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	`MULT Rx, Ry, z;`	`Rx = Ry · z`
Increment	`INC, Rx`	`Rx = Rx + 1`
	Logic instructions	
Bitwise OR	`OR 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`
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.*

OPCODE (4 bits) 31···28	Rd (4 bits) 27···24	R1 (4 bits) 23···20	R2 (4 bits) 19···16	Value/address (16 bit) 15···0

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	`INC`
0111	`OR`
1000	`LOADI`
1001	`LOAD`
1010	`STORE`
1011	`JMP`
1100	`JEQ`
0000	`NOP`
1111	`END`

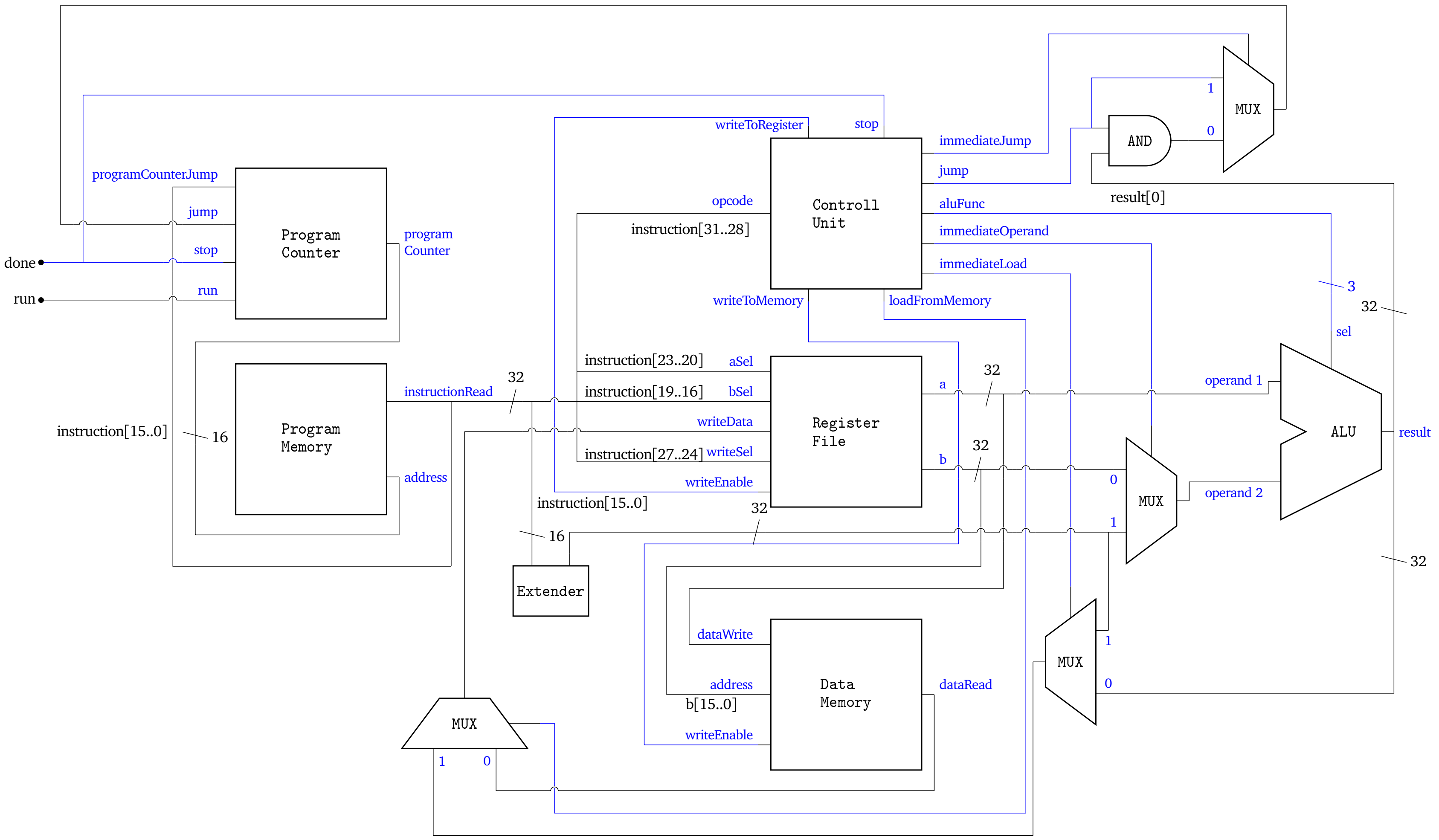
2.2. Compile and encode

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

Listing 1: *The program compiled to assembly*

```
        # Initial values
LOADI R0, 0;        # x counter
LOADI R1, 0;        # y counter
LOADI R2, 19;       # Pixel limit
LOADI R3, 0;        # Zero value
LOADI R4, 255;      # 255 value
        # For loop conditions
JEQ R0, R2, 58      # Check x, GOTO END
JEQ R1, R2, 54      # Check y, GOTO INC X
        # Output image adress
MULT R5, R1, 20;    # y * 20
ADD R6, R0, R5;     # x + y * 20
ADDI R5, R6, 400;   # Out image address
        # Process border pixel
JEQ R0, R3, 48      # If x or y = 0
JEQ R1, R3, 48      #
JEQ R0, R4, 48      # If x or y = 19
JEQ R1, R4, 48      # GOTO erosion
        # Process inner pixel
MULT R6, R1, 20;    # y * 20
ADD R7, R0, R6;     # x + y * 20
LOAD R8, R7;        # Get input pixel
JEQ R8, R3, 48;     # If 0, GOTO erosion
        # Process outer pixels
SUBI R12, R0, 1     # x - 1
ADDI R13, R0, 1     # x + 1
SUBI R14, R1, 1     # y - 1
ADDI R15, R1, 1     # y + 1
MULT R6, R1, 20;    # y * 20
ADD R7, R6, R12;    # (x - 1) + y * 20
LOAD R8, R7;        # Save pixel in R8
MULT R6, R1, 20;    # y * 20
ADD R7, R6, R13;    # (x + 1) + y * 20
LOAD R9, R7;        # Save pixel in R9
OR R10, R8, R9;     # OR R8 and R9, save to R10
MULT R6, R14, 20;   # (y - 1) * 20
ADD R7, R6, R0;     # x + (y - 1) * 20
LOAD R8, R7;        # Save pixel in R8
OR R9, R8, R10;     # OR R8 nad R10, save to R9
MULT R6, R15, 20;   # (y + 1) * 20
ADD R7, R6, R0;     # x + (y + 1) * 20
LOAD R10, R7;       # Save pixel in R10
OR R8, R9, R10;     # OR R9 and R10, save to R8
JEQ R8, R3, 48;     # If = 0 GOTO Erosion
              # No erosion
STORE R4, R5;       # Set pixel to 255
JMP 51;             # GOTO increment y
              # Erosion
STORE R3, R5;       # Set Pixel to zero
JMP 51;             # GOTO increment y
              # Increment y
INC R1;             # Increment y
JMP 8;              # Continue nested loop
              # Increment x
INC R0;             # Increment x
LOADI R1, 0;        # Zerorise y
JMP 7;              # Continue main loop
              # Terminate program
END;                # Terminate program
```

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*