Byte Squad

November 24, 2023

Phase-2
Digital Logic Design

Zehra Ahmed 26965
Anusha Randhawa 27095
Muhammad Haaris 27083
Hamna Inam Abro 27113
Hussain Ali Shah 27131
Abdul Nafay Indrawala 26951
Saqlain Kazmi 26869
Zara Masood 26928

Contents:

0.1 Work Breakdown:

The Simple As Possible (SAP-1) is an 8-bit computer .In this project, We aim to recreate and simulate the SAP-1 architecture using the Wokwi platform to explore the fundamentals of computer architecture.

Wokwi, as the simulation platform provides a user-friendly interface and comprehensive support for hardware simulation. We will use Wokwi's drag-and-drop feature to choose from the various components, such as switches, LED's, Resistors, Gates and Adders to set up with SAP-1 registers, the ALU, and the control unit.

The SAP-1 comprises the following components:

- 4 bit Program counter
- 4 bit MAR (Memory Address Register)
- 8 bit IR (Instruction Register)
- 8 bit Accumulator
- 8 bit B register
- 8 bit Output Register
- Arithmetic Logic Unit (ALU)
- Microcontroller (manages the memory (RAM) and control unit)

We've divided the work such that each group member will cover each of the above components, including the circuitry and the simulation. As for the Microcontroller component, we plan to use Raspberry Pi that will allow us to control and manage the components connected in a SAP-1 computer, as well as implement the memory (RAM).

Along with this, For programming the SAP-1, we're using a simple instruction set that will be implemented through Python. Instructions such as ADD, OUT, LDA, and HLT, each with corresponding machine code representations.

Following this, we'll test our architecture through running sample programs and observing the expected output. For this, Wokwi's simulation tools can be used, that allow for step-by-step execution and signal visualization.

In conclusion, simulating the SAP-1 on Wokwi will provide us with a better understanding of computer architecture. This simulation might be expanded in the future to, for example, implement a traffic light indicator system or a simple calculator, in short, implementing a real-life application. This would require integrating some more advanced features, components that provide the required functionalities and a more refined instruction set.

0.2 Inclusion:

We'll be building our SAP-1 architecture using the following components:

- ➔ *Input* = provides data to the SAP-1 computer. This could be a set of DIP switches for manual input to test each component separately or through our program in python by configuring the GIOP(general input/output pins) of the Raspberry pi board.
- ➔ *Output* = displays the results produced by the SAP-1. This could be a set of LEDs, or an interface that prompts the user.
- ➔ *Memory* = stores both data and instructions. The SAP-1 typically uses a random-access memory (RAM) for this purpose. We'll be implementing the memory through the microcontroller and our program.
- ➔ *Accumulator* (AC) = an 8-bit register for performing arithmetic and logic operations. The Accumulator stores the data before and the result after these operations.
- ➔ *B-Register* = an 8-bit register used for storing data. The B-Register is often used to store the second value being added through arithmetic and logic operations or the 2's complement form of the value being subtracted from the data in the accumulator.
- ➔ *Instruction Register (IR)* = an 8-bit register, holds the current instruction being executed.
- ➔ *Memory Address Register (MAR)* = a 4 bit register that contains the address of the memory location to be read from or written to. This address has gotten into the MAR from the Program Counter.
- ➔ *Program Counter (PC)* = a 4 bit register that keeps track of the memory address of the next instruction to be fetched and executed.
- ➔ *Control Unit* = controls and coordinates the operations within the SAP-1 computer, based on the decoded instruction and generates control signals. We will implement this using the Raspberry Pi Microcontroller and our program in Python.
- ➔ *Arithmetic and Logic Unit (ALU)* = Performs arithmetic and logic operations based on some combinational circuits, on data from the Accumulator and the B-Register. We will only be implementing the adder part of the ALU.
- ➔ *Output Register* = holds the 8-bit result produced by the ALU before it is stored in the Accumulator or transferred to memory.
- ➔ *7-segment Binary Display* = displays the 8-bit result from the Output register through the eight light emitting diodes.
- ➔ *Clock* = provides the timing signals necessary for the synchronized operation of the SAP-1 components.
- ➔ *W-Bus* = a single 8 bit bus for address and data transfer. All register outputs to the W Bus are three-state, allowing orderly transfer of data.

0.3 Exclusion:

- ➔ We aim to implement the memory unit of the SAP-1 through the Raspberry Pi microcontroller board along with the control unit, rather than as a separate memory component.
- ➔ *Flag Register* = indicates the status of the contents of the accumulator or other conditions happening within the processor. For example, the occurrence of a carry, negative result or zero result. We do not require the flag register, since we're only implementing the adder part of ALU.
- ➔ *Index Register* = used as one means of addressing the memory during Indexed addressing. We'll be implementing the instruction set and our program through Direct addressing.
- ➔ *Stack Pointer* = a register that is used for during the occurrence of program interrupts. We aim to implement the basic functionality of the SAP-1 computer hence this is an exclusion in our project.

