University College London

Department of Electronic and Electrical Engineering

Performance characterisation of 8-bit RISC and OISC architectures

Author:
Mindaugas
Jarmolovicius
zceemja@ucl.ac.uk

Supervisor:
Prof. Robert
Killey
r.killey@ucl.ac.uk

Second Assessor:
Dr. Ed
Romans
e.romans@ucl.ac.uk

A BEng Project Interim Report

December 13, 2019

1 Abstract

This project investigates area / performance / power / complexity benefits and trade-offs of One Instruction Set Computer (OISC) comparing to traditional Reduced Instruction Set Computer (RISC) architectures in general computing application. It is shown that `SUBLEQ` OISC performs better in special cases, however there is lack of research in more complicated OISC architectures designed for general computing. At this point project is mainly following schedule with mainly benchmark being behind the schedule.

2 Introduction

Since 80s there been a raise of many processor architectures that try to fulfil specific performance and power application constraints. One of noticeable cases is ARM RISC architecture being used in mobile devices instead of more popular and robust x86 CISC (Complex Instruction Set Computer) architecture in favour of simplicity, cost and lower power consumption [6, 2]. Its been shown that in low power applications such as IoTs (Internet of Things), OISC[1] (One Instruction Set Computer) implementation can be superior in power and data throughput comparing to traditional RISC architectures [10, 1]. This project proposes to compare two novel RISC and OISC architectures and compare their performance, design complexity and efficiency.

Project is split into 3 main objectives:

• Design and build a RISC based processor. As it is aimed for low power and performance applications this will be 8bit data size processor with 4 general purpose registers.

• Design and build an OISC based processor. There are multiple different implementations such as `SUBLEQ` or one proposed in chapter 3.1.

• Design a fair benchmark that both processors could execute. Benchmark may include different algorithms that are commonly used in controllers, IoT devices or similar low power microprocessor applications.

Following chapters will describe the estimated project outcome, project schedule and work done so far.

3 The Work Performed to Date

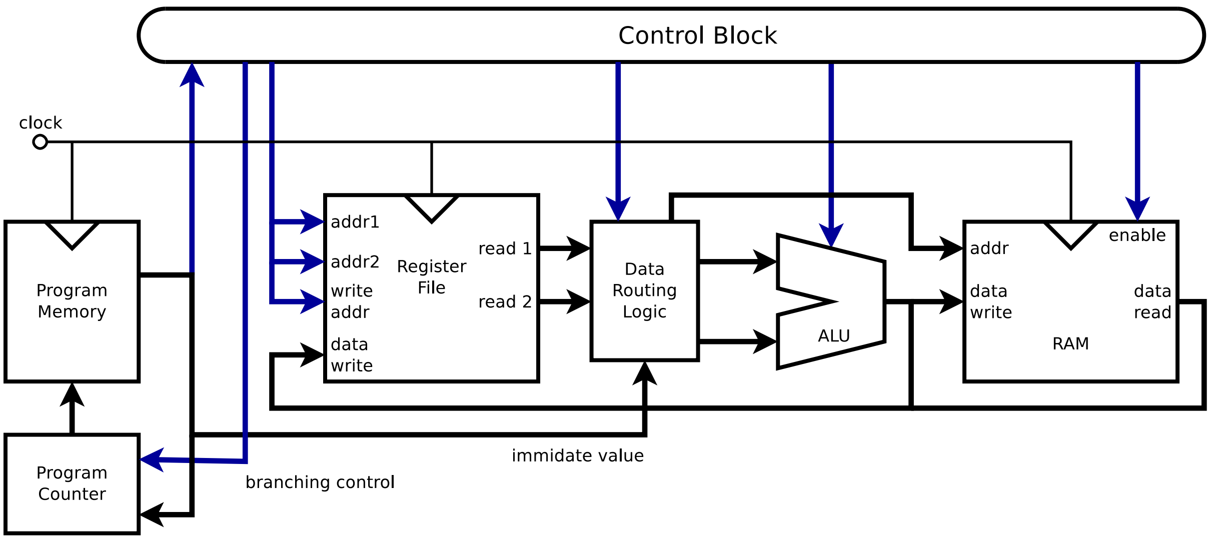
3.1 Supporting Theory

This section explains the theory and predictions of RISC and OISC architectures.

