

Comparison between CISC and RISC

Yi Gao, Shilang Tang, Zhongli Ding

{ygao1, stang2, zding1}@cs.umbc.edu

ABSTRACT

Performance comparison between RISC (Reduced Instruction Set Computer) and CISC (Complex Instruction Set Computer) is a quite popular topic, and many conclusions are made in previous work. This paper compares the set of two different architectures in a comprehensive way, try to capture the purely architectural advantages of RISC and CISC, and the development trends of the future architectures. Our comparison is not only based on the theoretical point of view, but also based on experimental results to support our conclusions. We choose MIPS R2000 (RISC) and Intel 80386 (CISC) as the comparison microprocessors, we compare them on several integer and floating-point benchmarks to verify and draw conclusions.

1. Introduction

RISC and CISC stand for two different competing philosophies in designing modern computer architecture. The debate between them has been going on for a long time and will likely continue. The difference between RISC and CISC can lay on many levels, lots of plausible arguments are put forward by both sides, such as code density, transistor counts, memory bottlenecks, compiler and decode complexity etc. This paper intends to compare the set of two different ideas in detail, their distinct characters, their possible specific application domains, their current scope and their future development to the CPU design.

The experiment is mainly based on the MIPS R2000 and Intel 80386 instruction sets, MIPS R2000 is a typical product of pure RISC while Intel 80386 is a typical kind of pure CISC chip. They appeared almost at the same time (mid 80's), also, they are both 32-bit processors, so we choose them as our targets. We select a series of integer and float-pointing benchmarks, use some tracing tools and make some statistics to make a series of comparisons between the set of two instruction sets.

In Section 2, we will present some basic knowledge about RISC and CISC, the evolution history from CISC to RISC, and the basic difference between them. In Section 3, we present a comparison between two kinds of chips: MIPS R2000 and Intel 80386, and obtain some meaningful statistical results, based on these results, we make some conclusions. Finally, in Section 4, we will have some discussion and figure out who wins in this battle over CISC and RISC, what's the future status.

2. Background

2.1 History

The IBM 360 system, created in 1964, was probably the first modern processor system, which initiated the idea of computer architecture in computer science and adopted micro-coded control. Micro-coded control facilitated the use of complex instruction sets and provided flexibility, thus appeared so-called Complex Instruction Set Computer (CISC). CISC was primarily motivated by a desire to reduce the "semantic gap" between the machine language of the processor and the high-level languages in which people were programming, the theory was that such a processor would have to execute fewer instructions and thus would have better performance. [2] Also, at that time, hardware was extremely expensive, thus fewer memory occupancy was strongly preferred. On the other hand, the compiler technology at that time was in its infancy, same as advanced programming language, people always used assembly language at that time, thus, such a design as CISC made people need not consider the influence of compiler on CPU performance.

CISC computers are based on a complex instruction set in which instructions are executed by microcode. Microcode allows developers to change hardware designs and still maintain backward compatibility with instructions for earlier computers by changing only the microcode, thus making a complex instruction set possible and flexible. Although CISC designs allow a lot of hardware flexibility, the supporting of microcode slows microprocessor performance because of the number of operations that must be performed to execute each CISC instruction. A CISC instruction set typically includes many instructions with different sizes and execution cycles, which makes CISC instructions hard to pipeline.

From the 60's CISC microprocessors became prevalent, each successive processor having more and more complicated hardware and more and more complex instruction sets. This trend still continues today from Intel 80486, Pentium MMX to Pentium III.

However, in the middle of 70's, people began to doubt the design philosophy behind CISC. With more and more complex instruction sets, decoding and execution of such instructions were complicated and time-consuming, also, the expensive overhead brought by them slowed down the execution of those more frequently used simple instructions. Moreover, with the development of high-level languages, making good use of the instruction set posed a problem to compilers, people recognized that compilers were unable to take advantage of the complex instruction sets. All these may finally decrease the performance. On the other hand, with the declining cost of memory devices and improved compiler technology, it may be feasible to consider simplifying the instruction set with the cost of larger code size and higher memory bandwidth requirements.

Based on above observation, RISC (Reduced Instruction Set Computer) chips evolved around the mid-1970s as a reaction to CISC chips. In 70's, John Cocke at IBM's T.J. Watson Research Center provided the fundamental concept of RISC, the idea came from the IBM 801 minicomputer built in 1971 which is used as a fast controller in a very large telephone switching system. This chip contained many traits a later RISC chip should have: few instructions, fixed-size instructions in a fixed format, execution on a single cycle of a processor and a Load/Store architecture. [3] These ideas were further refined and articulated by a group at University of

California Berkeley led by David Patterson, who coined the term "RISC". [4] They realized that RISC promised higher performance, less cost and faster design time.