0.4 Component-Wise Breakdown:

0.41 Program Counter:

Functional Requirement:

The program counter is a 4 bit binary counter to address the 16 memory locations.The program counter begins from 0000 when the program starts (i.e. when the computer is powered on) or if the clock resets. This 4 bit address is passed onto the Memory Address Register (MAR), which in turn allows the fetch the contents at this address from the memory. The counter increments (i.e. points to the next instruction in the program) after each instruction fetch. The program counter will also include boundary checking mechanisms to ensure that memory locations beyond the available address space are not accessed. In addition, the program counter includes a clock signal that ensures synchronization, such that the program counter state is updated only at the next positive clock edge. This also ensures synchronization with other components in the SAP-1.

Input and Output Specifications:

The Program Counter in the SAP-1 architecture comprises D-flip flops (used to store the current value of the program counter. This value is then updated on the basis of control signals such as increment, load and reset. Moreover, there is a Clock source in this component that provides clock pulses for synchronization and step-by-step execution. This clock pulse updates the program counter at every positive clock edge. The Program counter also includes Half Adders (constructed using logic gates) that perform the incrementing mechanism after every instruction clock cycle. The inputs to the Program counter include a Clock signal (CLK) and Clear signal (CLR) - resets the PC to 0. The outputs of a program counter include a 4-bit memory address, that is being read from or written to. The output is connected to the W bus, however to test and visualize the current value of the component itself we used LED’s.

0.42 Microcontroller Processing (Controller/Sequencer and Memory) :

Functional Requirement:

The sequencer in the SAP-1 (Simple As Possible) computer system serves a pivotal role by orchestrating signals that regulate the computer's operations, ensuring precise timing and synchronization. Its primary function involves dispatching control words—12-bit output signals that dictate how the system's registers respond during the subsequent positive clock edge. These control words, structured as CON = Cp Ep ~Lm ~CE ~Li ~Ei ~La Eu Su Eu ~Lb ~Lo, determine the specific actions and behavior of registers within the SAP-1 computer. In our version of SAP-1 it performs an action such that the output is shown on a 7 segment display.

It oversees the flow of data within the system by controlling various components, including memory, arithmetic logic unit (ALU), registers, and input/output devices. By generating precise control words and managing the timing of operations, the sequencer ensures that each step in the instruction execution process occurs in the correct order and at the right time.

The sequencer's role extends to enabling the fetch, decode, execute, and store phases of instruction execution. It governs the fetching of instructions from memory, decodes these instructions, triggers the appropriate operations within the ALU, and manages the storing of results back into memory or registers.The sequencer in SAP-1 is the vital control unit responsible for coordinating and synchronizing the various components of the computer, ensuring smooth and accurate execution of instructions by generating and dispatching precise control words at specific points in the system's clock cycle.

Input and Output Specifications:

The Sequencer takes 4 bits as input from the Instruction Register. These 4 bits are then used for computation according to the instruction performed. The Sequencer looks over all the components ensuring synchronization of bits and such that everything performs in order. It also implements Memory (Random Access Memory) operations which includes fetch - execute cycle. After completing all the computation, it shows output of the value given as input to perform addition on a 7 segment display.

Source:

https://www.slideshare.net/Jawad_Ahmad/sap-1-47400149

https://karenok.github.io/SAP-1-Computer/

https://prezi.com/fv-ygb2jxu_m/sap1-architecture-instruction-set

https://thecomputerstudents.com/fundamentals/architecture-of-sap-1-microprocessorcomputer/

0.43 Instruction Register(IR):

The instruction register is responsible for holding the instruction currently being executed or decoded and its output is available to the other components of the control unit and the microcontroller(controller sequencer).

The 8 bit instructions received by the instruction register are split mainly into 2 components(nibbles) namely into the opcode and the operand/Address bits.

- The opcode specifies the operations that the Control unit must perform.
- The operand provides information about the data in the memory locations involved in the instruction.

Functional Requirements:

The Instruction Register can be implemented using D-type flip-flops or edge-triggered flip-flops. These flip-flops store the individual bits of the instruction, allowing for the retrieval of the instruction when needed

Common gates such as AND gates and OR gates may be used for decoding control signals or generating clock pulses, but the primary storage of the instruction bits is achieved using D flip-flops

Input and Output Specifications:

Input:

The Instruction Register receives 8-bit instructions via the W bus (data bus). This is the pathway through which external instructions are loaded into the registers.

The Instruction Register requires control signals such as the clock (CLK) for updating the register contents and the clear (CLR) signal for resetting the register contents

The Instruction Register operates in synchronization with the clock signal, ensuring precise timing for the retrieval and decoding of instructions.

Output:

A 4 bit output is given to theW bus (data bus) to be manipulated(computations performed) by other components of the control unit as per the requirement of the 8 bit instruction.

