Performance characterisation of 8-bit RISC and OISC architectures

Presentation speech

In this presentation I will describe my 3rd year project that compares reduced instruction set computer (RISC) and one instruction set computer (OISC) architectures.

I will start with a bit of history — since 70s a wide variety of computer architectures have risen and fallen, some of them, most noticeably x86, dominating market due to available software, marketing and so on. x86 is a complex instruction set computer (CISC), meaning it attempts to perform many complicated tasks in single instruction and been always orientated for performance rather than efficiency. For this reason ARM, which is reduced instruction set computer (RISC), found its way to low-power devices, most noticeably mobile phones over the past two decades. RISC architecture philosophy aims to have a few simple instruction that can be utilised by the software to carry out complex task, making the processor less complicated and more efficient. These two examples x86 and ARM are just a few to name.

The motivation behind this project is to take one more step back and design the ultimate RISC — only one instruction set computer or OISC and compare it to more traditional RISC architecture.

In order to perform a fair comparison between RISC and OISC, two novel architectures have been designed and implemented on FPGA with same design criteria and constrains. Designs were to be kept simple and with the aim of their being used in microcontroller applications, similar to Atmel's AVR architecture.

RISC is highly based on MIPS architecture and general structure *(see figure 1)*. Instruction is decoded by control block, which then controls how data travels in among different blocks as shown in diagram. One of the key points about this particular implementation is that instruction size is very small, as instruction is only 1 byte *(see machine code section)* and up to 3 additional bytes can be used depending on instruction. As an example, this is used to load a value from memory and these additional 3 bytes are used as address.

Moving to OISC, there are different implementations of it. Subtract and branch on condition is a common variant, however there seems to be not that much research on move variant which was implemented in this project. It uses 2 common data and instruction buses, both connected to a set of buffers *(see figure 2)*. Upon instruction execution, address matching buffers are enabled, connecting two source and destination blocks via data bus. Therefore, a general computing can be performed just moving data from one block to other, for example addition can be done by moving data to two accumulators, ACC0 and ACC1 blocks in diagram, and addition between them can be taken from "add" block as source.

After implementing both processors, I have written assemblers using Python and written a number of benchmarking programs to compare the performance of the RISC and OISC processors *(see result section)*.

Comparing two processors shows that they are quite similar in terms of power consumption *(see figure 4)* and FPGA resources *(see table 1)*. OISC requires more instructions to execute the same function *(see figure 3)*, which can be reduced by implementing multiple instruction and data buses. This solution would be quite easy to implement and should not require significantly greater resources, however it will be left for future work. In additional, OISC can be easily pipelined, and multiple buses can used for parallel computation, all hazards could be avoided by software. On the other hand achieving pipelining in RISC would require fairly complicated redesign.

Until the end of project I will investigate other factors such as instruction binary size and maximum frequency *(see future work section)*

In conclusion, this project showed that OISC-move architecture is simpler to implement and potentially better in few fields *(see conclusion section)*, however future investigation is needed.

Note to assessors: *The presentation was to have included experimental demonstrations of both processors on FPGAs, connected to a laptop via UART, running benchmarking programs that allow the user to type in integers between 0 and 60,000 and carry out primality tests on them.*