

LOGIN SAVER

Nicholas Sajjakulnukit, Brett Shoup, David Martell

Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI

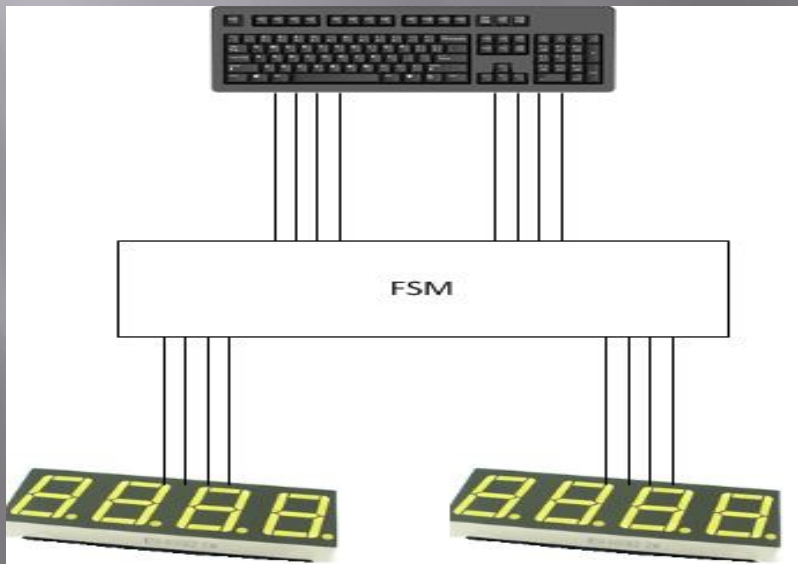
The main motivation for the project is the fact that due to the average person in today's world having multiple passwords and usernames for a wide range