CSARCH2 Case Study 2

T3 AY2024-2025

1 Introduction

Microprogramming is a form of communication sent from the control unit to coordinate the different components of the CPU to perform processes. The CSARCH2 Case Study 2 features the alternative way of implementing Microprogramming into a safe, fun, and more visual way of learning to give students better understanding of the lesson. The Case Study 2 will be implemented in Minecraft Java 1.20. We aim to provide a better platform for students to learn and understand each circuitry in a same and fun manner while being able to maximize full visualization of each component of the circuit. It will mainly focus on the Arithmetic Logic Unit and the Control Unit, as these topics have been taught through the CSARCH series.

1.1 Learning Objectives

Upon successful completion of this project, students are expected to:

- Explain how microinstructions control data flow and component interaction within a CPU.
- Design and implement a functional ALU using redstone logic components.
- Simulate instruction execution based on a predefined opcode set without control flow operations (e.g., loops, jumps).
- Evaluate the effectiveness of using gamified platforms like Minecraft in supporting the understanding of complex architectural concepts.

1.2 Background Knowledge

Students should possess prior knowledge and skills in the following areas:

- Boolean logic and the behavior of basic logic gates (AND, OR, NOT, NAND, NOR, XOR)
- Design and analysis of combinational and sequential digital circuits (covered in CSARCH1)
- Fundamental components and operation of a CPU, including registers, ALU operations, microprogramming principles, and the instruction execution cycle (introduced in CSARCH2 and LBYARCH)

1.3 Masterclass

PTS will be holding a Masterclass on July 16 to teach students the Minecraft implementations of different components needed for this case study. Look out for their announcement for the event, it will also be published in the Canvas Announcement.

2 Specifications

Students are tasked with constructing a partial implementation of a 16-bit Von Neumann-style CPU using Minecraft Java Edition 1.20. The focus will be on the Arithmetic Logic Unit (ALU) and the execution of a limited instruction set. The implementation must be built within a standardized Minecraft world template provided by the instructors. Students will use redstone-based digital logic to simulate the behavior of micro-programmed instruction execution, register and memory transfers and ALU operations.

2.1 Template

The Main Memory, Registers, and Output Console will be provided as a template since Main Memory is not part of the syllabus. The Control Unit will be provided as a blank template and the Op code will be provided in the specifications since Op code is not part of the syllabus, students only need to connect the Control Unit to the Control Signals for each Execution Cycle for this component. The Instruction Register will be provided but the Register Controller will be blanked for students to implement. The ALU will have some blanks that the students must fill for the operations they need to implement. The input and output console will be provided as a standard and cannot be moved, including all mentioned components above, as testing the circuitry on the demo will rely on specific commands predefined with the original location. Some of the wires will be intentionally cut which students must connect and check if signals are passing through properly for all components. Students will be taught on how to use these components in the Masterclass for CSARCH2 Case Study 2.

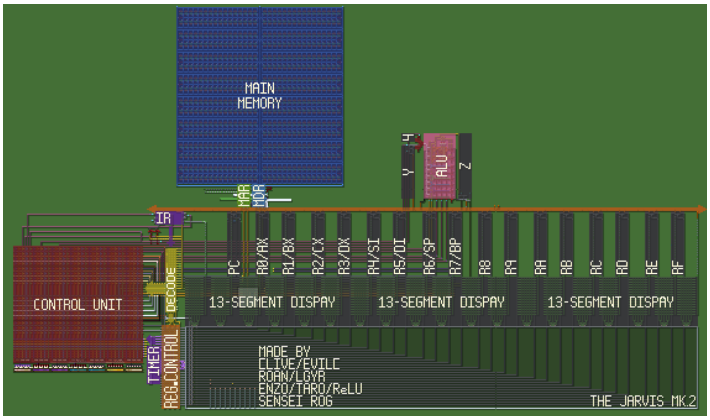

Figure 1: Map of the Entire Machine

Listed below this are the components that are needed to be implemented for the project.

