Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2020-2021

ΑΘΗΝΑ 26 - 3 - 2021

1η ΟΜΑΔΑ ΑΣΚΗΣΕΩΝ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Συστήματα Μικροϋπολογιστών"
(παράδοση μέχρι 14 Απριλίου 2021)

Εργασία Φοιτητών:

Αριάδνη Καζδάγλη el18838, Βικέντιος Βιτάλης el18803

Άσκηση 1:

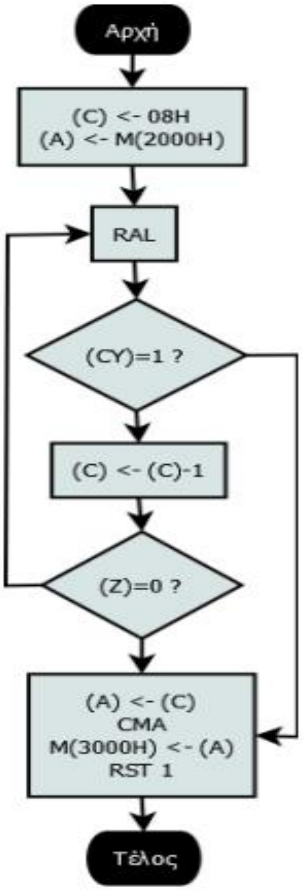
1,3)Το πρόγραμμα που δίνεται διαβάζει τον αριθμό που αντιστοιχεί στο MSB των διακοπτών εισόδου σε δεκαδική μορφή και τον εμφανίζει στις λυχνίες εξόδου σε δυαδική μορφή. Το αρχικό πρόγραμμα της εκφώνησης είναι:

```
MVI C,08H
LDA 2000H

LABEL3:
       RAL
       JC LABEL2
       DCR C
       JNZ LABEL3

LABEL2:
       MOV A,C

LABEL1:
       CMA
       STA 3000H
       RST 1

END
```

Για να επαναλαμβάνεται χωρίς τέλος η λειτουργία του , θα πρέπει να βάλουμε μια ετικέτα START στην αρχή του προγράμματος και μια εντολή JMP START στο τέλος του. Στη συνέχεια μπορούμε να πατήσουμε RUN όταν θα θέλουμε το πρόγραμμα να συνεχίσει τη λειτουργία του για τη νέα είσοδο. Παρακάτω παρουσιάζεται ο επισυναπτόμενος κώδικας:

```
;Αυτό το πρόγραμμα διαβάζει ποιός dip switch είναι ON(απο δεξία προς τα αριστερά, δεκαδικό σύστημα)
;και κάνει flash το bit εξόδου του ίδιου αριθμού, αλλά στη δυαδική του μορφή

START:
       MVI C,08H      ;C<--08H
       LDA 2000H      ;Διάβασμα εισόδου από τα dip switches

LABEL1:
       RAL            ;Ολίσθηση των bits του A αριστερά κατά 1
       JC LABEL1      ;Αν το κρατούμενο είναι 1 jump σε LABEL1
       DCR C          ;C--
       JNZ LABEL2     ;Αν όχι 0 jump LABEL2

LABEL2:
       MOV A,C
       CMA            ;A-->A Συμπληρωματικό
                      ;λόγω της συμπληρωματικής λογικής εξόδου
                      ;Το bit εξόδου μετατρέπεται από
                      ;το hardware του.
       STA 3000H      ;Τύπωσε τον καταχωρητή A
       JMP START

END
```

2) Διάγραμμα ροής προγράμματος:



Άσκηση 2:

```
        IN 10H          ;remove memory protection
        LXI B,01F4H     ;01F4H = 500
        MVI A,01H       ;lightByte is 01H
        CMA             ;A complement(LED's are negative logic)
        STA 3000H       ;light the LSB of the output byte
        CMA             ;A complement
        MOV D,A         ;now D has the lightByte
INPUT:
        LDA 2000H       ;read input
        ANI 01H         ;mask the LSB
        JZ INPUT        ;if LSB 0 -->INPUT
        CALL DELB       ;delay 0.5s
        LDA 2000H       ;read input
        ANI 80H         ;mask the MSB
        JNZ RIGHT       ;if MSB 1-->RIGHT

LEFT:
        MOV A,D         ;return the lightByte in A
        RLC             ;rotate bits of lightByte left by one
        CMA             ;A complement
        STA 3000H       ;light the next bit of the output
        CMA             ;A complement
        MOV D,A         ;save the lightByte in D
        JMP INPUT       ;repeat

RIGHT:
        MOV A,D         ;return the lightByte in A
        RRC             ;rotate bits of lightByte right by one
        CMA             ;A complement
        STA 3000H       ;light next bit of the output
        CMA             ;A complement
        MOV D,A         ;save the lightByte in D
        JMP INPUT       ;repeat
END
```

Άσκηση 3:

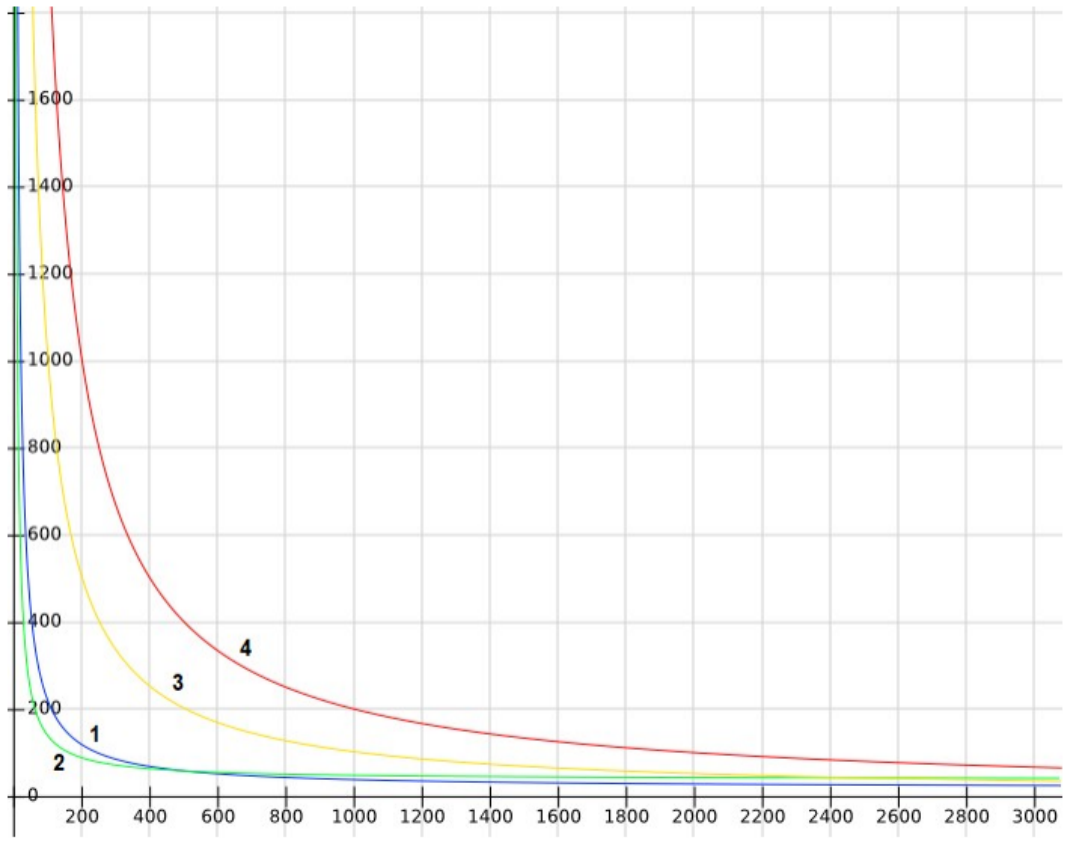
```
;Example:
;67 57 47 37 27 17 7 -3 7
;   0  1  2  3  4  5  6

       LXI B,01F4H
       MVI H,0FH
       MVI L,F0H
START:
       CALL DELB
       LDA 2000H
       CPI 64H
       JNC GREATER_EQUAL_100
       MVI D,FFH
DECA:
       INR D ;Counting dekades
       SUI 0AH ;Substract 10
       JNC DECA ; CY = 1 -> Negative number, continue, otherwise repeat.
       ADI 0AH ; Monades ston A register
       MOV E,A
       MOV A,D
       RLC  ; Left shift 4 times
       RLC
       RLC
       RLC
       ADD E
       CMA  ;Complement A
       STA 3000H ;Show the result on switches
       JMP START
GREATER_EQUAL_100:
       CPI C8H
       JNC GREATER_EQUAL_200
LOOPM:
       MOV A,H
       ANI 0FH ;Masking A, holding the 4 LSB digits for blink, setting 4 MSB's to zero.
       CMA    ;Complement A
       STA 3000H ;Show the result on switches
       CALL DELB ; Delay
       MOV H,A
       JMP START ; Repeat for next number

GREATER_EQUAL_200:
       MOV A,H
       ANI F0H ;Masking A, holding the 4 MSB digits for blink, setting 4 LSB's to zero.
       CMA    ;Complement A
       STA 3000H ;Show the result on switches
       CALL DELB ; Delay
       MOV H,A
       JMP START ; Repeat for next number

END
```

