
- The relatively large cache (the IBM 3090 provides 128 to 256 Kbytes of cache) enables the system to keep large sets of program code and data local to the processor,
- off-loading the main memory bus
- avoiding the need for a large register file with the accompanying windowing logic.

Instruction Set

- MIPS R series instructions are encoded in a single 32-bit word format.
- data operations are register to register
- memory references are pure load/store
- R4000 makes no use of condition codes
- Avoids the need for special logic to deal with condition codes
- conditions mapped onto the register files

- single instruction length (32 bit)simplifies instruction fetch and decode
- Simplifies the interaction of instruction fetch with the virtual memory management unit

MIPS Instruction Formats



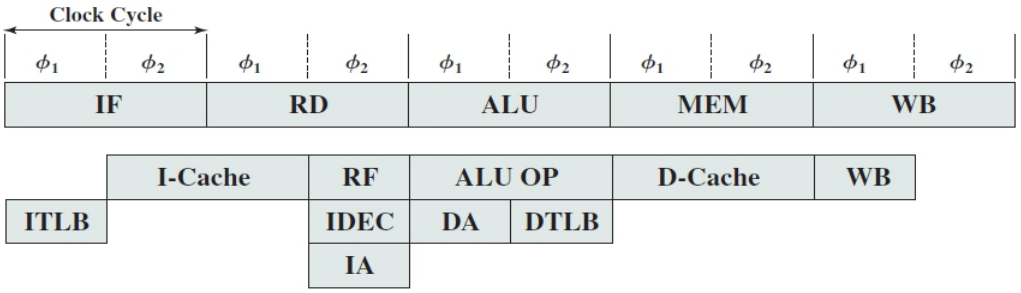
Operation	Operation code
rs	Source register specifier
rt	Source/destination register specifier
Immediate	Immediate, branch, or address displacement
Target	Jump target address
rd	Destination register specifier
Shift	Shift amount
Function	ALU/shift function specifier

Instruction Pipeline

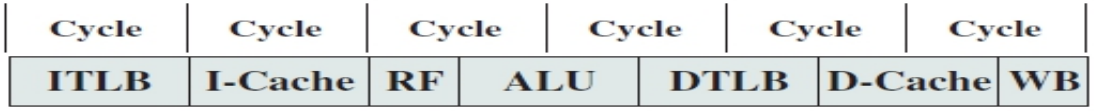
five pipeline stages:

- Instruction fetch;
- Source operand fetch from register file;
- ALU operation or data operand address generation;
- Data memory reference;
- Write back into register file.

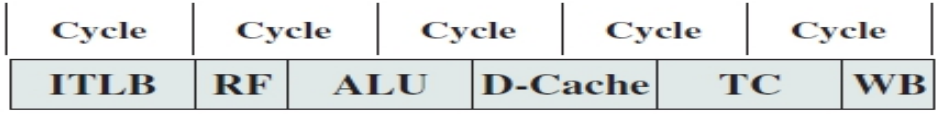
Enhancing the R3000 Pipeline



(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

R3000 Pipeline Stages

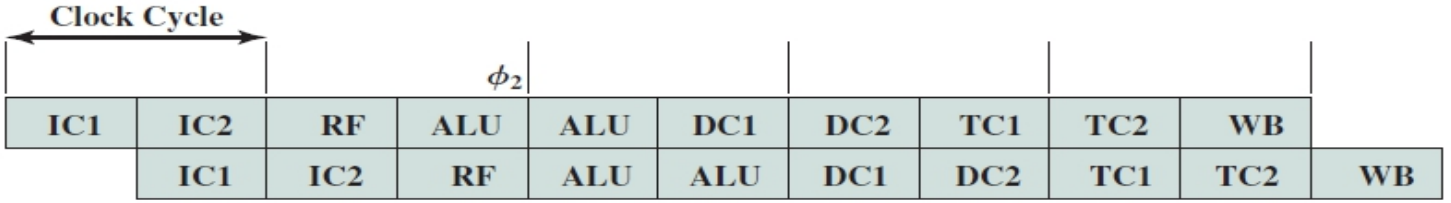
Pipeline Stage	Phase	Function
IF	$\phi 1$	Using the TLB, translate an instruction virtual address to a physical address (after a branching decision).
IF	$\phi 2$	Send the physical address to the instruction address.
RD	$\phi 1$	Return instruction from instruction cache.
		Compare tags and validity of fetched instruction.
RD	$\phi 2$	Decode instruction.
		Read register file.
		If branch, calculate branch target address.
ALU	$\phi 1 + \phi 2$	If register-to-register operation, the arithmetic or logical operation is performed.
ALU	$\phi 1$	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	$\phi 2$	If a memory reference, translate data virtual address to physical using TLB.
MEM	$\phi 1$	If a memory reference, send physical address to data cache.
MEM	$\phi 2$	If a memory reference, return data from data cache, and check tags.
WB	$\phi 1$	Write to register file.