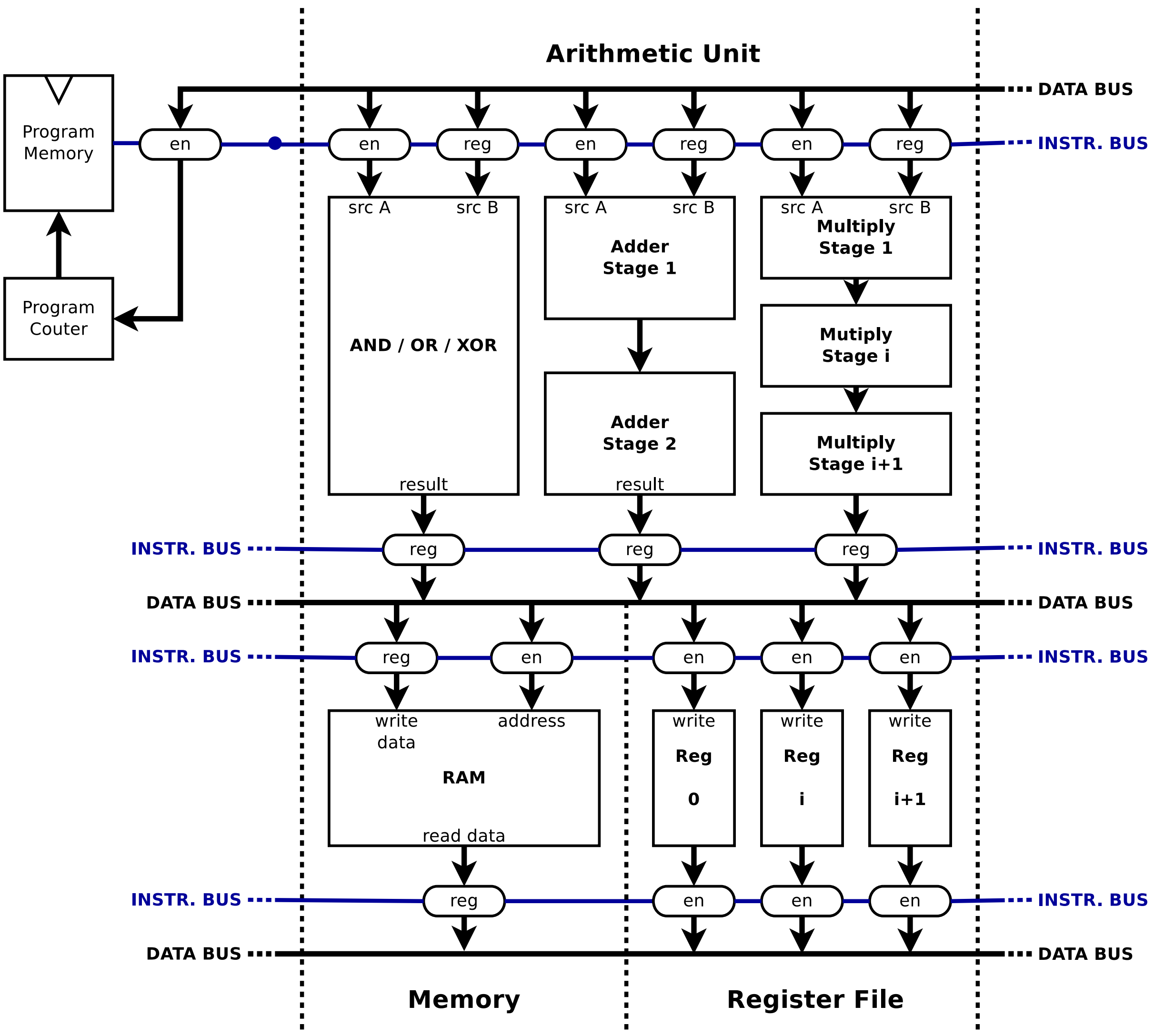
Figure 3.1.1 represents simplified diagrams of RISC and OISC architectures. In RISC and CISC architecture, program data travels from program memory to control block where instruction is decoded and further decided where what data is directed in datapath block which is described in section 3.3.6. Such structure requires complicated control block and additional data routing blocks. In order to increase performance of such processor one would need to add pipelining or add multiple cores. Both methods bring big disadvantages - multicore processor requires software adjustments and each core doubles the control and datapath substantially increasing die area; pipelinig allow operation at higher frequencies however it brings design complications such as complicated hazard prevention logic and instruction lookup. RISC architecture is mainly based on theory in [5]. Simplicity of OISC architecture overcomes these disadvantages:
Pipelining can be done by individual blocks and programmibly waiting for results, this is represented in figure 3.1.1b Adder and Multiply vertical blocks, multicore can be simulated by adding more data and instruction buses, hazards can be prevented with

