Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

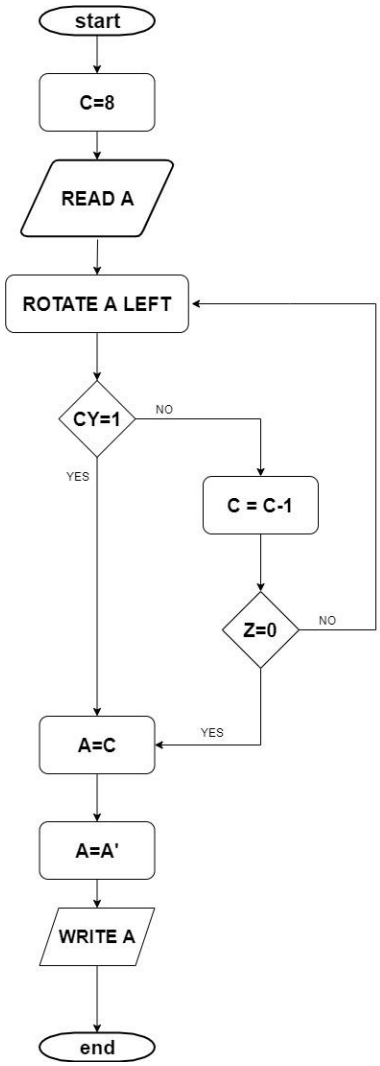
Ακ. έτος 2020-2021, 6ο Εξάμηνο: Συστήματα Μικροϋπολογιστών

Παναγιώτα-Μικαέλα Ξυλια **AM:03118859**

Νικολέτα Σάλιαρη **AM:03118864**

1η ΟΜΑΔΑ ΑΣΚΗΣΕΩΝ

Άσκηση 1

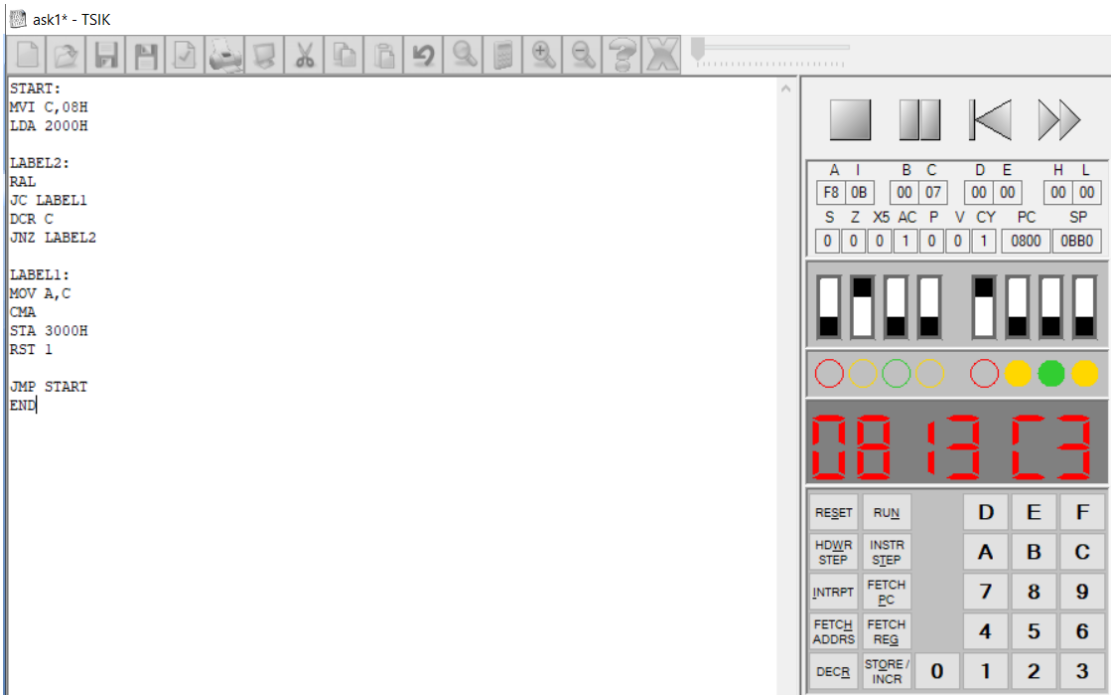


Με την χρήση του πίνακα 2 του παραρτήματος 2 που μας δίνεται, κάνουμε αποκωδικοποίηση του προγράμματος σε γλώσσα μηχανής που μας δίνεται, το οποίο το παραθέτουμε ξεχωριστά.

