02132 Assignment 2 report

Hardware implementation in Chisel of a small CPU running the image erosion

Group: 22

Mikkel Arn Andersen **s224187**
Niclas Juul Schæffer **s224744**
Rasmus Kronborg Finnemann Wiuff **s163977**
github.com/rwiuff/02132Assignment2

November 5th

1 Work distribution

Table 1 shows the work distribution in the group for this project.

Table 1: *Work distribution on the project*

Name	Development tasks	Report tasks
Mikkel Arn Andersen Niclas Juul Schæffer Rasmus Wiuff	ISA	Section 2.1

2 Design

2.1. ISA and encoding

The ISA instructions are inspired from Appendix A in the assignment description. It is listed in Table 2.

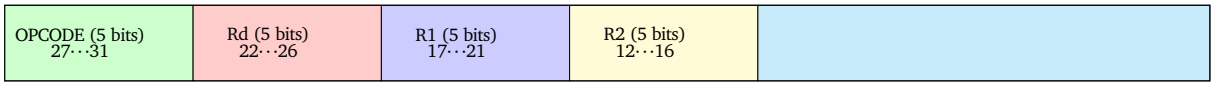
Table 2: *Instruction-set architecture used in the assignment*

Instruction	**Syntax**	**Meaning**
	Arithmetic instructions	
Addition	`ADD Rx, Ry, Rz;`	`Rx = Ry + Rz`
Subtraction	`SUB Rx, Ry, Rz;`	`Rx = Ry - Rz`
Immediate addition	`ADDI Rx, Ry, z;`	`Rx = Ry + z`
Immediate subtraction	`SUBI Rx, Ry, z;`	`Rx = Ry - z`
Multiplication	`MUL Rx, Ry, Rz;`	`Rx = Ry · Rz`
	Logic instructions	
Bitwise OR	`OR Rx, Ry, Rz;`	`Rx = Ry | Rz`
Bitwise AND	`AND Rx, Ry, Rz;`	`Rx = Ry & Rz`
Bitwise NOT	`NOT Rx, Ry;`	`Rx = ~Ry`
	Memory instructions	
Load immediate	`LI Rx, y;`	`Rx = y`
Load data	`LD Rx, Ry;`	`Rx = memory(Ry)`
Store data	`SD Rx, Ry;`	`memory(Ry) = Rx`
	Control and flow instructions	
Jump	`JMP x`	`GOTO INST x`
Jump if equal	`JEQ Rx, Ry, z;`	`if(Rx == Ry) GOTO INST z`
Jump if less than	`JLT Rx, Ry, z;`	`if(Rx < Ry) GOTO INST z`
Jump if greater than	`JGT Rx, Ry, z;`	`if(Rx > Ry) GOTO INST z`
Do nothing	`NOP;`	`No operation`
END	`END;`	`Terminate`

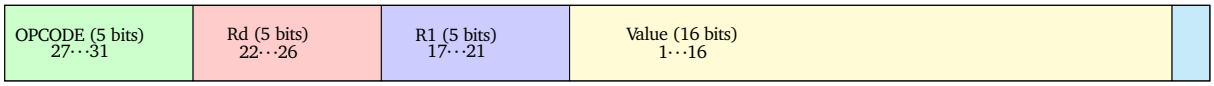
To design the instructions, first the bit sizes are considered. Some are given in the assignment. If there are 16 registers, these can be reached with $\log_2 16 = 4$ bits. Values for the logic and arithmetic operations are 16 bit as well as addresses in the memory. The opcodes fit within 5 bits.
The instructions have 3 types: Register type, immediate type and jump type. They are laid out as shown in Fig. 1.

Figure 1: *Instruction layouts.*

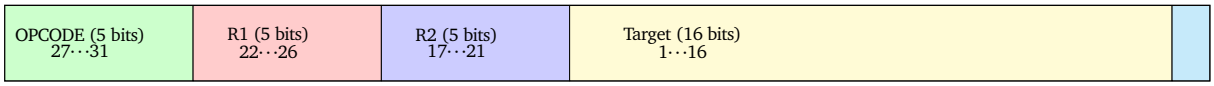
(a) *Register type instruction. R1 and 2 are operands, Rd is the destination register.*



(b) *Immediate type instruction. R1 is an operand. Rd is the destination register. Value is the other operand.*



(c) *Jump type instruction. R1 and R2 are operands. Target is the target address.*



2.1.1. Opcodes As seen in Fig. 1 there are 5 bits allocated to opcodes. Table 2 accounts for eight register type operations, four jump types, three immediate types and two runtime operations. The encoding scheme uses significant bits to determine the type of operation. The first bit tells if the instruction is runtime or not. Second bit tells if the instruction is of register type or jump/immediate type. For the latter the third bit determines if the instruction is of jump or immediate type. The scheme is laid out in Table 3. Using bits to determine the instruction type, designing the chip decoding becomes easier.

Table 3: *OPCODE instruction bits.*

Instruction type	OPCODE bits	Instruction
Register	00000	ADD
	00001	SUB
	00010	MUL
	00011	OR
	00100	AND
	00101	NOT
	00110	LD
	00111	SD
Jump	01000	JMP
	01001	JEQ
	01010	JLT
	01011	JGT
Immediate	01100	ADDI
	01101	SUBI
	01110	LI
Runtime	10000	NOP
	11111	END

2.2. Compiled program

2.3. CPU block

3 Implementation

Briefly discuss the implementation in Chisel of your design. You can include some code snippets if these are relevant to explain certain aspects of the implementation. In other words, try to answer the question "What does a reader need to know about your Chisel implementation?"

4 Test and analysis

Report here the results from the test you have carried out. Present the test you have developed (if any). Remember to discuss the results and the test you have carried out, do not just present them, but explain and argue their meaning. Address the design evaluation questions listed in Task 11 in the Assignment 2 document.

References

[1] Arduino, José Bagur, Taddy Chung *Arduino Memory Guide (19/09/2023)*
https://docs.arduino.cc/learn/programming/memory-guide