Faculty of Engineering and Technology

Electrical and Computer Engineering Department

COMPUTER ARCHITECTURE

ENCS4370

Project 2

Simple multi cycle processor

Prepared by:

Sara Ibraheem Awaysa. **ID**: 1211642 **Section:** 3

Hala Hamed Jebreel. **ID**: 1210606 **Section:** 3

Instructor: Dr. Aziz Qaroush.

Date: 22/6/2024

Abstract

This project aims to improve understanding of processor architecture as well as the capacity to track down and detect design flaws or faulty functionality.

Designing a multi-cycle processor, including its Datapath and control units, involves creating a system capable of efficiently processing instructions over numerous clock cycles.

This project investigates the complex architecture required to process instructions like as arithmetic, logic, memory access, and control flow processes. The Datapath, which includes registers, ALUs, and memory components, communicates fluidly with control units that orchestrate instruction decoding, timing, and state changes. Using a modular approach, each instruction passes through phases adapted to its specific needs, maximizing resource use and performance.

Table of contents

Table of figures

List of tables

1. Design and Implementation

1.1 Data Path

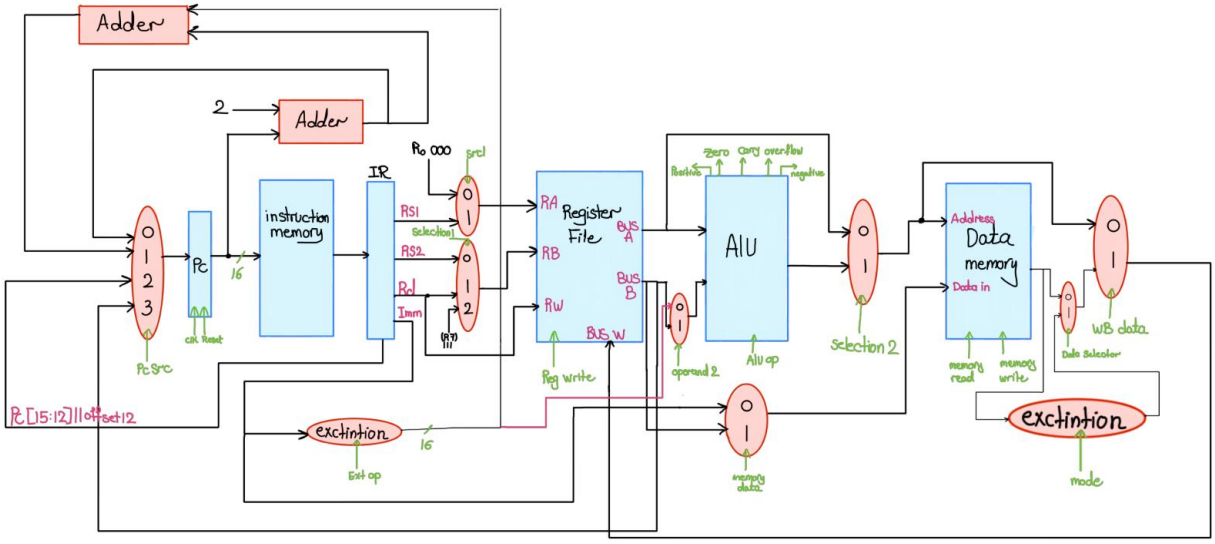

Figure1 : Our data path.

1.1.1 Program Counter:

The PC (Program Counter) Register is one of the components in data path. It is a CPU register in the computer processor that stores the address of the next instruction to be executed from memory and it is updated during the fetch stage. It allows for branching and jumping instructions, enabling control flow changes within the program. pc very important maintaining the execution flow of the program. [1]

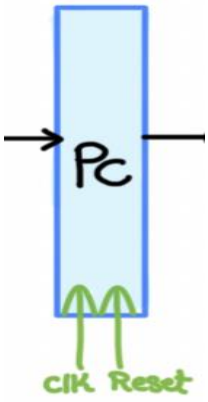

Figure 2: Program counter.

1.1.2 Adder Unit

It is a necessary component of the arithmetic logic unit (ALU) that conducts binary addition. This digital circuit takes two binary values as inputs and returns their sum as an output. This unit was used in our design to calculate the next pc in the fetch stage, depending on the current instruction.

