Digital Logic And Design

PROJECT REPORT

Group Name: Abstraction

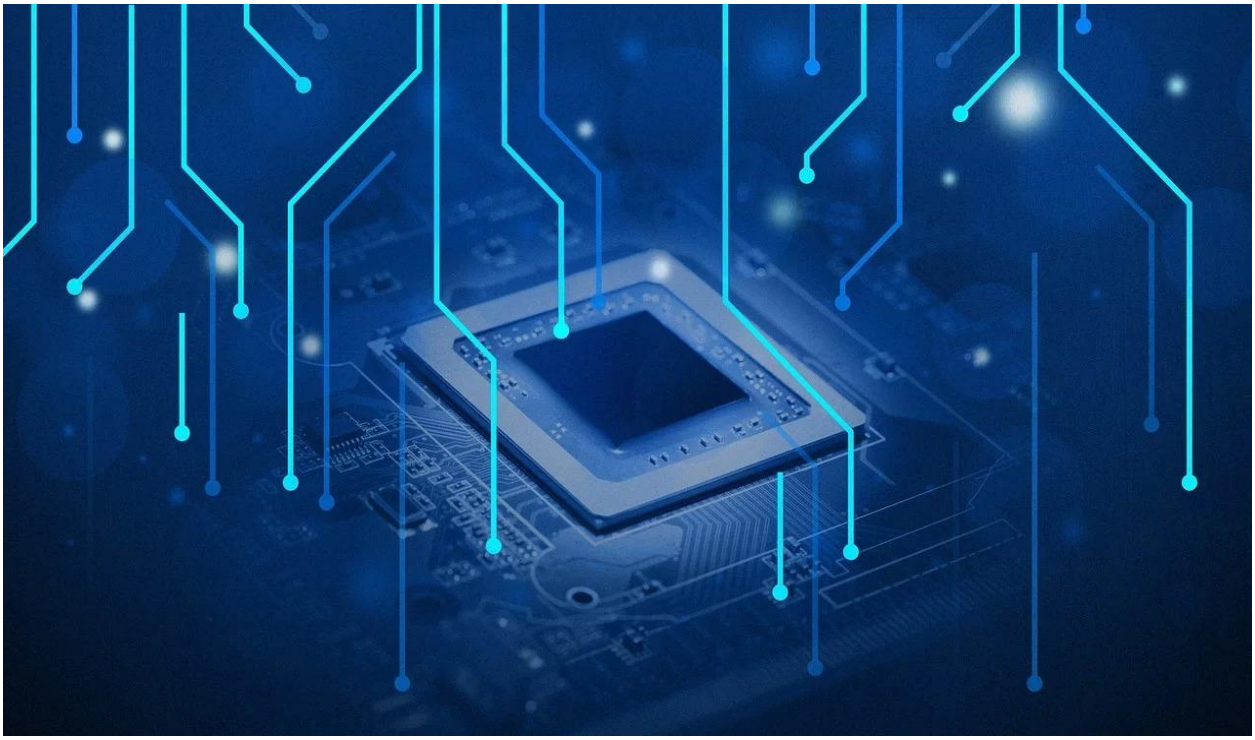


Table Of Contents

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

Introduction

Starting our exploration into computer architecture, our project is all about building the SAP-1 computer, an important step in understanding how computers work. We're aiming to create a 4-bit computer using the Raspberry Pi microcontroller.