[1]Also known as URISC (Ultimate Reduced Instruction Set Computer)

software or/and integrated into address registers.
ALU and any other processor component can be divided by adding different address registers thus allowing to utilise multiple components at the same time given that multiple data buses are used. This is represented in figure 3.1.1b Arithmetic Unit horizontal blocks. Assuming 4 data and instructions buses are used, **AND** and **OR** blocks sources A and B can be all written during one cycle utilising both blocks at the same time.
These principal functions should allow OISC architecture to have advantage in performance and power consumption while having lower transistor count and this is the main motivation behind project. This expectation is supported mainly by following papers:
• Using OISC `SUBLEQ` as a coprocessor for MIPS-ISA processor to emulate the functionality of different classes shows desirable area/performance/power trade-offs [1].
• Comparing OISC `SUBLEQ` multicore to RISC achieves better performance and lower energy for streaming data processing [10].
There is a lack of research investigating and comparing more general purpose OISC non-`SUBLEQ` processor. The main theory for OISC used in this project will be based on [9, 4, 7, 3].



(a) RISC microarchitecture diagram



(b) Single data bus OISC microarchitecture diagram

Figure 3.1.1: Simplified diagrams of both architectures. Blue lines indicate control/instruction buses and black - data buses

3.2 Project Scheduling

As it can be seen in table 1 below, project is mainly split into Term 1 which is dedicated for RISC and Term 2 which is dedicated for OISC implementation. Approximately 3 weeks are left until final report to have enough spare time to finish all tests and complete poster & report itself. It is also expected to have a lot of coursework around this time. Currently mainly benchmark development is behind schedule, more information in section 4.1.1.

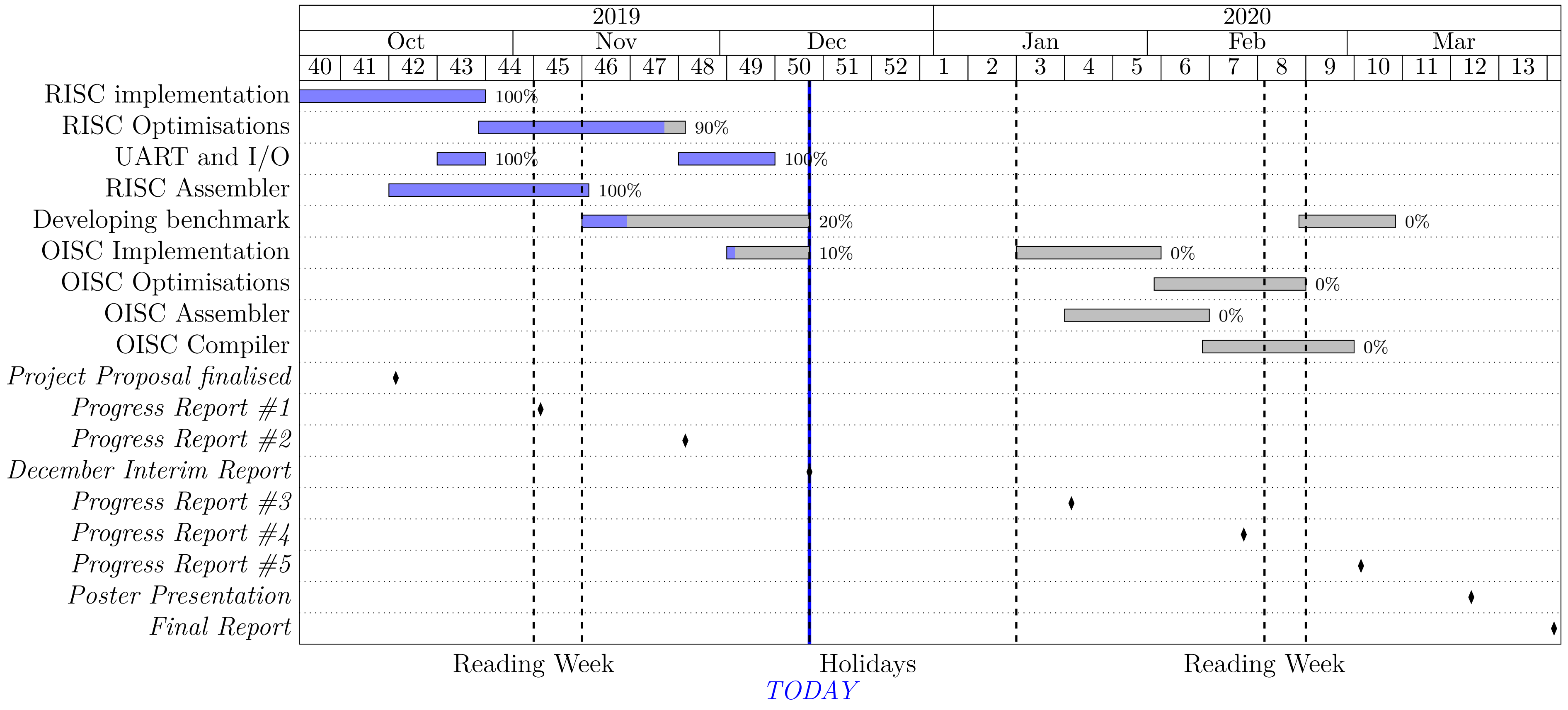


Table 1: Project schedule Grantt chart

3.3 Progress to Date

This section mainly includes progress on RISC processor and its components.

