Multi-core RISC Processor Design and Implementation (Rev. 1.00)

ELEC5881M - Interim Report

Ben David Lancaster
Student ID: 201280376

Submitted in accordance with the requirements for the degree of
Master of Science (MSc)
in Embedded Systems Engineering

Supervisor: Dr. David Cowell
Assessor: Mr David Moore

University of Leeds
School of Electrical and Electronic Engineering

April 15, 2019

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Revision History

Date	Version	Changes
20/05/2018	3.14	Add background research to appendix.
19/05/2018	3.13	Update abstract to align with guidelines.
19/05/2018	3.12	Fix ISA pseudo-codes.
11/03/2018	1.00	Initial section outline.

Table 1: Document revisions.

Acknowledgements

I would like to thank my project supervisors Nigel Barlow and Serge Thill for their support and guidance throughout this project.

I would also like to thank James Spalding (Spirent Communications) and firmware team for their encouragement, ideas, and industrial sponsorship supporting this final project.

Declaration of Academic Integrity

The candidate confirms that the work submitted is his/her own, except where work which has formed part of jointly-authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated in the report. The candidate confirms that appropriate credit has been given within the report where reference has been made to the work of others.

This copy has been supplied on the understanding that no quotation from the report may be published without proper acknowledgement. The candidate, however, confirms his/her consent to the University of Leeds copying and distributing all or part of this work in any forms and using third parties, who might be outside the University, to monitor breaches of regulations, to verify whether this work contains plagiarised material, and for quality assurance purposes.

The candidate confirms that the details of any mitigating circumstances have been submitted to the Student Support Office at the School of Electronic and Electrical Engineering, at the University of Leeds.

Name: Ben David Lancaster
Date: April 15, 2019

Table of Contents

[1] [2] [3]

References

[1] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang *et al.*, "Openpiton: An open source manycore research framework," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2. ACM, 2016, pp. 217–232.

[2] N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore gpus," in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–10.

[3] S. Binet, P. Calafiura, S. Snyder, W. Wiedenmann, and F. Winklmeier, "Harnessing multicores: Strategies and implementations in atlas," in *Journal of Physics: Conference Series*, vol. 219, no. 4. IOP Publishing, 2010, p. 042002.

Chapter 1

Introduction

1.1 Why Multi-core?

1.2 Why RISC?

1.3 Why FPGA?

Chapter 2

Background

2.1 Single core vs. Multi-core vs. Many-core

2.2 Network-on-chip

2.2.1 OpenPiton

2.3 Summary

Chapter 3

Project Overview

3.1 Project Deliverables

The project's deliverables are split into two sections: core deliverables (CD) – each deliverable must be satisfied for the project to be a minimum viable product (MVP), and extended deliverables (ED) – deliverables that are not required for a MVP – features that only improve upon an existing feature.

3.1.1 Core Deliverables (CD)

The project's core deliverables are described below.

CD1 Design a compact 16-bit RISC instruction set architecture.

CD2 Design and implement a Verilog RISC core that implements the ISA in **CD1**.

CD3 Design and implement an on-chip interconnect for multi-core processing (2 to 32 cores) using the RISC core from **CD2**.

CD4 Analyse performance of serial and parallel software algorithms, such as parallel DFT [**?**], on the processor.

CD5 Allow the RISC core to be easily compiled to multiple FPGA vendors (Xilinx, Altera).

3.1.2 Extended Deliverables (ED)

The project's extended deliverables are described below.

ED1 Design a RISC core with an instructions-per-clock (IPC) rating of at least 1.0 (a single-cycle CPU).

ED2 Design a RISC core with a pipe-lined data path to increase the design's clock speed.

ED3 Design a scalable multi-core interconnect supporting arbitrary (more than 32) RISC core instances (manycore) using Network-on-Chip (NoC) architecture.

ED4 Design a compiler-backend for the PRCO304 [**?**] compiler to support the ISA from1 **CD1**. This will make it easier to build complex multi-core software for the processor.

ED5 The RISC core can communicate to peripherals via a memory-mapped addresses using the Wishbone [**?**] bus.

ED6 Implement various memory-mapped peripherals such as UART, GPIO, LCD, to aid visual representation of the processor during the demonstration viva.

ED7 Store instruction memory in SPI flash.

ED8 Reprogram instruction memory at runtime from host computer.

ED9 Processor external debugger using host-processor link.