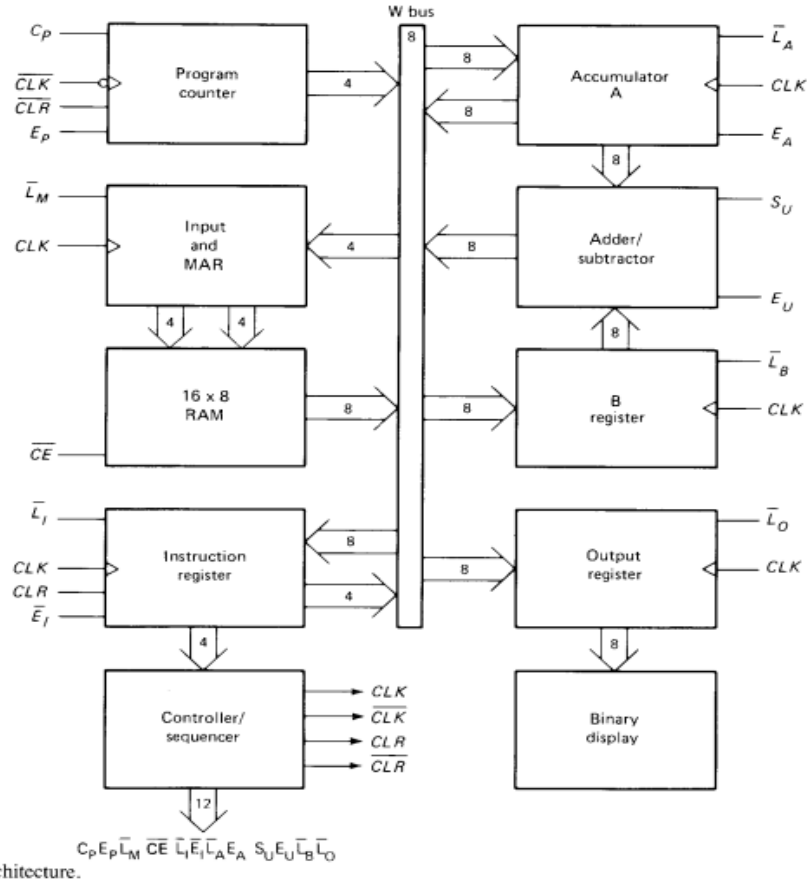


Fig. 10-1 SAP-1 architecture.

This report is meant to give a detailed look at how computers function, covering things like memory interactions and how different parts of the system, like the **Program Counter**, **Instruction Register**, **Accumulato**r, **Arithmetic and Logic Unit(ALU)**, and Output Display, all work together smoothly.

Program Counter:

The Program Counter, as the name suggests, is responsible for counting and storing the address of the next instruction in the program sequence. It is a register that holds the memory address of the instruction to be fetched and executed next. The Program Counter is crucial for the sequential execution of instructions in a program.

Components of the Program Counter:

1. **Counter Register (CR):** The Counter Register is a binary register that holds the current memory address. It is a part of the Program Counter and is used to store the address of the next instruction.
2. **Incrementer:** The Incrementer is a combinational circuit that takes the current address from the Counter Register and increments it by 1. The output of the Incrementer becomes the address of the next instruction.
3. **Control Logic:** The Control Logic is responsible for coordinating the operation of the Program Counter. It generates control signals to enable the incrementing of the Counter Register and to load the new address into it.

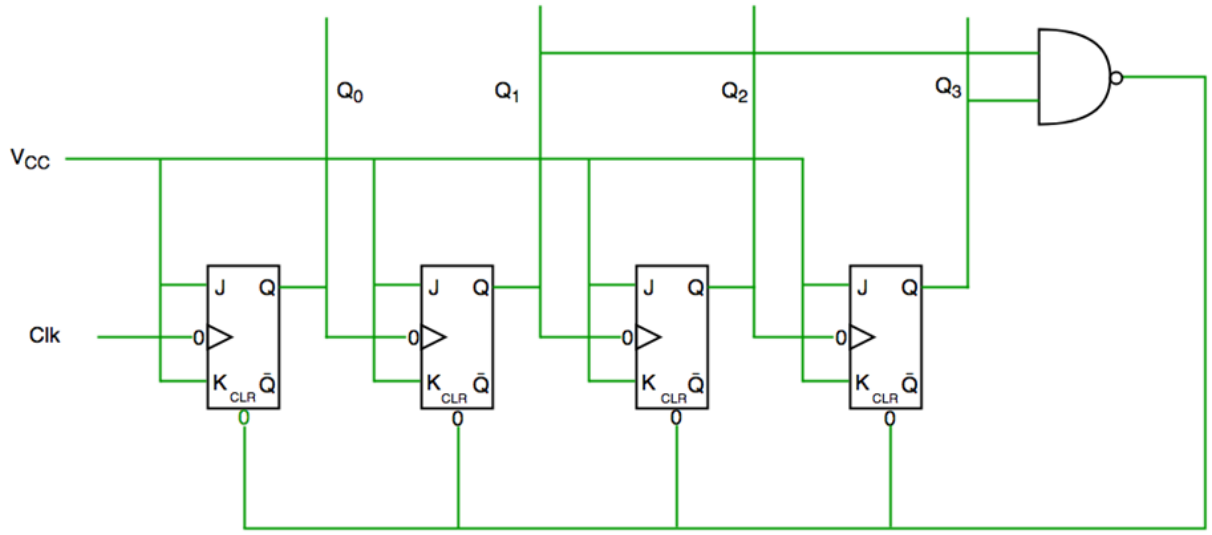
Operation of the Program Counter:

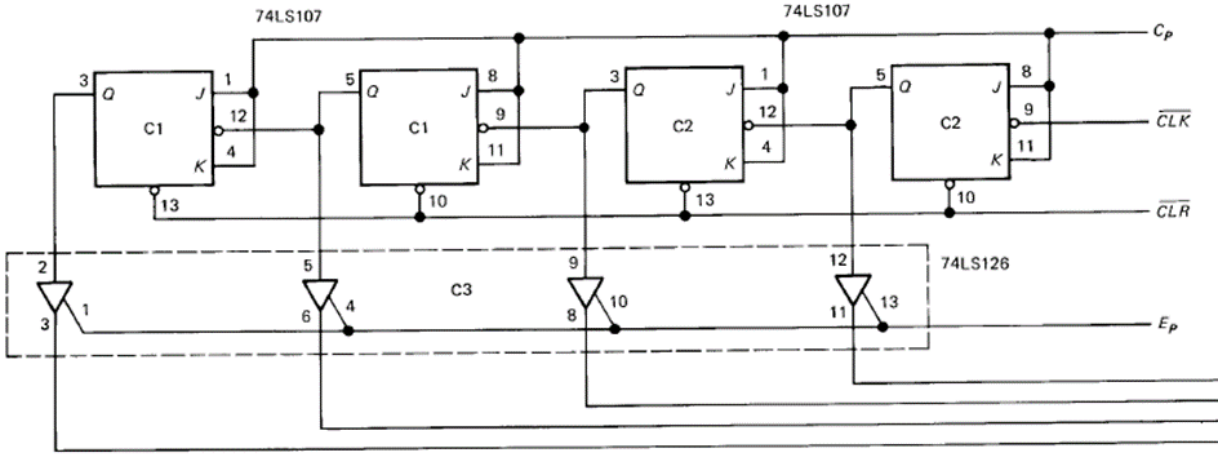
1. **Initialization:** The Program Counter is initialized to the starting address of the program. This is usually done during the start of the execution or after a jump instruction.
2. **Incrementing:** After each instruction is fetched and executed, the Program Counter is incremented by the Incrementer, preparing it for the next instruction in the sequence.
3. **Branching and Jumping:** In the case of conditional or unconditional branches, or jump instructions, the Program Counter may be loaded with a new address, causing the program to jump to a different location in memory.
4. **Fetching Instructions:** The address stored in the Program Counter is used to fetch the instruction from the memory unit. The fetched instruction is then placed in the Instruction Register for decoding and execution.

Control Signals for Program Counter:

The control signals associated with the Program Counter module include:

- **Load Counter Register (LdCR):** Enables loading a new value into the Counter Register.
- **Increment Counter Register (InCR):** Enables incrementing the Counter Register.





Instruction Register:

The instruction register as the name implies is used to store the current instruction and forward it to the control unit to be decoded. It will read 4 bits from the bus. It holds the four bits of instructions to be performed and the four bits of memory address where the specific manipulation needs to be done.

Once the values have been read into the register it will output 4 of those bits (the instruction part) to the control unit to be decoded. While the other 4 bits(the value) will be output back to the bus.

Components of the Instruction Register:

1. **Flip-flops:** The IR is commonly implemented using a D flip-flop. The D flip-flop's input (D) determines the value stored in the flip-flop on the rising edge of the clock signal.
2. **Control Signals:** The IR receives control signals that determine its operation, such as a load signal to transfer data into the register and a clear signal to reset the register to zero.

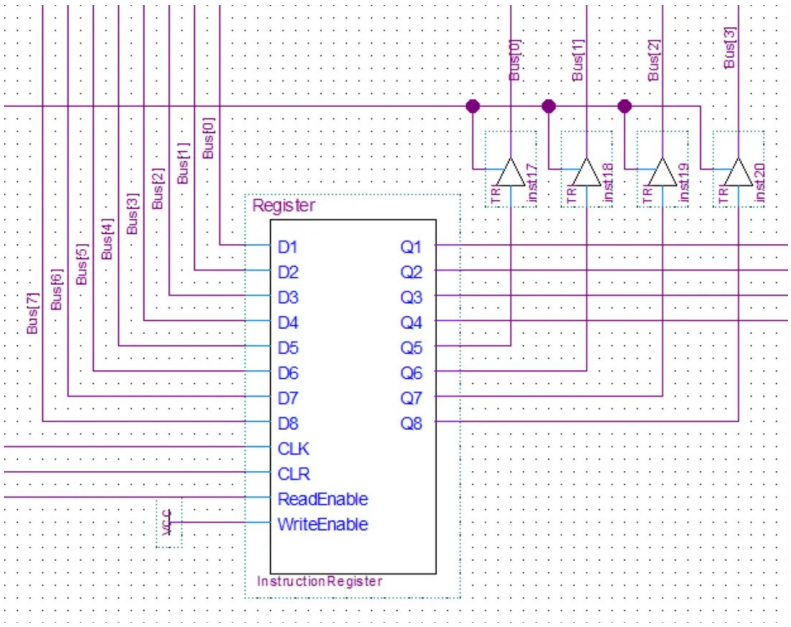
Operations of Instruction Register:

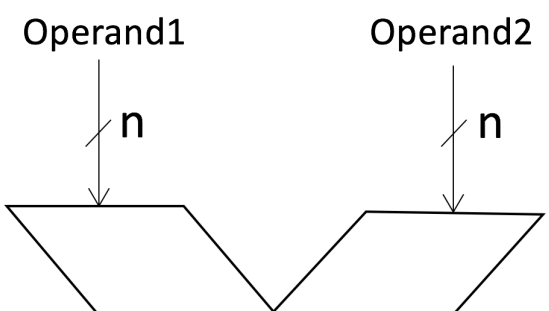
1. **Clearing:** The IR is typically cleared to zero when the CPU is reset. This is done so that the IR does not contain any leftover data from a previous instruction.
2. **Instruction loading:** The IR is typically loaded with the next instruction by the PC. This is done by connecting the Q output of the PC to the D input of the IR.
3. **Instruction decoding:** The IR is typically decoded by the instruction decoder. This is done by connecting the Q outputs of the IR to the inputs of the instruction decoder.

Control Signals for Instruction Register:

The control signals for an instruction register (IR) in an 8-bit computer typically include:

1. **Load (LD):** This signal instructs the IR to load the data from the data bus into the register.
2. **Clear (CLR):** This signal resets the IR to zero, clearing any previously stored instruction.
3. **Enable (EN):** This signal enables the IR to receive and process control signals.
4. **Address Select (AS):** This signal determines whether the data on the data bus is an instruction or an address.
5. **Read/Write (R/W):** This signal specifies whether the IR is reading data from the data bus or writing data to the data bus.





Arithmetic and Logic Unit (ALU)

The Arithmetic and Logic Unit (ALU) is a crucial component responsible for performing arithmetic and logic operations on data.

