+

Chapter 15 Reduced Instruction Set Computers (RISC)

William Stallings, Computer Organization and Architecture, 9th Edition

Introduction

Two trends in CPU architecture:

- **CISC**: Complex Instruction Set Computing/Computer such as IBM System/360, PDP-11, Motorola 6809, 68000, Intel 8080, x86,… CPU is set up to execute many instructions.

- **RISC**: Reduced Instruction Set Computing/Computer: Idea: All complex instruction is association of some basic instructions. So, a smaller set of basic instructions is needed. Examples: Sun UltraSPARC microprocessor

- More details:

  http://en.wikipedia.org/wiki/Complex_instruction_set_computing

  http://en.wikipedia.org/wiki/Reduced_instruction_set_computing

Comparing several RISC and Non-RISC Systems

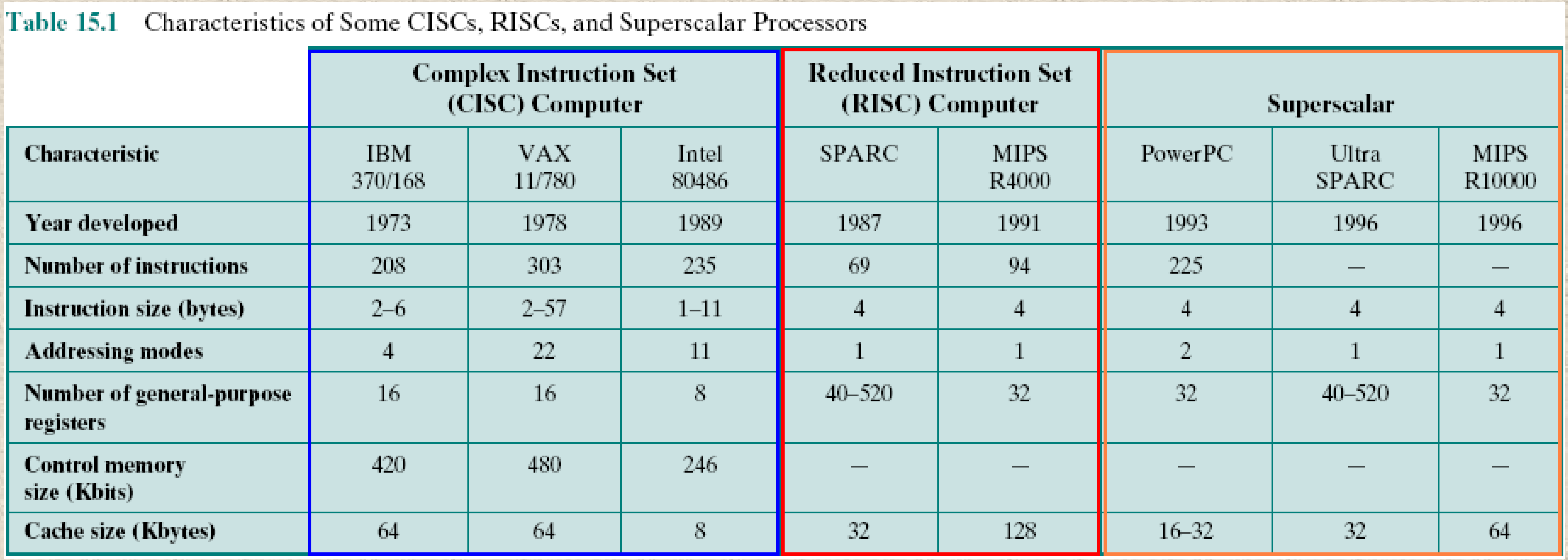


Table 15.1 Characteristics of Some CISCs, RISCs, and Superscalar Processors

	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

Scalar processor: CPU processes one datum at a time
Vector processor: CPU processes multiple data items at a time
Superscalar processor: Architecture implements a form of parallelism called instruction-level parallelism within a single processor. (Wiki)

Objectives

After studying this chapter, you should be able to:

- 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.

Contents

15.1- Instruction Execution Characteristics

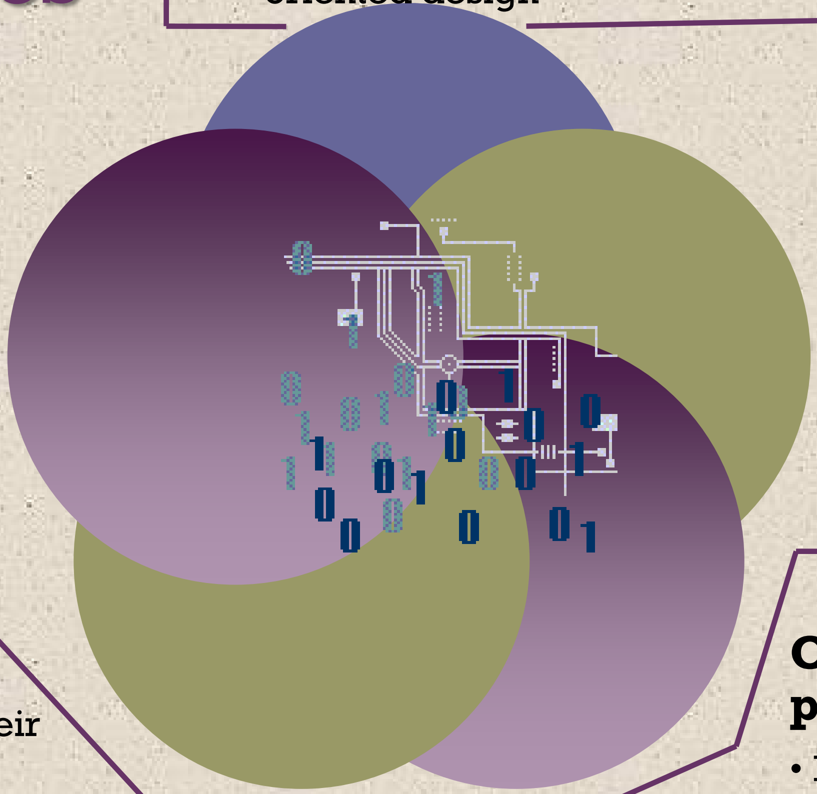
High-level languages (HLLs)