3.2 Project Timeline

3.2.1 Project Stages

The project is split up into many stages to aid planning and management of the project. There are 8 unique stage areas: 1. Inital project conception; 2 Basic RISC core development; 3. Extended RISC core development; 4. Multi-core development; 5. Processor quality-of-life (QoL) improvements; 6. Compiler development; 7. Demo preparation, and 8. Final report.

The project stages are shown in Table 4.1.

3.2.2 Timeline

The project stages from Table 4.1 are displayed below in a Gantt chart.

Stage	Title	Start Date	Days	Core	Applicable Deliverables
1.0	Research	Feb 04	7	x	
1.1	Requirement gathering/review	Feb 11	14	x	
1.1	Processor specification, architecture, ISA	Feb 18	100	x	**CD1**
1.2	Stage/Time Allocation Planning	Feb 25	7	x	
2.1	Decoder, Register Set, impl & integration	Feb 25	14	x	**CD2**
2.2	Register set impl & integration	Mar 04	14	x	**CD2**
2.3	Local memory impl & integration	Mar 11	14	x	**CD2**
3.1	Memory mapped register layout & impl	Apr 01	21		**ED5**
3.2	Wishbone peripheral bus connected to MMU	Apr 08	21		**ED5**
3.3	Pipelined implementation and verification	Apr 15	21		**ED2**
3.4	Cache memory design & impl	Apr 22	28		**ED2**
4.1	Multi-core communication interface	TBD	TBD	x	**CD3**
4.2	Shared-memory controller	TBD	TBD	x	**CD3**
4.3	Scalable multi-core interface (10s of cores)	TBD	TBD	x	**CD3**
4.4	Multi-core example program (reduction)	TBD	TBD	x	**CD4**
5.1	SPI-FPGA interface for OTG programming	TBD	TBD		**ED7**
5.2	FPGA-PC interfacing	TBD	TBD		**ED9**
5.3	FPGA-PC debugging (instruction breakpoints)	TBD	TBD		**ED9**
6.1	Compiler backend for vmicro16	TBD	TBD		**ED4**
6.2	Compiler support for multi-core codegen	TBD	TBD		**ED4**
7.1	Wishbone peripherals for demo	TBD	TBD	x	**CD4**
8.1	Final Report	TBD	TBD	x	

Table 3.1: Project stages throughout the life cycle of the project.

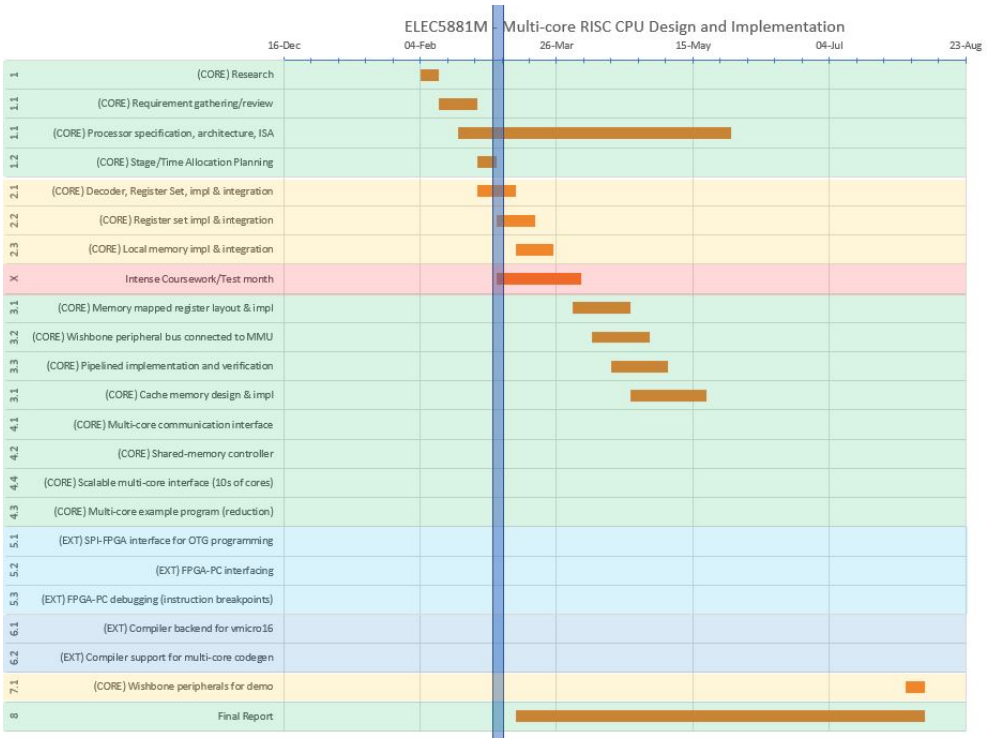

Figure 3.1: Project stages in a Gantt chart.