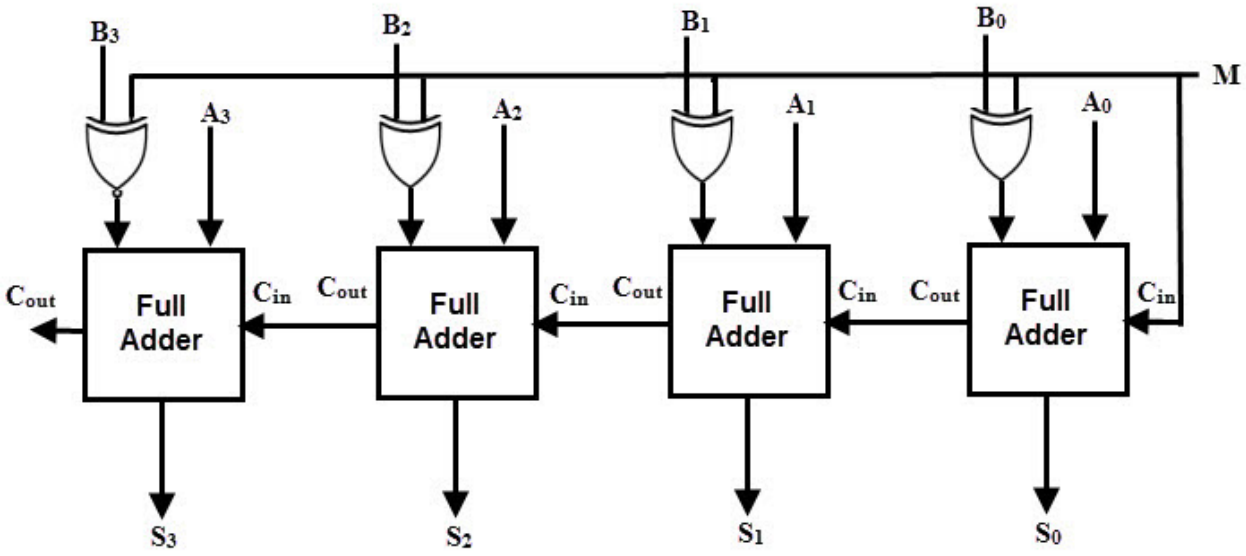
Data Inputs: The ALU typically receives two sets of binary inputs, labeled A and B. These inputs come from registers, other storage locations within the CPU, or directly from the microcontroller.

Interaction with Instruction Set: The instruction register breaks the instruction into two parts: the opcode and the data inputs or locations of the two data acting as inputs to the ALU. The decoded opcode generates a control signal. The ALU receives that control signal dictating the type of operation it should perform. Common control signals include addition, subtraction, logical AND, logical OR, shift operations, and comparison operations (for checking equality, less than, etc.). In our case we would be focusing primarily on addition and subtraction so our control bit M in the adder subtractor circuit will receive the control signal.

Adder/Subtractor Circuit: 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 connected to the Accumulator.

EXAMPLE: If the instruction specifies an addition operation, the control unit 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.



Accumulator:

Accumulator:

The accumulator is a pivotal element within the SAP 1 (Simple As Possible) CPU, serving as a central hub for arithmetic and logic operations. Functioning as a register, the accumulator stores interim results of various operations, contributing significantly to arithmetic calculations, logical comparisons, and data manipulation.

Constituents of the Accumulator:

Registers: Employing one or more flip-flops, the accumulator utilizes these registers to store binary data. The outputs of these flip-flops collectively determine the value held within the accumulator.
Adder/Subtractor Circuit: The accumulator is intricately linked to an adder/subtractor circuit responsible for executing arithmetic operations. This circuit performs additions or subtractions on the accumulator's contents based on control signals received from the CPU's control unit.
Data Bus: Establishing a connection to the data bus enables the accumulator to both receive and transmit data to other CPU components, such as memory or input/output devices.