- 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

Requirements

Semantic gap

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



Execution sequencing

- Determines the control and pipeline organization

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

Responses from architecture

Operations and Operands are used:

Statistic

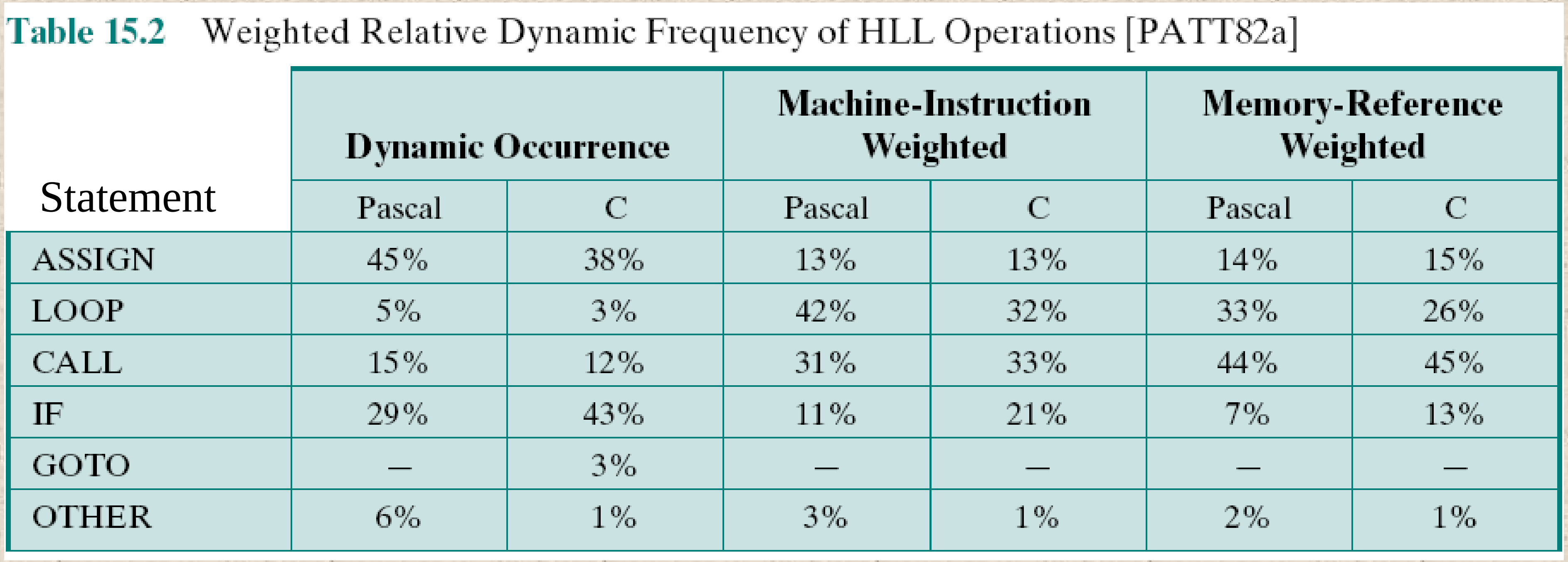


Table 15.2 Weighted Relative Dynamic Frequency of HLL Operations [PATT82a]