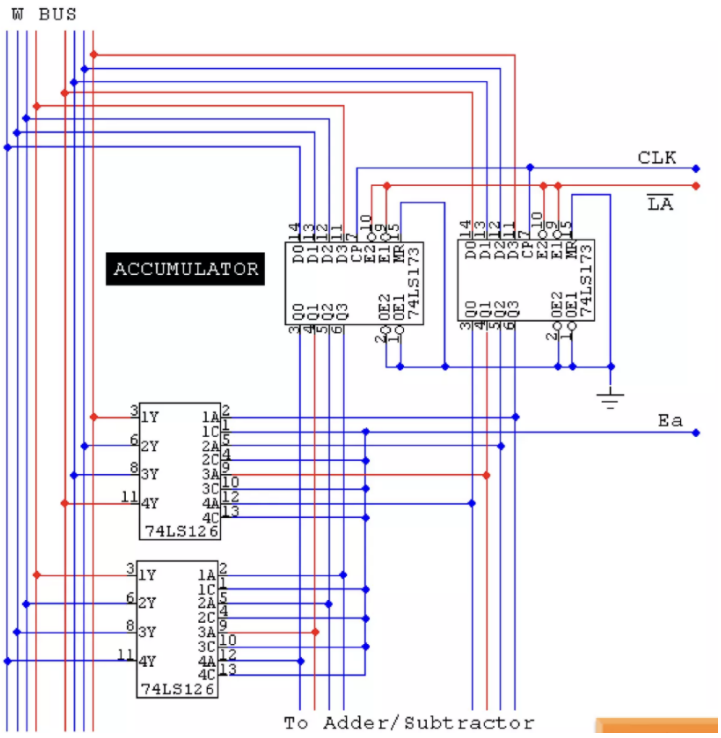
The sequencer scheduler (micro-controller) takes the 4 bit opcode from the Instruction register and generates control signal based on this opcode.

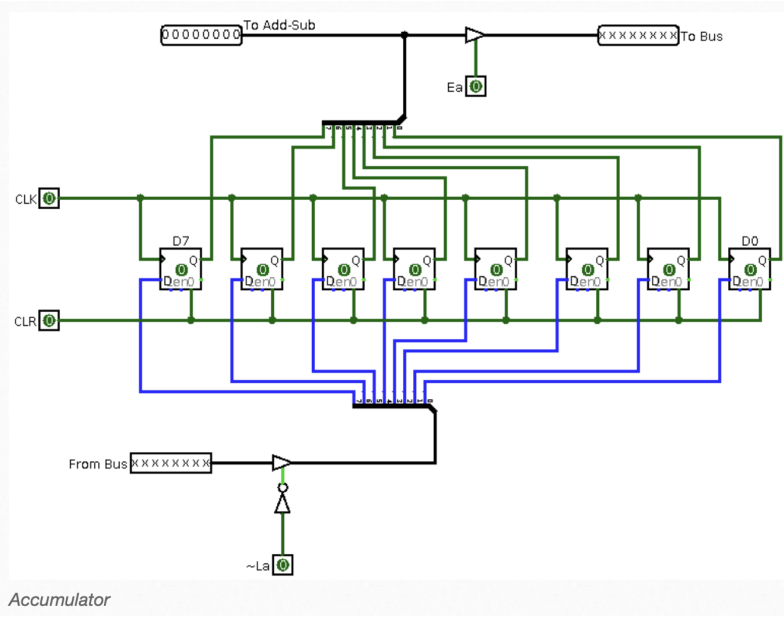
These specifications ensure that the instruction register effectively stores,splits and transfers the instructions within the Control Unit contributing to the overall functionality of the system.

0.44 Accumulator

Functional Requirement:

The accumulator is an 8-bit buffer register that stores intermediate results during a computer run. Normally in SAP-1, an accumulator is always one of the operands of ADD, SUB and OUT. Since our ALU implementation only contains ADD, the accumulator will always be one of the operands for ADD and OUT in our implementation. ADD is a memory reference instruction, but OUT is not.





Accumulator

Input and Outputs for the accumulator are controlled through the Instruction Set. Instructions such as LDA and ADD are used as Input instructions, whereas OUT is used as an Output Instruction. Both input and output of an accumulator are 8 bit binary data values.

LDA stands for “Load the Accumulator.” It is used to Load RAM data into the accumulator. That is ACC ← RAM[MAR]. A complete LDA instruction includes the hexadecimal address of the data to be loaded. For Example, LDA 8H means “load the accumulator with the contents of memory location 8H”, therefore, if RAM[8H] contains 1111 0000, this 8-bit value will become the input to the Accumulator and stored in it.

ADD instruction is used to perform addition in ALU and make use of and store the result temporarily in the accumulator. That is ACC ← ACC + B. For example the instruction ADD 8H means “add the data of memory location 8H with data stored in the accumulator and save the result in the accumulator”. Suppose the accumulator currently contains the binary equivalent of 3, ACC = 0000 0011, and the memory location contains the binary equivalent of 2, RAM[8H] = 0000 0010. During the execution, first the data at address 8H in RAM is loaded onto the B register, B = 0000 0010. The ALU performs addition taking the contents of B and ACC and adding them to form SUM = 0000 0101. Then this SUM is loaded into the ACC as input, overwriting the previously held value. The ACC now contains 0000 0101. This answer is stored in the Accumulator through the W bus, when ~La is LOW.

OUT instruction is used to tell the SAP-1 computer to transfer the accumulator contents to the output port via the W bus when Ea is HIGH. The output is again an 8 bit value being sent from the accumulator to the output port to be displayed on the LED Display.