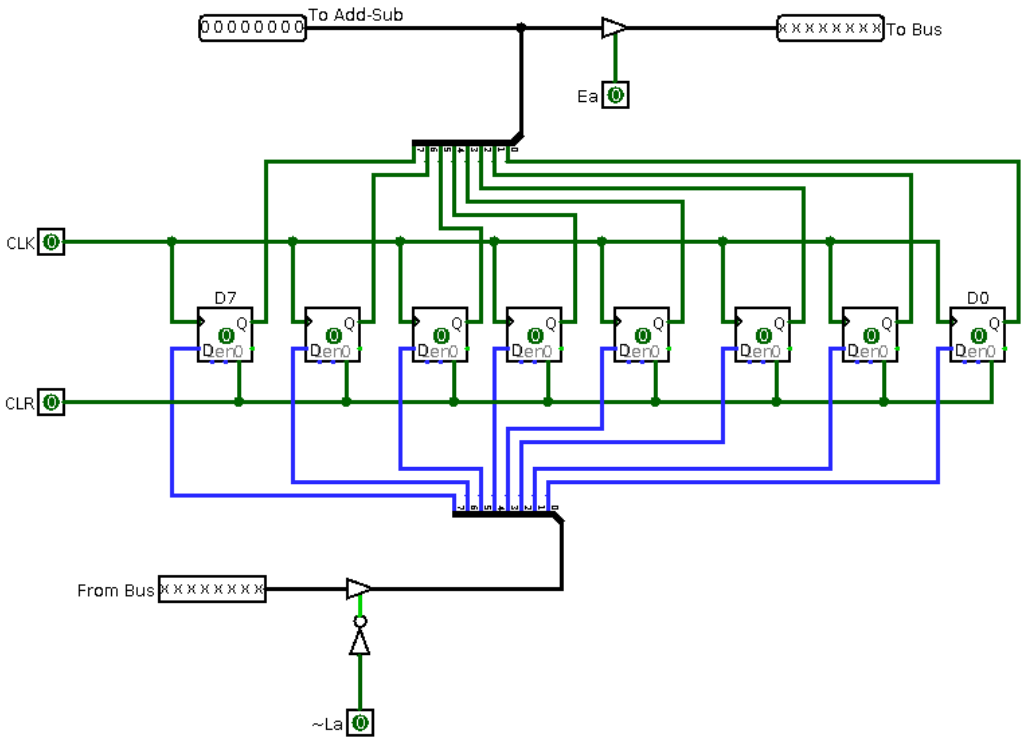
Control Signals for the Accumulator:

Load Signal: Activating the load signal facilitates the transfer of data from the data bus into the accumulator. This signal is instrumental when storing a new value in the accumulator.
Addition/Subtraction Control: The accumulator relies on control signals to determine whether to execute addition or subtraction operations. These signals, conveyed by the control unit, dictate the specific operation to be performed.

Clear Signal: Similar to the instruction register, the accumulator can be reset to zero through a clear signal. This action is typically undertaken during CPU resets or when it is necessary to initialize the accumulator before a new operation.

Operations of 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: Activation of the load signal results in the transfer of data from the data bus into the accumulator, a common prelude to arithmetic or logic operations.
Arithmetic Operations: Integral to arithmetic operations such as addition and subtraction, the accumulator actively engages the adder/subtractor circuit, which processes data stored within the accumulator based on control signals.
Result Output: Following an operation, the final result stored in the accumulator can be forwarded to the data bus or other CPU components for subsequent processing or storage.



Final Implementation

Now that we know about all the components lets see how all of these together would operate together to give you a working example of how SAP-1 actually works.

Operation	Description
Load	Load data from controller to the accumulator
Add	Add data from controller to value in the accumulator
Subtract	Subtract data from controller to value in the accumulator
Output	Output data from accumulator
Halt	Stop processing

- At the start, when the circuit starts to operate the program counter sends the address of any one pin of the microcontroller to get an instruction from there.
- Once the instruction is received it goes into the instruction register which breaks that instruction into 2 parts, where the first part indicates what operation is to be performed while the 2nd part is the operand.
- Now if the instruction is to load any data then the data would simply go into the accumulator.
- If the instruction is to add the data then this is where ALU would come. The data that is coming from the instruction would be added with the present data in the accumulator and the final result after addition would go back into the accumulator.
- The output instruction would simply display the current value in the accumulator to the terminal or LED while the halt instruction would stop our system.