of purposes such as email and website accounts, such a system as the Login Registers would eliminate the need to memorize this login data.

Methodology

B

The devices used to implement the digital system are the following:

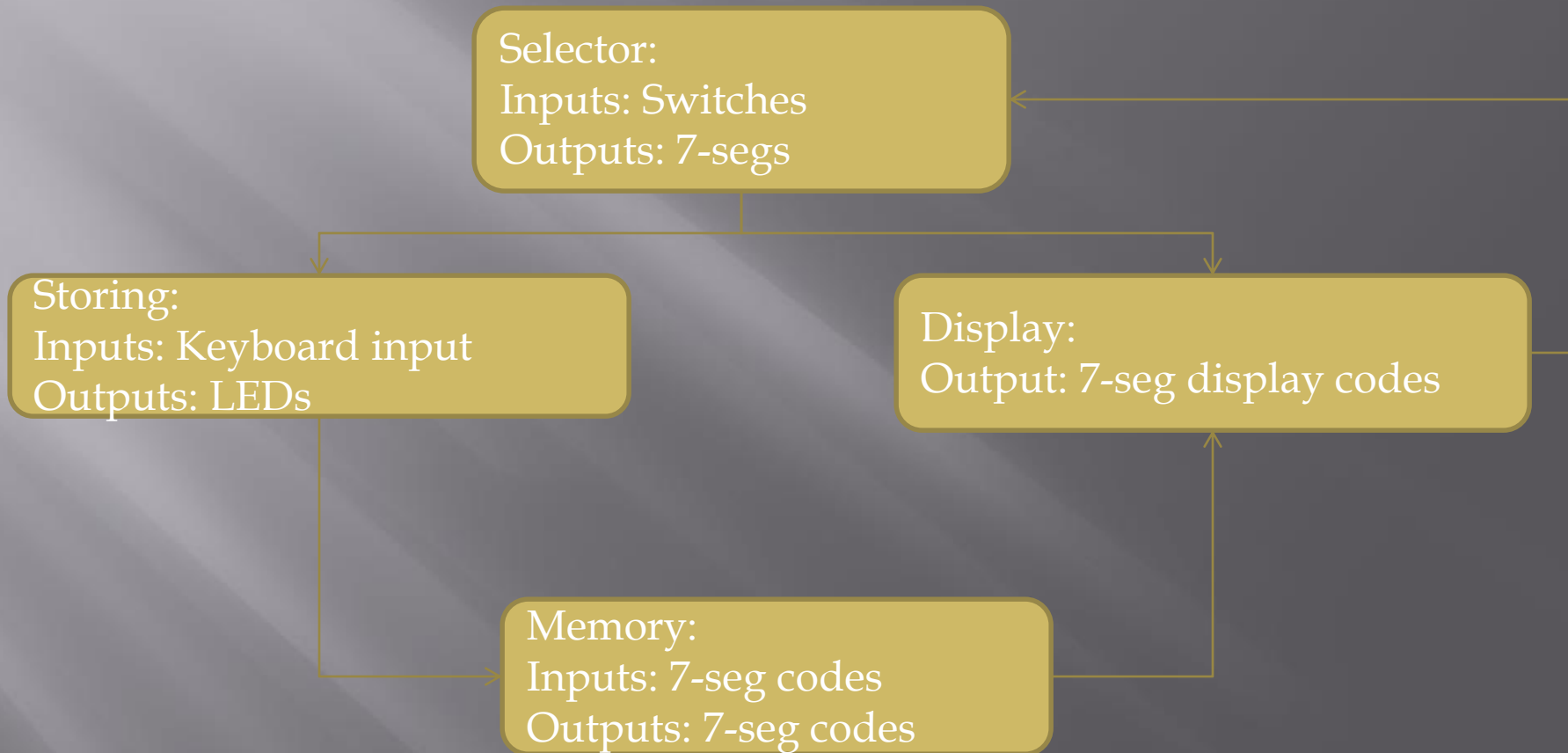
- Keyboard
- Nexys-4 FPGA Board
 - 7-seg display



