TAMPERE UNIVERSITY OF TECHNOLOGY

JILL STEELE

INSTRUCTION DECODER DESIGN FOR AN EMBEDDED PROCESSOR

Master's thesis

Examiner: Jari Nurmi
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 3 April 2013.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
STEELE, JILL : Instruction Decoder Design for an Embedded Processor
Master of Science Thesis, 50 pages, 8 Appendix pages
December 2013
Major: Digital and Computer Electronics
Examiner: Jari Nurmi
Keywords: RISC processor, decoder, digital design

This thesis describes the integration of a RISC core processor with the MIPS assembly language. The COFFEE Core is a RISC core processor developed at Tampere University of Technology. The compiler and tools, based on GCC and GNU Binutils, are several versions behind the current releases. To become more widely adopted in research and industry, the processor would need to use up-to-date industry standard tools. Modifying the processor to use the MIPS or ARM instruction set would allow the associated tools to be used as well. The COFFEE architecture is compared with both the MIPS and ARM architectures to determine which architecture would provide the most benefits to developers and how the COFFEE Core might be adapted to meet the architectural requirements. When compared with the COFFEE instruction set, the ARM instruction set is found to have an overlap of 7 identical instructions and 32 similar instructions and the MIPS instruction set is found to have an overlap of 22 identical instructions and 54 similar instructions. MIPS and ARM were also found to be comparably beneficial to developers. After these comparisons, the MIPS architecture was selected as the most compatible, due to the larger overlap in the instruction set compared to ARM. A subset of overlapping MIPS instructions was chosen to be mapped to the corresponding COFFEE instructions. The Decoder and Control Unit of the COFFEE Core was modified and the processor was tested with MIPS assembly, finding the implemented instructions to be functional. The integration of MIPS with the COFFEE Core is therefore shown to be feasible. Additional modifications outside the Decoder and Control Unit of the COFFEE Core would be required to implement the remaining MIPS instructions.

PREFACE

The research presented in this master's thesis was completed as part of the COFFEE Project at Tampere University of Technology under the supervision of Dr. Jari Nurmi. The thesis topic represents my interest in embedded technology, which is quickly becoming ubiquitous in our everyday lives.

I would like to thank Dr. Jari Nurmi for providing the opportunity to work with his team throughout the years and for his continued support and guidance. A big thank you to Roberto Airoldi for his advice throughout the thesis process. Thank you also to Juha Kylliäinen for answering many questions about the design of the COFFEE Core and to Guoqing Zhang for his assistance with the COFFEE and MIPS compilers. A warm thank you to the members of Team Nurmi for providing a welcoming lab environment.

I would like to thank my friends in Tampere, Vancouver, and abroad for their continued friendship and encouragement. Finally, I would like to thank my family for their love and support, and for encouraging me to pursue my goals and dreams however far away it takes me.

Jill Steele
1 November 2013
Tampere, Finland

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND DEFINITIONS

APSR	Application Program Status Register
Binutils	Binary Utilities
C	Carry Flag
CCU	Central Control Unit (COFFEE)
CISC	Complex Instruction Set Computing
COFFEE	COFFEE Project at Tampere University of Technology
CPSR	Current Program Status Register (MIPS)
CR	Condition Register (COFFEE)
GCC	GNU Compiler Collection
GNU	Unix-like operating system
GPR	General Purpose Register
LR	Link Register
N	Negative Flag
PC	Program Counter
PSR	Processor Status Register
RISC	Reduced Instruction Set Computing
SPSR	Saved Processor Status Register
TLB	Translation Look-aside Buffer
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
Z	Zero Flag

1. INTRODUCTION

The reduced instruction set computing (RISC) design philosophy was originally developed to increase the timing performance of processors. Complex instruction set computing (CISC) processors had previously been developed while focusing on reducing the semantics gap when compiling high level programming languages to computer language. This resulted in processors with complex instructions which were not all that quick to execute. With the focus then shifting to processor performance, the RISC computer design philosophy was developed [22] [8].

Early RISC processors had the following design choices in common [22] [16].

- Memory is only accessed by loading and storing to registers
- Operations are register to register
- Addressing modes are simplified
- Operations are simplified
- Instruction formats are simplified and of fixed length
- One instruction is completed per clock cycle

As RISC processor cores are highly simplified, software takes a significant role in optimizing the compilation of high level languages to machine language [8].

RISC core processors are now widely used in embedded processing and companies such as ARM, MIPS, and Atmel produce RISC core processors [6] [21] [15]. Processor cores can be purchased either as part of a physical product or as a licence to use the architecture.

The COFFEE Project at Tampere University of Technology has developed a RISC soft-core processor called the COFFEE Core with supporting software [14]. A soft-core is synthesizable and can be run on an FPGA, allowing for easy modification of the implementation and distribution of the hardware. The current plans are to offer the core for free under an open-source licence [16]. If the COFFEE Core

were widely distributed, the software would need frequent updating to maintain industry standard. As this is rather labour intensive, it would seem logical to prefer to use software from an established RISC architecture. Thus, a COFFEE Core taking in an industry standard instruction set such as from ARM or MIPS would be beneficial. An added benefit to minimized labour would be the use of additional software tools, libraries, and technology associated with the new architecture. This thesis will investigate the feasibility of adapting the COFFEE Core to read an industry standard architecture.

1.1 Background

The COFFEE project is a project at Tampere University of Technology. The COFFEE Core is a RISC Core developed for the COFFEE project.

The current COFFEE Core uses an instruction set and software tools developed specifically for the COFFEE Core [14]. The tools take advantage of COFFEE's unique features that differentiate it from other commercial cores. The software tools, although customized, are based on the open-source GCC tool-chain. A new updated GCC tool-chain is released about once a year [1], meaning that the COFFEE tools must also be updated. As updating the tool-chain takes additional resources, it is not always possible to update the tool-chain in a timely manner, and therefore the current COFFEE tool-chain is out of date compared to the current version of the GCC tool-chain.

The current COFFEE tool-chain is based on GCC version 3.44. The most updated version of the GCC tool-chain is GCC version 4.8.0, which was released on March 22, 2013 [1], and so the COFFEE tool-chain does not have any of the features added since the version 3.44 release.

In addition to the COFFEE compiler, a set of binary utilities, or Binutils, are also provided for COFFEE. The current version of the COFFEE Binutils is based on GNU Binutils version 2.17. The most up-to-date version of GNU Binutils as of the writing of this thesis is version 2.23.1 [2], meaning that the COFFEE Binutils is also out of date and possibly missing new features.

COFFEE also does not have software tools comparable to the tools provided by many commercial cores, such as software tools provided by MIPS or ARM and compatible tools by third party vendors for development and debugging. Associated developer tools are also provided for both cores by the MIPS and ARM companies and by third party vendors [17] [7]. MIPS and ARM processors are commonly used in industry.

1.2 Problem Definition

The difficulty faced is then how to keep the existing COFFEE compiler and software tools up to date while also providing an industry standard development environment for the COFFEE Core. This thesis will evaluate which industry standard development environment is the most beneficial to the developer, and then propose an approach to interface the COFFEE core with that environment.

If the COFFEE Core were interfaced with an industry standard development environment, it will become more useful because tools, libraries, and operating systems associated with the chosen environment can then be used with the COFFEE core.

1.3 Work Description

The objectives of this work are as follows:

1. To propose an approach to integrate the COFFEE Core with an industry standard development environment while eliminating the need to update the COFFEE compiler and tools
2. To integrate the COFFEE Core with the development environment

The approach followed to achieve the above objectives are as follows:

- Perform a review on MIPS, ARM and COFFEE tools and architectures
- Compare tools and architectures of MIPS and ARM with COFFEE to determine which architecture, when interfaced with COFFEE, offers the most benefits to the developer
- Propose an approach for integrating COFFEE with the chosen architecture
- Implement and demonstrate, as proof of concept, the integration of COFFEE with the chosen architecture

1.4 Scope

The integration of COFFEE with the chosen architecture will be done at the hardware level rather than the software level. This will require the COFFEE processor to understand machine-level instructions from the software tools of the chosen architecture.

It is expected that the integration will require significant modification of the COFFEE Core. Thus, for the first iteration of the integration the modifications will be limited to only the decoding and control processes of the COFFEE Core. These parts of the processor determine the incoming instruction and what corresponding controls need to be set inside the processor. The reasoning for this limitation is two-fold.

- To simplify the problem and the chosen approach
- To limit the modifications made to the COFFEE Core by only modifying what is necessary

This simplification allows the thesis to focus on overlaps and similarities between the two architectures and what approach needs to be taken for a minimally functioning integrated processor. Since the purpose of this thesis is to integrate COFFEE with another architecture, rather than to write a new processor, most of the original architecture will be retained and modifications are made only when necessary.

1.5 Thesis Outline

This thesis will first begin with a theoretical background describing the fundamentals behind the RISC design philosophy. The RISC design has allowed for improved timing performance in comparison to CISC architectures.

The COFFEE Core is introduced as part of a project at Tampere University of Technology. The basic architecture and instruction set of the COFFEE Core are described. The COFFEE Core is a RISC design, and is thus pipelined with an optimized instruction set. The challenges facing the COFFEE Core in becoming widely adopted in industry is addressed.

As the COFFEE Core would need to be integrated with an industry standard environment, the MIPS and ARM environments are evaluated and the architectures are compared with MIPS to determine which environment provides the most benefit to the developer and to evaluate the similarities with COFFEE. The MIPS environment is chosen for its similarities to the COFFEE Core, wide industry adoption and open-source tools, and developer benefits.

The MIPS and COFFEE designs are then more closely compared and a design is developed to integrate the COFFEE Core with the MIPS environment. The integration will require the COFFEE Core to decode MIPS instructions and direct the correct controls to the rest of the Core. This will involve modification of the COFFEE Core hardware responsible for decoding and control. A small subset of MIPS

instructions deemed to be similar in function to COFFEE instructions are implemented. The challenges in implementing the remaining instructions are discussed.

The integration design is implemented and tested using simple MIPS assembly programs. The implemented MIPS instructions are found to function according to the corresponding COFFEE instruction specifications. Recommendations are made for further work in implementing the remaining instructions and adapting the COFFEE architecture to the MIPS architecture.

2. RISC CORE PROCESSORS

The acronym RISC stands for *reduced instruction set computing* and describes a processor design philosophy with the goal of increasing the timing performance. Before RISC was Complex Instruction Set Computing (CISC) which was focused on the challenge of reducing the semantics gap when compiling high-level computer language to machine language. An instruction set with complex instructions would simplify the compilation and required software, and made sense at a time when software costs were high [22]. As software costs declined, the focus shifted to improving processor speed, and thus RISC was introduced. A reduced and simplified instruction set and a pipelined design executes more quickly, thus improving processor timing. The following design choices were common in the early RISC core processors [22] [16].

- Memory is only accessed by loading and storing to registers and operations are register to register
- Addressing modes are simplified
- Operations are simplified
- Instruction formats are simplified and of fixed length
- One instruction is completed per clock cycle

Memory is only accessed by loading and storing to registers. This reduces memory accesses and cache misses. Operations are then register to register and do not require memory access. This also helps to simplify the instruction set. Addressing modes are simplified allowing for faster processing. Operations are also simplified. Simple operations execute more quickly than complex operations and thus reduce the overall execution time. The operations can also be broken down into parts and executed in steps, ideally in one clock cycle. If more complex operations are necessary, they can be executed in a co-processor or in software. Instruction formats are simplified and of fixed length. A simplified instruction format reduces decode time. Because instruction formats are consistent, determining the instruction and accessing registers can take place in the time clock cycle.

A RISC processor uses a pipelined design. For a brief definition of a pipeline, this means that the design consists of a series of consecutive stages where each stage runs independently and all stages can run simultaneously. It takes one clock cycle for all the stages to run once, if they all run simultaneously. Instructions are broken down into a series of steps. An instruction moves through each of the stages and at each stage a step is completed. The instruction moves to the next stage for the next clock cycle. Since each stage may run independently of every other stage, one instruction can enter the pipeline and one instruction can leave the pipeline at a time in a first-in-first-out (FIFO) like manner. One instruction exits the pipeline in each clock cycle, and thus the design is one instruction per cycle.

In a one instruction per cycle design, the actual latency of the instruction does not matter, since it is broken down into parts. Rather, only the latency of the slowest pipeline stage matters when determining the clock rate of the core. As additional pipeline stages are added, the instructions can be broken into smaller steps resulting in even faster possible clock rates. Thus, the pipelined one-instruction-per-cycle design results in faster processing.