The accumulator has two outputs:

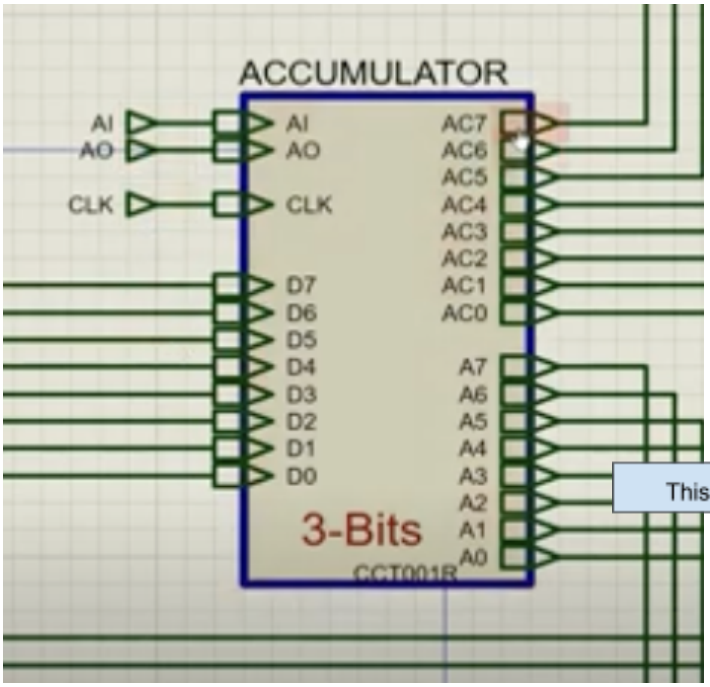
1. One output will go to the Adder in ALU (this output is when ~La is LOW, and value needs to be passed to the ALU to be used in addition. It is a result of ADD instruction)
2. The other output goes to the W Bus through the tri-state buffers. (this is the output when Ea is HIGH, and the contents of the accumulator need to be sent to the output port for LED displays. It is a result of OUT instruction)

Input and Output Specifications:

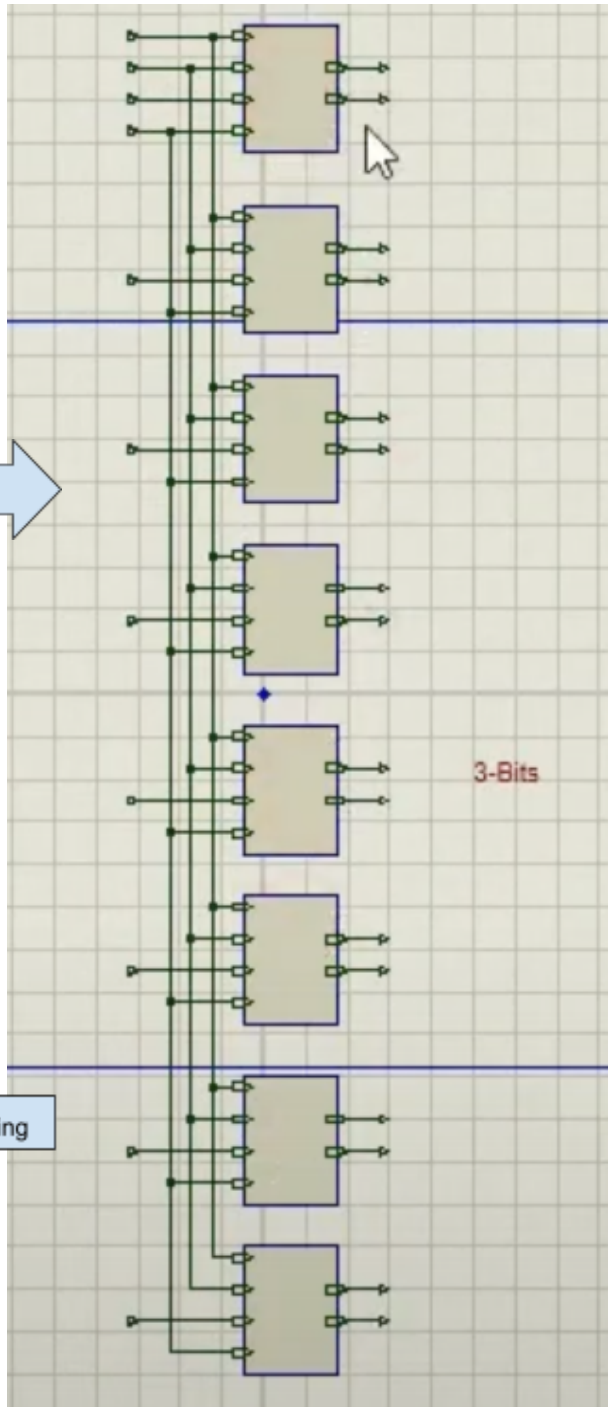
The accumulator will have 11 bits for input: 8 bits for the data, D0, D1… D7, 1 bit for AI, 1 bit for AO, and 1 bit for clock CLK. The accumulator will have two sets of 8 bit outputs each. A0 to A7 and AC0 to AC7. AI and AO are instructions from the controller, when AI is enabled, that is AI = 1, the accumulator will load the data from the data bus into D0 to D7. When AO is enabled, that is AO = 1, the accumulator will send the data loaded into D0 to D7, through output bits AC0 to AC7 to the bus. The main purpose of the accumulator is to load and temporarily hold the data, therefore, in the next clock pulse, it sends out the data through output bits A0 to A7 to the ALU.

