Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Εθνικό Μετσόβιο Πολυτεχνείο

Συστήματα Μικροϋπολογιστών

Πρώτη ομάδα ασκήσεων

Σπουδαστές
Κατσάμπουλα Χριστίνα Σοφία (Α.Μ.: 03114910)
Παπασκαρλάτος Αλέξανδρος (Α.Μ.: 03111097)

Ημερομηνία Παράδοσης Αναφοράς: 3 Απριλίου 2018

Άσκηση 1

Μετά από χρήση του πίνακα 2 του παραρτήματος 2 καταλήξαμε ότι οι παραπάνω δοθέντες εντολές αντιστοιχούν στον εξής κώδικα

0800	06	MVI B, 01H
0801	01	
0802	3A	LDA 2000H
0803	00	
0804	20	
0805	FE	CPI 00H
0806	00	
0807	CA	JZ 0813
0808	13	
0809	08	
080A	1F	RAR
080B	DA	JC 0812
080C	12	
080D	08	
080E	04	INR B
080F	C2	JNZ 080A
0810	0A	
0811	08	
0812	78	MOV A,B
0813	2F	CMA
0814	32	STA 3000H
0815	00	
0816	30	
0817	CF	RST 1

Δηλαδή το πρόγραμμα σε μορφή assembly είναι το ακόλουθο :

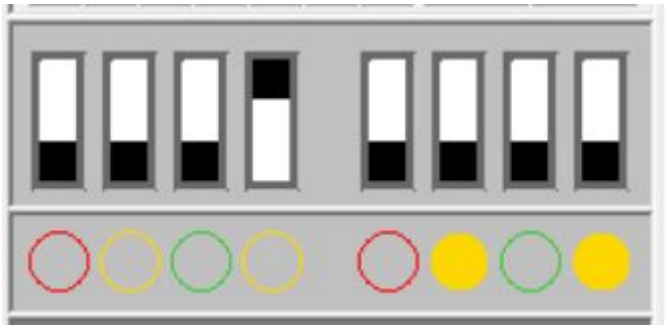
```
START:
MVI  B,01H    ; (B) <- 01H
LDA  2000H    ; (A) <- M(2000H)
CPI  00H      ; Συγκρίνω (A) με 00H
JZ  L3        ; Εάν z=1, δηλαδή (A) == 00H, τότε κάνω άλμα στην L3

L1:
RAR           ; Δεξιά ολίσθηση (μέσω σημαίας κρατουμένου CY)
JC   L2       ; Αν CY = 1, τότε κάνω άλμα στην L2
INR B         ; Αυξάνω το (B) κατά 1
JNZ L1        ; Αν z=0, τότε κάνω άλμα στην L1

L2:
MOV A,B       ; (A) <- (B)
```

```
L3:
CMA         ; Συμπληρώνω τα bits του (A)
STA 3000H   ; M(3000H) <- (A)
RST 1       ; Διακόπτω τον κώδικα και επιστρέφω στο 0800H
END
```

Ο κώδικας πραγματοποιεί την εξής λειτουργία:

Χρησιμοποιούμε τον καταχωρητή B για να αποθηκεύσουμε έναν “δείκτη” που δείχνει ποιες φωτοδίοδοι θα πρέπει να ανάψουν.
Αποθηκεύουμε τα bits που αντιστοιχούν στος διακόπτες στον καταχωρητή A.
Εάν όλοι οι διακόπτες είναι κλειστοί, το πρόγραμμα τερματίζει.
Διαφορετικά, κάνουμε διαδοχικές δεξιές ολισθήσεις, μέχρι να βρούμε το πρώτο bit που να είναι 1 (που αντιστοιχεί σε ανοιχτό διακόπτη).
Με κάθε ολίσθηση αυξάνουμε τον “δείκτη” στον (B) κατά 1.
Όταν βρούμε τον πρώτο ανοιχτό διακόπτη (ξεκινώντας από το LSB), μεταφέρουμε το περιεχόμενο του B στον A.
Συμπληρώνουμε τα ψηφία και μεταφέρουμε το περιεχόμενο στη θέση μνήμης 3000H. Θυμίζουμε πως οι φωτοδίοδοι ανάβουν όταν το αντίστοιχο bit ισούται με 0.