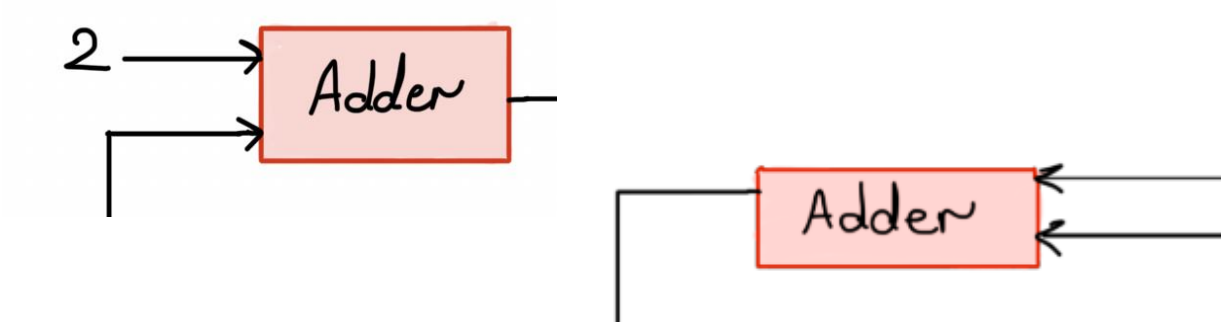

Figure 3:Adder unit.

1.1.3 Multiplexers (MUXs)

It is a device that picks one of multiple analog or digital input signals and routes it to a single output. Select lines are a separate set of digital inputs that control the selecting process. A multiplexer with 2^n inputs has n bit select line, which are used to determine which input line to send to the output. [2]

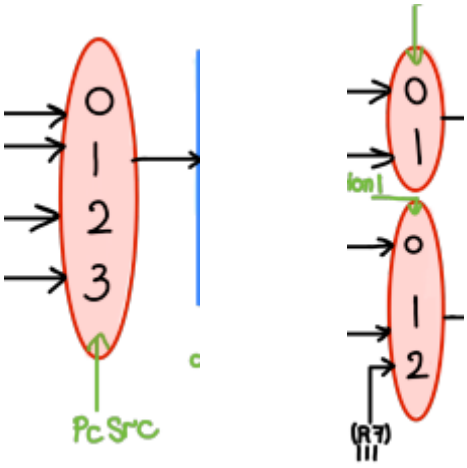

Figure 4 : Multiplexers.

1.1.4 Instruction Memory

The Instruction Memory is a critical component of a computer system's design and the data path. It stores and distributes execution instructions, ensuring quick access and efficient processing. [3]

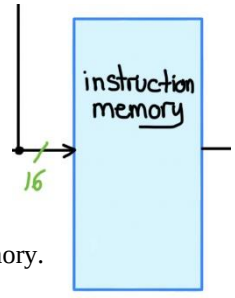

Figure 5:
Instruction memory.

1.1.5 Register file

A register file is made up of a set of registers that are used to stage data between memory and functional units. It also has inputs and outputs, which include the destination and source registers. So, in our design it consists of 2 read buses -for inputs- and one write bus -for output-. [4]

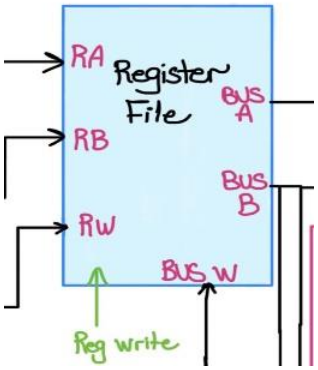

Figure 6 :Register file.

1.1.6 ALU

It is a combinational logic circuit that performs arithmetic and bitwise logical operations on integer binary numbers, in execution stage. The necessary operations for each instruction were executed according to incorporating the appropriate opcode and condition bit. [5]

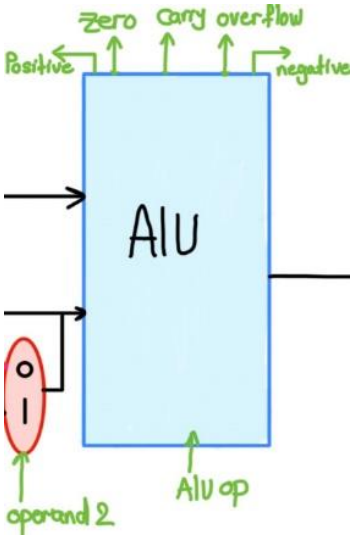

Figure 7: ALU.

