Συστήματα Μικρουπολογιστών

1η Ομάδα Ασκήσεων

Μετζάκης Ιωάννης

Α.Μ.: 03116202

ΣΗΜΜΥ 8ο

1η ΑΣΚΗΣΗ

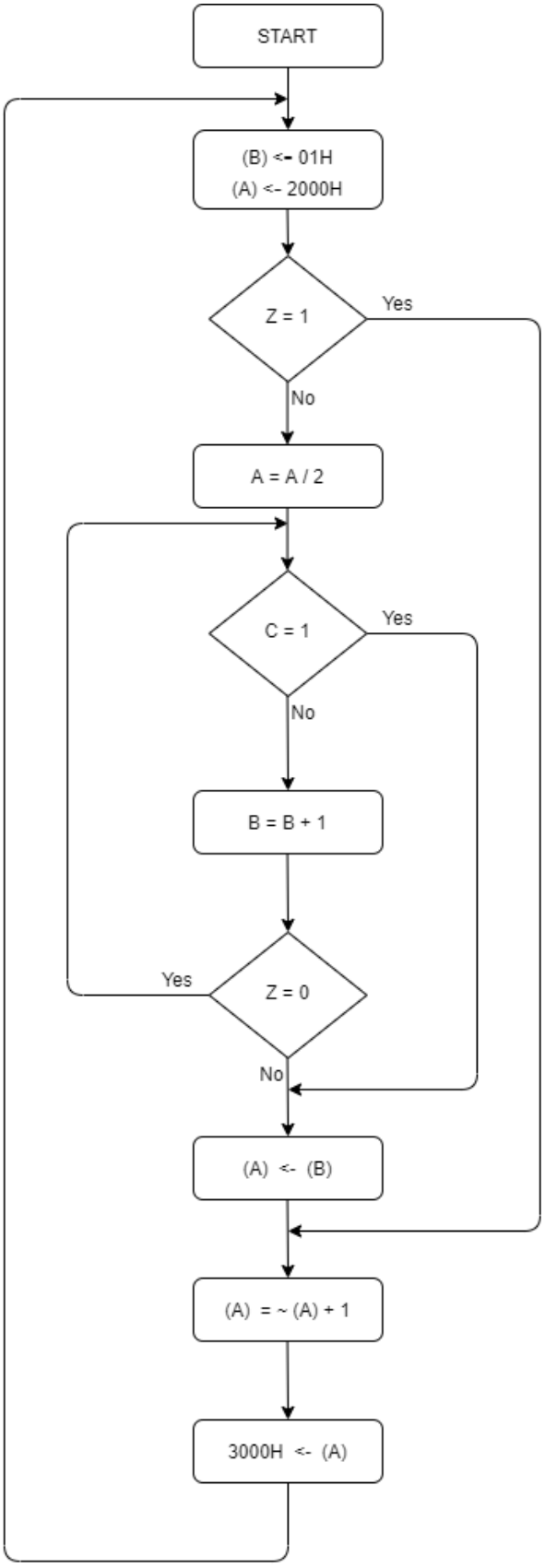
Μία σύντομη επεξήγηση των εντολών του προγράμματος:

06:	**MVI B,byte**	Φορτώνει στον καταχωρητή B έναν αριθμό.
3A:	**LDA adr**	Φορτώνει στον καταχωρητή A το περιεχόμενο της διεύθυνσης adr.
FE:	**CPI byte**	Συγκρίνει το περιεχόμενο του καταχωρητή A με έναν αριθμό.
CA:	**JZ adr**	Αν Z=1 μεταβαίνει στη διεύθυνση adr (συνήθως αντικαθείσταται από μία label).
1F:	**RAR**	Περιστρέφει τα bits του καταχωρητή A κατά μία θέση δεξιά.
DA:	**JC adr**	Ότι και η **CA**, αλλά όταν C=1
04:	**INR B**	Αυξάνει το περιεχόμενο του καταχωρητή B κατά 1.
C2:	**JNZ adr**	Ότι και η **CA**, αλλά όταν Z=0
78:	**MOV A,B**	Φορτώνει στον καταχωρητή A το περιεχόμενο του καταχωρητή B.
2F:	**CMA**	Συμπληρώνει ως προς 1 το περιεχόμενο του A.
32:	**STA adr**	Αποθηκεύει στη διεύθυνση adr το περιεχόμενο του καταχωρητή A.
CF:	**RST 1**	Επιστρέφει στην αρχική διεύθυνση.

