Reduced Instruction Set computers

Computer Organization and Architecture

SOUTH CHINA UNIVERSITY OF TECHNOLOGY
FALL 2016

DR. MAO Aihua
ahmao@scut.edu.cn

Background: Major Advances

- The Family Concept
  - Separate architecture from
  - IBM System/360 1964
  - DEC PDP-8
- Microprogrammed Control Unit (cf CH 17)
  - Ease the design for control unit
  - Suggested by Wilkes 1951
  - Produced by IBM S/360 1964
- Cache Memory
  - IBM S/360 model 85 1969

2

Major Advances in Computers(2)

- **Pipelining**
  - Introduces parallelism into fetch execute cycle
- **Multiple processors**
- **Reduced Instruction Set Computer (RISC) architecture**
  - Dramatic departure from the historical processor architecture

3

Driving force for CISC

- **CISC-Complex Instruction Set Computer**
- **Why CISC?**
  - *Software costs far exceed hardware costs*
  - *Increasingly complex high level languages (HLL): structured programming or object-oriented design, allow algorithms more concisely*
  - *Semantic gap: Difference between operations provided in HLLs , lead to complier complexity*

4

Driving force for CISC

- To close the gap, Leads to:
  - *Large instruction sets*
  - *More addressing modes*
  - *Hardware implementations of HLL statements*
    - *e.g. CASE (switch) on VAX*

5

Intention of CISC

- Ease compiler writer
- Improve execution efficiency, because complex operations can be implemented in microcode
- Support more complex HLLs

A different approach of the studies results is to make the architecture that supports the HLL simpler rather more complex- RISC

6

Why CISC ?

- **Smaller programs?**
  - Program takes up less memory

    But memory is now cheap
  - Fewer instructions to be fetched, reducing page faults.

    Occupy more bits in symbolic machine language
    - More instructions in CISC require longer op-codes
    - RISC tend to emphasize register, and register references require fewer bits

7

Why CISC ?

- **Faster programs?**
  - 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**

8

The Next Step - RISC

- RISC-Reduced Instruction Set Computer
- 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*

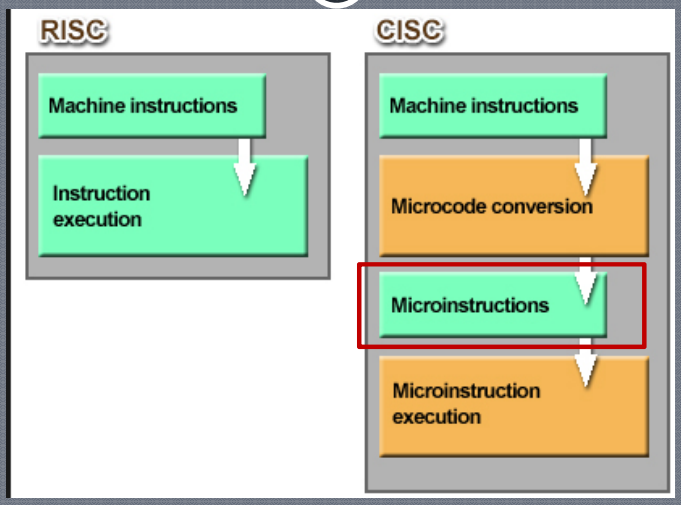
9

Key Points

- The simple instruction set of a RISC allows efficient pipelining because there are fewer & more predictable operations.
- A RISC Instruction Set architecture leads itself to the delayed branch technique (instruction are rearranged to occur later so to improve pipeline efficiency).

10

CISC VS. RISC



11

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

12

Execution Characteristics of RISC

- The development of RISC was based on the study of *instruction execution characteristics*
  - Operations performed
    - determine functions to be performed and interactions with memory
  - Operands used (types and frequencies)
    - determine memory organization and addressing modes
  - Execution sequencing (Procedure calls)
    - determines the control and pipeline organization

13

Execution Characteristics

- The following table summarizes the results of a number of studies of high-level-language programs. All of the results are based on dynamic measurements.
- Vocabulary
  - Dynamic studies are measured during the execution of the program.
  - Static measurements merely perform these counts on the source text of a program.