Statement	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%

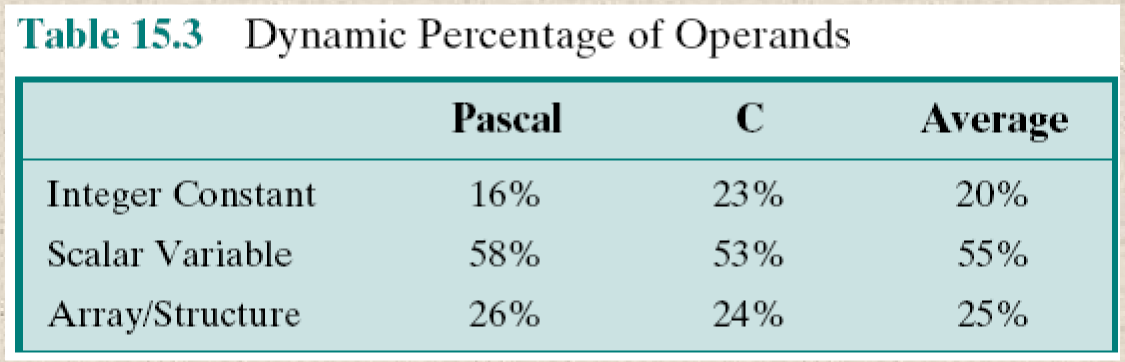


Table 15.3 Dynamic Percentage of Operands

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

Procedure Call: Arguments and Local Scalar Variables

Statistic

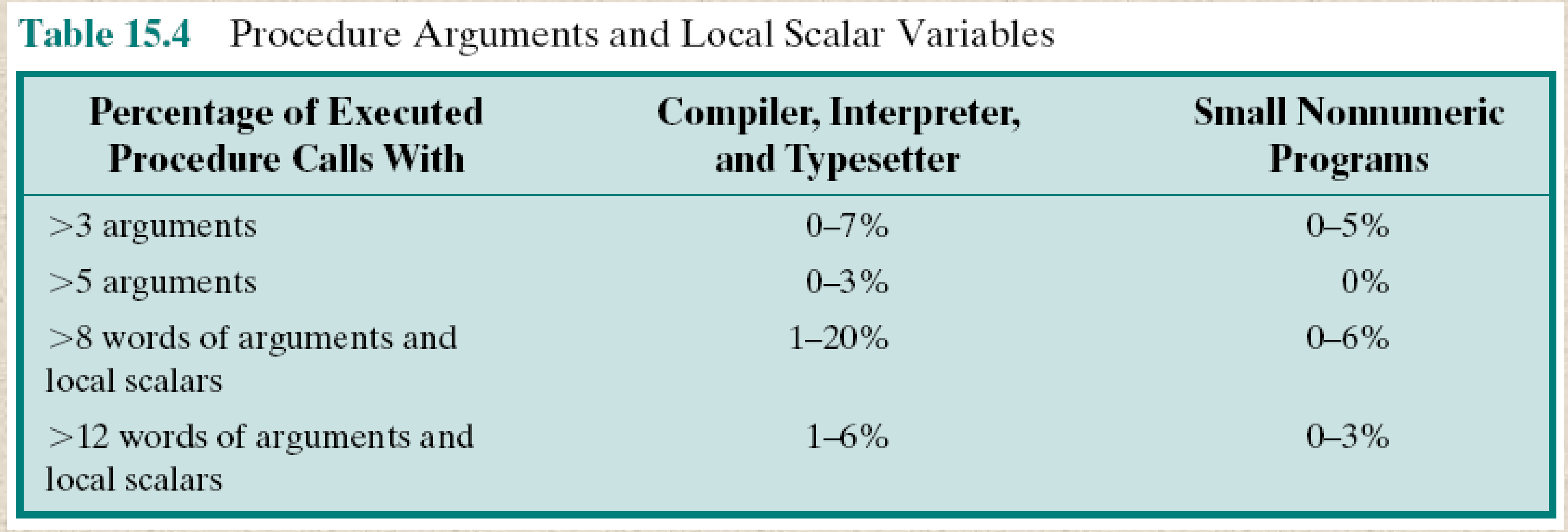


Table 15.4 Procedure Arguments and Local Scalar Variables

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%

Scalar variable: Simple variable storing only one value

+

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

+

15.2- The Use of a Large Register File

Registers are accessed faster than cache or memory
➔ More registers are used

Software Solution	Hardware Solution
Requires compiler to allocate registers	More registers
Allocates based on most used variables in a given time	Thus more variables will be in registers
Requires sophisticated (complex) program analysis	

+

Overlapping Register Windows

Register windows is a group of registers which are used to pass arguments between procedure calls.

The use of **register windows** is a technique to improve the performance of a particularly common operation, the procedure call. This was one of the main design features of the original Berkeley RISC design, which would later be commercialized as the SPARC, AMD Am29000, and Intel i960 (Wiki).

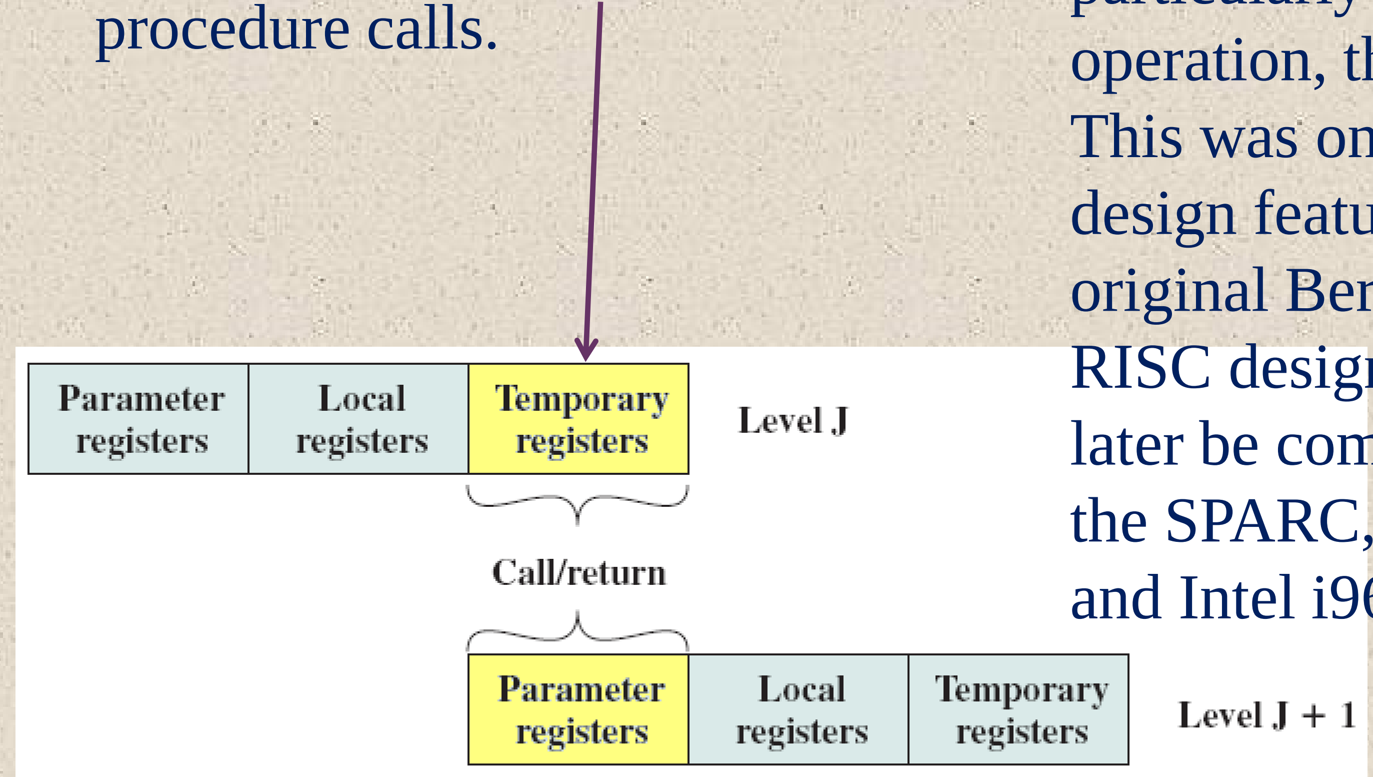


Figure 15.1 Overlapping Register Windows

Circular Buffer Organization of Overlapped Windows

A called B; B called C; C called D

The procedure D is active

Overlapped registers are used between callings.
A curcular chain of register references is created