3.3 Resources

This section describes the hardware and software resources required to design and implement the project.

3.3.1 Hardware Resources

Core deliverable CD5 requires the designed RISC core to be implemented and demonstrated on multiple FPGA devices. Although my design should synthesise for physical IC implementation, due to high costs and length production times, it is not a primary development target.

Due to having past experience with Xilinx FPGAs from my placement work and experience with Altera from university modules it was decided to target the Xilinx Spartan 6 XC6SLX9 and the Altera Cyclone V.

Terasic DE1-SoC Development Board

The Terasic DE1-SoC development board features a large Cyclone V FPGA and many peripherals, such as seven-segment displays, 64 MB SDRAM, ADCs, and buttons and switches, which will aid demonstration of the project. The development board is available through the university so the cost is negligible. Figure 3.3 shows the peripherals (green) available to the FPGA.

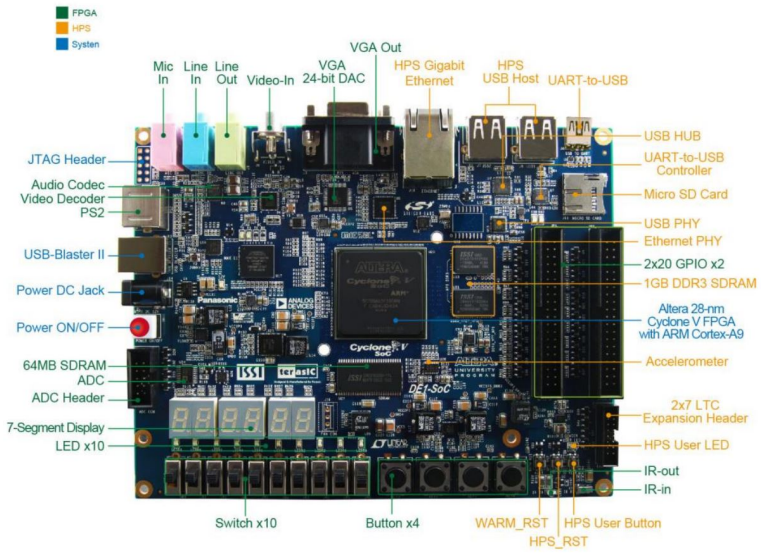

Figure 3.2: Terasic DE1-SoC development board featuring the Altera Cyclone V FPGA and many peripherals. Image source: [**?**].

Minispartan 6+ FPGA Development Board

The Minispartan 6+ is a hobbyist FGPA development board with fewer peripherals than the DE1-SoC. The board's simplicity will ease debugging. The board features a Xilinx Spartan 6 XC6LX9

Testing testing

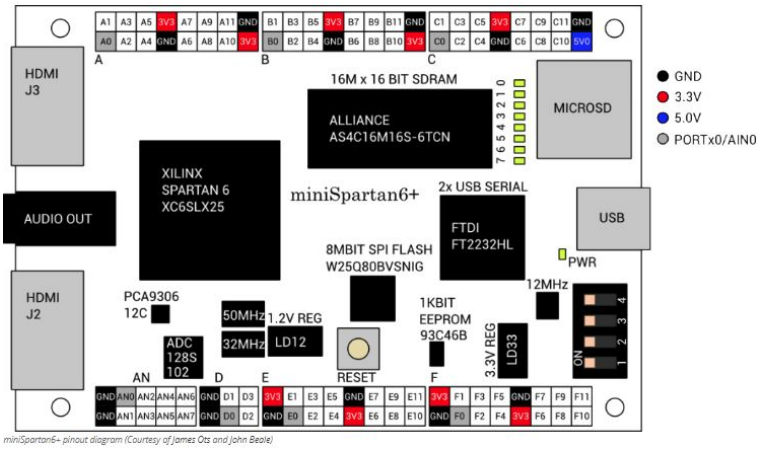

Figure 3.3: Minispartan-6+ development board featuring the Xilinx Spartan 6 XC6SLX9. Image source: [**?**].