14

Relative Dynamic Frequency

Relative frequency of occurrence of various HLL instructions in different programs

	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%

15

Operations

- The results from the table suggest that:
  - Assignment statements predominate, movement of data is of high importance
  - Preponderance of Conditional statements (IF, LOOP)
- The results are representative for the CISC

16

Operands

- The dynamic frequency of occurrence of classes of variables
- Majority are local scalar (non-composite) variables
- Optimization is the mechanism for storing and accessing local variables

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

17

Procedure Calls

- Is the most time consuming, it is profitable to improve these operations efficiently.
- Two aspects are significant:
  - *The number of parameters and variables* that a procedure deals with
  - The level of nesting
- Most procedures were passed fewer than 6 arguments and used fewer than 6 local scalar variables
- A high proportion of operand references is to local scalar variables

18

Implications

- Making instruction set architecture close to all HLL is not most effective design strategy (*only a few used most*)
- Best support is to optimize the most used and most time consuming features of typical HLL programs
- From the work of a number of researchers, three elements characterize RISC architectures.

19

Implications

1. Use a large number of registers
   - *Operand referencing optimization coupled with locality of references ⇒ memory references reduced*
2. Careful design of instruction pipelines
   - *Conditional branch and procedure call*
3. Simplified (reduced) instruction set is indicated

20

Use of a Large Register File

- From the analysis
  - *Large number of assignment statements in HLL program, A←B*
  - *Most accesses to local scalars*
- Heavy reliance on register storage is suggested
  - *Minimizing memory access*
  - *allows the most frequently accessed operands to be kept in registers and minimizes the register-memory operations*

21

Registers for Local Variables

- Store local scalar variables *in registers*
  - Reduces memory access
- Some problems
  - Every procedure call changes locality
  - On every call, local variables must be saved to memory so the register can be reused
  - On return, the local variables from the calling program must be loaded back to the register, so the variables from calling programs are restored.

22

RISC Characteristics

- One instruction per machine cycle
- Register to register operations
- Simple addressing modes
- Simple instruction formats

23

One Instruction Per Machine Cycle

- A machine cycle , defined to be the time taken to
  - fetch two operands from registers
  - perform an ALU operation
  - store the result in a register
- With one-cycle instructions, there is little or no need for microinsruction
- Machine instructions can be hardwired

24

Register-to-Register Operations

- Most operations is register-to-register
- Only LOAD and STORE accessing memory
- Simplify instruction set and control unit
  - RISC include only 1 or 2 ADD instructions
  - While VAX has 25 different ADD instructions
- Encourages the optimization of register use
  - Frequently accessed operands remain in high-speed storage

25

Simple Addressing Modes

- Almost all RISC instructions use simple register addressing
- May include several additional modes
  - Displacement and PC-relative
- Simplify instruction set and control unit

26

Simple Instruction Formats

- Instruction length is fixed and aligned on word boundaries
- Field locations, especially the opcode, are fixed
  - Opcode decoding and register operand accessing can occur simultaneously
  - Simplify control unit

27

Typical of a RISC

- A single instruction size (typically 4 bytes)
- A small number of data addressing modes (typically less than five)
- No indirect addressing
- No operations that combine load/store with arithmetic
  - Add from memory
- No more than one memory-addressed operand per instruction

28

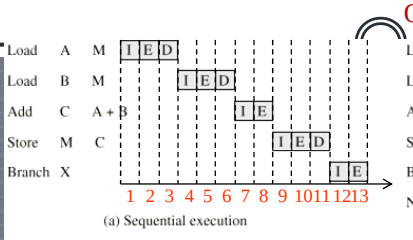
RISC Pipelining

- RISC: 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 stages
  - I: Instruction fetch
  - E: Execute (Calculate memory address)
  - D: Memory (Register to memory or memory to register operation)