If the procedure F makes preparation to call another procedure, registers of A are conflicted and an interrupt must be thrown ➔ N windows permits N-1 calls only

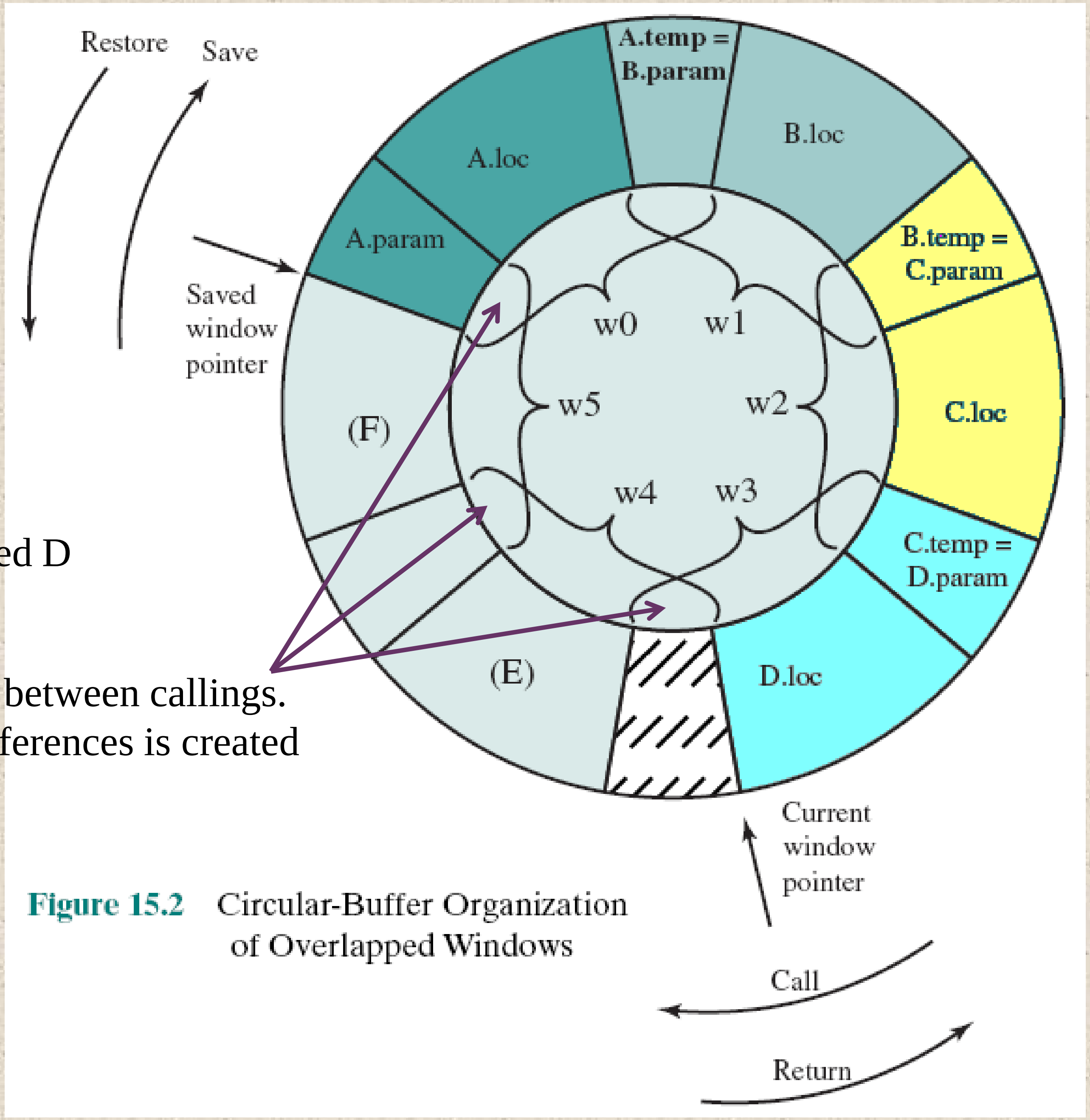


Figure 15.2 Circular-Buffer Organization of Overlapped Windows

+

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 (gánh nặng) to accommodate (supply) the split in register addressing
- In addition, the linker (a part of compiler) must decide which global variables should be assigned to registers