Το πρόγραμμα στην ουσία αποτυπώνει στα leds(3000H) τη δυαδική αναπαράσταση της θέσης του δεξιότερου ανοιχτού διακόπτη.

Για να έχουμε σε συνεχόμενη μορφή τη λειτουργία του προγράμματος, αρκεί να προσθέσουμε μία label (πχ “START”) στην αρχή του προγράμματος, και την εντολή “JMP START” αντί της RST 1.

Παρακάτω βρίσκεται το διάγραμμα ροής για το πρόγραμμα συνεχούς λειτουργίας, και στη συνέχεια ο κώδικας σε assembly 8085:



Και ο κώδικας σε assembly 8085:

```
START:
MVI B,01H     ;vale B = 01H
LDA 2000H     ;fortwse to periex. ths 2000H
CPI 00H       ;sigrine ton A me to 0

JZ FIRST      ;an Z=1 phgaine FIRST

THIRD:
RAR           ;deksia olisthish tou A
JC SECOND     ;an C=1 phgaine SECOND

INR B         ;afksise kata 1 ton B
JNZ THIRD     ;an Z=0 phgaine THIRD

SECOND:
MOV A,B       ;ton B ston A

FIRST:
CMA           ;arnhtikh logikh ta leds
STA 3000H     ;emfanise sthn 3000H

JMP START     ;phgaine START

END
```

2η ΑΣΚΗΣΗ

Μία σύντομη περιγραφή υλοποίησης:

Ξεκινάμε την κίνηση από το LSB της θύρας εξόδου 3000 Hex.
Έχουμε τον καταχωρητή L ως σημαία για να ξέρουμε την κατεύθυνση που θέλουμε να πάμε στην επόμενη 'λούπα'.
Ελέγχουμε πρώτα το 2^{ο} LSB της θύρας εισόδου 2000 Hex, και όσο είναι 1 (ON ο διακόπτης), ανάβουμε χωρίς να κινείται το led της θέσης 0.
Κάθε φορά ελέγχουμε αν είμαστε σε οριακή κατάσταση, δηλαδή στα δύο άκρα των leds(MSB και LSB).
Αν όχι, συνεχίζουμε κανονικά με τις αντίστοιχες ολισθήσεις και την απεικόνιση.
Αν είμαστε ωστόσο, ελέγχουμε την κατάσταση του LSB διακόπτη της θύρας εισόδου 2000 Hex, και αν είναι ON, μεταβαίνουμε στην εναλλακτική ρουτίνα(αν είμασταν στην left, μεταβαίνουμε στην right και αντίστροφα).