All 8 inputs are converted to a signal that would output a letter or number on the seven segment display. The keyboard outputs are placed into a FSM and this FSM is designed in such a way as to have the outputs appear as if they are all being displayed on a seven segment display simultaneously when the read/write switch in the read position. Furthermore, when the read/write switch is in the write position, it will display PASS on the seven segment display using a separate FSM.

Login Saver

B

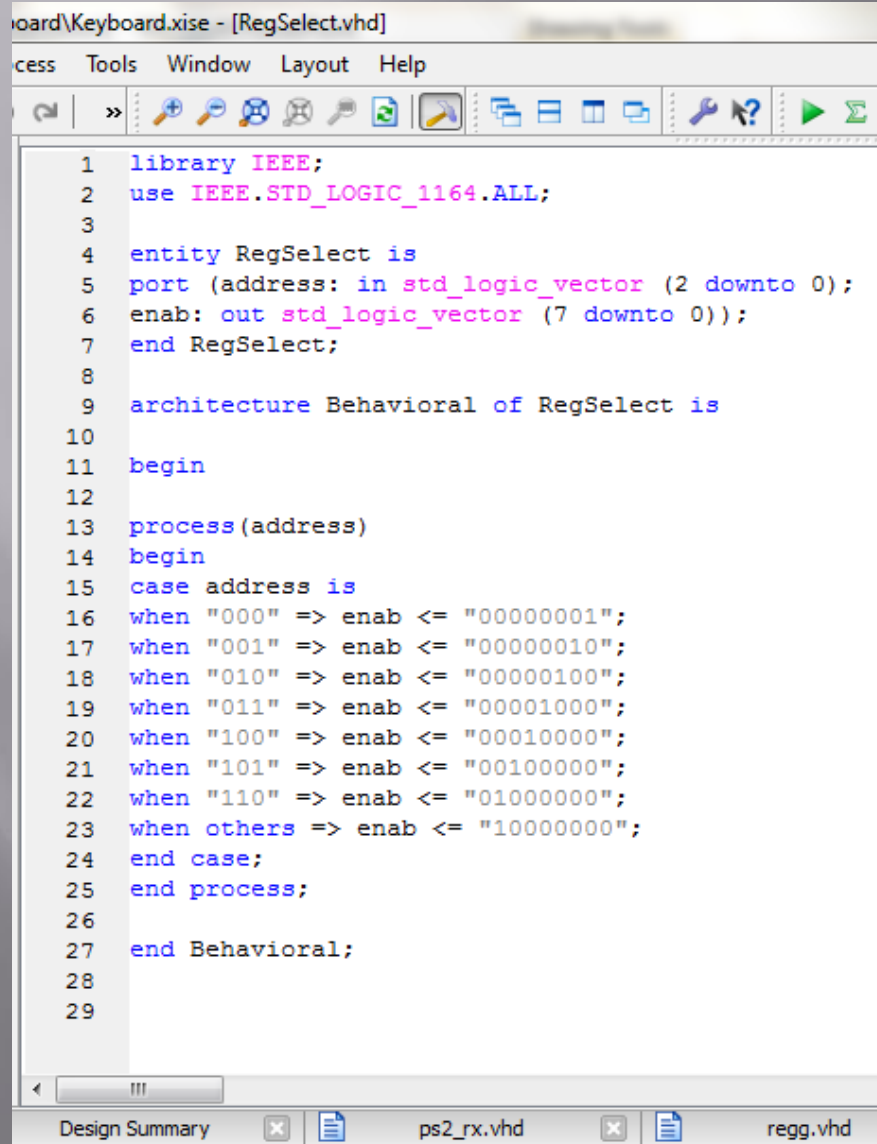


Control: Selector

- ▣ Select store/display with one switch
- ▣ Other switches select each “account”
- ▣ If store/display switch is on, display “save” on disp2. If off, display “disp” on disp 2
- ▣ If no accounts, disp1 is blank. If 1, go to save or display. If 2, display “Err”

Address Selector

D



```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity RegSelect is
5 port (address: in std_logic_vector (2 downto 0);
6       enab: out std_logic_vector (7 downto 0));
7 end RegSelect;
8
9 architecture Behavioral of RegSelect is
10
11 begin
12
13 process (address)
14 begin
15     case address is
16     when "000" => enab <= "00000001";
17     when "001" => enab <= "00000010";
18     when "010" => enab <= "00000100";
19     when "011" => enab <= "00001000";
20     when "100" => enab <= "00010000";
21     when "101" => enab <= "00100000";
22     when "110" => enab <= "01000000";
23     when others => enab <= "10000000";
24     end case;
25 end process;
26
27 end Behavioral;
28
29
```

Reads the address switches and outputs a vector of enable values for the registers
Each enable vector only allows one register to be enabled at one time, meaning only one address is updated at a time

Storing

- ▣ Valid keyboard inputs light up LEDs.
- ▣ These LEDs light up in sequence until all available values within the register are full.
- ▣ Invalid keyboard inputs are not processed as valid, meaning they do not light up any LEDs

Display

- ▣ Calls memory and alternates between stock “USER” and “PASS” messages and the values of the register when reading
- ▣ Simply prompts for input when writing

