ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

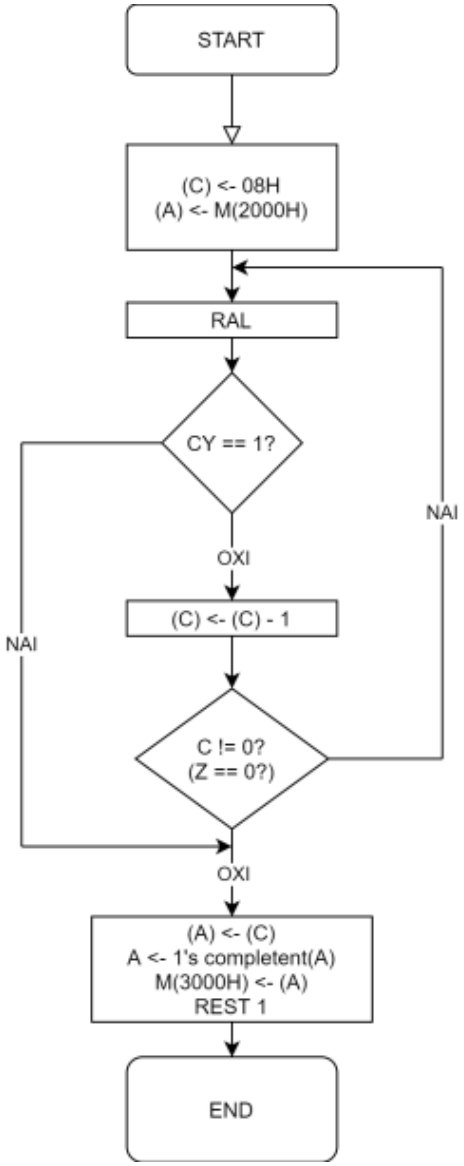
Συστήματα Μικροϋπολογιστών
1η Ομάδα Ασκήσεων

Αναστάσιος Λαγός - el13531
Κωνσταντίνος Βασιλάκης - el16504

Ασκηση 1

Ο κώδικας είναι ο παρακάτω

```
0800    0E - opcode         MVI C, 08H
0801    08 - data
0802    3A - opcode         LDA 2000H
0803    00 - low address
0804    20 - high address
0805    17 - opcode         RAL
0806    DA - opcode         JC 080DH
0807    0D - low address
0808    08 - high address
0809    0D - opcode         DCR C
080A    C2 - opcode         JNZ 0805H
080B    05 - low address
080C    08 - high address
080D    79 - opcode         MOV A, C
080E    2F - opcode         CMA
080F    32 - opcode         STA 3000H
0810    00 - low address
0811    30 - high address
0812    CF - opcode         RST 1
```



Η λειτουγία του παραπάνω προγράμματος είναι η εξής. Το πρόγραμμα διαβάζει από την διεύθυνση 2000H τα swiches στον Accumulator και κάνει shift left μέχρι να βρεί το πρώτο bit που έχει την τιμή ένα (μέχρι 8 φορές). Μόλις το βρει βγαίνει από το εσωτερικό loop και ανάβει τα LED που δείχνουν σε binary τον αριθμό της θέσης του πρώτου bit που είναι 1. (Με την σύμβαση αναμμένο - '1' , σβηστό - '0').

Το πρόγραμμα μπορεί γραφτεί έτσι ώστε να τρέχει συνεχόμενα βάζοντας στο τέλος του προγράμματος ένα jmp πάλι στην αρχή του.