3.3.1 Memory

Initial plan was to use 32MiB 16bit SDRAM chip located at FPGA board. After successfully simulating most of processor functions, next step was to synthesise and run it on FPGA which brought 2 problems:

(1) Uploading program into ROM is not simple as generic Verilog unpacked register array cannot be initialised from file. Solution to this problem was solved by using FPGA built-in M9K memory that allows flexible RAM/ROM configuration. In addition, M9K memory can be read from/written to via JTAG connection without affecting FPGA operation which enabled quick method to upload programs without need to resynthesising processor code.

(2) Timing of SDRAM memory controller runs at much higher frequency than processor (at 100MHz versus 1MHz), initial implementation of interface between them was multiple 1-word length FIFO registers which caused memory read operation take 2 processor cycles. 3 possible solutions were considered - suspend processor clock while memory data is read which would harm performance; remove FIFO registers and rely on SDRAM clock being much greater than processor clock which is not ideal if higher processor clocks are intended to be used; use M9K. Last option was chosen due to ease of implementation and ability to read RAM content via JTAG for debugging purposes.

3.3.2 Instructions

Table 2 below represents RISC processor instructions that been implemented so far. In this processor design, instructions are 8 bit size, where 5th and 6th bits points to register 1 address and 7th and 8th bits to register 2 address. Instructions are separated in 3 groups:

- 2 register - uses both registers,
- 1 register - uses only register 1,
- 0 register - do not use registers.

Such grouping allowed to encode more instructions in 8 bits. At every processor cycle 4 bytes of instructions are fetched, first byte encodes instruction, last 3 may encode immediate value. This allows jump and conditional branch instructions to use 16bit jump address. Memory instructions `LWHI`, `SWHI`, `LWLO` and `SWLO` can use 24bit address that allows to access all addresses of SDRAM chip that been described in section 3.3.1. `MOVE` instruction is special case, if register 1 and 2 are different data will be copied from register 2 to 1. If these register values are the same, value from immediate will be copied to register 1.

Table 2: Instruction set for RISC processor

Instr.	Description	I-size[2]	Completed
	2 register instructions		
MOVE	Copy intimidate or register	1 or 0	x
ADD	Arithmetical addition	0	x
SUB	Arithmetical subtraction	0	x
AND	Logical AND	0	x
OR	Logical OR	0	x
XOR	Logical XOR	0	x
MUL	Arithmetical multiplication	0	x
DIV	Arithmetical division (inc. modulus)	0	x

Table 2: Instruction set for RISC processor

Instr.	Description	I-size[2]	Completed
	1 register instructions		
CI0	Replace intimidate value byte 0 for next instruction	1	x
CI1	Replace intimidate value byte 1 for next instruction	1	x
CI2	Replace intimidate value byte 2 for next instruction	1	x
SLL	Shift left logical	1	
SRL	Shift right logical	1	
SRA	Shift right arithmetical	1	
LWHI	Load word (high byte)	3	x
SWHI	Store word (high byte, reg. only)	0	x
LWLO	Load word (low byte)	3	x
SWLO	Store word (low byte, stores high byte reg.)	3	x
INC	Increase by 1	0	x
DEC	Decrease by 1	0	x
GETAH	Get ALU high byte reg. (only for MUL & DIV)	0	x
GETIF	Get interrupt flags	0	x
PUSH	Push to stack	0	x
POP	Pop from stack	0	x
COM	Send/Receive to/from com. block	1	x
ADDI	Arithmetical addition with intimidate	1	x
SUBI	Arithmetical subtraction with intimidate	1	x
ANDI	Logical AND with intimidate	1	x
ORI	Logical OR with intimidate	1	x
XORI	Logical XOR with intimidate	1	x
BEQ	Branch on equal	3	x
BGT	Branch on greater than	3	x
BGE	Branch on greater equal than	3	x
BZ	Branch on zero	2	x
	0 register instructions		
CALL	Call function, put return to stack	2	x
RET	Return from function	0	x
JUMP	Jump to address	2	x
RJUMP	Relative jump	2	
RETI	Return from interrupt	0	x
INTRE	Set interrupt entry pointer	2	x
CLC	Clear ALU carry-in	0	
SETC	Set ALU carry-in	0	
CLS	Clear ALU sign	0	
SETS	Set ALU sign	0	
SSETS	Enable ALU sign	0	
CLN	Clear ALU negative	0	
SETN	Set ALU negative	0	
SSETN	Enable ALU negative	0	