Άσκηση 4:

Οι συναρτήσεις κόστους ανά τεμάχιο για κάθε τεχνολογία φαίνονται στον παρακάτω πίνακα:

1η	2η	3η	4η
$\frac{20000+20x}{x}$	$\frac{10000+40x}{x}$	$\frac{100000+4x}{x}$	$\frac{200000+2x}{x}$

ενώ οι αντίστοιχες γραφικές παραστάσεις φαίνονται στο επόμενο διάγραμμα



Εξισώνοντας τις εκφράσεις των καμπυλών ανά 2, βρίσκουμε τα σημεία τομής των καμπυλών μεταξύ τους

1η-2η	2η-3η	1η-3η	2η-4η	1η-4η	3η-4η
$x=500$	$x=2500$	$x=5000$	$x=5000$	$x=10000$	$x=50000$

Μελετώντας τον προηγούμενο πίνακα και το διάγραμμα των γραφικών παραστάσεων, εξάγουμε τα διαστήματα τιμών του αριθμού τεμαχίων που ελαχιστοποιούν το κόστος κατασκευής για κάθε τεχνολογία

$$0<x<500 \quad : \textbf{2η}$$
$$500<x<5000 \quad : \textbf{1η}$$
$$5000<x<50000 \quad : \textbf{3η}$$
$$x>50000 \quad : \textbf{4η}$$

Παρατηρούμε ότι οι τεχνολογίες με υψηλό κόστος σχεδίασης γίνονται συμφέρουσες μόνο σε υψηλούς αριθμούς τεμαχίων. Αν z το κόστος ανά IC για την τεχνολογία των FPGAs, τότε το συνολικό κόστος κατασκευής για τη 2η τεχνολογία θα είναι μικρότερο αυτού της 1ης όταν

$$\frac{10000+(z+10)x}{x}<\frac{20000+20x}{x} \quad \Rightarrow \quad z<\frac{10000}{x}+10$$

Για z≤10 το συνολικό κόστος κατασκευής της 2ης τεχνολογίας είναι πάντα μικρότερο από αυτό της 1ης, άρα για να αποκλειστεί η 1η τεχνολογία πρέπει το κόστος ανά IC της 2ης να είναι το πολύ 10€ ανά τεμάχιο.