Αριστερά φαίνεται το διάγραμμα ροής του προγράμματος το οποίο ακολουθεί πιστά την γλώσσα assembly που αποκωδικοποιήσαμε.

Κάνοντας χρήση του προσομοιωτή, επαληθεύουμε την λειτουργία του προβλήματος. Συγκεκριμένα, δίνουμε ως είσοδο έναν αριθμό 8-bit και μας επιστρέφεται η υψηλότερη θέση η οποία έχει ψηφίο ίσο με 1. Για παράδειγμα, με είσοδο **01001000**, παίρνουμε έξοδο **00000111** (7 δυαδικό), δηλαδή ότι το υψηλότερο ψηφίο που είναι ίσο με το 1 είναι το $7^{ο}$.

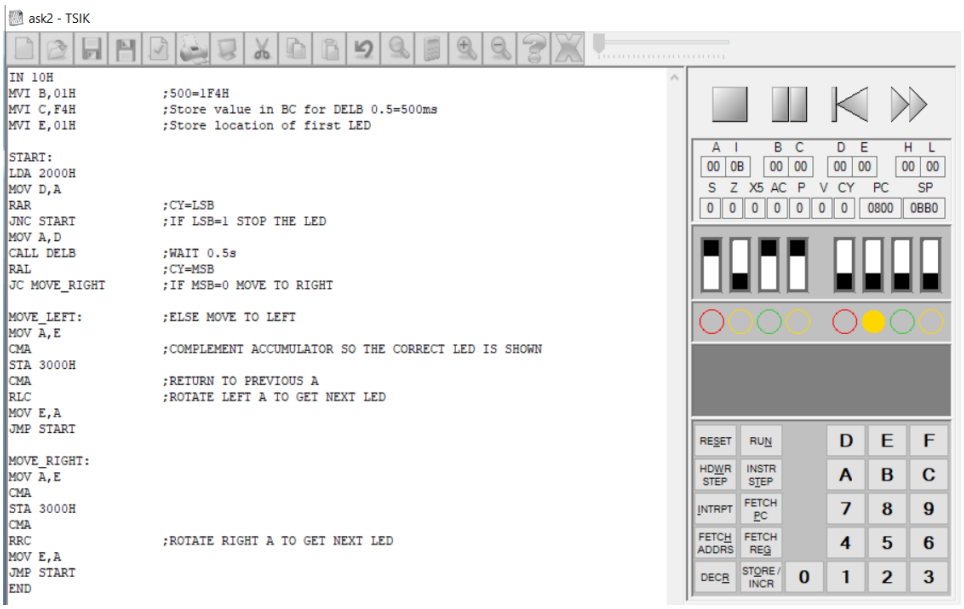
Για να συνεχίζεται το πρόγραμμα χωρίς να τερματίζεται, αρκεί να βάλουμε μία εντολή JMP στο τέλος η οποία θα οδηγεί στην αρχή του προγράμματος. Η εντολή RST 1 αποσκοπεί στην παύση της λειτουργίας, ώστε να μπορούμε να δούμε το αποτέλεσμα και να εισάγουμε νέα είσοδο.



