William Stallings
Computer Organization and Architecture
7th Edition

Chapter 13

Reduced Instruction Set Computers

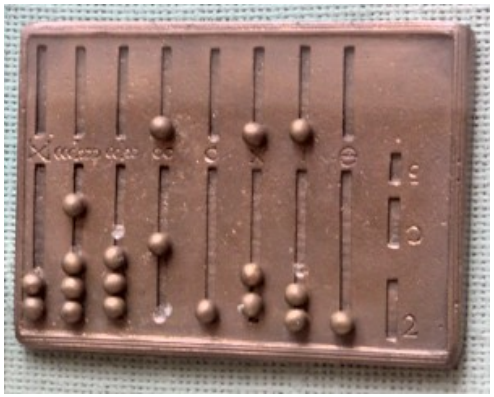
Key points

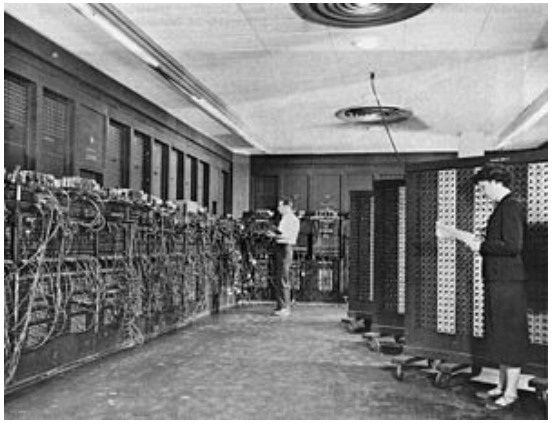
- Studies of the execution behavior of ***high-level language programs*** provide guidance in designing RISC:
  - Assignment predominate->simple ***movement of data*** should be optimized
  - Many IF and LOOP instructions->***sequence control*** mechanism needs to be optimized for pipelining
  - Operand reference patterns->keeping a moderate number of operands in ***registers***
- Key characteristics of RISC
  - a ***limited instruction set*** with a ***fixed*** format (*RI*)
  - A ***large number of registers***, optimization of register usage (*Fast*)
  - Optimized ***pipeline*** (*Efficient*)
- RISC is efficient for pipeline (*Why?* ***delay branch*** technique is fit for RISC)

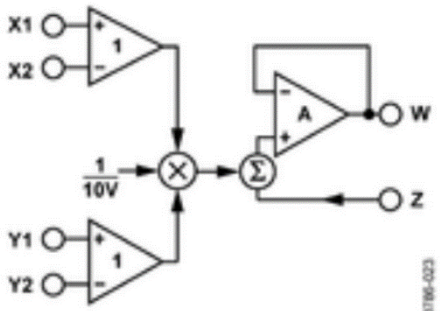
Major Advances in Computers

- The ***family*** concept
  - —IBM System/360 1964
  - —DEC PDP-8
  - —Separates *architecture* from *implementation*

X*Y=?







The Roman Abacus, 500 B.C. The ENIAC, 1946

Major Advances in Computers

- ***Microprogrammed*** control unit
  - —Idea by Wilkes 1951
  - —Produced by IBM S/360 1964
- ***Cache*** memory
  - —IBM S/360 Model 85 in 1968
- ***Pipelining***
  - —Introduces *parallelism* into fetch execute cycle
- ***Multiple processors***

The Next Step - RISC

- **R**educed **I**nstruction **S**et **C**omputer
- Key features
  - Large number of general purpose ***registers***
  - or use of compiler technology to optimize register use
  - ***Limited and simple*** instruction set
  - Emphasis on optimising the instruction ***pipeline***

Comparison of 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

13.1 instruction execution characteristics

- Operations performed
- Operands used
- Execution sequencing

Driving force for CISC

- Software costs far exceed hardware costs
- Increasingly complex *high level languages*
- Semantic gap
- Leads to:
  - Large instruction sets
  - More addressing modes
  - Hardware implementations of HLL statements
    - e.g. CASE (switch) on VAX

Intention of CISC

- Ease compiler writing
- Improve execution efficiency
  - —Complex operations in microcode
- Support more complex HLLs

A different approach

- Make the architecture that support the HLL simpler, rather than more complex

Execution Characteristics

- ***Operations performed***
- ***Operands used***
- ***Execution sequencing***
- Studies have been done based on programs written in HLLs
- *Dynamic studies* are measured during the execution of the program

Operations

- Assignments
  - —Movement of data
- Conditional statements (IF, LOOP)
  - —Sequence control
- Procedure *call-return* is very time consuming
- Some HLL instruction lead to *many machine code operations*
- (*next diagram*)

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%

Operands

- Mainly local scalar variables
- Optimisation should concentrate on accessing local variables

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

Procedure Calls

- Very time consuming
- Depends on *number of parameters* passed
- Depends on *level of nesting*
- Most programs do not do a lot of calls followed by lots of returns
- Most variables are local
- (c.f. ***locality of reference***)

Implications

- Best support is given by optimising *most used* and *most time consuming* features
- Large number of ***registers***
  - —Operand referencing
- Careful design of ***pipelines***
  - —Branch prediction etc.
- **S**implified (**r**educed) **i**nstruction **s**et

13.2 The use of a large register file

- Register windows
- Global variables
- Large register file versus cache

Large Register File

- Software solution
  - —Require *compiler* to allocate registers
  - —Allocate based on *most used variables* in a given time
  - —Requires *sophisticated program analysis*
- Hardware solution
  - —Have *more registers*
  - —Thus more variables will be in registers

Registers for Local Variables

- Use of *large set of registers* reduces memory access
- Design task is to *organize the registers*
- Store ***local scalar*** variables in registers
- Every procedure (function) call changes ***locality***
  - —*Parameters* must be passed
  - —*Results* must be returned
  - —*Variables* from calling programs must be restored

