CO221: Digital Design - Hardware Project

SAP-I Computer
from discrete components

Project Team:

- Jaliyagoda A.J.N.M. (E/15/140)
- Herath H.M.M.E.W.L. (E/15/123)
- Karunarathna S.D.D.D. (E/15/173)
- Samarasinghe U.G.S.B. (E/15/316)
- Premathilaka M.P.U. (E/15/280)
- Irfan M.M.M. (E/15/138)
- Tennakoon T.M.P.B. (E/15/350)

Contents

1. Introduction

"SAP" is the shortened form of "Simple As Possible". SAP-1 is the first stage in the evolution of the modern computers. This design contains all the most essential components of a modern computer and it covers many advanced concepts in Computer Architecture.

SAP-1 is a bus organized computer. All the components are connected with a single 8-bit bus (W) through tri-state buffers.

Specifications:

- 8-bit Bus
- 4-bit Program Counter
- 4-bit Memory Address Register (MAR)
- 16-byte Memory
- 8-bit Instruction Register (IR)
- 5 cycle micro-instructions with 16-bit words
- 8-bit Accumulator
- 8-bit B register
- 8-bit Full Adder/ Subtractor
- 8-bit Output Register

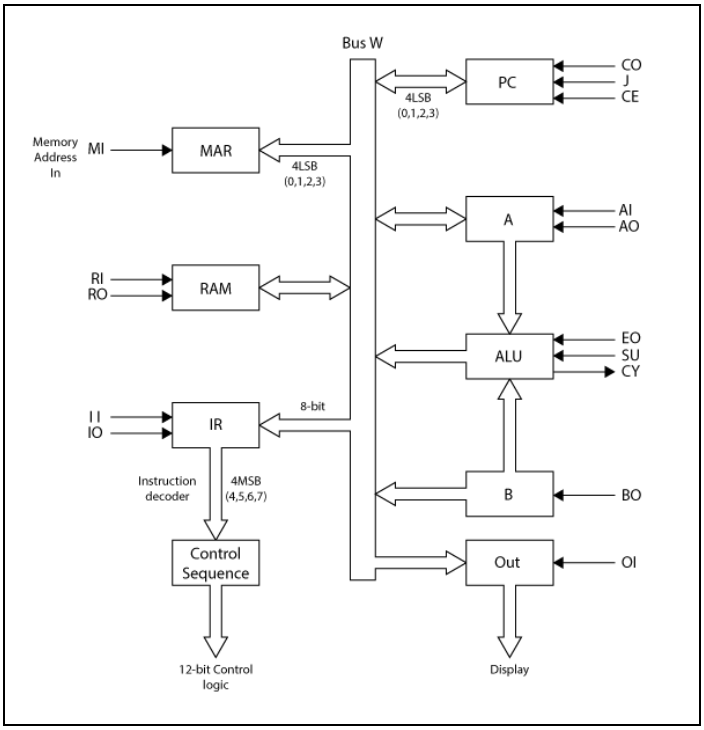

Figure 1.1: SAP-1 Organization

2. Design Procedure

2.1 Modules

All the modules were separately designed, fabricated and finally connected each other on a single circuit board; motherboard which includes all the data and signal buses

Here is the list of modules we were designed

- Clock Module
- Program Counter
- Registers (Register A, Register B, Instruction Register)
- Arithmetic and Logic Unit (ALU)
- Output Register
- RAM and Programmer
- Control Sequencer
- Motherboard

All the modules were designed as *Printed Circuit Boards (PCB)*. The design was done by using Fritzing (A open-source PCB designing software) and Fabricated using PCB Milling Machine.

2.1.1 Clock Module

The computer's clock module is used to synchronize all operations. Basically, there are some methods to implement a clock module for a computer. So this 8-bit computer's clock module was built based on 555 timer IC.

There are two modes in the clock module.

- **A-stable mode:** In this mode, clock pulses are generated automatically. The frequency of the clock pulses depends on the capacitance of the capacitor (connected to pin 2) and the resistance of the resistor (connected between pin 6 and 7). A potentiometer and a resistor are connected between pin 6 and 7 of the IC. So by changing the resistance of the potentiometer, the frequency of the clock pulses can be changed.

- **Monostable mode**: Whenever the user wants to generate clock pulses manually, the monostable mode can be used. But the bouncing effect must be considered. Therefore another 555 timer IC is used to de-bounced the circuit.

Bi-stable mode

There must be a way to switch between astable mode and monostable mode. So the bistable mode is used. Since a toggle switch is used, the circuit must be debounced. Therefore another 555 timer IC is used for that. So the bistable circuit gives a debounced stable output.

Halt Status

When the user wants to halt the computer programmatically, the halt state can be used. Normally the halt input is in law state and it is inverted by a NOT gate. When the HALT instruction was called (in 3rd microinstruction), the circuit was implemented such that the computer halts the execution of instructions.