Άσκηση 2

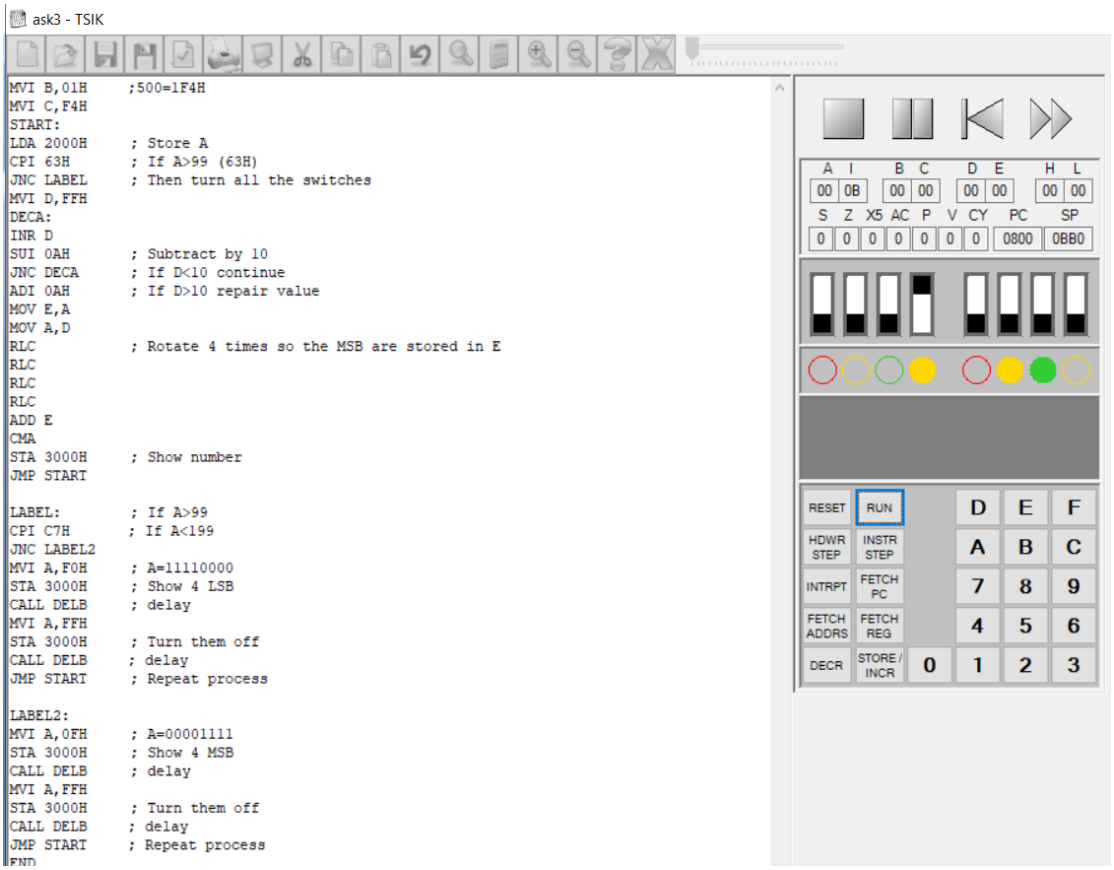
Για την σωστή λειτουργία του προγράμματος μας χρειάζεται ο διαχωρισμός του σε τρία τμήματα (εκτός του αρχικών εντολών), την START, την MOVE_RIGHT και την MOVE_LEFT. Στην START εξετάζουμε αν το LSB των dipswitch είναι OFF, ώστε να σταματήσει πρόωρα την διαδικασία. Αλλιώς, ελέγχει αν το MSB είναι ON ή OFF, ώστε να κινηθεί αναλόγως δεξιά ή αριστερά.

Ακόμα χρησιμοποιούμε την εντολή **CALL DELB** για να καθυστερήσουμε την αλλαγή κατάστασης των LED κατά 0.5s. Για να ορίσουμε τον χρόνο που θέλουμε ως καθυστέρηση στην αρχή του προγράμματος φορτώνουμε στους καταχωρητές τις τιμές B=01H και C=F4H, εφόσον 1F4H=$500_{(10)}$.