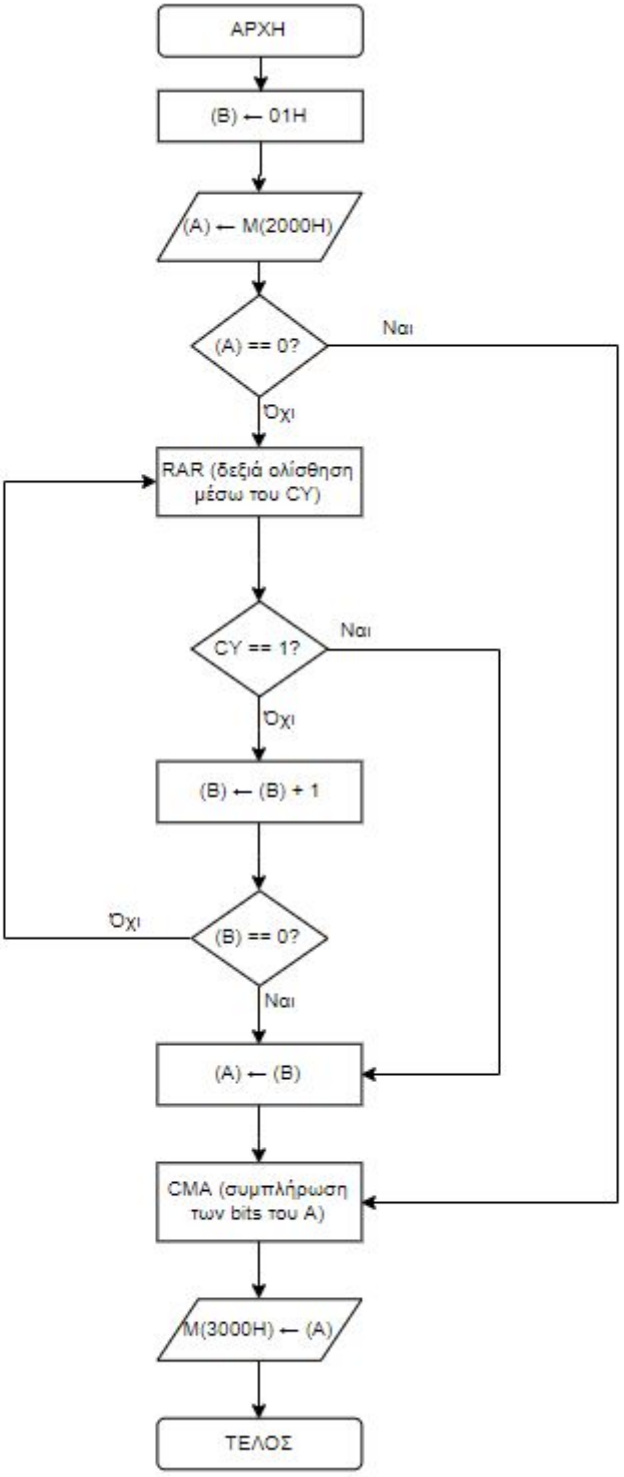
Στην πράξη, θεωρώντας 1 έως 8 από τα δεξιά προς τα αριστερά, όποιος διακόπτης είναι ανοικτός ,δηλαδή τα επάνω, ανοίγουν τα αντίστοιχα λεντάκια από κάτω για την αναπαράστασή του ως δυαδικό αριθμό. Εάν περισσότεροι από έναν διακόπτη είναι ανοιχτοί, έχει προτεραιότητα ο δεξιότερος. Για παράδειγμα αν ανοίξουμε το 5ο διακόπτη θα ανοίξουν το τρίτο και το πρώτο λεντάκι αφού ο αριθμός 5 σε δυαδική μορφή είναι ο 0101. Παρακάτω απεικονίζεται αυτό το αποτέλεσμα:



Για να έχουμε συνεχή λειτουργία του παραπάνω προγράμματος, δηλαδή να επαναλαμβάνεται χωρίς τέλος θα πρέπει με μια εντολή να αντί να τερματίζει να πηγαίνει πάλι στην αρχή START. Αυτό μπορεί να συμβεί αν αντικαταστήσουμε την εντολή RST 1 με την εντολή άλματος χωρίς έλεγχο JMP START.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ:

Επειδή η θέση μνήμης 2000H αντιστοιχεί στους διακόπτες, τη θεωρούμε είσοδο.
Όμοια, επειδή η θέση μνήμης 3000H αντιστοιχεί στα LED, τη θεωρούμε έξοδο.



Άσκηση 2

Θα χρησιμοποιήσουμε τον καταχωρητή H για να αποθηκεύουμε έναν “δείκτη” που θα δείχνει τη φορά της κίνησης μας. Εάν (H) = 1, τότε κινούμαστε προς τα αριστερά, εάν (H) = 0, τότε κινούμαστε προς τα δεξιά.