3.3.2 Software Resources

Xilinx ISE

Intel Quartus

3.4 Legal and Ethical Considerations

The RISC core is designed to be used as an academic research and educational tool to aid learning and understanding of RISC and multi-core machines.

Chapter 4

Current Progress

This chapter discusses the current progress made towards the project, including designs, implementation, and current results.

4.1 RISC Core

Following the project time line described in section 3.2, the first couple months have been dedicated to the design and implementation of the instruction set architecture and RISC core with stages 1-3. Good progress has been made in both deliverables, the ISA and the RISC core, and the progress is on-time with the initial project time line.

4.1.1 Instruction Set Architecture

A 16-bit instruction set architecture (ISA) has been designed using an iterative approach. There currently exists 32 unique instructions covering most generic RISC operations (add, load/store, branch, compare, etc.) and atleast 16 opcodes available to be provide multi-core communication and functionality. This number should be adequate to support these features when the work begins on the multi-core project stages (stages 4-7).

Design Goals

Having past experience designing and implementing ISAs for previous projects, I wanted to use that knowledge to design an even more efficient and compact instruction set that could provide much greater functionality. The technical design goals of the ISA are described below:

ISA1 Use a fixed width of 16-bits for all instructions.
This will significantly reduce RTL resources and encourage efficiency by not wasting spare bits. In addition, many SPI flash and RAMs support 16-bit wide data reads which will allow each instruction fetch to only require one clock cycle, thus increasing processor performance.

ISA2 Be able to select at least two registers for common instructions.
This will reduce the number of required instructions to manipulate register data. A disadvantage of using two instead of three reigster selects is that instructions are always destructive – they always *destroy* existing data in the destination register (e.g. R0 = ADD R0 R1) unlike constructive instructions that provide a unique register select for the destination (e.g. R2 = ADD R0 R1).

ISA3 Reduce bit-space for frequently used instructions (MOV, MOVI, ADD).
Due to the 16-bit limit, two register selects, and immediate values, the opcode bits are reduced resulting in fewer unique instructions. To overcome this constraint, spare bits in other instructions will be appended to the opcode bits to extend the opcode range. This however, will require a more complex decoder that must first switch the opcode, then switch any spare bits to determine the final opcode. This method will significantly increase the number of unique instructions provided by the instruction set.

ISA4 Provide frequently used actions as options for existing instructions.
In software, frequently used actions include incrementing/decrementing by 1 and performing logical comparisons which usually take more than one instruction on some RISC architectures. As they are common actions, the instruction overhead and time may be significant and can affect performance. To provide a solution to this problem, in addition to using spare bits to extend the opcode range, spare bits will be used to signify a frequently used action action to be performed by the ALU.

As shown in Figure 4.1, frequently used commands such as incrementing/decrementing and logical comparions are provided by setting spare bits to special values. For example, the instructions `ARITH_UADDI` and `ARITH_SSUBI` extend the `ARITH_U` and `ARITH_S` opcodes by filling the spare bit, 4. If this bit is not set (0), the instruction allows for a 4-bit immediate value to be added in addition to the two register selects. The 4-bit immediate allows adding a small number to the ALU which is useful in the case of software for loops where an increment/decrement of more than 1 is required.

Another example is the `SETC` instruction. Inspired by Intel's x86 `SETCC`, the instructions sets the destination register to zero or one depending on the result of the `CMP` instruction's flags. Without this instruction, multiple branches would be required to convert the comparion's flags to logical zeros and ones.

ISA5 Provide instructions for performing bitwise manipulations.
RISC processors are commonly used for microprocessing and microcontroller actions which typically includes bit manipulation. The ISA provides bitwise OR, XOR, AND, NOT, and shifting instructions under a single opcode to fill this need.

ISA6 Provide instructions for explicitly performing signed and unsigned arithmetic.
Performing signed and unsigned arithmetic is a key requirement for RISC applications and so it was decided to provide such instructions. Software programmers can easily switch between signed and unsigned arithmetic by setting bit 11 in the `ARITH` instruction family. Being able to change between signed and unsigned arithmetic instructions by changing a single bit will make the RISC processor's decoder module smaller and less complex.

Without explicit unsigned and signed instructions, extra instructions would be required to perform addition and subtraction. In addition, due to two's complement representation of signed numbers, the highest immediate operand value would be halved, resulting in more instructions to reach the desired value.