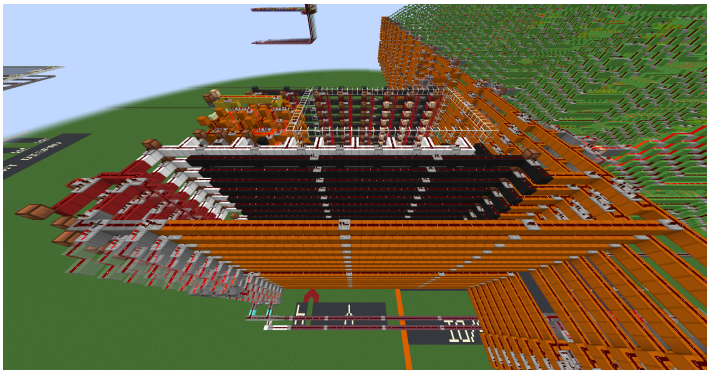

Figure 2: ALU with missing components indicated with Glass

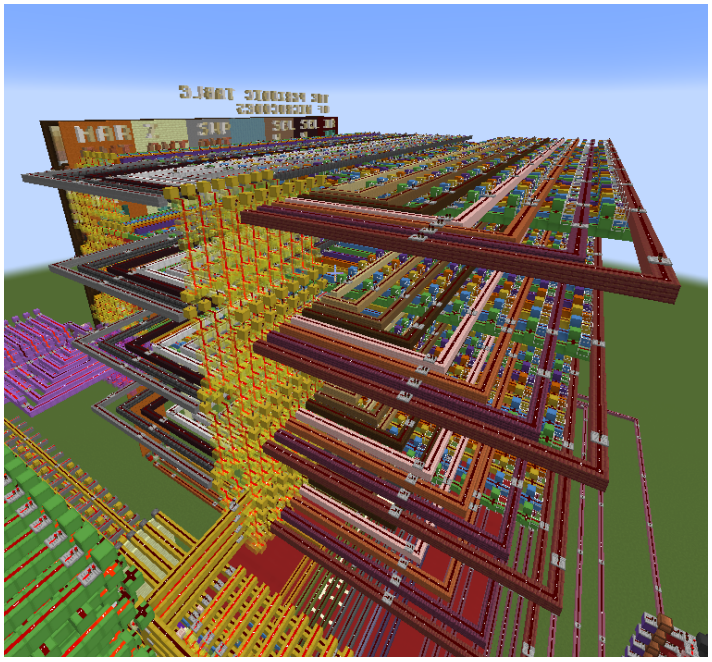

Figure 3: Control Unit with no connected control signals

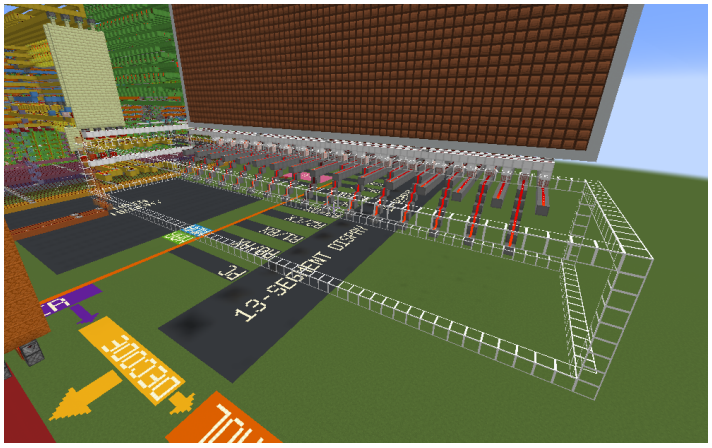

Figure 4: Register Controller missing to be implemented

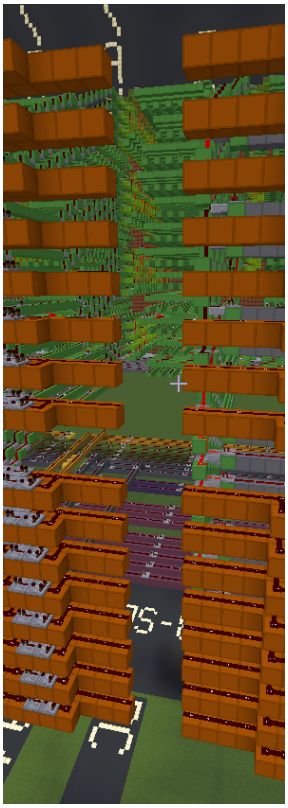

Figure 5: Example of a wire that was cut

2.2 Special Modifications

The components made are not fully optimized like the real life implementation due to limitations of the game so each instruction will run through 8 sets of CPU cycles divided into 3 cycles for the Fetch and Decode while the remaining 5 cycles for the Execute. Students do not need to implement the Fetch and Decode cycle and the HALT instruction and are only to implement the Execution cycle for the other Assembly codes given in the requirements for the group.