The design philosophies behind RISC chip are "make common case faster" and "simple is best", which are based on the premise that 20% of a computer's instructions do 80% of the work. In a CISC chip, many very complex instructions never or seldom used, but they make the control unit extremely complex and thus have a high control unit delay. A RISC instruction set includes fewer and simpler instructions with hard-wired control, simpler processor pipeline, a larger number of registers, a smaller transistor count which makes it easier to design and cheaper to produce, and a higher clock rate etc. Since fewer instructions exist, it's also easier to write powerful optimized compilers. Also, with simpler and fixed-size instructions and hardware decoding, further performance improvements such as superscalar and speculation is possible easier.

As researchers continued into RISC during the 1970's and 1980's it became clear that the factors described above resulted in a speed increase over CISC designs.

However, with the fleeting of time, the battle over RISC and CISC became blur, though pure RISC machine may outperform pure CISC machine, but both of each have some bad faces which interfere their further improvement of performance. In 90's, the trend is migrating toward each other, RISC machines may adopt some traits from CISC, while CISC may also do it vice versa.

An example is Intel microprocessors, though they use a CISC instruction set and are considered CISC chips, the internal architecture has gradually migrated to RISC. Beginning with the Pentium Pro, Intel used a RISC core, converting CISC instructions to RISC-like instructions that Intel calls micro-ops (Figure 2.1). The micro-op overcomes much of the speed penalty of CISC architecture by converting all instructions to the same length before they are processed. Micro-ops also eliminate arithmetic operations that directly change memory by loading memory data into registers before processing. Also, Pentium is compatible with 80486 and outperforms RISC machine in performance by adopting superscalar and pipeline structure. With the addition of RISC core technology, MMX, and SSE, CISC performance has become very competitive with that of RISC computers.

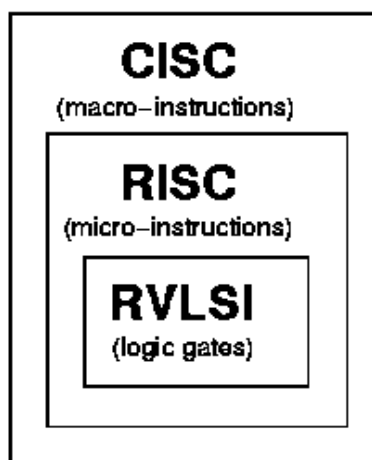


Figure 2.1

Another important thing we want to mention here is that although a significant number of microprocessors are based on RISC technology today, RISC never achieved the market penetration that its early proponents hoped for. In part, this limited acceptance was because the performance improvement offered by RISC was offset by a very large installed base of x86-compatible CISC computers. With large investments in software for CISC computers, corporation decision-makers could not justify switching to RISC in many cases.

In this section 2.2 we provide a simple comparison between CISC and RISC that is based on the architecture itself.

2.2 RISC versus CISC

A CISC processor has most of the following properties:

- Richer instruction set, some simple, some very complex
- Instructions generally take more than 1 clock to execute
- Instructions of a variable size
- Instructions interface with memory in multiple mechanisms with complex addressing modes
- No pipelining
- Upward compatibility within a family
- Microcode control
- Work well with simpler compiler

As time passed, one of the non-RISC architectures with a large market is the Intel x86 family, it has some specific characteristics became associated with CISC:

- Segmented memory model
- Few registers
- Crappy floating point performance

Typically CISC chips have a large amount of different and complex instructions. It is believed that hardware is always faster than software; therefore one should make a powerful instruction set, which provides programmers with assembly instructions to do a lot with short programs. In common CISC chips are relatively slower per instruction compared to RISC chips, but use less instructions than RISC.

Most actual RISC machines such as the RISC I and RISC II from the University of California at Berkeley and the MIPS from Stanford University have most of the following common properties: [6][9]

- Simple primitive instructions and addressing modes
- Instructions execute in one clock cycle
- Uniform length instructions and fixed instruction format
- Instructions interface with memory via fixed mechanisms (load/store)
- Pipelining

- Instruction set is orthogonal (little overlapping of instruction functionality)
- Hardwired control
- Complexity pushed to the compiler

Also, more ideas added to new RISC technology, including:

- Superscalar and out-of-order execution
- Large number of registers
- Fast floating point performance

The essence of RISC architecture is that it allows the execution of more operations in parallel and at a high rate than possible with a CISC architecture employing similar implementation complexity. It cannot only improve parallelism by pipelining, but also makes superscalar and out-of-order execution possible.

Back in the middle to late 80's, the battle over RISC and CISC is mainly non-Intel versus Intel x86, and RISC seemed to have a clearly upside, until the appearing of i486, Pentium and now PII, PIII. Now Intel's machines still run the old instruction set, but they adopt some RISC-like characteristics such as one clock execution, clean memory models, deep pipelining, superscalar operations, lots of registers and even out-of-order execution. They run faster and faster with a decent floating point performance. On the other hand, some RISC machines added more instructions to their architectures for new data types. So, it seems the RISC-CISC gap is narrowed.

So, nowadays, the difference between RISC and CISC is no longer one of instruction sets, but of the whole chip architecture and system. The designations RISC and CISC are no longer meaningful in the original sense. What counts in a real world is always how fast a chip can execute the instructions it is given and how well it runs existing software. [7]

In this section 3, we using two chips appeared in mid 80's to make a simple comparison between pure RISC and pure CISC architecture .