Άσκηση 5:

- F1 = A(BC + D) + B'C'D

Περιγραφή σε επίπεδο πυλών

```
module circuit_F1(F1,A,B,C,D)
        output F1;
        input A, B, C, D;
        wire b, c, w1, w2, w3, w4;

        not G1 (b, B),
            G3  (c, C);

        and G2 (w1, B, C),
            G5 (w4, b, c, D),
            G6 (w4, A, w2);

        or G3 (w2, w1, D),
           G7 (F1, w4, w3);
endmodule
```

Μοντελοποίηση ροής δεδομένων

```
module circuit_F1(F1, A, B, C, D)
        output F1;
        input A, B, C, D;
        assign F1 = (A & ((B & C) | D) | (~B & ~C &D));
endmodule
```

- F2(A, B, C, D) = (0, 2, 3, 5, 7, 9, 10, 11, 13, 14)

Περιγραφή σε επίπεδο πυλών

```
primitive F2 (F2, A, B, C, D)
        output F2;
        input A, B, C, D;
        table // Pinakas alitheias
```

```
// A   B   C   D:  F2  //header
   0   0   0   0:  1;
   0   0   0   1:  0;
   0   0   1   0:  1;
   0   0   1   1:  1;
   0   1   0   0:  0;
   0   1   0   1:  1;
   0   1   1   0:  0;
   0   1   1   1:  1;
   1   0   0   0:  0;
   1   0   0   1:  1;
   1   0   1   0:  1;
   1   0   1   1:  1;
   1   1   0   0:  0;
   1   1   0   1:  1;
   1   1   1   0:  1;
   endtable
endprimitive
```

Μοντελοποίηση ροής δεδομένων

```
primitive F2(F2, A, B, C, D)
    output F2;
    input A, B, C, D;
    assign F2 = (~A & ~B & ~C & ~D) | (~A & ~B & C & ~D) | (~A & ~B & C & D) | (~A & B & ~C & D) | (~A & B & C & D) | (A & ~B & ~C & D) | (A & ~B & C & ~D) | (A & ~B & C & D) | (A & B & ~C & D) | (A & B & C & ~D);
endprimitive
```

- F3 = ABC + (A + BC)D + (B+C)DE

Περιγραφή σε επίπεδο πυλών

```
module circuit_F3 (F3, A, B, C, D, E)
    output F3;
    input A, B, C, D, E;
    wire w1, w2, w3, w4, w5, w6;
```

```
        and G1 (w1, B, C),
            G4 (w4, A, B, C),
            G5 (w5, w3, D),
            G6 (w6, w2, D, E);

        or  G2 (w2, B, C),
            G3 (w3, A, w1),
            G7 (F3, w4, w5, w6);
endmodule
```

Μοντελοποίηση ροής δεδομένων

```
module circuit_F3 (F3, A, B, C, D, E)
        output F3;
        input A, B, C, D, E;
        assign F3 = (A & B & C) | ((A | (B & C)) & D) | ((B | C) & D & E);
endmodule
```

- F4 = A(B + CD + E) + BCDE

Περιγραφή σε επίπεδο πυλών

```
module circuit_F4 (F4, A, B, C, D, E)
        output F4;
        input A, B, C, D, E;
        wire w1, w2, w3, w4;

        and G1 (w1, C, D),
            G3 (w3, B, C, D, E),
            G4 (w4, A, w2);

        or G2 (w2, B, w1, E),
           G5 (F4, w4, w3);
endmodule
```

Μοντελοποίηση ροής δεδομένων

```
module circuit_F4 (F4, A, B, C, D, E)
```

output F4;