	15-11	10-8	7-5	4-0	rd ra simm5
	15-11	10-8	7-0		rd imm8
	15-11	10-0			nop
	15	14:12	11:0		extended immediate
NOP	00000	X			
LW	00001	Rd	Ra	s5	Rd <= RAM[Ra+s5]
SW	00010	Rd	Ra	s5	RAM[Ra+s5] <= Rd
BIT	**00011**	**Rd**	**Ra**	**s5**	**bitwise operations**
BIT_OR	00011	Rd	Ra	00000	Rd <= Rd | Ra
BIT_XOR	00011	Rd	Ra	00001	Rd <= Rd ^ Ra
BIT_AND	00011	Rd	Ra	00010	Rd <= Rd & Ra
BIT_NOT	00011	Rd	Ra	00011	Rd <= ~Ra
BIT_LSHFT	00011	Rd	Ra	00100	Rd <= Rd << Ra
BIT_RSHFT	00011	Rd	Ra	00101	Rd <= Rd >> Ra
MOV	00100	Rd	Ra	X	Rd <= Ra
MOVI	00101	Rd	i8		Rd <= i8
ARITH_U	**00110**	**Rd**	**Ra**	**s5**	**unsigned arithmetic**
ARITH_UADD	00110	Rd	Ra	11111	Rd <= uRd + uRa
ARITH_USUB	00110	Rd	Ra	10000	Rd <= uRd - uRa
ARITH_UADDI	00110	Rd	Ra	0AAAA	Rd <= uRd + Ra + AAAA
ARITH_S	**00111**	**Rd**	**Ra**	**s5**	**signed arithmetic**
ARITH_SADD	00111	Rd	Ra	11111	Rd <= sRd + sRa
ARITH_SSUB	00111	Rd	Ra	10000	Rd <= sRd - sRa
ARITH_SSUBI	00111	Rd	Ra	0AAAA	Rd <= sRd - sRa + AAAA
BR	**01000**	**Rd**	**i8**		**conditional branch**
BR_U	01000	Rd	0000 0000		Any
BR_E	01000	Rd	0000 0001		Z=1
BR_NE	01000	Rd	0000 0010		Z=0
BR_G	01000	Rd	0000 0011		Z=0 and S=O
BR_GE	01000	Rd	0000 0100		S=O
BR_L	01000	Rd	0000 0101		S != O
BR_LE	01000	Rd	0000 0110		Z=1 or (S != O)
BR_S	01000	Rd	0000 0111		S=1
BR_NS	01000	Rd	0000 1000		S=0
CMP	01001	Rd	Ra	X	SZO <= CMP(Rd, Ra)
SETC	01010	Rd	Ra	X	Rd <= Imm8 == SZO ? 1 : 0
MOVI_LARGE	1	Rd	i12		Rd <= i12

Figure 4.1: Initial Vmicro16 16-bit instruction set architecture. Coloured regions represent instruction families (bitwise, branching, arithmetic, etc.).

The ISA table is shown in Figure 4.1. The top 5 bits (15-11) are dedicated to the opcode resulting in 32 unique values. Currently only the bits 14-11 are used (`NOP` to `SETC`) leaving the top bit spare. Initially, this bit was reserved to indicate an extended immediate instruction, `MOVI12`, supporting a large 12-bit immediate value, however later in the design it was decided that the top bit would indicate special instructions dedicated for multi-core operation. This leaves 16 spare unique opcodes for this purpose.

4.1.2 Design and Implementation

The RISC core design is a traditional 5-stage processor (fetch, decode, execute, memory, write-back).

To satisfy **CD5**, the Verilog code will be self-contained in a single file. This reduces the hierarchical complexity and eases cross-vendor project set-up as only a single file is required to be included. A disadvantage with this single file approach is that some external Verilog verification tools that I plan to use, such as Verilator, do not currently support multiple Verilog modules (due to an unfixed bug) within a single file.

Instruction and Data Memory

The design uses separate instruction and data memories similar to a Harvard architecture computer. This architecture was chosen due to it's simplicity to implement.

Register File

To support design goal **ISA2**, the register set features a dual-port read and single-port write. This allows instructions to read 2 registers simultaneously for any instruction. The single-port write allows the instruction output to be written to the register file.