Άσκηση 3

Αρχικά, αν ο αριθμός μας είναι μεγαλύτερος του 99, τότε πηγαίνει στην ειδική περίπτωση όπου αναβοσβήνουν τα LED. Έπειτα, ελέγχουμε ξανά αν είναι μεγαλύτερος του 199, ώστε αναλόγως να αναβοσβήνουν τα 4 MSB ή τα 4 LSB. Αν ο αριθμός μας είναι μικρότερος του 99, για να δούμε πόσες δεκάδες έχει, αφαιρούμε το 10 μέχρι να έχουμε κρατούμενο. Φορτώνουμε τον A στον E, ώστε να κρατήσουμε τα υπόλοιπα ψηφία. Έπειτα, μεταφέρουμε τον καταχωρητή D που μέτραγε τις δεκάδες στον A και περιστρέφουμε προς τα αριστερά, ώστε να αποθηκευτούν στα MSB. Τέλος, προσθέτουμε τον E και έχουμε με αυτόν τον τρόπο χωρίσει τις δεκάδες στα 4 πρώτα bit και τις μονάδες στα 4 τελευταία.



Άσκηση 4

Έστω Q ο αριθμός τεμαχίων για κάθε τύπο. Επομένως, οι εξισώσεις για κάθε τύπο γίνονται:

1. IC: 20.000+20Q
2. FPGA: 10.000+40Q
3. SoC-1: 100.000+4Q
4. SoC-2: 200.000+2Q

Τα σημεία τομής που μας ενδιαφέρουν:

IC-FPGA=> 20000+20Q=10000+40Q=> Q=500

IC-SoC-1=>20000+20Q=100000+4Q=> Q=5000

SoC-1-SoC-2=>100000+4Q=200000+2Q=> Q=50000

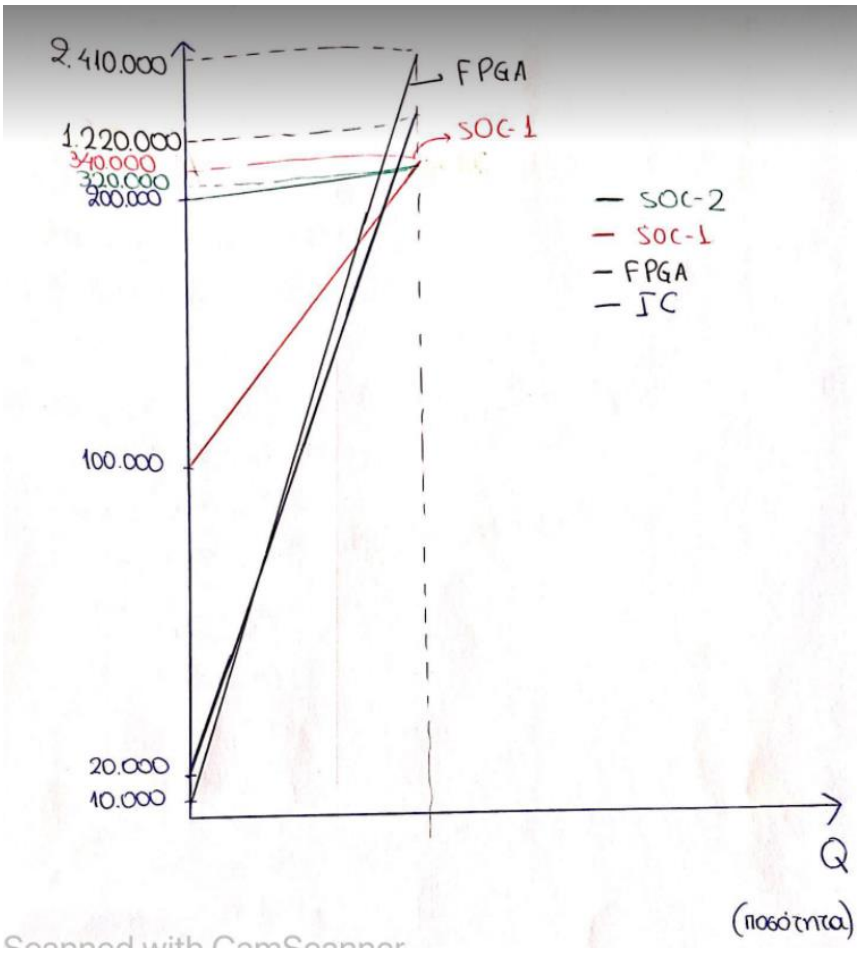
Άρα έχουμε τις εξής συμφέρουσες περιοχές για κάθε τεχνολογία:

ποσότητα Q	1-500	500-5000	5000-50000	50000+
συμφέρουσα	FPGA	IC	SoC-1	SoC-2

Για να εξαλείψουμε την τεχνολογία των IC και να έχουμε 3 συμφέρουσες τεχνολογίες πρέπει το σημείο τομής της ευθείας κόστους FPGA με την ευθεία κόστους SoC-1 να είναι το λιγότερο 5000 τεμάχια. Έτσι, για 5000 τεμάχιο το συνολικό μεταβλητό κόστος ανα τεμάχιο FPGA θα είναι

5000K+10000=40000+100000=>5K=130=> K=26

Θεωρώντας ότι το κόστος πλακέτας ανά τεμάχιο και συναρμολόγησης δεν άλλαξε και έμεινε 10€, τότε το κόστος ανά τεμάχιο των I.C. στην τεχνολογία των FPGAs είναι 16€.



Άσκηση 5

(i)F1=A(BC +D) + B'C'D ,
F2(A, B, C, D)=Σ (0, 2, 3, 5, 7, 9, 10, 11, 13, 14),
F3=ABC + (A + BC)D+(B+ C)DE ,
F4=A(B + CD + E) + BCDE .

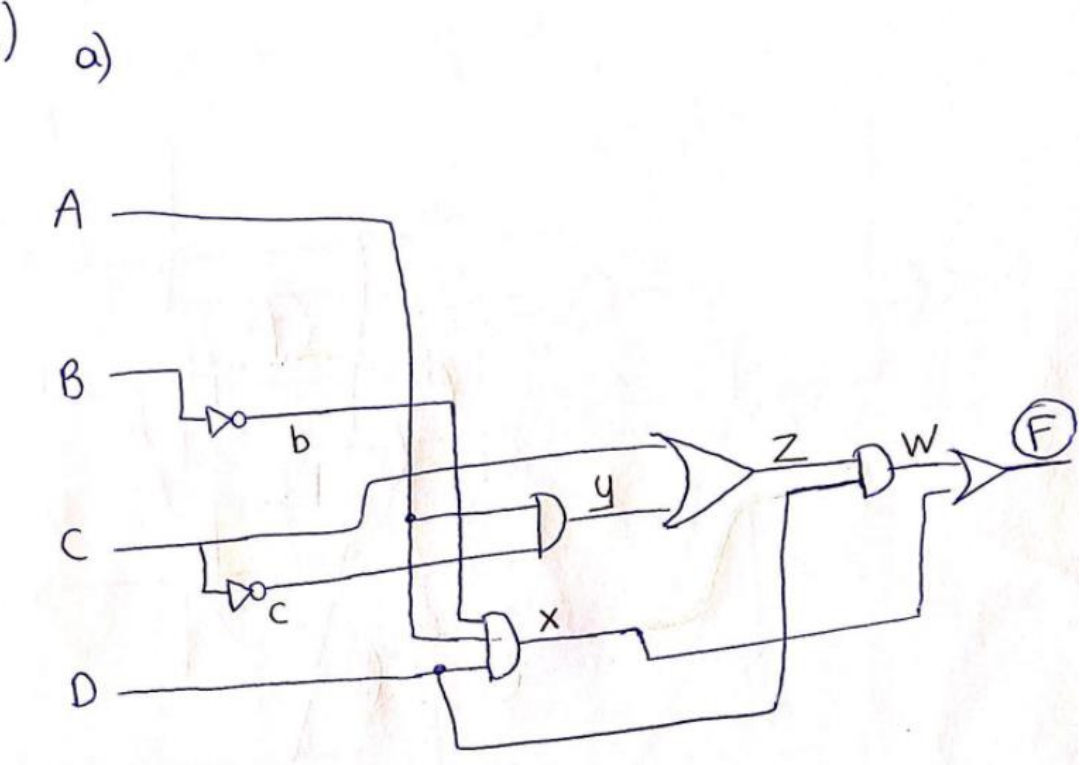
```
module_ex5(A, B, C, D, E, F1, F2, F3, F4);
        output F1, F2, F3, F4;
        input A, B, C, D, E;
        wire Anot, Bnot, Cnot, Dnot, w1, w2, w3, w4;
                not
                        (Anot, A);
                        (Bnot, B);
                        (Cnot, C);
                        (Dnot, D);
                and (w1, B,C);
                or (w2, w1, D);
                and (w4, D, Bnot, Cnot);
                or ( F1, w4, w3);

        wire m0, m2, m3, m5, m7, m9, m10, m11, m13, m14;
                and
                        (m0, Anot, Bnot, Cnot, Dnot);
                        (m2, Anot, Bnot, C, Dnot);
                        (m3, Anot, Bnot, C, D);
                        (m5, Anot, B, Cnot, D);
                        (m7, Anot, B, C, D);
                        (m9, A, Bnot, Cnot, D);
                        (m10, A, Bnot, C, Dnot);
                        (m11, A, Bnot, C, D);
                        (m13, A, B, Cnot, D);
                        (m14, A, B, C, Dnot);
                or( F2, m0, m2, m3, m5, m7, m9, m10, m11, m13, m14);

        wire w5, w6, w7, w8, w9;
                and ( w5, A, B, C);
                or ( w6, A, w1);
                and ( w7, w6, D);
                or ( w8, B, C);
                and ( w9, w8, D,E);
                or (F3, w5, w7, w9);

        wire w10, w11, w12;
                and ( w10, C, D);
                or ( w11, w10, B, E);
                and ( w12, w11, A);
                and ( w13, B, C, D, E);
                or ( F4, w12, w13);

        endmodule
```