A few complications arise when a pipeline design is used. Data hazards are caused when the same data is needed in two separate places in the pipeline. As the number of stages in a pipeline increases, the likeliness of a hazard occurring also increases. Consider the case of a branch. While a branch is in the pipeline and before the branch condition is evaluated, several instructions may have entered the pipeline. If the branch is taken, the later instructions are then no longer needed, requiring them to be flushed from the pipeline. There is also the case that a later instruction may depend on the outcome of the earlier instruction, such as in a series of arithmetic operations. Usually this conflict can be avoided at compile time by scheduling the instructions to avoid data hazards and inserting stalls or bubbles into the pipeline to avoid a data conflict. Within the pipeline, data forwarding can be used to send data from later stages to earlier stages where the data is needed. If the data is not ready, a stall is introduced until the data is received from the execute stages [16].

Several embedded RISC processors are available on the market from companies such as ARM, MIPS, and Atmel [6] [21] [15]. The COFFEE core from the COFFEE Project at Tampere University of Technology is also a RISC core processor.

3. COFFEE RISC CORE

The Coffee Project is a project at Tampere University of Technology. The project consists of hardware blocks, software tools, and four platforms.

The hardware blocks consist of the COFFEE RISC Core, the MILK co-processor, and the BUTTER co-processor [14]. The COFFEE RISC Core is a harvard-type, soft-core RISC processor [16]. For this thesis, only the COFFEE RISC Core was considered and will be described in more detail in the following sections.

The software tools consist of a compiler and associated binary utilities. The C Compiler is based on GCC v3.4.4 and further supports the MILK FPU (floating point unit), co-processing, and optimization. The binary utilities includes a linker, assembler, and disassembler and is based on GNU Binutils version 2.17 [14].

3.0.1 COFFEE Hardware Architecture

The COFFEE Core is based on a harvard architecture, meaning that the data and instructions are stored and accessed via separate interfaces. The processor was designed according to RISC principles and is a pipelined design executing one instruction per clock cycle [12].

COFFEE as a RISC Core

The COFFEE Core is considered a RISC core because it is designed to use an optimized set of instructions called a RISC Instruction Set Architecture (ISA). In a RISC core, the architecture and instructions are optimized for fast execution, versus the more complex instructions of CISC architectures that may take longer to execute. The faster execution per instruction of RISC ISAs results in increased processor performance.

There are a few common considerations when designing a RISC core, as was addressed in the chapter RISC Core Processors, and many of these are reflected in the design of the COFFEE Core, as outlined in the paper "COFFEE - A Core For Free" [16].

One instruction per cycle COFFEE is a pipelined design, thus resulting in a throughput of one instruction per cycle.

Fixed Instruction Length The COFFEE core accepts both 16 bit and 32 bit instructions. A fixed length instruction simplifies the instruction encoding.

Load and Store Memory Access COFFEE is a load-and-store architecture, meaning that data from memory needs to be loaded into registers before being operated on. This helps to optimize memory accesses and cache misses.

Simplified Address Mode A simplified addressing mode results in faster processing times and thus a faster clock cycle.

Fewer, Simpler Instructions Simpler instructions allow the pipeline to be designed in fewer, quickly executing stages, and thus increasing the pipeline throughput. This results in a faster clock rate, and faster processing. In the RISC core, instructions can be executed as quickly as one clock cycle, or up to three clock cycles in the case of the 32 x 32 bit multiplication.

COFFEE Pipeline Structure

The COFFEE architecture is a pipelined structure as in Figure 3.1. To achieve a fast clock rate, each instruction is broken into separate steps and executed over several clock cycles. The pipeline is designed in stages to execute each step of each instruction. Instructions are then pipelined, meaning that one instruction enters the first stage of the pipeline and one instruction exits the last stage of the pipeline in a FIFO manner for every clock cycle. This gives a throughput of one instruction per cycle, regardless of the latency of an instruction. The COFFEE pipeline consists of mainly a control unit and six pipeline stages: FETCH, DECODE, EXE1 (execute stage 1), EXE2 (execute stage 2), EXE3 (execute stage 3 and memory stage), and WRITE-BACK [16].

FETCH The fetch stage obtains a new instruction from the location indicated by the Program Counter (PC) and increments the PC by four in the 32 bit instruction mode, or the size of one instruction [16].

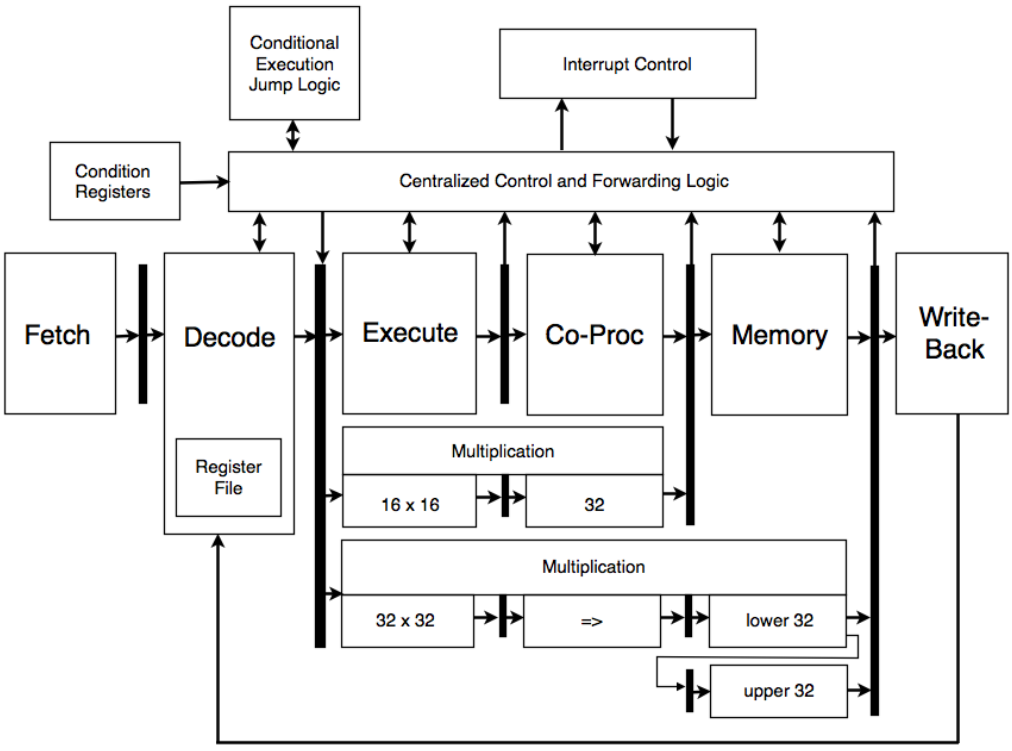

Figure 3.1: COFFEE Pipeline [16]

DECODE The DECODE stage determines the instruction accepted in the FETCH stage. A new processor status register (PSR) is set depending on the incoming instruction, and exception logic is evaluated. Immediate values from immediate instructions are extended to 32 bits. Operands consisting of registers or immediate values are then routed towards either the arithmetic logic unit (ALU) or the co-processor interface contained in the EXE2 stage. Jump and branch instructions are also executed immediately after the decode stage. That is, the offset defined by the instruction is added to the PC [16].

As instructions run through the pipeline, a data dependency may occur where a later instruction depends on the result of an earlier instruction. The COFFEE pipeline forwards the data of an earlier instruction to a later instruction as soon as it is available. However, if the data is not able to be forwarded, a stall will occur in the fetch or decode stages. For example, condition flags Z, N, and C, denoting a zero result, negative result, or a carry bit respectively are updated after an arithmetic operation is completed and are then immediately available to the DECODE stage [16].

EXE1 In the first execute stage, addition, subtraction, byte manipulation, shifting, and boolean operations are performed. Multiplication for both 16 and 32 bit

operands begins in this stage and continues into the co-processor stage, also referred to as EXE2. Multiplication for 16 bits x 16 bits has a two cycle latency and 32 bits x 32 bits has a three cycle latency. Condition flags are evaluated and available for the decode stage [16].

EXE2 The co-processor stage manages access to the co-processors [16]. The multiplications for 16 and 32 bit operands continues into the co-processor stage. The 16 bit x 16 bit multiplication completes in this stages, with a 32 bit result. The 32 bit x 32 bit multiplication continues and completes in the memory stage, called EXE3, with a 64 bit result. Condition flags are evaluated in both the co-processor and memory stages and are then immediately available to the decode stage [16].

EXE3 The memory stage, or EXE3, forwards all results to the WRITE-BACK stage to ensure all instructions with one, two, or three cycle latencies enter the write-back stage at the same time. Load and store instructions access the data memory [16].

WRITE-BACK The write-back stage writes the results of the instruction to the specified destination register in the register file [16].

Core Control Unit

The Core Control Unit (CCU) is described according to the COFFEE Core Control documentation [13] and with reference to the actual VHDL code.

The CCU controls the data as it moves through the processor, and manages the control signals controlling the pipeline stages. A simplified block diagram of the CCU is shown in Figure 3.2. The CCU controls the pipeline as an instruction moves from stage to stage. To simplify the control, the CCU itself also has a sequential stage structure. The flush and enable controls for managing data hazards also affect the control entities. The control unit consists of a master control entity, a flow control entity, and five decoding entities. Each decoding entity is connected to a stage of the pipeline, ensuring the right controls are provided at the right time. The instruction or opcode flows through the pipeline from stage to stage where it is decoded and the appropriate controls are set. Decoding occurs separately at each stage as sending an opcode to each entity is more efficient than sending several signals to indicate the selected instruction.

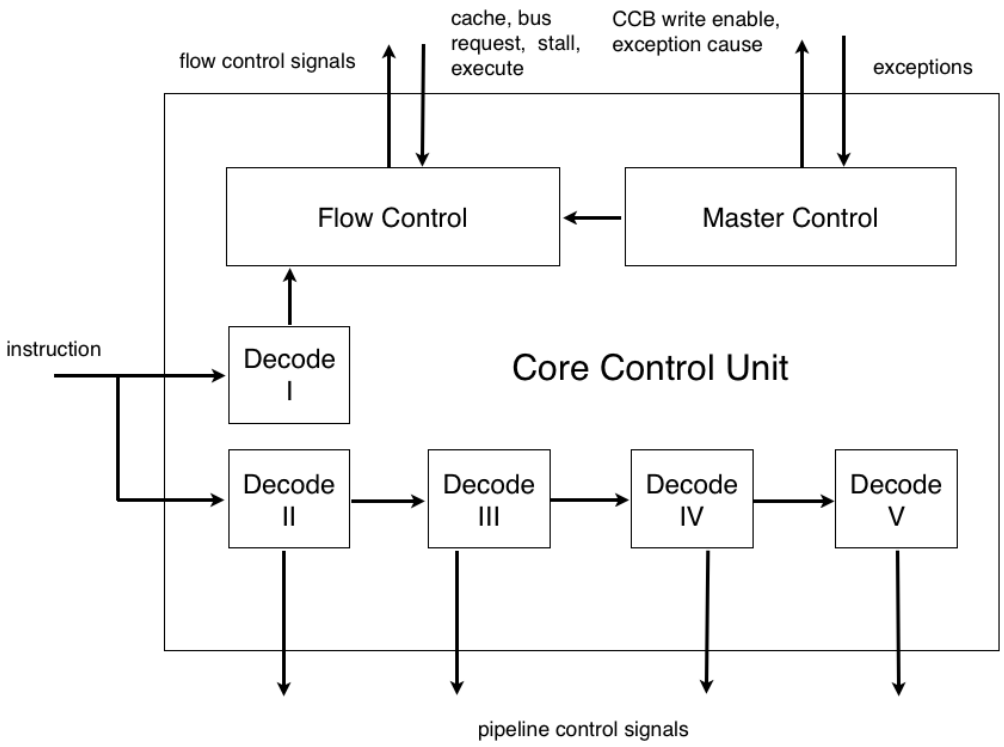

Figure 3.2: Simplified Block Diagram of Core Control Unit [13]

Master Control Entity The Master Control Entity manages context switching and flushing in the case of interrupts and exceptions. As the master control, it can override the PC, status flags, and flush any stage, regardless of the signals set by any other part of the control unit.

Flow Control Entity The Flow Control Entity is responsible for datapath control and manages the movement and timing of data through the pipeline. The entity also manages data, control, and resource hazards, the data bus and co-processor interfaces, context switching, and the hardware control stack.

The following decode units each correspond to a stage or section of the pipeline and determine the corresponding control signals. The decode units form a parallel control pipeline with the instruction decoding occurring in each entity as the instruction moves through the pipeline.