Θα αποθηκεύουμε στον κατχωρητή E τα bits που θα δείχνουν ποιοι διακόπτες θα είναι ανοιχτοί κάθε φορά.

```
    IN 10H          ; Αίρουμε την προστασία της μνήμης
    LXI B,0200H     ; (B)(C) <- 512D
    MVI H,01H       ; Αρχικοποιούμε τον δείκτη που δείχνει τη φορά κίνησης
    MVI A,FEH       ; Αρχικοποιούμε τα bits των LEDs στο 11111110
    MOV E,A         ; Τα αποθηκέυουμε και στον E
    STA 3000H       ; Θα ανάψει μόνο το LSB LED

START:
    CALL DELB       ; Καλούμε την DELB για καθυστέρηση ίση με 512ms
    LDA 2000H       ; Φορτώνουμε τα bits των διακοπτών στον A
    MOV D,A         ; Και τα αποθηκεύουμε στον D
    RRC             ; Κάνουμε δύο δεξιές ολισθήσεις
    RRC             ; για να ελέγξουμε τον δεύτερο διακόπτη
    JC  PAUSE       ; Αν ο δεύτερος διακόπτης είναι ON κάνουμε άλμα στο PAUSE

                    ; Διαφορετικά συνεχίζουμε στην κίνηση
    MOV A,H         ; Μεταφέρουμε το περιεχόμενο του δείκτη φοράς στον A
    CPI 00H         ;
    JZ  RIGHT       ; Αν είναι 0, πάμε στην κίνηση προς τα δεξιά
                    ; Αν είναι 1 συνεχίζουμε εδώ (κίνηση προς τα αριστερά)
    MOV A,D         ; Μεταφέρουμε τα bits των διακοπτών στον A
    RRC             ; Δεξιά ολίσθηση για να δούμε το status του LSB διακόπτη
    JC LEFTP        ; Αν είναι ON κάνουμε άλμα στην αριστερή παλινδρομική

LEFTC:              ; Αν είναι OFF συνεχίζουμε στην αριστερή κυκλική
    MOV A,E         ; Μεταφέρουμε στον A την τελευταία θέση των LEDs
    RLC             ; Κάνουμε αριστερή ολίσθηση
    STA  3000H      ; Αλλάζουμε τα bits των LEDs
    MOV E,A         ; Αποθηκεύουμε στον E το νέο status των LEDs
    JMP START       ; Ξεκινάμε από την αρχή

AA:                 ; Θα φτάσουμε εδώ ανν η δεξιά παλινδρομική κίνηση φτάσει
    MVI H,01H       ; στο τελευταίο (LSB) bit, οπότε αλλάζουμε το δείκτη φοράς

LEFTP:              ; και συνεχίζουμε με αριστερή παλινδρομική
    MOV A,E         ; Όμοια με την αριστερή κυκλική, βρίσκουμε τη θέση των LEDs
    RLC             ; και κάνουμε αριστερή ολίσθηση
```

```
    JNC DD          ; Όμως, εάν ξεπεράσαμε το τελευταίο (αριστερότερο bit)
                    ; Κάνουμε άλμα στη δεξιά παλινδρομική αντ' αυτού
    STA 3000H       ; Αλλάζουμε τα bits των LEDs
    MOV E,A         ; Μεταφέρουμε στον E
    JMP START       ; Πάμε στην αρχή

RIGHT:              ; Αν ο δείκτης φοράς στον H είναι 1, κάνουμε δεξιά κίνηση
    MOV A,D         ; Όμοια με την αριστερή κίνηση,
    RRC             ; ελέγχουμε τον LSB διακόπτη
    JC RIGHTP       ; για να δούμε αν θέλουμε παλινδρομική ή κυκλική

RIGHTC:             ; Δεξιά κυκλική κίνηση
    MOV A,E         ; Όμοια με την αριστερή κυκλική,
    RRC             ; Με την προφανή διαφορά ότι εδώ κάνουμε δεξιά ολίσθηση
    STA  3000H
    MOV E,A
    JMP START

DD:                 ;Θα φτάσουμε εδώ ανν η αριστερή παλινδρομική φτάσει
    MVI H,00H       ;στο τελευταίο (MSB) bit, οπότε αλλάζουμε το δείκτη φοράς

RIGHTP:             ; Δεξιά παλινδρομική κίνηση
    MOV A,E         ; Όμοια με την αριστερή παλινδρομική
    RRC             ; Με την προφανή διαφορά ότι κάνουμε δεξιά ολίσθηση
    JNC AA
    STA 3000H
    MOV E,A
    JMP START

PAUSE:              ; Θα φτάσουμε εδώ ανν ο δεύτερος διακόπτης είναι ON
    MVI A,FEH       ; Μεταφέρουμε στον A το 11111110
    STA 3000H       ; Αλλάζουμε τα bits των LEDs ώστε να ανάβει μόνο το LSB
    JMP START
    END
```

Άσκηση 3

START:

```
 LDA 2000H     ; Φόρτωση της εισόδου στον καταχωρητή A
 MVI B,FFH     ; Συμπλήρωμα ως προς 2 του -1 και αποθήκευση στον B
 CPI 63H       ; Σύγκριση για το αν A>(63)H = 99D
 JC DECA       ; Εάν ο αριθμός είναι μικρότερος του 99, πάμε στο DECA
 SUI 64H       ; Διαφορετικά αφαιρούμε 100D και συνεχίζουμε ομοίως
```

DECA:

```
 INR B         ; Αύξηση του περιεχομένου του B κατά 1
 SUI 0AH       ; Μείωση του περιεχομένου του A κατά 10
 JNC DECA      ; Αν A>0. συνέχισε τις αφαιρέσεις
 ADI 0AH       ; Αν A<0, διόρθωσε το αρνητικό υπόλοιπο
 MOV E,A       ; Back up του περιεχομένου του καταχωρητή A στον E
 MOV A,B       ; Βάζω στον A το περιεχόμενο του B δηλαδή τις δεκάδες
 RLC           ; Ολίσθηση 4 φορές ώστε οι δεκάδες να πάνε
 RLC           ; στα 4 πρώτα (MSB) bits του A
 RLC
 RLC
 ADD E         ; πρόσθεση του E στον A. Έτσι οι μονάδες (E)
               ; θα αποθηκευτούν στα 4  τελευταία (LSB) bits του A
 CMA           ; συμπλήρωση της εξόδου
 STA 3000H     ; αποθήκευση στην έξοδο (θέση μνήμη 3000H)
 JMP START     ; πήγαινε ξανά στην αρχή
```

