Advanced Computer Architecture

Instructor

Dr. Dinh-Duc Anh-Vu

http://www.cse.hcmut.edu.vn/~anhvu

Chapter 4

REDUCED INSTRUCTION SET COMPUTERS

Lecture 4 – RISC

- Introduction
- Program execution features
- RISC characteristics
- RISC vs. CISC
- RISC Pipelining

Major Advances in Computers (1)

- The family concept
  - IBM System/360 in 1964
  - DEC PDP-8
  - Separates architecture from implementation
- Microprogrammed control unit
  - Idea by Wilkes in 1951
  - Produced by IBM S/360 in 1964
  - Microprogramming eases the task of designing and implementing the control unit and provides support for the family concept
- Cache memory
  - IBM S/360 model 85 in 1968
  - Improves performance

Major Advances in Computers (2)

- Solid State RAM
  - (See memory notes – lecture 2)
- Microprocessors
  - Intel 4004 in 1971
- Pipelining
  - Introduces parallelism into the essentially sequential nature of a machine-instruction program
- Multiple processors

The Next Step – RISC

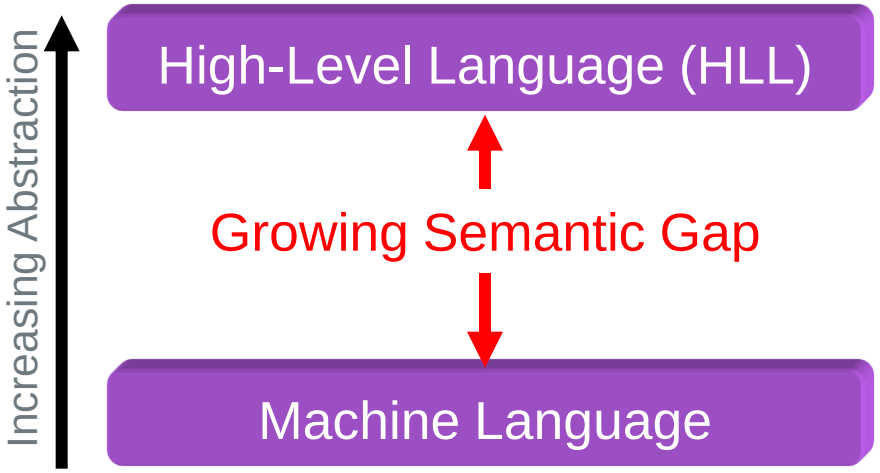
- **Reduced Instruction Set Computer** (RISC) represents an important innovation in computer architecture.
- It is an attempt to produce more CPU power by simplifying the instruction set of the CPU.
- Key features
  - Large number of general purpose registers or use of compiler technology to optimize register usage
  - Limited and simple instruction set
  - Emphasis on optimising the instruction pipeline

CISC

- The opposed trend to RISC is that of **Complex Instruction Set Computer** (CISC).
- Both RISC and CISC architectures have been developed as an attempt to cover the **semantic gap**, and consequently to reduce the ever growing software costs.

Semantic Gap

- In order to improve efficiency of software development, new and powerful high-level programming languages have been developed (e.g., Ada, C++, Java), which provide high level of abstraction.
- This evolution has increased the semantic gap between programming languages and machine languages.



CISC/RISC Examples

Characteristics	CISC Machines			RISC Machines	
Processor	IBM370	VAX	i486	SPARC	MIPS
Year developed	1973	1978	1989	1987	1991
No. of instructions	208	303	235	69	94
Inst. size (bytes)	2-6	2-57	1-12	4	4
Addressing modes	4	22	11	1	1
No. of G.P. registers	16	16	8	40-520	32
Cont. M. size (Kbits)	420	480	246	-	-

Lecture 4 – RISC

- Introduction
- Program execution features
- RISC characteristics
- RISC vs. CISC
- RISC Pipelining

Evaluation of Program Execution

- What are programs doing most of the time?
- We need to determine the execution characteristics of machine instruction sequences generated from HLL programs.
- Aspects of interest:
  - The frequency of operations performed.
  - The types of operands and their frequency of use.
  - Execution sequencing
  - Control flow frequency:
    - branches,
    - loops,
    - subprogram calls.

Evaluation Results

- Frequency of machine instructions executed:
  - See tables in the next slide
- Addressing modes:
  - The majority of instructions uses simple addressing modes.
  - Complex addressing modes (memory indirect, indexed+indirect, etc.) are only used by ~18% of the instructions.