Ο κώδικας βρίσκεται παρακάτω:

```
IN 10H
LXI B,0244H          ;orismos kathisterishs, opou 0244H=500d
MVI A,FEH
STA 3000H            ;emfanish MSB led(arnhtikh logikh sta LED,gi auto 7FH)
CMA
MOV D,A              ;o D tha exei to apotelesma pou tha emfanizoume
CALL DELB            ;routina kathisterishs
MVI L,01H            ;L kataxwriths gia katefthinsi (1 left, 0 right)
LOOP1:
       LDA 2000H
       ANI 02H       ;apomonwsh 2ou LSB diakopth kai elegxos
       JZ OKK
       MVI A,FEH     ;emfanise LSB led kai kane loupa mexri o diakoptis OFF
       STA 3000H
       JMP LOOP1

OKK:
       MOV A,L
       CPI 01H       ;elegxoume to L, kai an einai iso me 01H
       JZ LEFT       ;an nai, tote phgaine LEFT
       JMP RIGHT     ;alliws phgaine RIGHT

LEFT:
       MOV A,D       ;metaferoume to byte sto A kai elegxoume an vrisketai
       CPI 80H       ;sto MSB tou output
       JZ ELEGXOS    ;an nai phgaine ELEGXOS gia na doume ti theloume meta

       RLC           ;alliws olisthish aristera kai phgaine DISPLAY1
       JMP DISPLAY1

ELEGXOS:
       LDA 2000H     ;diavazoume 2o lsb diakopth apo 2000H
       ANI 01H
       CPI 01H
       JZ RIGHT      ;an einai ON, pame right
       MVI D,01H     ;alliws vazoume D = 01H kai to emfanizoume sto DISPLAY1
       MOV A,D

DISPLAY1:
       CMA           ;arnhtikh logikh ta LED, ara antistrefoume
       STA 3000H     ;emfanizoume
       CMA           ;kai epanaferoume
       MOV D,A
       CALL DELB     ;routina kathisterishs
       MVI L,01H     ;fortwsh tou 01H sto L wste na ksanapaei aristera meta
       JMP LOOP1     ;epistrofi sto loop1


RIGHT:
       MOV A,D              ;paromoios elegxos me prin
       CPI 01H              ;sto LSB tou output
       JZ ELEGXOS1          ;an nai phgaine ELEGXOS1

       RRC                  ;olisthish deksia
       JMP DISPLAY2         ;alliws de xreiazetai elegxos

ELEGXOS1:
       LDA 2000H
       ANI 01H
       CPI 01H
       JZ LEFT       ;to idio me prin, antistoixa
       MVI D,80H
       MOV A,D

DISPLAY2:
       CMA
       STA 3000H
       CMA
       MOV D,A
       CALL DELB
       MVI L,00H     ;fortwnoume sto L to 00H gia na ksanapaei deksia
       JMP LOOP1
END
```

3η ΑΣΚΗΣΗ

Παρακάτω βρίσκεται ο τελικός κώδικας της 3ης άσκησης μαζί με τα απαραίτητα σχόλια:

```
LOOPA:
       MVI D,00H
       LDA 2000H
       CPI 64H       ;elegxoume an >=100
       JNC BIGGER    ;an nai pame sth routina BIGGER

DEKADES:
       CPI 0AH       ;elegxoume an >=10
       JC MONADES    ;an nai pame MONADES

       INR D         ;alliws afksanoume kata 1 to D
       SUI 0AH       ;kai meiwnoume kata 10 to A
       JMP DEKADES   ;ksana to idio mexri A<10

MONADES:
       MOV E,A       ;apothikeoume to A(tis monades mas diladi)
       MOV A,D       ;metaferoume to D(tis dekades mas)
       RLC           ;kai kanoume 4 olisthiseis aristera gia na
       RLC           ;paei sta 4 MSB
       RLC
       RLC
       MOV D,A
       MOV A,E

       ADD D         ;prosthetoume sto apotelesma twn RLC, tis monades
       CMA           ;antistrefoume gt exoun arnhtikh logikh ta LED
       STA 3000H
       JMP LOOPA     ;ksana ap thn arxh


BIGGER:
       CPI C8H       ;elegxoume an >=200
       JC UNDERDIAK

       SUI 64H       ;an einai, afairoume prwta 100,
                     ;wste me ta epomena 100 na afairethoun sinolika 200
UNDERDIAK:
       SUI 64H
       JMP DEKADES   ;exoume ton arithmo mas mod 100, opote pame DEKADES

END
```

Όλα τα παραπάνω (3) προγράμματα υπάρχουν σε ξεχωριστά αρχεία στον ίδιο φάκελο με την παρούσα αναφορά.