Figure 6: Periodic Table of Microcode Control Signals

Listed above is the chart of Microcode Control Signals which are mainly derived from the lecture though some special control signals were added to fit the environment of Minecraft Redstone. WMFC (Wait for Memory Function to Complete) is not part of the signals due to timing issues of Minecraft Redstone, it is already timed

and assumed that the memory operation is complete before the next cycle.

Instruction Register	
IRin	Inputs into the Instruction Register to Decode
IRDFout	Outputs the Data from the Decoded Instruction
IRAFout	Outputs the Address from the Decoded Instruction
SELECT Other	Selects the middle 8 bits instead of the last 8 bits

Arithmetic Logic Unit	
SELECT Y	Selects the Y Register for the ALU
SELECT 4	Selects 4 for the ALU
Yin	Inputs into the Y Register for ALU
Cout	Outputs the Carry Flag into the Adders
SET C	Sets the Carry Flag to 1 into the Adders
ALU OP0	Sets the LSb of the ALU Operation to Decode which Operation to do
ALU OP1	Sets the middle bit of the ALU Operation to Decode which Operation to do
ALU OP2	Sets the MSb of the ALU Operation to Decode which Operation to do
Zin	Inputs into the Z Register for the ALU Result
Zout	Outputs the Z Register for the ALU Result

ALU Operations	
0 0 0	ADD
0 0 1	SUB
0 1 0	Left blank for the Students to Implement
0 1 1	Left blank for the Students to Implement
1 0 0	Left blank for the Students to Implement
1 0 1	Left blank for the Students to Implement
1 1 0	Left blank for the Students to Implement
1 1 1	Left blank for the Students to Implement

Main Memory	
MARin	Inputs to the Memory Address Register
MARout	Outputs the Memory Address Register
MDRin	Inputs to the Memory Data Register
MDRout	Outputs the Memory Data Register
READ	Reads the Address in MAR and stores the data in MDR
WRITE	Writes the data in MDR into address in MAR

Program Counter	
PCin	Inputs to the Program Counter to indicate the Address of the Instruction
PCout	Outputs the Program Counter or the current Instruction Address
RESET PC	Resets the Program Counter to 0

Register	
REGin	Inputs to the Register (Decoder is implemented to select which register)
REGout	Outputs to the Register (Decoder is implemented to select which register)
SWAP OUT	Allows to output the 1st Operand instead of input

2.3 Operational Constraints

To ensure standardization across all implementations:

- Students may not alter the position or structure of the provided IR, Main Memory, or I/O consoles.
- The use of command blocks is strictly prohibited, both to prevent unintended behavior and to maintain academic integrity.
- Opcode mappings and instruction formats are fixed per group and will be released as part of the project package.

2.4 Resources

The resources needed for this project are located in this GitHub link:
Archcraft-Nibbles GitHub Repository

3 Grading System

The grading for this case study is divided into 3 parts (Demo, Test Cases, Reflection) with the ff. point distribution.

- Demo (30)
- Test Cases (60)
- Reflection (10)

3.1 Demo

Students are to prepare a 5 minute demonstration to explain each component and how they connected the components together with the provided Op code.
Provided here are guide questions for the demo:

- What does each component do?
- How does the Control Unit work?
- How does the Control Unit tell each component of the CPU what to do?
- How does each opcode execute?

Criteria	**Points**
Explanation of Each Component	5
Explanation of Control Unit	10
Explanation of Opcode Execution	10
Proper length, structure, and format	5

3.2 Test Cases

Sample test cases will be provided below to assist in finding certain cases but students are expected to do quality assurance to find all possible edge cases as there will be hidden test cases that will be tested on checking the submission.

