


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.

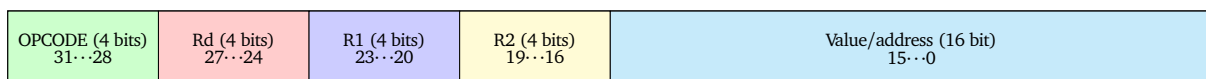
Table 2: Instruction-set architecture used in the assignment

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;	$Rx = Ry - z$
Immediate multiplication	MULI Rx, Ry, z;	$Rx = Ry \cdot z$
Increment	INC, Rx	$Rx = Rx + 1$
Logic instructions		
Bitwise OR	OR Rx, Ry, Rz;	$Rx = Ry \mid Rz$
Bitwise AND	AND Rx, Ry, Rz;	$Rx = Ry \& Rz$
Bitwise NOT	NOT Rx, Ry;	$Rx = \sim Ry$
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	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 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.

Instruction type	OPCODE bits	Instruction
Register	0001	ADD
	0010	SUB
	0011	OR
	0100	AND
	0101	NOT
	0110	LD
	0111	SD
Jump	1000	JMP
	1001	JEQ
	1010	JLT
	1011	JGT
Immediate	1100	ADDI
	1101	SUBI
	1110	LI
Runtime	0000	NOP
	1111	END