3. Experiments with MIPS2000 and Intel80386

Typical RISC system includes HPPA-RISC, IBM RT-PC, IBM RS6000, Intel's i860 and i960, MIPS R2000 (and soon), Motorola's 88K, Motorola/IBM's PowerPc, and Sun's SPARC etc. Typical CISC system includes DEC VAX, Motorola 68K and 680x0, Intel 80x86 etc. In this experiment part, we choose to compare the instruction set of Intel 80386 (CISC, 1985) and MIPS R2000 (RISC, 1986) that appeared almost at the same era.

In 1985, Intel extended the 80286 architecture to 32-bit 80386. The 32-bit 80386 is an advanced microprocessor optimized for multitasking operating systems and designed for applications needing very high performance. Its 32-bit registers and data path support 32-bit addresses and data types. In addition to a 32-bit architecture with 32-bit address space, the 80386 added new

addressing modes and additional operations. The 80386 also added paging support in addition to segmented addressing. Like 80286, the 80386 has a mode to execute 8086 programs without change. If we don't count detail instruction formats and operand combinations, the 80386 has 96 kinds of operators. Totally, 80386 has 16 registers (8 of them can be general purpose registers) and 8 additional floating point registers. Also, the Intel family adopts Little Endian addressing. The processor can address up to 4 G of physical memory and 64 T of virtual memory. The on-chip memory-management facilities include address translation registers, advanced multitasking hardware, a protection mechanism, and paged virtual memory. Intel 80386 introduced linear addressing, used a 32-bit design and contained 275,000 transistors. The first versions of Windows and OS/2 were introduced with this chip.

On the other hand, the MIPS R2000 (1986) design came from the Stanford MIPS project, which stood for Microprocessor without Interlocked Pipeline Stages, and was arguably the first commercial RISC processor. It is a 32-bit processor with an off-chip split cache for instructions and data. There are 32 general purpose integer registers and 16 separate 64-bit floating point registers. The integer pipeline has a depth of 5 and the floating-point pipeline a maximal depth of 6. It has only 2 addressing modes, its opcode is 6 bits, which limits the total number of instructions and it has one delay slot used for delayed branch. [8]

Table 3.0 gives a general survey of Intel 80386 and MIPS 2000. Both of them are 32-bit machines, and have the comparable limitation of hardware and compiler technologies since they appeared almost at the same time. Also, they are rather typical in their own category. From 80486, though the instruction set design was not changed, Intel adopted some RISC ideas gradually in its x86 series CPU. All above make us choose to compare these two chips. We believe such a comparison is meaningful.

Table 3.0 Summary of 80386 and MIPS R2000 architectures

	MIPS R2000	Intel 80386
Date announced	1986	1985
Instruction size (bits)	32	Variable
Address space (size, model)	32 bits, flat	32 bits, segmented with paging support
Data alignment	Aligned	No
Data addressing modes	2	11
Protection	Page	Segmented Scheme
Integer registers (number, model, size)	31 GPR * 32 bits	8 GPR * 32 bits, 6 segment registers * 16 bits, 2 other * 16 bits
Separate floating-point registers	16 * 32 or 16 * 64 bits	8 * 80 bits
Floating-point format	IEEE 754 single, double	IEEE 754 single, double, extended

3.1 The benchmarks

1. Benchmark selection

We select benchmarks used for comparison carefully; we try to select benchmarks that can cover as many faces as possible. The series of benchmarks we selected varies from real programs to

some famous ones we got from web, and they are all C programs. Following is a simple introduction of each of them.

Integer Benchmarks:

- *test1*: Test 1 is a program from Nullstone Corp. which is part of a compiler optimization analysis program. In this program, the code in the routines `cmppt()` and `cmppt_variant()` are identical with the exception that `cmppt()` returns -1, 0, or 1; and `cmppt_variant()` returns -1, 0, or 2. Therefore, the generated code for these two routines should be just as similar. However, some compilers use pattern matching to identify `cmppt()` as a SPEC routine and will generate code for `cmppt()` that is considerably faster than `cmppt_variant()`, even though the routines are virtually identical.
- *stanford*: This is a suite of benchmarks that are relatively short, both in program size and execution time. It includes 7 small programs: bubblesort, towers of Hanoi, the 8 queens problem, quicksort, integer matrix multiply, Perm and Sieve. The unusually small size of the codes overemphasizes the benefit of small caches. [10]
- *poly*: It is a very common and typical real-world integer program by a beginning programmer, since the percentage of such users is high in real world, for example, there are a lot of such programs running on GL machines every day, it is meaningful to see the behavior of such program. A good chip should make such users feel satisfying, that's why we choose it.

