

# 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](https://github.com/rwiuff/02132Assignment2) 

November 5<sup>th</sup>

### 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

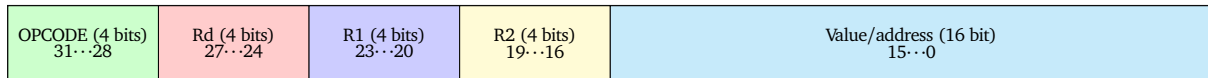
The ISA instructions are inspired from Appendix A in the assignment description. It is listed in Table 2.

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;	$Rx = Ry - z$
Immediate multiplication	MULT Rx, Ry, z;	$Rx = Ry \cdot z$
Logic instructions		
Bitwise OR	OR Rx, Ry, Rz;	$Rx = Ry \mid Rz$
Bitwise AND	AND Rx, Ry, Rz;	$Rx = Ry \& Rz$
Memory instructions		
Load immediate	LOADI Rx, y;	$Rx = y$
Load data	LOAD Rx, Ry;	$Rx = \text{memory}(Ry)$
Store data	STORE Rx, Ry;	$\text{memory}(Ry) = Rx$
Move data	MOVE Rx, Ry;	$Rx = Ry$
Control and flow instructions		
Jump	JMP x	GOTO INST x
Jump if equal	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 4 bits. The instruction layout is laid out in Fig. 1.

**Figure 1:** Instruction layout. *R1* and *2* are operands, *Rd* is the destination register. Remaining bits are used for either memory address or immediate value.



**2.1.1. Opcodes** As seen in Fig. 1 there are 4 bits allocated to opcodes. Table 2 accounts for seven register type operations, four jump types, three immediate types and two runtime operations.

**Table 3:** OPCODE instruction bits.

OPCODE bits	Instruction
0001	ADD
0010	SUB
0011	ADDI
0100	SUBI
0101	MULT
0110	OR
0111	AND
1000	LOADI
1001	LOAD
1010	STORE
1011	MOVE
1100	JMP
1101	JGT
0000	NOP
1111	END

## 2.2. COMPILE AND ENCODE

*Listing 1: The program compiled to assembly*

```
0      # Initial values
1  LOADI R0, 0;      # x counter
2  LOADI R1, 0;      # y counter
3  LOADI R2, 19;     # Pixel limit
4  LOADI R3, 0;      # Zero value
5  LOADI R4, 255;    # 255 value
6      # For loop conditions
7  JEQ R0, R2, 58     # Check x, GOTO END
8  JEQ R1, R2, 54     # Check y, GOTO INC X
9      # Output image adress
10 MULT R5, R1, 20;   # y * 20
11 ADD R6, R0, R5;    # x + y * 20
12 ADDI R5, R6, 400;  # Out image address
13      # Process border pixel
14 JEQ R0, R3, 48     # If x or y = 0
15 JEQ R1, R3, 48     #
16 JEQ R0, R4, 48     # If x or y = 19
17 JEQ R1, R4, 48     # GOTO erosion
18      # Process inner pixel
19 MULT R6, R1, 20;   # y * 20
20 ADD R7, R0, R6;    # x + y * 20
21 LOAD R8, R7;       # Get input pixel
22 JEQ R8, R3, 48;    # If 0, GOTO erosion
23      # Process outer pixels
24 SUBI R12, R0, 1    # x - 1
25 ADDI R13, R0, 1    # x + 1
26 SUBI R14, R1, 1    # y - 1
27 ADDI R15, R1, 1    # y + 1
28 MULT R6, R1, 20;   # y * 20
29 ADD R7, R6, R12;   # (x - 1) + y * 20
30 LOAD R8, R7;       # Save pixel in R8
31 MULT R6, R1, 20;   # y * 20
32 ADD R7, R6, R13;   # (x + 1) + y * 20
33 LOAD R9, R7;       # Save pixel in R9
34 OR R10, R8, R9;    # OR R8 and R9, save to R10
35 MULT R6, R14, 20;  # (y - 1) * 20
36 ADD R7, R6, R0;    # x + (y - 1) * 20
37 LOAD R8, R7;       # Save pixel in R8
38 OR R9, R8, R10;    # OR R8 nad R10, save to R9
39 MULT R6, R15, 20;  # (y + 1) * 20
40 ADD R7, R6, R0;    # x + (y + 1) * 20
41 LOAD R10, R7;      # Save pixel in R10
42 OR R8, R9, R10;    # OR R9 and R10, save to R8
43 JEQ R8, R3, 48;    # If = 0 GOTO Erosion
44      # No erosion
45 STORE R4, R5;      # Set pixel to 255
46 JMP 51;            # GOTO increment y
47      # Erosion
48 STORE R3, R5;      # Set Pixel to zero
49 JMP 51;            # GOTO increment y
50      # Increment y
51 INC R1;            # Increment y
52 JMP 8;             # Continue nested loop
53      # Increment x
54 INC R0;            # Increment x
55 LOADI R1, 0;       # Zerorise y
56 JMP 7;             # Continue main loop
57      # Terminate program
58 END;              # Terminate program
```

## Page 4 of 5

### 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>