Criteria	Points
Test Case 1 (Basic Register Transfer)	10
Test Case 2 (Basic Memory Read)	10
Test Case 3 (Arithmetic Operations 1)	20
Test Case 4 (Arithmetic Operations 2)	20

3.3 Reflection

Students are to submit a minimum of 400 words essay explaining their experience through the case study.

Learning and Understanding

- Did implementing Microprogramming in Minecraft help you understand the topic better?
- How did the visual and hands-on nature of building with redstone help you understand abstract computer architecture concepts that you previously learned only through theory?
- After completing this project, how has your understanding of real-world computer systems and CPU design changed or improved?

Engagement and Experience

- Was it more fun and engaging to learn microprogramming in this manner compared to without Minecraft (e.g., studying by text)?
- Would you recommend continuing this project for the next batches?
- Would you want to see other subjects incorporate this type of learning for their major course outputs?

Feedback and Recommendations

- What do you recommend to help improve this project?
- What are some insights and limitations you experienced from this project?

Criteria	Points
Personal Reflection	10

- Reflections must be original and specific to the student's personal experience.
- These responses will also contribute to a research study on the use of gamified learning tools in computer architecture education. All individual responses will be kept confidential.

3.4 Submission

Each group is required to submit the following components in their respective submission pages:

- Minecraft World Folder: The completed world used for the implementation. This folder must contain all necessary files for the world to be opened and tested.
- Demo Video: A pre-recorded video (maximum of 5 minutes) demonstrating the implementation. The video should explain how the CPU components work, how the Control Unit was implemented for each Opcode, and show the system in operation.
- Individual Reflections: Each group member must submit a personal written reflection in either .pdf or .docx format. These files must be clearly labeled with the student's full name.

4 Allowed vs. Prohibited Behaviors

To maintain fairness and academic integrity, the following guidelines must be observed throughout the project.

4.1 Permitted Activities

- Reviewing redstone tutorials and online references to learn basic redstone logic and component behavior.
- Reusing simple redstone components or design patterns (e.g., flip-flops, logic gates) as long as they are not part of an entire working CPU or full subsystem.

4.2 Prohibited Activities (Considered Academic Misconduct)

- Discussing general circuit design concepts and debugging strategies with members of other groups.
- Sharing or copying world files, redstone circuits, or opcode implementations between groups.
- Replicating another group's exact design, even if obtained through informal means (e.g., screenshots, videos, word-of-mouth).
- Using command blocks or any form of automation to manipulate or alter circuitry in a way that bypasses the logic design (e.g., teleporting data, triggering hidden commands).
- Altering another group's project or attempting to interfere with the evaluation system.

Violations of these rules will be investigated and, if confirmed, will result in penalties up to and including a failing grade for the case study and disciplinary action as per the university's academic integrity policy.

S12 - Group 8

Opcode	Instruction	Example Instruction	Example Opcode
0	HALT	HALT	0000 0000 0000 0000
1	MOV REG, REG	MOV R1, R2	0001 0001 0000 0010
2	MOV REG, POINTER	MOV R1, [R2]	0010 0001 0000 0010
3	XOR REG, REG	XOR R4, R5	0011 0100 0000 0101
4	ADD POINTER, REG	ADD [R4], R5	0100 0100 0000 0101
5	ADD REG, REG	ADD R3, R1	0101 0011 0000 0001
6	MOV REG, IMMEDIATE	MOV R0, 0x05	0110 0000 0000 0101
7	MOV ADDRESS, REG	MOV [0x02], R3	0111 0000 0010 0011
8	—	—	—
9	AND REG, REG	AND R4, R5	1001 0100 0000 0101
10	SUB REG, REG	SUB R3, R1	1010 0011 0000 0001
11	INC REG	INC R1	1011 0001 0000 0000
12	—	—	—
13	MOV REG, ADDRESS	MOV R3, [0x02]	1101 0011 0000 0010
14	NOT REG	NOT R1	1110 0001 0000 0000
15	MOV POINTER, REG	MOV [R1], R2	1111 0001 0000 0010