Large-Register-File vs. Cache

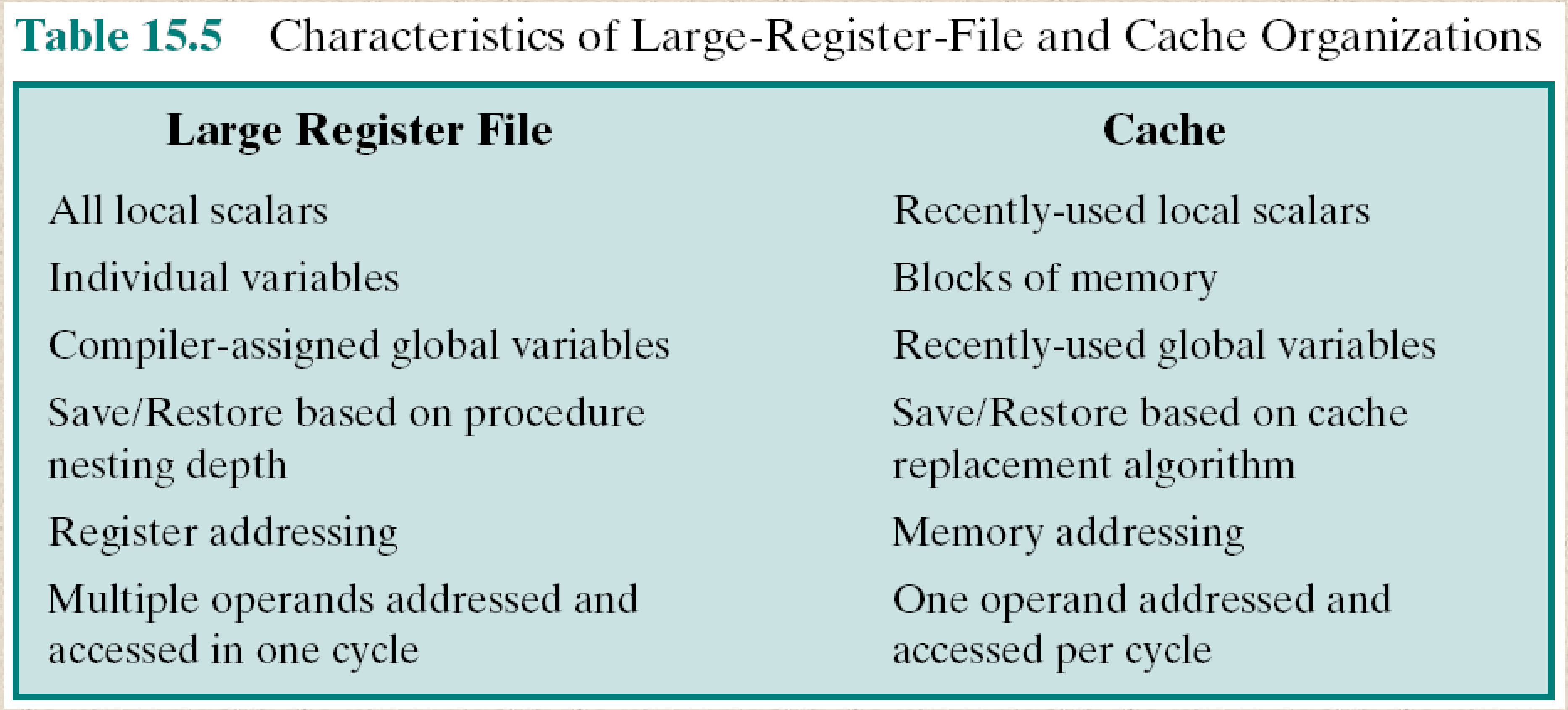


Table 15.5 Characteristics of Large-Register-File and Cache Organizations

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

+

Referencing a Scalar

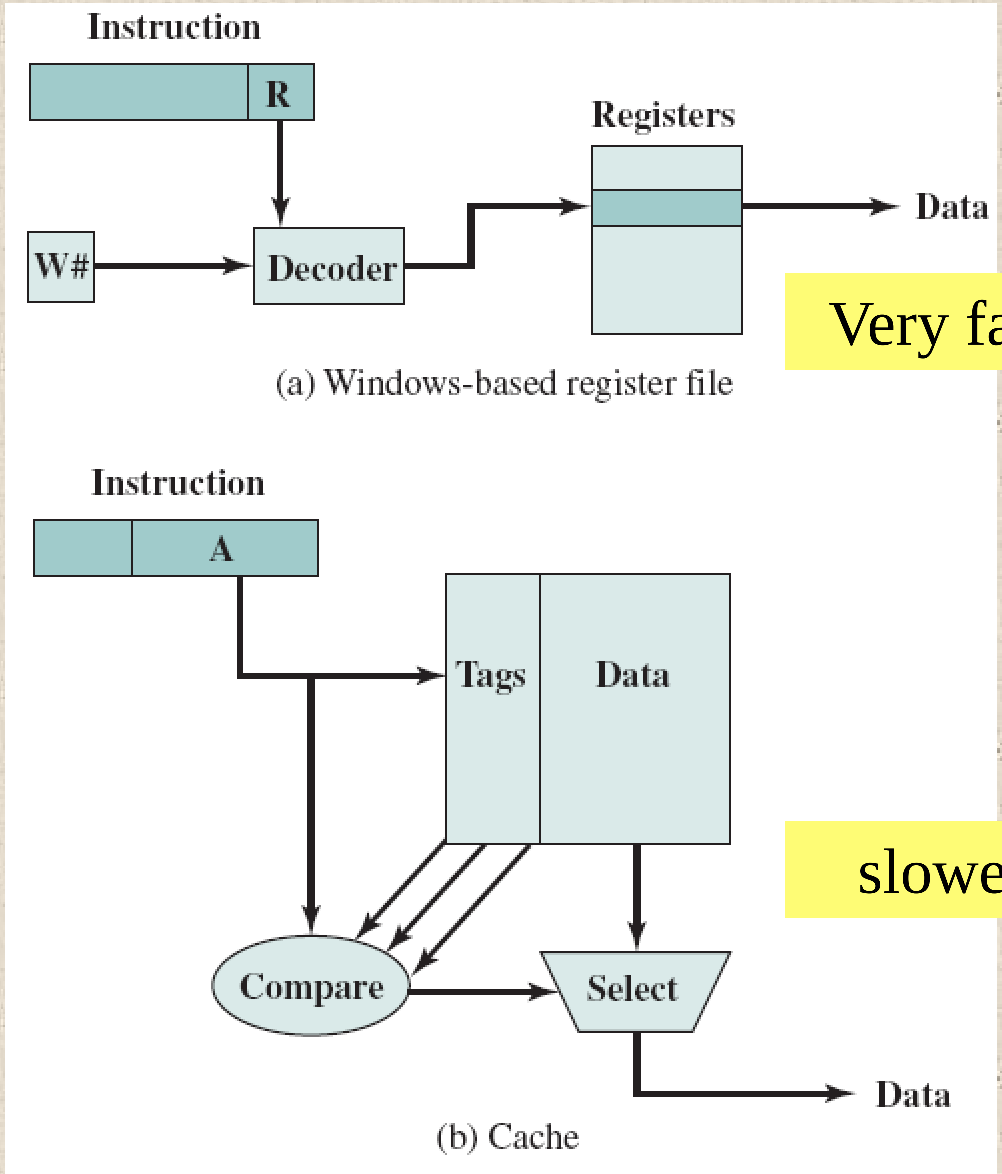


Figure 15.3 Referencing a Scalar

Very fast

slower

12.5- RISC Pipelining

Instruction pipelining is often used to enhance performance. Most instructions in RISC are register to register.