END

Άσκηση 4

Αρχικά σημαντικό είναι να εκφράσουμε τις συναρτήσεις κόστους, και συνάρτηση κόστους ανά τεμάχιο για τις τεχνολογίες, 1η τεχνολογία IC, 2η τεχνολογία FPGAs, 3η τεχνολογία SoC.

1η τεχνολογία(IC):

Συνάρτηση κόστους: κόστος=20.000+(15+15)χ =20.000 + 30χ
χ>0

Συνάρτηση κόστους ανά τεμάχιο: $\kappa o \sigma \tau o \varsigma / \tau \varepsilon \mu \acute{\alpha} \chi \iota o$ = (20.000 /χ) +30
χ>0

2η τεχνολογία (FPGAs):

Συνάρτηση κόστους : Κόστος=10.000+(60+10)χ=10.000+70χ
χ>0

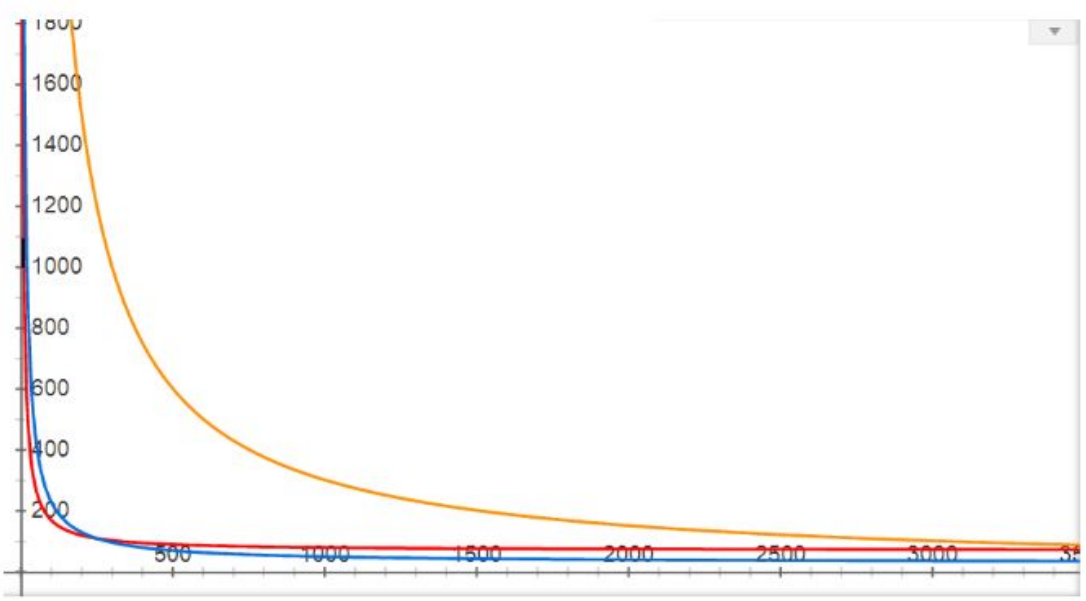
Συνάρτηση κόστους ανά τεμάχιο: $\kappa o \sigma \tau o \varsigma / \tau \varepsilon \mu \acute{\alpha} \chi \iota o$ =(10000 /χ) + 70
χ>0

3η τεχνολογία (SoC):

Συνάρτηση κόστους: Κόστος=300.000+(1+1)χ=300.000+2χ
χ>0

Συνάρτηση κόστους ανά τεμάχιο: $\kappa o \sigma \tau o \varsigma$ /τεμάχιο = (300.000/χ) +2 χ>0

Συνάρτηση κόστους ανά τεμάχιο:



Με μπλε χρώμα απεικονίζεται η γραφική παράσταση της 1ης τεχνολογίας

Με κόκκινο χρώμα απεικονίζεται η γραφική παράσταση της 2ης τεχνολογίας

Με κίτρινο χρώμα απεικονίζεται η γραφική παράσταση της 3ης τεχνολογίας

Ο άξονας των y συμβολίζει τον αριθμό των τεμαχίων.

Για να μπορέσουμε να βρούμε τα συμφερότερα διαστήματατης κάθε τεχνολογίας πρέπει να εντοπίσουμε τα σημεία τομής των παραπάνω καμπυλών.

1. Η 1η με την 2η τεχνολογία θα έχουν ίδιο κόστος για έστω κ1 τεμάχιά.

Άρα 20.000 + 30*κ1 = 10.000 + 70*κ1 => κ1 = 250 τεμάχια.