1.1.7 Data Memory

This unit is used only in memory access stage to read from and write to the main memory according to the instruction -Load or store-. Data is transferred between the processor and the memory subsystem, where operations like fetching data, storing data, and performing necessary conversions occur. The Data Memory stage is crucial for executing instructions that involve memory operations, as it ensures correct data access and manipulation.

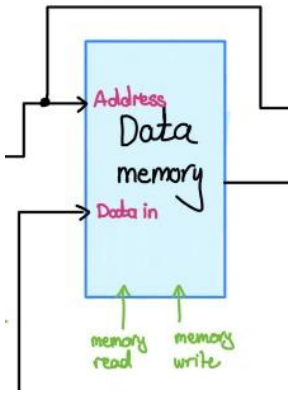

Figure 8:Data Memory

1.1.8 Extender

It is an essential component in a data path when there is a need to increase the size of a data signal by a specific number of bits. In our design, extenders were needed to extend the immediate to calculate the pc address in case of I-type instructions and also to extend the byte loaded in LB instructions.

Figure 9: Extender.

The implementation details, and the design choices you made with justification:

Stages:

- Instruction Fetch: in this stage, processor fetches the instruction from memory (instruction memory) and loads it into the instruction register.

- Instruction Decode: In this stage, processor decodes the instruction, Decode the instruction and read the necessary registers.

- Execution: Execute the operation specified by the instruction.
- Memory Access: If the instruction requires data to be read from or written to memory, the processor accesses the memory to perform the operation.
- Write Back: The processor writes the result of the operation back to destination register.

- From the data phat above in figure 1 , pc work based on clock and reset. Pc choose from the mux 4*1 according to pcSrc selection, then 16 bit from pc go to instruction memory , then instruction memory fetches the instruction and loads it into the instruction register(IR) , from IR we can Know Rs1, RS2, RD and Immediate (based on the type R-type, I-type , S-type and J-type) . Then go to the next stage (Register File). In ALU stage the input is Bus A from the register file or the mux 2*1 with (operand2 selection) as an input to the ALU , ALU have many flag (zero,carry,over flow. Positive and negative) it have there value based on the result from the ALU . Then the result from the Alu go to mux 2*1 with the value from the Register file to choose from them according to the selection 2 , after that it go to Data memory as an input (Address) and data in from other mux 2*1 which choose from the selection (memory data selection) as data in . Finally, the result from Data memory go to mux 2*1 with result from (mux 2*1 selection 2).

1.2 Control units

1.2.1 Control units' description

Table 1 : Control units' description.

Sara Awayssa-1211642

Hala Hammad-1210606

Signal	PC src	Ext op	Selection 1	Src1	RegWrite	Zero	Carry	Overflow	Positive	Negative	ALU op	Selection 2	Memory data	Wb data	Memory read	Memory write	Operand 2	Mode	Data selector
Description	Used to choose the suitable value of next PC.	Used to select if the required extension is zero or signed extension .	Used to choose the second operand which will appear at the 2nd read bus.	Used to select the first operand which enters the ALU.	A flag if there is a required data to write back it to the register file.	A flag resulted from ALU to indicate if the result is 0.	A flag resulted from ALU to indicate if the result contains a carry.	A flag resulted from ALU to indicate if there is an overflow .	A flag resulted from ALU to indicate if the result is positive .	A flag resulted from ALU to indicate if the result is negative .	Used to determine the used ALU unit.	Used to select the required data memory address.	Used to select the data in to be written to the data memory is store instructions .	Used to select the required data to be written back to the register file.	Used to indicate if the data memory will be read. (Load)	Used to indicate if the data memory will be written. (store)	Used to choose the second operand which enters the ALU.	Used to determine the type of extension to the loaded byte. (0 or sign)	Used to select the data out in load operation (word or extended byte)

1.2.2 Control units' Boolean equations

- PC src (as it requires 3 bits):

  PC src [0] = ~R-type + branch + Ret

  Pc src [1] = JUMP + CALL + RET

  Pc src [3] = 0

- Ext op = ~ANDI

- selection 1 (As it requires 2 bits) :