The timing of the decode entities with respect to the instruction in the processor pipeline is shown in the following Figure 3.3. Each decode entity is represented as a labelled box and takes an instruction as input in the corresponding stage. The arrows show to which stage the control outputs are directed. Instructions are moving through each stage of the pipeline and the data is then used when the instruction arrives at that stage. Instructions are decoded at each step by the control unit

decode entities. The first two decode units, CCU DECODE I and CCU DECODE II, take in the instruction currently in the DECODE stage. The CCU DECODE I provides control outputs available immediately for the DECODE stage, whereas CCU DECODE II registers the results for use in the EXE 1 stage in the next clock cycle. When the instruction moves to the EXE 1 stage, the control signals from CCU DECODE II are used as control inputs. The third decode unit, CCU DECODE III, runs as the instruction is in the EXE 1 stage, and registers the outputs for use as control inputs when the instruction moves to the EXE 2 stage. CCU DECODE IV runs when the instruction is in the EXE 2 stage and registers the results for use in EXE 3. CCU DECODE V runs when the instruction is in the EXE 3 stage and registers the results for use in the WRITE-BACK stage.

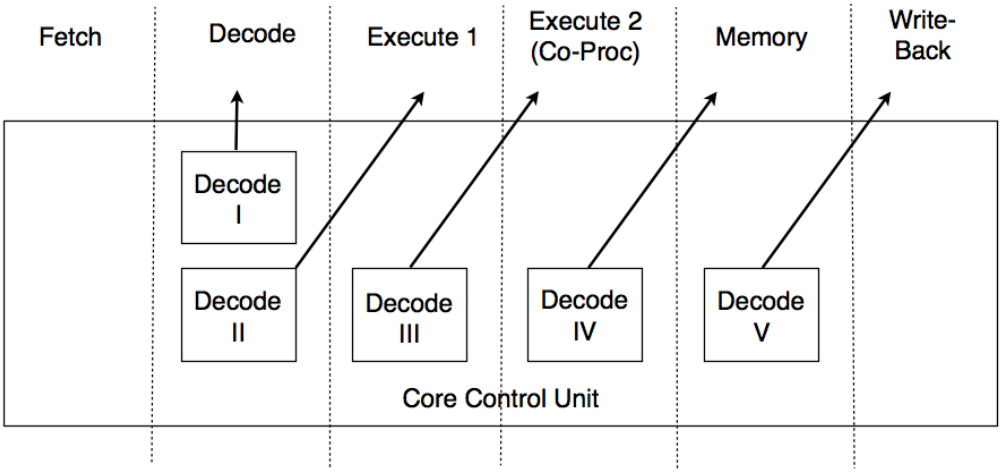

Figure 3.3: Timing of Control Pipeline

CCU DECODE I *This entity runs when the corresponding instruction is currently in the pipeline DECODE stage. The resulting control signals go immediately to the DECODE stage (stage 1).*
The first decode entity provides control signals for the instruction currently in the DECODE stage. The outputs of this entity are non-registered and available immediately in the same cycle. The register file index of the first and second register operands, as well as the index of the target register is extracted from the instruction word. The conditional execution information is also extracted from the instruction word. Decoding then occurs to determine if the instruction is a branch instruction, if one or two operands are needed, how many ALU cycles are needed to complete the instruction, and if the instruction writes to the register file. Controls to the ALU in the form of ALU opcodes are set depending on the instruction and decoded within the ALU. The safe state, which is the pipeline stage at which an instruction will no

longer cause an exception or change the PSR or condition register, is evaluated.

CCU DECODE II *This entity runs when the corresponding instruction is currently in the DECODE stage. The resulting control signals go to the pipeline EXE1 stage (stage 2).*
The second decode entity provides control signals for the first execution stage. The outputs are registered, meaning that the entity runs when the instruction is currently in the DECODE stage and the resulting control signals will be available for the first execution stage, EXE1, in the next clock cycle. The entity manages mapping of instructions and data to co-processor registers. The entity also has flush controls for flushing an instruction in the pipeline.

CCU DECODE III *This entity runs when the corresponding instruction is currently in the EXE1 stage. The resulting control signals go to the pipeline EXE2 stage (stage 3).*
The third decode entity provides control signals for the second execution stage, EXE2. As the outputs of this entity are also registered, the control signals are calculated when the instruction is in EXE1. The entity controls the data going from the second execution stage, EXE2, to the third execution stage, EXE4. Depending on the instruction, the data could come from the ALU after one cycle of execution, a multiplication after two cycles of execution, or the condition register. Signals to the data bus interface also indicate if the next memory access is a read or write. A signal informs the address checker to validate the data address for load and store instructions.

CCU DECODE IV *This entity runs when the corresponding instruction is currently in the EXE2 stage. The resulting control signals go to the EXE3 stage (stage 4).*
The fourth decode entity provides control signals for the third execution stage. As the outputs of this entity are also registered, the control signals are calculated when the instruction is in EXE2. A control signal to the memory interface indicates if a memory load occurs in the next clock cycle. The correct data from either the co-processor interface or the core pipeline is selected to proceed from EXE3 to the RFW stage. The data to be written to the register file is selected from either the pipeline or the memory interface.

CCU DECODE V *This entity runs when the corresponding instruction is currently in the EXE3 stage. The resulting control signals go to the WRITE-BACK*

stage (stage 5).
The fifth decode entity provides control signals for the instruction currently in the write-back stage and selects if the data to be written to the register file is the result of the pipeline or from the memory interface.

3.0.2 COFFEE Instruction Set Architecture

The COFFEE Instruction Set consists of 67 instructions [9]. The instruction set was first designed while considering instruction sets of processors currently on the market, thus resulting in many overlapping instructions [16]. The instruction set contains arithmetic and logic instructions, bit manipulation and shift instructions, load and store instructions, co-processor instructions, conditional branches, and jump instructions [10].

The COFFEE core ISA allows individual instructions to be executed conditionally, thus reducing the need for jumps. Conditional instructions, such as compare instructions, save the resulting flags of the compare to one of eight condition registers in the COFFEE core. The branch instruction then takes the result of the compare from the condition register and branches accordingly [11].

CPU Registers

Before the instructions are reviewed in detail, the registers used in COFFEE are briefly described as they are presented in the COFFEE Core User Manual [12].

The COFFEE register bank consists of two register sets, user registers (SET 1) and superuser registers (SET 2), and eight condition registers (CRs). SET 1 and SET 2 each contain thirty-two registers. SET 1 is used by applications programs and SET 2 is a privileged instruction set used by an operating system and protected from application programs. The registers are allocated as shown in Table 3.1 and consist of general purpose registers (GPRs), a processor status register (PSR), and a saved processor status register (SPSR). Two GPRs may also be optionally used as link registers (LRs).

There are no hardware restrictions as to the type of information that can be stored in a GPR. Register 31 is a combination LR and GPR, meaning that linking information for interrupt and exception returns or jump and link instructions can be stored in register 31. All registers except the PSR are 32 bits in length.

The PSR is 8 bits in length and stores status information including if interrupts are enabled or disabled, if the instruction length is 16 or 32 bits, the selected register

Table 3.1: COFFEE Register Bank [12]

Register Number	Set 1 (User Registers)	Set 2 (Superuser Registers)
0	GPR	GPR
1	GPR	GPR
...	...	...
28	GPR	GPR
29	GPR	PSR
30	GPR	SPSR
31	GPR/LR	GPR/LR

set, and the user mode. The SPSR saves the PSR when the user mode changes.

In addition to the registers in SET 1 and SET 2, the eight condition registers are used to store results from compare instructions or condition flags from arithmetic instructions. Each condition register has a length of 3 bits and consists of three flags, Z (zero flag), N (negative flag), and C (carry flag).

Instructions

The COFFEE ISA, contains 67 instructions of 32 bit length [9]. The instructions are divided into the following categories [10]:

- Integer Arithmetic
- Byte and Bitfield Manipulation
- Boolean Bitwise Operators
- Conditional Jumps (Branches)
- Other Jumps
- Integer Comparison
- Shifts
- Memory load and store and data moving
- Coprocessor instructions
- Mode changing instructions
- Miscellaneous Instructions

Many of the COFFEE instructions have similar instructions in the MIPS and ARM instruction sets. This is especially true for many of the more standard instructions, such as basic arithmetic operations, jumps and branches, and shifts. As COFFEE, MIPS, and ARM are RISC architectures, comparable load and store instructions also exist.

Instruction Format

The COFFEE instructions can contain a number of different fields used to indicate source and destination register, immediate values, or for additional features such as conditional execution. The fields and the functionality they support are described in Table 3.2 adapted from the COFFEE Instruction Encoding document and the encoding of the instructions is described in detail in the document COFFEE Instruction Encoding [9].

Table 3.2: COFFEE Instruction Fields [9]

Field	Size (bits)	Description
cond	3	specifies execution condition
creg	3	specifies one of eight condition registers to either store or read condition flags
cp_sreg/sr	5	specifies co-processor source register
cp_dreg/dr	5	specifies co-processor destination register
dreg	5 or 3	specifies destination register
imm, imm1, imm2	-	an immediate value
imml	-	left part of an immediate value
immr	-	right part of an immediate value
msb	1	most significant bit of a 16 bit immediate
sregi	5 or 3	specifies a source register
cex	1	enables or disables conditional execution
xxx	-	unused bits

The length of some fields, such as for immediate values and for destination and source registers, is variable and the size of the field depends on the particular instruction. Additionally, the encoding of a particular instruction also depends on whether or not conditional execution is used, since when conditional execution is not used there are more available bits in the instruction for other uses, such as a longer immediate value.

There is some consistency in the arrangement of the instruction fields, meaning that the placement of the fields stays the same for different instructions. As an example of the arrangement of fields, consider the ADD instruction for the addition of two values stored in registers in Table 3.3.

Table 3.3: ADD Instruction

opcode (000001)	cex	creg	cond	xxx	sreg2	sreg1	dreg
31...26	25	24...22	21...19	18...15	14...10	9...5	4...0

The instruction contains one destination register in bits 4 to 0 and two source registers from bits 9 to 5 and bits 14 to 10. The *cex* bit specifying if conditional execution is at 25, the condition register at bits 24 to 22, and the condition in bits 21 to 19. In all other instructions requiring the same fields, the fields consists of the same bits. That means, for example, when the destination register is required by an instruction it is always from bits 4 to 0.

Some instructions have two decodings for instances when an instruction does not use conditional execution. The additional bits are then used for an expanded immediate value. Consider the ADDI instruction that adds a register value to an immediate value. Table 3.4 shows the encoding with conditional execution and Table 3.5 shows the encoding without conditional execution.

Table 3.4: ADDI Instruction (with conditional execution)

opcode (101101)	cex (1)	creg	cond	imm	sreg1	dreg
31...26	25	24...22	21...19	18...10	9...5	4...0

Table 3.5: ADDI Instruction (without conditional execution)

opcode (101101)	cex (0)	imm	sreg1	dreg
31...26	25	24...10	9...5	4...0

For conditional execution of the ADDI instruction, the condition specified by *cond* is compared to the flags in the condition register specified by *creg*. When *cex* is 0, the instruction is always executed and *creg* and *cond* are no longer necessary. By freeing up the *creg* and *cond* fields, there is now more space in the instruction for other types of data. The immediate value is then expanded from 9 bits to 15 bits.

The remaining instructions generally follow the same format and expand immediate values to the unused fields. In the case of some instructions, such as CMPI (compare immediate), this can mean dividing the immediate value into two parts (Table 3.6).

The destination register field is unused, allowing part of the immediate value to be stored in bits 4 to 0 as well as bits 21 to 10, resulting in a split immediate value. This immediate value is recombined during the DECODE stage of the pipeline.

Table 3.6: CMPI Instruction

opcode (110111)	cex (0)	creg	immr	sreg1	imml
31...26	25	24...22	21...10	9...5	4...0

3.0.3 COFFEE Software and Tools

The COFFEE compiler and binary utilities have been developed specifically for the COFFEE Core. The compiler tool-chain is based on GCC version 3.44. The COFFEE binary utilities, or Binutils, are based on GNU Binutils version 2.17.

3.0.4 Challenges of the COFFEE Core

The COFFEE Core has been used primarily for research purposes, as it is highly modifiable with all parts of the processor accessible to the researchers. As discussed in the Introduction, maintaining the software for the COFFEE Core requires additional labour every time the GCC tool-chain or GNU Binutils is updated. By interfacing with an industry-standard architecture, researchers will be able to take advantage of already available up-to-date compilers and tools. In the next chapter, the two candidate architectures, ARM and MIPS, are compared with COFFEE to determine the similarities to COFFEE and the potential benefit to the researcher using the COFFEE Core.

4. MIPS AND ARM ARCHITECTURES AND COMPARISON

A brief comparison of the COFFEE architecture with the MIPS and ARM architectures was performed to determine which architecture is most similar, thus being easier to implement, and which would provide the most benefits to the end user in terms of available tools.

4.1 Comparison of MIPS and ARM

COFFEE, ARM, and MIPS are all RISC architectures. All three architectures use load and store models for managing data.