Register Windows

- Only *few* parameters
- *Limited* range of depth of call

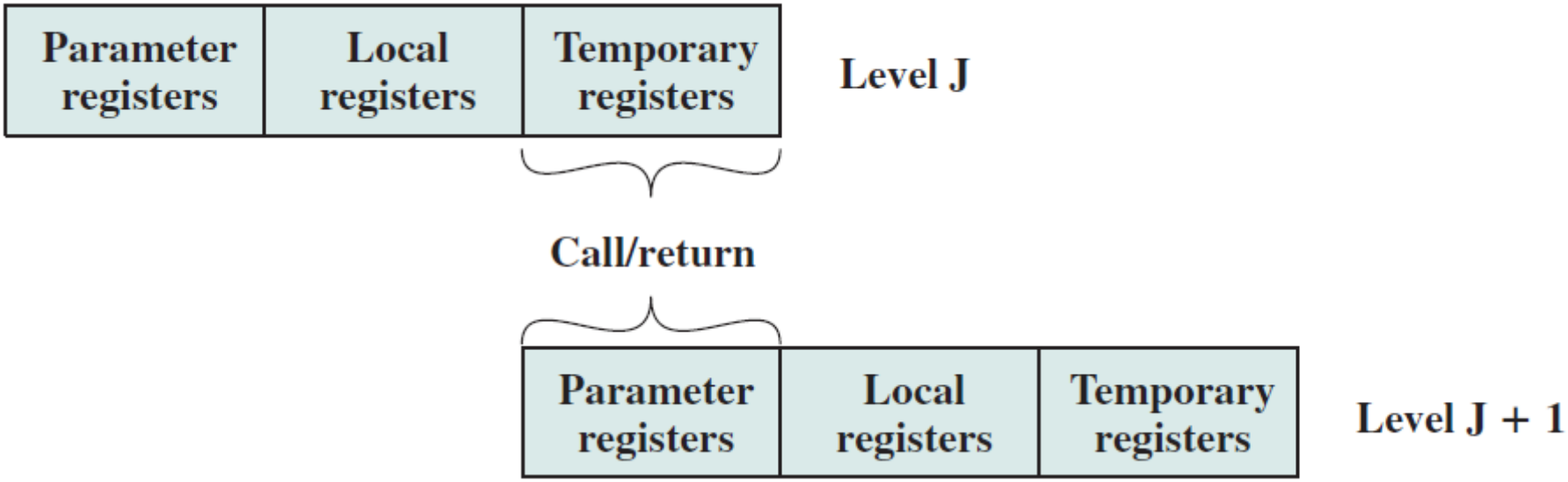
- Use *multiple small sets* of registers
- Calls *switch to* a different set of registers
- Returns *switch back to* a previously used set of registers

Register Windows cont.

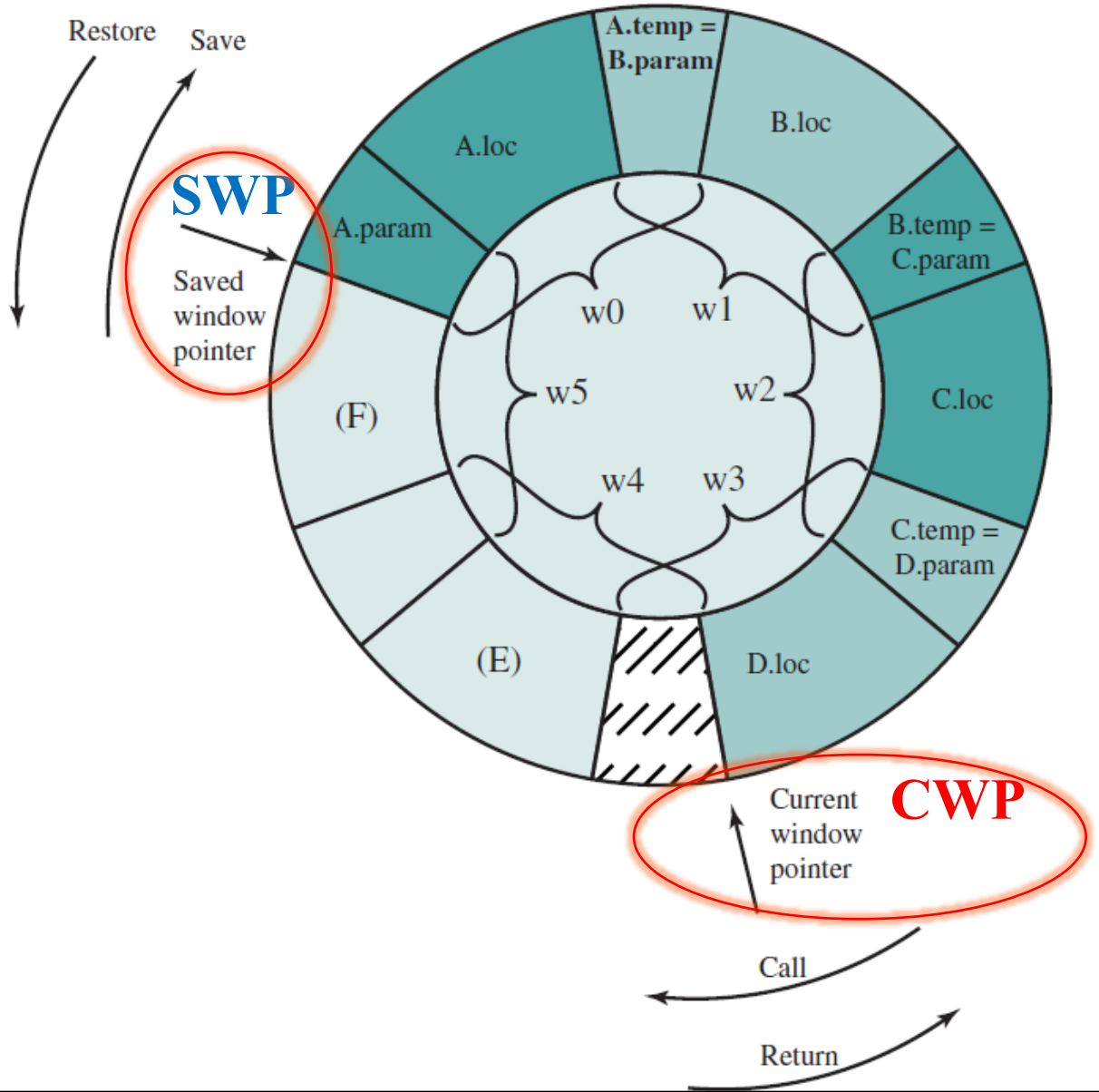
- Three areas within a register set
  - **Parameter** registers
  - **Local** registers
  - **Temporary** registers