Instruction cycle: two stages:

• I: Instruction fetch.

• E: Execute, ALU operation, Input and output are registers.

Load and store operations, three stages:

I: Instruction fetch.

E: Execute. Calculates memory address.

D: (direction) Memory. Register-to-memory or memory-to-register operation.

The Effects of Pipelining: An Example

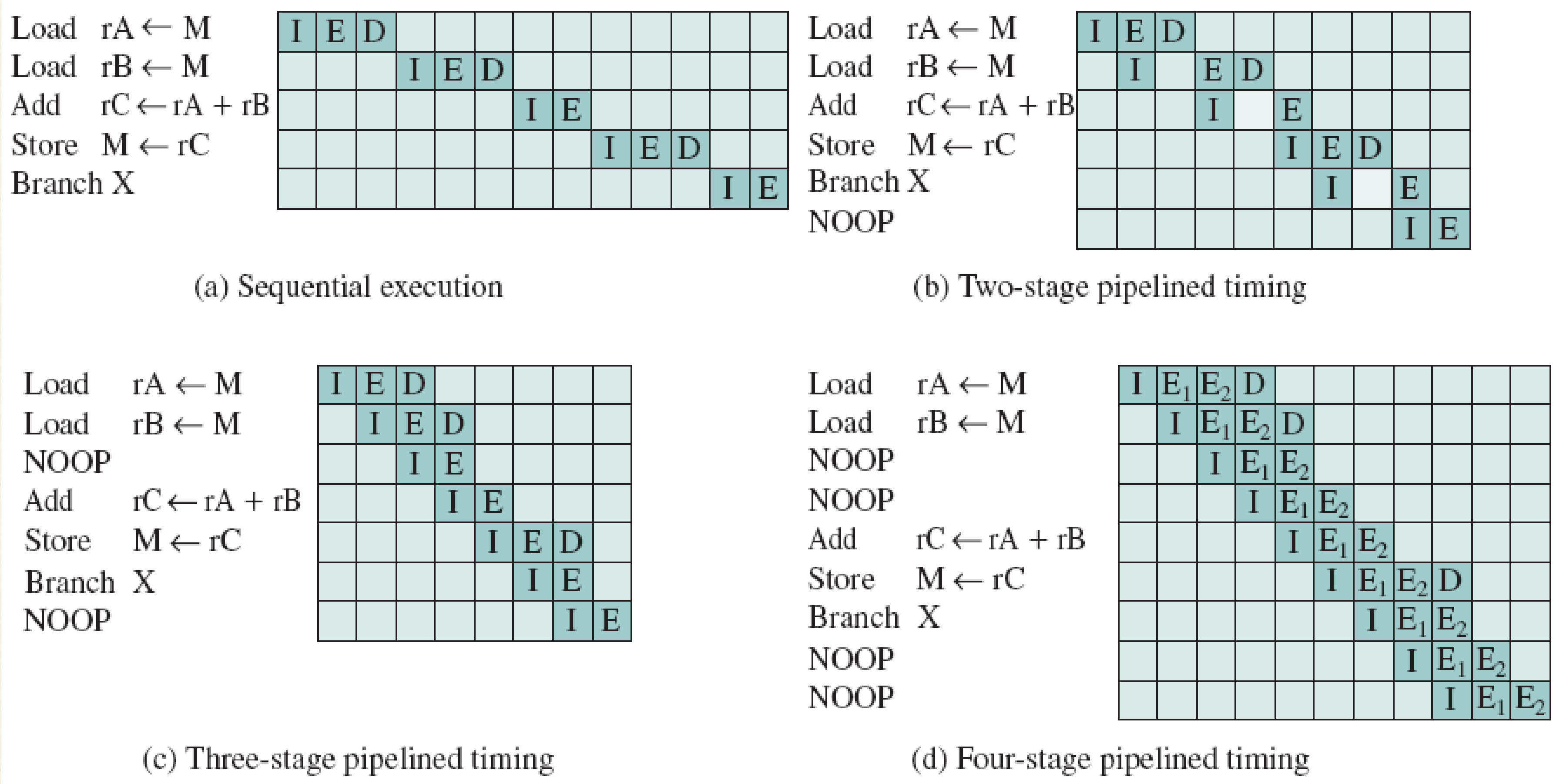


Figure 15.6 The Effects of Pipelining

NOOP: No operation → Wait

+

Optimization of Pipelining

- **Delayed branch**
  - Does not take effect until after execution of **following instruction**
  - This location immediately following the branch is the delay slot → Insert the instruction NOOP
- **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 (mở rộng vòng lặp)**
  - 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