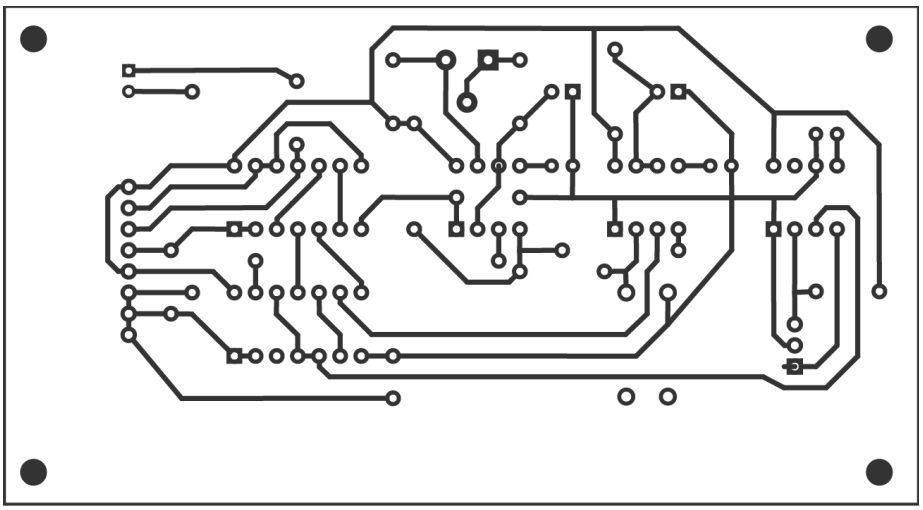

Figure 2.1: Clock Module (PCB Layout)

2.1.2 Program Counter

The program is stored in the RAM as a listed instruction. In order to execute them, the instructions must be fetched from the RAM. So, the program counter is used to keep track of which address is where. It contains the address or the location of the instruction being executed at the current time.

This is a 4-bit counter as the memory has a 4-bit address. This works similar to register A or register B because, it stores a value, sends and receives values from the bus.

Program counter contains 3 control signals.

- **Program count out (CO)**: Sends the stored value to the bus

- ***Jump (Program counter in)*:** Normally the program counter counts in order from 0. This signal is used to jump around or to run a loop. When the jump signal is enabled it takes the 4 least significant bits from the bus (as this is a 4-bit program counter) and put it to the program counter. That will be the next address that executes.

- **Count enable (increment)*:*** To increase the program counter once per instruction cycle, not per clock cycle (because the instruction might take multiple clock cycles). When this is enabled program counter will increase on each clock cycles. So, this signal is enabled to take next instruction (increase the counter value) and then the signal is disabled (store and doesn't continue the increment while executing that instruction until the next instruction).

74LS161 was used as a synchronous 4-bit binary counter. This chip gives the ability to control the "jump signal" by the "load pin 9" and "count enables signal' by the "enable pin 7 and 10". But to the "program counter out signal" a tri-state-logic interface is needed for the bus. So, it can control whether the output of the chip is as the same as the value that going to the bus. For that 74LS245 is used which have tri-state buffers that allow enabling or isolate.

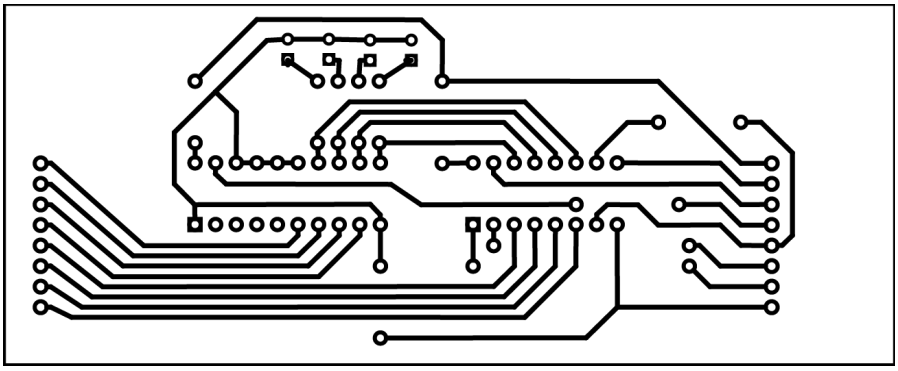

Figure 2.2: Program Counter (PCB Layout)

2.1.3 Registers

The SAP-1 computer is designed with three similar 8-bit register modules. Two of them, Register A and Register B are reserved for storing two input numbers to the ALU. The other module is the instruction register. Each module has two 4-bit D-type registers and one 8-bit tri-state buffer to connect and disconnect the output stream to the main bus.

Control Signals (Register A and B):

- AI, BI, II: Register input enable
- AO, IO: Register output enable
- RST: Reset memory
- CLK: Clock Signal

Data Buses (Register A and B):

- 8-bit Data In (input) from W Bus
- 8-bit Data Out (output) to W Bus
- 8-bit Data Out (output) to ALU

The instruction register is little different from the two other registers. Its 4 least significant bits are used to store current Instruction and wired to the control unit. Most significant 4 bits are used to store instruction data.