  - Temporary registers from one set ***overlap*** parameter registers from the next
  - This allows parameter passing *without moving data*

Overlapping Register Windows



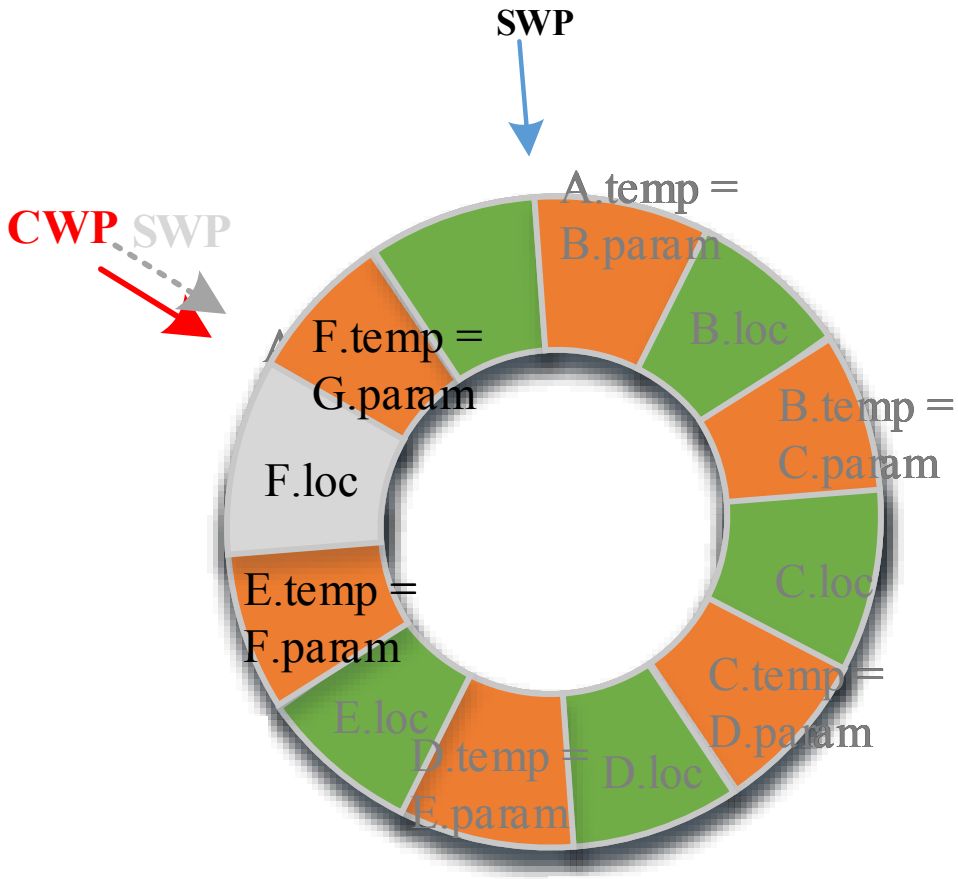
Circular Buffer diagram

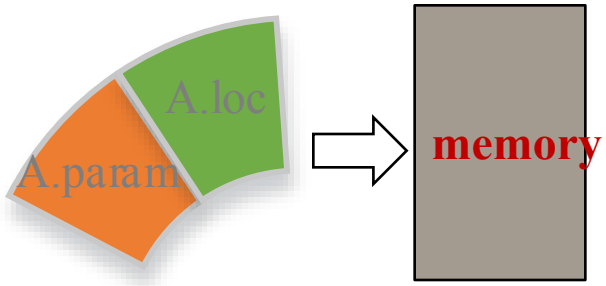


Operation of Circular Buffer

- When a call is made, a current window pointer is moved to show the currently active register window
- If all windows are *in use*, an **interrupt** is generated and the oldest window (the one furthest back in the call nesting) is *saved* to *memory*
- A *saved window pointer* indicates where the next saved windows should *restore* to
- An N-window register file can hold only *N-1 procedure* activations

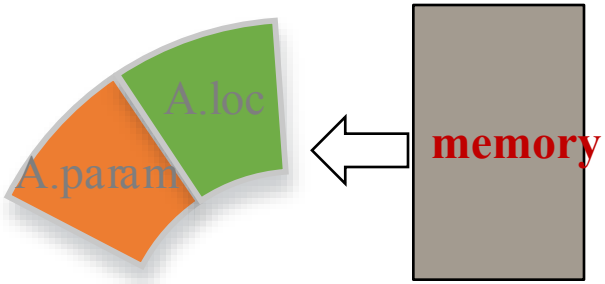
Operation of Circular Buffer





When CWP is incremented and becomes equal to SWP,

- **An interrupt occurs,**
- **Window A is saved to memory**
- **The SWP is incremented**
- **The call to F proceeds.**



When B returns to A, CWP is decremented and becomes equal to SWP. This causes an interrupt that results in the **restoration** of A's window.