Target of JUMP is delayed → ADD is executed before STORE

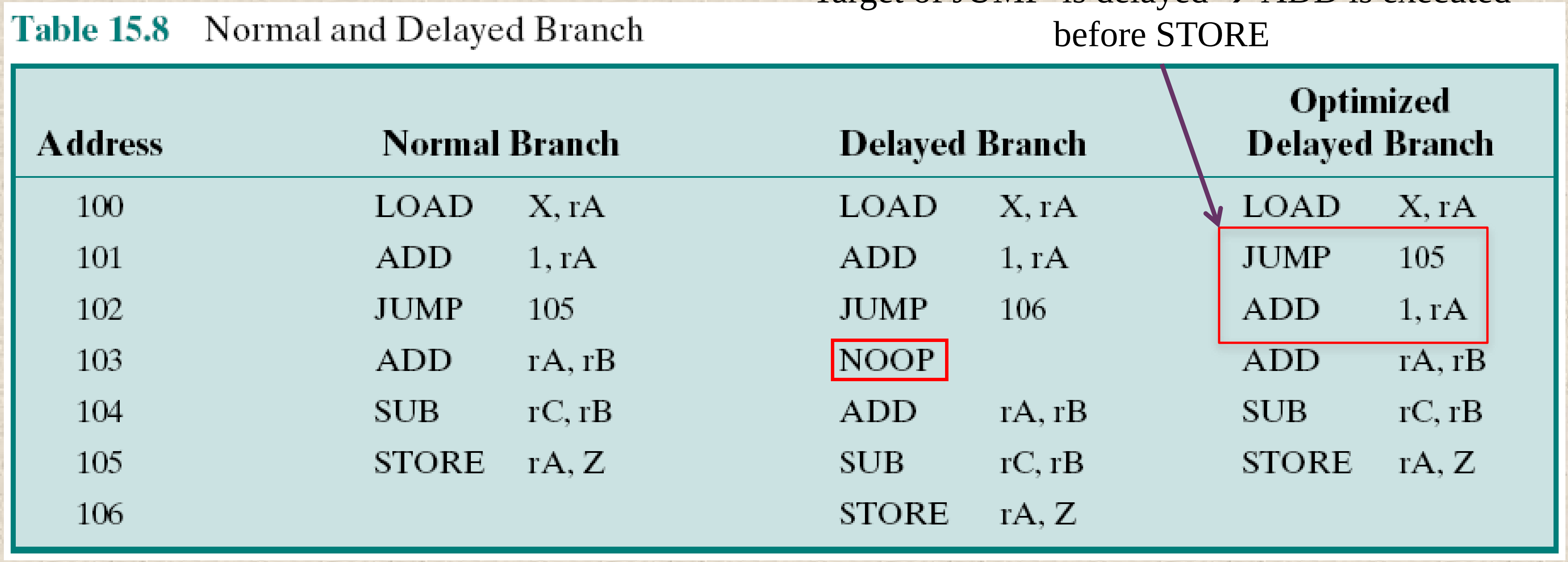


Table 15.8 Normal and Delayed Branch

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		

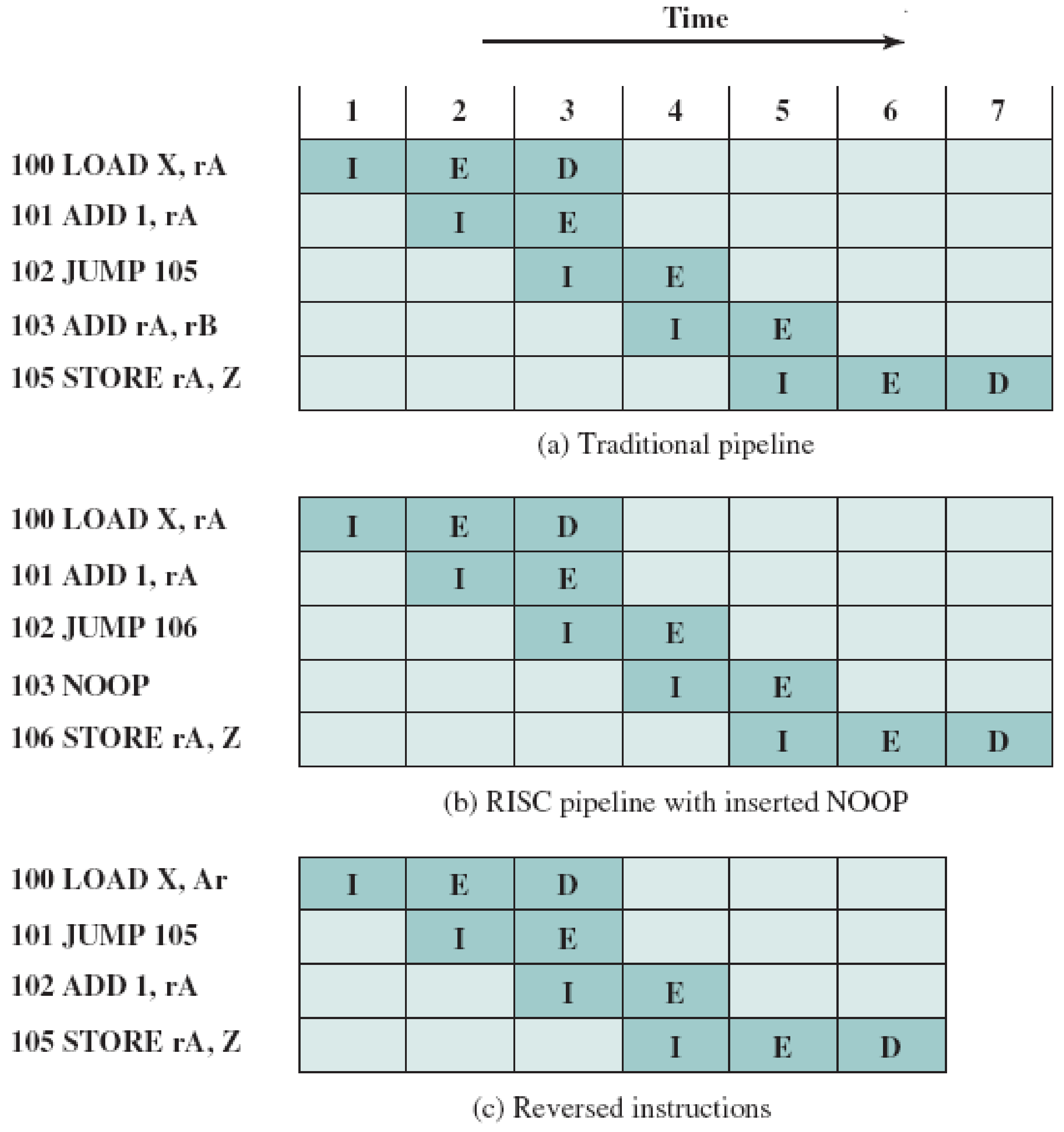
After 102 is executed, the next instruction to be executed is 105

To regularize the pipeline, a NOOP is inserted after this branch (previous slide)

Increased performance is achieved **only** if the instructions at 101 and 102 are interchanged.