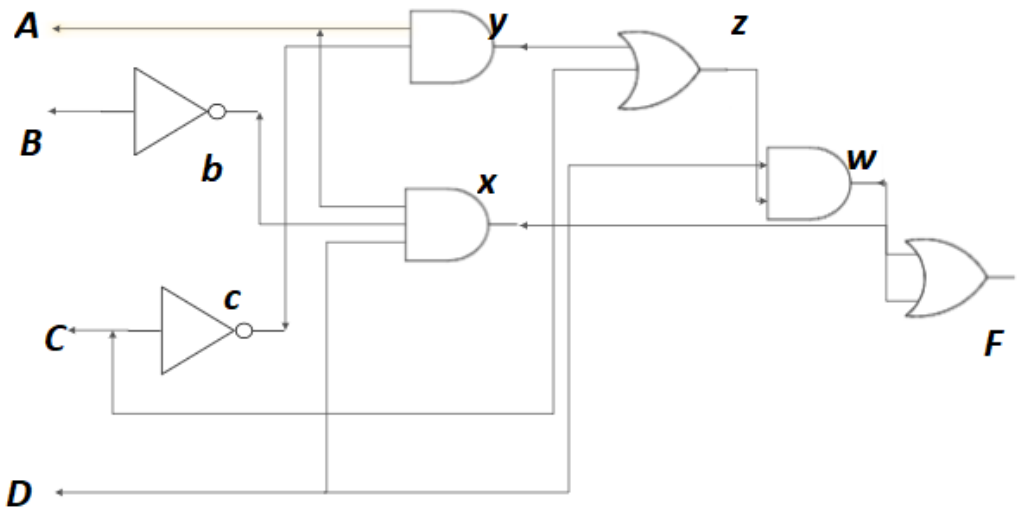
input A, B, C, D, E;

assign F4 = (A & (B | (C & D) | E)) | (B & C & D & E);

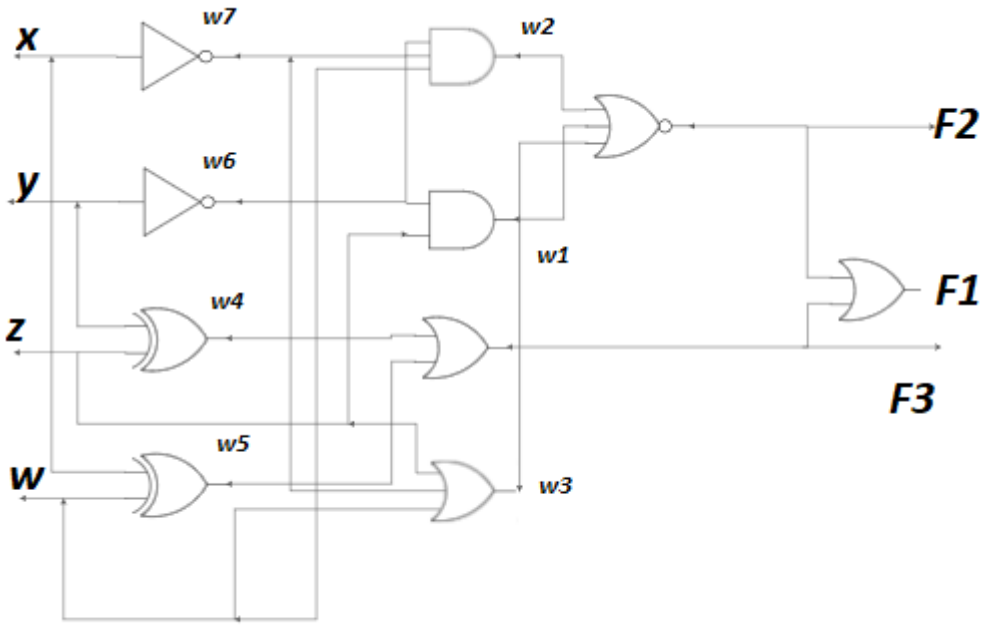
endmodule

Άσκηση 6:

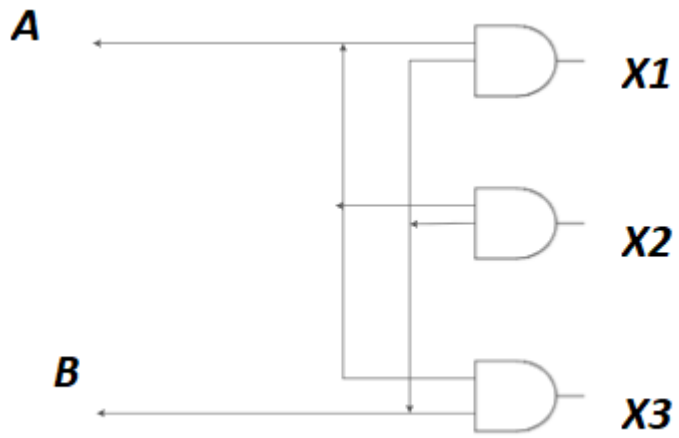
1α)



1β)



1γ)



2)Περιγραφή του ζητούμενου αθροιστή-αφαιρέτη σε επίπεδο πυλών:

```
module half_adder(output S, C, input x, y);
        xor(S, x, y);
        and(C, x, y);
endmodule
module full_adder(output S, C, input x, y, z);
wire S1, C1, C2;
half_adder
                HA1(S1, C1, x, y),
                HA2(S, C2, S1, z);
        or G1(C, C2, C1);
endmodule
module 4_bit_adder_subtractor(output [3:0] Sum, output C4, input [3:0] A, B, input C0);
wire C1, C2, C3;
wire [3:0] M;
xor
        (M[0], B[0], C0),
```

```
        (M[1], B[1], C0),
        (M[2], B[2], C0),
        (M[3], B[3], C0);
full_adder
        FA0(Sum[0], C1, A[0], M[0], C0),
        FA1(Sum[1], C2, A[1], M[1], C1),
        FA2(Sum[2], C3, A[2], M[2], C2),
        FA3(Sum[3], C4, A[3], M[3], C3);
endmodule
```

3)Ο ίδιος αθροιστής-αφαιρέτης με περιγραφή ροής δεδομένων:

```
module 4_bit_adder_subtractor(output [3: 0] Sum, output C4, input [3: 0] A, B, input C0);
        assign {C4, Sum} = C0 ? (A+(~B)+1) : (A+B);
endmodule
```

Άσκηση 7:

1) Μηχανή Mealy

```
module Mealy_Machine (x, y, clock, reset)
        output reg y;
        input x, clock, reset;

        reg [1 : 0] state, next_state;
        parameter a = 2'b00,
                  b = 2'b01,
c = 2'b10,
                  d = 2'b11;
        always @ (posedge clock, negedge reset)
                if(reset == 0)state <= a;
                else state state <= next_state;
        always @ (state, x)
                case (state)
                        a: if(x) next_state = a; else next_state = d;
                        b: if(x) next_state = a; else next_state = c;
                        c: if(x) next_state = b; else mext_state = d;
```

```
                d: if(x) next_state = d; else next_state = c;
        endcase

    always @ (state,x)
        case (state)
                a, b, c: y = ~x;
                d: y = x;
        endcase
endmodule
```

2)Μηχανή Moore

```
module Moore_Machine (x, y, clock, reset)
    output y;
    input x, clock, reset;

    reg [1 : 0] state;
    parameter a = 2'b00,
              b = 2'b01,
              c = 2'b10,
              d = 2'b11;
    always @ (posedge clock, negedge reset)
if(reset == 0) state <=a;
        else case (state)
                a: if(x) state <= a; else state <= d;
                b: if(x) state <= a; else state <= c;
                c: if(x) state <= d; else state <= b;
                d: if(x) state <= d; else state <=c;
        endcase
    always @ (state)
        case(state)
                a, d: y = 1'b0;
                b, c: y = 1'b1;
        endcase
endmodule
```