

Processor Designs for Embedded Systems

Some microprocessor history, and how the Imsys architecture fits in

The IBM PC and the Macintosh, both launched around 1982, are the last well-known computer lines that were based on traditional CISC processors. That is a processor having a rich and diverse instruction set thanks to an internal read-only microprogram store for the control of its internal operation. In 1980, Patterson (UC Berkeley) and Hennessy (Stanford) proposed a different kind of computer processor, a simplified but efficiently pipelined type of processor called RISC (“Reduced Instruction Set Computer”), which lead to SPARC and MIPS, indirectly to ARM, and also to most CPU architectures defined after 1984 – but many of those have failed.

The RISC was faster than other small computers, but the CEO of MIPS Technologies now admits that the RISC revolution has been a mistake. He says it built on a balance between CPU and memory speeds, but unfortunately that balance was transient¹.

In the mid eighties the CMOS IC processing technologies had already begun to make the RISC CPU too fast for its main memory, but RISC didn’t go away. The reason for this was the success of RISC-based Unix workstations. These were CMOS microcomputers with high performance for C language programs. C compilers were optimized for the small instruction sets of simple instructions used by RISC processors. The workstations soon had small cache memories integrated on their RISC processor chips, which allowed them to run the simple instructions much faster than personal computers (based on CISC microprocessors) could do. More important at the time, however, was that workstations became faster than much more expensive minicomputers, which then disappeared.

On-chip cache, in addition to the CISC control store, would have been too expensive for personal computers, especially since the PCs – before the advent of the graphical user interface and the onset of the PC-Macintosh and Intel-AMD competition – actually didn’t need particularly high speed. The fastest microprocessors were used in laser printers rather than in computers.

Once again the RISC had a perceived but transient speed advantage. As Maurice Wilkes, the computer technology veteran and inventor of microprogramming, later pointed out – due to the ever increasing density of CMOS logic, it was only a matter of time before the CISCs (like the X86 of the Windows PC) could also be equipped with on-chip cache. (Nowadays on-chip cache memories are much larger and more power consuming than a typical CISC control store, and a RISC needs more cache than a CISC due to the bad code density of RISC designs.)

¹ (Ref.: EE Times, 07/11/2005)



By that time, RISC had caught on – its increasing memory speed gap was a challenge for computer scientists, and the old CISC architectures were frozen by the great mass of legacy software and by their proprietary inner designs. The fight on the market between new RISC and established CISC architectures went on year after year; most people were convinced that RISC would win in the end, but that didn't happen.

There has been one important exception – one new type of application where RISC has gained dominance. This is in the area of software-intensive consumer applications (e.g. mobile phones), where the early availability of RISC cores for ASICs was a special circumstance. This should also prove to be transient, as the Java Virtual Machine and other open virtual machine interfaces replace the old proprietary CISC architectures with new reconfigurable CISC-type processor architectures. Power consumption is important in these applications, and that will ultimately make the RISC uncompetitive.

The increasing problems of RISC

The main problem with RISC is that its clock frequency is assumed to be the maximum frequency the arithmetic/logic unit (ALU) can handle, and that memory accesses need to match this frequency. Improvements in CMOS manufacturing technology increase the maximum ALU frequency by about 40% every 18 months as long as Moore's law continues to hold. But neither the possible frequency of main memory nor the performance requirements of applications increases at that rate.

The increasing "speed gap" must be bridged by increasing the size of the cache. The cache consumes more silicon area and power than the processor core itself. Thus, this is a RISC disadvantage that grows with time.

If the application requirements are lower and allow the clock frequency to be reduced, the cache can be reduced or eliminated and the speed advantage of a RISC design is lower. RISC may then no longer be the best choice at all, because its fundamental principles lead to bad code density (which requires more expensive and power-consuming main memory) and bad efficiency for byte and bit manipulation, signal processing, and interpretation (e.g. of Java bytecode). All of these processing modes are much more important now than they were in the computers of the eighties.