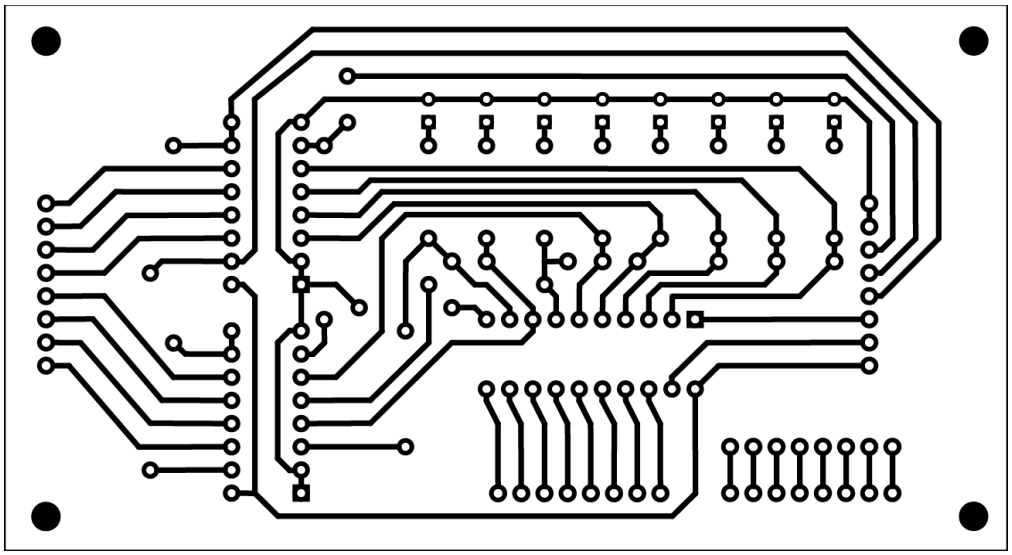

Figure 2.3: Register (PCB Layout)

2.1.4 Arithmetic and Logic Unit

SAP-I is capable of doing two binary operations and they are addition and subtraction. These mathematical operations are done by the Arithmetic and Logic Unit (ALU). Two registers are used to feed data to the ALU.

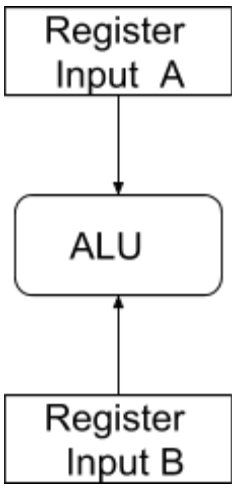

Figure 2.4: Block Diagram of the ALU

Components

- 74LS283 (4-bit binary full adder with fast carry) x 2
- 74LS245 (Octal bus transceiver) x 1
- 74LS86 (Quad 2-input Exclusive OR gate) x 2

Design

There are 8 input lines to the ALU from both register A and register B. Therefore, two cascaded 4-bit adders are used. If the chosen operation is subtraction the XOR panel is in active state. Thus, ALU gets the inverted value in the register B.

Two inputs addition is typical bitwise addition and there is a specific method used to get the binary subtraction. When the subtracting instruction is given to the computer it calculates the 2's complement value of the input B and adds it to the input A to get the result A-B

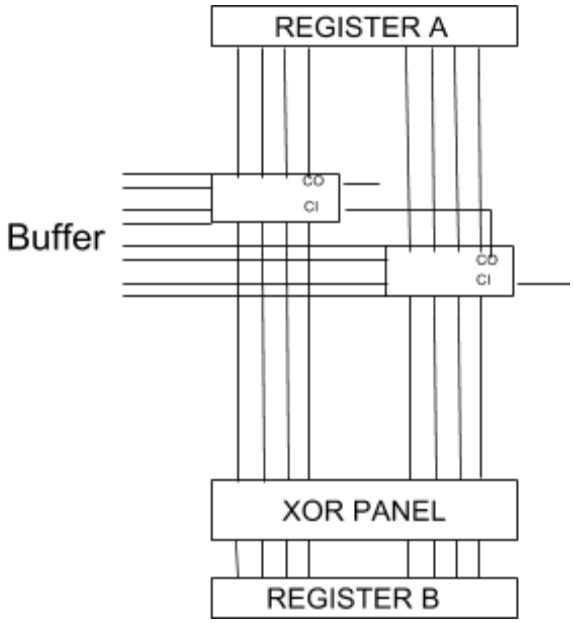

Figure 2.5: Schematic diagram of the 8-bit ALU

The output lines of adders are directed to the tri-state buffer and the buffer is connected to the main bus.