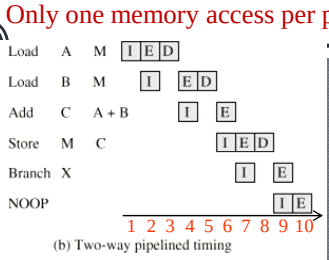
29

Effects of Pipelining

Only one memory access per phase

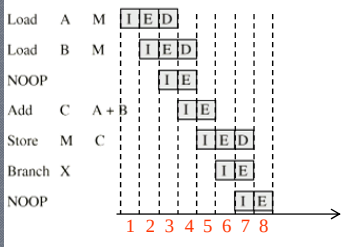


(a) Sequential execution

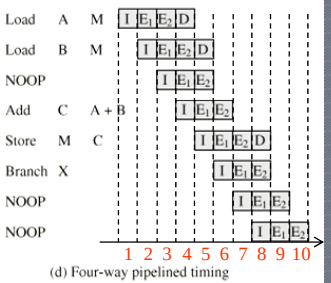


(b) Two-way pipelined timing

Permitting two memory access per phase



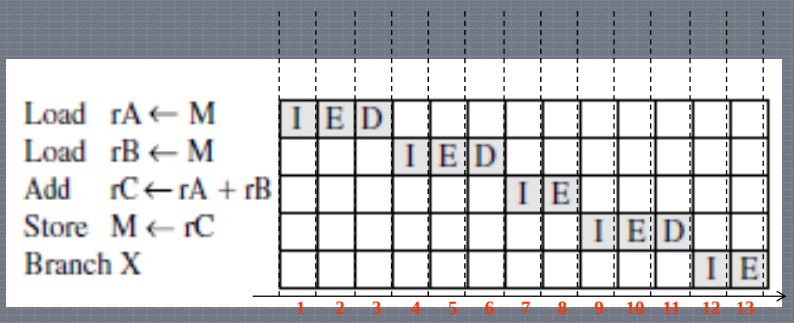
(c) Three-way pipelined timing



(d) Four-way pipelined timing

30

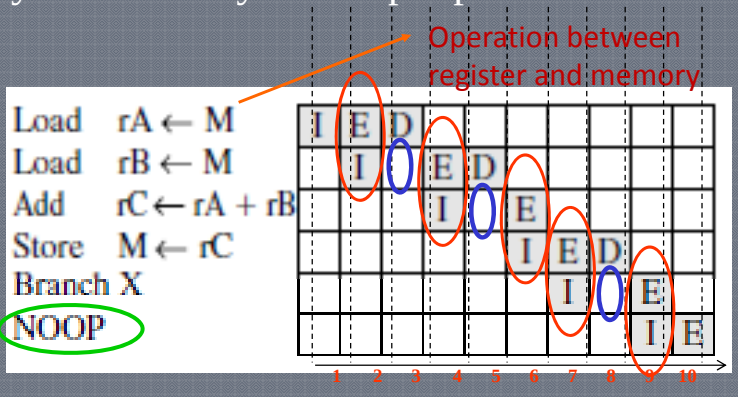
Effects of Pipelining



31

Two-stage Pipelining

- Only one memory access per phase



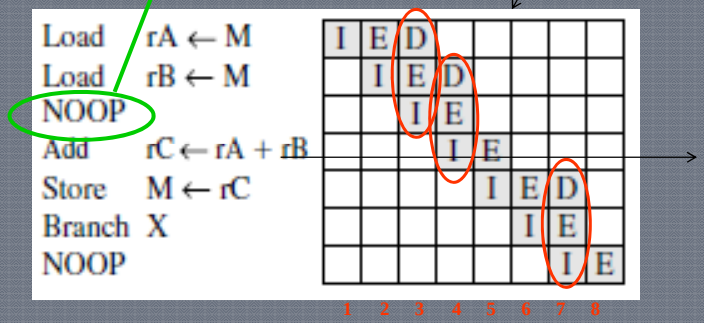
E and I can be performed simultaneously

32

Three-stage Pipelining

- Permitting two memory access per phase

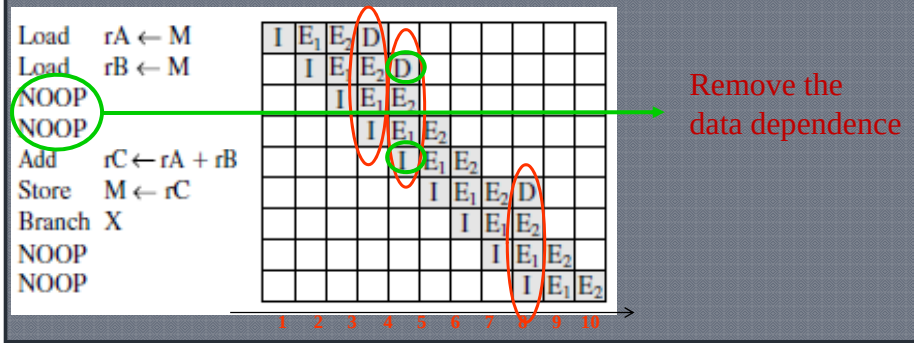
Add is dependent from the previous load. A delay is needed thus a NOOP is inserted



33

Four-stage Pipelining

E can further divided into two substages of equal duration:
E1: Register file read
E2: ALU operation and register write



Remove the data dependence

34

Optimization of Pipelining

- Delayed branch
  - Makes use of a branch that does not take effect until after execution of the following instruction.
  - The instruction location immediately following the branch is referred to as the delay slot.