Relative Dynamic Frequency of HLL Operations

Study	[HUCK83]	[KNUT71]	[PATT82a]		[TANE78]
Language	Pascal	FORTRAN	Pascal	C	SAL
Workload	Scientific	Student	System	System	System
Assign	74	67	45	38	42
Loop	4	3	5	3	4
Call	1	3	15	12	12
IF	20	11	29	43	36
GOTO	2	9	—	3	—
Other	—	7	6	1	6

	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

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

- Operand types
  - 74-80% of the operands are scalars (integers, reals, characters, etc.).
  - the rest (20-26%) are arrays/structures; 90% of them are *global variables*.
  - about 80% of the scalars are *local variables*.
- The majority of operands are local variables of scalar type, which can be stored in registers.
- Optimisation should concentrate on storing and accessing local scalar variables

Procedure Calls

- Studies has also been done on the percentage of the total execution time spent executing different high-level language (HLL) statements.
  - Most of the time is spent executing CALLs and RETURNs in HLL programs.
  - Even if only 15% of the executed HLL statements is a CALL or RETURN, they are executed most of the time, because of their complexity.
  - A CALL or RETURN is compiled into a relatively long sequence of machine instructions with a lot of memory references.
- Consuming time
  - Depends on number of parameters passed
    - Most procedure calls (98%) were passed fewer than 6 arguments
    - Most variables (92%) are local scalar
  - Depends on level of nesting
    - Most programs do not do a lot of calls followed by lots of returns

Implications

- An overwhelming dominance of simple (ALU and move) operations over complex operations.
- Dominance of simple addressing modes.
- Large frequency of operand accesses; on average each instruction references 1.9 operands.
- Most of the referenced operands are scalars (can be stored in registers) and are local variables or parameters.
- Optimizing the procedure CALL/RETURN mechanism promises large benefits in speed.

Evaluation Conclusions

- The attempt to make the instruction set architecture close to HLLs is not the most effective design strategy
- Best support is given by optimising most used and most time-consuming features

- 3 elements used to characterize RISC architectures
  - Use a large number of registers or use a compiler to optimize register usage
    - Intended to optimize operand referencing
  - Careful design of pipelines
    - Branch prediction etc.
  - Simplified (reduced) instruction set

=> A RISC machine will deliver better performance!

Lecture 4 – RISC Computers

- Introduction
- Program execution features
- RISC characteristics
- RISC vs. CISC
- RISC Pipelining

Main Characteristics of RISC

- A small number of simple instructions (desirably < 100).
  - Simple and small decode and execution hardware are required.
  - A hard-wired controller is needed, rather than using microprogramming.
  - The CPU takes less silicon area to implement, and runs also faster.
- Main characteristics
  - Execution of one machine instruction per clock cycle
  - Register-to-register operations
  - Simple addressing modes
  - Simple instruction formats

One Instruction per Clock Cycle

- Machine cycle is defined to be the time it takes to fetch 2 operands from registers, perform an ALU operation and store the result in a register.
- The instruction pipeline performs more efficiently due to simple instructions and similar execution patterns.
- Complex operations are executed as a sequence of simple instructions.
  - In the case of CISC they are executed as one single or a few complex instruction.

One Instruction per Clock Cycle

An illustrative example with the following assumption:

- A program with 80% of executed instructions being simple and 20% complex.
- CISC: simple instructions take 4 cycles, complex instructions take 8 cycles; cycle time is 100 ns.
- RISC: simple instructions are executed in one cycle; complex operations are implemented as a sequence of instructions (14 instructions on average); cycle time is 75 ns.

How much time takes a program of 1 000 000 instructions?

- CISC: $(10^6 \times 0.80 \times 4 + 10^6 \times 0.20 \times 8) \times 10^{-7} = 0.48$ s
- RISC: $(10^6 \times 0.80 \times 1 + 10^6 \times 0.20 \times 14) \times 0.75 \times 10^{-7} = 0.27$ s

Register-to-Register Operation (1)

- **Load-and-store architecture**
  - Only LOAD and STORE instructions reference data in memory.
  - All other instructions operate only with registers (are register-to-register instructions).
- This characteristic simplifies the instruction set and therefore the control unit.
  - A RISC instruction set may include only 1 or 2 ADD instructions (e.g. integer add, add with carry)
  - VAX has 25 different ADD instructions
- This architecture encourages the optimization of register use, so that frequently accessed operands remain in high-speed storage.