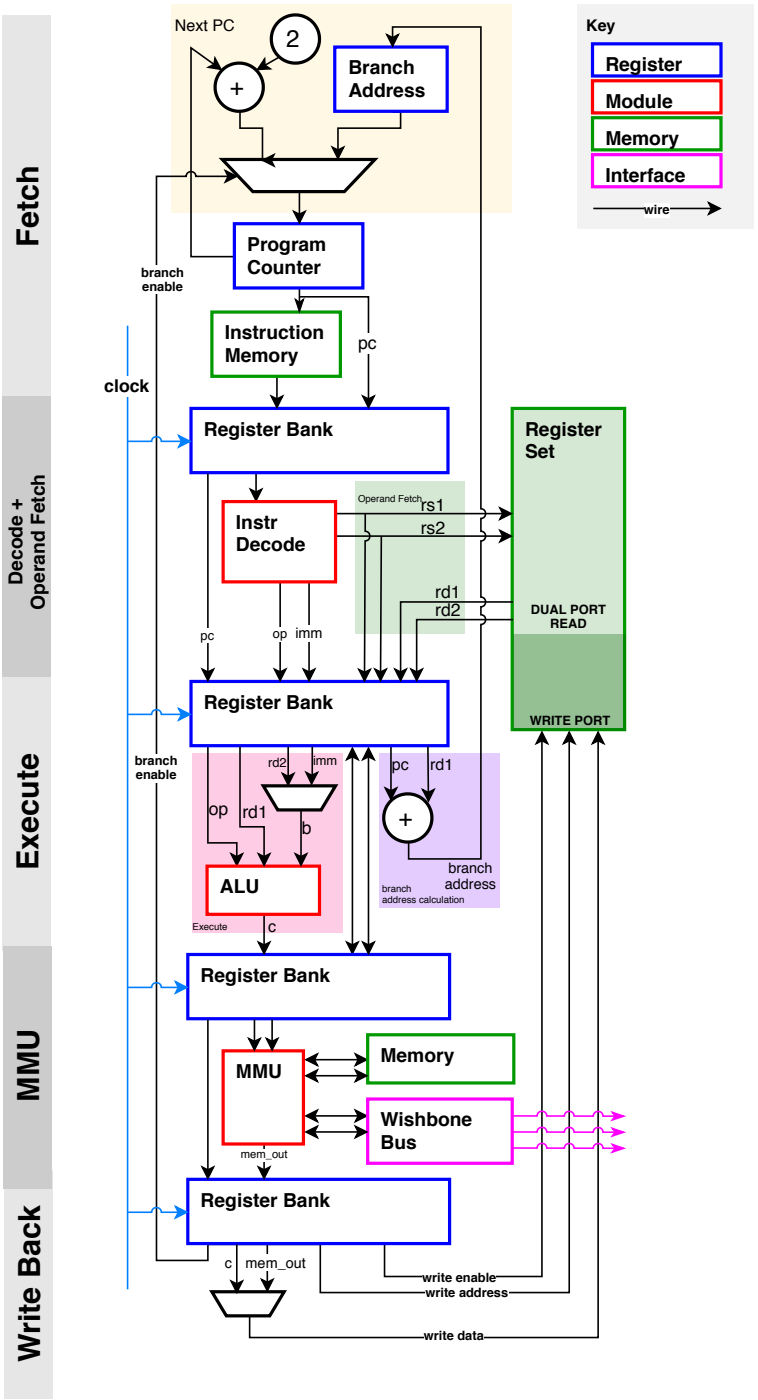

Figure 4.2: Vmicro16 RISC 5-stage RTL diagram.