3.3.3 Program Counter

Figure 3.3.2 represents digital diagram for program counter. There are few key features about this design: it can take values from memory for stack pointer for `RET` instruction; immediate value (*PC_IMM2* is shifted by 1 byte to allow `BEQ`, `BGT`, `BGE` instructions as first immediate byte used as ALU source B); can jump to interrupt address; produces *pc_halted* signal when memory is read (`RET` instruction takes 2 cycles, because 1 takes to fetch address from stack and second to fetch instruction from instruction memory).

3.3.4 Immediate Override

Due to limited amount of instructions available and processor data width being 1 byte comparing to 2 bytes for instruction address and 3 bytes for memory address, it is no easy way to operate program pointers that can be dynamically calculated which limits program flexibility. Solution to this is IMO (Immediate Override), which logic is presented in figure 3.3.3. This circuit is connected between instruction memory higher 3 bytes (immediate value, *immr* bus) and immediate bus (*imm*) at datapath. Override bytes can be written with `CI0`, `CI1` and `CI2` instructions. On `CI0` a flag for next clock cycle is enabled that replaces *imm* value by ones stored in IMO registers. This circuit has two disadvantages:

1. Overriding immediate bytes take one or more clock cycles,
2. At override, *immr* bytes are ignored there wasting instruction memory space.

Point 2 can be resolved by designing circuit that would subtract amount of overridden IMO bytes from *pc_off* signal (program counter offset that is dependant on i-size value) at program counter, thus effectively saving instruction memory space. This solution however would introduce complication in assembler as additional checks would need to be done during compiling to check if IMO instruction are used.