R-to-R and M-to-M Approaches

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

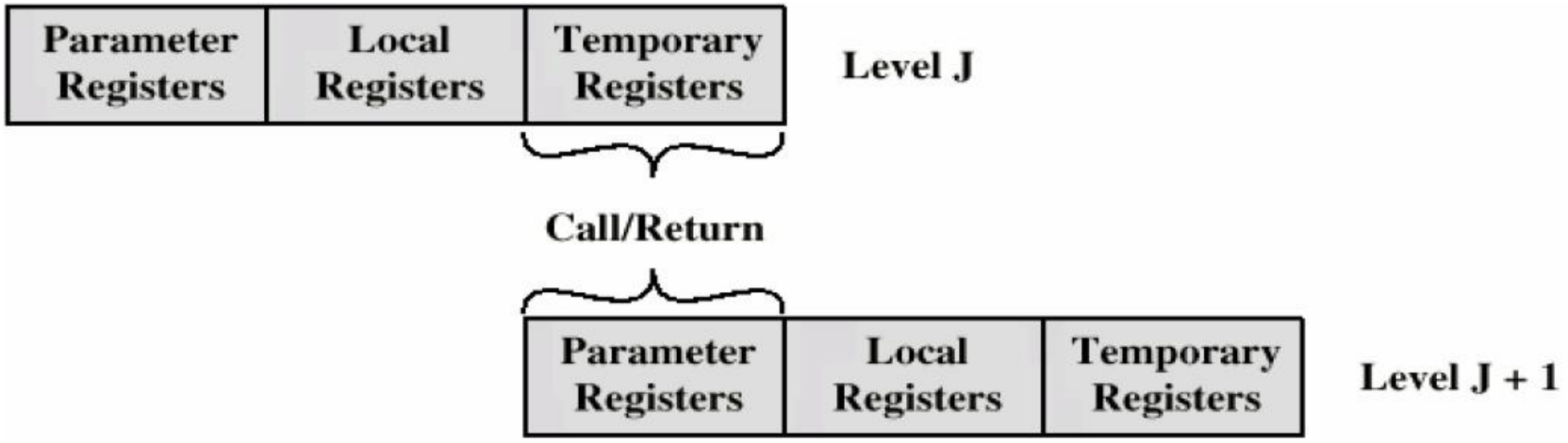
(b) A ← B + C; B ← A + C; D ← D – B

Register-to-Register Operation (2)

- A large number of registers is available.
  - Variables and intermediate results can be stored in registers and do not require repeated loads and stores from/to memory.
  - All local variables of procedures and the passed parameters can be stored in registers.
  - The large number of registers is typical for RISC, because the reduced complexity of the processor means that we have silicon space on the processor chip to implement them. This is usually not the case with CISC machines.

