Theory:

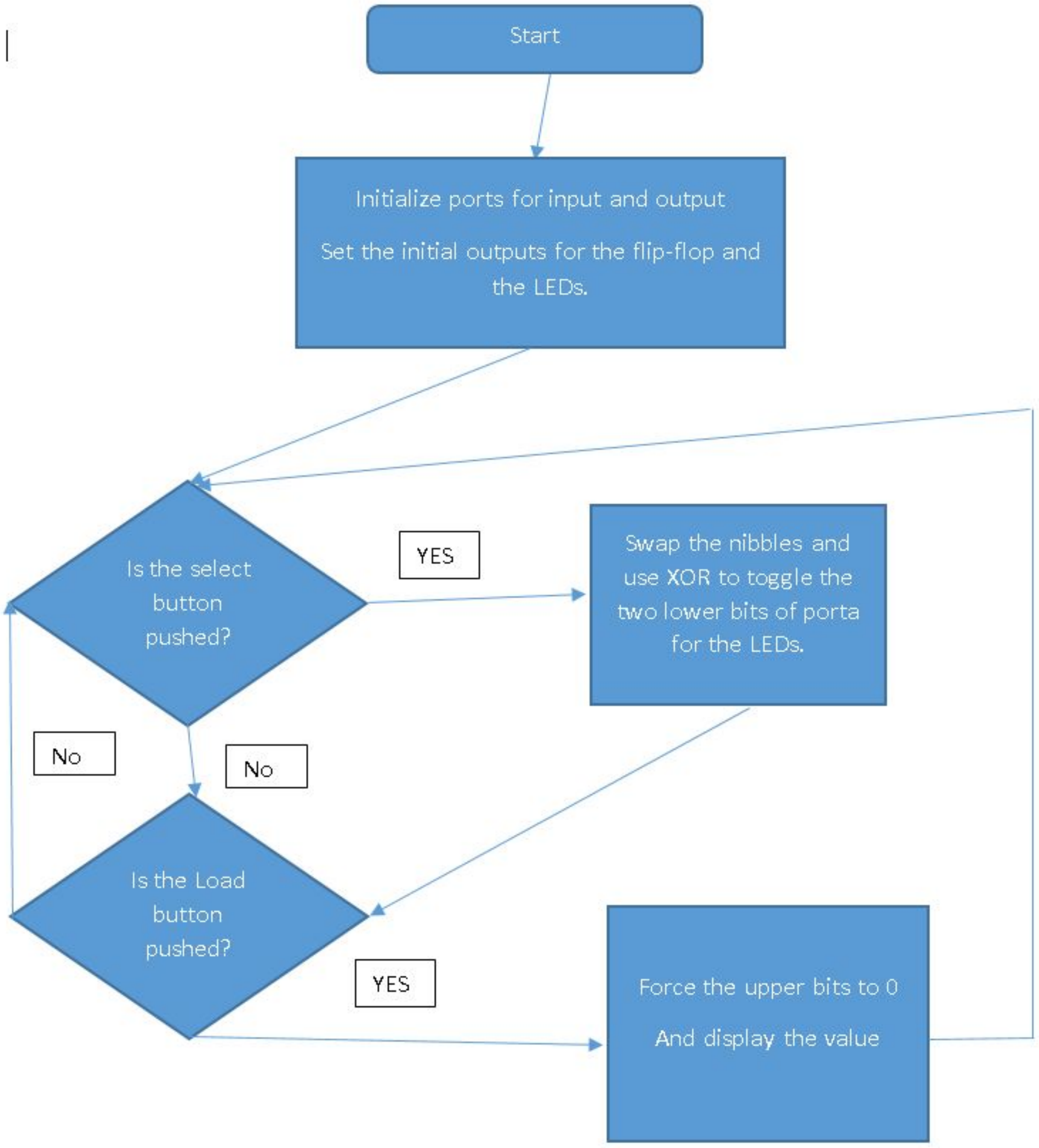
The hardware was almost kept same, except the changes made to the flipflop. The D flipflop from previous lab was rewired to work as Set reset(SR) flipflop. The Advantages of using an SR Flipflop will enable the circuit to retain the value of the pushbutton, So the Chip will not miss the press, while processing the code. The D and CLR of the D Flipflop is rewired as S and R of the SR Flipflop. The CLK of the flipflop is wired to the PBSW 3, thus every press or release of a button acts a clock signal. Unlike other switches in the circuit, pushbutton 3 is wired as active high. The flipflop is a positive edge triggered flipflop, and whenever the pushbutton is pressed value of S is passed on to Q. The Preset and D is connected to vcc. Since the preset is inverted, connecting it to a vcc 1 will be inverted to gnd 0. Thus the value of the flipflops will be 0, until its set. This means when clock is high, Q will output 1 and whenever the button is released the chip will active the CLR by outputting 1, since the CLR is inverted the input of 1 will be considered as 0 and will reset the flipflop.

S = D	R' = CLR'	CLK	Q	Q'
1	0	1	0	1
1	1	1	1	0
1	1	0	previous Q	previous Q'

A pull down resistor is placed between the pushbutton 3 and d flipflop. The pull down resistor will make the circuit output 0 when the pushbutton is not pressed, and 1 when it is.