The eight pipeline stages are as follows

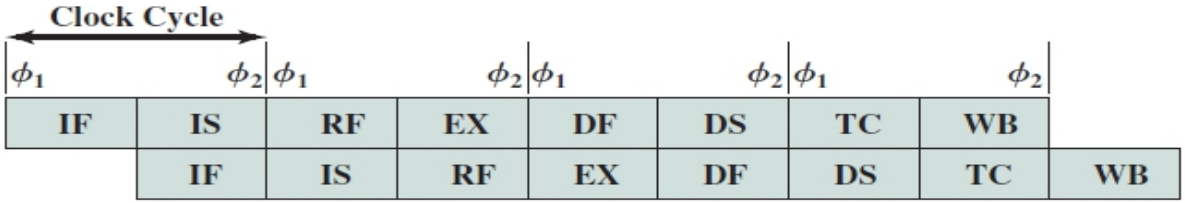
- **Instruction fetch first half:** Virtual address is presented to the instruction cache
- and the translation lookaside buffer.
- **Instruction fetch second half**: Instruction cache outputs the instruction and the
- TLB generates the physical address.
- **Register file:** Three activities occur in parallel**:**
- — Instruction is decoded and check made for interlock conditions (i.e., this
- instruction depends on the result of a preceding instruction).
- —Instruction cache tag check is made.
- —Operands are fetched from the register file.
-

Cont..

- **Instruction execute:** One of three activities can occur**:**
- — If the instruction is a register-to-register operation, the ALU performs thearithmetic or logical operation.
- —If the instruction is a load or store, the data virtual address is calculated.
- — If the instruction is a branch, the branch target virtual address is calculated and branch conditions are 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.
- **Tag check:** Cache tag checks are performed for loads and stores.
- **Write back**: Instruction result is written back to register file