Rotating Displays

```
Outputs: process (y)
begin
  case y is
    when S1 => sb <= UN0out_reg0;
    pb <= "11111110";
    when S2 => sb <= UN1out_reg0;
    pb <= "11111101";
    when S3 => sb <= UN2out_reg0;
    pb <= "11111011";
    when S4 => sb <= UN3out_reg0;
    pb <= "11110111";
    when S5 => sb <= UN4out_reg0;
    pb <= "11101111";
    when S6 => sb <= UN5out_reg0;
    pb <= "11011111";
    when S7 => sb <= UN6out_reg0;
    pb <= "10111111";
    when S8 => sb <= UN7out_reg0;
    pb <= "01111111";

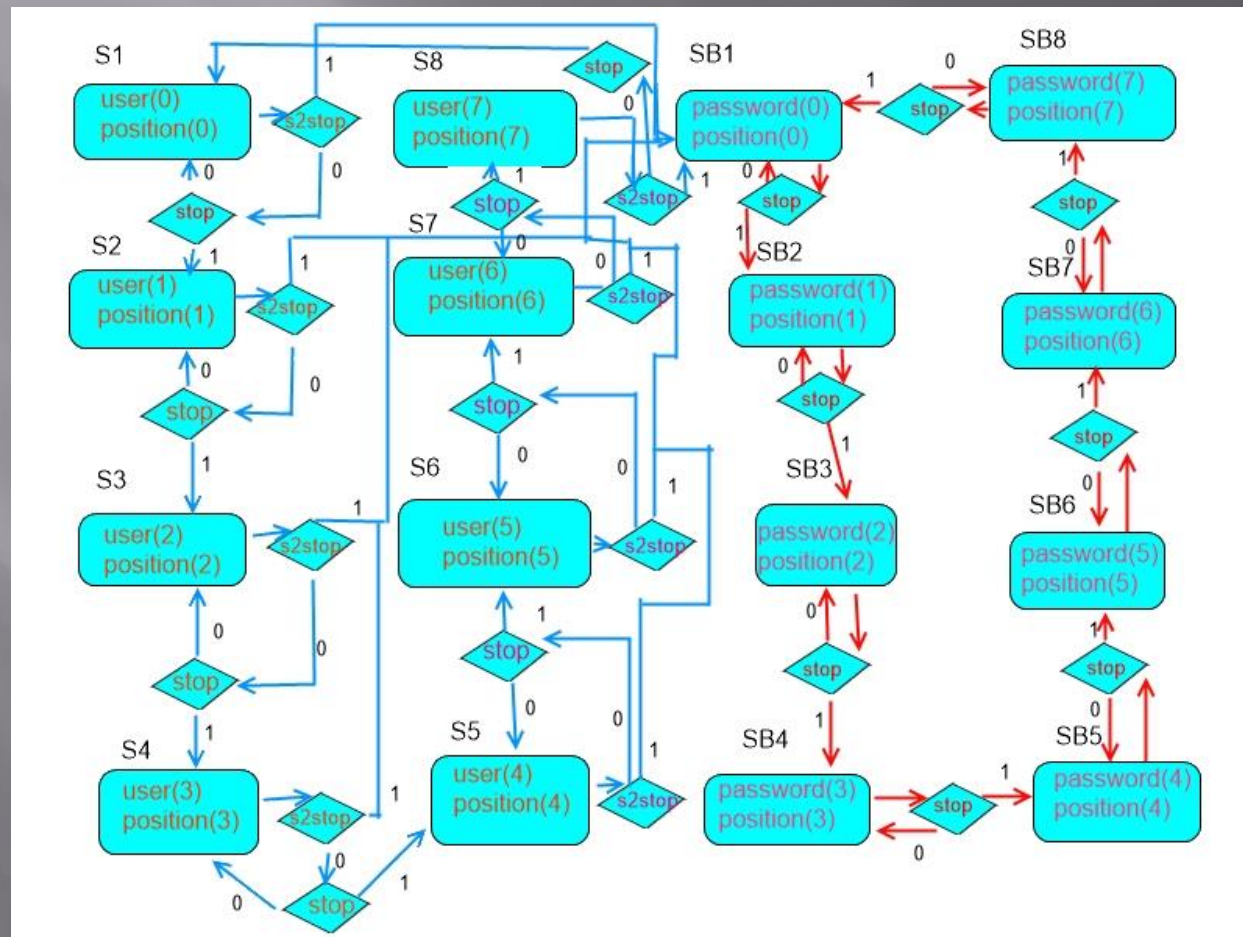
    when Sb1 => sb <= PW0out_reg0;
    pb <= "11111110";
    when Sb2 => sb <= PW1out_reg0;
    pb <= "11111101";
    when Sb3 => sb <= PW2out_reg0;
    pb <= "11111011";
    when Sb4 => sb <= PW3out_reg0;
    pb <= "11110111";
    when Sb5 => sb <= PW4out_reg0;
    pb <= "11101111";
    when Sb6 => sb <= PW5out_reg0;
    pb <= "11011111";
    when Sb7 => sb <= PW6out_reg0;
    pb <= "10111111";
    when Sb8 => sb <= PW7out_reg0;
```

- ▣ Using a statemachine, the message displayed on the LEDs rotates between stock messages and register data periodically