Ασκηση 2

Ο κώδικας είναι ο παρακάτω

```
    IN 10H
    MVI E,FEH       ;Light up LSB of LED
    LXI H,3000H     ;Address of LED
    LXI B,01F4H     ;500 * 1ms

CHECK:
    LDA 2000H       ;Address of switch
    MOV D,A
    RRC             ;CY is modified according to bit D0
    JC CHECK        ;If LSB of switch is set then loop till not set
    MOV A,D         ;Get address again without having to access memory
    RLC             ;CY is modified according to bit D7
    JC RIGHT        ;If MSB of switch is set then right routine
                    ;else continue to left routine

LEFT:
    MOV A,E         ;E acts as temp storage for when accum
                    ;is checking conditions
    MOV M,A         ;light up LED
    RLC             ;If we use RLC we cant initiate A from FFH.
                    ;Need 1 zero.
    MOV E,A         ;Store temp value to E so i can check condition.
    CALL DELB
    JMP CHECK

RIGHT:              ;Pretty much the same as LEFT routine.
    MOV A,E
    MOV M,A
    RRC
    MOV E,A
    CALL DELB
    JMP CHECK

END
```

Ασκηση 3

Ο κώδικας είναι ο παρακάτω

```
    LXI B,03E8H         ; 40ms delay
    MVI L,00H           ; L is used for keeping the
                        ; state of the lights in case
                        ; input is bigger than 100
START:
    LDA 2000H           ; get the switch input
    CPI C8H             ; check if input is bigger than 199
                        ; and jump to the correct tag
    JNC ABOVE_199
    CPI 64H             ; check if input is bigger than 99
                        ; and jump to the correct tag
```

```
    JNC ABOVE_99
    MVI D,FFH           ; else the number is 99 or less so
                        ; calculcate the two digits
DECA:
    INR D               ; D will contain the number of number
                        ; of dozens
    SUI 0AH
    JNC DECA
    ADI 0AH
    MOV E,A             ; Store in E the number of number of units
    MOV A,D             ; Store in A the number of dozes
    STC
    CMC                 ; These two commands set the carry flag
                        ; to 0 in order to use left shift
    RAL
    RAL
    RAL
    RAL                 ; Shift A left four times to move dozens
                        ; to the 4 MSB
    ADD E               ; Add to A the units to the 4 LSB

    JMP UPDATE_LEDS
ABOVE_99:               ; When the number if bigger then 99
                        ; and less than 200
    MOV A,L             ; Get the state of lights
    CPI 00H
    JZ ABOVE_99_1
    MVI L,00H           ; If the state was switched on update
                        ; state to on off
    MVI A,00H           ; and switch off the leds
    JMP UPDATE_LEDS
ABOVE_99_1:
    MVI L,01H           ; If the state was switched off update
                        ; state to on
    MVI A,0FH         ; and switch on the 4 LSB leds
    JMP UPDATE_LEDS
ABOVE_199:              ; When the number if bigger then 199
    MOV A,L             ; Get the state of lights
    CPI 00H
    JZ ABOVE_199_1
    MVI L,00H           ; If the state was switched on update
                          ; state to on off
    MVI A,00H           ; and switch off the leds
    JMP UPDATE_LEDS
ABOVE_199_1:
    MVI L,01H           ; If the state was switched off update
                        ; state to on
    MVI A,F0H           ; and switch on the 4 MSB leds
UPDATE_LEDS:
    CMA                 ; Find the complement of A
                        ; so leds light on 1
    STA 3000H           ; Send the result to the output lights
    CALL DELB
    JMP START
END
```

Ασκηση 4

$$Καμπυλη\ διακριτων\ στοιχειων = \frac{20000 + 20x}{x}$$

$$Καμπυλη\ FPGA = \frac{10000 + 40x}{x}$$

$$Καμπυλη\ SoC_1 = \frac{100000 + 4x}{x}$$

$$Καμπυλη\ SoC_2 = \frac{200000 + 2x}{x}$$

Εξισωνουμε για να βρουμε τομες

IC-FPGA: x = 500
IC-SoC-1: x = 5000
IC-SoC-2: x = 10000
FPGA-SoC-1: x = 2500
FPGA-SoC-2: x = 5000
SoC-1-SoC-2: x= 50000

