

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 12th

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

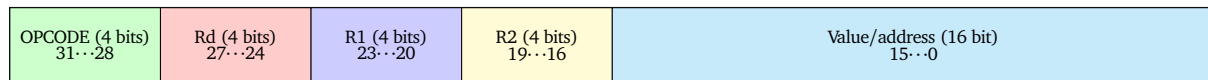
The ISA instructions are inspired from Appendix A in the assignment description. Chosen instructions are listed in Table 2.

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;	$Rx = Ry \cdot z$
Increment	INC, Rx	$Rx = Rx + 1$
Logic instructions		
Bitwise OR	OR Rx, Ry, Rz;	$Rx = Ry \mid Rz$
Bitwise AND	AND Rx, Ry, Rz;	$Rx = Ry \& Rz$
Memory instructions		
Load immediate	LOADI Rx, y;	$Rx = y$
Load data	LOAD Rx, Ry;	$Rx = \text{memory}(Ry)$
Store data	STORE Rx, Ry;	$\text{memory}(Ry) = Rx$
Control and flow instructions		
Jump	JMP x	GOTO INST x
Jump if equal	JEQ Rx, Ry, z;	if($Rx > Ry$) GOTO INST z
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	OPCODE bits	Instruction
0001	ADD	1000	LOADI
0010	SUB	1001	LOAD
0011	ADDI	1010	STORE
0100	SUBI	1011	INC
0101	MULT	1100	JMP
0110	OR	1101	JEQ
0111	AND	1111	END

2.2. COMPILE AND ENCODE

2.2.1. Compiled to assembler Listing 1 shows the erosion algorithm compiled to assembler using the ISA provided in Table 2.

2.2.2. Encoding the program The program in Listing 1 is encoding using the opcodes in Table 3 and instruction scheme in Fig. 1 to the machine code in Listing 2.

Listing 1: The program compiled to assembly

```

# Initial values
00. LOADI R0, 0;      # x counter
01. LOADI R1, 0;      # y counter
02. LOADI R2, 19;     # Pixel limit
03. LOADI R3, 0;      # Zero value
04. LOADI R4, 255;    # 255 value

# For loop conditions
05. JEQ R0, R2, 47;   # Check x, GOTO END
06. JEQ R1, R2, 44;   # Check y, GOTO INC X

# Output image address
07. MULT R5, R1, 20;  # y * 20
08. ADD R6, R0, R5;   # x + y * 20
09. ADDI R5, R6, 400; # Out image address

# Process border pixel
10. JEQ R0, R3, 40;   # If x or y = 0
11. JEQ R1, R3, 40;   #
12. JEQ R0, R4, 40;   # If x or y = 19
13. JEQ R1, R4, 40;   # GOTO erosion

# Process inner pixel
14. MULT R6, R1, 20;  # y * 20
15. ADD R7, R0, R6;   # x + y * 20
16. LOAD R8, R7;      # Get input pixel
17. JEQ R8, R3, 40;   # If 0, GOTO erosion

# Process outer pixels
18. SUBI R12, R0, 1;  # x - 1
19. ADDI R13, R0, 1;  # x + 1
20. SUBI R14, R1, 1;  # y - 1
21. ADDI R15, R1, 1;  # y + 1
22. MULT R6, R1, 20;  # y * 20
23. ADD R7, R6, R12;  # (x - 1) + y * 20

24. LOAD R8, R7;      # Save pixel in R8
25. MULT R6, R1, 20;  # y * 20
26. ADD R7, R6, R13;  # (x + 1) + y * 20
27. LOAD R9, R7;      # Save pixel in R9
28. OR R10, R8, R9;   # OR R8 and R9, save to R10
29. MULT R6, R14, 20; # (y - 1) * 20
30. ADD R7, R6, R0;   # x + (y - 1) * 20
31. LOAD R8, R7;      # Save pixel in R8
32. OR R9, R8, R10;   # OR R8 nad R10, save to R9
33. MULT R6, R15, 20; # (y + 1) * 20
34. ADD R7, R6, R0;   # x + (y + 1) * 20
35. LOAD R10, R7;     # Save pixel in R10
36. OR R8, R9, R10;   # OR R9 and R10, save to R8
37. JEQ R8, R3, 40;   # If = 0 GOTO Erosion

# No erosion
38. STORE R4, R5;     # Set pixel to 255
39. JMP 42;           # GOTO increment y

# Erosion
40. STORE R3, R5;     # Set Pixel to zero
41. JMP 42;           # GOTO increment y

# Increment y
42. INC R1;           # Increment y
43. JMP 6;            # Continue nested loop

# Increment x
44. INC R0;           # Increment x
45. LOADI R1, 0;      # Zerorise y
46. JMP 5;            # Continue main loop

# Terminate program
47. END;              # Terminate program

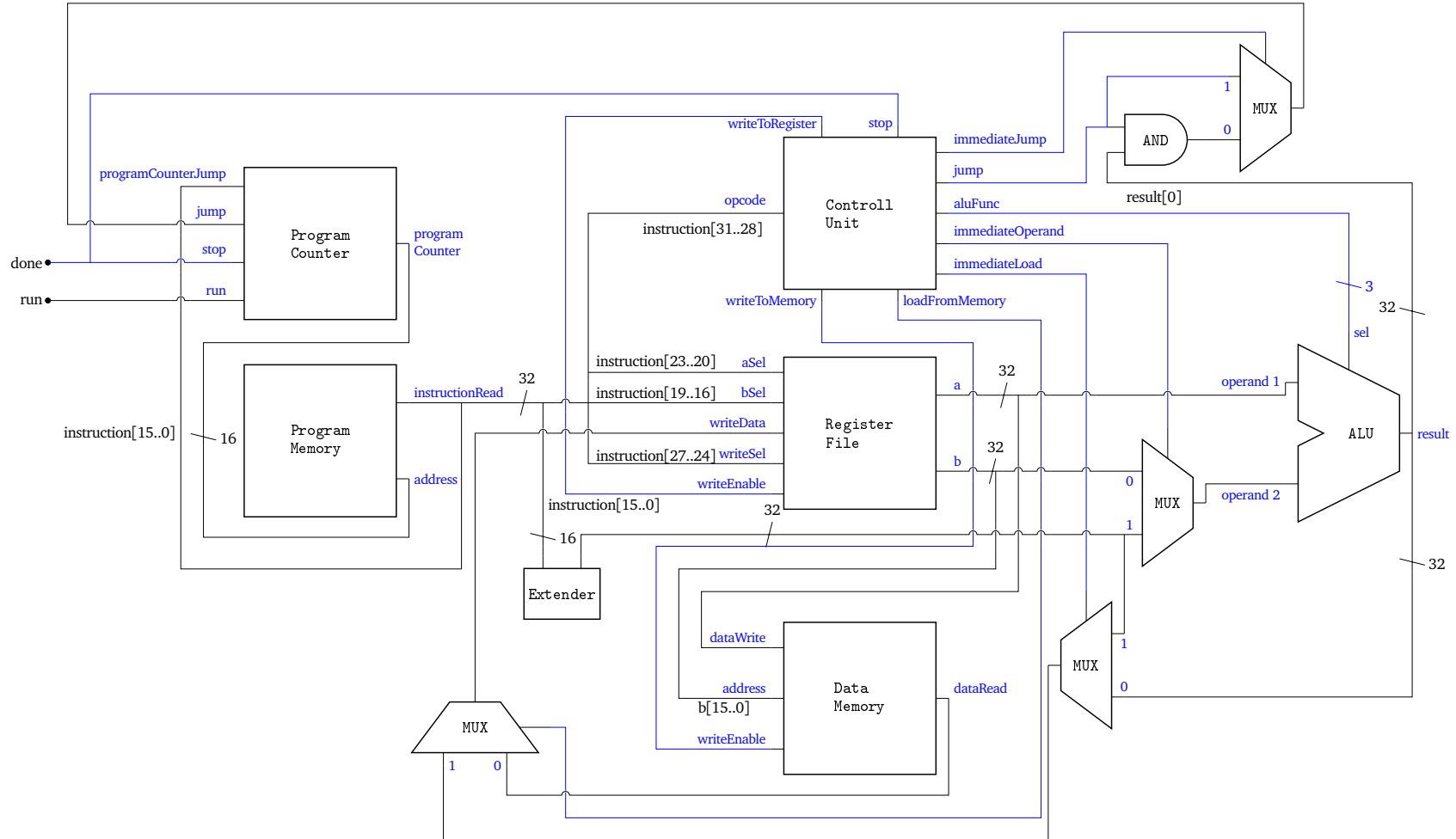
```