Methodology

D

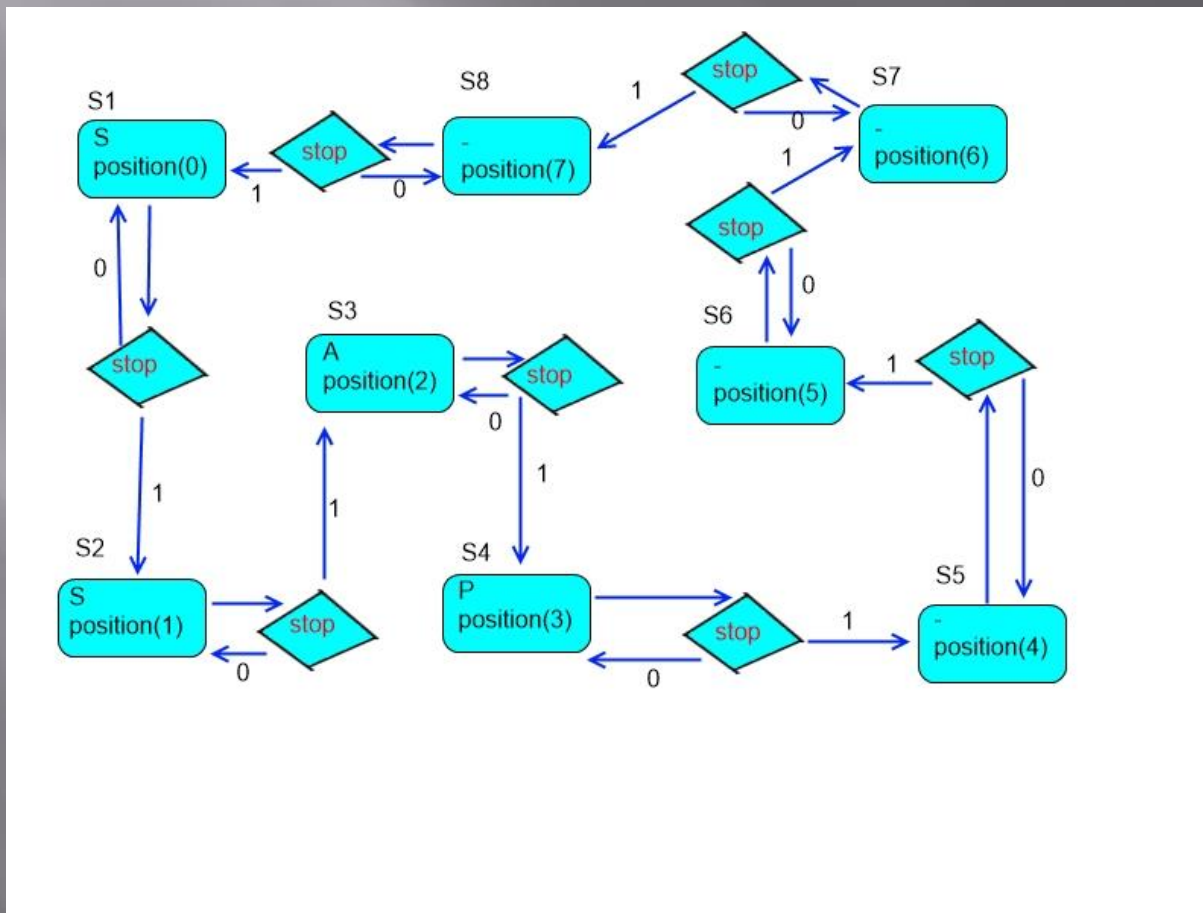
Below is the FSM diagram for the output to the 7-seg display.



Methodology

D

Below is the FSM diagram for the read/write switch output to the 7-seg display.



Methodology

B

The data received and added to the registers were taken from the serialized bitstream sent by the keyboard and processed through a signal decoder in order to create a specific character code, which was then processed through a MUX to determine which particular character is stored in the register. This process is enabled through the read/write switch being placed in the “write” position, else the keyboard input is discarded without being stored.