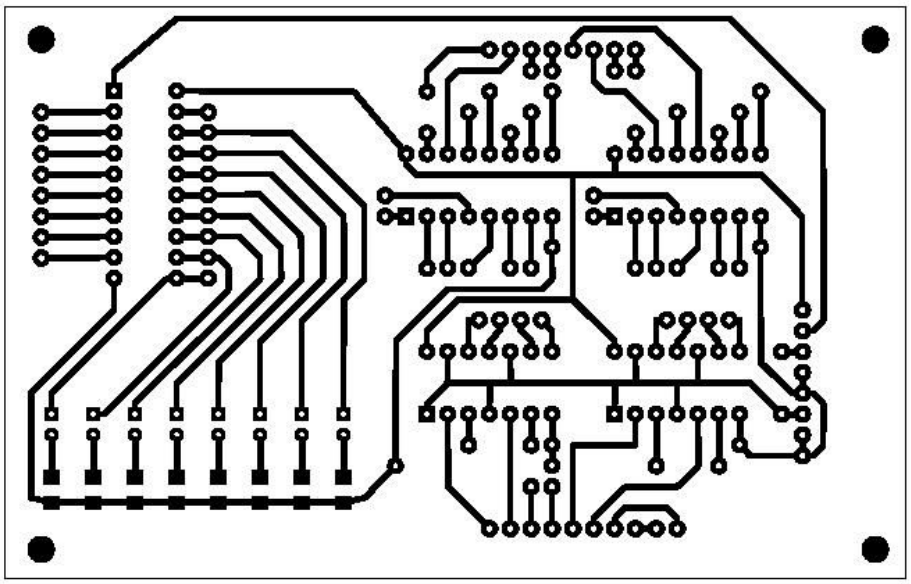

Figure 2.6: ALU (PCB Layout)

2.1.5 Output Register

The Output Display of the SAP-1 computer is designed to display a decimal representation of an 8-bit binary number (0-255). In this design, we have programmed a single EEPROM to display each of the 3 digits of a given decimal number using four 7-segment displays.

Components

- 1kΩ resistor
- 100kΩ resistor
- 0.01μf capacitor
- 555 timer IC
- 74LS08 (QUAD AND gate)
- 74LS76 (Dual JK flip-flop)
- 74LS139 (Dual 2-line to 4-line decoder)
- 74LS273 (Octal D flip-flop)
- 28C16 EEPROM
- Common Cathode 7-segment display x 4

Design

Initially, we have to create a separate clock pulse for the output display that runs at a different speed than the SAP-1 computer. To do this we use a 555 timer connected with a 0.01μF capacitor, a 100kΩ resistor, and a 1kΩ resistor.

We wanted to drive four 7-segment displays sequentially. To do that we wanted a ring counter. We used a 74LS76 IC which include two JK flip-flops. By feeding the output of the first flip-flop to the clock of the second flip-flop we can create a counter which counts from zero to three in binary and that controls which 7 segment display turned on in given time.

After that we need to program the EEPROM to display a certain digit at a certain time at each of the display to create the illusion of the number is displayed continuously without any delay.

The output of the A register has to connect to 8 of the address lines of the EEPROM to input the register the number which is needed to display, leaving 3 empty address lines. By connecting the 2 outputs of the counter to a two remaining address lines we can program the EEPROM to display each digit of the decimal value of the number inputted to the EEPROM using three 7-segments displays (We leave the remaining display empty for this circuit, but hope to use it to display negative numbers with 2s complement notation).

This allows the 7-segment displays to display each digit of the inputted value at each clock pulse in all of the 7-segment displays. Therefore we need a way to keep only the appropriate display on while all of the other displays are at off state for a single clock pulse, therefore, each of the display shows the appropriate digit of the number. To do that we use a 74LS139 decoder. By connecting the two outputs of the counter to the inputs of the decoder we can create a cyclic pulse that goes low at each output in turn. By connecting outputs of the decoder to the four 7-segment displays, we can make the displays to show the appropriate digit of the number inputted to the EEPROM, therefore, creating the display of the SAP-1 computer.