Listing 2: The encoded program

OP	Rd	R1	R2	Value/address	OP	Rd	R1	R2	Value/address
00.	1000	0000	0000	0000000000000000	24.	1001	1000	0000	0111 0000000000000000
01.	1000	0001	0000	0000000000000000	25.	0101	0110	0001	0000 000000000010100
02.	1000	0010	0000	00000000000010011	26.	0001	0111	0110	1101 0000000000000000
03.	1000	0011	0000	0000000000000000	27.	1001	1001	0000	0111 0000000000000000
04.	1000	0100	0000	00000000111111111	28.	0110	1010	1000	1001 0000000000000000
05.	1101	0000	0000	0010 0000000000101111	29.	0101	0110	1110	0000 000000000010100
06.	1101	0000	0001	0010 0000000000101100	30.	0001	0111	0110	0000 0000000000000000
07.	0101	0101	0001	0000 000000000010100	31.	1001	1000	0000	0111 0000000000000000
08.	0001	0110	0000	0101 0000000000000000	32.	0110	1001	1000	1010 0000000000000000
09.	0011	0101	0110	0000 0000110010000	33.	0101	0110	1111	0000 000000000010100
10.	1101	0000	0000	0011 0000000000101000	34.	0001	0111	0110	0000 0000000000000000
11.	1101	0000	0001	0011 0000000000101000	35.	1001	1010	0000	0111 0000000000000000
12.	1101	0000	0000	0100 0000000000101000	36.	0110	1000	1001	1010 0000000000000000
13.	1101	0000	0001	0100 0000000000101000	37.	1101	0000	1000	0011 000000000000101000
14.	0101	0110	0001	0000 000000000010100	38.	1010	0000	0100	0101 0000000000000000
15.	0001	0111	0000	0110 0000000000000000	39.	1100	0000	0000	0000 00000000101010
16.	1001	1000	0000	0111 0000000000000000	40.	1010	0000	0011	0101 0000000000000000
17.	1101	0000	1000	0011 0000000000101000	41.	1100	0000	0000	0000 0000000000101010
18.	0100	1100	0000	0000 000000000000001	42.	1011	0001	0000	0000 0000000000000000
19.	0011	1101	0000	0000 000000000000001	43.	1100	0000	0000	0000 000000000000110
20.	0100	1110	0001	0000 000000000000001	44.	1011	0000	0000	0000 0000000000000000
21.	0011	1111	0001	0000 000000000000001	45.	1000	0001	0000	0000 0000000000000000
22.	0101	0110	0001	0000 000000000010100	46.	1100	0000	0000	0000 000000000000101
23.	0001	0111	0110	1100 0000000000000000	47.	1111	0000	0000	0000 0000000000000000

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>