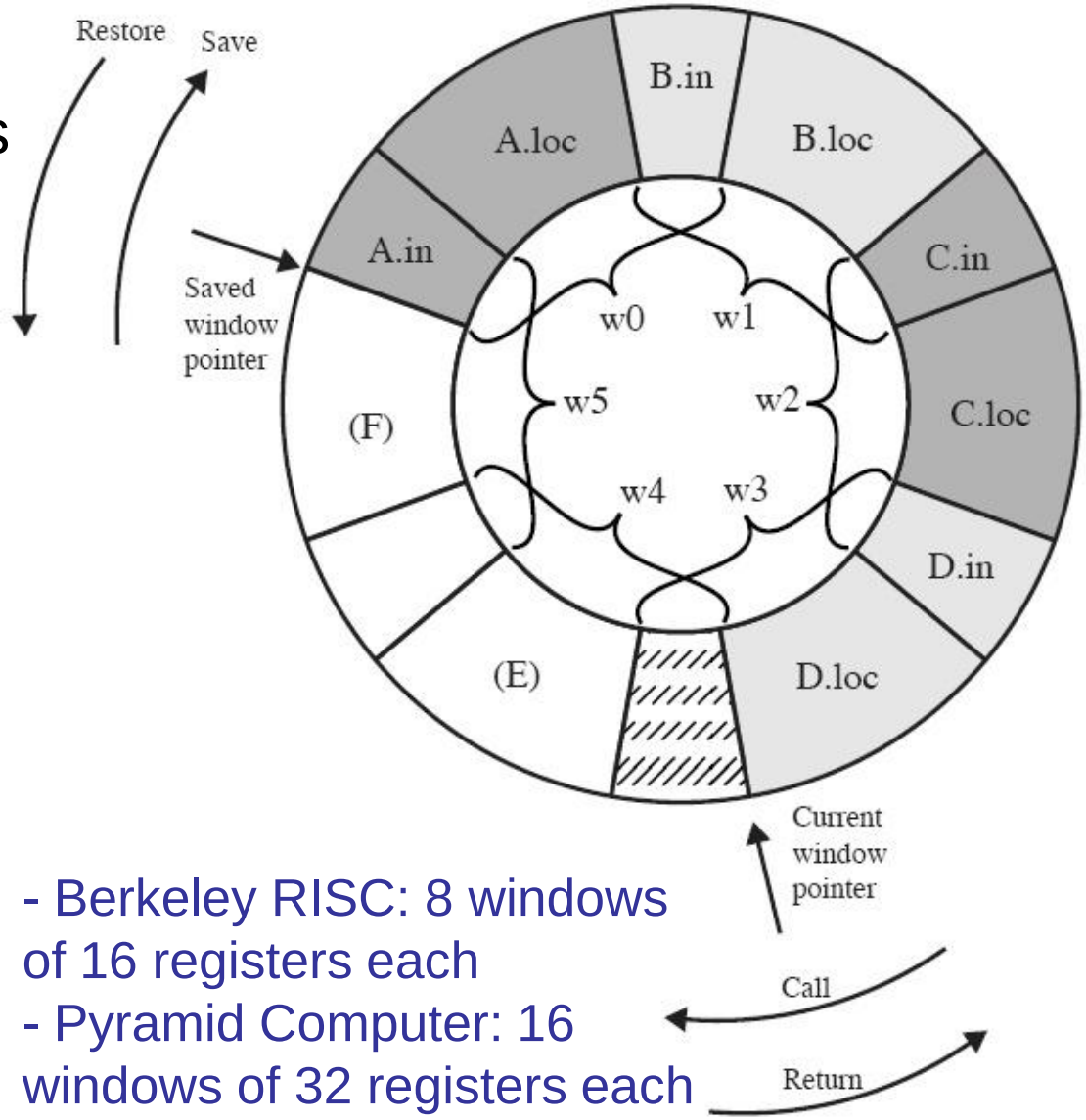
Register Windows

- A large number of registers is usually very useful.
- If contents of all registers must be saved at every procedure call, however, more registers mean longer delay.
- A solution to this problem is to divide the register file into a set of fixed-size windows.
  - Each register window is assigned to a procedure.
  - Windows for adjacent procedures are overlapped to allow parameter passing.



Circular-Buffer Organization of Overlapped Windows

- 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



- Berkeley RISC: 8 windows of 16 registers each
- Pyramid Computer: 16 windows of 32 registers each

Main Characteristics of RISC (Cont'd)

- Only a few simple addressing modes are used.
  - Almost all RISC instructions use simple register addressing
  - Complex modes can be synthesized in software from the simple ones
  - Ex. register, direct, register indirect, displacement.

- Instructions are of fixed length and uniform format.
  - Loading and decoding of instructions are simple and fast; it is not needed to wait until the length of an instruction is known in order to start decoding it;
  - Decoding is simplified because the opcode and address fields are located in the same position for all instructions.

Main Advantages of RISC

- Best support is given by optimizing most used and most time consuming architecture aspects.
  - Frequently executed instructions.
  - Memory reference.
  - Procedure call/return.
  - Pipeline design.
- Less design complexity, reducing design cost, and reducing the time between designing and marketing.

Criticism of RISC

- An operation might need two, three, or more instructions to accomplish.
  - More memory access might be needed.
  - Execution speed may be reduced in certain applications.
- It usually leads to longer programs, which needs larger memory space to store.
- Difficult to program **machine codes** and **assembly programs**.
  - More time consuming.