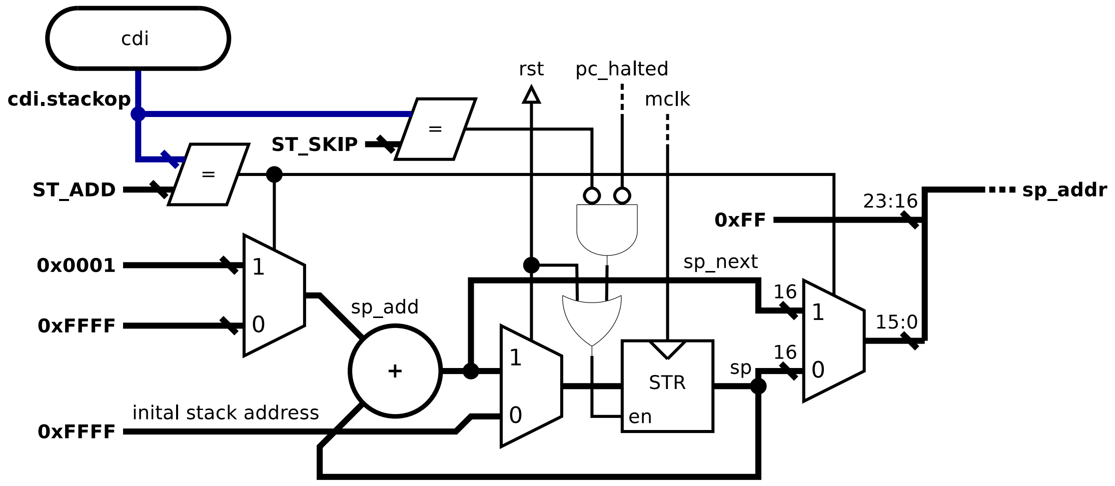


Figure 3.3.1: Digital diagram of RISC8 program counter

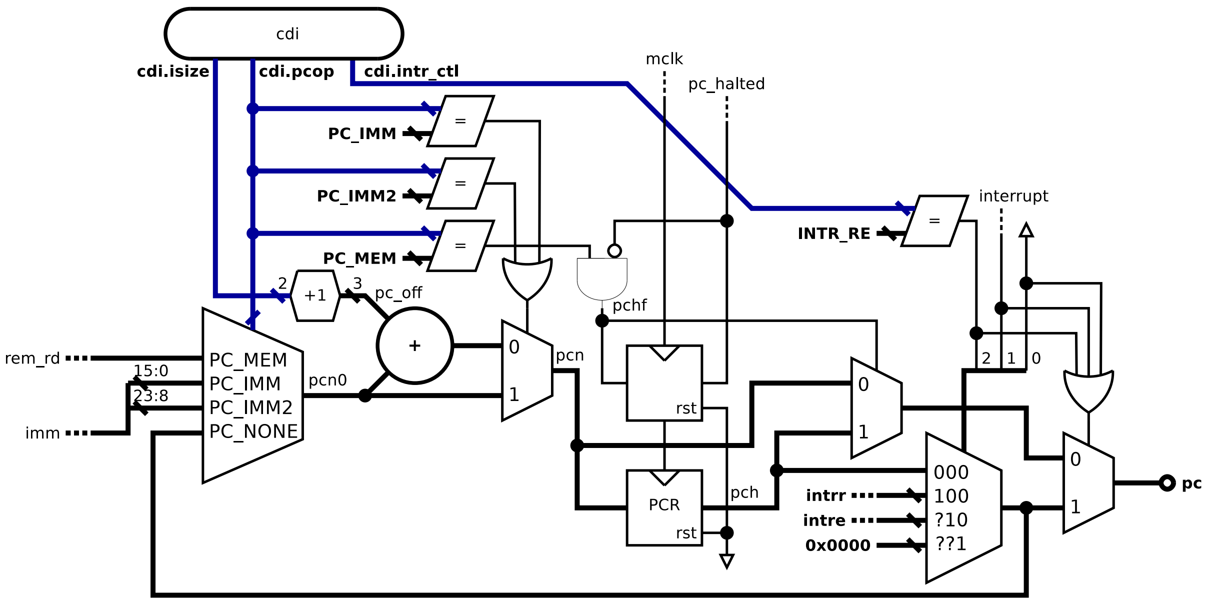


Figure 3.3.2: Digital diagram of RISC8 program counter

3.3.5 Stack Pointer

RISC processor implements stack pointer that is used in `PUSH`, `POP`, `CALL` and `RET` instructions. Stack initial address starts at highest memory address (`0xFFFF`) and subtracts 1 when data is put to stack. Figure 3.3.1 represents digital diagram for stack pointer. Note that stack is only 16bit in size and most significant byte is set to `0xFF`. It also supports *pc_halted* signal from program counter to prevent stack pointer been added by 1 twice during `RET` instruction.

One of the problems with current stack pointer implementation is 8bit data stored in 16bit memory address, wasting a byte. This can improved avoided by adding high byte register, however then it causes problems when 16bit program pointer is stored with `CALL` instruction. This can still be improved with more complicated circuit, or by using memory cache. Solution to this may be implemented in future.

