**Microprocessor Systems**

**Assignment-1**

**Report**

**Date:** 21-02-2023

**Batch & Section:** BEE-13A

**Group Members:**

Mehboob Khaliq (372682)

M. Saad Munawar (393188)

**Problem Statement:**

**Design a 4-bit number crunching machine. Make a .hex file for it and implement it on the hardware.**

**Working:**

* By following the steps specified in the assignment we designed the 4-bit number crunching machine.
* We made a .hex file for all the instructions we want the machine to perform. Each instruction is specified using 8-bits, in hexadecimal each digit is represented by 4 bits, so in the .hex file we use 2 hexadecimals per instruction.
* The .hex file is then burned onto the EPROM (Ac28c64). This particular EPROM has 8-bits per location and 2^13 number of locations.
* On staring the machine, the program counter represented using the 74160 timer IC is zero. It then executes the instruction the instruction located at the first address.
* Each instruction takes one clock cycle (positive edge triggered) to complete. After the instruction is completed the program counter increments and the next instruction is loaded. In this way all the instructions are executed.

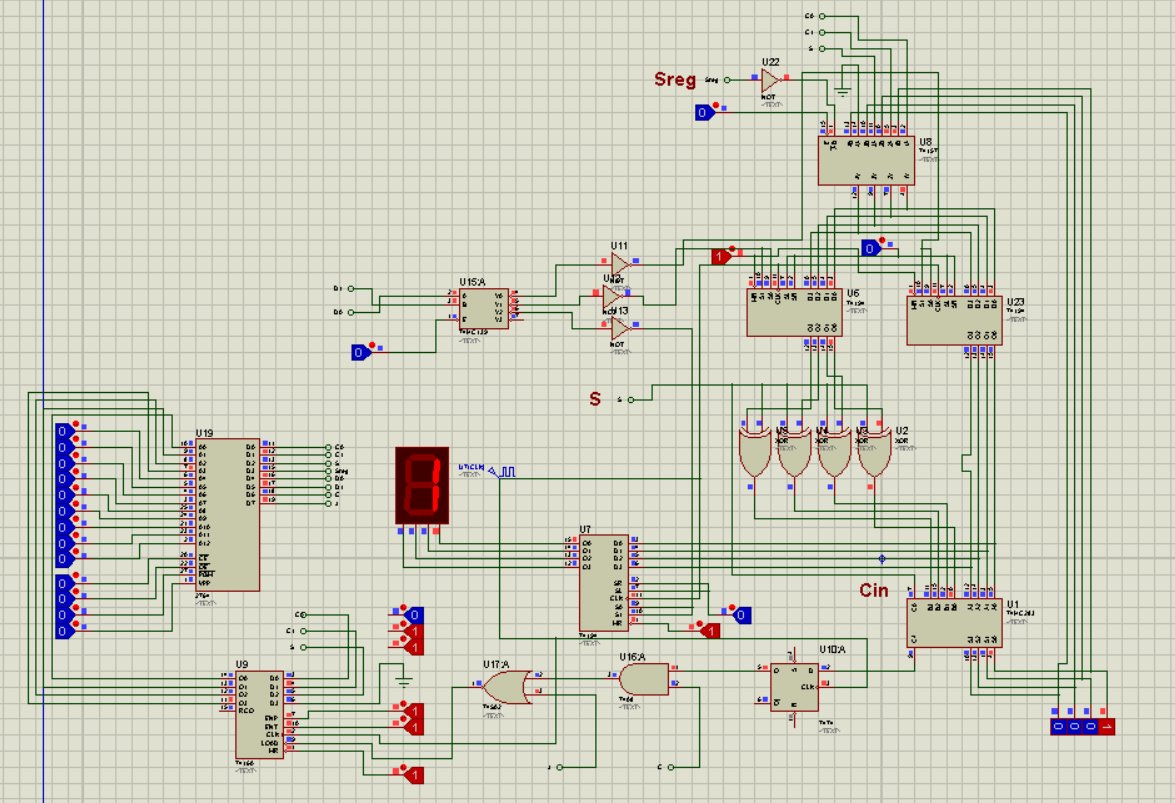
**Components:**

* We have used 74194 ICs for our registers, 7474 D flip-flop IC, 74HC283 IC for our 4-bit binary adder, 74HC139 2-4 bit decoder, 74157 MUX, 74160 counter IC and Ac28c64 EPROM.

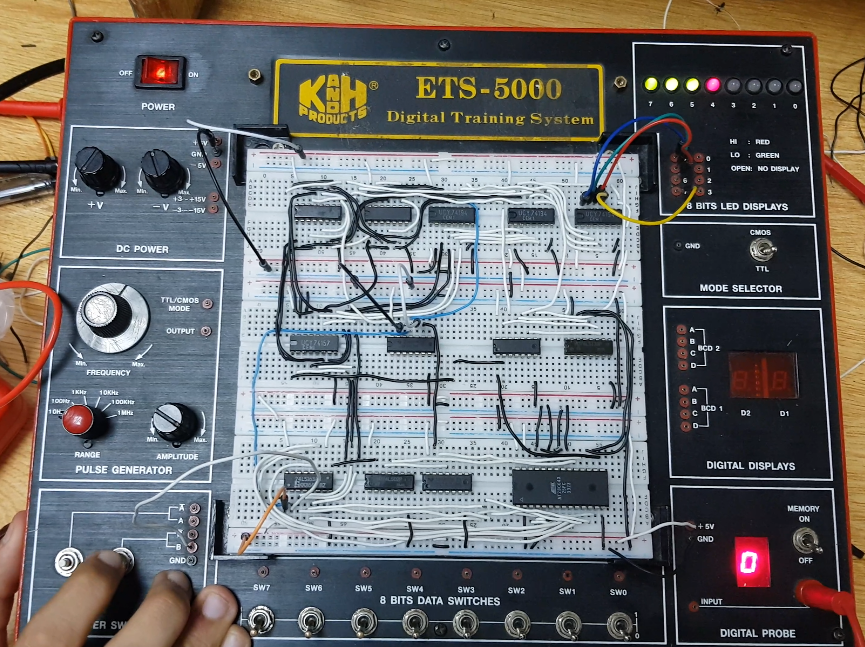
**Instructions:**

* We have a total of 9 types of instructions. According to the table given in the assignment.
* In the first four. While **loading data to a register**, J and C are set to ‘0’. D1 and D0 are selected to enable the register in which the value is to be loaded. When A or B registers are enabled Sreg is set to ‘1’ (so that custom inputs are passed through) and the last three bits represent the number to be loaded. When R0 is enabled then the last 4 bits are don’t cares.
* In the 5th, 6th & 7th. During any **addition or subtraction** operation, J=C=0. D1 & D0 are selected to enable either Register A or B, in which the result is to be stored. Sreg is set to ‘0’ in order to store the ALU output. The S bit tells us whether addition is taking place(0) or subtraction(1). The last two bits are don’t care.
* In the last two. We can also **jump** to another instruction when either J=1, or C=1 and we are getting a carry out (this acts as a carry flag). In this case, D1 and D0 are set to 1 so that no register is enabled and Sreg is set to ‘0’ to avoid custom input being sent to MUX as input. Lastly the last three bits represent the address to be jumped to.

**Proteus Screenshot:**



**Hardware:**

****