Use of the Delayed Branch

Program in the table 15.6


Time →

	1	2	3	4	5	6	7
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
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

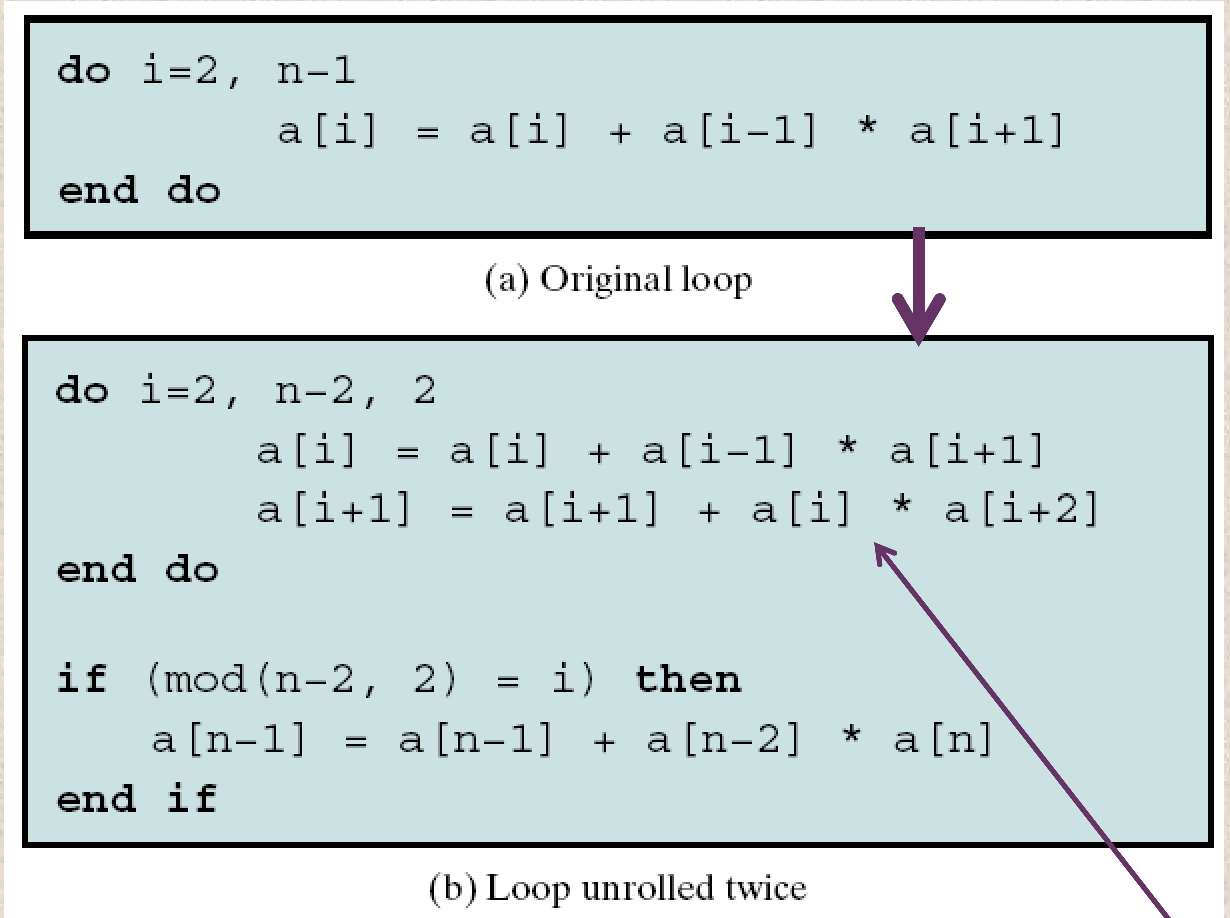
(b) RISC pipeline with inserted NOOP

	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

Figure 15.7 Use of the Delayed Branch

Loop Unrolling Twice 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

Figure 15.8 Loop Unrolling

Compiler technique to improve instruction parallelism is loop unrolling .
Unrolling can improve the performance by:
Reducing loop overhead, increasing instruction parallelism by improving pipeline performance, improving register, data cache, or TLB locality

Number of loops decreases 2 times

+

15.8-RISC versus CISC Controversy

- **Quantitative – So sánh định lượng**
  - Compare program sizes and execution speeds of programs on RISC and CISC machines that use comparable technology
- **Qualitative – so sánh chất lượng**
  - Examine issues of high level language support and use of VLSI real estate (very large scale integration chip)
- **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 complier effects
  - Most comparisons done on "toy" rather than commercial products
  - Most commercial devices advertised as RISC possess a mixture of RISC and CISC characteristics

Chưa biết mèo nào cắn mèo nào!
The battle has no end!

Controversy: tranh luận

+

Exercises

- 15.1 What are some typical distinguishing characteristics of RISC organization?
- 15.2 Briefly explain the two basic approaches used to minimize register-memory operations on RISC machines.
- 15.3 If a circular register buffer is used to handle local variables for nested procedures, describe two approaches for handling global variables.
- 15.4 What are some typical characteristics of a RISC instruction set architecture?
- 15.5 What is a delayed branch?

+

Summary

Chapter 15

Reduced Instruction Set Computers (RISC)

- **Instruction execution characteristics**
  - Operations
  - Operands
  - Procedure calls
  - Implications
- **The use of a large register file**
  - Register windows
  - Global variables
  - Large register file versus cache
- **RISC pipelining**
  - Pipelining with regular instructions
  - Optimization of pipelining
- **Compiler-based register optimization**
- **RISC versus CISC controversy**