Από τα παραπάνω βρίσκουμε ότι οι πιο συμφέρουσε τεχνολογίες είναι οι ακόλουθες

$$FPGA,\ \ x \leq 500$$

$$Διακριτα\ Στοιχεια,\ \ 500 \leq x \leq 5000$$

$$SoC_1,\ \ 5000 \leq x \leq 50000$$

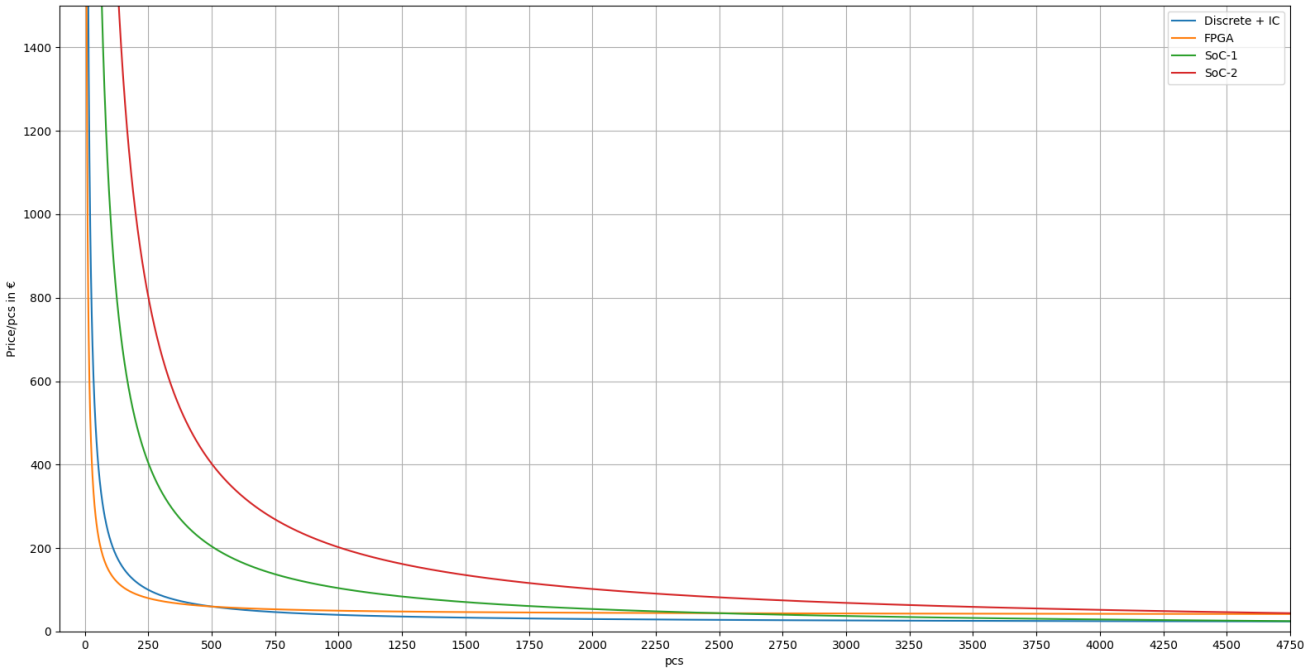
$$SoC_2,\ \ 50000 \leq x$$

Γενικά οι τεχνολογίες που χρειαζονται μεγάλη αρχική επένδυση συμφέρουν για μεγαλύτερο αριθμό τεμαχίων.

Έχουμε

$$20000 + 20x > 10000 + (a + 10)x \Rightarrow a < 10 + \frac{10000}{x}$$

Βλέπουμε ότι για πολύ μεγάλα x το το κόστος του FPGA παραμένει μικρότερο αν το κόστος της πλακέτας γίνει μικρότερο από 10 ευρώ.