As the design of the COFFEE instruction set took into account the designs of other RISC instruction sets, it was assumed that there would be overlap in available instructions and features of the ISA [16].

Comparison of Available Tools

Both MIPS and ARM provide development tools for producing code to run on MIPS and ARM architectures. In addition, open-source GNU tools are available for both MIPS and ARM.

MIPS Tools MIPS provides several tools and software for different development needs when developing for the MIPS platform. The tools consist of the MIPS Corporation's own software and plugins to commonly used software. The MIPS Navigator provides low level debugging C/C++ tool-chains, and an instruction set simulator. Eclipse plugins are provided to profile software running on a MIPS core. Probe software for debugging MIPS cores and that take advantage of any special features of the core is available. MIPS also provides new releases of the Sourcery G++ GNU Toolchain for systems without an operating system, and systems with GNU/Linux operating systems. Additional MIPS tool-chains and tools have also

been developed by third parties. MIPS also contributes to the Linux project, thus ensuring that embedded Linux runs smoothly on MIPS platforms [17].

ARM tools ARM provides several tools and software for development on the ARM platform. The ARM Development Studio provides a ARM compilation toolchain and tools such as a debugger, performance analyzer, simulator, and Eclipse plugins. A third party development environment is also provided by Keil, called MDK-ARM Microcontroller Development Kit, which in addition to development and debug tools, also provides its own real-time operating system and middleware libraries [7].

The MIPS and ARM tools are comparable and neither offers any obvious advantages over the other. The one downside of both MIPS and ARM tools is the cost. However, there are open-source GNU tool-chains for both MIPS and ARM.

Comparison of Common Instructions

As the COFFEE ISA was designed in keeping commercial processors in mind, there are many overlapping instructions and features with MIPS and ARM ISAs. The MIPS and ARM instruction sets were compared to the COFFEE instruction set and classified as identical, similar, and non-similar instructions as shown in Table 4.1. Identical instructions have the exact functionality as instructions in COFFEE. Instructions classified as similar mean that a comparable instruction exists in COFFEE, but would need some modifications within the processor to have exactly the same functionality. Non-similar instructions, or the remaining instructions, may still have some relation to COFFEE instructions, but are different enough that they may require more time to implement or need additional hardware. The number of remaining instructions was not counted because the number is variable depending on the features of the processor.

Table 4.1: Instruction Set Comparison

Instruction Set	Identical	Similar
MIPS	22	51
ARM	7	32

As is evident from the above table, there is more overlap between MIPS and COFFEE than between ARM and COFFEE. The MIPS instructions set has most of the basic arithmetic and logic instructions as does COFFEE, whereas ARM eliminates many of the simpler instructions and instead has more complicated arithmetic and logic instructions that are not available in either MIPS or COFFEE. Additionally, the ARM instruction set seems to have many complex instructions that may not

necessarily abide by the principles of simple RISC instructions [3]. For example, in ARM the instruction ADD (addition) is equivalent to ADDU (add unsigned) in MIPS and COFFEE. An equivalent to a signed ADD does not exist because, as ARM documentation states, the instruction is not commonly used by compilers in any case and is not necessary [3]. ARM also contains more complicated instructions such as MLA (multiply and accumulate), and MLS (multiply and subtract) which would need hardware modification within COFFEE to implement.

ARM instructions do use conditional execution in a similar way that COFFEE uses conditional execution and reserves four bits in the instruction format. The four bits are reserved to represent the sixteen possible execution conditions [5; 4]. The condition codes are compared to the application program status register (APSR) to determine execution. One major difference between ARM and COFFEE is that COFFEE has only three bits reserved for conditional execution, meaning there are only eight possible execution conditions, instead of the sixteen for ARM. This also means that the status register of ARM stores one additional condition flag in comparison to COFFEE's condition registers, which stores only three ALU flags. This unfortunately means that the ARM and COFFEE conditional execution is not compatible and ARM code can not be executed as the ARM compiler has intended when using the COFFEE processor. Conditional execution is not present in the MIPS ISA and instructions are always executed.

ARM uses several instruction formats for the instruction encodings, varying depending on the version of the instruction set. MIPS has three instruction formats.

Comparison of MIPS and ARM Register Sets

The MIPS register set has 32 general purpose registers available at any one time. The first register, GPR 0, is zero at all times. The ARM register set has only 16 available general purpose registers at any one time. The main purpose of this is to save bit space in the instruction, as it requires only four bits in the instruction to specify a register instead of five bits for MIPS [3]. Additionally, MIPS has HI and LO registers to save the result of multiplications and divisions [19]. These registers are not present in ARM and MIPS.

ARM stores all the ALU condition flags in the ARM APSR which are then compared with condition codes specified by conditional instructions [3]. In COFFEE, the ALU flags are saved in the first condition register (CR 0). The conditions can then be tested by any conditional instruction specifying CR 0.

MIPS, on the other hand, does not use condition codes to make comparisons. In-

stead, comparisons are made that compare two registers to each other, compare a register value to an immediate, or compare a register value to zero. For conditional branch instructions, a branch occurs if the comparison is true. For compare instructions such as SLT (set if less than), the true or false result is stored in a GPR for use by a later instruction [19].

4.2 MIPS

MIPS was the chosen architecture because of the instruction overlap with COFFEE, the architectural similarities, and the simple instruction format of MIPS instructions, making the instructions easier to decode. The MIPS architecture will be reviewed in the following chapter based on the MIPS Architecture documentation [18] [19] [20].

The MIPS architecture reviewed here is known as MIPS32 Release 3, meaning the most recent release of the architecture for 32 bit instructions as of this writing. Different implementations of the MIPS architecture exist in industry, and each may have slight differences in the hardware design. However, the overall architecture of the processors is always the same and compliant with MIPS [18].

4.2.1 MIPS Architecture

The basic MIPS architecture is described in this section and excludes any additional features such as co-processing and floating-point units.

The basic pipeline of the MIPS architecture consists of four stages: Fetch, Arithmetic operation, Memory Access, and Write-back. The pipeline fetches a new instruction each clock cycle, and moves the instructions through the pipeline stage by stage [18].

The MIPS architecture also meets the RISC guidelines mentioned earlier in this thesis in that it uses a load and store architecture to manage memory access, operations are register to register, operations and addressing are simplified, and the architecture is pipelined with a throughput of one instruction per cycle.

Each MIPS register is 32 bits. MIPS contains one set of 32 GPRs and two special HI and LO registers. In the 32 GPR register set, the first register, R0, is tied to low, meaning that the value of the register is unchangeable and always contains zero. The last register, R31, is used to store address information for instructions such as for JAL (Jump and Link). When the register is not being used to store link data, the register acts as any other GPR. The HI and LO registers act as containers for the 64 bit result of a 32 by 32 bit multiplication or division. The HI register contains

the upper 32 bits, and the LO register contains the lower 32 bits. The multiplication and division instructions automatically store the result in these registers [18].

The MIPS ISA also refers to an interface called CP0 (co-processor zero), which acts as an interface to the privileged resource architecture to accomodate an operating system. The processor status and user modes are stored in the CP0 register CPSR (Current Program Status Register). The APSR (Application Program Status Register) is contained in the CPSR and stores the zero, carry, sign, and overflow flags (Z,C,N,V) of an ALU operation or compare instruction [18]. This is comparable to the condition register of COFFEE with the exception that COFFEE does not track the overflow flag (V).

4.2.2 MIPS Instruction Set Architecture

The MIPS32 Release 3 instructions can be subset to remove instructions that are not needed if the processor does not contains the functionality. Subsetting must be performed according to the subsetting rules outlined in the MIPS documentation [18]. Basically, all CPU instructions must be implemented and all FPU and co-processor instructions are optional.

The instructions remaining in the subset can be categorized as follows:

- CPU Arithmetic Instructions
- CPU Branch and Jump Instructions
- CPU Instruction Control Instructions
- CPU Load, Store, and Memory Control Instructions
- CPU Logical Instructions
- CPU Insert/Extract Instructions
- CPU Move Instructions
- CPU Shift Instructions
- CPU Trap Instructions
- Privileged Instructions

The above MIPS instructions can be divided into three main types of instructions, which also correspond to the instruction formats used to describe each instruction [18].

- I-Type Instructions - Immediate instructions
- R-Type Instructions - Register instructions
- J-Type Instructions - Jump instructions

The instruction formats for each type make use of several fields to describe different parts of the instructions such as the opcode, source and destination registers, and immediate values shown in Table 4.2 [18].

Table 4.2: MIPS Instruction Fields [18]

Field	**Size (bits)**	**Description**
opcode	6	opcode
rd	5	destination register
rs	5	source register
rt	5	second source register
shift amount	5	shift amount
function	6	opcode extension
imm	16	immediate value

An I-Type instruction, or immediate instruction, is any instruction needing to take in an immediate value. The name of the instruction usually ends in an 'i' to denote an immediate but can also include other instructions. Instructions such as ADDI (add immediate), ADDIU (add immediate unsigned), ORI (or immediate), and XORI (xor immediate) take in immediate values. The second input value comes from a register specified by *rs*. The encoding is shown in Table 4.3.

Table 4.3: I-Type Instruction Format

opcode	rs	rt	immediate
31...26	25...21	20...16	15...0

An R-Type instruction, or register instruction, is an instruction that takes in registers as operands. These include instructions such as ADD, MULT, and SUB. The encoding is shown in Table 4.4.

Table 4.4: R-Type Instruction Format

opcode	rs	rt	rd	shift amount	function
31...26	25...21	20...16	15...11	10...6	5...0

The function field is necessary to extend the instruction set. With an opcode of 6 bits, only 64 instructions are possible. The extension of the 6 bit function field

allows a possible 40096 instructions, much more than a RISC processor will likely ever need.

A J-Type instruction, or jump instruction, takes in a target address. The PC is set to the target address specified in the target field. There are two jump instructions: J (jump), and JAL (jump and link). The encoding is shown in Table 4.5.

Table 4.5: J-Type Instruction Format

opcode	target
31...26	25...0

Branch instructions, although similar in function to jump instructions, use the I-Type instruction format.

After reviewing the MIPS architecture, it was more closely compared with COFFEE to determine the best approach to integrating the two architectures. Emphasis is placed on similar instruction specifications and similarities in the instruction encodings.

5. APPROACH AND DESIGN

The MIPS and COFFEE architectures are now more closely compared to determine the exact process and method of interfacing the COFFEE Core to MIPS. The architecture consists of both the hardware architecture and the instruction set architecture.

A design to integrate the two architectures based on the results of the comparisons is also proposed.

5.1 Hardware Architecture Comparison

General comparisons between the COFFEE architecture and the MIPS architecture were made to gain an overall understanding of the differences that may affect the execution of MIPS instructions on the COFFEE core. The following architectural comparisons were made:

- Pipeline
- GPR Registers
- Condition Registers
- Special Registers
- Privileged Architecture

5.1.1 Pipeline

The COFFEE pipeline consists of six stages: FETCH, DECODE, EXE1, EXE2, EXE3, and WRITE-BACK. The MIPS pipeline may contain a number of stages depending on the implementation, but will always contain the stages fetch, execute, memory access, and write-back. The differences between the two pipelines should not affect the decoding of the MIPS instructions in the COFFEE core.

5.1.2 GPR Registers

There are 32 GPR registers in COFFEE with no restrictions on what can be stored in a register. In MIPS, there are also 32 registers. However, register zero in MIPS, R0, is always zero and can not be set to any other value. The MIPS compiler depends on R0 always being zero and a program would need R0 to always be zero for the program to properly run. To run MIPS instructions properly on COFFEE the GPR 0 register would also have to remain zero. To prevent changes made to R0, it could be detected in the Decoder when R0 is used as a destination register and the assignment either prevented, or the value of zero written to the register.

5.1.3 Condition Registers

There are 8 condition registers in COFFEE. The condition register stores the result of a comparison from the CMP instruction or the flags from an arithmetic operation. The flags N, Z, and C denote negative, zero, and carry. If the result of an operation is negative, the N flags is set. If the result is zero, the Z flags is set. If there is a carry bit in the result, the C flags is set. When the result is needed, such as is the case for a conditional branch such as BNE, branch when not equal, the branch instruction reads the condition register, which already contains the result of the comparison. MIPS, on the other hand, does not use condition registers for comparisons. When an instruction requires a comparison, such as BGEZ, branch if greater than or equal to zero, the comparison and branch are done in one instruction. That means that a register is compared to zero, and if the result is greater than zero a branch occurs. There are no condition registers involved in the comparison and branch. Some COFFEE instructions, such as arithmetic instructions like ADD (Addition) and SUB (Subtraction), automatically save the condition flags resulting from the operation to the COFFEE condition register zero to be used by later instructions. MIPS arithmetic instructions do not save any condition as condition registers are not used.