Floating-point Benchmarks:

- *whetstone*: Whetstone consists of a mix of floating-point and integer arithmetic, function calls, array indexing, conditional jumps and transcendental functions. Whetstone has been carefully arranged to defeat many simple compiler optimizations. It makes little use of memory and benefits little from the addition of cache memory.
- *var*: Corresponding to *poly*, it is a very common real-world FP program by a beginning programmer.
- *lre*: This is a real-world computation intensive program used in EE for research of Partial likelihood for Channel Equalization.

2. Benchmark compilation

Compiler has great influence on performance, to make a fair comparison, we use `cc` which comes with the SGI Unix OS along with an argument: `cc -mips1 *.c` to get MIPS R2000 instructions; use `VC's debug` to get 80386 instructions. In these two cases, both compilers don't do optimization to the assembly instructions, thus we get rather pure MIPS R2000 and Intel 80386 instructions.

3. Data collection

To MIPS R2000 we used *pixie* and *apixie-statistical program* written by yourself to collect dynamic instruction counts. To 80386, we used *gcov* and our own statistical program to dynamically count all kinds of instructions and other data. *Pixie* and *gcov* instrument programs to

collect enough information to determine how often each user-level instruction is executed, while our own statistical programs produce summary data based on these counts.

3.3 Results

Table 3.1a shows the frequency of instructions and instruction classes for 3 integer benchmarks for the 80386 instruction set and Table 3.1b shows the same data for 3 floating point benchmarks. These are some general statistics we make on our benchmark programs.

The observation that the most widely executed instructions are the simple operations of an instruction set can be verified in our Table 3.1c. Table 3.1c shows 10 simple operations of the 80386 instruction set that account for 96.25% of instructions executed for the collection of four benchmark programs. Also, we observe that our benchmarks use only around 15% of the whole instruction set.

Based on this observation, make these common case instructions faster can have a great performance improvement. One design philosophy behind RISC chip lies here, by including fewer and simpler instructions with hard-wired control, and implementing complex functionality using the combination of these simple instructions, a RISC chip is easier to design and cheap to produce and gets a higher clock rate.

Table 3.1a 80386 instruction mix for three integer benchmarks

Instruction	Stanford	Poly	Test1	Average
Mov(Load)	28.87%	44.47%	41.91%	38.42%
Move(Store)	11.51%	10.31%	11.46%	11.09%
Add	9.36%	15.99%	3.84%	9.73%
Sub	2.23%	0.73%	0.04%	1%
Mul	2.49%	3.24%		1.91%
Div	0.01%			
Cmp	9.83%	5.32%	15.26%	10.14%
Mov(Reg-Reg)	2.11%	0.83%	0.08%	1.01%
Mov(Imm)	1.27%	0.76%	0.08%	0.70%
Push,pop	11.93%	5.31%	0.37%	5.87%
Condbranch	9.57%	5.17%	15.29%	10.01%
Uncondbranch	5.62%	6.17%	3.84%	5.21%
Call	2.71%	0.71%	0.04%	1.15%
Ret	0.11%	0.42%	0.04%	0.19%
Shift	0.09%	0.04%		0.04%
And,or	1.68%	0.04%		0.57%
Other(Xor,not...)	0.60%	0.49%	7.53%	2.87%
FPload				
FPstore				
FPadd				
FPsub				
FPmul				
FPdiv				
FPcmp				
FPOthers				

Figure 3.1b80386instructionmixforthreefloatingpointbenchmarks

Instruction	Whetstone	Lre	Var	Average
Mov(Load)	26.28%	38.38%	20.92%	28.53%
Move(Store)	11.51%	7.43%	9.96%	9.63%
Add	4.26%	4.87%	8.98%	6.04%
Sub	2.8%	0.10%	4.69%	2.53%
Mul	1.44%	4.89%	2.34%	2.89%
Div				
Cmp	3.26%	4.30%	4.27%	3.94%
Mov(Reg-Reg)	2.99%	0.19%	0.04%	1.07%
Mov(Imm)	2.99%			1%
Push, pop	17.68%	1.47%	4.36%	7.83%
Condbranch	3.25%	4.57%	4.29%	4.04%
Uncondbranch	4.18%	5.26%	4.59%	4.68%
Call	2.08%	0.76%	4.43%	2.42%
Ret	1.49%	0.18%	0.04%	0.57%
Shift				
And, or				
Other(Xor, not...)	1.49%	0.28%	0.03%	0.6%
FPload	2.65%	5.33%	8.98%	5.65%
FPstore	4.13%	5.65%	8.35%	6.04%
FPadd	3.60%	4.39%	5.40%	4.46%
FPsub	0.40%	0.45%	1.83%	0.89%
FPmul	2.23%	10.28%	3.61%	5.37%
FPdiv	1.11%	0.51%	2.68%	1.43%
FPcmp		0.28%	0.03%	0.1%
FPOthers	0.09%	0.28%	0.03%	0.13%

Table 3.1cTop10integerinstructionsfor80386

Rank	Instruction	%total
1	Move	51.22%
2	Compare	10.14%
3	Condbranch	10.01%
4	Add	9.73%
5	Uncondbranch	5.21%
6	Push	2.94%
7	Pop	2.94%
8	Mul	1.91%
9	Call	1.15%
10	Sub	1%
Total		96.25%

Accordingly, to MIPS R2000, Table 3.2 shows the frequency of instructions for 3 integer benchmarks and Table 3.2b shows the same data for 3 floating point benchmarks. Generally, we can divide MIPS R2000 instructions into 4 categories: data transfer instructions (load, store, load imm, load FP, store FP etc.), arithmetic & logical instructions (add, sub, mul, div, shift, and, or, xor, not etc.), control instructions (compare, condbranch, uncondbranch, call, return, jump etc.) and floating-point instructions (FPadd, FPsub, FPmul, FPdiv, FPcompare, moveR-RFP, etc.).