4η ΑΣΚΗΣΗ

Παρακάτω παρατίθενται οι συναρτήσεις κόστους για κάθε μία από τις 3 τεχνολογίες της εκφώνησης:

1. Χρήση διακριτών στοιχείων και ολοκληρωμένων μονάδων (I.C.) όπως μικροελεγκτών, περιφερειακών, μνημών κ.λπ. τα οποία συναρμολογούνται σε μια σε μια σχετικά μεγάλη πλακέτα.

   Κόστος / τεμάχιο = (20000 + 30x) , όπου x είναι το πλήθος των τεμαχίων

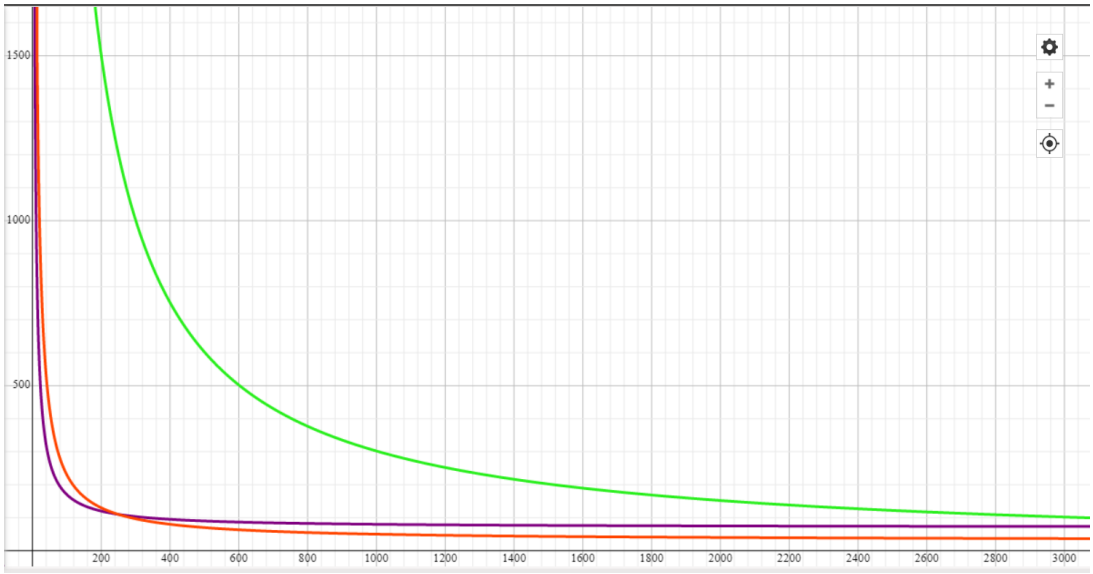
2. Χρήση FPGAs και μικρού αριθμού περιφερειακών τοποθετημένα σε μια μικρή πλακέτα.

   Κόστος / τεμάχιο = (10000 + 70x) , όπου x είναι το πλήθος των τεμαχίων

3. Σχεδίαση ειδικού SoC με μια πολύ μικρή πλακέτα.

   Κόστος / τεμάχιο = (300000 + 2x) , όπου x είναι το πλήθος των τεμαχίων

Οι καμπύλες κόστους κάθε τεχνολογίας φαίνονται στο παρακάτω διάγραμμα:



Όπου ο άξονας x είναι το πλήθος των τεμαχίων, ο άξονας y το κόστος ανά τεμάχιο, ενώ η **πορτοκαλί** καμπύλη κόστους αντιστοιχεί στην **1η** τεχνολογία, η **μοβ** στη **2η**, και η **πράσινη** στην **3η**.

Παρατηρούμε ότι για χαμηλούς αριθμούς τεμαχίων, η $2^{η}$ τεχνολογία έχει το χαμηλότερο κόστος ανά τεμάχιο, ενώ από ένα αρκετά μεγάλο πλήθος τεμαχίων και έπειτα, η $3^{η}$ τεχνολογία είναι η προτιμότερη.