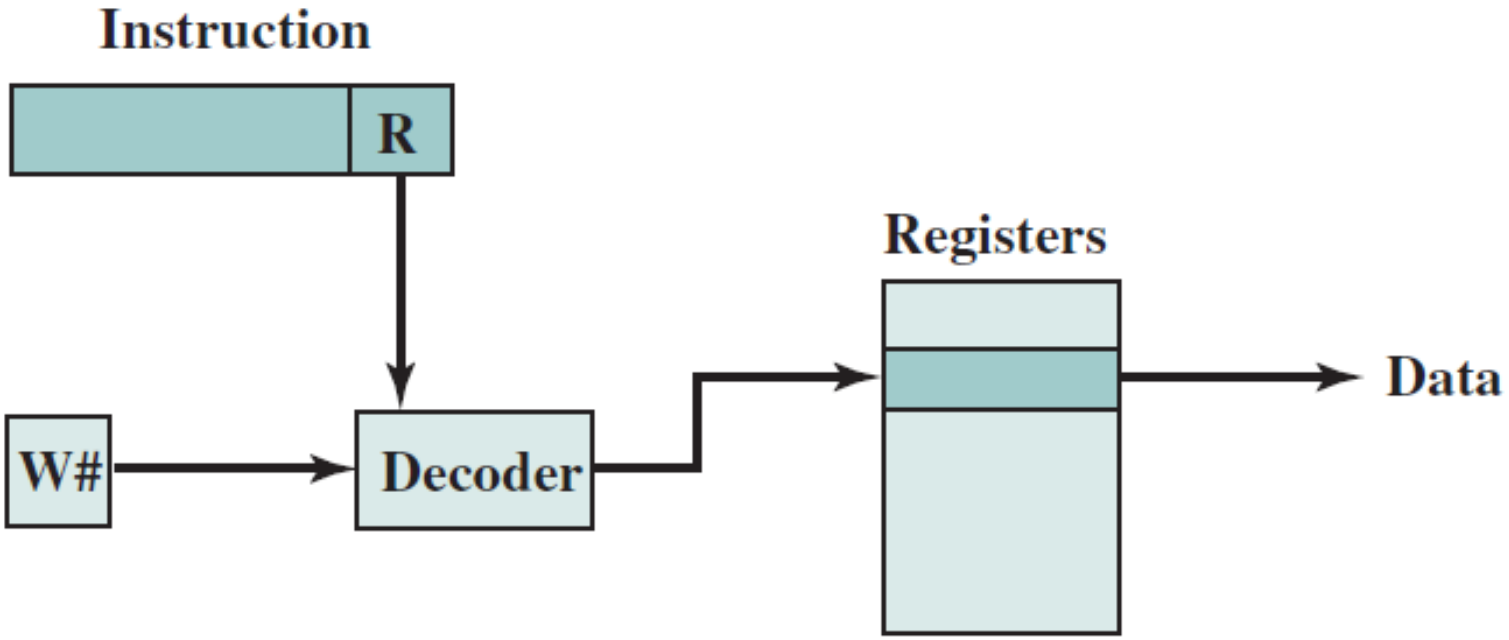
Global Variables

- Allocated by the compiler to *memory*
  - —Inefficient for frequently accessed variables
- Have a set of registers for *global variables*

Registers v Cache

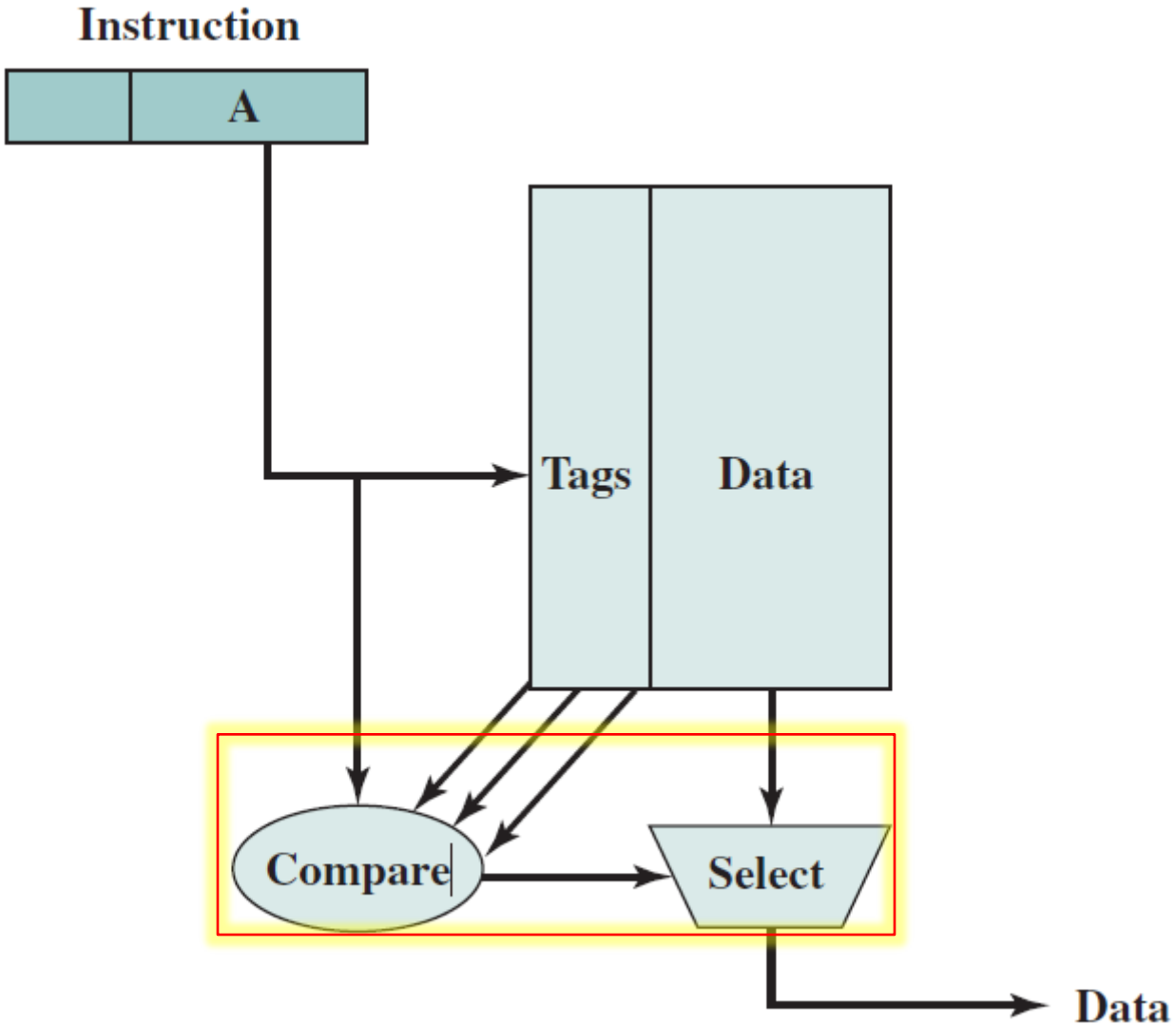
Large Register File	Cache
All **local** scalars	**Recently-used local** scalars
Individual variables	**Blocks** of memory
Compiler-assigned global variables	**Recently-used global** variables
Save/Restore based on procedure **nesting depth**	Save/Restore based on cache **replacement** algorithm
Register addressing	**Memory** addressing