(ii) Μοντελοποίηση ροής δεδομένων:

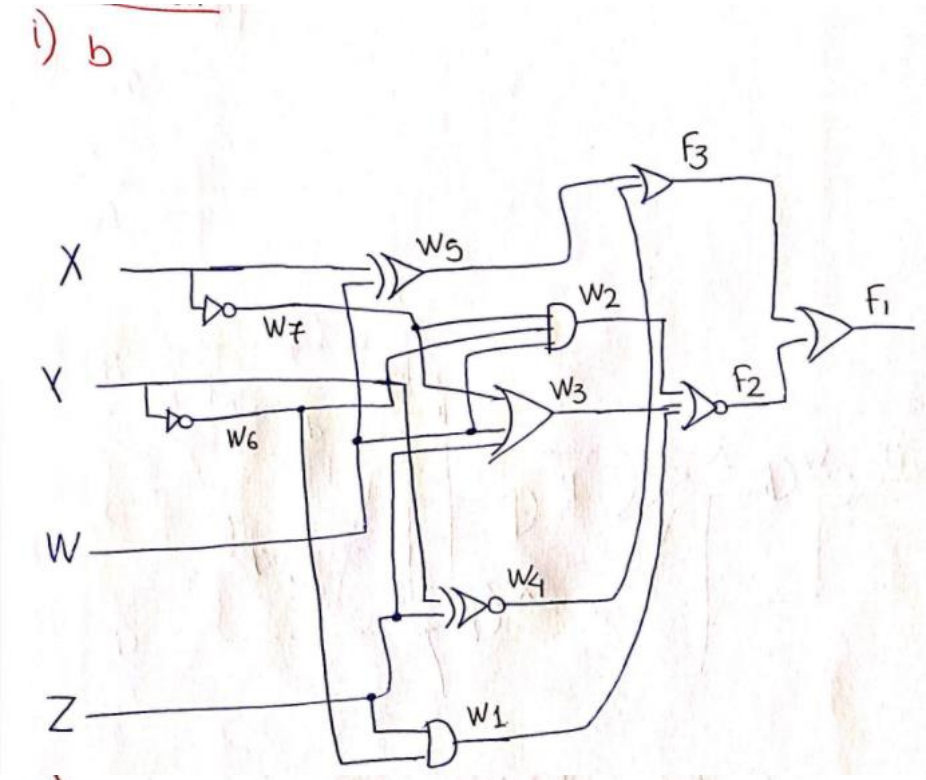
```
module_ex5(A, B, C, D, E, F1, F2, F3, F4);
        output F1, F2, F3, F4;
        input A, B, C, D, E;

        assign
                F1 = (A((B&C)|D)|(~B&~C&D),
                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),
                F3=(A&B&C)|((A|(B&C))&D)|((B|C)&(D&E)),
                F4=(A&(B|((C&D)|E))|(B&C&D&E);

endmodule
```

