Performance characterisation of 8-bit RISC and OISC architectures

Mindaugas Jarmolovičius
Supervisor: Prof. Robert Killey

Department of Electronic & Electrical Engineering, University College London

Introduction

Motivation:
RISC (Reduced Instruction Set Computer) architecture is usually chosen over CISC (Complex Instruction Set Computer) due to simplicity and lower power consumption. This project goes one step further and investigates OISC (One Instruction Set Computer) MOVE variant architecture to determinate if it can achieve even better performance.

About:
The aim of this project was to design two novel RISC and OSIC architectures with following points:

- Design processors such that could be used for microcontroller application (like 8-bit Atmel AVR)
- Use same design criteria to make fair comparison
- Implement processors on FPGA board
- Design an assembly compiler and common functions

Decided design criteria:

- Minimal instruction size
- Minimalistic design
- 8bit data bus width
- 16bit ROM address width
- 24bit RAM address width
- 16bit RAM word size

RISC Architecture

Microarchitecture inspired by MIPS. Separate control and datapath.

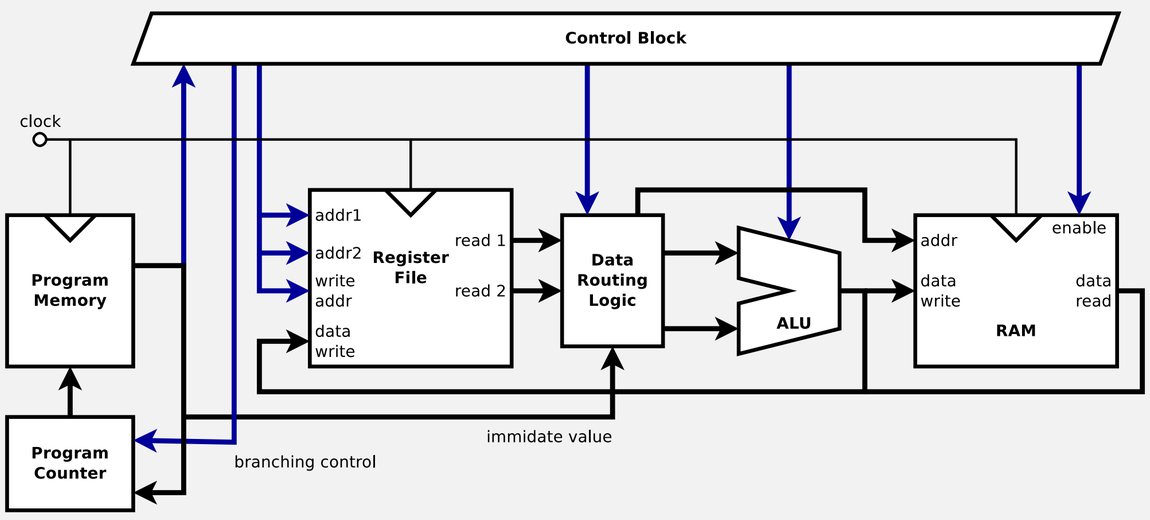

Figure 1: RISC architecture general block diagram

Machine code

RISC has three types of instructions - with 2, 1 and 0 operands maximising space efficiency. Each type can have from 0 to 3 additional immediate bytes.

45 Total instructions:

- **8** 2-operand instructions
- **28** 1-operand instructions
- **9** 0-operand instructions
- **4** General purpose registers
- Hardware stack
- Hardware multiply / divide
- Hardware call / return

OISC Architecture

Common data and instruction bus. No control block. Data is moved around using single MOVE instruction by enabling buffers connected to data bus.

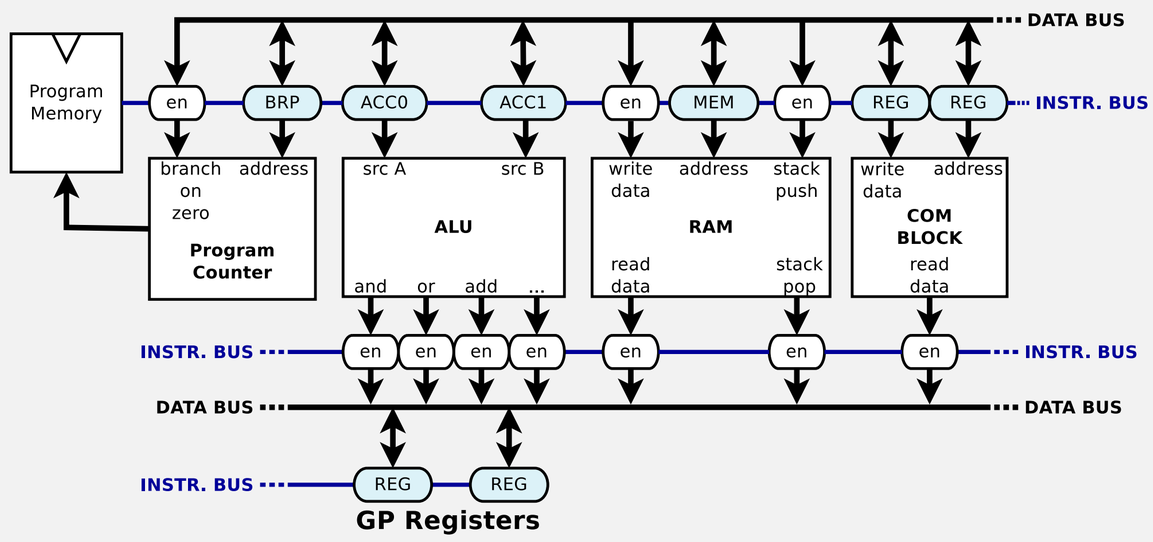

Figure 2: OISC architecture general block diagram

Machine code

OISC instructions are fixed 13bit width, 1 bit to set source as immediate value, 4bits for destination address and 8bit for source or immediate.