Pipelining

The extended deliverable **ED1**, to provide atleast 1 instructions per clock

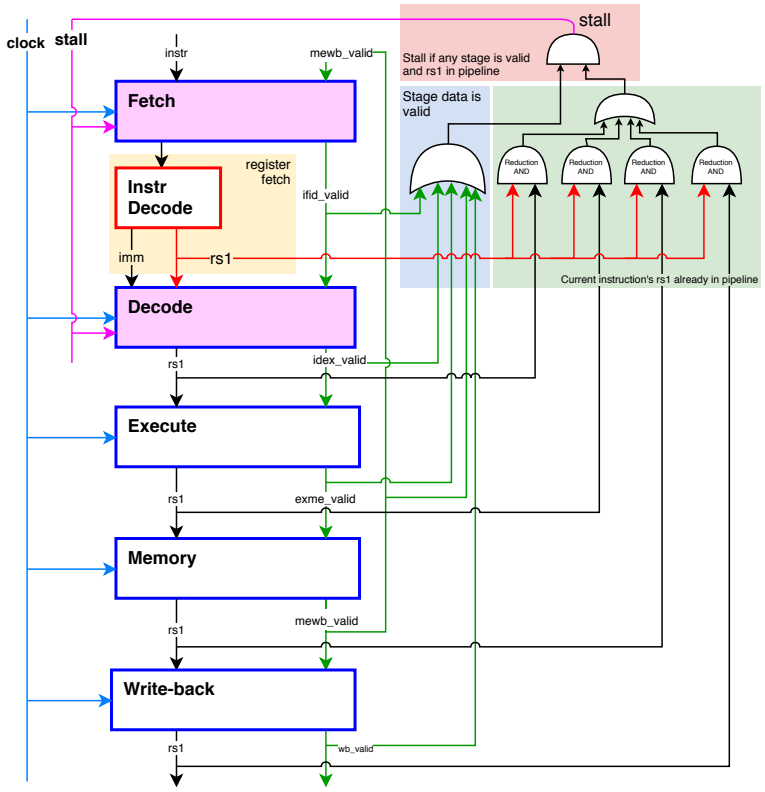

Figure 4.3: Pipeline stall detection logic.

Memory Management Unit

It was decided to use a memory management unit (MMU) to make it easier and extensible to communicate with external peripherals or additional registers. This method would transparently use the existing `LW/SW` instructions which removes the requirement for a unique instruction for each peripheral.

Proposed Memory Mapped Addresses

The proposed memory mapped adresses for each system and peripheral are listed below.

Address (16-bit aligned)	Peripheral Name
`0x0000`	NOP (reads returns 0, writes do nothing)
`0x00ZZ`	Per-core scratch RAM (ZZ = 8-bit RAM address)
`0x0100`	Extended Core Registers 1
`0x0200`	Extended Core Registers 2
`0x03ZZ`	Wishbone Master controller select (ZZ contains 8-bit wishbone slave address)
`0x1XYZ`	Master core controller (X = slave select, Y = instruction, Z = data)

Table 4.1: Project stages throughout the life cycle of the project.

ALU Design

The Vmicro16's ALU is an asynchronous module that has 3 inputs: data a; data b; and opcode op, and outputs data value c. The ALU is able to operand on both register data (`rd1` and `rd2`) and immediate values. A switch is used to set the `b` input to either the `rd2` or `imm` value from the previous stage.

Currently, the ALU does not store flags to indicate overflow, equality, or zero values in the module itself. Instead the ALU outputs the result of the `CMP`, which calculates such flags, to be written back to the register set in the write-back stage. This means that in order to perform a conditional operation, such as a branch, the register containing the `CMP` flags must be included in the instruction.

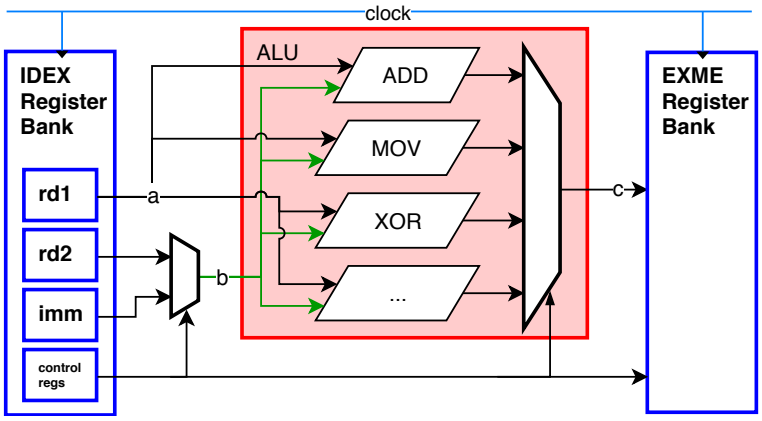

Figure 4.4: Vmicro16 ALU diagram showing clocked inputs from the previous IDEX stage being