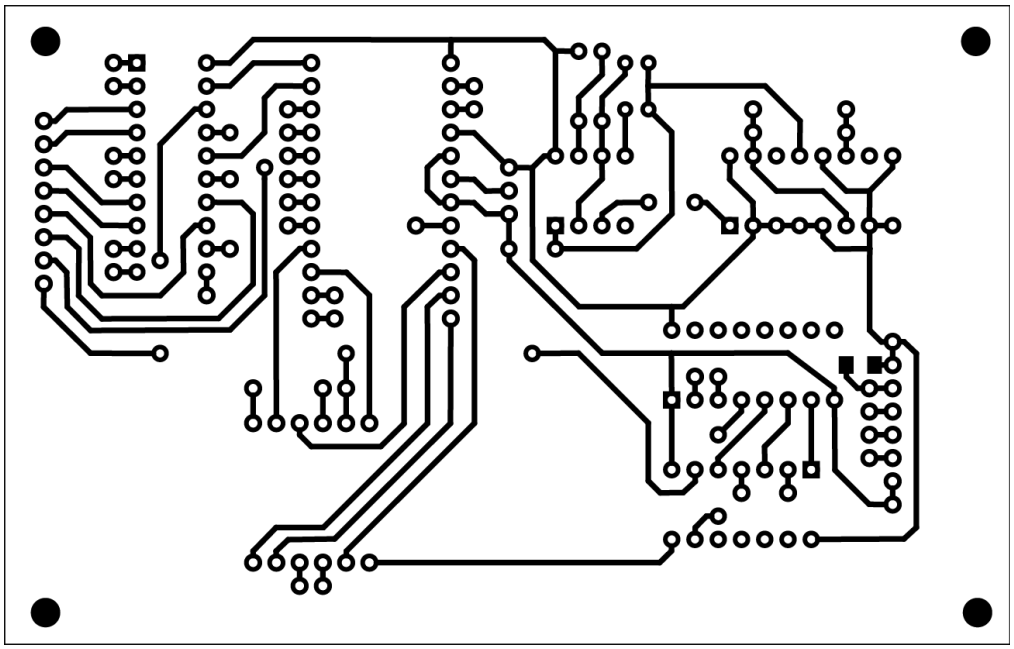

Figure 2.7: Output Register (PCB Layout)

2.1.6 RAM and Programmer

RAM is the module contains all the instructions and it should be programmable from outside. That means the inputs for the RAM may be given through W bus or Manual Switches.

To fulfill this requirement, RAM was designed as two separate modules, Programmer and Memory module.

3.1.6.1 Programmer

The programmer contains 2:1 multiplexer (74LS157) for 4 address bits and for 8 data bits. A toggle switch was used as the control signal for the multiplexer. When it is pressed, module acts as a programmer and input can be gives using small DIP switches. When the toggle button was released, it acts as regular RAM and inputs were taken from the W bus. Memory Address Register also contains in the Programmer Module.

Control Signals:

- ~W: Write memory address (output signal)
- RI: RAM In
- CLK: Clock
- ~MI: Memory address in

Data Buses:

- 4-bit Address (output) to RAM
- 8-bit Word (output) to RAM
- 8-bit Data In (input) from W Bus

3.1.6.2 RAM Memory

Memory module contains two *74LS189* 4-bit RAM ICs. 74LS189 has inverted output pins, so all the outputs were inverted using 74LS04 ICs and connected to W Bus through 74LS245 tri-state buffer.

Control Signals:

- ~RO: RAM Out
- ~W: Write memory address

Data Buses:

- 4-bit Address (input) from Programmer
- 8-bit Data Out (output) to W Bus

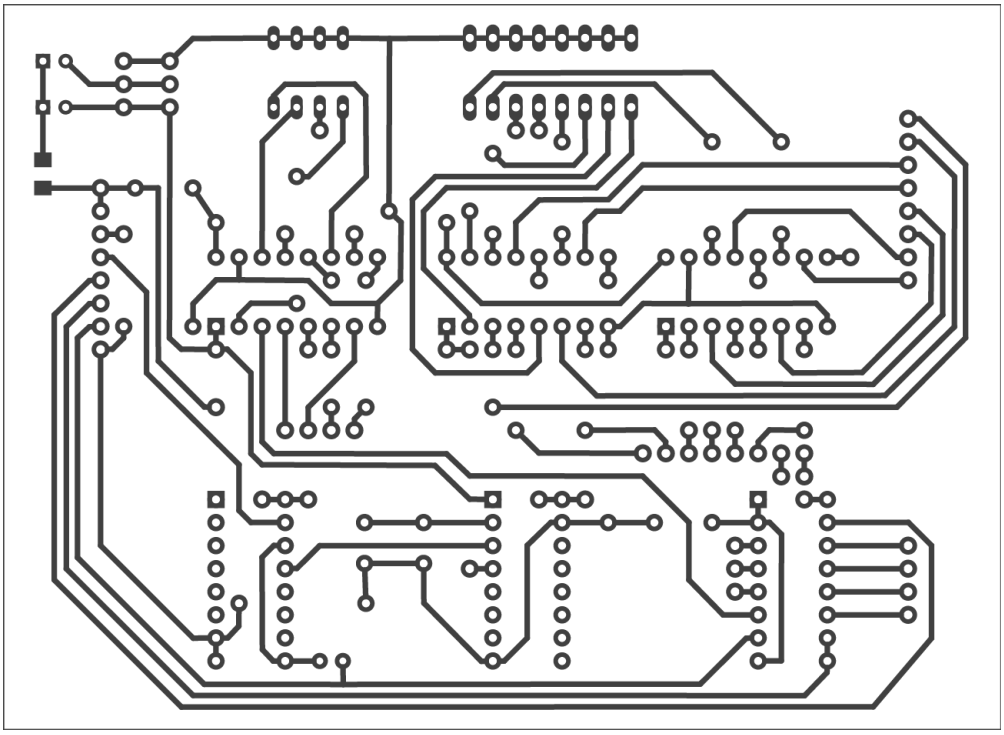

Figure 2.8: RAM Programmer (PCB Layout)

