RISC versus CISC

Complex instruction set computers evolved over time

The desire was to close the “semantic gap”

- the difference in the way operations are specified in an expressive high-level language such as Java or C++ and the their hardware implementation

more sophisticated instructions were included in the instruction set such as:

- string search, string compare, string translation
- automating looping

RISC versus CISC

An instruction that just adds to integers to produce a sum is considered a simple instruction

One that copies an element from one array to another and automatically updates both array subscripts is considered a complex instruction

CISC systems have intricate addressing modes

- to ease processing high level data structures

A goal was to simplify high level instruction translation

- by making the CISC machine instructions more closely resemble the high level functions

RISC versus CISC

Microprogramming provides these more complex features

- microinstructions carry out the many intricate steps
- easier to implement than hardwired logic

A machine instruction is interpreted by a microprogram

- may be as many as a dozen or more microinstructions
- they direct hardware in performing the required actions

Microprograms are stored in an internal control memory

- the same chip as the CPU
- take additional time to process compared to hardwired logic

RISC versus CISC

The use of these complex features reached a peak during the 1970s and early 1980s.

Processors of that era following this design approach included:

- the Intel 8086, 80286, 80386 and 80486
- Motorola 68K series
- Digital Equipment (DEC) VAX processors

Memory was very expensive and relatively small in capacity
Using complex instructions made programs smaller

RISC versus CISC

CISC type instructions may use multiple memory operands

    each operand may be referenced using a different addressing mode

The instructions support a large and flexible set of operations

They can vary in size from one byte to 15 or more bytes

They must be partly decoded to know how many additional bytes makeup the instruction

This prevents pre-fetching instructions to speed processing

Example CISC Instructions

x86 CISC type instructions include:

scas (string scan) - scans a block of memory looking for a match to the contents of a register and automatically updates counter and pointers.

xlat (translate) – performs a table lookup to convert the contents of a register to a number stored in a memory table.

loop – decrements and tests a counter and jumps based on the outcome

Example CISC Instructions

VAX CISC type instructions include:

PUSHR   #^M<R0, R1, R2, R3, R4, R5>
- which saves a series of registers on the stack based on a specified bit mask.

POPR   #^M<R0, R1, R2, R3, R4, R5>
- which restores a series of registers from the stack based on a specified bit mask.

These are functions that would be performed as part of a call to a procedure or in returning from a procedure

Example CISC Instructions

A Motorola 68K CISC type instruction is:

CAS (compare and swap) - which compares the value in a memory location with the value in a data register, and copies a second data register into the memory location if the compared values are equal, otherwise copies the contents of the memory location into the first register

CISC Disadvantages

The instructions are difficult to implement in hardware and require microprogramming

microcode slows down the system – after fetching each machine instruction, a series of microinstructions must be retrieved and executed to interpret the machine instruction

Compiler writers often avoid using CISC instructions that are unique to one machine in favor of generating groups of more standard simple instructions to perform the same task

CISC Disadvantages

Intricate addressing modes and variable length instructions require a more complex and thus slower control unit

Allowing most instructions to reference memory operands makes the instruction take longer to execute

Complex instructions make pipelining more difficult