Referencing a Scalar - Window Based Register File



(a) Windows-based register file

Referencing a Scalar - Cache



(b) Cache

Homework

- Reading chapter 13
- Translate Key terms

complex instruction set computer (CISC) delayed branch delayed load	high-level language (HLL) reduced instruction set computer (RISC)	register file register window SPARC

- Review Questions
  - 1,2

13.3 Compiler Based Register Optimization

- Assume *small number* of registers (16-32)
- Optimizing use is up to *compiler*
- HLL programs have no explicit references to registers
- Compiler task
  - —*Assign* symbolic or virtual register to each candidate variable

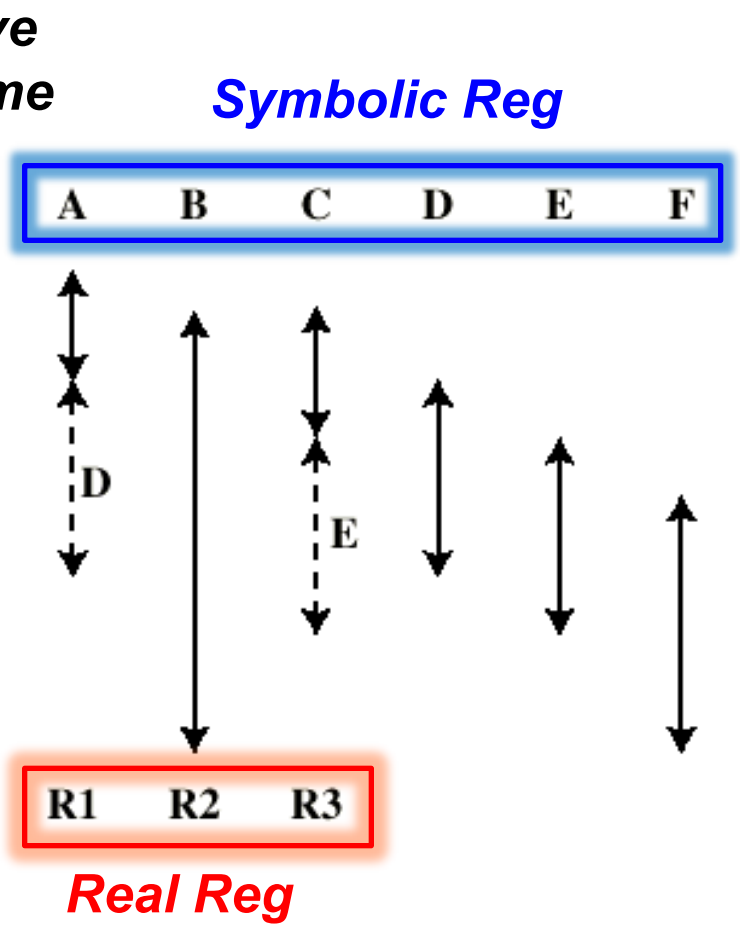
    (register int a; //for Turbo C2.0, limited by 2;)
  - —*Map* (*unlimited*) symbolic registers to real registers
  - —Symbolic registers that *do not overlap* (?) can share real registers
- If you run out of real registers some variables use *memory*

Graph Coloring – Mapping Method

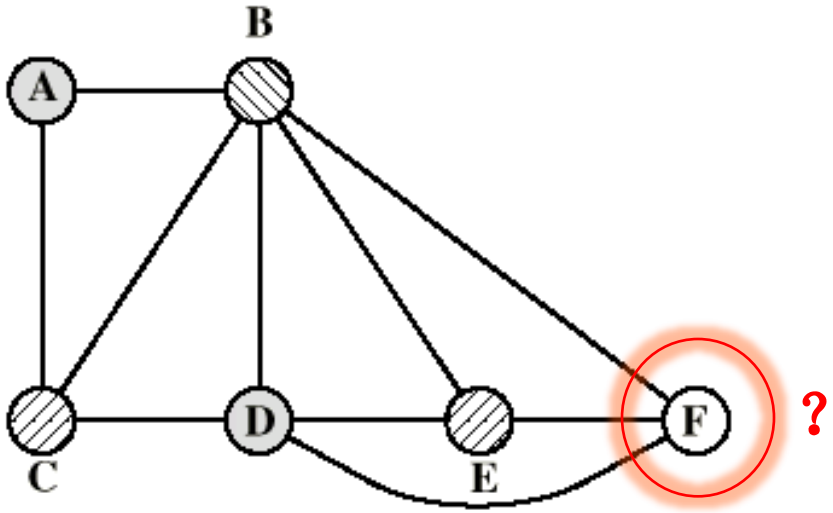
- Given a graph of *nodes* and *edges*
- Assign a color to each node
- Adjacent nodes have different colors
- Use minimum number of colors

- Nodes are symbolic *registers*
- Two registers that are *live in the same program fragment* are joined by an edge
- Try to color the graph with *n* colors, where *n* is the number of real registers (diagram)
- Nodes that can not be colored are placed in ***memory***

Graph Coloring Approach



(a) Time sequence of active use of registers



(b) Register interference graph

13.4 Reduced instruction set architecture

- Why CISC
- RISC Characteristics
- RISC v CISC

Why CISC

- Compiler simplification?
  - Complex machine instructions *harder* to exploit
  - Optimization more *difficult*
- Smaller programs?
  - Program takes up less memory but memory is now *cheap*
  - May **not** occupy less bits (Table 13.6)
- Faster programs?
  - Bias towards use of *simpler* instructions
  - More *complex* control unit
  - Microprogram control store *larger*
  - thus simple instructions take longer to execute
- It is far from clear that CISC is the appropriate solution

RISC Characteristics

- One *machine* instruction per *machine* cycle
  - Machine Cycle: Read Operand + ALU + Write Operand
  - no need for Microprogram/microprogram control store
  - Thus **faster** than CISC
- Register to register operations
  - most operations from Reg to Reg
  - only LOAD, STORE for memory access
  - Thus, **Simpler (?)** and **faster (?)** than CISC
- Few, simple addressing modes
  - most **Reg** addressing mode
- Few, simple instruction formats
  - fixed instruction length, word unit (adv**?**)
  - fixed field configuration (adv**?**)

RISC v CISC

- Not clear cut – CISC / RISC

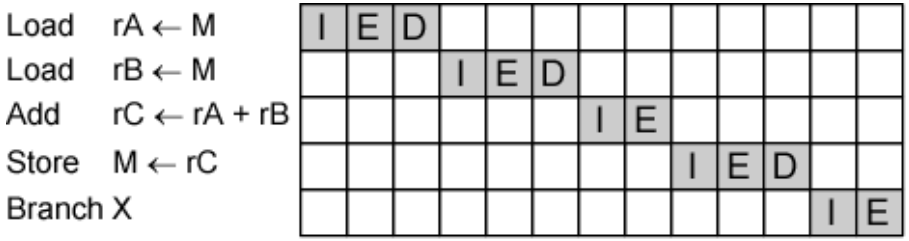
  who is stronger than the other ???

- Many designs borrow from both philosophies
- e.g. PowerPC (not "pure" RISC) and Pentium II (not "pure" CISC) (table 13.7)

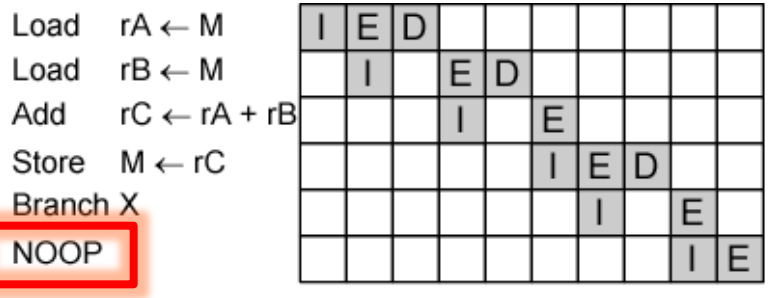
13.5 RISC Pipelining

- Most instructions are *register to register*
- Two phases of execution
  - I: Instruction fetch
  - E: Execute
    - ALU operation with register input and output
- For load and store, three phases:
  - I: Instruction fetch
  - E: Execute
    - Calculate memory address
  - D: Memory
    - Register to memory or memory to register operation

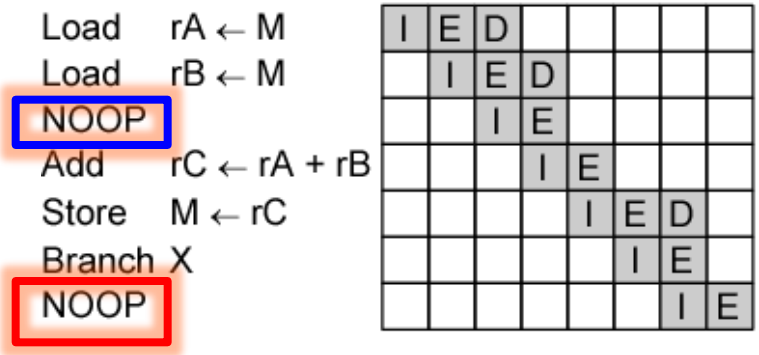
Effects of Pipelining



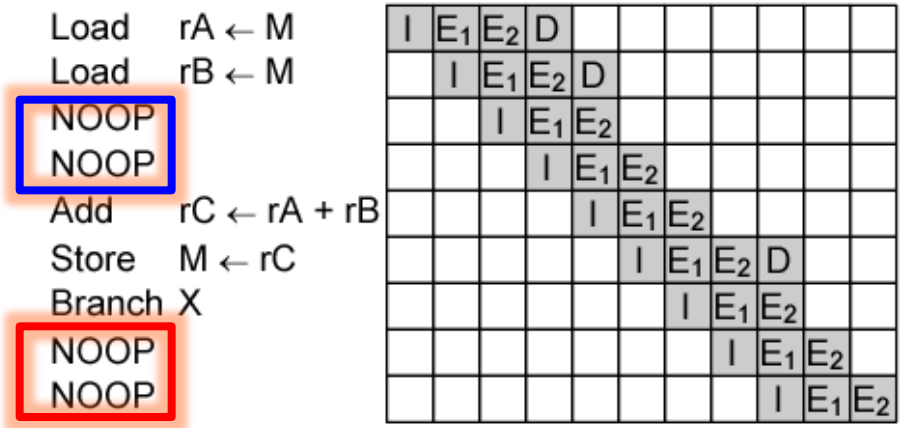
(a) Sequential execution *(No pipeline)*