  Selection 1 [0] = ~R-type + ~RET

  Selection 1 [1] = RET

- src1 = ~BGTZ + ~BLTZ + ~BEQZ + ~BNEZ

- RegWrite = R-type + ADDI + ANDI + LW+ LBU + LBS

- Zero = BEQ + BEQZ

- Over flow = ~ BGT + ~BLT

- Positive = BGT + BGTZ

- Negative = BLT + BLTZ

- Selection2 = ~SV

- Memory data = SW

- WB data = LW+ LBU + LBS

- Memory read = LW + LBU + LBS

- Memory write = SW + SV

- Operand2 = ~I-Type

- Mode = LBS

-Data Selector = LBU + LBS

1.2.3 Control units' Truth table

		PC ctrl	Main ctrl				ALU ctrl						Main ctrl							
Instruction	opcode	PC src	Ext op	Selection1	Src1	RegWrite	Zero	Carry	Overflow	positive	negative	ALU op	Selection2	Memory data	Wb data	Memory read	Memory write	OPERAND2	Mode	Data selector
AND	0000	0	X	0	1	1	X	X	X	X	X	AND	1	X	0	0	0	1	X	X
ADD	0001	0	X	0	1	1	X	X	X	X	X	ADD	1	X	0	0	0	1	X	X
SUB	0010	0	X	0	1	1	X	X	X	X	X	SUB	1	X	0	0	0	1	X	X
ADDI	0011	0	1	X	1	1	X	X	X	X	X	ADD	1	X	0	0	0	0	X	X
ANDI	0100	0	0	X	1	1	X	X	X	X	X	AND	1	X	0	0	0	0	X	X
LW	0101	0	1	X	1	1	X	X	X	X	X	ADD	1	X	1	1	0	0	X	0
LBU	0110	0	1	X	1	1	X	X	X	X	X	ADD	1	X	1	1	0	0	0	1
LBS	0110	0	1	X	1	1	X	X	X	X	X	ADD	1	X	1	1	0	0	1	1
SW	0111	0	1	X	1	0	X	X	X	X	X	ADD	1	1	X	0	1	0	X	X
BGT	1000	1	1	1	1	0	0	X	0	1	0	SUB	X	X	X	0	0	X	X	X
BGTZ	1000	1	1	1	0	0	0	X	X	1	0	SUB	X	X	X	0	0	X	X	X
BLT	1001	1	1	1	1	0	X	X	0	0	1	SUB	X	X	X	0	0	X	X	X
BLTZ	1001	1	1	1	0	0	X	X	X	0	1	SUB	X	X	X	0	0	X	X	X
BEQ	1010	1	1	1	1	0	1	X	X	X	X	SUB	X	X	X	0	0	X	X	X
BEQZ	1010	1	1	1	0	0	1	X	X	X	X	SUB	X	X	X	0	0	X	X	X
BNE	1011	1	1	1	1	0	0	X	X	X	X	SUB	X	X	X	0	0	X	X	X
BNEZ	1011	1	1	1	0	0	0	X	X	X	X	SUB	X	X	X	0	0	X	X	X
JUMP	1100	2	X	X	X	0	X	X	X	X	X	X	X	X	X	0	0	X	X	X
CALL	1101	2	X	X	X	0	X	X	X	X	X	X	X	X	X	0	0	X	X	X
RET	1110	3	X	2	X	0	X	X	X	X	X	X	X	X	X	0	0	X	X	X
SV	1111	0	X	X	1	0	X	X	X	X	X	X	0	0	X	0	1	X	X	X

Sara Awayssa-1211642

Hala Hammad-1210606

Table 2: Control units truth table.

To achieve an efficient multi-cycle processor design, multiple factors were carefully evaluated and handled during the design and implementation phases. To ensure optimal functionality, each CPU component passed extensive testing. The ALU was tested for all arithmetic and logical operations. The control unit was tested for proper state transitions and output signals. The register file was verified to handle simultaneous reads and writes appropriately, and memory units were validated for data consistency and addressing. All phases were thoroughly tested, beginning with fetch and ending with writeback.

