Arquitecturas y Organización de Computadoras : una entrevista que revela sus diferencias

Original: https://www.bcs.org/content/conWebDoc/7929

Computer architecture is dead - Long live computer organisation

Alan Clements, University of Teesside

Why has the speed of computer performance increased so much since 1970? In answering the question, Alan Clements of the University of Teesside explains the difference between computer architecture and computer organisation, and the significance of the latter with regard to how computers have evolved.

Year by year computers get faster at a rate unprecedented in any other human activity. From prehistoric to modern times, humans have gone from running at 15mps to flying fighters at 2,500mph - an increase of just over two orders of magnitude. Since 1970 computer performance has increased by four orders of magnitude. Why are today's computers so much faster? Partially it's due to improvements in semiconductor technology. Partially it's due to the way in which we organise computers. Surprisingly it has little to do with advances in computer architecture.

Hardware, architecture and organisation

The title of any text dealing with the operation of computers will include some arrangement of the words 'hardware', 'architecture' or 'organisation'[1,2]. Although 'architecture' and 'organisation' have been used synonymously in the past, today they have different meanings.

A computer's **architecture** is its abstract model and is the programmer's view in terms of instructions, addressing modes and registers. A computer's **organisation** expresses the realization of the architecture. Architecture describes **what** the computer does and organisation describes **how** it does it.

Architecture and organisation are independent; you can change the organisation of a computer without changing its architecture. For example, a 64-bit architecture can be internally organised as a true 64-bit machine or as a 16-bit machine that uses four cycles to handle 64-bit values.

The difference between architecture and organisation is best illustrated by a non-computer example. Is the gear lever in a car part of its architecture or organisation? The architecture of a car is simple; it transports you from A to B. The gear lever belongs to the car's organisation because it implements the function of a car but is not part of that function (a car does not intrinsically need a gear lever).

Computers read instructions from memory, fetch data, perform operations on the data and store results. They provide basic arithmetic, logical and shift operations. Some computers let you specify three operands, whereas some use two operands, forcing you to over-write a source operand.

Some computers allow you to operate data in memory whereas others restrict operations to registers. Some computers have lots of registers and some very few. Some let you access data in memory at a given address; others force you to access data indirectly via a pointer register.

And that's about it. There really is little variation between the instruction sets (i.e. architecture) of different computers.

Over the years, architecture has hardly changed and a computer pioneer of the 1950s would have little difficulty in following a modern microprocessor's assembly language. The lack of architectural change may be due to limited on-chip resources (e.g. the inability to address thousands of registers), the inability of compilers to exploit complex features or the need for backward compatibility (having to run last year's software on this year's PC has been a key factor in the PC's success).

Figure 1, from Hewlett Packard[3], illustrates the growth in computer performance. The straight line represents growth due to technological factors. The additional increase in performance is due to changes in organisation.