- **15** Destination addresses
- **41** Source addresses
- **2** General purpose registers
- Hardware stack
- Hardware multiply / divide
- Software call / return

Results

Implemented functions in assembly:

- Print Strings, values in Binary, Hexadecimal and Decimal (8 and 16bit)
- 16bit multiplication
- 16bit division
- 16bit modulus
- Sieve of Atkins (prime number calculator, up to 16bit number)

	Baseline	RISC	OISC
Logic Elements	293	1771	1705
Registers	169	602	724

Table 1: Number of FPGA resources

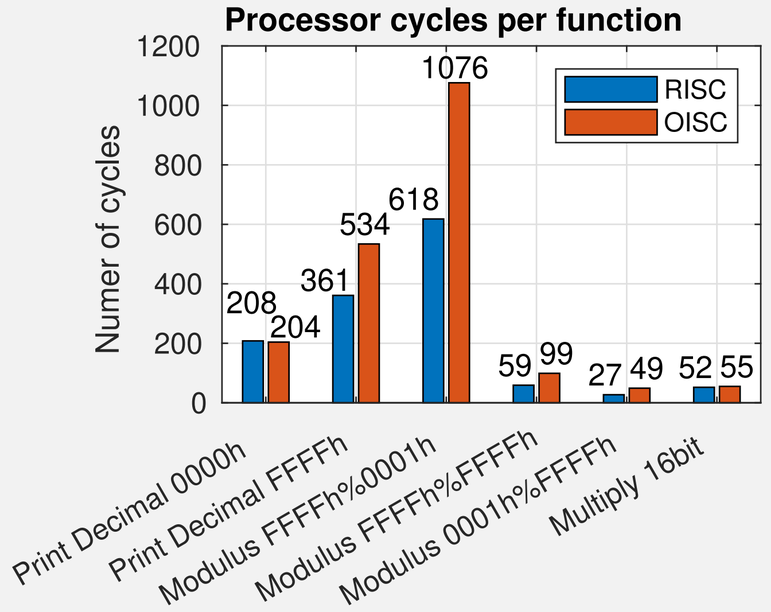

Figure 3: Simulated results of cycles that taken to perform function.

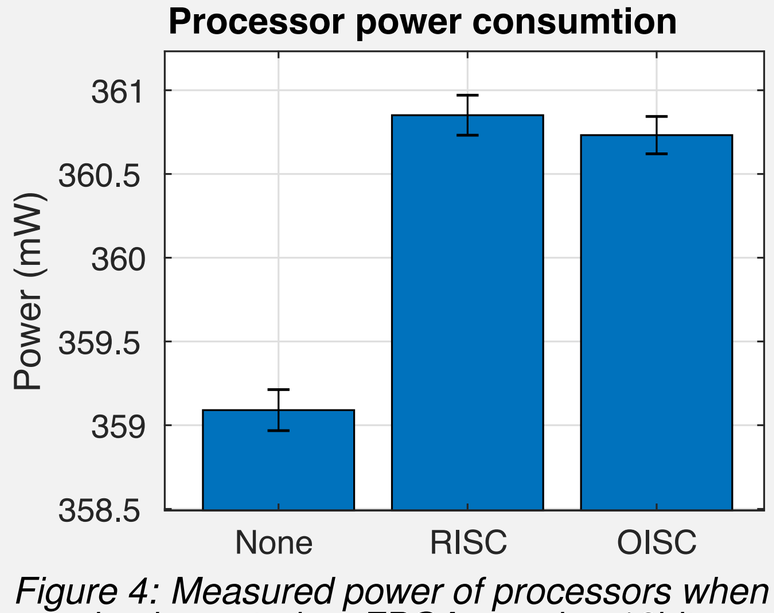

Figure 4: Measured power of processors when implemented on FPGA, running 16bit multiplication function in loop. None indicates auxiliary-only power

Conclusion

- Processor achieved similar performance in power consumption and FPGA resources
- OISC seems to be **easier** to implement and expand, easily enabling **pipelining** with hazard control implemented by software.
- OISC takes more instructions to perform same function.
- OISC assembly is more difficult to write, however this can be optimised with higher level compiler.
- Further research is need to investigate benefits of multi-data-bus OISC design.

Future work

Write more tests:

- Test Spongent (crypto-hashing) algorithm
- Array and matrix calculations (sorting, search, calculating mean, etc.)
- Other minimalistic cryptographic functions

Investigate:

1. Critical path and maximum frequencies
2. Power activity factor
3. Most commonly used instructions and patterns
4. Compare instruction memory efficiency

Future research:

1. Implement OISC emulator for easier debugging
2. Implement multiple data & instruction buses for OISC
3. Write higher level language compiler (such as BASIC or C)
4. Expand to 16bit / 32bit data bus
5. Compare to commercial Atmel AVR / ARM / MIPS architectures

Poster No. **C27**