(a) Superpipelined implementation of the optimized R3000 pipeline



(b) R4000 pipeline

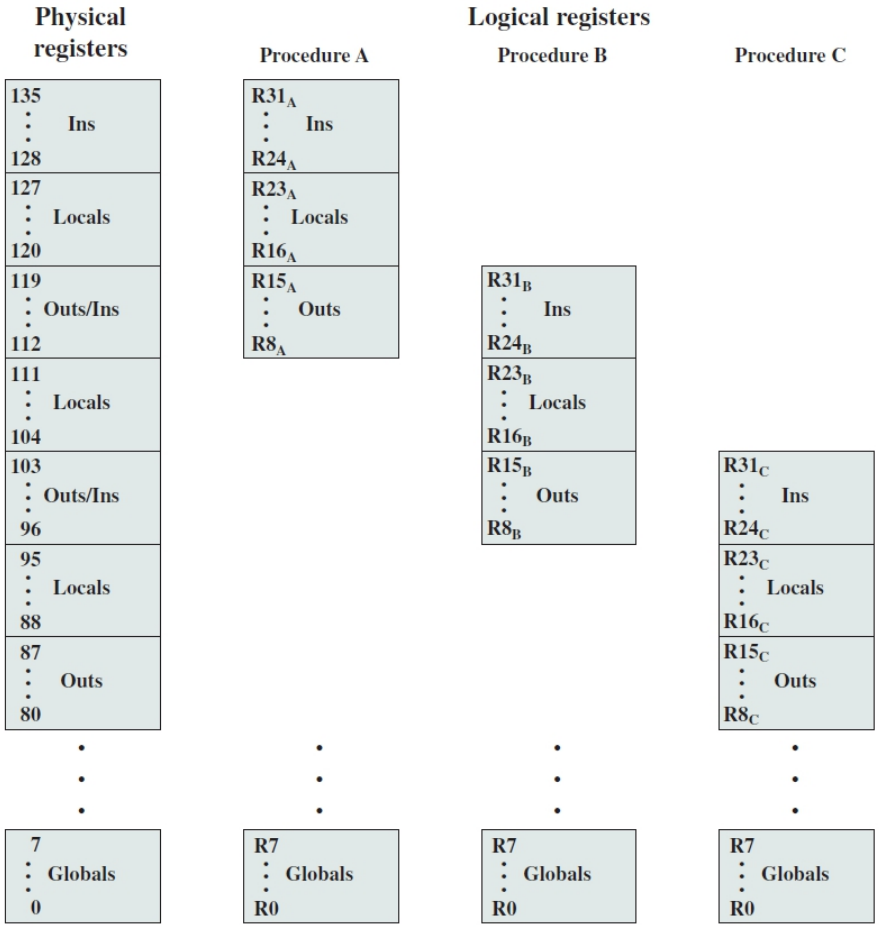
IF = Instruction fetch first half
IS = Instruction fetch second half
RF = Fetch operands from register
EX = Instruction execute
IC = Instruction cache
DC = Data cache
DF = Data cache first half
DS = Data cache second half
TC = Tag check
WB = Write back to register file

