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 for this project.

Table 1: Work distribution on the project

Name	Development tasks	Report tasks
Mikkel Arn Andersen Niclas Juul Schæffer Rasmus Wiuff	ISA, assembler compilation, machine instruction encoding, CPU Block design, Program Counter implementation	Sections 2.1 to 2.3

2 Design

The project requirements were open to broad interpretation, so first step of the design was to put on restrictions. The following was stipulated:

1. All blocks in the assignment would be laid out as described in the material.
2. The architecture would be 32-bit
3. The instruction set architecture should be as simple as possible, but still versatile.

The next step was designing the instruction set architecture and the encoding scheme.

2.1. ISA and encoding scheme

The instruction set needs to contain basic arithmetic operations as well as logical ones for computations. Instructions are also needed for creating variables in the registers as well as reading and writing data between registers and memory. A branch (or in this report jump) instruction is needed to navigate instructions and at least one conditional branch instruction. Lastly the program needs to terminate. The chosen instructions are listed in Table 2. A total of 14 instructions are settled upon. This requires $\lceil \log_2 14 \rceil = 4$ bits for the opcode part of the instruction. As defined in the assignment, the memory block has an input of 16 bits. This, with the 4 opcode bits, leaves 12 bits for register addresses. Consequently this limits the chip to 16 registers. The instruction scheme is shown in Fig. 1. Table 3 shows how each instruction can be encoded into 32 bits. Notice how, when writing or reading to/from memory, `Rb` points to the register with the memory address.

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

OPCODE (4 bits)	Rd (4 bits)	Ra (4 bits)	Rb (4 bits)	Value/address (16 bit)
$31 \cdots 28$	$27 \cdots 24$	$23 \cdots 20$	$19 \cdots 16$	$15 \cdots 0$

***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 · z`
Increment	`INC, Rx`	`Rx = Rx + 1`
	Logic instructions	
Bitwise OR	`OR Rx, Ry, Rz;`	`Rx = Ry | 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 = memory(Ry)`
Store data	`STORE 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`
END	`END;`	`Terminate`

***Table 3:** Encoding instructions under the instruction set and encoding scheme*

Instruction	OPCODE	Rd	Ra	Rb	Value/address
`ADD R1, R2, R3;`	0001	0001	0010	0011	0000 0000 0000 0000
`SUB R1, R2, R3;`	0010	0001	0010	0011	0000 0000 0000 0000
`ADDI R1, R2, 1;`	0011	0001	0010	0000	0000 0000 0000 0001
`SUBI R1, R2, 1;`	0100	0001	0010	0000	0000 0000 0000 0001
`MULT R1, R2, 1;`	0101	0001	0010	0000	0000 0000 0000 0001
`OR R1, R2, R3;`	0110	0001	0010	0011	0000 0000 0000 0000
`AND R1, R2, R3;`	0111	0001	0010	0011	0000 0000 0000 0000
`LOADI R1, 1;`	1000	0001	0000	0000	0000 0000 0000 0001
`LOAD R1, R2;`	1001	0001	0000	0010	0000 0000 0000 0000
`STORE R1, R2;`	1010	0000	0001	0010	0000 0000 0000 0000
`INC R1;`	1011	0001	0001	0000	0000 0000 0000 0000
`JMP 1;`	1100	0000	0000	0000	0000 0000 0000 0001
`JEQ R1, R2, 1;`	1101	0000	0001	0010	0000 0000 0000 0001
`END;`	1111	0000	0000	0000	0000 0000 0000 0000

2.2. Compile and encode

2.2.1. Compiled to assembler In this project the provided algorithm is used. It is compiled by hand to assembler using the defined instruction set in Table 2. The registers are utilised in the following way. R0 and R1 contains the current x and y counter. R2 contains the picture width in pixels. R3 and R4 hold the values 0 and 255 for conditional comparison and for writing either black or white pixels to the output image addresses. R5 holds the output image address for the current pixel. The rest of the registers are used for storage during calculations.