Lecture 4 – RISC Computers

- Introduction
- Program execution features
- RISC characteristics
- RISC vs. CISC
- RISC Pipelining

Typical RISC

- A single instruction size
- Instruction size is typically 4 bytes
- A small number of data addressing modes, typically less than 5
- No indirect addressing (make one memory access to get the address of another operand in memory)
- No operations that combine load/store with arithmetic (e.g. add to/from memory)
- No more than one memory-addressed operand per instruction
- Does not support arbitrary alignment of data for load/store operations
- Maximum number of uses of the memory management unit (MMU) for a data address in an instruction
- Number of bits for integer register specifier equal to 5 or more (at least 32 integer registers can be explicitly referenced at a time)
- Number of bits for floating-point register specifier equal to 4 or more (at least 16 floating-point registers can be explicitly referenced at a time)

RISC vs. CISC

- A lot of performance comparisons have shown that benchmark programs run often faster on RISC processors than on CISC machines.
- However, it is difficult to identify which RISC feature really produces the higher performance.
- Some "CISC fans" argue that the higher speed is not produced by the typical RISC features but because of technology, better compilers, etc.
- An argument in favor of the CISC: the simpler RISC instruction set results in a larger memory requirement compared with the CISC case.

Most recent processors are not typical RISC or CISC, but combine advantages of both approaches.

  - e.g. PowerPC and Pentium II.

Characteristics of Some Processors

	Processor	Number of instruc-tion sizes	Max instruc-tion 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
RISC	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
CISC	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]

- Column 1 → 3: indication of instruction decode complexity
- Column 4 → 8: indication of the ease or difficulty of pipelining
- Column 9 and 10: ability to take good advantage of compilers

Main features of CISC

- A large number of instructions (> 200) and complex instructions and data types.
- Many and complex addressing modes.
- Direct hardware implementations of high-level language statements.
  - e.g. CASE (switch) on VAX
- Microprogramming techniques are used so that complicated instructions can be implemented.
- Memory bottleneck is a major problem, due to complex addressing modes and multiple memory accesses per instruction.

Arguments for CISC

- A rich instruction set should simplify the compiler by having instructions which match the high-level language instructions.
- Since the programs are smaller in size, they have better performance:
  - They take up less memory space.
  - Fewer number of instructions are executed and need fewer instruction fetch cycles, which may lead to smaller execution time.
  - In a paging environment, smaller programs occupy fewer pages, reducing page faults.
- Program execution efficiency is also improved by implementing complex operations in microcode rather than machine code.

Problems with CISC (1)

- Compiler simplification?
  - Disputed…
  - 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, just look shorter in symbolic form
    - More instructions require longer op-codes
    - Register references require fewer bits

Code Size Relative to RISC I

	11 C Programs [Patterson]	12 C Programs [Katevenis]	5 C Programs [Heath]
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	

Problems with CISC (2)

- 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

Problems with CISC

- A large instruction set requires complex and potentially time consuming hardware steps to decode and execute the instructions.
- Complex machine instructions may not match high-level language statements exactly, in which case they may be of little use.
  - This will be a major problem if the number of languages is getting bigger.
- Instruction sets designed with specialized instructions for several high-level languages will not be efficient when executing program of a given language.
- Complex design tasks.

A CISC Example – Intel i486

- 32-bit integer processor
  - Registers 8 general
    6 address (16-bits)
    2 status/control
    1 instruction-pointer (PC)
  - On-chip floating-point unit
  - Microprogrammed control
- Instruction set:
  - Number of instructions: 235
    - E.g., “FYL2XP1 x y” performs y•log2(x+1) in 313 clock cycles.
  - Instruction size: 1-12 bytes (3.2 on average).
  - Addressing modes: 11

A CISC Example – Intel i486 (Cont'd)

- Memory organization:
  - Address length: 32 bits (4 GB space)
  - Memory is segmented for protection purpose
  - Support virtual memory by paging
  - Cache-memory: 8 KB internal (on-chip) (96% hit rate for DOS applications)
- Instruction execution:
  - Execution models: reg-reg, reg-mem, mem-mem
  - Five-stage instruction pipeline: Fetch, Decode1, Decode2, Execute, Write-Back.

A RISC Example – SPARC

Scalable Processor Architecture:

- 32-bit processor with the following components:
  - An Integer Unit (IU) — to execute all instructions.
  - A Floating-Point Unit (FPU) — a co-processor which can work concurrently with the IU and is used to perform arithmetic operations on floating point numbers.
- has 69 basic instructions.
- has a linear, 32-bit virtual-address space (4G).
- has 40 to 520 general purpose 32-bit registers which are grouped into 2 to 32 overlapping register windows
  (16/group + 8) => Scalability.

