Computer Organization & Architecture

Chapter 2 – RISC and CISC Styles

Zhang Yang 张杨

cszyang@scut.edu.cn

Autumn 2025

Content of this lecture

- 2.10 CISC Instruction Sets
  - Additional Addressing Modes
  - Condition Codes
- 2.11 RISC and CISC Styles

CISC Instruction Sets (1)

- CISC instruction sets are not constrained to the *load/store architecture*, in which arithmetic and logic operations can be performed only on operands that are in processor registers.
- CISC instructions do not necessarily have to fit into a single word. Some instructions may occupy a single word, but others may span multiple words.
- Most arithmetic and logic instructions use the *two-address* format.
  - Operation *destination*, *source*
  - Example: Add B, A
    - Performs the operation B ← [A] + [B] on memory operands.

CISC Instruction Sets (2)

- The Move instruction includes the functionality of the Load and Store instructions.
  - Move destination, source
  - Example C = A + B
    - Move C, B
    - Add C, A
  - In some CISC processors one operand may be in the memory but the other must be in a register.
    - Move R*i*, A
    - Add R*i*, B
    - Move C, R*i*

Additional Addressing Modes (1)

- Autoincrement and Autodecrement Mode
  - Autoincrement Mode
    - The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.
    - (Ri)+
    - EA = [Ri] Increment Ri
    - Useful for adjusting pointers in loop body:

      Add SUM, (R*i*)+
      MoveByte (R*j*)+, R*k*

      - Increment by 4 for words, and by 1 for bytes

Additional Addressing Modes (2)

- Autoincrement and Autodecrement Mode (ctd.)
  - Autodecrement Mode
    - The contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand.
    - −*(*R*i)*
    - Use autoinc. & autodec. for stack operations:

      Move –(SP), NEWITEM (push)

      Move ITEM, (SP)+ (pop)

Additional Addressing Modes (3)

- Relative Mode
  - The effective address is determined by the Index mode using the program counter in place of the general-purpose register Ri.
  - EA = [PC] + X, X is a signed number.
  - Usage
    - Access data operand.
    - Specify the target address in branch instructions.
      - Example Branch > 0 Loop
        - The branch target location can be computed by specifying it as an offset from the current value of the program counter.

Condition Codes (1)

- Condition Codes
  - Processor can maintain information on results to affect subsequent conditional branches.
  - Results from arithmetic/comparison & Move.
  - Condition code flags in a status register:

N	(negative)	1 if result negative, else 0
Z	(zero)	1 if result zero, else 0
V	(overflow)	1 if overflow occurs, else 0
C	(carry)	1 if carry-out occurs, else 0

Condition Codes (2)

- Branches Using Condition Codes
  - CISC branches check condition code flags.
  - For example, decrementing a register causes N and Z flags to be cleared if result is *not* zero.
  - A branch to check logic condition $N + Z = 0$:

    Branch>0 LOOP
  - Other branches test conditions for $<$, $=$, $\neq$, $\leq$, $\geq$
  - Also Branch_if_overflow and Branch_if_carry.
  - Consider CISC-style list-summing program (next page)

Condition Codes (3)

- Branches Using Condition Codes (ctd.)

```
        Load            R2, N           Load the size of the list.
        Clear           R3              Initialize sum to 0.
        Move            R4, #NUM1       Get address of the first number.
LOOP:   Load            R5, (R4)        Get the next number.
        Add             R3, R3, R5      Add this number to sum.
        Add             R4, R4, #4      Increment the pointer to the list.
        Subtract        R2, R2, #1      Decrement the counter.
        Branch_if_[R2]>0 LOOP           Branch back if not finished.
        Store           R3, SUM         Store the final sum.
```

Figure 2.8 Use of indirect addressing in the program of Figure 2.6.

Condition Codes (4)

- Branches Using Condition Codes (ctd.)

```
        Move        R2, N          Load the size of the list.
        Clear       R3             Initialize sum to 0.
        Move        R4, #NUM1      Load address of the first number.
LOOP:   Add         R3, (R4)+      Add the next number to sum.
        Subtract    R2, #1         Decrement the counter.
        Branch>0    LOOP           Loop back if not finished.
        Move        SUM, R3        Store the final sum.
```

Figure 2.26 A CISC version of the program of Figure 2.8.

RISC and CISC Styles (1)

- RISC characteristics include:
  - Simple addressing modes
  - All instructions fitting in a single word
  - Fewer total instructions
  - Arithmetic/logic operations on registers
  - Load/store architecture for data transfers
  - More instructions executed per program
- Simpler instructions make it easier to design faster hardware (e.g., use of pipelining)

RISC and CISC Styles (2)

- CISC characteristics include:
  - More complex addressing modes
  - Instructions spanning more than one word
  - More instructions for complex tasks
  - Arithmetic/logic operations on memory
  - Memory-to-memory data transfers
  - Fewer instructions executed per program
- Complexity makes it somewhat more difficult to design fast hardware, but still possible.