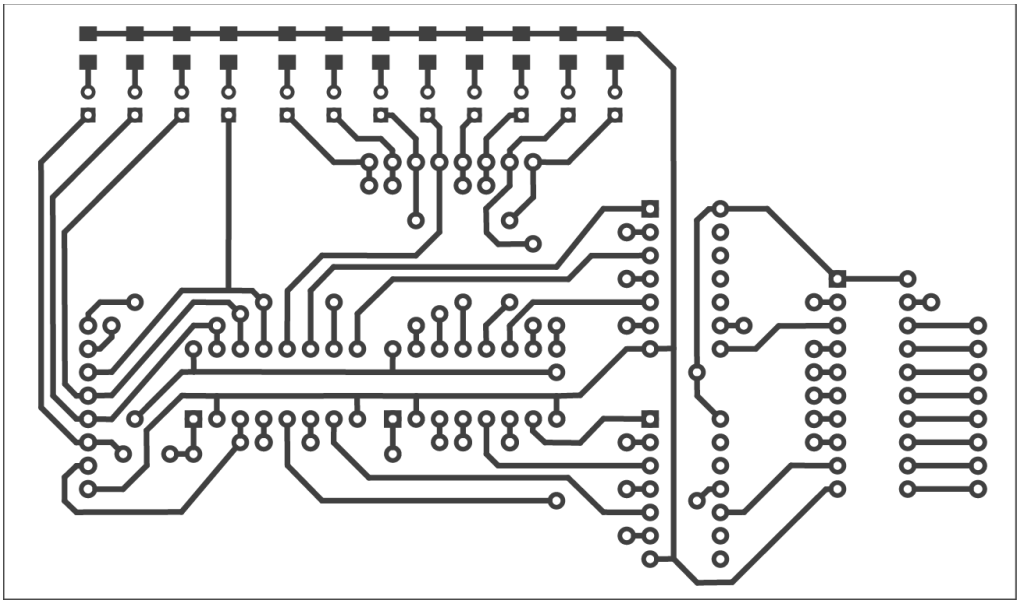

Figure 2.9: RAM Memory (PCB Layout)

2.1.7 Control Sequencer

Control Sequencer is the module that controls all other modules in the SAP-1 computer. Instructions were given from RAM and sequencer contains internal ring counter to execute 5 microinstructions itself.

In addition to providing control signals, it contains a reset switch, which can be used to reset the computer by clearing all memory elements.

Control Sequencer was designed as two modules. The first one contains pre-programmed EEPROM to output control signals for given instruction and microinstruction signals. The second module is for invert control signals if destination module works on active low condition. It acts as an indicator of control signals too.

Inputs/Outputs:

- 4-bit Instruction (input) from Instruction Register
- 12-bit Control Signal (output)
- Reset. ~Reset signals

Control Signals:

- HLT: Standby the computer
- MI: Memory Address In
- RI: RAM In
- RO: RAM Out
- IO: Instruction Register Out
- II: Instruction Register In
- AI: Register A In
- AO: Register A Out
- ZO: ALU Out
- SU: Subtract
- BI: Register B In
- OE: Output Enable
- CE: Counter Enable
- CO: Counter Out
- J: Jump
- X: Unused Pin

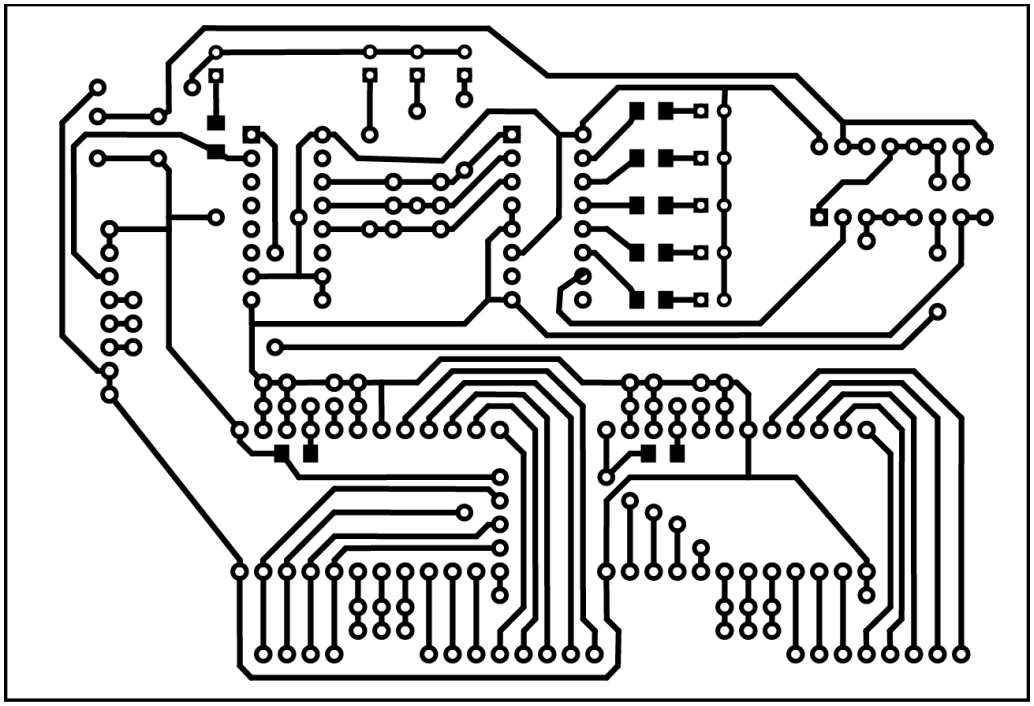

Figure 2.10: Control Unit A (PCB Layout)