5.1.4 Special Registers

There are two special registers in MIPS called LO and HI used for storing the 64 bit result of a multiplication or division. The additional instructions, MFLO and MFHI which denotes move from LO and move from HI respectively, are then used to retrieve the results from LO and HI and store the result in a GPR. The LO and HI registers do not exist in COFFEE and would need to be added for the dependent instructions to work properly. The COFFEE core does have a method for obtaining

a 64 bit result for a multiplication using the MULS or MULU instruction for the lower 32 bits and MULHI instruction to retrieve the upper 32 bits. The result is saved in two GPRs specified by the instructions. The affected instructions are summarized in Table 5.1.

Table 5.1: Affected Special Register Instructions

Instruction	**Description**	**Reason why affected**
MTHI, MTLO	Move to HI and LO registers	HI and LO registers do not exist in COFFEE
MFHI, MFLO	Move from HI and LO registers	
MULT, MULTU	Multiplication to HI and LO registers	

MIPS also provides an instruction to read a value from a hardware register and store the result in a GPR. The hardware registers store information such as which CPU is currently in use, and have some relation to the privileged interface. Access to the register is controlled by CP0. No comparable instruction exists in COFFEE. The affected instructions are summarized in Table 5.2.

Table 5.2: Affected Hardware Register Instructions

Instruction	**Description**	**Reason why affected**
RDHWR	Read hardware register	No comparable instruction in COFFEE

5.1.5 Privileged Architecture

The privileged architecture in COFFEE and MIPS is used by an operating system running on the processor. The COFFEE privileged architecture is mainly implemented using special instructions and a separate register set. The COFFEE processor can also run in user mode or superuser mode. The superuser mode is used for the operating system and allows default access to the privileged register set, a set of 32 GPR's reserved just for the superuser mode. The SCALL (system call) instruction is used to switch the COFFEE core into the superuser mode and access the privileged architecture and the RETU instruction switches back to the user mode. The privileged register set can also be accessed using the CHRS, change register set, instruction. In MIPS the privileged architecture (PRA) is accessible through an interface called CP0, co-processor zero, or also referred to as the system control processor. The PRA in MIPS is responsible for "exception handling, memory management, scheduling, and control of critical resources" [18] and contains several control and status registers relating to these functions. Specific co-processor instructions are used to load and store data to the registers of CP0. The instruction MTC0

(move to co-processor zero) moves data from a GPR to a specified co-processor register and MFC0 (move from co-processor zero) moves data from a GPR to a co-processor register. Although both MIPS and COFFEE contain privileged register sets, the manner of accessing registers is different for MIPS and COFFEE meaning the COFFEE privileged interface would need to be modified to accomodate MIPS.

Interrupts and Exceptions Certain MIPS privileged instructions have similar instructions in COFFEE. The interrupt instructions such as EI and DI (enable interrupt and disable interrupt) and ERET (return from interrupt) have similar COFFEE instructions. The MIPS enable and disable interrupt instructions, EI and DI, can be decoded and use the same control signals as the enable and disable interrupt instructions for COFFEE. The MIPS exception return instruction, ERET, is used to return from interrupts, exceptions, and error traps and clears execution and instruction hazards. The similar COFFEE instruction, RETI for return from interrupt, manages interrupt returns and restores the PC, CR0, and PSR from the stack. The compiler ensures the instruction is followed by two NOPs to clear hazards. Because the specification of the two instructions have small differences, it is unclear if RETI from COFFEE can be used in place of ERET from MIPS, since errors will likely occur if the instruction is used for an exception or error trap return. The MIPS compiler also does not guarantee the two NOP instructions necessary after RETI. The affected instructions are summarized in Table 5.3.

Table 5.3: Affected Interrupt and Exception Instructions

Instruction	**Description**	**Reason why affected**
ERET	Exception Return	Different specifications
MFC0	Move from Co-processor Zero (Privileged Interface)	Differences in privileged architecture
MTC0	Move to Co-processor Zero (Privileged Interface)	

Memory Resources The COFFEE core does not currently have instructions for cache access and does not use virtual memory. Therefore, the related cache access and translation look-aside buffer (TLB) instructions are not implementable. The affected instructions are summarized in Table 5.4.

Other Privileged Instructions Additional instructions include access to shadow sets and entering a low power standby mode. COFFEE does not currently use shadow sets or have a low power mode. Comparable instructions also do not exist in

Table 5.4: Affected Memory Resource Instructions

Instruction	Description	Reason why affected
TLBP, TLBR, TLBWI, TLBWR	Read, Write, and Probe TLB	No existing COFFEE TLB instructions or architecture
CACHE, SYNCI	Cache operation and synchronization	No existing COFFEE cache instructions

COFFEE. Hardware modifications and additions would be necessary. The affected instructions are summarized in Table 5.5.

Table 5.5: Other Affected Privileged Instructions

Instruction	Description	Reason why affected
WRPGPR	Write to GPR in Previous Shadow Set	No matching COFFEE instruction
RDPGPR	Read from GPR in Previous Shadow Set	No matching COFFEE instruction
WAIT	Enter Standby Mode	Standby Mode does not exist in COFFEE

5.2 Instruction Set Comparison

The instruction set and functions of instructions were compared for MIPS and COFFEE to determine how the MIPS instructions could be integrated into the COFFEE architecture. The following comparisons were made.

- Instruction Specifications
- Instruction Format
- Instruction Encodings
- Comparisons, Conditions, and Flags

5.2.1 Instruction Specifications

The specifications of all MIPS and COFFEE instructions are compared to determine the overlap between the two instruction sets and to gauge the similarities between the remaining instructions. Overlapping instructions with the exact same specifications would be the easiest to implement on the COFFEE core, whereas instructions with similar specifications may need some modification. A significant number of MIPS instructions are expected to have no matching specifications in the COFFEE instruction set.

Identical instructions The COFFEE instruction set and MIPS instruction set have an overlap of 22 instructions with the same specifications. It is expected that these will be the simplest instructions to implement and are shown in Table 5.6 as well as Appendix A.1.

Table 5.6: Instructions with similar specifications

MIPS Instruction	Description	COFFEE Instruction
ADDI	Add immediate	ADDI
ADDIU	Add immediate unsigned	ADDIU
ADDU	Add unsigned	ADDU
AND	Bitwise AND	AND
ANDI	Bitwise AND immediate	ANDI
DI	Disable interrupts	DI
EI	Enable interrupts	EI
EXT	Extract bit field	EXBF
JAL	Jump and link	JAL
JALR	Jump and link register	JALR
J	Jump	JMP
JR	Jump register	JMPR
LW	Load word	LD
MUL	Multiply word to GPR	MULS
NOP	No operation	NOP
NOR	Bitwise NOR	NOR
OR	Bitwise OR	OR
ORI	Bitwise OR immediate	ORI
SUB	Subtract	SUB
SUBU	Subtract unsigned	SUBU
XOR	Bitwise exclusive OR	XOR

Similar and easily implementable instructions Some MIPS instructions have specifications that are similar but not identical, or are instructions that could be implemented with some minor modification using the existing functionality of the COFFEE Core. The differences and reasons why are addressed throughout this chapter and are summarized in Appendix A.2.

Instructions with no comparable match A number of MIPS instructions have no match at all to the COFFEE instructions and are described in the section Remaining Instructions. These instructions would likely need significant modification. The number of instructions in this category is variable, as it depends on the features of the processor. A number of instructions in the smallest subset of the MIPS instructions set are reviewed in this Chapter and the analysis is summarized in

Appendix A.3.

5.2.2 Instruction Format

The instruction format of MIPS is compared to the format of COFFEE to find comparable fields. Each field of each instruction format is compared to determine how compatible the instructions are and what changes would need to be made in the COFFEE Core instruction decoding to accomodate the MIPS instructions. When comparing, all the MIPS instruction fields will need to match with COFFEE instruction fields at least in field type. There are some differences in lengths of fields and how each field is used in the decoding, which is addressed in the section Instruction Encodings.

The Figure 5.1 shows the comparable COFFEE fields for the MIPS I-Type instruction format.

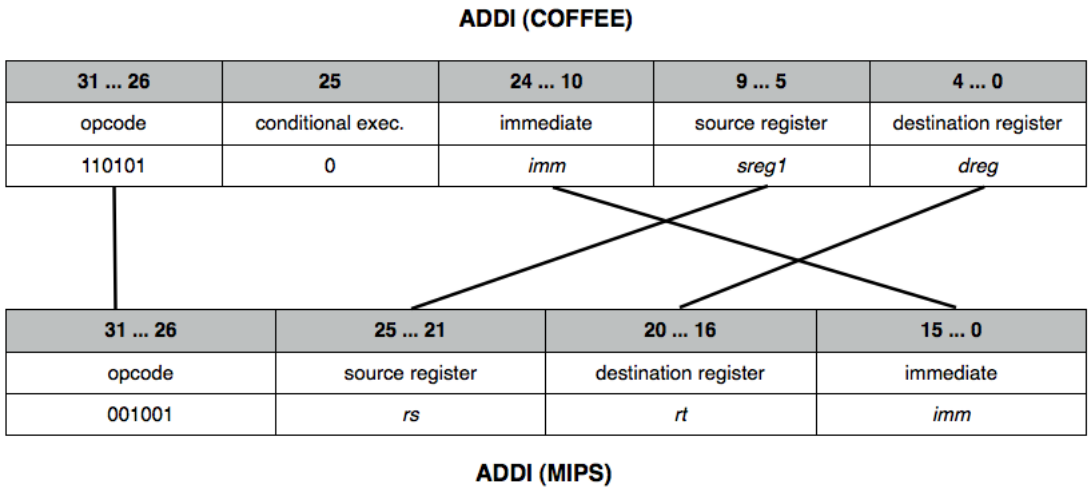

Figure 5.1: MIPS I-Type format compared to COFFEE Immediate format

The length in bits of immediate values in COFFEE is variable. In MIPS, however, the length of the immediate value is always 16 bits. This makes the sign and zero extension of the immediate value much simpler than it is currently for the COFFEE instructions.

The Figure 5.2 shows the comparable COFFEE fields for the MIPS J-Type instruction format.

The target address of the MIPS and COFFEE version are of different lengths with 25 bits for COFFEE and 26 bits for MIPS. One difference between the two jump instructions, however, is how the target address is evaluated. To calculate the effective address in COFFEE, the value in the target field is shifted left by one bit and

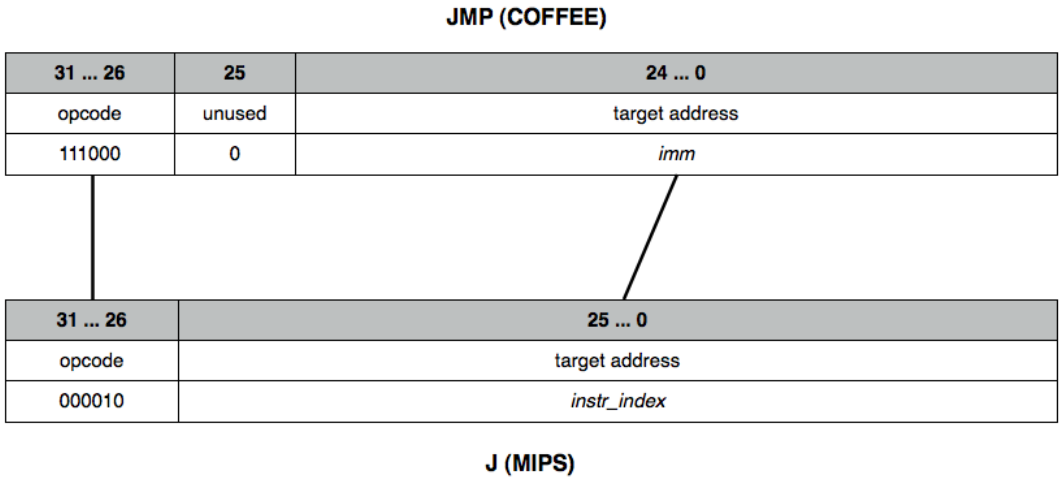

Figure 5.2: MIPS J-Type format compared to COFFEE Jump format

sign extended. To calculate the target address in MIPS, the value in the target field is shifted two bits.

The Figure 5.3 shows the comparable COFFEE fields for the MIPS R-Type instruction format.

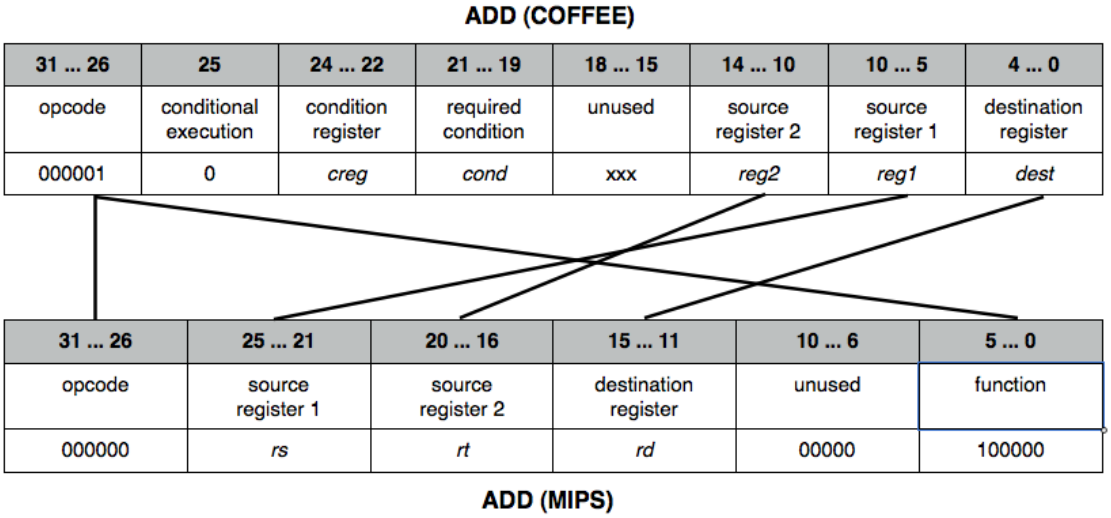

Figure 5.3: MIPS R-Type format compared to COFFEE two-register format

In the MIPS R-Type format, the instruction is determined by the combination of the *opcode* and *function* fields, and is comparable to the *opcode* field of the COFFEE instruction. The source and destination registers match up directly for both formats, with the same number of bits for determining each register. As the COFFEE conditional execution is not used, the fields *creg* and *cond* are not required and have no match in the MIPS format.

From these comparisons, it is seen that the MIPS instruction fields for the three instruction formats have comparable fields for the COFFEE instruction format. This means that to decode MIPS instructions on the COFFEE Core, the COFFEE decoding can simply be modified to account for the fields being in different locations in the instruction.

5.2.3 Instruction Encodings

The instruction encodings for MIPS and COFFEE instructions were compared in more detail to determine which instructions would translate well from MIPS to the COFFEE Core and which instructions would need additional changes in the decoding.

There are some MIPS instructions that are very similar to COFFEE instruction except for small differences in the encoding. Some of these differences could prevent an instruction from functioning in COFFEE. Probably the easiest way to determine if these instructions would work or not would be to implement and test them.

5.2.4 Shift and Rotate Instructions

The shift instructions in MIPS have very similar matching instructions in COFFEE with one minor difference between them. In COFFEE, the instructions allow there to be 6 bits to determine the amount of bits to shift. In MIPS, the instruction allows there to be only 5 bits to determine the amount of shift. This does not seem to be too much of a problem for immediate values, since an extra zero can simply be added to make the 5 bits from the MIPS instruction the required 6 bits for COFFEE. Currently, the Decoder sends 11 bits to the ALU where the appropriate bits are then selected. This can be modified to set the upper 6 bits to zero and to retain the lower bits. For MIPS shift instructions taking the shift amount from the lower 5 bits of a register, it could also be ensured that the 6th bit of the register is zero. Also affected are the rotate instructions, ROTR and ROTRV for rotate word right with the rotate amount defined either as an immediate value or a register value. Additional hardware may need to be added to the ALU to allow for a rotate operation. The affected instructions are summarized in Table 5.7.

5.2.5 Comparisons, Conditions, and Flags

COFFEE has eight condition registers. The first condition register stores the Z, N, and C flags (zero, negative, and carry), and all registers store the result of

Table 5.7: Affected Shift and Rotate Instructions

Instruction	**Description**	**Reason why affected**
ROTR, ROTRV	Rotate instructions	COFFEE requires 6 bits rather than 5
SLL, SLLV, SRA, SRAV, SRL, SRLV	Shift instructions	

comparisons made using the CMP or CMPI (compare) instructions. The results are then used in later instructions, such as BNE (branch if not equal). The instruction refers to the flags in the specified condition register when determining if a branch should occur.

MIPS, on the other hand, has no condition registers and uses other methods for comparisons. A few MIPS compare instructions exist, such as SLT (set on less than), which stores a boolean result of a comparison in a GPR. The result can then be used by later instructions. However, most MIPS instructions requiring a comparison to be made are compound instructions, with both the compare and an additional function within the same instruction. For example, the BNE instruction in MIPS does both a comparison and performs a branch, rather than using two instructions as in COFFEE. As a result of these two differences, none of the compare instructions from MIPS can be matched to COFFEE instructions. This includes the compare and branch, compare and trap, compare and store, and compare and move instructions.

One possible modification would be to stall the processor one clock cycle at the FETCH stage, so that two COFFEE instructions can be used in place of one MIPS instruction. For example, in place of the MIPS BNE instruction, the COFFEE instructions CMP and BNE can be executed consecutively. When the CMP instruction is in the EXE1 stage where the comparison is evaluated, the result is forwarded to the DECODE stage where it can be used by the BNE instruction. The target address of the branch can be extracted from the MIPS BNE instruction for use in the COFFEE BNE instruction.

The affected instructions are summarized in Table 5.8.

5.2.6 Remaining Instructions

There are a number of instructions, that are without a match for a reason other than one mentioned in the previous architecture comparisons. Some of these instructions could be implemented with minor modifications to the architecture, but most of these instructions simply have no matching instruction and no supporting hardware

Table 5.8: Affected Comparison and Condition Instructions

Instruction	**Description**	**Reason why affected**
SLT, SLTI, SLTIU, SLTU	Set on less than	No matching COFFEE instruction
All TRAP instructions	Trap exception	Requires 2 instructions in COFFEE
All BRANCH instructions	Branches	Requires 2 instructions in COFFEE

in COFFEE.

Load, Store and Memory Instructions Several load and store instructions exist in MIPS for loading and storing words, halfwords, and bytes, loading value from unaligned memory addresses, and loading a lower immediate value. COFFEE contains LD (load word), LUI (load upper immediate) and LLI (load lower immediate). The load word instructions for MIPS and COFFEE can be matched because the specification is the same. The LUI MIPS instruction, however, can not be matched to the LUI COFFEE instruction because the MIPS version loads an immediate into the upper 16 bits and clears the lower 16 bits, whereas the COFFEE version preserves the lower 16 bits. This could be resolved by using the COFFEE instructions LLI and LUI to first clear the lower bits and then load the upper bits. The processor would need to be stalled one clock cycle at the FETCH stage. The MIPS compiler frequently uses the LUI instruction in conjunction with LW to load a value from memory. The LUI instruction is necessary to refer to a 32 bit memory address. The shared memory instruction SYNC acts as either a completion barrier or ordering barrier for instructions loading from or storing to shared memory. The SYNC instruction will cause the processor to stall until, in the case of a completion barrier, all load and store instructions in the instruction stream before SYNC are completed or, in the case of an order barrier, all load and store instructions before SYNC are ordered before instructions after SYNC. This instruction could be implemented by stalling the processor when needed. The instruction SYNCI is used to synchronize caches after changes are made to the instruction stream so that the new instructions can be fetched. The synchronization is done by writing the instructions to an address provided by the SYNCI that specifies to which cache line to write for all caches. The PREF instruction is used to increase performance of a program by moving data between memory and cache. Data is not always moved when a PREF instruction is executed and the instruction may do nothing. The action of the instruction depends on what the data will be used for and if it improves program performance [19]. SYNC, SYCI, and PREF do not have matching instructions in

COFFEE. The two instructions LL (load linked word) and SC (store conditional) are used for atomic read-modify-write in synchronized memory. The PAUSE instruction checks if the atomic operation started by LL is still being performed. There are no comparable instructions in COFFEE. The affected instructions are summarized in Table 5.9.

Table 5.9: Affected Load and Store Instructions

Instruction	**Description**	**Reason why affected**
LB, LBU, LH, LHU, LL, LWL, LWR	Load Instructions	No matching COFFEE instructions
SB, SH, SWL, SWR	Store Instructions	
SYNC, SYNCI	Shared Memory Load and Store and Cache Synchronization	
LL, SC, PAUSE	Atomic Load and Store	
PREF	Prefetch, moving data between memory and cache	

Arithmetic Instructions Instructions such as CLO (count leading ones), CLZ (count leading zeros) and DIV (divide) do not have any matching instruction or supporting hardware in COFFEE. MIPS sign extend instructions SEH and SEB (sign extend halfword and sign extend byte) are similar to the COFFEE SEXTI (sign extend immediate) instruction, but are still not a match. The COFFEE core can sign extend immediate values, but currently does not have the ability to sign extend a value from a register. The modification to extend a register value can be done in the Decoder. Some arithmetic instructions can be implemented using two COFFEE instructions. The arithmetic instruction MADD implements a multiply and addition in a single instructions. The instruction MSUB is a multiply and subtraction in a single instruction. COFFEE does not currently support several arithmetic operations within one instruction. The LUI instruction mentioned in the Load, Store, and Memory instructions is an arithmetic instruction. The affected instructions are summarized in Table 5.10.

Insert and Extract The INS (insert) instruction in MIPS is used to insert a number of bits into a register value at a specified bit position, whereas the EXT (extract) instruction extracts a number of bits from a register value. The WSBH instruction (word swap byte halfword) swaps the bytes within each halfword of a register value. The COFFEE ISA has several instructions for extracting bits, a byte,

Table 5.10: Affected Arithmetic Instructions

Instruction	**Description**	**Reason why affected**
CLO, CLZ	Count Leading Zeros/Ones	No matching instruction in COFFEE
SEH, SEB	Sign Extend Halfword/Byte	Not an exact match to COFFEE instruction SEXTI
DIV, DIVU	Division	Division instruction is not available in COFFEE
MADD, MADDU, MSUB, MSUBU	Multiply and Add/Subtract	Combined instructions can not be implemented in the current COFFEE pipeline
LUI	Load Upper Immediate	Instruction specification for LUI not the same as in COFFEE

or a halfword from a register value. The EXBFI instruction can extract a specified number of bits from a register value. There is some difference in the how the immediate inputs are defined for MIPS EXT and COFFEE EXBFI. However, with some modification for the immediates in the Decoder it would be possible implement the MIPS EXT instruction. The INS instruction and WSBH instruction have no matching instructions in COFFEE and would require additional modifications. The affected instructions are summarized in Table 5.11.

Table 5.11: Affected Insert and Swap Instructions

Instruction	**Description**	**Reason why affected**
INS	Insert	No matching instruction in COFFEE
WSBH	Word Swap Byte Halfword	No matching instruction in COFFEE
EXT	Extract	Inputs do not match

Trap instructions Most of the MIPS trap instructions also include a comparison, such as TEQ for trap if equal and can not currently be implemented in COFFEE. Other trap instructions include BREAK, to cause a breakpoint exception, and SYSCALL, to cause a system call exception. The COFFEE core uses only the TRAP instruction to cause an exception, for which there is no directly equivalent in MIPS. The similarly named system call instruction, SCALL, in COFFEE transfers the processor to the superuser mode, but does not call an exception. The affected instructions are summarized in Table 5.12.

Table 5.12: Affected Trap Instructions

Instruction	Description	Reason why affected
TRAP Instructions	Causes Trap exception if evaluated as true	No comparable evaluate and trap instructions in COFFEE
BREAK	Causes Breakpoint exception	No matching instruction in COFFEE
SYSCALL	Causes System Call exception	No matching instruction in COFFEE

Execution Hazard Barrier Instructions There are a few instructions that address hazard barriers in the pipeline. The EHB (execution hazard barrier) instruction stops execution until all execution hazards are cleared and is interpreted as a logical shift left (SLL) with a shift of three bits. Instruction hazards as well as execution hazards are cleared using the JR.HB, JALR.HB and ERET instructions through use of a software barrier by inserting instructions into the instruction stream. The COFFEE core does not have any comparable instructions for clearing hazard barriers, instead relying on the NOP instruction. The MIPS EHB instruction can be reproduced in COFFEE using the SLL instruction. Implementation of the other hazard barrier instructions would require cache synchronization instructions. The affected instructions are summarized in Table 5.13.

Table 5.13: Affected Hazard Barrier Instructions

Instruction	Description	Reason why affected
EHB	Clear Execution Hazards	No matching instruction in COFFEE, but comparable to a series of NOPS or bit shifts
JR.HB, JALR.HB	Jump and Clear Hazard Barrier	No matching instruction in COFFEE
ERET	Returns from exception and clears instruction and hazard barriers	No matching instruction in COFFEE

5.2.7 Comparison Summary

There are a few key things to notice in the comparison.

- There is an overlap of 22 instructions in the MIPS and COFFEE instruction sets.

- The MIPS instruction format can be mapped to the COFFEE instruction format using comparable fields.
- MIPS has a simple instruction format and thus does not require use of conditional execution or condition registers.
- MIPS does not contain HI and LO registers meaning that instructions that require these register can not be implemented without them existing in COFFEE.
- MIPS compare and branch, compare and trap, or compare and store instructions are only one instruction each. However, two instructions are required in COFFEE meaning that MIPS compare instructions can not be implemented without hardware modifications.
- COFFEE does not tie the GPR 0 register to zero, as does MIPS, which may complicate a running program were the value to ever change.

5.3 Proposed Integration Approach

The proposed approach to integrating the MIPS instruction set to the COFFEE architecture is to initially concentrate on minimally modifying the hardware to accomodate as many instructions as possible. As there is an overlap of a few similar instruction in the MIPS and COFFEE instruction sets, it is expected that at least these instructions can function properly by using this approach. After these instructions are working, the approach can then be shifted to modify hardware for specific MIPS instructions.

The modifications needed to implement the MIPS instructions on the COFFEE Core are broad and would require changes and additional hardware to the entire processor. Therefore, the scope was limited to only the decoding and control processes in the COFFEE Core. This limits the modifications to only the DECODE stage of the pipeline and the Core Control Unit.

The first step to modifying the existing COFFEE hardware to accomodate the MIPS instruction set is to change the decoding step in the hardware to decode MIPS instructions rather than COFFEE instructions. In referring to the actual COFFEE hardware as described in the Chapter COFFEE RISC Core, it is seen that instruction decoding occurs in several pipeline stages. Instruction decoding happens in the DECODE pipeline stage and also in the five decoding blocks within the COFFEE control unit. All of these decoding entities would need to be modified. In addition to decoding, data and control signals going from the control unit to the

pipeline stages of the processor would also need to be set depending on the decoded instruction. The following would need to be considered:

- Decoding the MIPS instruction
  - Determining the instruction in the DECODER unit and the COFFEE control unit
  - Extracting immediate values and zero or sign extending where needed
  - Determining source and destination registers
- Setting control signals in the COFFEE control unit

The DECODE stage determines which COFFEE instruction has entered the DECODE stage from the previous FETCH stage. The DECODE stage also extends any immediate values and routes the operands to the ALU or co-processor. The following modifications would therefore need to occur in the DECODE stage.

In the COFFEE ISA, the instruction is determined by the 6 bit opcode which is unique for each instruction. In MIPS, the instruction is determined using both the 6 bit opcode and the 6 bit function. A simple modification is needed to determine the instruction from 12 bits for MIPS instead of from 6 bits for COFFEE.

MIPS opcodes can also be divided by format type of either R-Type, I-Type or J-Type. Once the type is known, the correct bits are used to determine the source registers, destination registers, and immediate values. Signal lines are also set to route the data to the appropriate places depending on the instruction format.

The implementation can then be tested using simple assembly programs.

The following chapter outlines the implementation procedure and results.

6. IMPLEMENTATION AND RESULTS

The proposed integration approach is to modify the DECODE and Control Unit parts of the COFFEE core to enable to the most MIPS instructions to function with minimal modifications. A subset of MIPS instructions was selected to be implemented. The COFFEE core is defined as a VHDL design. The design was modified then compiled and simulated using Modelsim.

For the modified COFFEE core to be tested, the MIPS instructions need to be read by the COFFEE core. The LLVM MIPS tool-chain was used to generate the executable. The test code was written in assembly.

The following subset of MIPS instructions was chosen to be implemented and consists of instructions in the MIPS Instruction Set that overlap instructions from the COFFEE Instruction Set. Modifications for decoding and setting control signals is necessary to implement the instructions.

Table 6.1: Implemented Instructions

MIPS Instruction	**Definition**
OR	Bitwise Logical OR
ORI	OR Immediate
XOR	Exclusive OR
AND	Bitwise Logical AND
ANDI	AND Immediate
ADD	Addition
ADDU	ADD Unsigned
ADDI	ADD Immediate
ADDIU	ADD Immediate Unsigned
SUB	Subtract
SUBU	SUB Unsigned
LW	Load Word
SW	Store Word

6.1 Implementation

The COFFEE VHDL designs for the DECODER and the Control Unit were modified. This section will briefly summarize each entity and the modifications needed.

DECODER The DECODE unit is responsible for decoding, updating the PSR, extending immediates, setting control signals, and routing data. The modifications are summarized below:

- Decoding was modified to determine the MIPS instruction and determine instruction format type (R-Type, I-Type, or J-Type)
- Immediates were retrieved and either zero or sign extended depending on the instruction
- The source and destination registers are set depending on the instruction and format type
- Operands are selected for the ALU based on the instruction

CONTROL UNIT The COFFEE Control Unit controls the operation of the pipeline based on the instructions, as well controlling the mode of operation and interrupts and updating the PSR. The Control Unit is divided into 5 decoding entities. Each entity corresponds to a section of the COFFEE pipeline. Each entity is named CCU Decode with a number indicating stages one to five.

CCU Decode I The first decode stage in the COFFEE Control Unit is executed when the associated instruction is in the DECODE stage. The results are non-registered and are used immediately. The ALU is controlled from CCU Decode I by setting ALU opcodes that are then decoded within the ALU. The opcode design means that modifications to the ALU behaviour, such as for new instructions, would need to be made within the ALU rather than the controller. Since this is outside the scope of the thesis, the ALU controls remain unmodified. The modifications for MIPS are summarized below:

- Decoding was modified to determine the MIPS instruction and determine instruction format type (R-Type, I-Type, or J-Type)
- Comparisons with condition register and writing to condition register is disabled
- Number of operands, source and destination registers are set according to the format type
- Controls to ALU are set depending on the instruction type

CCU Decode II The second decode stage in the COFFEE Control Unit is executed when the associated instruction is in the DECODE stage. The results are registered and used when the instruction moved to the first execution stage, EXE1. CCU Decode II is focused on register mapping to the co-processor. Since the co-processor will remain unused for the subset of instructions being implemented and modifications are not expected to affect the outcome, the CCU Decode II was left unmodified.

CCU Decode III The third decode stage in the COFFEE Control Unit is executed when the associated instruction is in the EXE1 stage. The results are registered and used when the instruction is in the second execution stage, EXE2. The modifications for MIPS are summarized below:

- Modifications were made to decode load and store instructions and set controls for read and write access
- Modifications were made to pass the instruction to the next decode module

CCU Decode IV The fourth decode stage in the COFFEE Control Unit is executed when the associated instruction is in the EXE2 stage. Minor modifications were made to decode the load instruction and to route data from the previous stage, CCB, or multiplication result to the next stage.

CCU Decode V The fifth decode stage in the COFFEE Control Unit is executed when the associated instruction is in the WRITE-BACK stage. No modifications were needed for this stage since the decoding input is from CCU Decode IV.

To test the modifications, several simple MIPS programs were written to ensure the proper working of the instructions. An example of one of the programs is shown in Listing 6.1.

Listing 6.1: Example MIPS Test Assembly

```
        .data
five:
        .4byte  5
        .globl five

        .text
_start:
        ori     $8, $0, 0x2     #stores 2 in Reg8
                                #(Reg 0 must be = 0)
        ori     $9, $0, 0x3     #stores 3 in Reg9
        addu    $10, $8, $9     #stores 5 in Reg10
        addi    $11, $10, 0x1   #stores 6 in Reg11
        addiu   $12, $11, 0x4   #stores 10 in Reg12
        sub     $13, $12, $11   #stores 4 in Reg13
        sub     $14, $11, $12   #stores -4 in Reg14
        subu    $15, $11, $12   #stores -4 in Reg15
        subu    $16, $12, $11   #stores 4 in Reg16
        and     $17, $8, $9     #stores "0...0010" in Reg17
        andi    $18, $9, 0x0    #stores 0 in Reg18
        or      $19, $8, $9
        xor     $20, $8, $9
        add     $22, $8, $9
```

There are some things to note about the example code. The variable *five* in *.data* is not necessary for the program and can be removed. However, the current COFFEE linker will give an error if there is no data present. Additionally, the expected value saved into some registers is dependent on Register 0 containing the value zero, such as the first *ori* instruction. All registers are initialized with the value zero as a default.

In addition to the above instructions, the LW and SW (Load Word and Store Word) instructions were also implemented. These instructions require full 32 bit addresses to access memory, which usually requires the LUI (Load Upper Immediate) instruction. Since the LUI instructions in MIPS and COFFEE do not match, this instruction was not implemented and the LW and SW instructions can not be used.

6.2 Results

The modifications made to the DECODER and COFFEE Control Unit have resulted in a few working MIPS instructions. The table below shows the instructions that function according to the MIPS specifications and were positively tested using MIPS assembly test programs.

Table 6.2: Implemented and Working Instructions

MIPS Instruction	**Definition**
OR	Bitwise Logical OR
ORI	OR Immediate
XOR	Exclusive OR
AND	Bitwise Logical AND
ANDI	AND Immediate
ADD	Addition
ADDU	ADD Unsigned
ADDI	ADD Immediate
ADDIU	ADD Immediate Unsigned
SUB	Subtract
SUBU	SUB Unsigned

The GPR 0 register is assumed to always be zero to a MIPS compiler. Since this is not necessarily the case in COFFEE, a case where GPR 0 is not zero can result in an error in a running program. Additionally, the LW and SW (load word and store word) instructions can not be properly used without a properly working LUI (load upper immediate) instruction, which is needed to load from or save to a 32 bit memory address and are thus not included in the list of positively tested instructions.

6.3 Discussion

The results have shown that with minimal modifications made to the COFFEE Core, MIPS instructions can be implemented and function according to MIPS specifications on the COFFEE core. As the modifications were limited to only the DECODER stage and the COFFEE Control Unit, it is expected that expanding the modifications to include other sections of the processor would allow even more MIPS instructions to function.

The results have also shown that some complications arise from the differences between the MIPS and COFFEE processors. The MIPS compiler expected that the GPR 0 register is always zero. This is not the case in COFFEE as the contents of GPR 0 can be changed at any time. This means that the program can fail if the program requires that GPR 0 is zero and it is in fact not.

In addition to a dependency on GPR 0, the results of testing also show that some instructions have a dependency on other instructions, such as is the case for LUI, LW, and SW. The LUI instruction is needed to load from or save to a 32 bit memory address.

By making small modification to the COFFEE core, a small number of MIPS instructions can be made to work correctly on the COFFEE core. For more MIPS instructions to work on the COFFEE core, modifications will need to be made outside the DECODE stage and Control Unit. Additional components and registers and modifications to the ALU will need to be made. From the research and design process, it can be assumed that as more instructions are added, more work will be required to modify the processor and more complications will arise.

6.4 Further Work

The next steps to accomodate the MIPS instructions in the COFFEE core will require modifications throughout the COFFEE Core in addition to modifications made in the DECODE and Control Unit. The following are suggestions as to the next steps which should cover most of the remaining MIPS instructions.

- Similar Instructions - The remaining similar instructions such as the Shift and Rotate instructions can be implemented
- Zero Register - GPR 0 can be tied to low so that the value can never be modified
- Load Upper Immediate - The LUI instruction can be implemented for load and store instructions
- Registers - Registers HI and LO can be added for storing results of multiplications and divisions
- Load and Store - Load and store instructions implemented for halfwords and bytes
- Combined Instructions - Instructions such as MADD (multiply and add) and MSUB (multiply and subtract) can be implemented
- Compare Combination Instructions - Instructions combined with compare, such as compare and branch or compare and trap can be implemented as there are several versions of these instructions

- Cache - Cache access and cache synchronization instructions can be implemented
- Privileged Interface - the COFFEE superuser mode and privileged register set can be modified to resemble the MIPS CP0 privileged interface, thus affecting traps, interrupts, exceptions, system calls, etc

7. CONCLUSIONS

The COFFEE Core is an open-source soft-core processor that is easy to modify and distribute. The challenge is that if the COFFEE core were widely distributed, the software would need constant updating to maintain an acceptable standard. The feasibility of adapting the COFFEE Core to read the MIPS instruction set was investigated.

After a comparison of the MIPS and ARM architectures, MIPS was chosen for its simplicity and similarity of the instruction set to the COFFEE instruction set. It was decided to modify the implementation of the COFFEE Core to accomodate as many MIPS instructions as possible while leaving the overall architecture of the COFFEE Core intact.

A small subset of MIPS instructions was implemented by modifying only the decoding and control elements of the processor. However, comparisons made between the instruction sets and architectures of MIPS and COFFEE reveal that implementing additional instructions would require changes in the core outside the decoding and control elements and may require altering the architecture of the COFFEE Core to be more similar to the MIPS architecture.