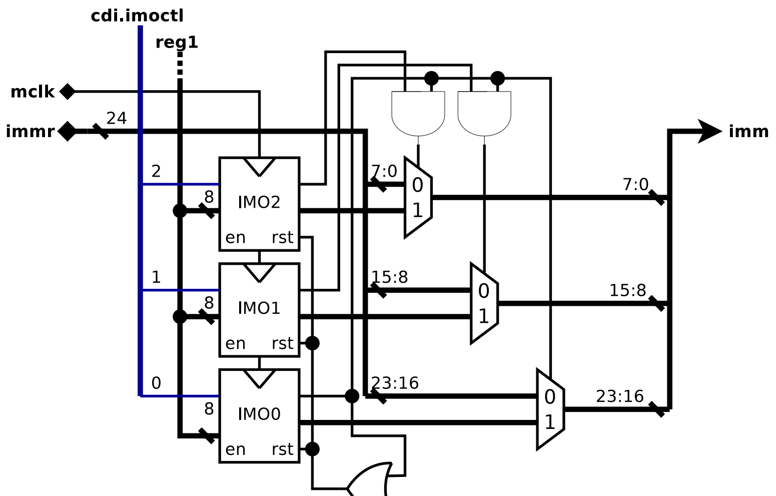


Figure 3.3.3: Digital diagram of RISC8 immediate override logic

3.3.6 Datapath

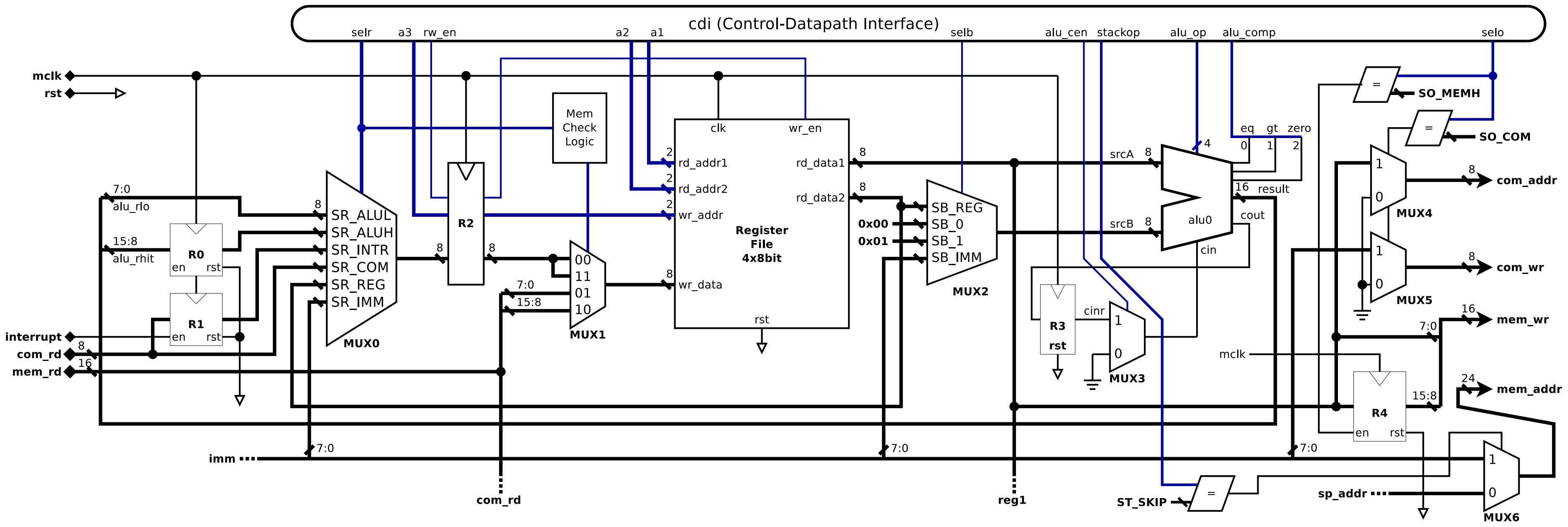


Figure 3.3.4: Digital diagram of RISC8 datapath

Figure 3.3.4 above represents partial datapath. Program counter, Stack pointer and immediate override logics are represented in figures 3.3.2, 3.3.1 and 3.3.3 respectively. CDI (Control-Data Interface) is HDL concept that connects datapath and control unit. Immediate value to datapath is provided by IMO block described in section 3.3.4.
Data to register file is selected and saved with *MUX0*. This data is delayed 1 cycle with *R2* to match timing that of data is taken from memory. If `LWLO` or `LWHI` is executed, *MUX1* select high or low byte from memory to read. To compensate for timing as value written to register file is delayed by 1 cycle, register file has internal logic that outputs *wr_data* to *rd_data1* or/and *rd_data2* immediately if *wr_en* is high and *rd_addr1* or/and *rd_addr2* matches *wr_addr*.

MUX2 allows override ALU source B, *R3* and *MUX3* enables control unit to enable ALU carry in allowing multi-byte number addition/subtraction. This function is not fully implemented yet.*MUX4* and *MUX5* allows to send data to COM block with `COM` instruction, if other instruction performed 0 byte for COM address and data is sent indicating no action. Data is stored in memory only with `SWLO` instruction writing to high byte whatever is stored in *R4* buffer. This buffer can be written to using `SWHI` instruction. *MUX6* selects memory address value from *imm* or stack pointer.

4 Summary of Difficulties and Issues

This chapter focuses on any difficulties and issues that are hindering project from moving forward.

4.1 List of Difficulties

This subsection will list difficulties currently encountered:

- Benchmark
- Assembler/Compiler

4.1.1 Benchmark

One of the difficulties is to design an appropriate benchmark that could test scenarios used in actual such processor applications. Other issue comes with writing benchmark itself, for instance benchmark test to finding prime number may have used many different algorithms where one of the fastest and used in actual industry may be "Sieve of Atkin" algorithm [8], however is it fairly complicated and time consuming especially when written in assembly.

4.1.2 Assembler/Compiler

In order to write more complicated code for benchmark, a proper compiler is required. Current assembler support definitions and labels, however not macros or imports from other files, nor linked libraries. Ideally, a common language compiler needs to be developed so that benchmark programs could be imported without need to completely rewrite them in assembly. However such compiler might take more time to develop than writing benchmarks in assembly. Therefore it needs to be decided which option to do.

4.2 Failure Assessment

This section describes likely possibilities of project failures:

As of current schedule OISC processor will be implemented in Term 2, however due to personal schedule project will be given about twice less time than in Term 1 which may result in not finishing OISC processor or developing all benchmarks for it on time. Mitigation for this is to closely follow schedule and adjust OISC design in such way as it would take less time to implement benchmark.