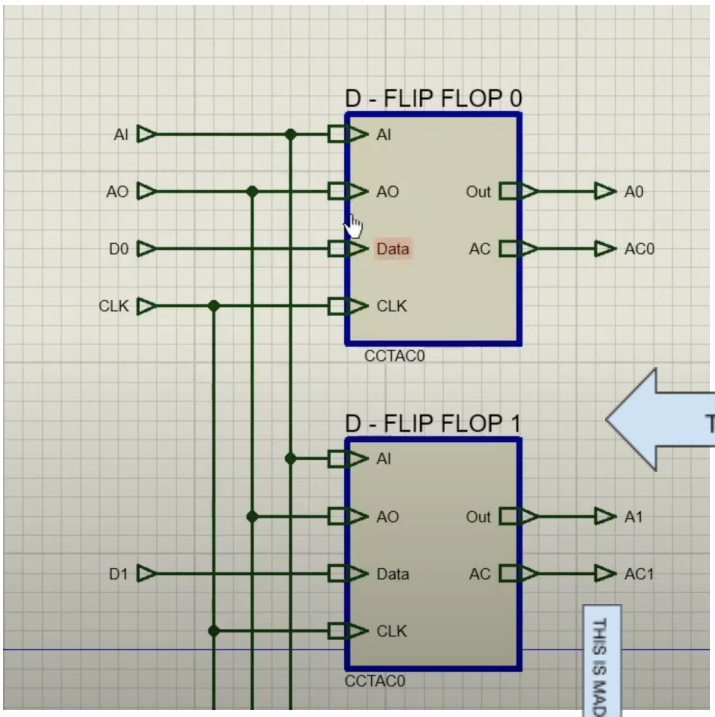
The Accumulator itself is built up using 8 bit registers. Each of the 8 registers is a D Flip Flop with inputs AI, AO, DO, CLK and outputs Ai and ACi, where i = number of flip flop.

Each register is built as follows: the inputs are AO, AI , Data, CLK. There is an AND gate between AI and Data. The CLK is attached to the D flip flop. The outputs are Out and AC. The output from Out is connected to the ALU. When AO is enabled, it is buffered with the output AC to go directly to the 8 bit bus.



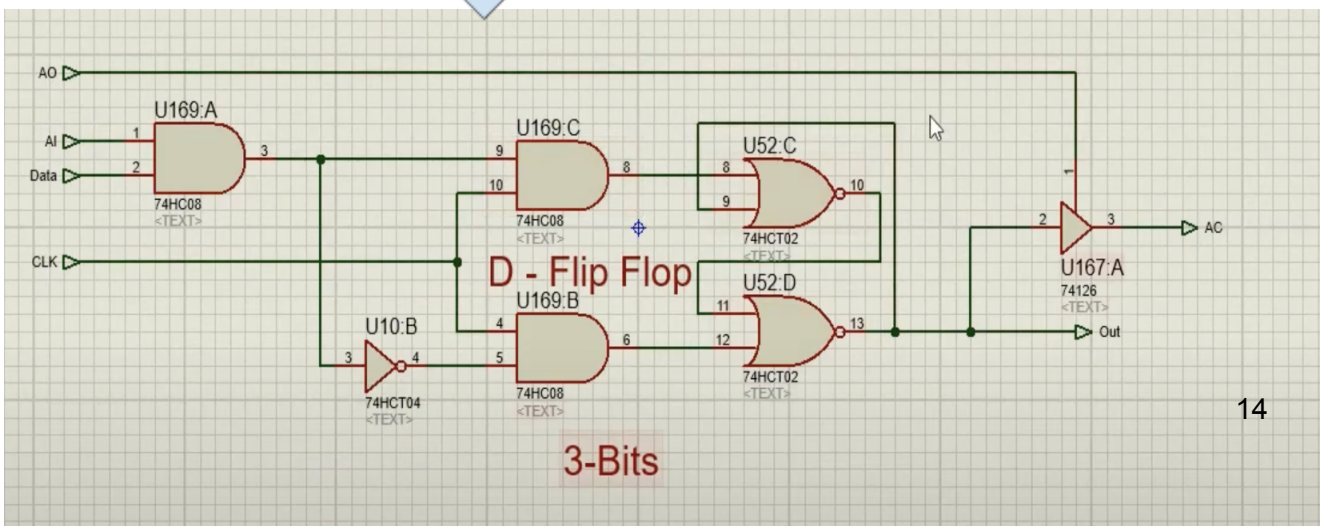
This is made using





This is made using

THIS IS MADE USING



Source:

https://deeprajbhujel.blogspot.com/2015/12/sap-1-instructions-and-instruction-cycle.html