Ασκηση 5

i. Έχουμε τον παρακάτω κώδικα

```
module gateModelCircuit (F, A, B, C , D, E);
    output [1: 4] F;
    input A, B, C, D, E;
    wire Anot, Bnot, Cnot, Dnot, w1F1, w2F1, w3F1, w4F1,
            w1F2, w2F2, w3F2, w4F2, w5F2,
            w1F3, w2F3, w3F3, w4F3, w5F3, w6F3, w7F3,
            w1F4, w2F4, w3F4, w4F4;
```

```
8
9       not
10          G1(Anot, A),
11          G2(Bnot, B),
12          G3(Cnot, C),
13          G4(Dnot, D);
14
15      and
16          G5(w1F1, B, C),
17          G6(w3F1, w2F1, A),
18          G7(w4F1, Bnot, Cnot, D),
19          G8(w1F2, Anot, Bnot, Dnot),
20          G9(w2F2, Anot, C, D),
21          G10(w3F2, B, Cnot, D),
22          G11(w4F2, A, Bnot, D),
23          G12(w5F2, A, C, Dnot),
24          G13(w1F3, A, B, C),
25          G14(w2F3, B, C),
26          G15(w4F3, D, E),
27          G16(w6F3, w3F3, w4F3),
28          G17(w7F3, w5F3, D),
29          G18(w1F4, C, D),
30          G19(w3F4, w2F4, A),
31          G20(w4F4, B, C, D, E);
32
33      or
34          G21(w2F1, w1F1, D),
35          G22(F[1], w3F1, w4F1),
36          G23(F[2], w1F2, w2F2, w3F2, w4F2, w5F2),
37          G24(w5F3, A, w2F3),
38          G25(w3F3, B, C),
39          G26(F[3], w1F3, w7F3, w6F3),
40          G27(w2F4, w1F4, B, E),
41          G28(F[4], w3F4, w4F4);
42  endmodule
```

ii.

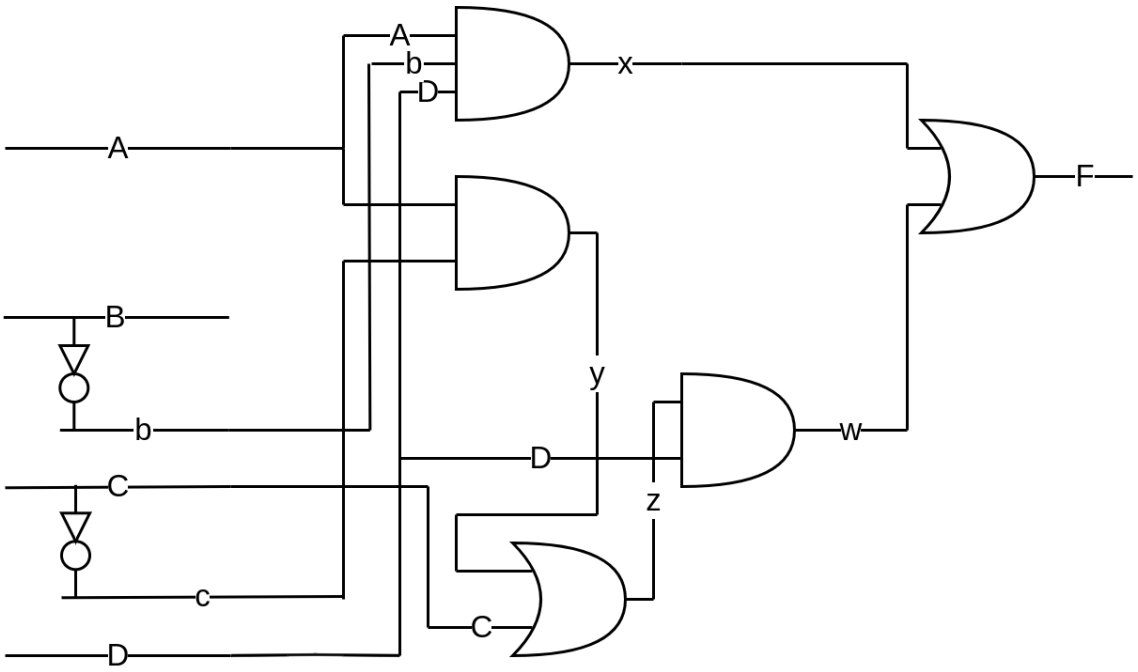
```
1   module dataflowCircuit (F A, B, C, D);
2       output [1:4] F;
3       input A, B, C, D;
4       assign F[1] = A & ((B & C) | D) + (~B & ~C & D),
5              F[2] = (~A & ~B & ~D) | (~A & C & D) |
6                     (B & ~C & D) | (A & ~B & D) | (A & C & ~D),
7              F[3] = (A & B & C) | ((A | (B & C)) & D) |
8                        ((B | C) & (D & E)),
```

```
9               F[4] = A & (B | (C & D) | E) | (B & C & D & E);
10 endmodule
```