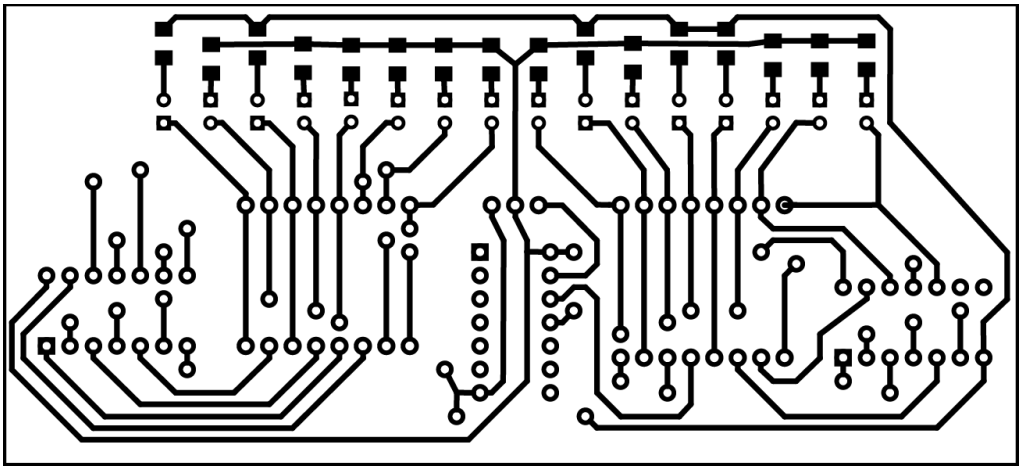

Figure 2.11: Control Unit B (PCB Layout)

2.1.8 Motherboard and Data Bus

We have designed 12 separate modules and wanted a way to connect these all modules. Therefore we designed a motherboard, which has dimensions of 300mm x 430mm. Due to its size, no way to make it as a printed circuit board. As a solution, we choose a plastic board and milled slots to pass the control and data pins from each module to the bottom and wired and soldered them using single core copper wires.

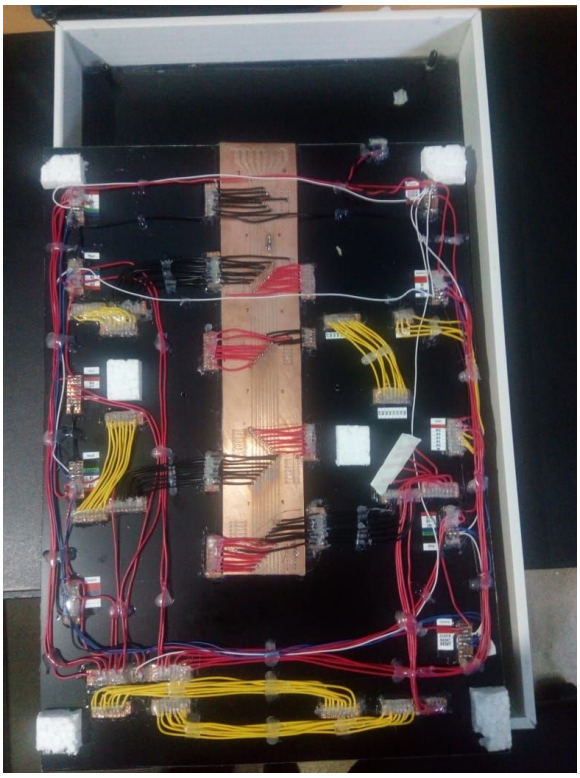

Figure 2.12: A picture of the wirings on the bottom of the motherboard

2.2 Programs and Additional Hardware

We were designed few additional hardware/software which was not included in final the prototype.

2.2.1 EEPROM Programmer

We wanted to program 2kB EEPROM which has 11 address bits and 8 data bits. Therefore we decided to make an EEPROM Programmer using Arduino Mega board (which has 54 digital IO pins)

2.2.2 Assembly Language Instructions

We made a truth table for each instruction and its microinstruction with 15 control signals.

								EEPROM 1								EEPROM 2						
								HLT	MEM ADDRES REG IN (MI)	RAM IN (RI)	RAM OUT (RO)	INSTRUCTION REG OUT(IO)	INSTRUCTION REG IN(II)	A REG IN (AI)	A REG IN (AO)	SUM OUT Σ SM	SUBSTRACT (SU)	B REG IN (BI)	OUTPUT REG IN (OI)	COUNT ENABLE (CE)	PRG COUNTER OUT (CO)	JUMP (J)
FETCH	x	x	x	x	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
	x	x	x	x	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0
NOP	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LOAD A	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADDITION	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
SUBSTRACTION	0	0	1	1	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
	0	0	1	1	0	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0
STORE A	0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LOAD IMMEDIATE	0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JUMP	0	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OUTPUT	1	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HALT SYSTEM	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2.13: The Table we prepared for each instruction

Truth table was programmed into two 28C16 EEPROMs using the EEPROM Programmer we designed with Arduino Mega Board.