https://prezi.com/fv-ygb2jxu_m/sap1-architecture-instruction-set/

https://www.slideshare.net/Jawad_Ahmad/sap-1-47400149

https://karenok.github.io/SAP-1-Computer/

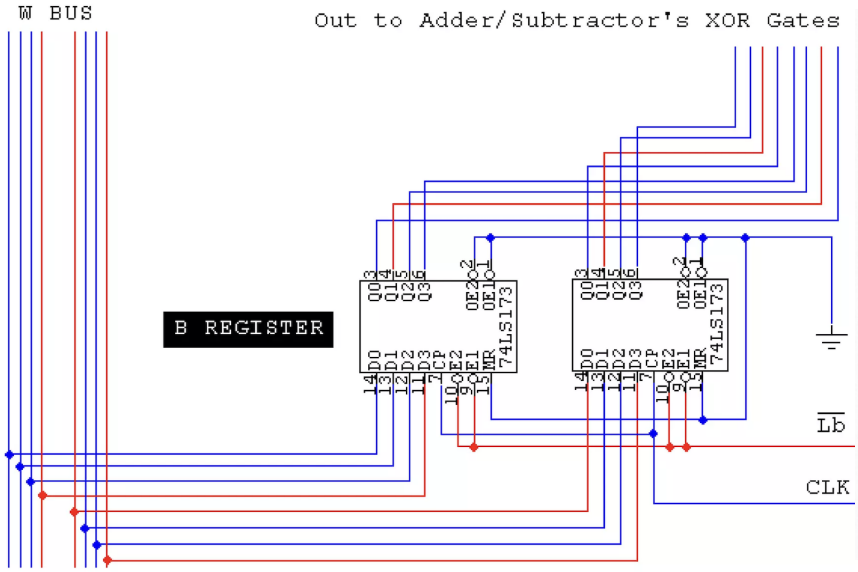
https://youtu.be/cJYIx9asTT0?si=4bWequtdDeBXgFhs

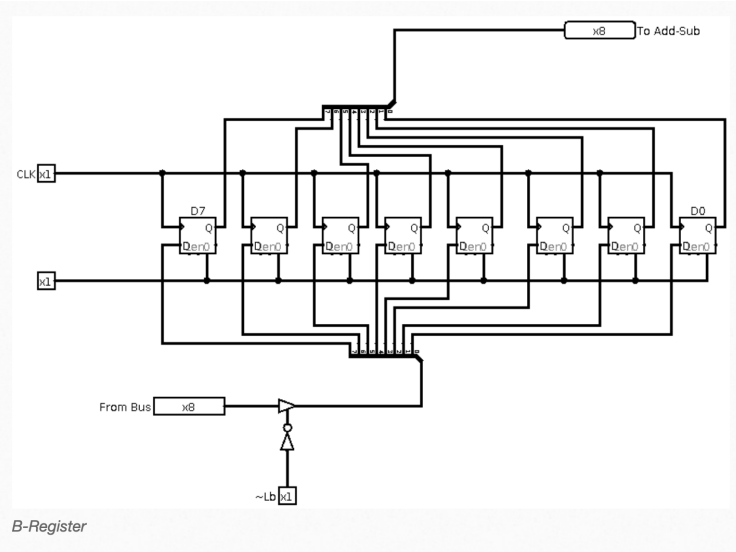
0.46 B Register

Functional Requirement:

In SAP-1, the B register is an 8-bit buffer register which is primarily used to hold one operand (an 8-bit number) of a mathematical operator, with the other operand being held in the accumulator. In our implementation, the B register is being used in Addition, that is ACC ← ACC + B. The addition operation goes like this: (1) MAR ← IR[3…0] , (2) B ← RAM[MAR] , (3) ACC ← ACC + B.

Unlike the accumulator, the B register output is two-state and continuously drives the boxes they are connected to. The main purpose of the B register is to supply the number to be added to the contents of the accumulator to the adder in ALU. When the data is available at the W-bus and Lb goes low, B register loads the data at the positive edge of the clock.