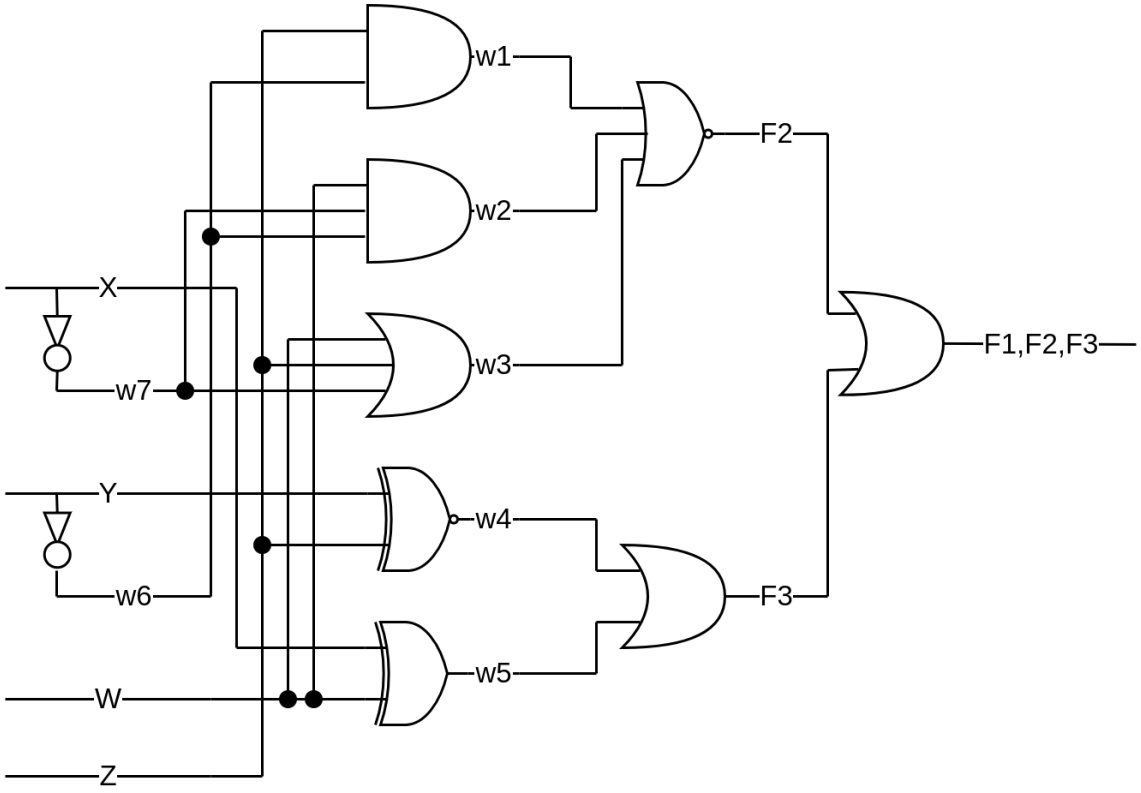
Ασκηση 6

i. Έχουμε τα παρακάτω λογικά διαγράμματα

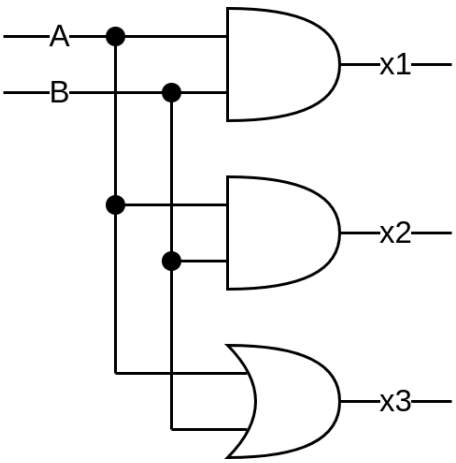
a.