(b) Two-stage pipelined timing



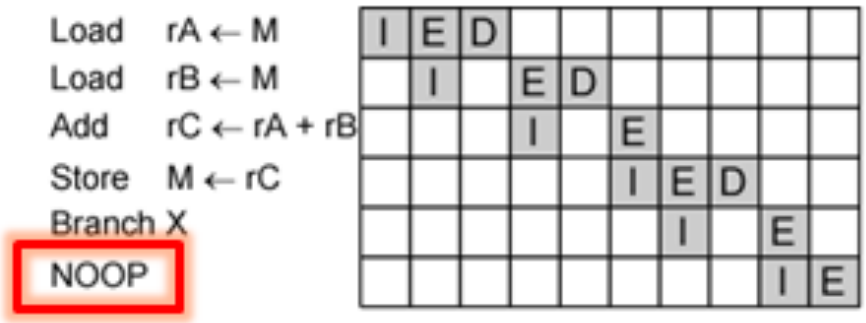
(c) Three-stage pipelined timing



(d) Four-stage pipelined timing

Optimization of Pipelining

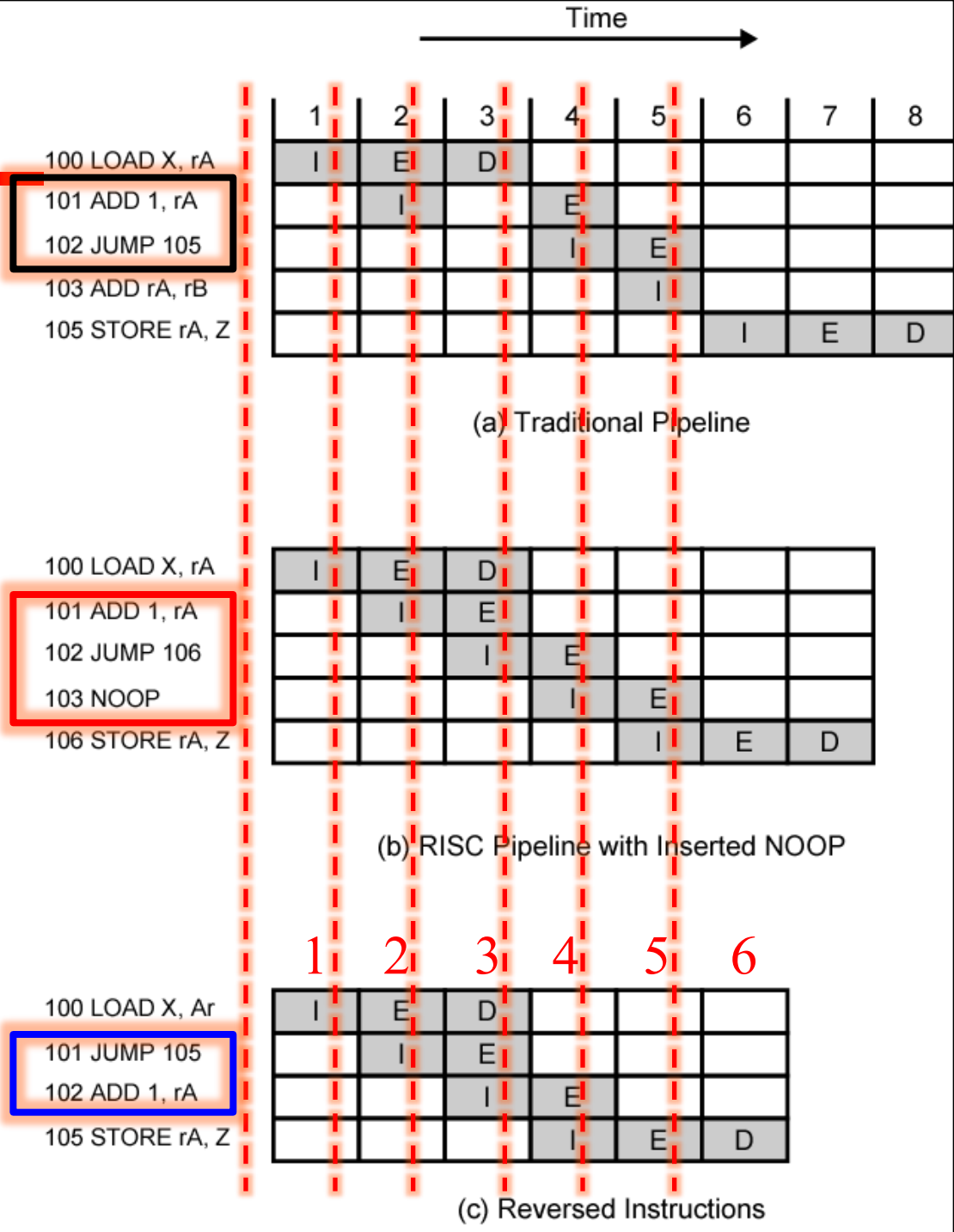
- Delayed branch
  - —Does not take effect until after execution of following instruction (*after NOOP*)
  - —This following instruction is the delay slot



Normal and Delayed Branch

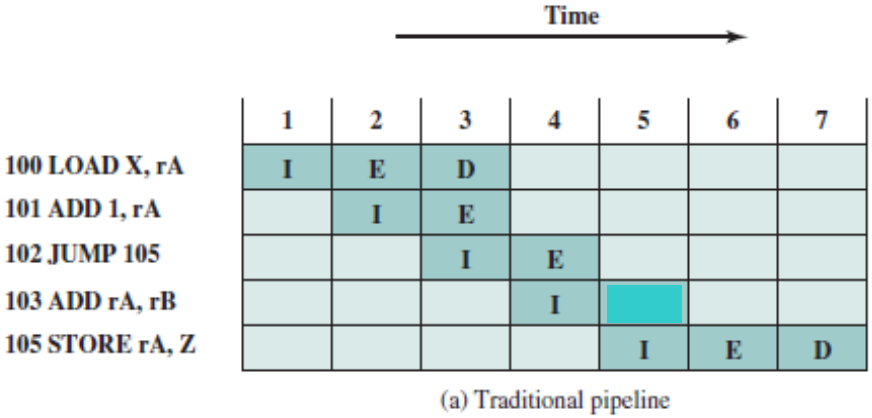
Address	Normal Branch	Branch with NOOP	Reversed Branch
100	LOAD X, rA	LOAD X, rA	LOAD X, rA
101	ADD 1, rA	ADD 1, rA	JUMP 105
102	JUMP 105	JUMP 106	ADD 1, rA
103	ADD rA, rB	NOOP	ADD rA, rB
104	SUB rC, rB	ADD rA, rB	SUB rC, rB
105	STORE rA, Z	SUB rC, rB	STORE rA, Z
106		STORE rA, Z	