Από την γραφική παράσταση μπορούμε να διακρίνουμε ότι όσο μεγαλύτερο αρχικό κόστος έχει μία τεχνολογία τόσο προτιμότερο είναι να έχει μεγαλύτερο πλήθος τεμαχίων. Έτσι θα ήταν προτιμότερη η τεχνολογία 1 για πλήθος μεγαλύτερο από 250 τεμάχια . Και αντίστοιχα η τεχνολογία 2 θα ήταν καλύτερη για αριθμό τεμαχίων μικρότερο από 250.

2. Η 1η με την 3η τεχνολογία θα έχουν το ίδιο κόστος για έστω κ2 τεμάχια.

Άρα 20.000 + 30*κ2 = 300.000 + 2 *κ2 => κ2 = 10.000 τεμάχια.

Ομοίως με πριν, <u>πιο συμφέρουσα τεχνολογία για πλήθος τεμαχίων μικρότερο από 10.000 είναι η 1η</u> , ενώ για πλήθος μεγαλύτερο από 10.000 είναι η 3η .

Έτσι, καταλήγουμε στα εξής διαστήματα προτεινόμενων τεχνολογιών :

- 1η τεχνολογία : 250<κ<10.000
- 2η τεχνολογία : 0<κ<250
- 3η τεχνολογία : κ> 10.000

Όπου κ είναι ο αριθμός των τεμαχίων.

Για να «εξαφανιστεί» η επιλογή της πρώτης τεχνολογίας (τεχνολογία IC) θα πρέπει να μην αποτελεί την πλέον συμφέρουσα τεχνολογία για κανένα αριθμό τεμαχίων. Ισοδυνάμως, για κάθε τεμάχιο (x>0) θα πρέπει σε ό,τι αφορά την τεχνολογία SoC:

<u>Αρχικό κόστος + 2*χ ≤ 20.000 + 30*χ ⇒ αρχικό κόστος ≤ 20.000+ 28*χ.</u> με χ>0 Για x = 251 τεμάχια, προκύπτει η ζητούμενη τιμή του αρχικού κόστους:
<u>αρχικό κόστος ≤ 27028 € και *οριακά* ⇒ αρχικό κόστος =27028 €</u>

Για να «εξαφανιστεί» η επιλογή της πρώτης τεχνολογίας (τεχνολογία IC) από τη ρύθμιση του μεταβλητού κόστος $FPGA$ θα πρέπει:
20.000+30*κ>=10.000+(χ+10)κ => 10.000>=(χ+10)κ-30*κ => 10.000 >=(χ-20)*κ => 10.000/κ>=χ-20 => (χ-20)*κ<=10.000, όπου κ τα τεμάχια , 251<=κ<=9999 και
Δηλαδή πρέπει χ-20>0 => χ>20
Συνεπώς, προκειμένου να είναι η 2η τεχνολογία προτιμότερη από την πρώτη, θα πρέπει χ≤ 20. Έστω, λοιπόν, χ=20.

Άσκηση 5

Θα κάνουμε απλοποίηση με χάρτες Karnaugh και θα υλοποιήσουμε τις απλοποιημένες συναρτήσεις

α)

$F_1 = A(CD + B) + BC'D'$

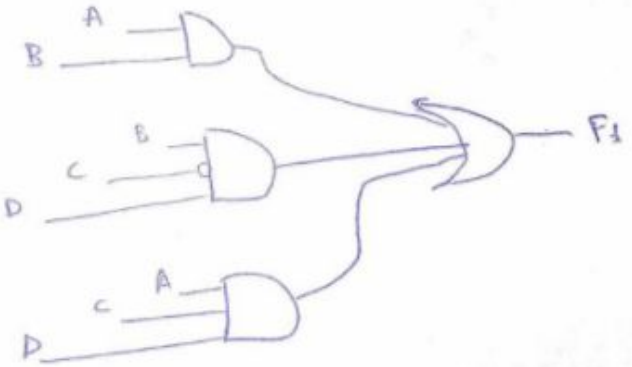
$F_1 = ACD + AB + BC'D'$

Απλοποιημένο:

$F_1 = AB + BC'D + ACD$

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

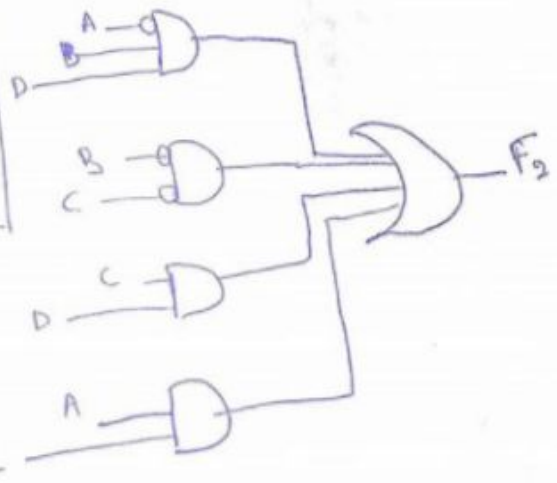
APA



$F2(A,B,C,D) = \Sigma(0,2,3,5,7,8,10,11,14,15)$

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

Απλ $F_2 = A'BD + B'C' + CD + AC$



$f_3 = ABC + (A+B)CD + (B+CD)E$

$f_3 = ABC + ACD + BCD + BE + CDE$

Για $(E=0)$ ο πίνακας →

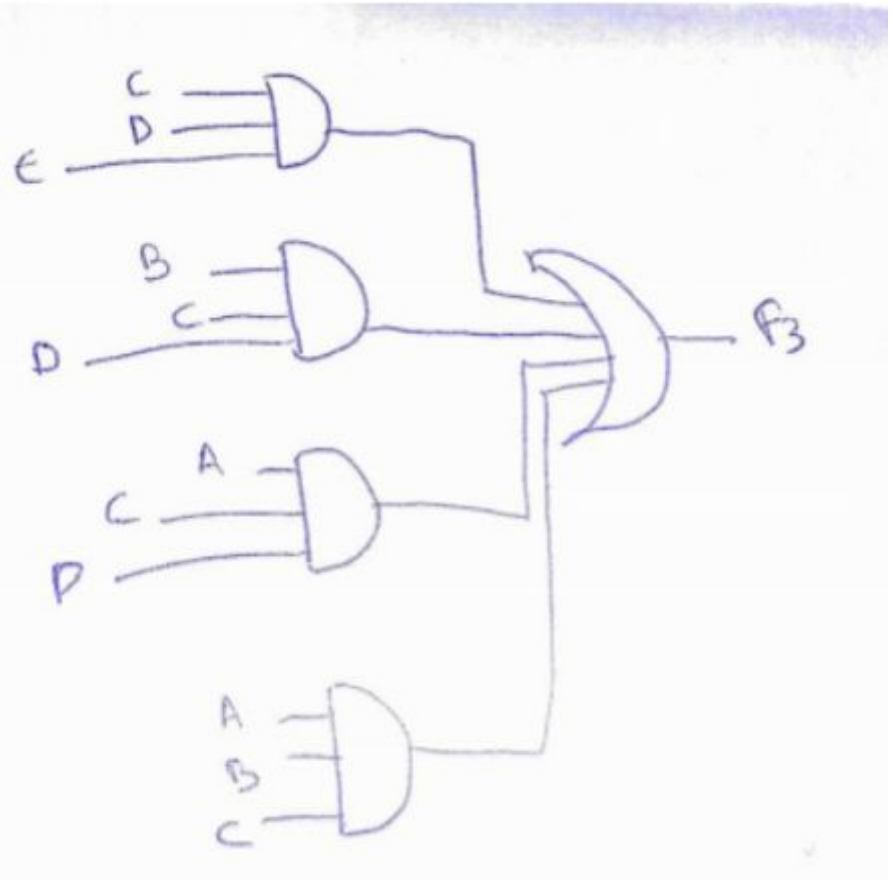
AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	0	1	1
10	0	0	1	0

Και για $(E=1)$ ↓

Άρα :

$f_3 = CDE + BCD + ACD + ABC$

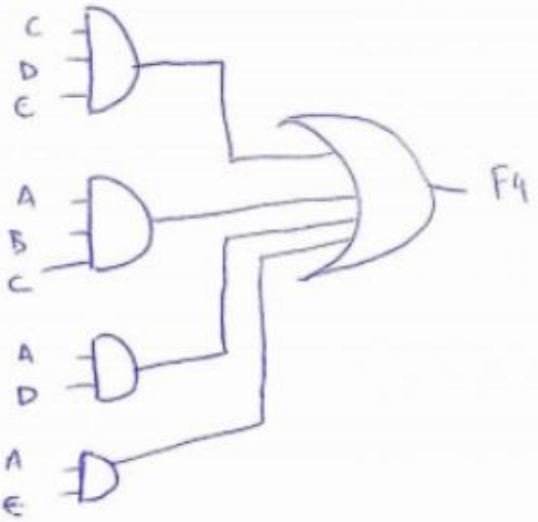
AB \ CD	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	1
10	0	0	1	0



$F_4 = A(BC + D + E) + CDE$

$F_4 = ABC + AD + AE + CDE$

Apa: $F_4 = CDE + ABC + AD + AE$



Fia (E=0)

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

Fia (E=1)

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

β) F1=AB +BC'D+ACD

```
module F_1(A,B,C,D,F1);
input A,B,C,D;
output F1;
assign  F1=((A&B)|(B&~C&D)|(A&C&D));
endmodule
```

F2=A'BD+B'C'+CD+AC

```
module F_2(A,B,C,D,F2);
input A,B,C,D;
output F2;
assign  F2=((~A&B&D)|(~B&~C)|(C&D)|(A&C));
endmodule
```

