Objective:

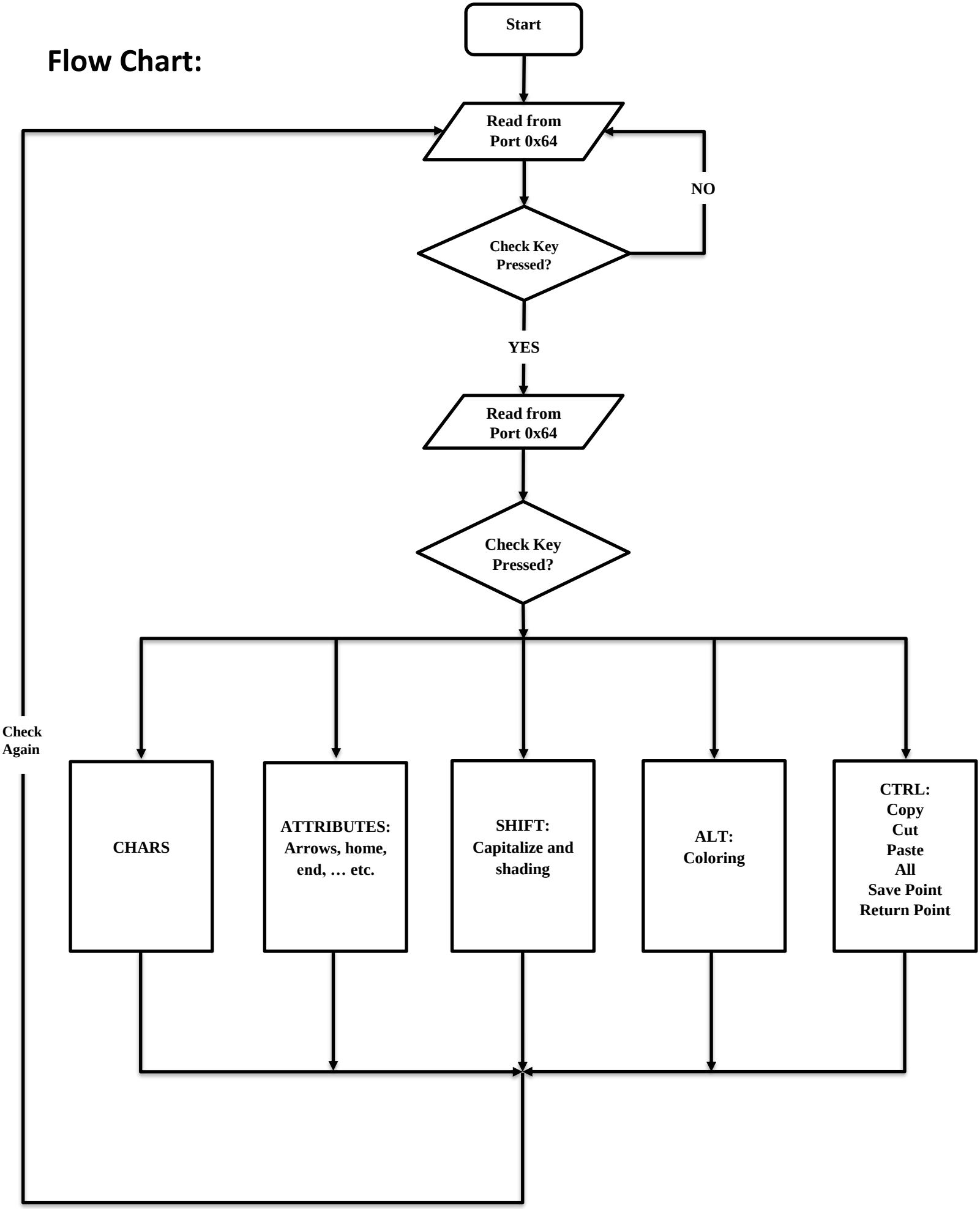
Develop a simplified multi-tab editor using x86 assembly language with the following features:

- Reading characters from keyboard and display them on the screen of the bare machine.
- Switching between capital and small letters using 'Shift' and 'Caps Lock' buttons.
- Switching between symbols and numbers while pressing 'Shift' button.
- Reaching start of line and end of it using **'Home'** and **'End'**.
- Using extended keyboard features using **'Num Lock'** button.
    - Arrows and Numbers
    - END, HOME, PAGE DOWN, PAGE UP and POINT.
- Removing characters using **'Delete'** and **'Backspace'** buttons.
- Moving cursor through text displayed on the screen using basic arrows and extended keys when **'Num Lock'** is deactivated.
- Jumping to new line using **'Enter'** button.
- Shifting text by 8 spaces when pressing **'Tab'**.
- Shading and un-shading text using **'Shift'** with arrows.
- When there is a shaded text we can replace, delete and un-shade when pressing any character, **'Delete'** or **'Backspace'** and **arrows**, respectively.
- Shade all text using **'Ctrl+A'**.
- Copy and cut Shaded text using **'Ctrl+C'** and **'Ctrl+X'** then pasting it in desired location throughout the page using **'Ctrl+V'**.
- Have a save-point using **'Ctrl+S'** in order to return to on a later time using **'Ctrl+Z'**.
- Colouring shaded text using **'Alt+FirstLetterofColors'**. i.e. (Red 'R', Blue 'B', Yellow 'Y', Green 'G', White 'W').
- Switching throughout multiple tabs (up to 8 tabs) using **'F'** keys and **'Page up'** or **'Page down'** while maintaining cursor in its last spot before switching.
- Transferring text between tabs using copy and paste features.