Theoretical R3000 and Actual R4000 Super pipelines

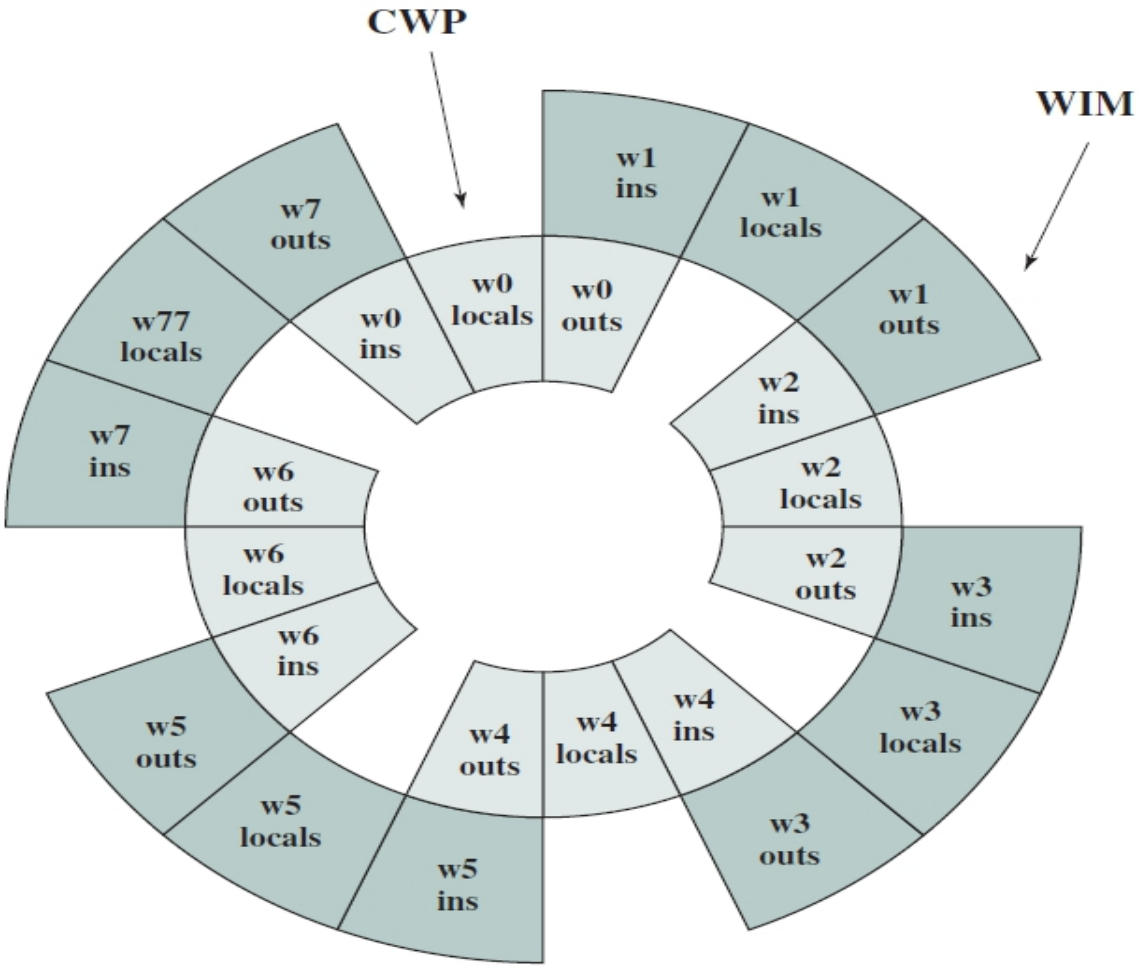
SPARC

SPARC

- SPARC (Scalable Processor Architecture) refers to an architecture defined by Sun microsystems.
- **SPARC Register Set**
- SPARC makes use of register windows
- Each window gives addressability to 24 registers
- total number of windows ranges from 2 to 32



SPARC Register Window Layout with Three Procedures



Eight Register Windows Forming a Circular Stack in SPARC

- **Circular stack in SPARC**
- the register overlap
- The calling procedure places any parameters to be passed in its *outs registers*
- *the called procedure treats* these same physical registers as it *ins registers.*
- *The processor maintains a current* window pointer (CWP), located in the processor status register (PSR), points to the window of the currently executing procedure.
- The window invalid mask (WIM) in the PSR, indicates which windows are invalid.

Instruction Set

- Register-to-register instructions have three operands and can be expressed in the form

$$R_d \rightarrow R_{S1} \text{ op } S2$$

- Rd and RS1 are register references;
- S2 can refer either to a register or to a13-bit immediate operand
- Register zero (R0) is hardwired with the value 0.

The available ALU operations can be grouped as

- Integer addition (with or without carry).
- Integer subtraction (with or without carry).
- Bitwise Boolean AND, OR, XOR and their negations.
- Shift left logical, right logical, or right arithmetic.

- All of these instructions, except the shifts, can optionally set the four condition codes (ZERO, NEGATIVE, OVERFLOW, CARRY).
- Signed integers are represented in 32-bit twos complement form.
- In Displacement mode the effective address (EA) of an operand consists of a displacement from an address contained in a register: depending on whether the second operand is immediate or a register reference

  EA = (RS1) + S2
- or EA = (RS1) + (RS2)
- second stage, the memory address is calculated using the ALU
- third stage, the load or store occur
- single addressing mode is quite versatile and can beused to synthesize other addressing modes.