B-Register

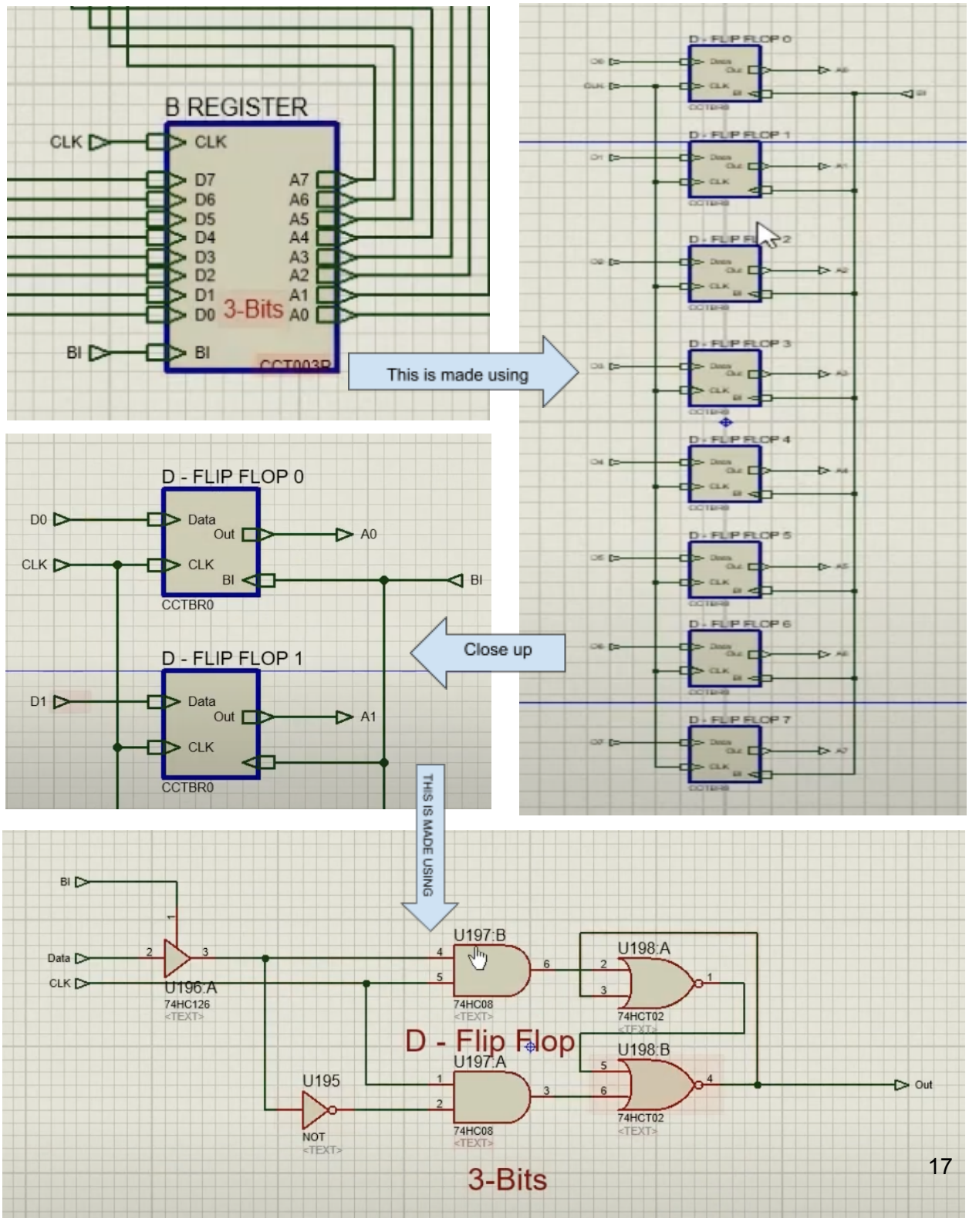
Input and Output Specifications:

The B register has 10 input bits, 1 bit for clock labeled as CLK, 8 bits for the data labeled as D0 to D7, and 1 bit for BI. It has 8 output bits, labeled A0 to A7.

BI is the instruction coming from the controller. When both CLK = 1 AND BI = 1, the B register takes an 8 bit input from the bus through D0 to D7. The bus at that point is bringing data from the RAM. On this same clock pulse, the B register sends out the data through A0 to A7 to ALU for addition.

This B register is in fact made from 8 sub-circuits, each of which is a 1 bit register (made using D flip flops) combined one after the other. Each sub-circuit gets a 1 bit input Di and gives a bit output Ai, where i is the number of the circuit, and 1 input bit for clock and 1 input bit for BI. All 8 sub-circuits have the same CLK and BI connection.

Each individual sub-circuit is made using a D Flip Flop. The D Flip Flop has Data and Bi connected via a tri-state buffer.



Source:

https://deeprajbhujel.blogspot.com/2015/12/sap-1-instructions-and-instruction-cycle.html

https://prezi.com/fv-ygb2jxu_m/sap1-architecture-instruction-set/

https://www.slideshare.net/Jawad_Ahmad/sap-1-47400149

https://karenok.github.io/SAP-1-Computer/

https://youtu.be/cJYIx9asTT0?si=4bWequtdDeBXgFhs

0.45 Memory Address Register

Functional Requirement:

In a Simple As Possible 1 (SAP-1) computer architecture, the implementation of a Memory Address Register (MAR) using flip-flops is a fundamental component. The number of D-type flip-flops in the MAR corresponds to the address bus width of the SAP-1 architecture. Each flip-flop's output is connected to a specific bit of the address bus. This arrangement allows the MAR to hold the binary representation of the memory address.

The SAP-1 design includes control logic for loading the MAR with a new address and incrementing it for sequential memory access. The load signal activates the flip-flops to take in new address values. In operation, when the control logic signals a load operation, the desired memory address is presented to the inputs of the flip-flops, and they latch onto these values. The MAR is then ready to transmit the address to the memory subsystem for read or write operations within the SAP-1 computer architecture.

Input and Output Specifications:

The MAR stores the 4-bit address of data or instructions which are placed in memory. When the SAP-1 is running, the 4-bit address is gotten from the Program Counter through the W-bus and then stored. This stored address is sent to the RAM where data or instructions are read from.