Thus, a MIPS compatible COFFEE Core is possible if additional modifications are made to the COFFEE Core to accomodate the remaining MIPS instructions.

REFERENCES

[1] GCC, the GNU compiler collection, April 2013.

[2] GNU binutils, April 2013.

[3] ARM Infocenter. Migrating from MIPS to ARM, April 2013.

[4] ARM Limited. *ARM Architecture Reference Manual*, ARMv7-a and ARMv7-r edition ed. 110 Fulbourn Road, Cambridge, Great Britain, 2011.

[5] ARM Limited. *ARMv8 Instruction Set Overview*, prd03-genc-010197 15.0 ed. 110 Fulbourn Road, Cambridge, Great Britain, November 2011.

[6] ARM Limited. ARM processor architecture, April 2013.

[7] ARM Limited. Software tools, April 2013.

[8] Chow, P. RISC-(reduced instruction set computers). *Potentials, IEEE 10*, 3 (1991), 28–31.

[9] COFFEE Project. *COFFEE Instruction Encoding.* Tampere University of Technology, Tampere, Finland.

[10] COFFEE Project. *COFFEE Instruction Set Summary.* Tampere University of Technology, Tampere, Finland.

[11] COFFEE Project. *COFFEE Instruction Specifications.* Tampere University of Technology, Tampere, Finland.

[12] COFFEE Project. *COFFEE Core User Manual.* Tampere University of Technology, Tampere, Finland, July 2007.

[13] COFFEE Project. *COFFEE Core Control.* Tampere University of Technology, Tampere, Finland, February 2013.

[14] COFFEE Project. COFFEE RISC core - a core for free, April 2013.

[15] Corporation, A. Advanced architecture optimizes the Atmel AVR CPU, April 2013.

[16] Kylliainen, J., Nurmi, J., and Kuulusa, M. Coffee - a core for free. In *System-on-Chip, 2003. Proceedings. International Symposium on* (2003), pp. 17–22.

[17] MIPS Technologies, I. Tools & software, April 2013.

[18] MIPS Technologies, Inc. *MIPS Architecture For Programmers Volume I-A: Introduction to the MIPS32 Architecture*, 3.02 ed. 955 East Arques Avenue, Sunnyvale, CA, March 2011.

[19] MIPS Technologies, Inc. *MIPS Architecture For Programmers Volume II-A: The MIPS32 Instruction Set*, 3.02 ed. 955 East Arques Avenue, Sunnyvale, CA, March 2011.

[20] MIPS Technologies, Inc. *MIPS Architecture For Programmers Volume III: The MIPS32 and microMIPS32 Privileged Resource Architecture*, 3.12 ed. 955 East Arques Avenue, Sunnyvale, CA, April 2011.

[21] MIPS Technologies, Inc. MIPS processors, April 2013.

[22] Patterson, D. A. Reduced instruction set computers. *Commun. ACM 28*, 1 (Jan. 1985), 8–21.

A. APPENDIX

A.1 MIPS Instructions with Identical COFFEE Instruction Specifications

These instructions had similar specifications in MIPS and COFFEE. Most can be implemented directly, whereas some may need minimal modifications.

MIPS Instruction	Instruction Type	Instruction Description	COFFEE Equivalent
ADDI	CPU Arithmetic	Add (with overflow)	ADD
ADDI	CPU arithmetic	Add immediate (with overflow)	ADDI
ADDIU	CPU arithmetic	Add immediate unsigned (no overflow)	ADDIU
ADDU	CPU arithmetic	Add unsigned (no overflow)	ADDU
AND	CPU Logical	Bitwise AND	AND
ANDI	CPU Logical	Bitwise and immediate	ANDI
DI	Privileged	Disable interrupts	DI
EI	Privileged	enable interrupts	EI
EXT	CPU Insert/Extract	extract bit field	EXBF
JAL	CPU Branch and Jump	Jump and link	JAL
JALR	CPU Branch and Jump	Jump and link register	JALR
J	CPU Branch and Jump	Jump	JMP
JR	CPU Branch and Jump	Jump register	JMPR
LW	CPU Load, Store, and Memory Control	Load word	LD

MUL	CPU arithmetic	Multiply Word to GPR	MULS
NOP	CPU Instruction Control	no operation	NOP
NOR	CPU Logical	Bitwise NOR	NOR
OR	CPU Logical	Bitwise OR	OR
ORI	CPU Logical	Bitwise OR immediate	ORI
SUB	CPU arithmetic	Subtract	SUB
SUBU	CPU arithmetic	Subtract unsigned	SUBU
XOR	CPU Logical	Bitwise exclusive OR	XOR

A.2 MIPS Instructions with Similar COFFEE Instruction Specification or Simple to Implement

These MIPS instructions have similar instructions to COFFEE or functionality that could be more easily implemented in COFFEE, but the COFFEE Core would need additional modification to function.

MIPS Instruction	Instruction Type	Instruction Description	COFFEE Equivalent and Notes
SLTU	CPU arithmetic	Set on less than unsigned	none
INS	CPU Insert/Extract	Insert bit field	none
WSBH	CPU Insert/Extract	Word swap bytes within halfword	none
MTHI	CPU Move	Move to HI Register	none (HI register does not exist)
LUI	CPU Logical	Load upper immediate	none
XORI	CPU Logical	Bitwise exclusive or immediate	none
BAL	CPU Branch and Jump	Branch and link	none
MOVN	CPU Move	Move conditional on not zero	ALU compare with r0 + ld/move

MOVZ	CPU Move	Move conditional on zero	ALU compare with r0 + ld/move
SSNOP	CPU Instruction Control	Superscalar No Operation	equivalent to SLL, R0, R0, 1
LH	CPU Load, Store, and Memory Control	Load Halfword	none, not immediate, similar LLI (no sign extension)
LHU	CPU Load, Store, and Memory Control	Load Halfword Unsigned	none, not immediate, similar LLI (no unsigned, no sign extension)
MULT	CPU arithmetic	Multiply Word to HI/LO 64 bit result	MULS + MULHI
MULTU	CPU arithmetic	Multiply unsigned to HI/LO 64 bit result	MULU + MULHI
SYSCALL	CPU Trap System Call	System Call	none, SCALL similar, MIPS causes an exception, whereas COFFEE changes PC, PSR
BEQ	CPU Branch and Jump	Branch on equal	BEQ
B	CPU Branch and Jump	Unconditional Branch	BEQ r0, r0, offset
BNE	CPU Branch and Jump	Branch on not equal	BNE
BGEZ	CPU Branch and Jump	Branch on greater than or equal to zero	none, similar BEGT, r0
BGEZAL	CPU Branch and Jump	Branch on greater than or equal to zero and link	none, similar BEGT, r0 + link
BLEZ	CPU Branch and Jump	Branch on less than or equal to zero	none, similar BELT, r0

BGTZ	CPU Branch and Jump	Branch on greater than zero	none, similar BGT, r0
BLTZ	CPU Branch and Jump	Branch on less than zero	none, similar BLT, r0
BLTZAL	CPU Branch and Jump	Branch on less than zero and link	none, similar BLT, r0 + link
SLT	CPU arithmetic	Set on less than (signed)	none, similar compare + ld
SEH	CPU arithmetic	Sign-extend halfword	none, similar sexti
SEB	CPU arithmetic	Sign-extend byte	none, similar sexti
SLTI	CPU arithmetic	Set on less than immediate (signed)	none, no immediate load
SLTIU	CPU arithmetic	Set on less than immediate unsigned	none, no immediate, but implementable?
ERET	Privileged	exception return	RETI, except reti requires to be followed by 2 NOPs
SLLV	CPU Shift	Shift left logical variable	SLL, 5 shift bits instead of 6
SLL	CPU Shift	Shift left logical	SLLI, 5 shift bits instead of 6
SRAV	CPU Shift	Shift word right arithmetic variable	SRA, 5 shift bits instead of 6
SRA	CPU Shift	Shift word right arithmetic	SRAI, 5 shift bits instead of 6
SRLV	CPU Shift	Shift right logical variable	SRL, 5 shift bits instead of 6
SRL	CPU Shift	Shift right logical	SRLI, 5 shift bits instead of 6
SW	CPU Load, Store and Memory Control	Store word	ST
TEQ	CPU Trap	Trap if equal	none, similar TRAP + ALU compare

TGE	CPU Trap	Trap if greater or equal	none, similar TRAP + compare
TGEI	CPU Trap	Trap if greater of equal immediate	none, similar TRAP + compare
TGEIU	CPU Trap	Trap if greater or equal immediate Unsigned	none, similar TRAP + compare
TGEU	CPU Trap	Trap if greater or equal unsigned	none, similar TRAP + compare
TLT	CPU Trap	Trap if less than	none, similar TRAP + compare
TLTI	CPU Trap	Trap if less than immediate	none, similar TRAP + compare
TLTIU	CPU Trap	Trap if less than immediate unsigned	none, similar TRAP + compare
TLTU	CPU Trap	Trap if less than unsigned	none, similar TRAP + compare
TNE	CPU Trap	Trap if not equal	none, similar TRAP + compare
TNEI	CPU Trap	Trap if not equal immediate	none, similar TRAP + compare
TEQI	CPU Trap	Trap if equal immediate	none, similar TRAP + immediate compare
SC	CPU Load, Store, and Memory Control	Store conditional word	none, works with LL + SC (mips)
RDHWR	CPU Move	Read Hardware Register	none

A.3 MIPS Instructions with No Comparable COFFEE Instruction Match

These MIPS instructions have no comparable match in COFFEE.

MIPS Instruction	Instruction Type	Instruction Description	COFFEE Equivalent
BREAK	CPU Trap	Breakpoint	none
CACHE	Privileged	Perform cache operation	none
CLO	CPU arithmetic	Count Leading Ones in Word	none
CLZ	CPU arithmetic	Count Leading Zeros in Word	none
DIV	CPU arithmetic	Divide	none
DIVU	CPU arithmetic	Divide unsigned	none
EHB	CPU Instruction Control	Execution Hazard Barrier	none
LB	CPU Load, Store, and Memory Control	Load byte	none
LBU	CPU Load, Store, and Memory Control	Load Byte Unsigned	none
LL	CPU Load, Store, and Memory Control	Load Linked Word	none
LWL	CPU Load, Store, and Memory Control	Load world left	none
LWR	CPU Load, Store, and Memory Control	Load word right	none
MADD	CPU arithmetic	Multiply and Add Word to Hi, Lo	none
MADDU	CPU arithmetic	Multiply and Add Unsigned Word to Hi, Lo	none
MSUB	CPU arithmetic	Multiply and Subtract Word to Hi, Lo	none
MSUBU	CPU arithmetic	Multiply and Subtract Unsigned Word to Hi, Lo	none
PAUSE	CPU Instruction Control	Wait for LLBit to clear	none

ROTR	CPU Shift	Rotate word right	none
ROTRV	CPU Shift	Rotate word right variable	none
SB	CPU Load, Store, and Memory Control	Store byte	none
SH	CPU Load, Store, and Memory Control	store halfword	none
SWL	CPU Load, Store, and Memory Control	store word left	none
SWR	CPU Load, Store, and Memory Control	store word right	none
SYNC	CPU Load, Store, and Memory Control	Synchronize shared memory	none
SYNCI	CPU Load, Store, and Memory Control	Synchronize caches to make Writes effective	none
MTLO	CPU Move	Move To LO Register	none, LO register does not exist
PREF	CPU Load, Store, and Memory Control	Prefetch	none, no cache instructions
MFC0	Privileged	Move from Coprocessor 0 to GPR	none
MTC0	Privileged	Move to Coprocessor 0 from GPR	none
MFHI	CPU Move	Move from HI reg	none, no designated HI reg in COFFEE
MFLO	CPU Move	Move from LO reg	none, no designated LO reg in COFFEE
TLBP	Privileged	Probe TLB for Matching Entry	none, no direct TLB/virtual memory access for COFFEE
TLBR	Privileged	Read Indexed TLB Entry	none, no direct TLB/virtual memory access for COFFEE

TLBWI	Privileged	Write Indexed TLB Entry	none, no direct TLB/virtual memory access for COFFEE
TLBWR	Privileged	Write Random TLB Entry	none, no direct TLB/virtual memory access for COFFEE
JALR.HB	CPU Branch and Jump	Jump and Link Register, clear Hazard Barriers	none, JALR + clearhazzards
JR.HB	CPU Branch and Jump	Jump Register, clear Hazzard Barriers	none, JMPR + clearhazzards
WAIT	Privileged	Enter Standby Mode	none, no low power mode
RDPGPR	Privileged	Read GPR from Previous Shadow Set	none, no shadow sets in COFFEE
WRPGPR	Privileged	Write GPR to Previous Shadow Set	none, no shadow sets in COFFEE