Description:

- The program is a bootable bare machine program:
    - Loading hard disk sectors into the main memory by interrupt (**0x13**).
    - Keyboard hardware drive:
        - Cursor is set according to the current position using interrupt (**0x10**).
        - Keyboard sends 1 in the **LSB** of port **0x64** to indicate if a key is pressed.
        - The make and break code of a key is read from port **0x60**.
        - Detect the key pressed or released from the code read above resulting in the following cases:
            - **Shift make:** enables writing capital letters and symbols.
            - **Shift break:** returns to small letters with numbers.
            - **CTRL make:** set ctrl status.
            - **CTRL break:** reset ctrl status.
            - **ALT make:** set alt status.
            - **ALT break:** reset alt status.
            - **CapsLock make:** toggle capsLock status.
            - **NumLock make:** toggle NumLock status.
            - **Backspace:** delate from left.
            - **Delete:** delete from right.
            - **Arrows:** navigate through text within allowed parts.
            - **Enter:** produce new line.
            - **Tab:** adds 8 spaces at cursor position.
            - **Home and end:** first and end of line.
            - **Normal characters:** read directly at cursor position.
            - **F1, F2, ... F8:** switches between tabs(pages).

Flow Chart:


Start
Read from Port 0x64
NO
Check Key Pressed?
YES
Read from Port 0x64
Check Key Pressed?
Check Again
CHARS
ATTRIBUTES: Arrows, home, end, … etc.
SHIFT: Capitalize and shading
ALT: Coloring
CTRL: Copy Cut Paste All Save Point Return Point

Algorithm:

Data allocated in memory:

- **ScanCodeTable**
- Shifted **ScanCodeTable**
- Page Number
- Page Address (of cursor)
- Page limit
- **MyMemory** to store copied text
- **Previous_Status** to store the save point
- **BoundedBy** (array of 2 numbers) to store start and end of shaded text
- length of shaded text
- Status, 1 byte defined as follows:

RSVD	RSVD	RSVD	ALT	CTRL	Shading	CapsLock	NumLock
			0x10	0x08	0x04	0x02	0x01

Psuedo Code:

```
label1:
     x = inport(0x64) ;
     if (!(x and 0x01))
        {
         goto label1
        }
     x = inport(0x60);
     switch(x){

     case(shift make):
     table = ShiftedScanCodeTable;
     break;

     case(shift break):
     table = scanCodeTable;
     break;

     case(CTRL make):
     STATUS =  STATUS or 0x08 ; (set ctrl)
     break;

     case(CTRL break):
     STATUS =  STATUS and 0xF7 ; (reset ctrl)
     break;

     case(ALT make):
     STATUS =  STATUS or 0x10 ; (set alt)
     break;

     case(ALT break):
     STATUS =  STATUS and 0xEF ; (reset alt)
     break;

     case(CAPSLOCK make):
     STATUS =  STATUS xor 0x02 ; (toggle)
     break;

     case(NUMLOCK make):
     STATUS =  STATUS xor 0x01 ; (toggle)
     break;

     case(Left arrow):
     if(Shift is pressed)
     {
     shade left letter to the cursor;
     cursor = cursor-1;
     }
     else
     {
     if(STATUS and 0x04) ; shading
     {
     remove shading
     }
     cursor = cursor-1;
     }
```

```
break;

case(Right arrow):
if(Shift is pressed)
{
shade right letter to the cursor;
cursor = cursor+1;
}
else
{
if(STATUS and 0x04) ; shading
{
remove shading
}
cursor = cursor+1;
}
Break;
case(Down arrow):
if(Shift is pressed)
{
shade 80 letters right to the cursor; may be going to the next line
cursor = cursor+80;
}
else
{
if(STATUS and 0x04) ; shading
{
remove shading
}
cursor = cursor+80;
}
Break;
case(Up arrow):
if(Shift is pressed)
{
shade 80 letters left to the cursor; may be going to the previous line
cursor = cursor-80;
}
else
{
if(STATUS and 0x04) ; shading
{
remove shading
}
cursor = cursor-80;
}
break;
case(Enter):
if(STATUS and 0x04) ; is there shading or not
{
delete Shaded;
}
;go to the next line
row = row+1;
column = 0;
break;
case(Enter):
```

```
if(STATUS and 0x04) ; is there shading or not
{
delete Shaded;
}
;go to the next line
row = row+1;
column = 0;
edi = (row*80+column)*2+ starting address of current page;
break;
case(Back Space):
if(STATUS and 0x04) ; is there shading or not
{
delete Shaded;
}
else
{
delete char left to the cursor;
for(i = end ; i > cursor ; i--){
    text[i]=text[i+1]; shifting
}
cursor -- ;
}
Break;
case(Delete):
if(STATUS and 0x04) ; is there shading or not
{
delete Shaded;
}
else
{
delete char right to the cursor;
for(i = end ; i > cursor ; i--){
    text[i]=text[i+1]; shifting
}
}
break;
case(Tab):
if(STATUS and 0x04) ; is there shading or not
{
delete Shaded;
}
for(i = 1;i<=8;i++){
 insert(" ");
}
Break;
case(charx):
if(STATUS and 0x04) ; is there shading or not
{
delete Shaded;
}
else if(STATUS and 0x10): // alt
{
    color(charx);
}
else if(STATUS and 0x08): //ctrl
{
```

```
        function(charx); // list of functions: copy,paste,cut,shade all,
save point and return to a save point
     }
     else{
         if(STATUS and 0x02)  //capslock
         {
             if ( 'a' < charx < 'z'){
                 charx = charx - 0x20 ;
             }
             else if('A' < charx < 'Z'){
                 charx = charx + 0x20 ;
             }
         }
      insert(charx); "text[cursor]= charx"
      char c = text[cursor+1];
      for(i = cursor+1 ; i < end ; i++){
         char c = text[i+1];
         text[i+1]= c ;
      }
     }
     break;

     case(F[i]):
     current page = pages[i];
     break;

     case(home):
     cursor = begin of the current line;
     break

     case(end):
     cursor = end of the current line;
     break

     case(page up):
     if(current page = 8){
         break;
     }
     else{
         current page ++ ;
     }
     break

     case(page down):
     if(current page = 1){
         break;
     }
     else{
         current page -- ;
     }
 Break;
    }
    goto label1;
```