Figure 3.2a MIPS2000 instruction mix for three integer benchmarks

Instruction	Stanford	Poly	Test1	Average
Load	34.21%	29.24%	26.27%	29.91%
Store	16.02%	13.79%	0.09%	9.97%
Add	18.85%	29.58%	13.14%	20.52%
Sub	1.76%	0.16%		0.64%
Mul	0.21%	2.62%		0.94%
Div				
Compare	6.15%	4.22%	8.69%	6.35%
Loadimm	2.49%	0.79%	21.52%	8.27%
Cond.Branch	7.49%	8.69%	17.29%	11.16%
Uncondbranch	1.73%	0.22%	4.28%	2.08%
Call	0.85%	0.68%	0.04%	0.52%
Return,jmpind	0.88%	0.73%	0.04%	0.55%
Shift	8.66%	9.18%	8.56%	8.8%
And	0.14%	0.09%		0.08%
Or	0.56%		0.08%	0.21%
Other(xor,not)		0.01%		
LoadFP				
StoreFP				
AddFP				
subFP				
MulFP				
divFP				
CompareFP				
Movreg-regFP				
OtherFP				

Figure 3.2b MIPS2000 instruction mix for the three floating point benchmarks

Instruction	Whetstone	Lre	Var	Average
Load	25.91%	8.31%	37.25%	23.83%
Store	5.78%	2.98%	4.86%	4.54%
Add	11.62%	28.19%	8.5%	16.1%
Sub	0.85%		2.52%	1.12%
Mul	1.19%			0.4%
Div				
Compare	2.16%	1.52%	3.03%	2.24%
Loadimm	3.95%	19.45%	2.99%	8.79%
Cond.Branch	4.53%	3.08%	3.68%	3.76%
Uncondbranch	0.42%	0.03%		0.15%
Call	1.73%	0.26%	1.03%	1.01%
Return,jmpind	1.73%	0.26%	1.03%	1.01%
Shift	3.35%	5.57	3.61%	4.18%
And		0.01%		
Or		4.89%	0.96%	1.95%
Other(xor,not)			0.96%	0.32%
LoadFP	17.52%	12.99%	6.74%	12.42%
StoreFP	11.89%	4.54%	2.66%	6.36%
AddFP	2.92%	1.52%	2.43%	2.29%
SubFP	0.42%	0.2%	0.79%	0.47%
MulFP	1.86%	3.6%	1.64%	2.37%
divFP	0.93%	0.18%	1.21%	0.77%
CompareFP		0.1%	0.02%	0.03%
Movreg-regFP	0.88%	1.55%	10.88%	4.44%
OtherFP	0.34%	0.77%	3.18%	1.43%

3.3.1 Instruction Count

The most obvious and important difference between RISC and CISC is their instruction counts. Table 3.3 shows the number of instructions executed for each of the 12 programs (the first 7 is part of Stanford benchmark) on the 80386 and MIPS R2000 and their ratio. Numbers less than 1.0 mean the MIPS R2000 executes fewer instructions than 80386. Also, it is surprising that though most of these programs have a larger IC on MIPS R2000 than on 80386, three of them have less IC on MIPS R2000 than on 80386, and the ratio is not very high, the average ratio is below 2. This means that RISC is possibly can have a higher performance with a higher clock rate and lower CPI but a not very larger IC.

Table 3.3 Executed instruction count comparison

Program	MIPS R2000	80386	Ratio(Mips/386)
Bubble	113946	139104	0.82
Hanoi	739	724	1.02
Matmul	53526	32123	1.67
Perm	15880	11277	1.41
Qsort	19134	18595	1.03
Queen	263206	290932	0.90
Sieve	498777	438048	1.14
Poly	50562	43371	1.17
Test1	467400410	525400305	0.89
Lre	565021997	196020959	2.88
Var	196311232	88392983	2.22
Whet	1216617	1022765	1.19

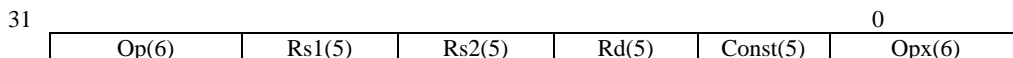
Table 3.4 Programs size comparison

Program	MIPS R2000	80386	Ratio(Mips/386)
Bubble	12900	7340	1.76
Hanoi	12116	7088	1.71
Matmul	13068	7904	1.65
Perm	12460	7136	1.75
Qsort	12916	7852	1.64
Queen	12524	7188	1.74
Sieve	12424	7104	1.75
Poly	16924	8148	2.08
Test1	17148	8048	2.13
Lre	17124	10836	1.58
Var	21824	11380	1.92
Whet	17608	10692	1.64