F3=CDE+BCD+ACD+ABC

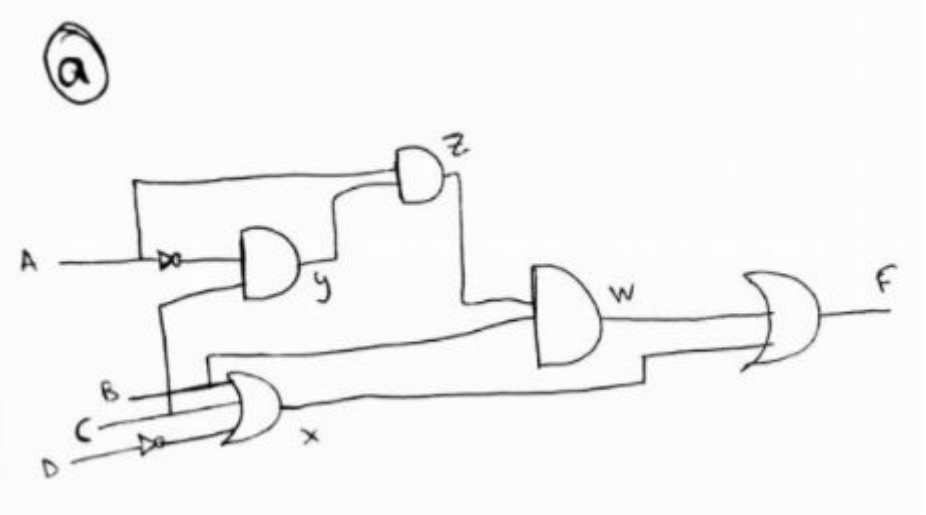
```
module F_3(A,B,C,D,E,F3);
input A,B,C,D,E;
output F3;
assign  F3=((C&D&E)|(B&C&D)|(A&C&D)|(A&B&C));
endmodule
```

F4=CDE+ABC+AD+AE

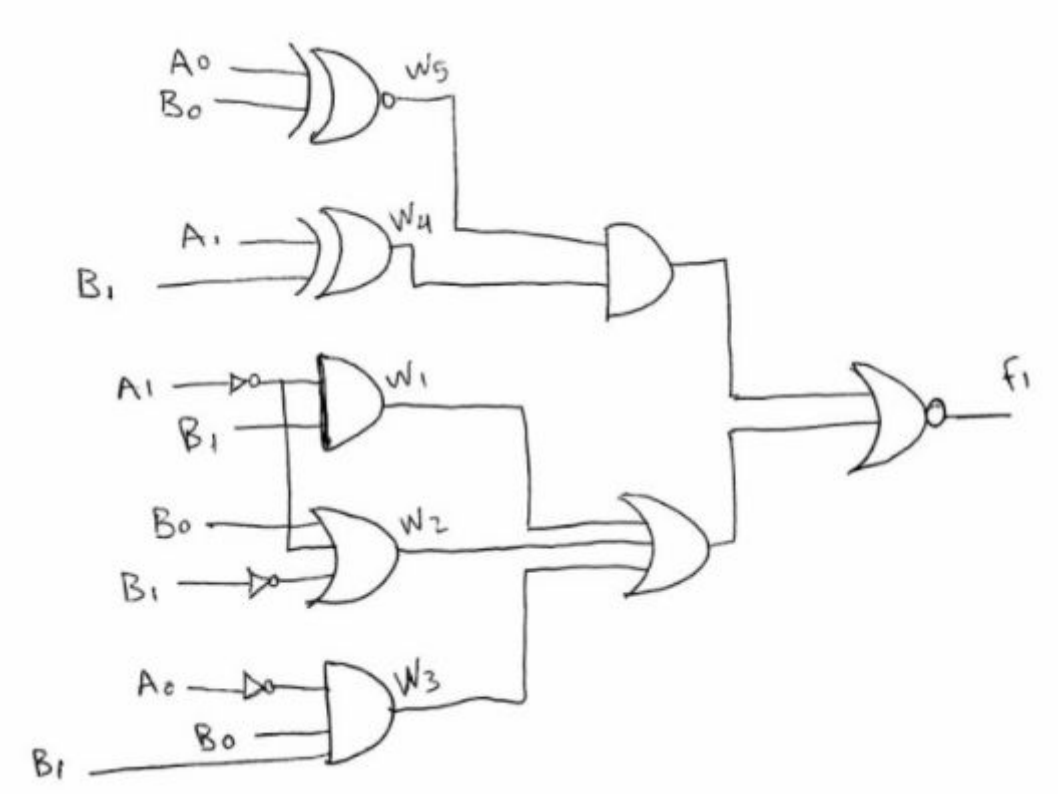
```
module F_4(A,B,C,D,E,F4);
input A,B,C,D,E;
output F4;
assign  F4=((C&D&E)|(A&B&C)|(A&D)|(A&E));
endmodule
```