Conclusion:

- **What did we learn?**
  1. How a simple keyboard sends characters and buttons through the port to be read in the main memory and displayed on a monitor.
  2. The correct paging concepts.
  3. Pages are stored Consequently in memory with **4KB** for each page.
  4. How keyboard features are represented in code. i.e. the "**how**" underlying the function.
  5. How hardware is connected and used by the system which is usually hidden from users for reliability and required levels of abstraction.
  6. Proper coding and organising can decrease debugging time and codes integration difficulty significantly.
  7. The use of interrupts for various functions (setting cursor, paging and expanding code segments).
  8. Thinking three steps ahead before. i.e. one hour of planning saves three of execution.
  9. We got acquainted with using assembly language, memory sectors and paging.
  10. A good team makes all the difference in achieving goals.

- **Problems we faced and how we managed to solve them:**
  1.
     - **Problem**: We had **'Times value is negative'** error because the main memory was set to **512 KB**.
     - **Solution**: We saved our program in the hard disk then used interrupt **0x13** to load from hard disk to main memory.
  2.
     - **Problem**: we couldn't implement proper way to undo multiple changes.
     - **Solution**: We created a save point using **'Ctrl+s'**, so as when we press **'Ctrl+z'** we retrieve the code at the save point.
     - **Note:** We could make it limited to only one change like in **Notepad**; each time we press **'Ctrl+z'** it switches between two states (current and previous) or we could save the whole page each time a change is made like in **MS Word**, but we decided to use our own method.
  3.
     - **Problem**: Finding pages addresses rather than page 1.
     - **Solution**: Using interrupt **'0x10'** while setting **AH=0x0A** and **BH=1** to write a sequence of letters in page 2, then we searched for this particular sequence in the entire main memory then printed its address in page 1.
       The address of the second page was found to be **0xB9000** then we replaced our current page address with the new address.

- **Note:** We concluded that each page is located 1000 hexadecimal digits from the previous one. i.e. **0xBA000**, **0xBB000**, … **0xBF000**.
  Every page is found to be **4KB** in size which corresponds to the difference in addresses. i.e. pages are in sequence in memory.
  The following code shows how we found the address:

```
mov ah, 0x02
mov dh, 0x00
mov dl, 0x00
mov bh, 1
int 0x10
mov ah,0x0A
mov al,'A'
mov bh,1
int 0x10
mov ah, 0x02
mov dh, 0x00
mov dl, 0x01
mov bh, 1
int 0x10
mov ah,0x0A
mov al,'M'
mov bh,1
int 0x10
mov ecx,0xB8000


again:

cmp byte[ecx],'A'
je one
add ecx,2
jmp again
one:
add ecx,2
cmp byte[ecx],'M'
je done
add ecx,2
jmp again
done:
mov ebp,ecx
sub ebp,
mov edi,ebp
mov ebx,table
mov cl,28
A:
mov eax,ebp
shr eax,cl ;
and eax,0x0F ;
xlat
mov [edi],al
add edi,2
sub cl,4
cmp cl,0
jnl A

table: db '0123456789ABCDEF',0
```

Link to Project:

https://github.com/AbduljabbarAhmed/multi-tab-editor