Use of Delayed Branch

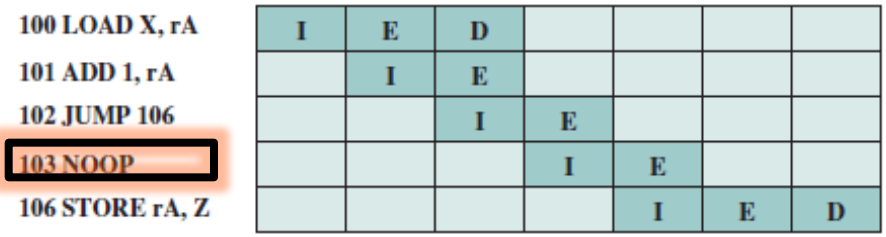


Use of Delayed Branch

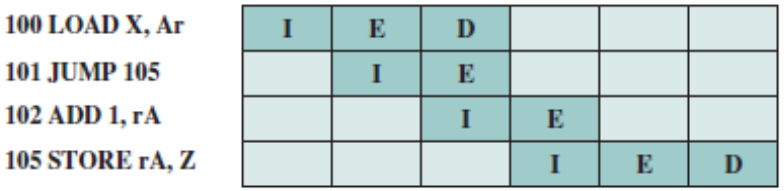
Without considering Data dependency



(a) Traditional pipeline

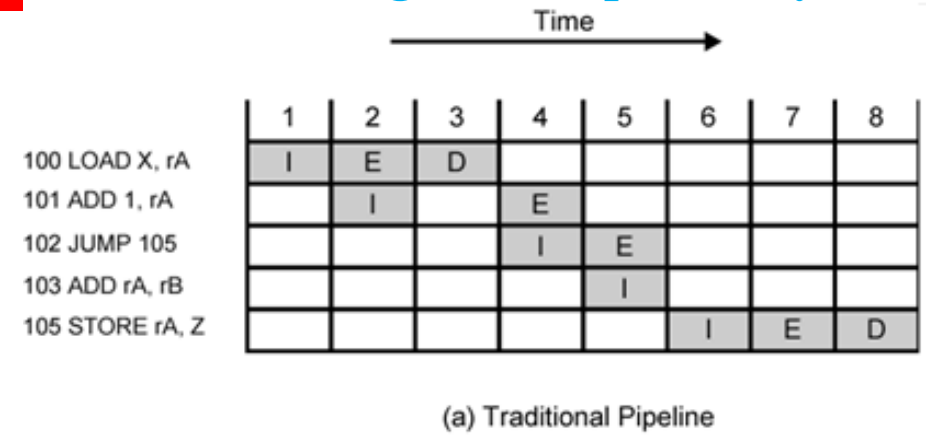


(b) RISC pipeline with inserted NOOP

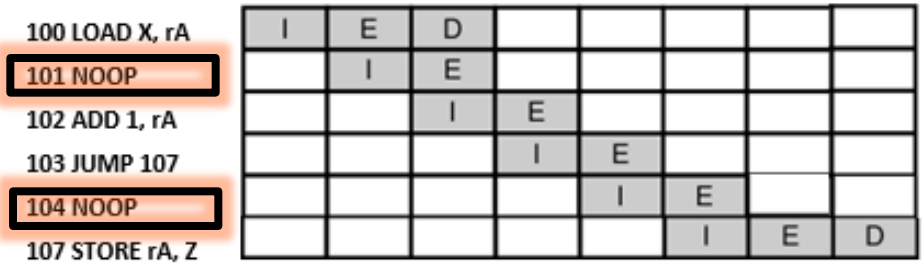


(c) Reversed instructions

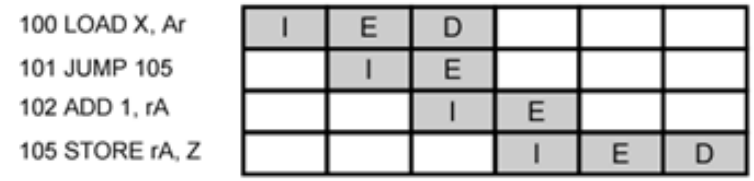
Considering Data dependency



(a) Traditional Pipeline



(b) RISC Pipeline with Inserted NOOP



(c) Reversed Instructions

RISC versus CISC Controversy

- Quantitative
  - compare program sizes and execution speeds
- Qualitative
  - examine issues of high level language support and use of VLSI real estate
- Problems
  - No pair of RISC and CISC that are directly comparable
  - No definitive set of test programs
  - Difficult to separate hardware effects from complier effects
  - Most comparisons done on “toy” rather than production machines
  - Most commercial devices are a mixture

CH13 Key points

- Studies of the execution behavior of ***high-level language programs*** provide guidance in designing RISC:
  - Assignment predominate->simple ***movement of data*** should be optimized
  - Many IF and LOOP instructions->***sequence control*** mechanism needs to be optimized for pipelining
  - Operand reference patterns->keeping a moderate number of operands in ***registers***
- Key characteristics of RISC
  - a ***limited instruction set*** with a ***fixed*** format (*RI*)
  - A ***large number of registers***, optimization of register usage (*Fast*)
  - Optimized ***pipeline*** (*Efficient*)
- RISC is efficient for pipeline (*Why?* ***delay branch*** technique is fit for RISC)

Homework

- Reading chapter 13
- Review Questions
  - 4,5
- Problems:
  - 2, 5, 6, 10

Appendix

- **The Four Color Problem** dates back to **1852** when **Francis Guthrie**, while trying to color the map of counties of England noticed that four colors sufficed. He asked his brother **Frederick** if it was true that **any** map can be colored using four colors in such a way that adjacent regions.
- The first printed reference is due to **Cayley** in 1878.
- **In 1890** a weaker assertion was proved, namely that every planar map can be coloured with five colours.
- The numerous attempts to solve the four-colour problem have influenced the development of certain branches of graph theory. **In 1976** an affirmative answer to the four-colour problem, with the use of a computer, was announced.