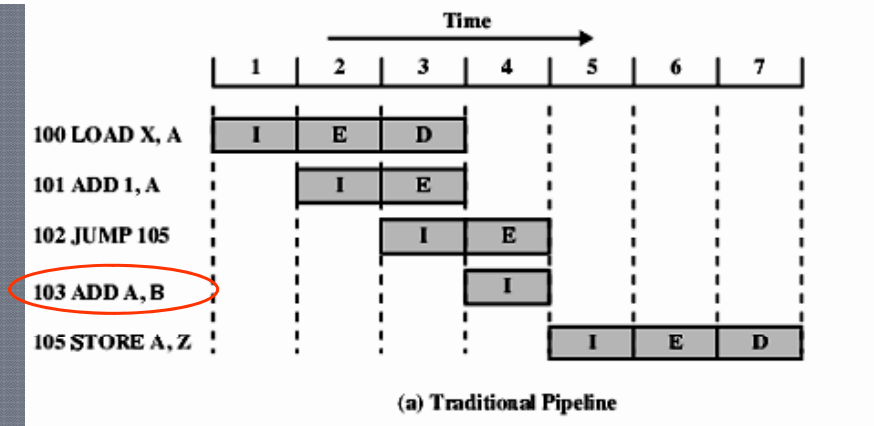
35

Normal and Delayed Branch

Address	Normal	Delayed	Optimized
100	LOAD X,A	LOAD X,A	LOAD X,A
101	ADD 1,A	ADD 1,A	JUMP 105
102	JUMP 105	JUMP 106	ADD 1,A
103	ADD A,B	NOOP	ADD A,B
104	SUB C,B	ADD A,B	SUB C,B
105	STORE A,Z	SUB C,B	STORE A,Z
106		STORE A,Z	

36

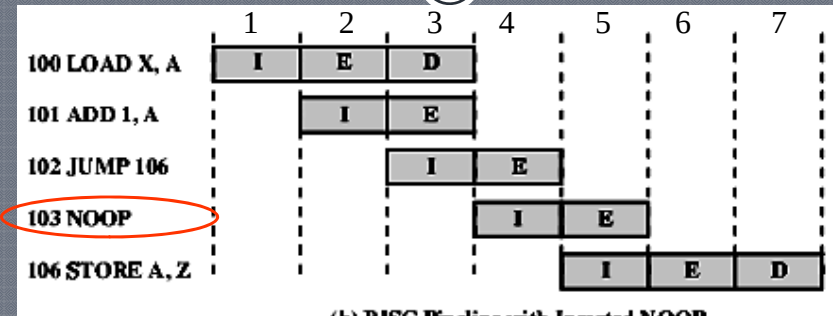
Use of Delayed Branch



(a) Traditional Pipeline

37

Use of Delayed Branch

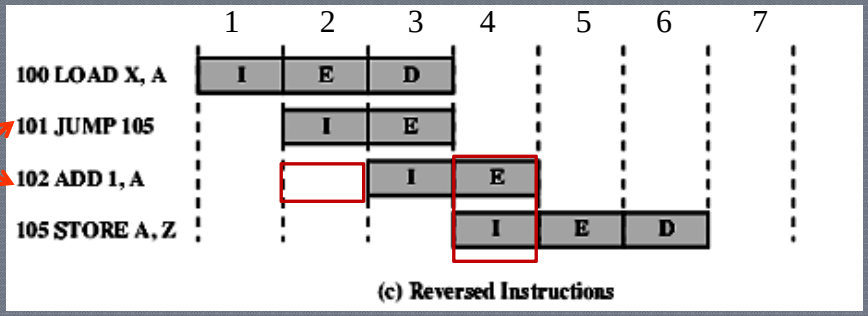


(b) RISC Pipeline with Inserted NOOP

To regularize the pipeline, insert NOOP, do not need special circuitry to clear the pipeline

38

Use of Delayed Branch



The ADD instruction is fetched before the execution of JUMP, thus, retain the semantics of the program

39

Controversy

- Problems
  - No pair of RISC and CISC that are directly comparable
  - No definitive 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

40

Homework

- Review Questions: 13.2
- Hand in 29th Nov.