RAM Addresses

00	00h	0000
01	01h	0001
02	02h	0010
03	03h	0011
04	04h	0100
05	05h	0101
06	06h	0110
07	07h	0111
08	08h	1000
09	09h	1001
10	0Ah	1010
11	0Bh	1011
12	0Ch	1100
13	0Dh	1101
14	0Eh	1110
15	0Fh	1111

Instructions

	HMRRIIAA LIIOOIIO T	ZSBOCCJ OUIEEO
NOP		
0000 000:	01000000	00000100
0000 001:	00010100	00001000
Load A		
0001 000:	01000000	00000100
0001 001:	00010100	00001000
0001 010:	01001000	00000000
0001 011:	00010010	00000000
Add		
0010 000:	01000000	00000100
0010 001:	00010100	00001000
0010 010:	01001000	00000000
0010 011:	00010000	00100000
0010 100:	00000010	10000000
Sub		
0011 000:	01000000	00000100
0011 001:	00010100	00001000
0011 010:	01001000	00000000
0011 011:	00010000	00100000
0011 100:	00000010	11000000
Store A		
0100 000:	01000000	00000100
0100 001:	00010100	00001000
0100 010:	01001000	00000000
0100 011:	00100001	00000000
LOI		
0101 000:	01000000	00000100
0101 001:	00010100	00001000
0101 010:	00001010	00000000
Jump		
0110 000:	01000000	00000100
0110 001:	00010100	00001000
0110 010:	00001000	00000010
Output		
1110 000:	01000000	00000100
1110 001:	00010100	00001000
1110 010:	00000001	00010000
Halt		
1111 000:	01000000	00000100
1111 001:	00010100	00001000
1111 010:	10000000	00000000

Figure 2.14: Finalized instruction sheet

3. Results

SAP-1 design can add or subtract two numbers, run an infinite loop by executing a given instruction sequence.

Sample Program:

Shows 100 in the display and increase the value shown in it by 7 infinitely.

```
N = 100;
print(N);
while(1) {
        N = N + 7;
        print(N);
}
```

Address	**Assembly Code**	**Machine Code**
0000	LDA 9h	0001 1001
0001	OUT	1110 xxxx
0010	ADD Ah	0010 1010
0011	OUT	1110 xxxx
0100	JUMP 2h	0110 0010
…	…	...
1001	100	0110 0100
1010	7	0000 0111

4. Expenditures

Here is the rough expenditures of our project.

Component	Unit Price	Quantity	Total
NE555	20.00	4	80.00
74LS00	35.00	2	70.00
74LS02	40.00	1	40.00
74LS04	40.00	5	200.00
74LS08	40.00	3	120.00
74LS32	35.00	1	35.00
74LS76	161.00	1	161.00
74LS86	40.00	2	80.00
74LS138	25.00	1	25.00
74LS139	40.00	1	40.00
74LS157	78.00	4	312.00
74LS161	27.00	4	108.00
74LS173	77.00	8	616.00
74LS189	89.00	2	178.00
74LS245	35.00	6	210.00
74LS273	70.00	1	70.00
74LS283	80.00	2	160.00
28C16	250.00	3	750.00
LED (3mm)	1.50	56	84.00
Copper Boards	265.00	2	530.00
Single Core Wire	15.00	25	375.00
7 Segment Display	130.00	1	130.00
Solder (0.8mm)	20.00	20	400.00
Long Leg Headers	6.00	15	90.00
Pin Headers	10.00	5	50.00
IC Bases (8 pin)	3.50	4	14.00
IC Bases (14 pin)	8.00	15	120.00
IC Bases (16 pin)	10.00	27	270.00
IC Bases (20 pin)	10.00	7	70.00
IC Bases (28 pin)	12.00	6	72.00
Total			**5460.00**

5. Conclusions and future works

We hope to add a Flag Register and give an ability to compare values and make conditional jumps. Then we can write programs which have selections (if structure)

Current clock module generates clock pulses on very small frequency range (0.5Hz to 2Hz) and unable to operate in high frequencies. We hope to increase the clock frequency and make it faster.

We hope to improve current design and upgrade it into SAP-II computer which has more instructions and better features.

6. References

1. Digital Computer Electronics *Third Edition, 1999, McGraw-Hills*
   Albers Paul and Jerald A. Brown
2. Build an 8-bit computer from scratch
   https://eater.net/8bit
3. Youtube: Building an 8-bit breadboard computer
   https://www.youtube.com/playlist?list=PLowKtXNTBypGqImE405J2565dvjafglHU
4. 74 Series IC Datasheets
   http://www.skot9000.com/ttl/

7. Appendix

7.1 Project Repository

https://github.com/NuwanJ/peraSAP-I (open-source under Apache License 2.0)

7.2 Project Page

http://nuwanjaliyagoda.com/projects/8bit-computer