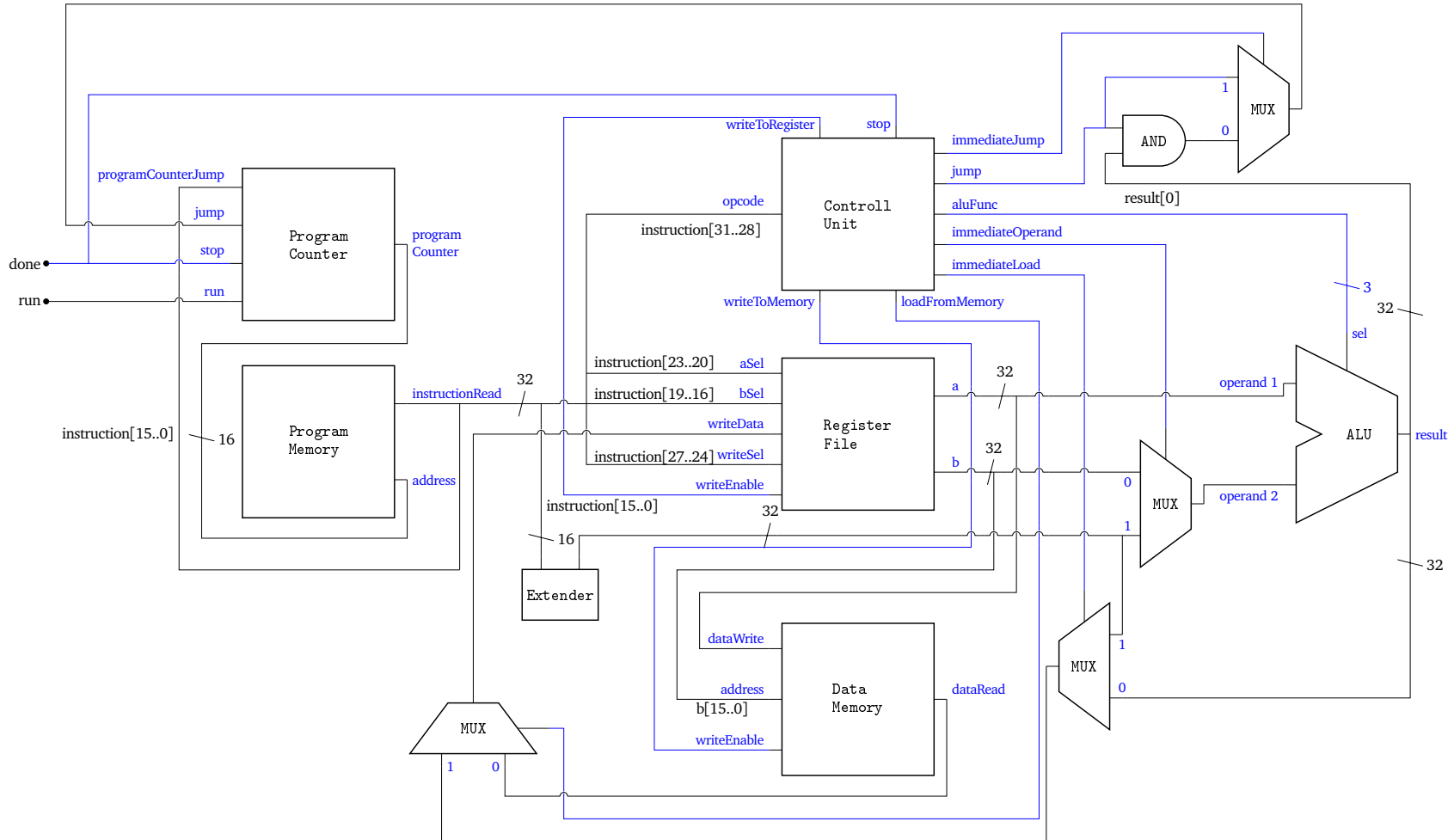
2.2. COMPILE AND ENCODE

Listing 1: The program compiled to assembly

```
0  LOADI R0, 0; // Initial values; x counter
1  LOADI R1, 0; // y counter
2  LOADI R2, 19; // Pixel limit
3  LOADI R3, 0; // Zero value
4  LOADI R4, 255; // 255 value
5  JEQ R0, R2, END // For conditions; Check x, GOTO program termination
6  JEQ R1, R2, INCX // Check y, GOTO incrementation of x
7  JEQ R0, R3, ZEROPIXEL // Process border pixel; If x = 0
8  JEQ R1, R3, ZEROPIXEL // If y = 0
9  JEQ R0, R4, ZEROPIXEL // If x = 255
10 JEQ R1, R4, ZEROPIXEL // If y = 255
11 MULI R5, R1, 20; // y * 20
12 ADD R6, R0, R5; // x + y * 200
13 LOAD R8, R6; // Get input image pixel
14 JEQ R8, R3, BLACK_PIXEL; // Check if 0, GOTO pixel zeroing
15 SUBI R9, R0, 1 // x - 1
16 ADDI R10, R0, 1 // x + 1
17 SUBI R11, R1, 1 // y - 1
18 ADDI R12, R1, 1 // y + 1
19 MULI R5, R1, 20; // y * 20
20 ADD R6, R9, R5; // (x - 1) + y * 200
21
22 MULI R5, R1, 20; // y * 20
23 ADD R6, R10, R5; // (x + 1) + y * 200
24
25 MULI R5, R11, 20; // (y - 1) * 20
26 ADD R6, R0, R5; // x + (y - 1) * 200
27
28 MULI R5, R12, 20; // (y + 1) * 20
29 ADD R6, R0, R5; // x + (y + 1) * 200
30
31
32
33
34 MULI R5, R1, 20; // y * 20
35 ADD R6, R0, R5; // x + y * 200
36 ADDI R7, R6, 400; // Offset to output image
37 STORE R3, R7; // Set Pixel to zero
38 JMP INCY; // Continue program after border process
39
40 MULI R5, R1, 20; // y * 20
41 ADD R6, R0, R5; // x + y * 200
42 ADDI R7, R6, 400; // Offset to output image
43 STORE R3, R7; // Set Pixel to zero
44 JMP 11; // Continue program after border process
45
46 INC R1; // Increment y
47 JMP 6; // Continue nested loop
48 INC R0; // Increment x
49 JMP 5; // Continue main loop
50 END; // Terminate program
```

2.3. CPU BLOCK

Figure 2: Block diagram of the CPU architecture. Blue lines are control signals.



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>