2. Simulation and testing:

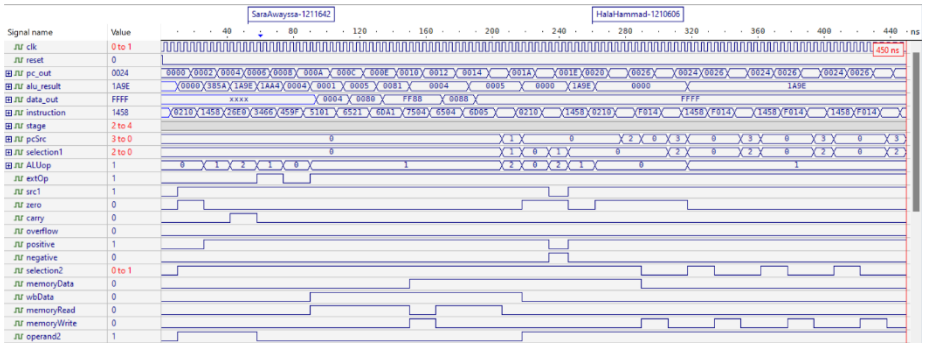

Figure 10 : Design simulation.

To make results clearer:

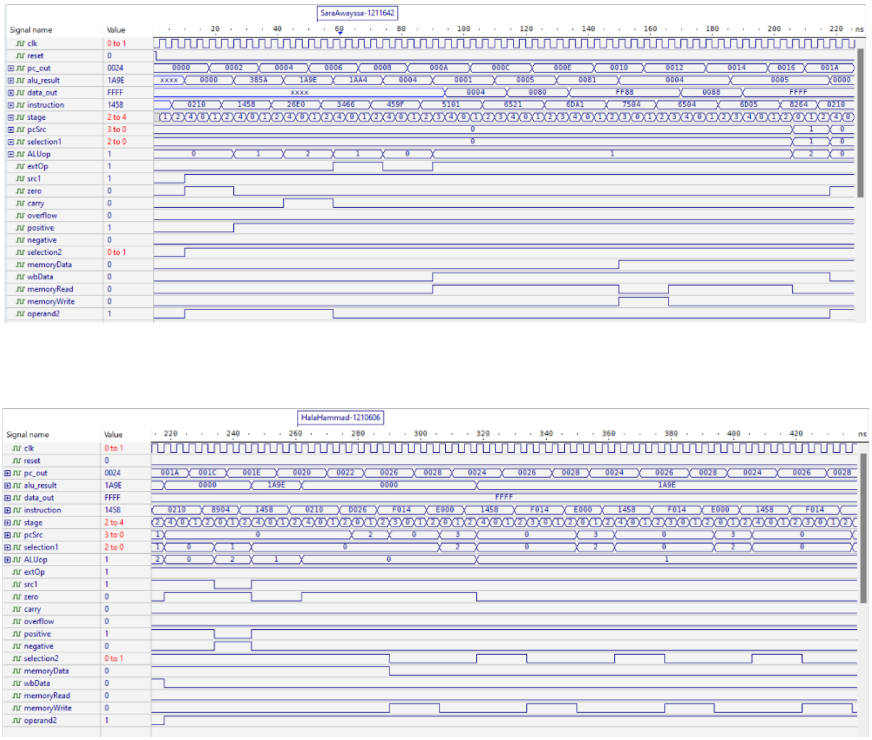

Figure 11 : Clearer simulation results.

All instructions were tested using the following instruction memory:

```
//_____________________________ Initialize memory with encoded instructions_____________________________

        mem[0]  = 8'h10;
        mem[1]  = 8'h02;
        mem[2]  = 8'h58;
        mem[3]  = 8'h14;
        mem[4]  = 8'hE0;
        mem[5]  = 8'h26;
        mem[6]  = 8'h66;
        mem[7]  = 8'h34;
        mem[8]  = 8'h9F;
        mem[9]  = 8'h45;
        mem[10] = 8'h01;
        mem[11] = 8'h51;
        mem[12] = 8'h21;
        mem[13] = 8'h65;
        mem[14] = 8'hA1;
        mem[15] = 8'h6D;
        mem[16] = 8'h04;
        mem[17] = 8'h75;
          mem[18] = 8'h04;
        mem[19] = 8'h65;
        mem[20] = 8'h05;
        mem[21] = 8'h6D;
          mem[22] = 8'h64;
          mem[23] = 8'h82;
          //mem[22] = 8'h44;
          //mem[23] = 8'h83;
          mem[24] = 8'h58;
          mem[25] = 8'h14;
          mem[26] = 8'h10;
        mem[27] = 8'h02;
          mem[28] = 8'h04;
          mem[29] = 8'h89;
          mem[30] = 8'h58;
          mem[31] = 8'h14;
          mem[32] = 8'h10;
          mem[33] = 8'h02;
          mem[34] =  8'h26;
          mem[35] = 8'hD0;
          mem[36] = 8'h58;
          mem[37] = 8'h14;
          mem[38] = 8'h14;
          mem[39] = 8'hF0;
          mem[40] = 8'h00;
          mem[41] = 8'hE0;
```

Figure 12 : Used instruction memory.