RISC and CISC Styles (3)

- Before the 1970s, all computers were of CISC type.
  - Move the complexity from the software level to the hardware level.
- RISC-style designs emerged as an attempt to achieve very high performance by making the hardware very simple.
- CISC and RISC implementations are becoming more and more alike. Many of today's RISC chips support as many instructions as yesterday's CISC chips. And today's CISC chips use many techniques formerly associated with RISC chips.

RISC and CISC Styles (4)

- CISC vs RISC

CISC	RISC
Complex instructions require multiple cycles	Reduced instructions take 1 cycle
Many instructions can reference memory	Only Load and Store instructions can reference memory
Instructions are executed one at a time	Uses pipelining to execute instructions
Few general registers	Many general registers

Instruction Set Architecture

- Job of a CPU: execute instructions
- Instructions: CPU's primitive operations
- CPUs belong to "families", each implementing its own set of instructions.
- CPU's particular set of instructions implements an Instruction Set Architecture (ISA).
  - X-86
  - ARM
  - RISC-V
  - MIPS
  - …

x86 Architecture

- The Intel x86 architecture
  - Totally dominates the server/laptop market.
  - Serves as an excellent example of CISC design.
  - Has evolved to remain backward compatible with earlier versions.
    - Backwards compatible up until 8086, introduced in 1978.
    - Added more features as time goes on.
  - Intel attempted radical shift from IA32 to IA64
    - Totally different architecture (Itanium)
    - Performance disappointing
  - AMD Stepped in with Evolutionary Solution
    - x86-64 (now called “AMD64”)

ARM Architecture (1)

- ARM is a family of RISC-based microprocessors and microcontrollers designed by ARM Inc., Cambridge, England.
- The company doesn't make processors but instead designs microprocessors and multicore architectures and licenses them to manufacturers.
- ARM History
  - ARM – Acorn RISC Machine (1983 – 1985)
    - Acorn Computers Limited, Cambridge, England
  - ARM – Advanced RISC Machine 1990
    - ARM Limited, 1990
    - ARM has been licensed to many semiconductor manufacturers.

ARM Architecture (2)

- ARM Evolution

Family	Notable Features	Cache	Typical MIPS @ MHz
ARM1	32-bit RISC	None	
ARM2	Multiply and swap instructions; Integrated memory management unit, graphics and I/O processor	None	7 MIPS @ 12 MHz
ARM3	First use of processor cache	4 KB unified	12 MIPS @ 25 MHz
ARM6	First to support 32-bit addresses; floating-point unit	4 KB unified	28 MIPS @ 33 MHz
ARM7	Integrated SoC	8 KB unified	60 MIPS @ 60 MHz
ARM8	5-stage pipeline; static branch prediction	8 KB unified	84 MIPS @ 72 MHz
ARM9		16 KB/16 KB	300 MIPS @ 300 MHz
ARM9E	Enhanced DSP instructions	16 KB/16 KB	220 MIPS @ 200 MHz
ARM10E	6-stage pipeline	32 KB/32 KB	
ARM11	9-stage pipeline	Variable	740 MIPS @ 665 MHz
Cortex	13-stage superscalar pipeline	Variable	2000 MIPS @ 1 GHz
XScale	Applications processor; 7-stage pipeline	32 KB/32 KB L1 512 KB L2	1000 MIPS @ 1.25 GHz

RISC-V Architecture

- An open-source instruction set architecture (ISA).
- “RISC-V” stands for Reduced Instruction Set Computing (RISC) and the “V” represents the fifth version of the RISC architecture.
- RISC-V is an open standard, allowing anyone to implement it without the need for licensing fees.
- RISC-V follows the principles of RISC, emphasizing simplicity and efficiency in instruction execution.
- The ISA is maintained by the RISC-V Foundation.
  - RISC-V Foundation was founded in 2015 and comprises more than 200 members from various sectors of the industry and academia.
  - RISC-V Foundation owns, maintains, and publishes the RISC-V Instruction Set Architecture (ISA), an open standard for processor design.

What do we mean by architecture?

- When we use the term architecture, we mean a functional specification.
- An architecture specifies how a processor will behave, such as what instructions it has and what the instructions do.
- You can think of an architecture as a contract between the hardware and the software.
- The architecture describes what functionality the software can rely on the hardware to provide.
- The architecture specifies:

Instruction set	• The function of each instruction • How that instruction is represented in memory (its encoding).
Register set	• How many registers there are. • The size of the registers. • The function of the registers. • Their initial state.
Exception model	• The different levels of privilege. • The types of exceptions. • What happens on taking or returning from an exception.
Memory model	• How memory accesses are ordered. • How the caches behave, when and how software must perform explicit maintenance.
Debug, trace, and profiling	• How breakpoints are set and triggered. • What information can be captured by trace tools and in what format.

Summary

- CISC Addressing Modes
  - 理解Autoincrement Mode，掌握有效地址的计算。
  - 理解Autodecrement Mode，掌握有效地址的计算。
  - 理解Relative Mode，掌握有效地址的计算。
  - 理解condition code: N,Z, C, V
- 掌握RISC Characteristics
- 掌握CISC Characteristics