The Verilog implementation of the ALU is shown in Figure 4.5. The ALU's asynchronous output is clocked with other registers, such as destination register `rs1` and other control signals, in the `EXME` register bank.

```
322         always @(*) case (op)
323                 // branch/nop, output nothing
324                 `VMICRO16_ALU_BR,
325                 `VMICRO16_ALU_NOP:          c = 0;
326                 // load/store addresses (use value in rd2)
327                 `VMICRO16_ALU_LW,
328                 `VMICRO16_ALU_SW:           c = b;
329                 // bitwise operations
330                 `VMICRO16_ALU_BIT_OR:       c = a | b;
331                 `VMICRO16_ALU_BIT_XOR:      c = a ^ b;
332                 `VMICRO16_ALU_BIT_AND:      c = a & b;
333                 `VMICRO16_ALU_BIT_NOT:      c = ~(b);
334                 `VMICRO16_ALU_BIT_LSHFT:    c = a << b;
335                 `VMICRO16_ALU_BIT_RSHFT:    c = a >> b;
```

Figure 4.5: Vmicro16's ALU implementation named vmicro16_alu. vmicro16.v

Decoder Design

Instruction decoding occurs in the between the IFID and IDEX stages. The decoder extracts register selects and operands from the input instruction. The decoder outputs are asynchronous which allows the register selects to be passed to the register set and register data

to be read asynchronously. The register selects and register read data is then clocked into the IDEX register bank.

```
224         always @(*) case (opcode)
225                 `VMICRO16_OP_HALT, // TODO: stop ifid
226                 `VMICRO16_OP_NOP:             alu_op = `VMICRO16_ALU_NOP;
227
228                 `VMICRO16_OP_LW:              alu_op = `VMICRO16_ALU_LW;
229                 `VMICRO16_OP_SW:              alu_op = `VMICRO16_ALU_SW;
230
231                 `VMICRO16_OP_MOV:             alu_op = `VMICRO16_ALU_MOV;
232                 `VMICRO16_OP_MOVI:            alu_op = `VMICRO16_ALU_MOVI;
233                 `VMICRO16_OP_MOVI_L:          alu_op = `VMICRO16_ALU_MOVI_L;
234
235                 `VMICRO16_OP_BR:              alu_op = `VMICRO16_ALU_BR;
236
237                 `VMICRO16_OP_BIT:         casez (simm5)
238                         `VMICRO16_OP_BIT_OR:      alu_op = `VMICRO16_ALU_BIT_OR;
239                         `VMICRO16_OP_BIT_XOR:     alu_op = `VMICRO16_ALU_BIT_XOR;
240                         `VMICRO16_OP_BIT_AND:     alu_op = `VMICRO16_ALU_BIT_AND;
241                         `VMICRO16_OP_BIT_NOT:     alu_op = `VMICRO16_ALU_BIT_NOT;
242                         `VMICRO16_OP_BIT_LSHFT:   alu_op = `VMICRO16_ALU_BIT_LSHFT;
243                         `VMICRO16_OP_BIT_RSHFT:   alu_op = `VMICRO16_ALU_BIT_RSHFT;
244                         default:                  alu_op = `VMICRO16_ALU_BAD; endcase
245
```

Figure 4.6: Vmicro16's decoder module code showing nested bit switches to determine the intended opcode. vmicro16.v

In Figure 4.6, it can be seen that the first 8 opcode cases are represented using the same 15-11 bits, however the `VMICRO16_OP_BIT` instructions require another bit range to be compared to determine the output opcode.

4.1.3 Verification

Currently, the only verification method used is manual inspection of the output waveforms of a test bench.

Known Bugs

Several known bugs exist within the RISC core however none are critical as they can be easily avoided in software.

BUG1 Stall detection does not consider load/store instructions.
Due to pipelining techniques used by the processor and lack of address checking in the `EXME` and `MEWB` stages, `LW` instructions immediately after `SW` instructions:

```
SW R0 (R2+16)
LW R1 (R2+16)
```

will not return the previously stored value. In addition, because of the target address is calculated by the ALU (e.g. `R2+16`), detecting matching addresses at IFID and IDEX stage is not trivial, and because of this, a hardware fix is not planned for the final version. It is possible to overcome this problem in software by placing at least 5 `NOP` instructions after each `SW`.

Chapter 5

Future Progress

5.1 Multi-core Functionality

5.2 Interconnect Goals

5.3 Interconnect Layout

5.3.1 Bus Design

5.3.2 Core Pin Assignments

5.4 Core-to-core Communication

5.5 Shared-Resource Control

5.5.1 Resource Scheduling

5.6 Verification

5.7 Conclusion