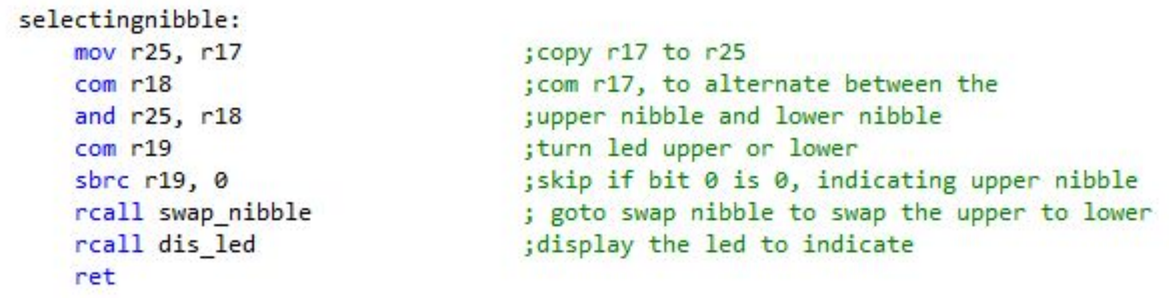
Two leds was added to the circuit as an indicator for the DIP switches. The Led will light up when corresponding DIP value is selected. Unlike the previous lab this led will not be Pulse Width modulated.

In the first task, was creating a program called nibble load. The program basically uses one register to store the value of the output obtained from portd and another is used to store the value $0F and force the upper nibbles to a logic 0 using the and function between the two registers. First the main loop of the program waits for a load button press. When a load button press is detected it will copy all the bits from portd, which contains the DIP switches, then force the upper nibble to 0 and then finally display the value that is inputted by the lower nibble onto the seven segment display. The load button is connected to PC0. Debouncing is not required for this because, all this program does is update the current value of the nibble into the seven segment display. Once a logic 0 is detected then it will update the value, even if it is noise it will not affect the overall purpose of the program, it will just read the switch values many times within a second.


Start
Initialize ports for input and output
Set the initial outputs for the flip-flop and the LEDs.
Is the select button pushed?
YES
Swap the nibbles and use XOR to toggle the two lower bits of porta for the LEDs.
No
No
Is the Load button pushed?
YES
Force the upper bits to 0
And display the value

The above flow chart describes the basic structure of the code. Checking for inputs, do something to indicate a swap in the nibble selection when select is pushed and go on to wait for another button push. If load is pressed then it breaks out of the button checking and swaps the nibbles based on the some value the select portion of the code has done and displays that value. It may be missing some things, but the basic structure is there. In the actual code all of these processes and more are addressed.

In the second program, two pushbuttons are utilized. PBSW1 connected to PC0 will be used as a select button and PBSW2 connected to PC6 will be used as a load button. Whenever the switch button is pressed it will alternate between the upper and lower nibble switches.



```
selectingnibble:
    mov r25, r17                    ;copy r17 to r25
    com r18                         ;com r17, to alternate between the
    and r25, r18                    ;upper nibble and lower nibble
    com r19                         ;turn led upper or lower
    sbrc r19, 0                     ;skip if bit 0 is 0, indicating upper nibble
    rcall swap_nibble               ; goto swap nibble to swap the upper to lower
    rcall dis_led                   ;display the led to indicate
    ret
```

In order to perform this function with ease, and reuse this code for bonus program. The selection of the nibble will done using a subroutine. The register 18 is initialized with $F0 and r19 is initialized with 1. With every press of the select pushbutton, the selecting nibble subroutine will be called. r18 and r19 will be complemented every time the subroutine is called, thus it will alternate between $0F and $F0, and 0 and 1 respectively. Like code one, and function is utilized to erase upper or lower nibble accordingly. R19 is complemented to predict when to swap, whenever the upper nibble is selected it will be swapped to lower nibble for later use. R19 is also used to light up led corresponding to the nibbles. In this particular program 1 in bit 0 of r19 will indicate the upper nibble and vice versa. With press of load button the selected nibble's value will be used to light up the 7segment display.

In the third program, the a service request flip flop is implemented as to catch the request not allow for the code to miss the button press. First, the ports are initialized. Ports D and C are set to be inputs by setting the ddrd and ddrc to $00. PortB is set to be output for the 7 segment display and the first two bits and the second to last bit of porta are set to be outputs and the rest are inputs, for the two LEDs, and the CLR output for the D flipflop, so the ddra was set to $43. Then in the main loop of the program, the select bit is repeatedly checked along with the load pushbutton. The program first checks for whether or not the select bit is set by the d flipflop. If the bit is set, then there will be a 10 ms delay for the debouncing and PC7 will be checked for whether or not there was an actual button press, if so then the next instruction will be skipped and the code will proceed to indicate the selection using r18 to toggle the first two bits in porta. If not then the program will branch into clear which basically sends a logic '0' pulse to clear the d flip flop. First the value of porta is taken using the in function and placed into r18, then the eor function is used such that the first two bits in r18 are toggled, using $03 as the other value that was in the xor operation. Then the new r18 is put back into porta. This works considering the first bit in port a was first set to be 1 initially, since when the program first starts, it starts with the lower nibble. After this operation is done the program will output a negative pulse to clear the d flipflop indicating the completion of the request. Then the program will wait for a load button press, if there is no press then it loops back to the main loop and starts checking for select push and then for a load push. If the select is pushed then the first bit of porta is read and if it is set then it will not swap the nibbles of the value in r17 obtained from portd of the dip switches, but if it is cleared then the nibbles will be swapped and the upper nibble will be forced to a 0 logic as so that only the lower register value will be counted. In order to check if the value is above 9, then an add operation was done with a register containing the value 6 and the register containing the lower nibble value, r17. If it is above 9 then the half carry flag will be set it will branch into overflow, where r17 will be assigned the value of 9. This way values above the possible display amount are accounted for and will display a value of 9, which is the highest value. Then the value is displayed in the 7 segment display.