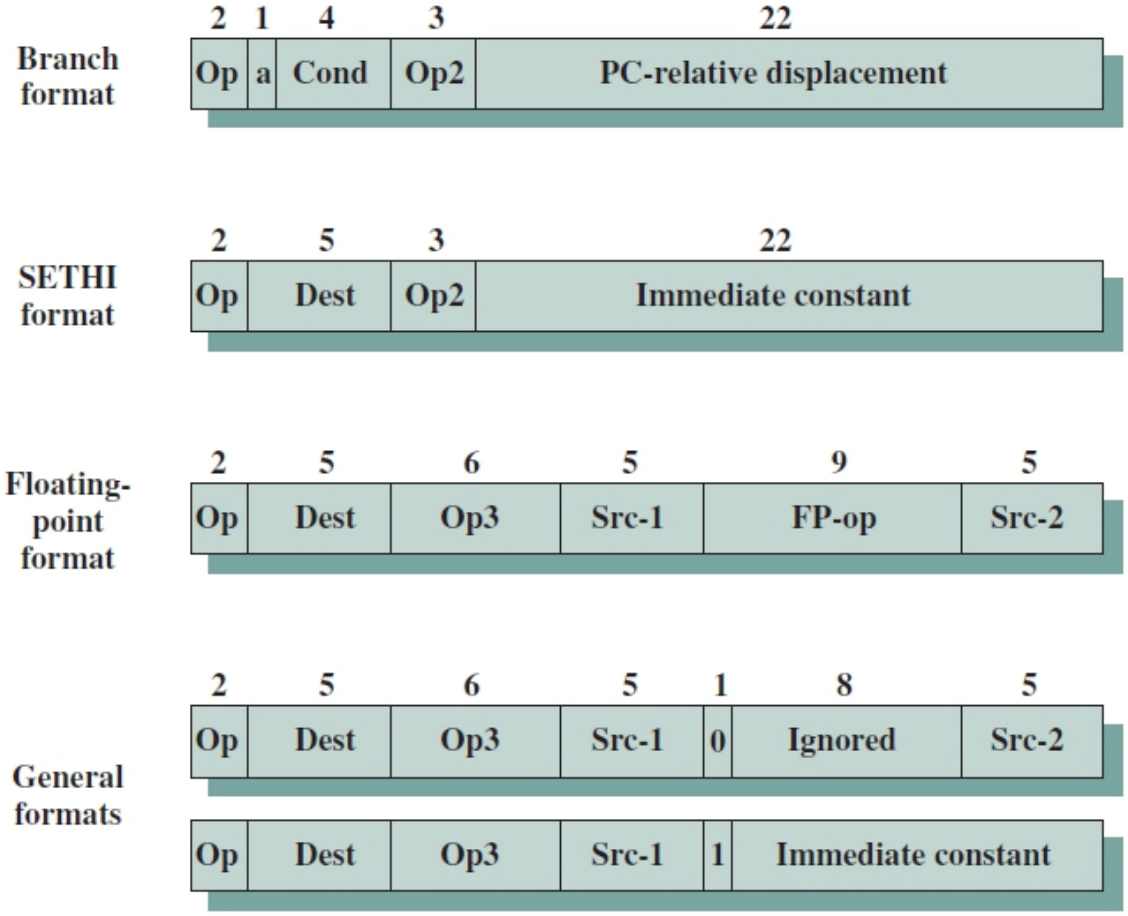
Synthesizing Other Addressing Modes with SPARC Addressing Modes

Instruction Type	Addressing Mode	Algorithm	SPARC Equivalent
Register-to-register	Immediate	operand = A	S2
Load, store	Direct	EA = A	$R_0 + S_2$
Register-to-register	Register	EA = R	R_{S1}, S_{S2}
Load, store	Register Indirect	EA = (R)	$R_{S1} + 0$
Load, store	Displacement	EA = (R) + A	$R_{S1} + S2$

Instruction Format

- MIPS R4000, SPARC uses a simple set of 32-bit instruction formats
- Instructions begin with a 2-bit opcode
- Call instruction,a 30-bit immediate operand is extended with two zero bits to the right to form a 32-bit PC-relative address in twos complement form.
- Instructions are aligned on a32-bit boundary so that this form of addressing suffices.

Instruction Format



SPARC Instruction Formats

RISC VERSUS CISC CONTROVERSY

RISC Versus CISC Controversy

RISC approach can be grouped into two categories:

Quantitative: Attempts to compare program size and execution speed of programs on RISC and CISC machines that use comparable technology.

Qualitative: Examines issues such as high-level language support and optimum use of VLSI real estate.

Problems

-No pair of RISC and CISC that are directly comparable

-No definite set of test programs

-Difficult to separate hardware effects from compiler effects

-Most comparisons done on toy rather than production machines

-Most commercial devices are a mixture of RISC and CISC characteristics

THANK YOU