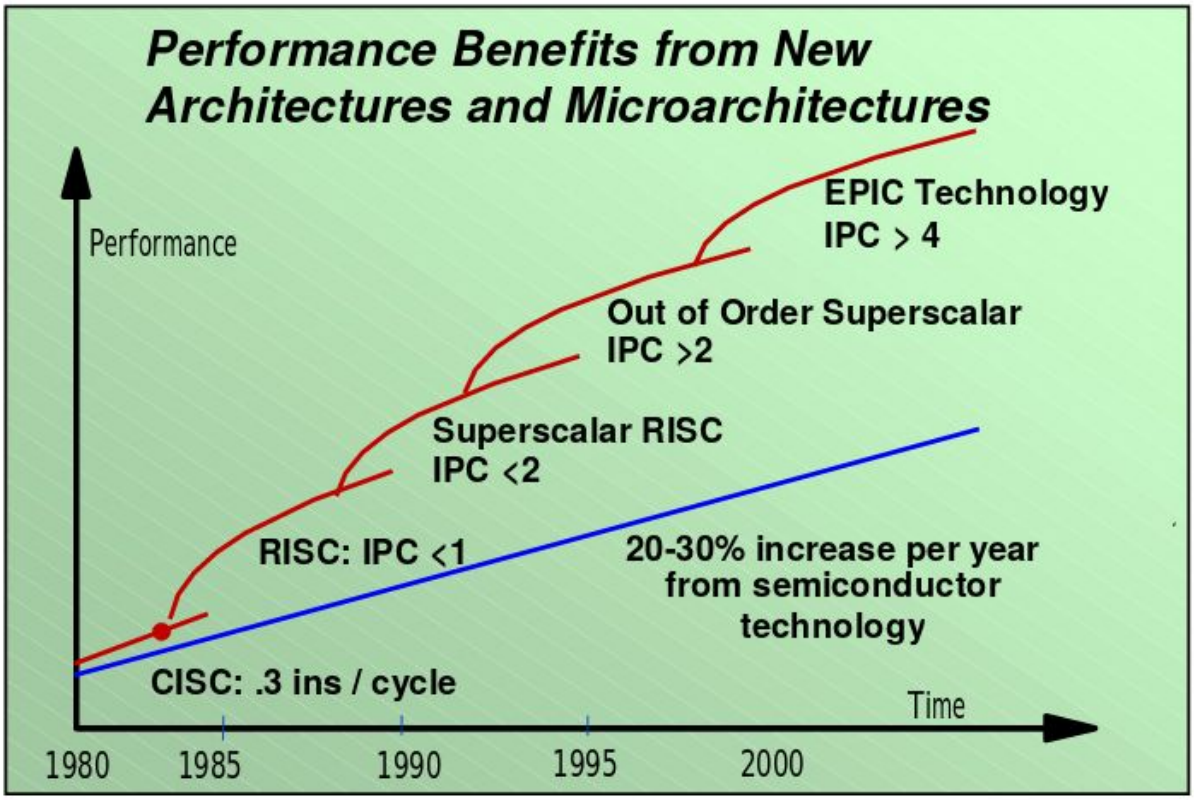

Figure 1.

Developments in computer organisation

Computers run faster not because of new instructions but because of the way in which instructions are implemented.

If you think of a computer as an **instruction factory** and how factories can be reorganised, you are well on the way towards understanding how computer organisation has advanced. A significant advance in computer organisation arrived with the RISC revolution of the 1980s when **pipelining** became popular.

Instead of executing an instruction to completion before starting the next one, instructions were executed in stages by dedicated functional units. As a new instruction was being fetched from memory, the previous one was being decoded; the operands required by the instruction before that were being fetched, and so on.

This technique split instruction execution into **n** phases and executed instructions up to **n** times faster than non-pipelined models.

Pipelining was not a radically new technique, not least because Henry Ford had long since applied it to the production of automobiles. Pipelining never lived up to its promise. Suppose an instruction that changes the flow of control such as a conditional branch is brought into the pipeline.

By the time it gets to the end of the pipeline and the computer realizes that it has to fetch instructions from a non-sequential address, all the instructions fetched after the branch are no longer needed. Such instructions are called **control hazards** and the wasted time slots in the pipelines are called stalls.

Computer designers have taken ever more sophisticated approaches to pipelining. Branch instructions are detected early and their outcome predicted (i.e. taken or not taken). If a branch is predicted as taken, the fetch engine immediately gets instructions from the **branch target address** into the pipeline.

For example, if BEQ XYZ is detected and the predictor guesses 'taken', instructions are fetched from location XYZ.

If the prediction is correct, the pipeline stall doesn't happen because the right data is in the right place at the right time. Of course if the prediction is wrong the pipeline has to be flushed. Remarkably clever techniques are used to predict the outcome of branches by looking at the semantic construct using the conditional branch.

For example, a **for ... loop** branches back on each iteration except the last one, so predicting taken is a safe bet. Other prediction techniques record the **branch pattern history** (rather like the gambler following horse races) and use the past history to guess future outcome of a branch.

