Digital Logic And Design

PLANNING (PHASE 2)

Group Name: Abstraction

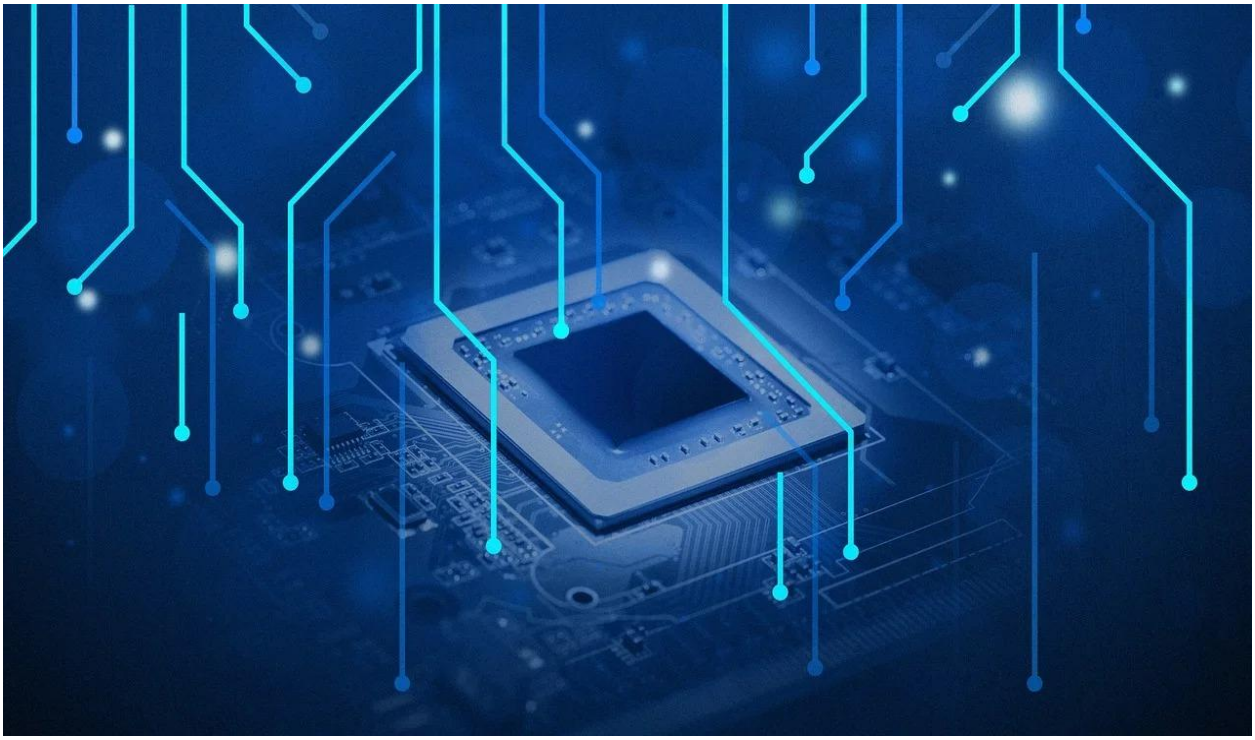

Table Of Contents

Group Members:

Lead: Muhammad Ibrahim Ayoubi 26269

Co-Lead: Ikhlas Ahmed Khan 27096

Abaan Noor 24911

Izma Khan 26926

Sameed Ahmad 26956

Itbaan Safwan 26197

Muhammad Hamza Zaman 27162

Muhammad Bilal Adnan 27151

Work Breakdown:

Program Counter— Abaan Noor and Itbaan Safwan.

Arithmetic and Logic Unit— Ibrahim Ayoubi and Ikhlas Khan.

Instruction Register—Izma Khan and Sameed Ahmed.

Accumulator— Hamza Zaman and Bilal Adnan.

Inclusions:

According to our current plan, we would be designing the following components of our Sap-1 computer: Program Counter, Instruction Register, Accumulator, Arithmetic and Logic Unit(ALU), and Output Display.

-The **Program Counter** is a register that holds the memory address of the instruction to be fetched and executed next. It is crucial for the sequential execution of instructions in a program. **Incrementing**: Automatically increase the program counter after each instruction fetch. **Reset**: Ability to reset the program counter. **Components of the Program Counter**: Counter Register (CR), Incrementer and Control Logic.
Since we have **not** planned to design the memory yet, the address given out by the program counter would go directly into our microcontroller where this address would be readed and in return a pre-hard coded instruction would be sent out towards the instruction register.

