# 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 5$^{th}$

# 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 · 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` |
| Bitwise NOT | `NOT Rx, Ry;` | `Rx = ~Ry` |
| | 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` |
| 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.*

| OPCODE (4 bits) | Rd (4 bits) | R1 (4 bits) | R2 (4 bits) | Value/address (16 bit) |
|---|---|---|---|---|
| $31 \cdots 28$ | $27 \cdots 24$ | $23 \cdots 20$ | $19 \cdots 16$ | $15 \cdots 0$ |

**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.

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

***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` |

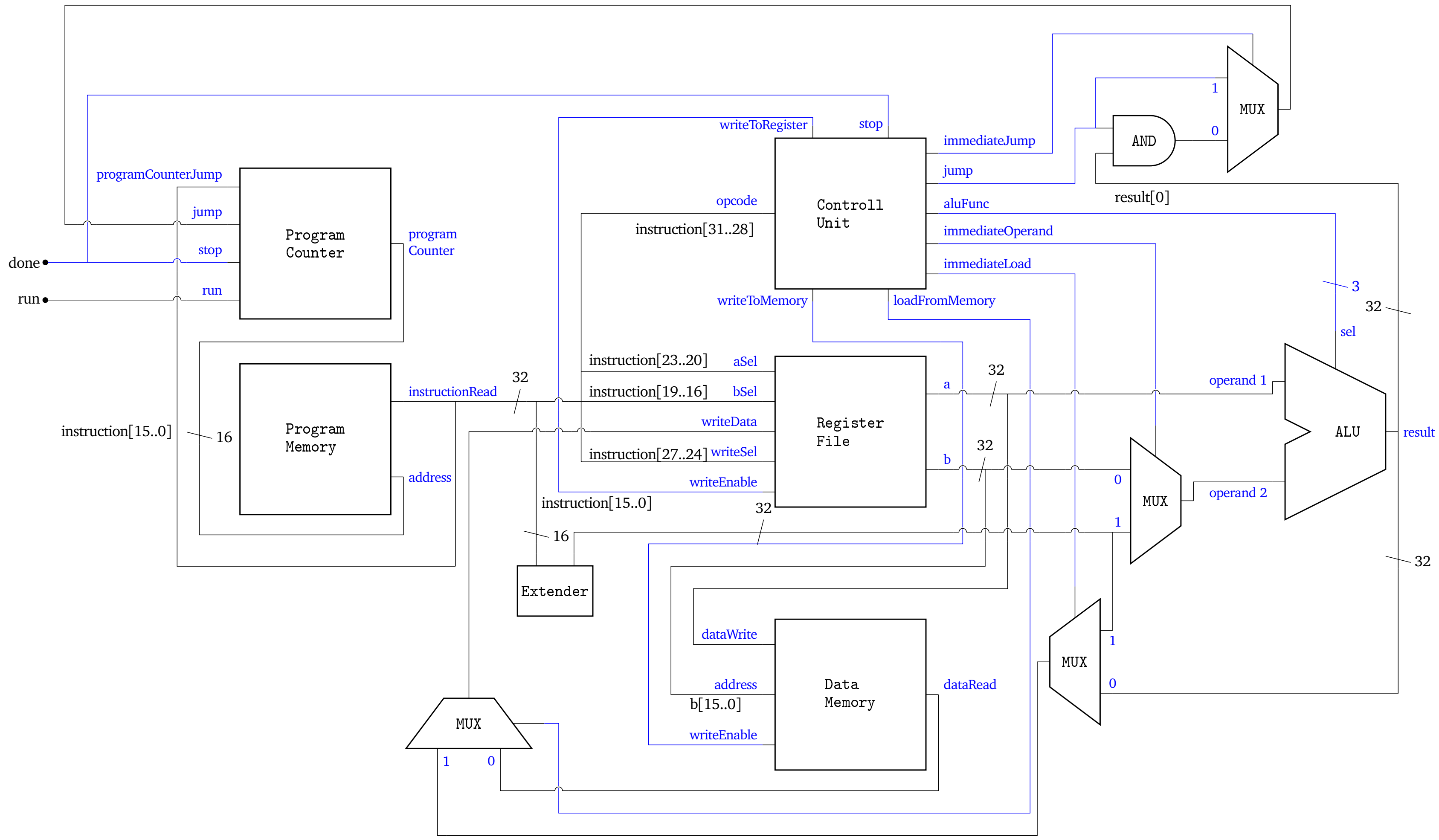
### 2.2. Compile and encode

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

```
LOADI R0, 0; // Initial values; x counter
LOADI R1, 0; // y counter
LOADI R2, 19; // Pixel limit
LOADI R3, 0; // Zero value
LOADI R4, 255; // 255 value
JEQ R0, R2, 55 // For conditions; Check x, GOTO program termination
JEQ R1, R2, 52 // Check y, GOTO incrementation of x
JEQ R0, R3, 45 // Process border pixel; If x = 0, zerorise
JEQ R1, R3, 45 // If y = 0, zerorise
JEQ R0, R4, 45 // If x = 19, zerorise
JEQ R1, R4, 45 // If y = 19, zerorise
MULI R5, R1, 20; // y * 20
ADD R6, R0, R5; // x + y * 200
LOAD R8, R6; // Get input image pixel
JEQ R8, R3, 40; // Check if 0, GOTO pixel zerorising
SUBI R9, R0, 1 // x - 1
ADDI R10, R0, 1 // x + 1
SUBI R11, R1, 1 // y - 1
ADDI R12, R1, 1 // y + 1
MULI R5, R1, 20; // y * 20
ADD R6, R9, R5; // (x - 1) + y * 200
LOAD R13, R6; // Save pixel in R13
MULI R5, R1, 20; // y * 20
ADD R6, R10, R5; // (x + 1) + y * 200
LOAD R14, R6; // Save pixel in R14
OR R15, R13, R14; // OR R13 and R14, save to R15
MULI R5, R11, 20; // (y - 1) * 20
ADD R6, R0, R5; // x + (y - 1) * 200
LOAD R13, R6; // Save pixel in R13
OR R14, R13, R15; // OR R15 nad R13, save to R14
MULI R5, R12, 20; // (y + 1) * 20
ADD R6, R0, R5; // x + (y + 1) * 200
LOAD R13, R6; // Save pixel in R13
OR R15, R13, R14; // OR R14 and R13, save to R15
JEQ R15, R3, 39; // Jump to erosion
MULI R5, R1, 20; // y * 20
ADD R6, R0, R5; // x + y * 200
ADDI R7, R6, 400; // Offset to output image
STORE R4, R7; // Set pixel to 255
JMP 50; // GOTO increment y
MULI R5, R1, 20; // y * 20
ADD R6, R0, R5; // x + y * 200
ADDI R7, R6, 400; // Offset to output image
STORE R3, R7; // Set Pixel to zero
JMP 50; // GOTO increment y
MULI R5, R1, 20; // y * 20
ADD R6, R0, R5; // x + y * 200
ADDI R7, R6, 400; // Offset to output image
STORE R3, R7; // Set Pixel to zero
JMP 11; // Continue program after border process
INC R1; // Increment y
JMP 6; // Continue nested loop
INC R0; // Increment x
LOADI R1, 0; // Zerorise y
JMP 5; // Continue main loop
END; // Terminate program
```

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

***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*