Reduced Instruction Set Computers

The acronym RISC stands for "**R**educed **I**nstruction **S**et **C**omputer".

RISC represents a design philosophy for the ISA (Instruction Set Architecture) and the CPU microarchitecture that implements that ISA.

RISC is not a set of rules; there is no "pure RISC" design.

The acronym CISC, standing for "**C**omplex **I**nstruction **S**et **C**omputer", is a term applied by the proponents of RISC to computers that do not follow that design.

The first designed called "RISC" date to the early 1980's. The movement began with two experimental designs

The IBM 801	developed by IBM in 1980
The RISC I	developed by UC Berkeley in 1981.

We should note that the original RISC machine was probably the CDC–6400 designed and built by Mr. Seymour Cray, then of the Control Data Corporation.

In designing a CPU that was simple and very fast, Mr. Cray applied many of the techniques that would later be called "RISC" without himself using the term.

Why CISC?

Early CPU designs could have followed the RISC philosophy, the advantages of which were apparent early. Why then was the CISC design followed?

Here are two reasons:

1. CISC designs make more efficient use of memory. In particular, the “code density” is better, more instructions per kilobyte.

   After all, memory is very expensive and prone to failure.

2. CISC designs close the “semantic gap”; they produce an ISA with instructions that more closely resemble those in a higher–level language. This should provide better support for the compilers.

Memory Is Expensive, Isn't It?

Here is an updated table for memory expenses. It will soon be obsolete.

Vendor	Date	Cost of memory	Cost of disk drive
MC6800	1974	$500 for 16KB RAM	$55,000 for 40 MB
MC68000	1979	$200 for 64 KB RAM	$5,000 for 10 MB
Micron 4	4/10/2002	$49 for 128 MB RAM	$149 for 20 GB
Hewlett Packard	3/13/2007	$120 for 1 GB $400 for 3 GB	$30 for 320 GB $50 for 400 GB

Note that the cost of each of random access memory and disk memory has dropped by about four orders of magnitude since 1974.

The costs of memory no longer justify a complex design.

What About Support for High Level Languages?

Experimental studies conducted in 1971 by Donald Knuth and
in 1982 by David Patterson showed that

1) nearly 85% of a programs statements were simple assignment, conditional, or procedure calls.

2) None of these required a complicated instruction set.

The following table summarizes some results from experimental studies of code emitted by typical compilers of the 1970's and 1980's.

Language	Pascal	FORTRAN	Pascal	C	SAL
Workload	Scientific	Student	System	System	System
Assignment	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

Summary of High–Level Language Support

Here is a summary of the results of the works cited above.

1. As time progresses, programs will be more and more written in a high–level language, with assembly reserved for legacy programs.
2. The compilers now written do not make use of complex Instruction Set Architectures, but tend to use very simple constructs: Assignments, Jumps, Calls, and simple math.
3. What compiler writers would really like is provision of a large number of general purpose registers.
4. A more complex ISA implies a slower control unit, as the clock rate must be set for the data path timing of the slowest instruction in the ISA, even if it is never used in actual code.

The RISC Design Strategies

The basic RISC principle: “A simpler CPU is a faster CPU”.

The focus of the RISC design is reduction of the number and complexity of instructions in the ISA.

A number of the more common strategies include:

1) Fixed instruction length, generally one word.
   This simplifies instruction fetch.
2) Simplified addressing modes.
3) Fewer and simpler instructions in the instruction set.
4) Only load and store instructions access memory;
   no add memory to register, add memory to memory, etc.
5) Let the compiler do it. Use a good compiler to break complex high-level language statements into a number of simple assembly language statements.

More RISC Design Principles

1. Use optimizing compilers that issue simpler instructions.
   Complex compilers are easy to develop and test.
2. Emphasize an ISA that allows simple and efficient instruction decoding.
3. Microcode is not magic. It should be used sparingly.
   Those instructions that require microcode for their implementation should be inspected closely and considered for replacement by simpler instructions.
4. Operations that access memory should be avoided as memory access is a very time consuming operation.
5. All arithmetic operations should take only registers as operands.
   Only register load from memory and register store to memory should access memory addresses.

Comparison of RISC and CISC

This table is taken from an IEEE tutorial on RISC architecture.

	CISC Type Computers			RISC Type	
	IBM 370/168	VAX-11/780	Intel 8086	RISC I	IBM 801
Developed	1973	1978	1978	1981	1980
Instructions	208	303	133	31	120
Instruction size (bits)	16 – 48	16 – 456	8 – 32	32	32
Addressing Modes	4	22	6	3	3
General Registers	16	16	4	138	32
Control Memory Size	420 Kb	480 Kb	Not given	0	0
Cache Size	64 Kb	64 Kb	Not given	0	Not given

Experience on the VAX
by Digital Equipment Corporation (DEC)

The VAX–11/780 is the classic CISC design with
a very complex instruction set.

DEC experimented with two different implementations of the VAX architecture.
These are the VLSI VAX and the MicroVAX–32.

The VLSI VAX implemented the entire VAX instruction set.

The MicroVAX–32 design was based on the following observation about the more complex instructions.

- they account for 20% of the instructions in the VAX ISA,
- they account for 60% of the microcode, and
- they account for less than 0.2% of the instructions executed.

The MicroVAX implemented these instructions in system software.

Results of the DEC Experience

The VLSI VAX uses five to ten times the resources of the MicroVAX.

The VLSI VAX is only about 20% faster than the MicroVAX.

Here is a table from their study.

	VLSI VAX	MicroVAX 32
VLSI Chips	9	2
Microcode	480K	64K
Transistors	1250K	101K

Notes:

1. The MicroVAX used two VLSI chips”
   One for the basic instruction set, and
   one for the optional floating–point processor.

2. Note that two MicroVAX–32 computers, used together,
   might have about 160% of the performance of the VLSI VAX
   at about half the cost.