2.2.2. Encoding the program Listing 1 is encoded to machine instructions using the instruction scheme in Table 3. Listing 2 shows the encoded program.

2.3. CPU block

The CPU block is shown in Fig. 2.

Mikkel Arn Andersen **s224187**
Niclas Juul Schæffer **s224744**
Rasmus Kronborg Finnemann Wiuff **s163977**

***Listing 1:** The program compiled to assembly*

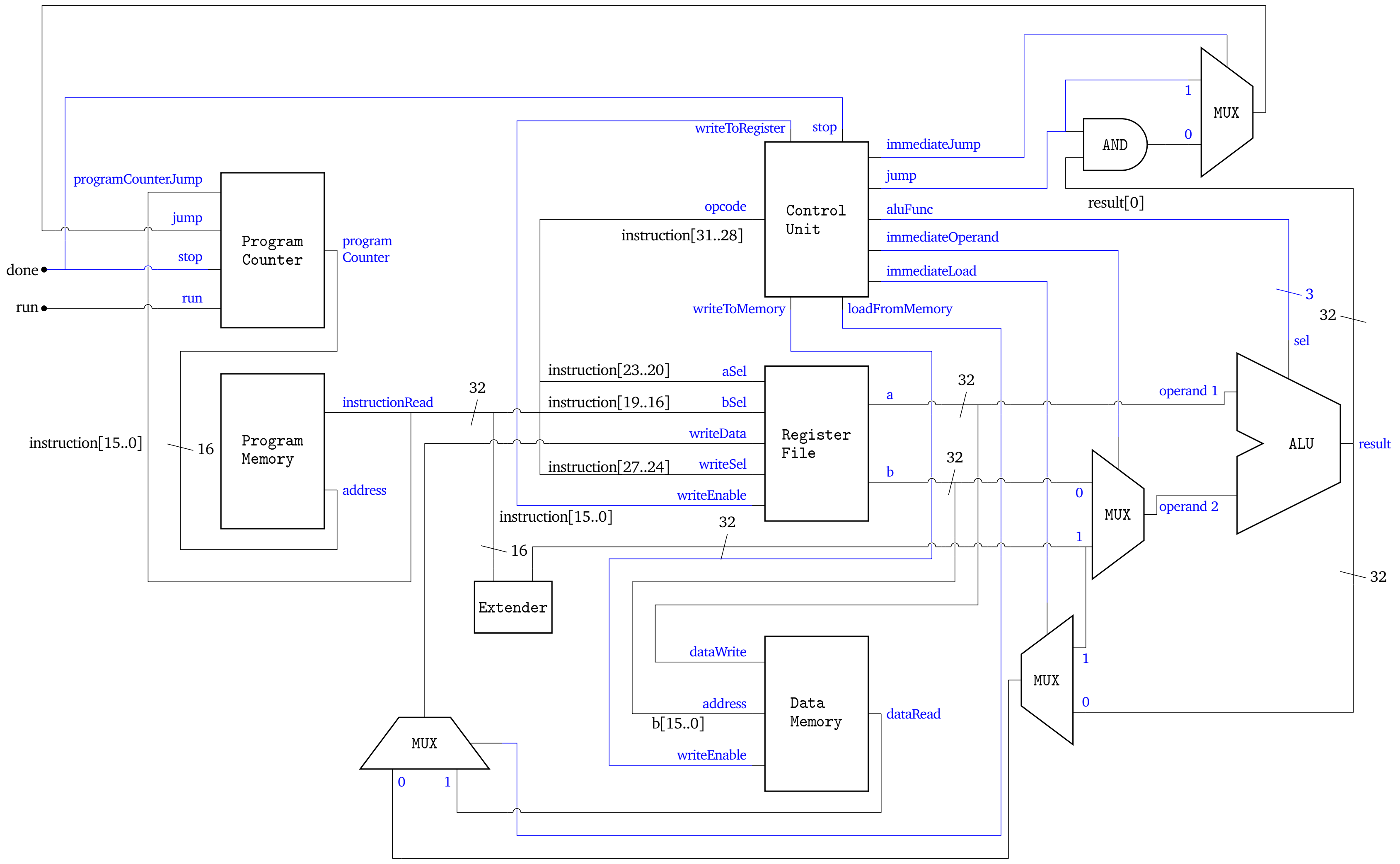
```
        # Initial values
00. LOADI R0, 0;       # x counter
01. LOADI R1, 0;       # y counter
02. LOADI R2, 19;      # Pixel limit
03. LOADI R3, 0;       # Zero value
04. LOADI R4, 255;     # 255 value
        # For loop conditions
05. JEQ R0, R2, 47;    # Check x, GOTO END
06. JEQ R1, R2, 44;    # 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. AND R10, R8, R9;   # AND 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. AND R9, R8, R10;   # AND R8 and 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. AND R8, R9, R10;   # AND 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   Ra   Rb   Value/address
00. 1000 0000 0000 0000 0000000000000000
01. 1000 0001 0000 0000 0000000000000000
02. 1000 0010 0000 0000 0000000000010011
03. 1000 0011 0000 0000 0000000000000000
04. 1000 0100 0000 0000 0000000011111111
05. 1101 0000 0000 0010 0000000000101111
06. 1101 0000 0001 0010 0000000000101100
07. 0101 0101 0001 0000 0000000000010100
08. 0001 0110 0000 0101 0000000000000000
09. 0011 0101 0110 0000 0000000110010000
10. 1101 0000 0000 0011 0000000000101000
11. 1101 0000 0001 0011 0000000000101000
12. 1101 0000 0000 0100 0000000000101000
13. 1101 0000 0001 0100 0000000000101000
14. 0101 0110 0001 0000 0000000000010100