The test programs were created to cover a wide variety of instructions and scenarios. All tested instructions (ADD, SUB, AND, ANDI, ADDI, LW, LBU, LBS, SW, BEQ, BEQZ, JUMP...) worked successfully and produced the desired effects. The multi-cycle processor's

functionality is thoroughly tested to ensure that it processes arithmetic, logical, data transfer, and control flow instructions correctly.

The inputs included the clock signal (clk) and reset one (reset), while the outputs were various control signals such as extOp, src1, selection1, ALUop, selection2, memoryData, wbData, memoryRead, memoryWrite, and operand2 in addition to registers like pc_out, alu_result, data_out, and instruction.

The test programs guaranteed that the stage signal flowed accurately through each cycle. For example, while testing the ADD instruction, the intended result was that alu_result included the correct sum and carry. Similarly, for a Load operation instruction, data_out and memoryRead were anticipated to indicate the proper data transfer to memory.

Each instruction passed only the stages it needs, so the cycle time was according to the longest instructions (LW, LBU, LBS) which needed 5 stages.

All results were as expected, and no errors occurred.

3. Conclusion

Designing a multi-cycle processor entails developing a thorough data pipeline and control units to properly handle multiple instruction types. Each instruction moves through the processor's stages as needed, ensuring that resources are used efficiently. The data path includes components such as ALUs, registers, and memory that are linked to perform arithmetic, logical, and data transfer operations. Control units monitor the sequential execution of instructions, ensuring that each stage (fetch, decode, execute, memory access, and write-back) occurs appropriately.

After finishing this project, our expertise and knowledge of multi-cycle processing have substantially improved as a result of the design and implementation of a processor architecture capable of efficiently handling complicated instructions. This effort has improved our understanding of how various processor components interact, from the data pipeline with registers, ALU, and memory units to the control units that orchestrate instruction execution across multiple stages. We learned how to optimize performance by ensuring that each instruction moves through the pipeline stages only as needed, reducing resource use and increasing overall efficiency. We have strengthened our understanding of how complex instructions are processed step by step through rigorous testing and validation, demonstrating a thorough mastery of multi-cycle processing principles.

The outcomes were as expected, with no significant difficulties.

4. Teamwork

Teamwork is the foundation of our project's success, with each team member's unique abilities and views combining to achieve our objectives successfully.

We built this project with a group of two students, Hala Jebreel-1210606 and Sarah Awaysa-1211642. We worked collectively on this project, starting with thinking about it, building the design of the data path, developing a table of control signal, building the code and the test bench, and correcting each other's mistakes. (We opened a zoom meeting and worked on the whole project together).

To give details, the project implementing began with opening a zoom meeting and discussing the project's ideas, hardships and even the very small tricks. Then, another zoom meeting was held to begin writing the code together, each line was written based on the approval of both of us, and all methods and strategies were chosen together. After finishing writing the code, Sarah started writing the test bench, the sequence of instructions and checking the correctness and effectivity of the project by tracking the simulation output. At the same moment, Hala was preparing the report. At the end, Sara continues the report and Hala arranged the code and added comments. So, in the report, Hala wrote about the components of the data path which was designed during a zoom meeting and wrote the conclusion and prepared for the tables of contents, figures and tables, while Sarah wrote about the control units and the simulation and testing in addition to the abstract.

The whole project was tested by each of us and we corrected each other's mistakes.

5. References

[1]: Techopedia. [Accessed on 21st June 2024]
https://www.techopedia.com/definition/13114/program-counter-pc

[2]: Wikipedia. [Accessed on 21st June 2024]
https://en.wikipedia.org/wiki/Multiplexer

[3]: Science Direct. [Accessed on 21st June 2024]
https://www.sciencedirect.com/topics/computer-science/instruction-memory

[4]: Wikipedia. [Accessed on 21st June 2024]
https://en.wikipedia.org/wiki/Register_file

[5]: Wikipedia. [Accessed on 21st June 2024]
https://en.wikipedia.org/wiki/Arithmetic_logic_unit