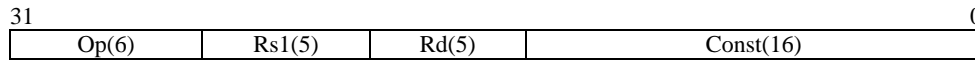
3.3.2 Instruction Format and Encoding

For MIPS R2000, every instruction is 32 bits (4 bytes) long with a fixed 6 bits opcode. It has 4 categories of formats: Register-Register, Register-immediate, Branch, and Jump/call.

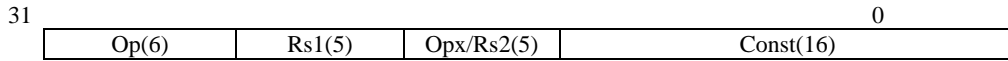
Register-Register:



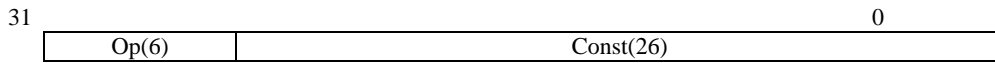
Register-immediate:



Branch:



Jump/Call:



The instruction encoding in 80386 is complex, with many different instruction formats. Instruction may vary from one byte, where there are no operands, to up to 17 bytes long. Table 3.5a shows the instruction formats for 80386. Table 3.5b shows the distribution of instruction sizes for 80386. The average number of bytes per instruction for integer programs is 2.82, while 3.76 for floating-point programs. This means CISChas better codedensity.

Table 3.5a The instruction format of the 80386. Every field is optional except the opcode.

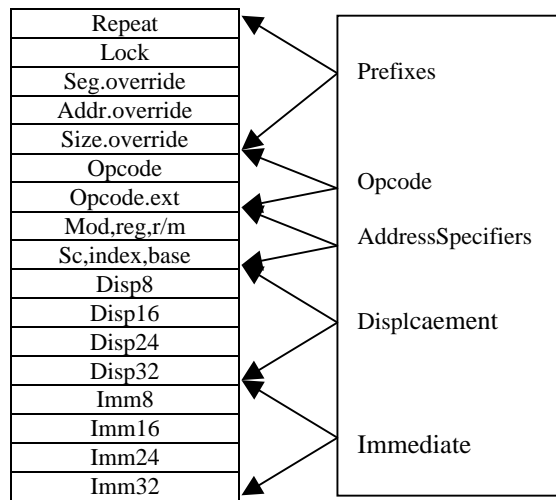
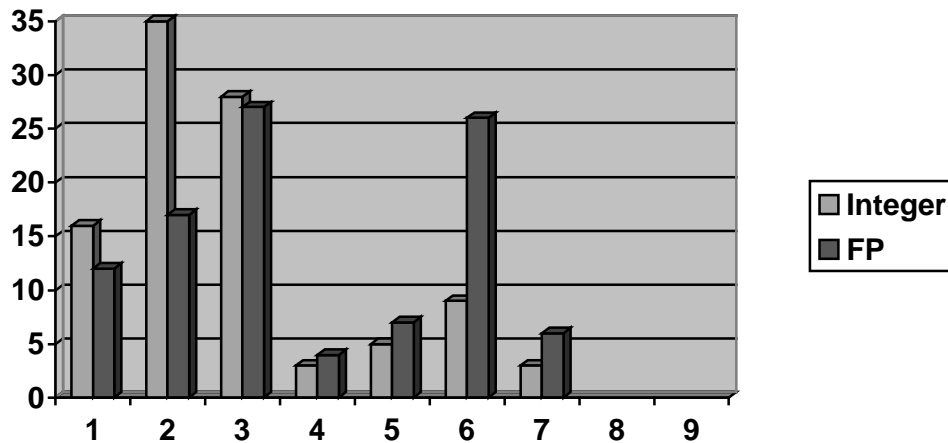


Table 3.5b 80386 instruction length distribution (Integer Ave. 2.82; FP Ave. 3.76)



3.3.3 Addressing modes

To MIPS R2000, it has only two kinds of addressing modes, which is “Register indirect” and “Immediate”. It accesses memory only through load/store instruction, to other instructions, the operands come from registers.

To 80386, the operand types of its 1st and 2nd operand can be “R,R”, “R,Imm”, “R,Mem”, “Mem,R” and “Mem,Imm”. It supports 7 data memory addressing modes:

- Absolute
- Register indirect
- Based
- Indexed
- Based indexed with displacement
- Based with scaled indexed and displacement

Displacements can be 8 or 32 bits, so if we count the size of the address as a separate addressing mode, 80386 has 11 addressing modes in total.

Table 3.6a gives operand addressing mode distribution by integer programs. This chart does not include the “8-bit direct” mode. Table 3.6b gives the same chart for floating-point programs.

Table 3.6a 80386 operand addressing mode distribution for integer benchmarks

Addressing mode	Stanford	Poly	Test1	Ave.
Register indirect	11.12%	0.42%	19.97%	10.51%
Base+8-bit displace	68.79%	79.71%	64.99%	71.16%
Base+32-bit displace		0.76%		0.25%
Indexed				
Based indexed+8-bit displace				
Based indexed+32-bit displace				
Base+scaled indexed		18.58%	9.99%	9.53%
Base+scaled indexed+8-bit displace		0.21%		0.07%
Base+scaled indexed+32-bit displace	18.46%	0.32%		6.26%
32-bit direct	1.62%		5.04%	2.22%

Table 3.6b 80386 operand addressing mode distribution for FP benchmarks

Addressing mode	Whetstone	Lre	Var	Ave.
Register indirect		0.70%	0.18%	0.29%
Base+8-bit displace		0.01%	0.11%	0.04%
Base+32-bit displace	58.38%	84.43%		47.60%
Indexed				
Based indexed+8-bit displace				
Based indexed+32-bit displace		9.77%		3.26%
Base+scaled indexed				
Base+scaled indexed+8-bit displace				
Base+scaled indexed+32-bit displace	16.68%	4.72%	1.35%	7.58%
32-bit direct	24.93%	0.36%	98.36%	41.22%

From these two tables, we can notice some very interesting things. First, we notice none of these programs use all 10 kinds of addressing modes, some modes such as “Indexed” and “Based indexed+8-bit displace” are never used by our selected benchmarks. Second, notice that integer

programs rely more on the generally shorter (not absolute) “Register indirect” and “Base+8-bit displace” addressing modes, while floating-point programs more frequently use the generally longer (also, not absolute) 32-bit displacement addressing modes such as “Base+32-bit displace”, “32-bit direct” and “Base+scaled indexed+32-bit displace”. This displays that the difference in average instruction length of these two kinds of programs in Table 3.5 arises partly from the differences in addressing modes.

3.3.4 Data Traffic

Finally, Table 3.7 shows the number of instructions in these programs that make accesses to memory. This table is useful for cache design.

Table 3.7 Data access distribution

	Data Read (Load)			Data Write (Store)		
	MIPSI	80386	Ratio	MIPSI	80386	Ratio
Bubble	50994	57365	0.89	11021	17760	0.62
Hanoi	247	128	1.93	132	43	3.06
Matmul	16360	12895	1.27	5004	3214	1.55
Perm	5874	1895	3.09	2396	1075	2.22
Qsort	8177	5973	1.37	1978	2149	0.92
Queen	78027	82365	0.94	54247	32236	1.68
Sieve	169584	110712	1.53	79804	86188	0.92
Poly	14786	19288	0.77	6873	4471	1.53
Test1	122800148	220200044	0.56	44000040	60200034	0.73
IntAve.	1.37			1.47		
Lre	47104715	75252704	0.63	16891273	14581224	1.16
Var	73133419	18490680	3.95	9545831	8806847	1.08
Whet	308582	268816	1.15	62320	117676	0.53
FP Ave.	1.91			0.92		

3.4 Conclusions

From our experiments described above, we get these conclusions:

- IC in 80386 is less than MIPS R2000, but not very much (average ratio < 2)
- CIS has richer instruction set, but in our case, the programs just use around 15% of all the instructions
- CIS has better coded density, 80386's average instruction length is less than MIPS R2000's
- 80386 has richer addressing modes, but in both integer and floating-point case, 3 addressing modes used for about 90% addressing, some addressing modes even never used
- To 80386, the operand type can be “Memory”, while in MIPS R2000, only load/store will access memory, all operands are in registers. Cache and memory designers should be aware of this difference.

The specific numbers of above measurements depend on the benchmarks chosen and the compiler technology used. Although we feel that our measurements are reasonably indicative of the usage of the two architectures, other programs or different compilers may yield slightly different numbers. But we believe what we do hereto Intel 80386 and MIPS R2000 primarily display the most common differences between the two architectures behind them.

4. Discussions

4.1 Who wins?

RISC processors gain a reputation for high performance, and our experiments above do verify this point. However, nowadays, the difference between RISC and CISC chips is getting smaller and smaller. RISC and CISC architectures are becoming more and more alike. Many of today's RISC chips support just as many instructions as yesterday's CISC chips. The PowerPC 601, for example, supports more instructions than the Pentium. Yet the 601 is considered a RISC chip, while the Pentium is definitely CISC. Furthermore today's CISC chips use many techniques formerly associated with RISC chips. Here may be we can simply say that RISC and CISC are growing to each other in their architecture in the theoretical point of view.

In reality, what counts is how fast a chip can execute the instructions it is given and how well it runs existing software. Today, both RISC and CISC manufacturers are doing everything to get an edge on the competition. In 90's, most new generations of processors employ a mixed bag of architectural features, including multiple execution units, pipelining, caches, and floating-point integration, thus making performance comparisons almost useless outside of a specific application.

Making a processor successful is more than just having the fastest chip available at an attractive price. There is a problem of ensuring quality software development tools are available. The typical high-end embedded product these days incorporates complex software, yet most deal with shrinking time-to-market requirements. This necessitates a productive software development environment. Many of the newer RISC processors cannot yet offer a tools environment comparable with x86 offerings. In an effort to accelerate the introduction of 5th generation processors on the desktop, Intel rapidly lowered Pentium prices, forcing down the price of the 486. Consequently high performance 486 processors - costing multi-millions of dollars to develop - became available at very aggressive prices. This all happened quickly and changed the landscape of the RISC vs. CISC battlefield.

It seems likely that the PowerPC, the i960, and many other low-cost RISC processors are not going to easily gain a significant performance advantage over future implementations of the x86 architecture. Embedded product designers are good at identifying where the best value is. When choosing a processor for an embedded real-time application, RISC generally doesn't have advantages over CISC, since most of real-time systems require very fast interrupt handling and high code density. In an embedded system, due to the size limitation of chips, it is unlikely to have large memory, so high code density is important. Also, CISC chips such as Motorola's 68K family provide better software availability for such a system. [5]

The debate between RISC and CISC will likely continue, even if the battle lines are now becoming fuzzy. This seems clear, no matter what your RISC or CISC persuasion [1]. But the future might not bring victory to one of them, but makes both extinct. So, who wins? No one wins.

But the

Finally, we want to point out that the biggest threat for CISC and RISC might not be each other, but a new technology called EPIC. EPIC stands for Explicitly Parallel Instruction Computing. Like the word parallel already says EPIC can do many instruction executions in parallel to one another.

EPIC is created by Intel and is in a way a combination of both CISC and RISC. This will in theory allow the processing of Windows-based as well as UNIX-based applications by the same CPU.

Intel is working on it under code-name Merced. Microsoft is already developing their Win64 standard for it. Like the name says, Merced will be a 64-bit chip.

If Intel's EPIC architecture is successful, it might be the biggest threat for RISC. All of the big CPU manufacturers but Sun and Motorola are now selling x86-based products, and some are just waiting for Merced to come out (HP, SGI). Because of the x86 market it is not likely that CISC will die soon, but RISC may.

So the future might bring EPIC processors and more CISC processors, while the RISC processors are becoming extinct. And finally, EPIC might make first RISC obsolete and later CISC too.

4.2 Summary

From our limited experience based on the results of four benchmarks, it appears that theoretically the pure RISC machines such as MIPS R2000 is a more promising style of computer design compared to Intel 80386 CISC chip that it has. With time fleeting, however, the bottom line between CISC and RISC becomes blur, in real world, people only care about how well a system can serve them, how fast a chip can execute the instructions it is given and how well it runs existing software. We think the adoption of each other's technology to overcome its own drawbacks may be more and more a trend in future CPU design. Also, other architectures different from CISC and RISC may appear.

Acknowledgements

We would like to express our sincere gratitude to Dr. Ethan L. Miller for his excellent teaching, timely help and support, and enlightening discussion.

References

- [1] Daniel Mann, "Why the x86 CISC beat RISC", from <http://www.amd-embedded.com/Benchmarks/whyx86.htm>
- [2] Keith Diefendorff, "History of the PowerPC architecture", *Commun.ACM* 37,6 (Jun. 1994), Pages 28-33
- [3] Radin, G. "The 801 Minicomputer", In *Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems* (March 1982), pp. 39-47
- [4] Patterson, D.S. and Ditzel, D.R. "The case for the reduced instruction set computer", *Computer Architecture News* 8:6 (Oct. 15, 1980), pp. 25-33.
- [5] Dennis Terry, "Choosing a Processor for Embedded Real-Time Applications", from <http://www.zytec.com/cp/html/choosingproc.html>
- [6] David A. Patterson, "Reduced Instruction Set Computers", *Commun.ACM* 28,1 (Jan. 1985), Pages 8-21
- [7] Jeff Prorise, "RISC vs. CISC: The Real Story -- What makes the PowerPC a RISC processor and the Pentium a CISC?", from <http://www.zdnet.com/pcmag/pctech/content/14/18/tu1418.001.html>
- [8] Margarita Esponda, Ra'ul Rojas, "The RISC Concept - A Survey of Implementations", from <http://www.inf.fu-berlin.de/lehre/WS94/RA/RISC-9.html>
- [9] David A. Patterson, Carlo H. Sequin, "RISCI: A Reduced Instruction Set VLSI Computer", 25 years of the international symposium on Computer architecture (selected papers), June 27 - July 2, 1998, Barcelona Spain, Pages 216-230
- [10] Thomas R. Gross, John L. Hennessy, Steven A. Przybylski, Christopher Rowen, "Measurement and evaluation of the MIPS architecture and processor", Volume 6, Issue 3 (1988), Pages 229-257