15. 0001 0111 0000 0110 0000000000000000
16. 1001 1000 0000 0111 0000000000000000
17. 1101 0000 1000 0011 0000000000101000
18. 0100 1100 0000 0000 0000000000000001
19. 0011 1101 0000 0000 0000000000000001
20. 0100 1110 0001 0000 0000000000000001
21. 0011 1111 0001 0000 0000000000000001
22. 0101 0110 0001 0000 0000000000010100
23. 0001 0111 0110 1100 0000000000000000

    OP   Rd   Ra   Rb   Value/address
24. 1001 1000 0000 0111 0000000000000000
25. 0101 0110 0001 0000 0000000000010100
26. 0001 0111 0110 1101 0000000000000000
27. 1001 1001 0000 0111 0000000000000000
28. 0111 1010 1000 1001 0000000000000000
29. 0101 0110 1110 0000 0000000000010100
30. 0001 0111 0110 0000 0000000000000000
31. 1001 1000 0000 0111 0000000000000000
32. 0111 1001 1000 1010 0000000000000000
33. 0101 0110 1111 0000 0000000000010100
34. 0001 0111 0110 0000 0000000000000000
35. 1001 1010 0000 0111 0000000000000000
36. 0111 1000 1001 1010 0000000000000000
37. 1101 0000 1000 0011 0000000000101000
38. 1010 0000 0100 0101 0000000000000000
39. 1100 0000 0000 0000 0000000000101010
40. 1010 0000 0011 0101 0000000000000000
41. 1100 0000 0000 0000 0000000000101010
42. 1011 0001 0001 0000 0000000000000000
43. 1100 0000 0000 0000 0000000000000110
44. 1011 0000 0000 0000 0000000000000000
45. 1000 0001 0000 0000 0000000000000000
46. 1100 0000 0000 0000 0000000000000101
47. 1111 0000 0000 0000 0000000000000000
```

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



Mikkel Arn Andersen **s224187**
Niclas Juul Schæffer **s224744**
Rasmus Kronborg Finnemann Wiuff **s163977**

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