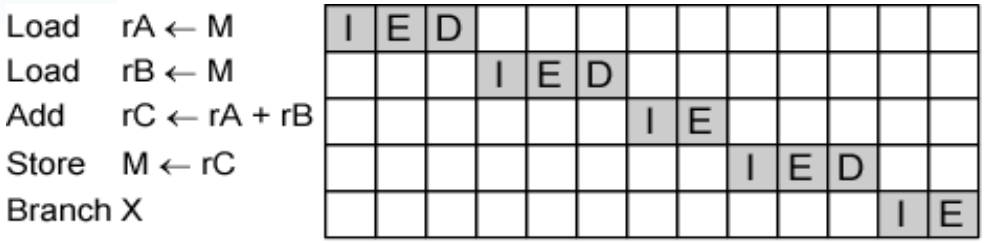
Lecture 4 – RISC Computers

- Introduction
- Program execution features
- RISC characteristics
- RISC vs. CISC
- RISC Pipelining

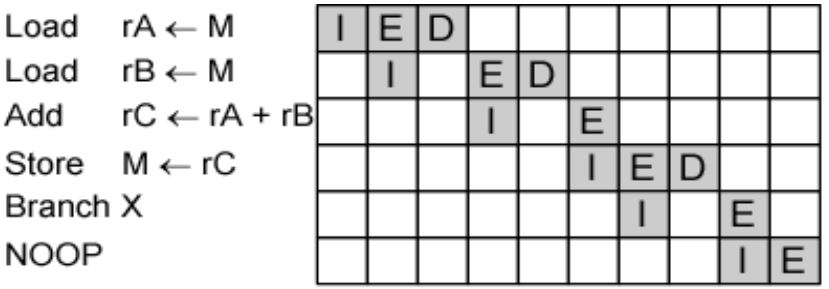
RISC Pipelining

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

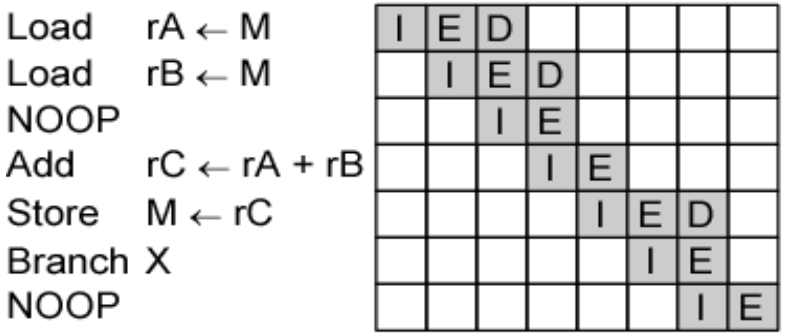
Effects of Pipelining



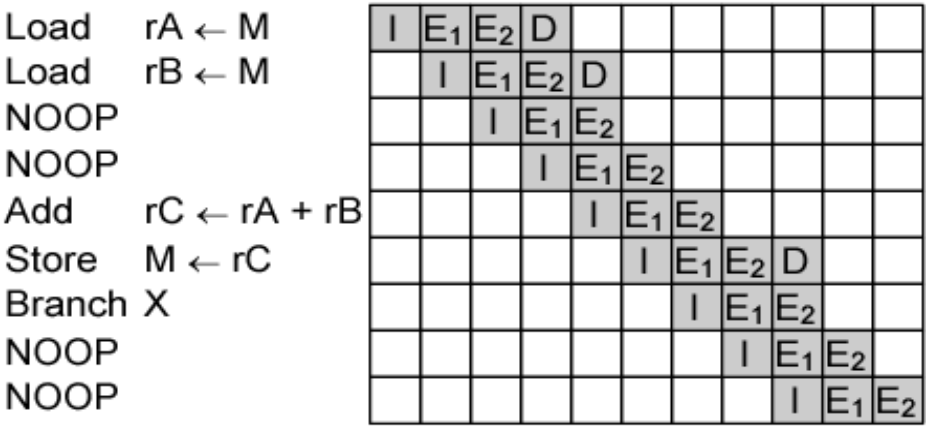
(a) Sequential execution



(b) Two-stage pipelined timing



(c) Three-stage pipelined timing



(d) Four-stage pipelined timing

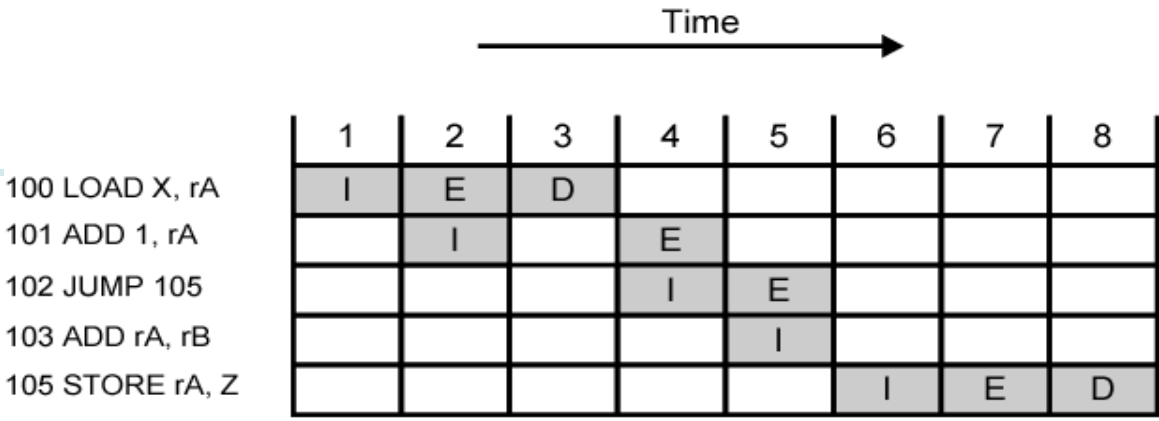
Optimization of Pipelining

- Delayed branch
  - Does not take effect until after execution of following instruction
  - This following instruction is the delay slot

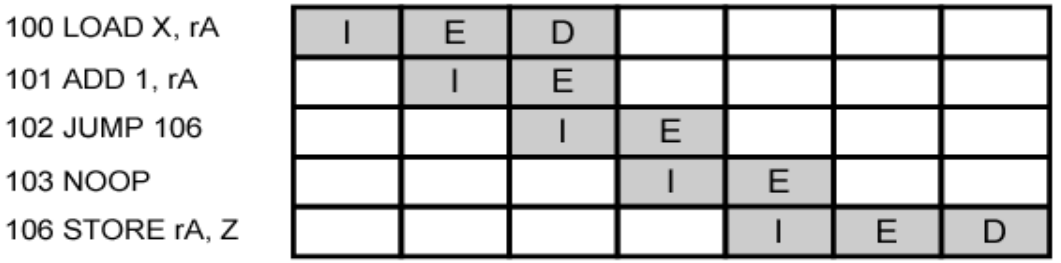
Normal and Delayed Branch

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

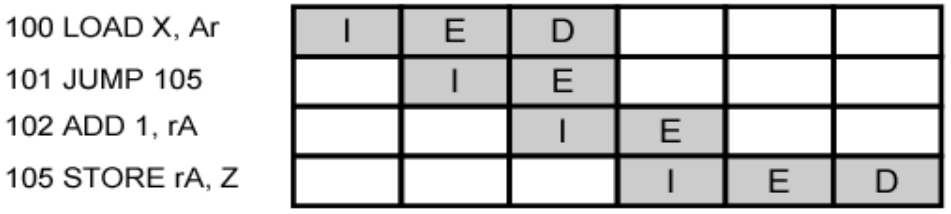
Use of Delayed Branch



(a) Traditional Pipeline



(b) RISC Pipeline with Inserted NOOP



(c) Reversed Instructions

RISC vs. 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

Summary

- Both RISCs and CISCs try to reduce the semantic gap, with different approaches.
  - CISCs follow the traditional way of implementing more and more complex instructions.
  - RISCs try to simplify the instruction set, while improving the instruction execution performance.
- Innovations in RISC architectures are based on a close analysis of typical programs.
- The main features of RISC architectures are: reduced number of simple instructions, few addressing modes, load-store architecture, instructions with fixed length and format, and a large number of registers.
- One of the main concerns of RISC designers was to maximize the efficiency of pipelining.
- Most modern architectures often include both RISC and CISC features.