b.



c.



ii. Μοντελοποιούμε τον αθροιστή χρησιμοποιώντας 4 αθροιστές του ενός bit σαν δομικές μονάδες. Ο κάθε αθροιστής αποτελείται από 2 ημιαθροιστές και μία πύλη OR

Αθροιστής - 1 bit

```
module full_adder(x, y, c_in, sum, c_out);
  input x, y, c_in;
  output sum, c_out;
  wire w1,w2,w3;


  xor G1(w1, x, y);
  xor G2(sum, w1, c_in);
  and G3(w2, w1, c_in);
  and G4(w3, x, y);
  or G5(c_out, w2, w3);
endmodule
```

Στην συνέχεια χρησιμοποιούμε την παραπάνω δομική μονάδα για να φτιάξουμε ένα αθροιστή κρατουμένου 4 bits.

Αθροιστής Κρατουμένου - 4 bit

```
module rippe_carry_adder(x, y, sum, c_out);

 input[3:0] x, y;
 output[3:0] sum;
 output c_out; //Output carry
 wire w1, w2, w3; //Wires to the next adder

 fulladder fa1(x[0], y[0], 1'b0, sum[0], w1);
 fulladder fa2(x[1], y[1], w1, S[1], w2);
 fulladder fa3(x[2], y[2], w2, S[2], w3);
 fulladder fa4(x[3], y[3], w3, S[3], c_out);
endmodule
```

iii.

Αθροιστής - Αφαιρέτης - 4 bit

```
module addSub(A, B, selection, Result);
input sel;
  input [3:0] A,B;
  output [3:0] Result;
  wire [3:0] Result;

  assign Result = (selection)? A + B : A - B;
endmodule
```

Ασκηση 7

i.

```
module mealy(y, x, clock, reset);
    output y;
    input x, clock, reset;
    reg [1: 0] state;
    reg output_reg;
    parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;

    always @ ( posedge clock, negedge reset)
    //Initialization at state A if reset == 0
    if (reset == 0) state <= a;
    else
        case (state)
            a:
            begin
                if (x == 0) begin
                    state <= d;
                    output_reg <= 1'b1;
                end else begin
                    state <= a;
                    output_reg <= 1'b0;
                end
            end
            b:
            begin
                if (x == 0) begin
                    state <= c;
                    output_reg <= 1'b1;
                end else begin
```

```
                    state <= a;
                    output_reg <= 1'b0;
                end
            end
            c:
            begin
                if (x == 0) begin
                    state <= d;
                    output_reg <= 1'b1;
                end else begin
                    state <= b;
                    output_reg <= 1'b0;
                end
            end
            d:
            begin
                if (x == 0) begin
                    state <= c;
                    output_reg <= 1'b0;
                end else begin
                    state <= d;
                    output_reg <= 1'b1;
                end
            end
        endcase

assign y = output_reg;

endmodule
```

ii.

```
module moore(y, x, clock, reset);
    output y;
    input x, clock, reset;
    reg [1: 0] state, output_reg;
    parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;

    always @ ( posedge clock, negedge reset)
    //Initialization at state A if reset == 0
    if (reset == 0) state <= a;
    else
        case (state)
            a:
            begin
```

```
                if (x == 0)
                    state <= d;
                else
                    state <= a;
                output_reg <= 1'b0;
            end
            b:
            begin
                if (x == 0)
                    state <= c;
                else
                    state <= a;
                output_reg <= 1'b1;
            end
            c:
            begin
                if (x == 0)
                    state <= b;
                else
                    state <= d;
                output_reg <= 1'b1;
            end
            d:
            begin
                if (x == 0)
                    state <= c;
                else
                    state <= d;
                output_reg <= 1'b0;
            end
        endcase

assign y = output_reg;

endmodule
```