Άσκηση 6

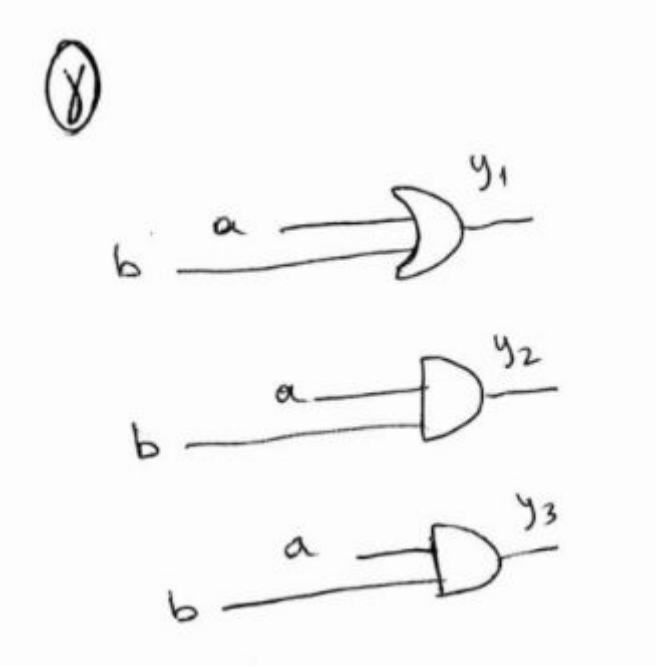
3.36



β)



y)



4.37

```
module half_adder (S,C,x,y);
output S,C ;
input x,y;
xor (S,x,y);
and (C,x,y);
endmodule

module full_adder (S,C,x,y,z);
output S,C;
input x,y,z ;
wire S1,C1,C2;
half_adder HA1(S1,C1,x,y);
half_adder HA2(S,C2,S1,z);
or G1(C,C2,C1);
endmodule
```

```
module ex_4_37(S,C,A,B,M);
output [3:0] S;
output C;
input [3:0] A;
input [3:0] B;
input M;
wire [3:0] B_xor_M;
wire C1,C2,C3;
xor (B_xor_M[0], B[0], M);
xor (B_xor_M[1], B[1], M);
xor (B_xor_M[2], B[2], M);
xor (B_xor_M[3], B[3], M);
full_adder FA1 (S[0], C1, A[0], B_xor_M[0], M);
full_adder FA2 (S[1], C2, A[1], B_xor_M[1], C1);
full_adder FA3 (S[2], C3, A[2], B_xor_M[2], C2;
full_adder FA4 (S[3], C, A[3], B_xor_M[3], C3);
endmodule
```

4.40

```
module 4_37(S,C,A,B,M);
input [3:0] A,B;
input M;
output [3:0] S;
output C;
assign {C,S} = (M) ? A-B : A+B;
endmodule
```

Άσκηση 7

5.48

```
module ex_5_48 (output reg y_out, input x_in, clock, reset);
        parameter s_a = 2'b00, s_b = 2'b01, s_c = 2'b10, s_d = 2'b11;
        reg [1: 0] state, next_state;

 always @ (posedge clock, negedge reset)
        if (reset == 0) state <= s_a;
        else state <= next_state;

always @ (state, x_in)
        case (state)
                s_a: if (x_in==0) begin next_state = s_b; y_out = 1; end
                else begin next_state = s_c; y_out = 0; end
                s_b: if (x_in==0) begin next_state = s_c; y_out = 0; end
                else begin next_state = s_d; y_out = 1; end
                s_c: if (x_in==0) begin next_state = s_b; y_out = 0; end
                else begin next_state = s_d; y_out = 1; end
                s_d: if (x_in==0) begin next_state = s_c; y_out = 1; end
                else begin next_state = s_a; y_out = 0; end

        endcase

endmodule
```

5.49

```
module ex_5_49 (output reg y_out, input x_in, clock, reset);
        parameter s_a = 2'b00, s_b = 2'b01, s_c = 2'b10, s_d = 2'b11;
        reg [1: 0] state, next_state;

always @ (posedge clock, negedge reset)
        if (reset == 0) state <= s_a;
        else state <= next_state;

always @ (state, x_in)
        case (state)
                s_a: begin y_out = 0; if (x_in == 0) next_state = s_b;
                else next_state = s_c; end
                s_b: begin y_out = 1; if (x_in == 0) next_state = s_c;
                else next_state = s_d; end
                s_c: begin y_out = 1; if (x_in == 0) next_state = s_b;
                else next_state = s_d; end
                s_d: begin y_out = 0; if (x_in == 0) next_state = s_c;
                else next_state = s_a; end

        endcase

endmodule
```

6.35f

```
module ex_6_35f (
	output reg [3: 0] A_par,
	input [3: 0] I_par, input MSB_in, LSB_in, s1, s0, clock, Clear
);

always @ (posedge clock, negedge Clear)
	if (Clear == 0) A_par <= 4'b0;
	else case ({s1, s0})
		2'b00: A_par <=A_par;
		2'b01: A_par <= 4'b0;
		2'b10: A_par <= ~A_par;
		2'b11: A_par <= I_par;

	endcase

endmodule
```

6.35l

```
module ex_6_35f (
	output reg [3: 0] A,
	input clock, Up, Down, reset
);

always @ (posedge clock, negedge reset)
	if (reset ==0) A <= 4'b0;
	else case ({Up, Down})
		2'b00: A <=A;
		2'b10: A <= A + 4'b0001;
		2'b01: A <= A - 4'b0001;
		2'b11: A <=A;

	endcase

endmodule
```