Moore's law can be utilized for improving performance or for reducing cost or power consumption. Decreasing cost and power consumption continues to lead to new applications for embedded processors, often with high volumes. This has led to new, more specialized processor types such as the digital signal processor and the network processor. Application niches remain for very high performance general-purpose processors made for Unix workstation-like workloads but their relative importance has declined

The decreasing problems of CISC

The main problem with developing new CISC platforms used to be legacy software. However, assembly level programming is now rare. That is, hardly anybody cares about the instruction architecture as long as the compilers can use it to produce efficient object code and there is a good platform of system software. A conventional CISC instruction architecture is then actually an unnecessary interface level. This was of course also one of the thoughts behind the original RISC-like ideas in the early seventies. The





designers of CISC architectures had the assembler programmer and the data path hardware in mind. More important now are interfaces designed to allow the efficient execution of high-level languages and application program interfaces.

A new reconfigurable CISC architecture

Efficiency cannot be achieved by making just the ALU work all the time; instead any important part of a high-level language program should be executed with as little overhead activity as possible. The interface, the building blocks of the object code, should be chosen by the designers of languages and compilers rather than by hardware engineers.

In the eighties simple stack-based processors were developed, inspired by the language Forth. (That language also inspired Adobe when they specified the document printing language Postscript.) But stack-based computers didn't enter the mainstream; the RISC principle had all the mind-share. In the nineties people at Sun developed the Java language – and also the Java Virtual Machine, which contains a complete, stack-based, instruction list inspired by Forth “words” but adapted for Java. It can be seen as a CISC instruction list, although quite unlike the traditional ones.

But compared to a RISC it is the opposite in every detail:

The JVM bytecode architecture is:

- not “reduced instruction set”
- not register-based
- not load-store (can combine memory access and execution)

The instructions are;

- not fixed-length (as many bytes as are needed – efficiently used)
- not one per cycle (they have different execution times)
- not simple (can do multiple memory accesses and/or multiple execution cycles)

Sun's own attempt to design a processor with the JVM instruction list was not successful. It was aimed at applications that could also be served by Intel X86 (or high-performance RISCs). Like those, it became much too big and expensive for most embedded applications, and the new computer-like types of products that should have used it failed on the market.

Since then, Java-enabling of embedded systems has usually been accomplished by software interpretation of the JVM bytecode instructions, which is very inefficient on a conventional processor, especially a RISC. This inefficiency sometimes led to slow performance, which in some designs motivated the addition of Java acceleration hardware. In other designs a JIT compiler was used, increasing memory requirement by many megabytes. Thus, for many years Java was not used in embedded systems because the cost was too high.



However, the advantages of Java are very important, and it is now used even in one very high quantity, very cost sensitive, and very silicon- and power-constrained type of product: the smartcard. The JavaCard VM is a simplified version of the JVM, but smartcards using the mobile phone version of JVM are now possible.

The Imsys processor is a reconfigurable CISC with very flexible execution logic and an unusually big control store (microprogram memory). It was not made to compete with computer processors but instead with the simplest, and by far most common, ARM processors and even the low-cost 8-bit microcontrollers that control the majority of the embedded systems in the world.

Its CISC instruction repertoire includes most of the Java bytecodes, i.e. it doesn't need to interpret these in a sequence of native CISC instructions.

The flexibility of the hardware machine is suitable not only for interpreting bytecodes in microcode, but also for execution based on other data formats and operations than those for which conventional CPUs are optimized. An example is multiply-accumulate on 16-bit fractions, which is common for signal processing, instead of addition of 32-bit integers as the RISC is optimized for. These kinds of operations are typically needed in relatively small algorithms - small enough to be assembly-coded. Furthermore they are often standardized and finite in number, and it is therefore worth spending some micro-programming effort to optimize them. Special microcode algorithms are therefore sometimes added, in the form of new CISC instructions, which can be used in assembly-level programs.

Imsys Java platform has been – and is still being – enhanced by microcode development. This includes new instructions for Java garbage collection, Ethernet MAC functions, RTOS functions, graphics primitives, audio and image processing, display and touch screen interfaces, MultiMediaCard and several communication interfaces, standard microprocessor bus emulation, etc. It can efficiently execute Java applications compiled for mobile phones. Also, the Imsys SNAP platform is a cost-competitive “snap-in replacement” for the least expensive complete Java-based microcontroller modules on the market – but has about 20 times higher Java performance.

Author: Stefan Blixt
CTO, Imsys Technologies AB