Figure 1 illustrates the performance boost due to three variations in computer organisation: **instruction level parallelism (ILP)** that uses multiple execution units; **out-of-order** ILP where instructions are not executed in their natural program order allowing the computer to look ahead; and **explicit parallel instruction execution** where the burden of handling out-of-order execution is moved from the CPU to the compiler.

Each of these alternative ways of organising a processor provides a useful increase in speed.

The memory wall

Over the years, the speed of computers has increased more rapidly than the speed of memory. 'Memory wall' refers to the barrier to progress that memory seemingly presents because it takes far longer to access data from memory than to perform internal operations[4]. Possibly the greatest challenges facing computer designers is overcoming the **memory wall**.

Cache memory provided an interim solution by keeping a copy of frequently accessed data in fast RAM. Over the years cache has expanded from a few bytes to 4 Mbytes and research has investigated ever more efficient cache mechanisms; for example, the **victim cache** hangs on to data ejected from a full cache in case it is needed again in the near future. However even cache memory cannot overcome increasingly long memory latencies.

Because you can't eliminate memory **latency**, designers have sought to hide it. The introduction of **hyperthreading** provides a processor with two sets of registers, each of which holds the context of a thread of code (program counter, registers, processor status).

Suppose one thread is running and memory is accessed. Rather than waiting for the data, the second set of registers is activated to switch the context and allow another thread to run while the current thread is waiting for memory.

As manufacturing capabilities have improved, multiple processors can be located on a single chip providing the advantages of hyperthreading and allowing the simultaneous execution of instructions in both threads, as well as latency-hiding. You can even apply hyperthreading to both processors to allow the execution of four threads.

Multimedia drives new architectural features

Although developments in computer architecture have been rather modest over the last two decades, the introduction of special-purpose small vector operations (sometimes called **multimedia extensions**) provides a significant exception.

Although modern processors support 64-bit wordlengths, the 8- or 16-bit data elements used by audio and video processing in multimedia applications cannot easily exploit wide wordlengths.

Short vector operations divide a 64-bit register into eight bytes and perform operations on pairs of 8-byte words simultaneously; for example ADDP R1,R2 might add eight bytes in R1 to eight bytes in R2 and put the eight one-byte sums in R2. In one clock cycle, this action can add eight pairs of pixels in an image.

Multimedia extensions also provide a new data type, the **saturated integer**. In conventional binary arithmetic, the addition 1111 + 1 yields 0000 due to rollover. In saturated binary arithmetic the operation 1111 + 1 = 1111 because you cannot go beyond the highest value (hence the term saturation). Similarly, 0000 - 1 = 0000. Saturated arithmetic models multimedia behaviour; for example, a pixel cannot be whiter than white or blacker than black.

Summary

In this article I have suggested that the increase in speed of processors beyond that due to technological enhancements has been largely due to changes in organisation (pipelining, branch prediction, latency hiding) rather than architecture. However one of the few significant architectural enhancements has been provided to accelerate computationally intensive multimedia operations.

As transistor densities continue to increase, it will be interesting to see whether the same trend continues over the next decade with conventional architectures remaining essentially static and much of the emphasis being placed on parallelism, latency hiding or even value prediction (i.e. guessing the value of an operand before it is fetched from memory), or whether further domain specific architectural extensions will be incorporated to suit multimedia operations such as MPEG encoding and decoding.

References

1. Patterson DA, Hennessey JL (2004) Computer Organisation and Design:The hardware/software interface 3rd ed. Morgan Kaufmann.

2. Clements A (2000) The undergraduate curriculum in computer architecture. IEEE Micro, 20(3): 13–22.

3. Hewlett Packard Technical Paper (2002) Inside the Intel Itanium 2 Processor.

4. Wulf WA, McKee SA (1995) Hitting the memory wall: Implications of the obvious. Computer Architecture News, 23(1): 20–24.