Άσκηση 6

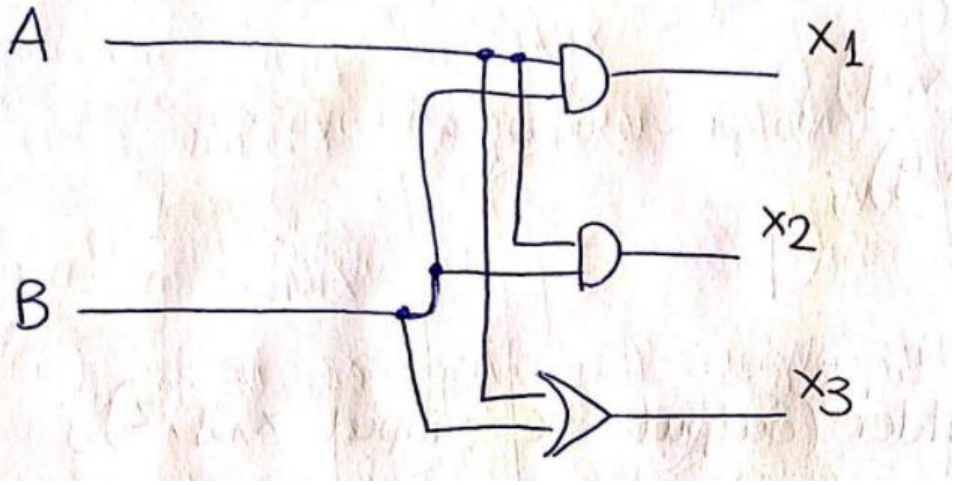
i) a)



i) b



i) c.



ii) Περιγραφή αθροιστή-αφαιρετή 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 HA( S1, C1, x, y);
                        HA( S2, C2, S1, z);
        or ( C, C2, C1);
endmodule

module 4_bit_adder( 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], C[0]);
                (M[1], B[1], C[1]);
                (M[2], B[2], C[2]);
                (M[3], B[3], C[3]);

endmodule


full_adder
        FA0 (M[0], B[0], C[0]);
        FA1 (M[1], B[1], C[1]);
        FA2 (M[2], B[2], C[2]);
        FA3 (M[3], B[3], C[3]);
endmodule
```

iii) Αθροιστής-αφαιρέτης 4 bit με περιγραφή ροής δεδομένων:

```
module 4_bit_adder( output [3:0] sum, output C4, input [3:0] A, B, input C0);
        assign {C3, sum} =C0 ? (A+(~B)+1) : (A+B);
endmodule
```

Άσκηση 7

i) Η απλοποίηση της εξόδου y:

AB\X	0	1
00	1	0
01	0	1
11	1	0
10	1	0

a-> 00
d->01
c->10
b->11

Άρα y=A’B’x’+ A’Bx + ABx’ + AB’x

```
module Mealy ( y, x, clock, reset);
        output y;
        input x, clock, reset;
        reg [1:0] state, next_state;
        parameter a=2'b00, d=2'b01, c=2'b10, b=2'b11;
        always@(posedge clock, nogedge reset)
            if(reset==0) state<=a;
            else case(state)
                a:if(~x) state<=d; else state<=a;
                d:if(~x) state<=c; else state<=d;
                c:if(~x) state<=d; else state<=b;
                b:if(~x) state<=c; else state<=a;
                endcase
        always@(state, x)
                case(state)
                a, b, c : y = ~x;
                    d : y = x;
                endcase
endmodule
```

ii)

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