Another possible failure may be FPGA failure which would delay testing and benchmarking processors. This been already encountered, as JTAG caused errors while communicating with FPGA which indicated that FPGA chip is damaged. After long investigation its been discovered that problem was caused by bug in Linux JTAG Daemon service that needs to be simply restarted. Mitigation for any further issues is to have access to backup FPGA board.

4.3 Updated Safety Risk Assessment

There are no changes to Safety Risk Assessment.

5 Appendix A: Safety Risk Assessment

RiskNet report is appended at the end of this document.

6 Appendix B: Computer Code

All code to HDL processor implementation and assembler are in git repository that can be accessed in `https://gogs.infcof.com/min/ucl_project_y3`

7 References

References

[1] Tanvir Ahmed et al. “Synthesizable-from-C Embedded Processor Based on MIPS-ISA and OISC”. In: *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing* (2015). DOI: `10.1109/euc.2015.23`.

[2] E. Blem, J. Menon, and K. Sankaralingam. “Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures”. In: *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)* (2013). DOI: `10.1109/hpca.2013.6522302`.

[3] K. S. Dharshana, Kannan Balasubramanian, and M. Arun. “Encrypted computation on a one instruction set architecture”. In: *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)* (2016). DOI: `10.1109/iccpct.2016.7530376`.

[4] William F Gilreath and Phillip A Laplante. *Computer Architecture: A Minimalist Perspective.* Kluwer Academic Publishers, 2003.

[5] David Money Harris and Sarah L Harris. *Digital design and computer architecture.* 2nd ed. Elsevier, 2013.

[6] T. Jamil. “RISC versus CISC”. In: *IEEE Potentials* 14.3 (1995), pp. 13–16. DOI: `10.1109/45.464688`.

[7] J. H. Kong et al. “Minimal Instruction Set FPGA AES processor using Handel”. In: *2010 International Conference on Computer Applications and Industrial Electronics* (2010). DOI: `10.1109/iccaie.2010.5735100`.

[8] François Morain. “Atkin’s Test: News from the Front”. In: *Lecture Notes in Computer Science* (1989), pp. 626–635. DOI: `10.1007/3-540-46885-4_59`.

[9] Jia Jan Ong, L.-M. Ang, and K. P. Seng. “Implementation of (15, 9) Reed Solomon Minimal Instruction Set Computing on FPGA using Handel-C”. In: *2010 International Conference on Computer Applications and Industrial Electronics* (2010). DOI: `10.1109/iccaie.2010.5735103`.

[10] Minato Yokota, Kaoru Saso, and Yuko Hara-Azumi. “One-instruction set computer-based multicore processors for energy-efficient streaming data processing”. In: *Proceedings of the 28th International Symposium on Rapid System Prototyping Shortening the Path from Specification to Prototype - RSP ’17* (2017). DOI: `10.1145/3130265.3130318`.

Risk Assessment

Summary

Reference: RA030726/1 | **Sign-off Status:** Authorised

Date Created:	09/10/2019	**Confidential?**	No

Assessment Title:	3rd year project: Performance characterisation of 8-bit RISC and OISC architectures
Assessment Outline:	New activity

Area Responsible (for management of risks)		**Location of Risks**	On-Site
Division, School, Faculty, Institute:	Faculty of Engineering Science	**Building:**	Roberts Building
Department:	Dept of Electronic & Electrical Eng	**Area:**	Ground and Above
Group/Unit:	All Groups/Units	**Sub Area:**	Laboratory

Further Location Information:	Roberts building Rooms 704, 905. Also working from home.

Assessment Start Date:	09/10/2019	**Review or End Date:**	31/03/2020

Relevant Attachments:	
	Description of attachments:
	Location of non-electronic documents:
Assessor(s):	Jarmolovicius, Min
Approver(s):	ROBERT KILLEY
Signed Off:	ROBERT KILLEY (09/10/2019 12:56)
Distribution List:	Gerald McBrearty (g.mcbrearty@ucl.ac.uk) - 09/10/2019 ANDREW MOSS (andrew.moss@ucl.ac.uk) - 09/10/2019

PEOPLE AT RISK (from the Activities covered by this Risk Assessment)

CATEGORY
Undergraduates

Risk Assessment

Activities, Hazards, Controls

Reference: **RA030726/1** **Sign-off Status:** **Authorised**

1. Working in a lab

Description of Activity:

Hazard 1. Using computer

RSI Eye strain Bad posture	**Existing Control Measures** Properly use mouse/keyboard Make constant breaks Make sure sit properly

Hazard 2. Testing FPGA

Burns from hot wires/chips due to short-circuits	**Existing Control Measures** Use current limit on power supplies

Risk Level

With Existing Controls:

Risk Level	🟩 **A - Very Low / Trivial**