Sources:

https://karenok.github.io/SAP-1-Computer/

https://www.sciencedirect.com/topics/engineering/memory-address-register

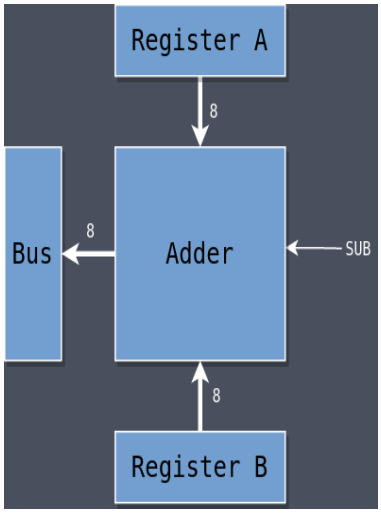
0.46 Arithmetic Logic Unit:

Functional Requirements:

The ALU can function as an 8 bit adder/ subtractor. We're implementing an ALU that only performs addition . It uses 8 XOR gates and 8 Full Adders to function as an 8 bit adder. Hence, the ALU , after receiving two values from the Accumulator and the B register , performs addition on them and sends the result out to the W bus. Since, we're not using flags in our SAP-1 model, the values to be added must not add to result in a value with an overflow bit.

Input and Output specifications:

The Adder takes inputs form Accumulator and the B register. The B register holds the second value to be added. The result after the operation is stored in the Accumulator. After performing the addition operation,when the EU bit (from the controller sequencer) is high, the result is sent to the 8-bit W-bus.
If the SAP-1 implemented performs subtraction as well as addition, then the ALU receives a bit called SU from the controller, which specifies which of the two operations (addition/subtraction) is to be carried out.



Sources:

https://study-for-exam.blogspot.com/2013/06/describe-sap-1-architecture.html
https://www.youtube.com/watch?v=CGKY7P-BHb8
https://www.slideshare.net/Jawad_Ahmad/sap-1-47400149
https://thecomputerstudents.com/fundamentals/architecture-of-sap-1-microprocessorcomputer/
https://www.rt-rk.uns.ac.rs/sites/default/files/materijali/predavanja/SAP_Malvino_Description_Digital_Computer_Electronics_Malvino_0.pdf

0.47 Output Register

Functional Requirement:

Basically, the purpose of output 8 bit Register is to store results to display, when requested, it receives data from the accumulator via bus. The output Register is a block made of 8 D flip-flops with 2 more inputs for clock and OI. Since it's an 8 bit register, it should have 8 bit data inputs which would be connected to 8 D flip-flops(D0-D7), 1 input connected to clock, and 1 to OI/WE(write enable),connected to controller sequence, which controls the register. When the clock and WE signal is high, they signal the register that a write operation is to be performed. When they are active, the data present on inputs should be loaded into the register.

Later, the 8 bit outputs of d flip-flops are connected to any display module through the converter.

Input and Output Specifications:

Input Specification: the register should have 8 bit data input for loading binary data into during writing operations.

Two more inputs:

one input is connected to the clock.

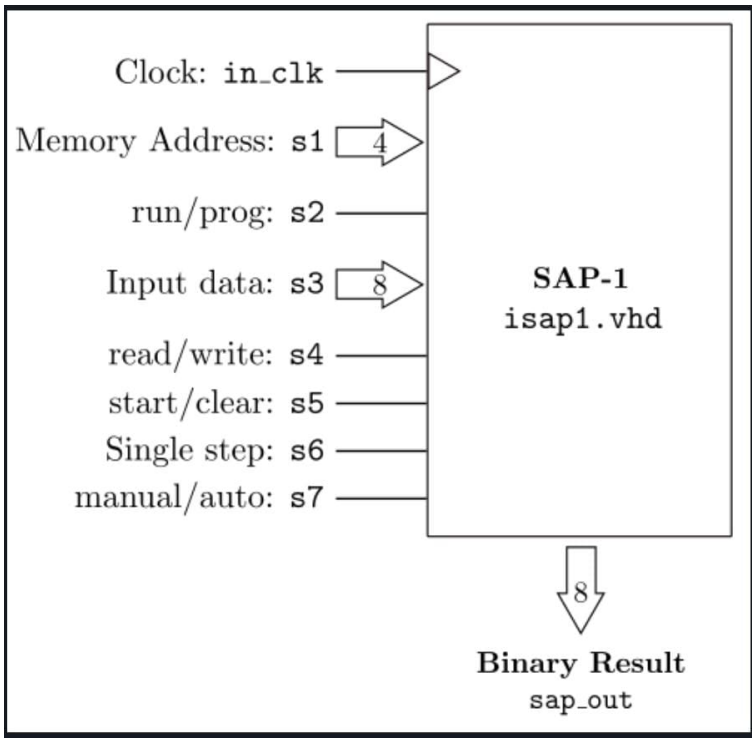
One connected to IO/WE(write enable which basically is a control input**.**

Output Specification:

The registers should have 8 data outputs (D0 - D7) for transmission of data during read operations. The 8 data outputs are then connected to the display module through a converter to show to the outer world.

0.5 Overall Input and Output Specification:

The figure below summarizes the overall input and output given to the SAP-1 architecture. This is Clock source, the 4-bit memory address provided by the MAR via the PC, the program initiation (run command through our program), an 8-bit input data provided by the IR and the required control signals (read, load, clear) provided by the control unit. The SAP-1 computer then produces an 8 bit output that we can visualize through binary displays or LED's.



0.6 Group Planning: