

A Comparison of Microprocessor Architectures

Jim Kleidon
WriteTex Communications

Rev 2.0



A Comparison of Microprocessor Architectures

INTRODUCTION.....	2
THE HISTORY OF MICROPROCESSOR DEVELOPMENT	2
PRIMARY MICROPROCESSOR ARCHITECTURES	4
CISC (Complex Instruction Set Computing)	4
<i>CISC Background</i>	<i>4</i>
<i>CISC Instruction Set Organization</i>	<i>4</i>
<i>CISC Hardware Implementation</i>	<i>5</i>
<i>CISC Applications.....</i>	<i>6</i>
RISC (Reduced Instruction Set Computing).....	7
<i>RISC Background</i>	<i>7</i>
<i>RISC Instruction Set Organization</i>	<i>7</i>
<i>RISC Hardware Implementation.....</i>	<i>8</i>
<i>RISC Applications.....</i>	<i>10</i>
VLIW (Very Long Instruction Word)	11
<i>VLIW Background.....</i>	<i>11</i>
<i>VLIW Instruction Set Organization/Hardware Implementation</i>	<i>11</i>
<i>VLIW Applications</i>	<i>11</i>
SUMMARY AND CONCLUSIONS.....	12
REFERENCES.....	13

Published by:

WriteTex Communications • 370 Quail Creek Rd. • Marble Falls, TX, 78654

Tel: 512-636-1976 • Email: jim@writetex.com • Web: www.writetex.com

February 2005

© Copyright 2005, WriteTex Communications. All rights reserved.

Introduction

After 35 years of development, the microprocessor is still the prime mover driving the global high-tech race. Three key microprocessor architectures have made this possible: Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), and Very Long Instruction Word (VLIW).

This whitepaper provides an overview comparing the three primary microprocessor architectures in reference to instruction sets, hardware implementation, performance tradeoffs, primary applications, and market proliferation.

Selected information sources include academic publications and technical literature provided by microprocessor manufactures. The discussion opens with a background and history of microprocessors, follows with descriptions and applications of CISC, RISC, and VLIW architectures, then concludes with a brief summary. The target audience includes industry professionals and engineering students with a solid background in computer technology.

The History of Microprocessor Development

J. Presper Eckert and John Mauchly at the University of Pennsylvania Moore School built the first electronic computer: ENIAC (Electronic Numerical Integrator and Calculator). Funded by the US Army during WWII and revealed in 1946, the ENIAC computed parabolic firing tables, a critical application for improving artillery targeting. ENIAC was 100 feet long and built with over 18,000 vacuum tubes and miles of wiring. Using ENIAC, a simple add operation took 200 microseconds. For comparison, today's microprocessors running at 2-GHz perform add operations in less than 1 nanosecond, using only a fraction of the hardware and power consumption.

By the 1950's, the UNIVAC series of mainframes became the first commercially used computers. Like the ENIAC, the UNIVAC machines consumed large amounts of space and power due to the continued use of vacuum tubes for implementing logic. It was not until the 1960's that transistors began to replace vacuum tubes, yielding an increase in compute speed and a reduction in hardware. During this period IBM, Burroughs, and Control Data dominated the infant commercial mainframe market.

The 1970's saw the development and use of integrated circuits—arrays of tiny transistors packed on a substrate of silicon. The integrated circuit revolution enabled the development of “mini computers” such as DEC's 16-bit addressing PDP-11, and “super computers” such as the Cray series of vector processing machines. By the late 1970's, Large Scale Integration (LSI) technologies enabled fabrication of a complete CPU on one silicon substrate.

A Comparison of Microprocessor Architectures.....

Intel Corporation gave birth to the term “microprocessor” by developing the 4-bit 4004 and 8-bit 8008 accumulator-style CPUs. In 1980, IBM selected Intel’s latest design, the 8088, to power the new IBM PC. IBM’s revolutionary PC launched the massive desktop personal computing industry we see today. Leveraging the success of the 8088, Intel’s 1984 development of the 80286 gave the IBM PC-AT 16-bit compute power and a 16-bit expansion bus for peripheral devices.

The 80286 executed in “real mode” to maintain the 640KB memory limitation of the 8086/8088. This limitation forced PC-AT software applications to execute strictly within the 640KB memory footprint. As software applications grew, the quest for faster CPU throughput, lower-cost, and lower-power technologies propelled microprocessor development from the low MHz speeds of 1980 to the multi-GHz speeds of 2005.

Primary Microprocessor Architectures

CISC (Complex Instruction Set Computing)

CISC Background

Because software compilers were in their infancy during the 1970's, computer instruction sets evolved around the needs of the human programmer, not the software. Ease of programming motivated rich instruction set development, and microcode updates enabled quick updates for new instructions. In these early years, memory capacity and price were at a premium, and program runtime footprints kept understandably small. This required that instruction sets be highly encoded, thus minimizing processor-memory bandwidth requirements. These restrictions led to the development of complex encoded instruction sets interpreted by microcode engines. The term CISC encompasses this approach.

CISC Instruction Set Organization

In the early days of the Intel x86 series, designers attempted to simplify and reduce the "semantic gap" between software and CPU instruction sets. This resulted in a large, richly typed instruction set to lower software development costs. To facilitate reuse and market proliferation, commercial CPU hardware required compatibility with existing computers of the same family. Hence, the creation of supersets and expansion of features to ensure hardware portability and accommodate high-level programming languages such as C, C++, and Java.

As a result, the mix of instruction types for a CISC machine can number in the hundreds. CISC instructions tend to vary in width (8/16/32/64-bit) and can specify individual or complete sequences of operations. As forward and backward compatibility requirements dictate, developers simply add more instructions to the CISC microcode ROM. See [Table 1](#) below for an overview of typical CISC instructions.

Operator Type	Examples
Arithmetic and Logical	Integer arithmetic and logical operations: ADD, SUB, AND, OR
Data transfer	Load/stores (move data: register/external memory, addressing modes)
Control	Branch, jump, procedure calls, traps
System	Operating system call, virtual memory management
Floating Point	Floating point operations: FADD, FMULT, FDIV
Decimal	Decimal: ADD, MULT, decimal→char conversions
String	String move, compare, search

Table 1: Typical CISC Instructions

CISC Hardware Implementation

At power-up, a CISC CPU accesses the microcode ROM, executing a bootstrap sequence of instructions to “enable” the processor into a startup state. During normal operation, the CPU microsequencer accesses microcode ROM as needed to execute instructions and active the appropriate control signals in a logical sequence. Due to the large number of clock states, a CISC instruction set must support microprogrammed control to implement the control unit logic as opposed to hardwired (hardware implemented) control. In most cases, hardware changes are not required as more instructions expand the microcode. In operation, the microcode program is stored inside an internal lookup ROM.

Because of shrinking transistor geometries, CISC developers were quickly able to boost performance by adding hardware resources. For example, the Intel 80386 was the first commercially available CISC to offer a complete 32-bit architecture. The 80386 included a 32-bit instruction set, a 32-bit data bus, and a 32-bit virtual address space while still maintaining object code compatibility with prior 16-bit Intel x86 processors.

In addition, the 80386 and follow-on CISC processors expanded integer data and address storage by simply adding more internal registers. Because a CISC microsequencer directly accesses local registers, memory I/O operations decrease, thus improving instruction execution efficiency. The 80386 was the first CISC to include an internal MMU (Memory Management Unit) supporting segmentation, paging, and protection through the full 32-bit virtual address space.

Beyond the 80386, the Intel Pentium series of processors built upon the earlier x86 designs, significantly enhancing the architecture’s performance by adding internal instruction and data caches, hardware branch prediction, and superscaler execution units. Superscaler CPUs fetch, decode, and execute multiple instructions during each clock cycle. Multiple instruction execution is made possible by implementing deep parallel instruction pipelines with multiple (ALU) Arithmetic Logic Units. See the [RISC Hardware Implementation](#) section below for a basic overview of instruction pipelining.

CISC Applications

One does not require statistics to see the impact of CISC in general, and the Intel x86 architecture in particular on the computing world. In 1994, the early Pentium designs offered little advantage over their RISC contemporaries. Not to be left behind, Intel's aggressive pricing, marketing, and rapid increase in clock speed enabled Pentium derivatives to dominate the desktop market by 1995, and the notebook market by 1996. By adding incremental superscaler features and improving the silicon process technology, Intel expanded the original Pentium design into the Pentium II-IV series and beyond.

Intel is not the only manufacturer of x86 CISC CPUs. AMD leveraged the x86 architecture with the K5 – K9 series, the K7 being the first commercially available microprocessor to hit 1GHz.

CPU	Year	Transistors	Speed	Cache	Architecture	Instruction Size
8086	1978	30 K	4 MHz	None	1 Integer ALU	16
80286	1982	134 K	6 MHz	None	1 Integer ALU	16
80386	1986	275 K	16 MHz	None	1 Integer ALU	16/32
80486	1989	1.2 M	33 MHz	Unified L1	1 Integer ALU	16/32
Pentium	1993	3.1 M	66 MHz	I/D L1	Superscaler	16/32
AthalonXP	2002	37 M	2.8 GHz	I/D L1-L2	Superscaler	16/32/64
PentiumIV	2002	40 M	3.0 GHz	I/D L1-L2	Superscaler	16/32/64

Table 2: Key CISC Microprocessors

Sources:

<http://www.intel.com/design/processor/index.htm>

<http://www.amd.com/us-en/Processors/ProductInformation>

RISC (Reduced Instruction Set Computing)

RISC Background

During the late 1970's, IBM began project 801 to create a load/store oriented instruction set architecture that would be two-five times faster than other CPU architectures. In 1980, David A. Patterson at UC Berkeley began a similar project and built two machines called "RISC-I" and "RISC-II". The Berkeley architecture included simple load/store instructions and a new concept termed "register windowing".

On a similar quest, John L. Hennessy at Stanford University published a description of an efficient pipelining and compiler-assisted scheduling machine termed the "MIPS" architecture. It was hoped that the new RISC architectures would have a lower Clock Per Instruction (CPI) rate than existing CISC architectures such as the Intel 80XXX and DEC VAX-11/780 architectures. This meant that RISC machines could execute more instructions within the same clock rate.

The initial premise of RISC vs. CISC was that 90% of clock cycles execute only 10% of the instruction set. To realize a high-performance single-chip CPU, the design would need to be simple, with just enough logic to perform basic tasks such as load-store of internal/external locations and arithmetic operations. With instruction sets growing more complex to support high-level compilers, CISC CPUs required more logic to implement extra functionality—thereby increasing transistor counts and die sizes. Larger die sizes translated to higher development/production costs and frequencies.

Patterson and Ditzel concluded that instructions added for a given compiler generally were useless for other compilers. They also discovered that replacing complex instructions with a small number of lower-level instructions yielded minimal loss in performance. Furthermore, compiler development for a CISC instruction set was more time-consuming and bug-prone than compiler development for the simpler RISC architecture.

RISC Instruction Set Organization

RISC microprocessors implement simple operations with fixed-width instructions (32/64). In general, most RISC processors include fewer than 100 instructions and execute at least one instruction per primary clock cycle. Even though the instruction set is small, RISC includes basic operations to accomplish system tasks. These tasks include data movement (internal/external load/store, internal register data movement), arithmetic, logic, shift operations, and simple branch instructions. Today, the RISC/CISC barrier is blurring, and many processors use hardware implementations derived from both classes of instruction sets.

RISC Hardware Implementation

Because a smaller instruction set yields simpler logic requirements, RISC uses hardwired control logic as opposed to the more complex CISC microcode implementation. Due to the reduced logic complexity, RISC processors can incorporate several performance enhancing concepts such as pipelining, multiple internal registers, and register windowing while still maintaining relatively low transistor counts.

Multi-stage Instruction Pipelining

An easy way to boost performance is to implement a design with deep instruction pipelines. An efficient instruction pipeline executes at least one instruction per clock-cycle. Multi-stage instruction pipelines are designed with pre-fetch, decode, execute, and storage logic that independently act on each pipeline stage.

Consider the optimal four-stage pipeline examples below:

Pipe1	Pipe2	Pipe3	Pipe4
empty			

Instruction Flow →

T0 CLK: pipeline is empty, the pre-fetch, decode, execute, and store logic are quiescent.

Pipe1	Pipe2	Pipe3	Pipe4
INSTR1			
Pre-fetch			
CLK			

Instruction Flow →

T1 CLK: pre-fetch logic loads the first instruction into Pipe1.

Pipe1	Pipe2	Pipe3	Pipe4
INSTR2	INSTR1		
Pre-fetch	Decode		
CLK+1	CLK+1		

Instruction Flow →

T2 CLK: INSTR1 moves to Pipe2 and decodes, the pre-fetch logic loads INSTR2 into Pipe1.

Pipe1	Pipe2	Pipe3	Pipe4
INSTR3	INSTR2	INSTR1	
Pre-fetch	Decode	Execute	
CLK+2	CLK+2	CLK+2	

Instruction Flow →

T3 CLK: INSTR1 moves to Pipe3 and executes, INSTR2 moves to Pipe2 and decodes, the pre-fetch logic loads INSTR3 into Pipe1.

Pipe1	Pipe2	Pipe3	Pipe4
INSTR4	INSTR3	INSTR2	INSTR1
Pre-fetch	Decode	Execute	Store/retire
CLK+3	CLK+3	CLK+3	CLK+3

Instruction Flow →

T4 CLK: INSTR1 moves to Pipe4 and stores, INSTR2 moves to Pipe3 and executes, INSTR3 moves to Pipe2 and decodes, the pre-fetch logic loads INSTR4 into Pipe1. As long as the pre-fetch unit maintains a full instruction pipeline, one instruction executes per clock-cycle.

Multiple Registers and Register Windowing

Most RISC and CISC designs use large numbers of general and special purpose registers (GPR/SPR) to store intermediate values. Extra registers enable fast data access since the CPU does not require external load/store to access required information. Reducing external memory referencing improves performance for applications requiring frequent calls to/from subroutines.

To boost performance, the SPARC (Scalable Processor ARChitecture) CPU from Sun Microsystems uses “register windowing” to pass parameters between subroutines. This concept utilizes fully accessible global registers and window pointer/window mask registers. Window pointer registers contain the address of active registers while window mask registers contain a bit flagging all registers containing valid data. As a result, the SPARC processor can pass parameters to subroutines through registers that overlap windows. Without this feature, parameters would be passed though external memory, consuming precious CPU and I/O cycles.

RISC Applications

The growth of RISC in the commercial market began in the late 1980's with the SPARC CPU from Sun Microsystems. Prior to this, Sun relied upon CISC architectures such as the Motorola 680x0 series of CPU's to power Sun's UNIX workstations. The UNIX operating system enabled multi-user/multi-tasking, as opposed to the single-tasking limitation of the original Microsoft Windows platforms.

Sun's quest for higher performance and lower CPU costs fueled initial SPARC development. Within a few years, SPARC replaced Sun's CISC-based machines. During this period, RISC-based UNIX workstations became the platform of choice for EDA (Electronic Design Automation) software.

Contemporary EDA tasks such as RTL simulation, RTL-gate synthesis, cell APR typically ran on HP/PA-RISC, MIPS RISC, and SPARC-based platforms. The raw compute power of RISC vs. contemporary CISC machines (80486 DOS/Windows PC's) allowed RISC to dominate the scientific and engineering world until the mid-1990's.

Following on the heels of Sun, the joint IBM/Motorola/Apple development of PowerPC RISC microprocessors from 1992 onward allowed Apple Computer to replace its line of 680x0 CISC machines. Apple's PowerPC machines target contemporary desktop users and professionals requiring high performance for tasks such as video editing, graphics design, and 3D-gaming applications. PowerPC established Apple as the only manufacturer of RISC-based systems with a measurable share of the desktop market.

CPU	Year	Transistors	Speed	Cache	Architecture	Instruction Size
SPARCI	1987	50 K	16 Mhz	None	1 Integer ALU	32-bit
PowerPC G4	2002	33 M	1.25 GHz	I/D L1-L3	Superscaler	64-bit
U-SPARCIII	2002	29 M	1.05 GHz	I/D L1-L2	Superscaler	64-bit
Itanium2	2002	221 M	1 GHz	I/D L1-L3	Superscaler	64-bit EPIC

Table 3: Key RISC Microprocessors

Sources:

- <http://www.apple.com/powermac/processor.html>
- <http://www.sun.com/processors/UltraSPARC-III>
- <http://www.intel.com/design/processor/index.htm>

VLIW (Very Long Instruction Word)

VLIW Background

The VLIW approach to CPU architecture is unique in comparison with CISC and RISC. As discussed above, both RISC and CISC architectures use superscaler methods with multiple execution units to achieve instruction parallelism at runtime. Thus, extra hardware is required to discover and exploit instruction parallelism. By contrast, instruction parallelism is explicit in VLIW. Consequently, VLIW architecture requires less hardware than RISC or CISC to achieve high performance. The downside is that VLIW requires sophisticated software compilers to assemble instruction streams into horizontal formats.

VLIW Instruction Set Organization/Hardware Implementation

A typical VLIW instruction set is similar to RISC except that the individual instructions are wider (64/128/256-bit) to enable multiple independent tasks. VLIW architectures are RISC-like in many other aspects. In fact, VLIW instructions resemble several RISC CPUs joined (bit-sliced) in horizontal connectivity. Without the need for sophisticated instruction decode and dispatch hardware to construct parallelism from a serial instruction stream, VLIW requires fewer transistors, consumes lower power, and uses a smaller silicon footprint than RISC. In practice, VLIW architectures move the complexity from hardware implementation to software implementation.

VLIW Applications

The VLIW architecture lends itself well to the embedded microprocessor market where application specific software dominates. Embedded products typically contain factory-installed software targeted at a specific usage model. VLIW is optimal for use in military and space-based applications where low power, fault tolerance, and real-time execution are crucial.

Applications requiring long and repetitive numeric computations are also key markets for VLIW since these applications do not require sophisticated branch prediction and dynamic scheduling. In addition, most DSP (Digital Signal Processors) are implemented using VLIW architectures. Examples of commercially available VLIW implementations include Intel's StrongARM, the Phillips TriMedia TM32A processor, and TI's TMS320C6x line of DSP cores.

Summary and Conclusions

Microprocessor architecture development continues in 2005, but not at the frenzied pace seen during the 1980's and 1990's. Today's microprocessor is a mature innovation, with development continuing at an evolutionary pace throttled by performance needs, market considerations, and the limitations of semiconductor technology.

During the 1990's, RISC architectures such as SPARC, MIPS, and HP/PA-RISC dominated the UNIX workstation environment. However, the recent development of Linux OS for x86 provides scientific and engineering applications ported to UNIX an opportunity to switch platforms. Low cost Linux/X86 platforms promise to eliminate the dominance of Sun Microsystems and other RISC-based workstations in the scientific and engineering market space.

Today the Intel x86 CISC architecture clearly owns the desktop consumer market. The surprising longevity of the CISC processor is due to at least two primary factors: First, the explosion of CMOS design and fabrication techniques enabled smaller transistor geometries and higher clock speeds. Second, the marketing power of Intel and Microsoft propelled open hardware and x86 application awareness into the consumer psyche.

In the near future, CPU designers will combine VLIW, RISC, and CISC architectures into one silicon die—enabling the best features of all three worlds. With this approach, software applications will seamlessly execute using the optimal architectural solution for the task.

References

Text References:

Carpinelli, John D. (2001). *Computer Systems Organization and Architecture*.
Boston, MA: Addison Wesley.

Hill, Mark D. (2000). *Readings in Computer Architecture*.
San Diego, CA: Academic Press.

Hennessy, J. L., Patterson, D. A. (1990). *Computer Architecture a Quantitative Approach*.
San Mateo, CA: Morgan Kaufmann.

Web References:

<http://www.apple.com/powermac/processor.html>

<http://www.sun.com/processors/UltraSPARC-III>

<http://lowendpc.com/tech/index.shtml>

<http://www.intel.com/design/processor/index.htm>

<http://www.amd.com/us-en/Processors/ProductInformation/>

<http://www.semiconductors.philips.com/acrobat/other/vliw-wp.pdf>