The data is then stored in 16-value registers, and these registers are set to enable if the read/write switch is set to the write position. Using a state machine with 18 states, each register is enabled one at a time and updated with the keyboard input. 8 of these registers are implemented and enabled depending on the address inputted by the switches.

Write Selector

Outputs: process (y)
begin

```

    case y is
        when S0 => Q <= "-----"; lights <= "000000000000000000";
        when S1 => Q <= "-----"; lights <= "000000000000000000";
        when S2 => Q <= "1111"; lights <= "000000000000000001";
        when S3 => Q <= "1110"; lights <= "0000000000000000011";
        when S4 => Q <= "1101"; lights <= "0000000000000000111";
        when S5 => Q <= "1100"; lights <= "000000000000001111";
        when S6 => Q <= "1011"; lights <= "00000000000011111";
        when S7 => Q <= "1010"; lights <= "0000000000111111";
        when S8 => Q <= "1001"; lights <= "0000000001111111";
        when S9 => Q <= "1000"; lights <= "0000000011111111";
        when S10 => Q <= "0111"; lights <= "0000000111111111";
        when S11 => Q <= "0110"; lights <= "0000001111111111";
        when S12 => Q <= "0101"; lights <= "0000011111111111";
        when S13 => Q <= "0100"; lights <= "0000111111111111";
        when S14 => Q <= "0011"; lights <= "0001111111111111";
        when S15 => Q <= "0010"; lights <= "0011111111111111";
        when S16 => Q <= "0001"; lights <= "0111111111111111";
        when S17 => Q <= "0000"; lights <= "1111111111111111";
    end case;
end process;

```

- In the Write selector, the signal for lit LEDs and value position are outputted based on state

Registers

N

```
\\Keyboard.xise - [regg.vhd]
Tools Window Layout Help
>> [Icons] [Run] [Stop] [Lightbulb]
41     elsif (clock_reg'event and clock_reg = '1') then
42         if en = '1' then
43             if y = '1' then
44                 if pos = "0000" then
45                     UN0out_reg0 <= UN0_in;
46                 elsif pos = "0001" then
47                     UN1out_reg0 <= UN1_in;
48                 elsif pos = "0010" then
49                     UN2out_reg0 <= UN2_in;
50                 elsif pos = "0011" then
51                     UN3out_reg0 <= UN3_in;
52                 elsif pos = "0100" then
53                     UN4out_reg0 <= UN4_in;
54                 elsif pos = "0101" then
55                     UN5out_reg0 <= UN5_in;
56                 elsif pos = "0110" then
57                     UN6out_reg0 <= UN6_in;
58                 elsif pos = "0111" then
59                     UN7out_reg0 <= UN7_in;
60                 elsif pos = "1000" then
61                     PW0out_reg0 <= PW0_in;
62                 elsif pos = "1001" then
63                     PW1out_reg0 <= PW1_in;
64                 elsif pos = "1010" then
65                     PW2out_reg0 <= PW2_in;
66                 elsif pos = "1011" then
67                     PW3out_reg0 <= PW3_in;
68                 elsif pos = "1100" then
69                     PW4out_reg0 <= PW4_in;
70                 elsif pos = "1101" then
71                     PW5out_reg0 <= PW5_in;
```

Each register holds 16 vectors of data

Positional vector determines which vector is updated

Top level (Control)

```
begin
--keyboard input and selecting which value to change in the register
slo: Slowmo port map (clock => clk, resetn => reset, slowclock => sloclk);
code: ps2keyboard port map (resetn => reset, clock => clk, ps2c => ps2c, ps2d => ps2d, dout => key, done => rxdone);
p: Parser port map (input => key, valid => valid, en => wr_rd, output => chara);
preg: Parsereg port map (clk => sloclk, resetn => reset, d => valid, q => validcheck);
valpos: Writeselector port map (clock => sloclk, resetn => reset, stop => validcheck, en => wr_rd, Q => pos, lights =>
sel: RegSelect port map (address => address, enab => enab);
```

This program works using a slowed clock that shifts every 0.2 seconds in a keyboard input program

The input is then given to a parser to get a character value

And the other components determine what register and what value will be edited