The **Instruction-Register** as the name suggests would be responsible for receiving the instruction in 8 bit format directly from the microcontroller(as of now), and breaking that instruction into 2 nibbles of 4 bit each of opcode and data. Now since our microcontroller would itself act as a control unit then that opcode would be interpreted by our microcontroller itself and an appropriate operation would be performed such as LOAD,ADD etc.
The **ALU** of our Sap-1 computer would be able to perform the arithmetic operations of Addition and Subtraction ONLY as of now of four bit binary

numbers. The ALU would typically receive two sets of binary inputs, labeled A and B.According to our design one input would come directly from the pre-existing value in the accumulator while another value would come from the data received at the instruction register..We would specifically be focusing on the adder/subtractor circuit in our designed ALU which can perform both addition and subtraction operations.

Output: The ALU produces a binary output, referred to as the ALU result, which would be forwarded to the Accumulator.

EXAMPLE: If the instruction specifies an addition operation, the control unit(Micro-controller in this case) sets the ALU control signals for addition. The ALU then adds the values received at its both inputs A and B, while eventually storing the result in the Accumulator.

The **Accumulator**:

Initialization: During CPU startup or reset, the accumulator is conventionally set to zero to eliminate any residual data from prior computations.

Loading Data: Now if the LOAD instruction is to be executed only then the data would come directly into the accumulator.

Arithmetic Operations: Integral to arithmetic operations such as addition and subtraction, the accumulator actively engages with our designed ALU, which processes data stored within the accumulator based on control signals.

Result Output: Following an operation, the final result stored in accumulator can be forwarded to an OUTPUT source.

Exclusions:

In the **Program Counter** we will exclude:

-Parallel Execution: Sequential execution without parallel processing.

-Interrupt Handling: Basic program counter functionality without interruption support.

In the **ALU** we will be excluding:

-Floating Point Operations: This ALU will focus on integer operations.
-Overflow of bits
-Operations of multiplication and division.
-Advanced Logic Operations: Complex logical operations beyond basic binary operations like AND,OR,XOR,NOT.
-Shift Operations like right shift,left shift etc.
-Comparisons.
-Negative Binary Result in Addition and Subtraction.

FOR our entire **SAP-1** we will exclude:
-Operations on 8 bits.
-Exclude instructions that involve direct addressing towards a memory since we have not planned on designing the memory yet.
-Interrupts and Exception handling.

Functional Requirements:

Our 4-bit Sap-1 computer should be able to add or subtract any two 4-bit binary numbers. The Program Counter would increment or reset at the appropriate phases of the instruction cycle. The PC would be 4-bit and interface seamlessly with the instruction decoder, a control unit which would be our microcontroller in other words.

The ALU should operate with a defined bit precision (e.g., 4-bit and perform arithmetic and logic operations within specified clock cycles while Interfacing seamlessly with the SAP 1 architecture.

The **IR** would be an 8-bit register that holds the instruction code during the execution of the instruction. It should interface with the controller to decode the instruction and determine the operation to be performed. The IR should be able to hold and execute various instruction codes, each corresponding to a specific operation such as ADD, SUB, LDA.The IR should be able to receive instructions

from the controller during the fetch phase of the instruction cycle.

The Accumulator should be a 4-bit register that holds one of the operands during an arithmetic or logical operation. It should interface with the ALU, providing an operand for the ALU when required and storing the result of the ALU operation. The Accumulator should be able to load data from the controller or the output of the ALU. The Accumulator should be cleared or updated at the appropriate phases of the instruction cycle.

Input and Output Specifications:

The **Program-Counter** is simply a counter without an input that counts to a specific range. At each cycle it generates an address which it sends out to the microcontroller as an output which is responsible for sending out instructions to the IR according to the address received.

The **IR** would take the instruction code as an INPUT from the controller as input during the fetch phase of the instruction cycle. THe opcode it generates is recognized by our microcontroller acting as a control unit which sends out control signals along with the data to other components e.g ALU or accumulator.

The **Accumulator** would take one of the operands for arithmetic or logical operations from the memory/controller or the output of the ALU. It would also take control signals from the controller to load data from the memory or the ALU, and to output data to the memory or the ALU. The Accumulator would output the stored operand to the ALU for arithmetic or logical operations. It would also output the result of the ALU operation to an OUTPUT source.

The **ALU** would take two operands for arithmetic operations as an INPUT and control signals specifying the type of operation(ADD/SUBTRACT) as input. It will output the result of the arithmetic or logic operation to the accumulator.