Αυτό ήταν αναμενόμενο βέβαια, αφού το αρχικό κόστος της $2^{ης}$ τεχνολογίας είναι αρκετά χαμηλό, σε αντίθεση με αυτό της $3^{ης}$, ωστόσο το μεταβλητό κόστος, όπου είναι συναρτήσει του πλήθους των πλακετών, αυξάνεται με πολύ μεγαλύτερο ρυθμό στη $2^{η}$ τεχνολογία (70x έναντι 2x , όπου x το πλήθος των τεμαχίων).

Οι πρώτες 2 τεχνολογίες (με χαμηλά αρχικά κόστη), εμφανίζουν ίδιο κόστος για πλήθος τεμαχίων x_1 ίσο με:

$$20000 + 30x_1 = 10000 + 70x_1 \quad => \quad \mathbf{x_1 = 250}$$

κάτι το οποίο επιβεβαιώνεται και από τη γραφική παράσταση.

Συνεπώς για $x < 250$ προτιμάται η $2^{η}$ τεχνολογία, όπως περιμέναμε δηλαδή, αφού είχε το μικρότερο αρχικό κόστος, άρα και το μικρότερο κόστος ανά τεμάχιο για χαμηλό πλήθος τεμαχίων.
Προφανώς, για $x > 250$ προτιμάται η $1^{η}$.

Οι τεχνολογίες 1 και 3, εμφανίζουν ίδιο κόστος για πλήθος τεμαχίων x_2 ίσο με:

$$20000 + 30x_2 = 300000 + 2x_2 \quad => \quad \mathbf{x_2 = 10000}$$

Άρα για $x > 10000$ προτιμάται η $3^{η}$ τεχνολογία, οπότε καταλήγουμε στις εξής τρεις περιοχές:

$0 < x < 250$	→	**$2^{η}$ τεχνολογία**
$250 < x < 10000$	→	**$1^{η}$ τεχνολογία**
$10000 < x$	→	**$3^{η}$ τεχνολογία**

Προφανώς, για τις οριακές τιμές 250 και 10000 οι δύο εκάστοτε τεχνολογίες έχουν το ίδιο κόστος.

Παρακάτω βρίσκουμε για ποια τιμή των I.C. στην τεχνολογία των FPGAs θα μπορούσε να εξαφανιστεί η επιλογή της πρώτης τεχνολογίας. Άρα, ψάχνουμε μία τιμή ώστε για κάθε αριθμό τεμαχίων, το συνολικό κόστος ανά πλακέτα στη $2^η$ τεχνολογία, να είναι χαμηλότερο από αυτό της $1^ης$.

$$20000 + 30x_2 > 10000 + (10 + y)x_2 \quad => \quad 10000 + (20 - y)x_2 > 0$$

Για να ισχύει για κάθε x αυτό, πρέπει η ποσότητα (20 – y) να είναι είτε θετική είτε μηδέν. Συνεπώς, $y \leq 20$.

Άρα, στην οριακή τιμή όπου το κόστος ανά πλακέτα των I.C. θα ήταν **20€**, η $1^η$ τεχνολογία δε θα ήταν ποτέ η προτιμότερη.

5η ΑΣΚΗΣΗ

i)

F1 = A(CD + B) + BC'D'

```
1 module ask5(A,B,C,D,F1);
2 output F1;
3 input A,B,C,D;
4 wire C_not,D_not,w1,CD,w1,w2,w3
5
6 not
7 G1(C_not,C),
8 G2(D_not,D);
9
10 and
11 G3(CD,C,D),
12 G4(w1,B,C_not,D_not);
13
14 or G5(w2,CD,B);
15 and G6(w3,A,w2);
16 or G7(F1,w3,w1);
17
18 endmodule
```

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

```
primitive ask5_2(A,B,C,D,F2);
output F2;
input A,B,C,D;
    table
    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 : 1;
    1 0 0 1 : 0;
    1 0 1 0 : 1;
    1 0 1 1 : 1;
    1 1 0 0 : 0;
    1 1 0 1 : 0;
    1 1 1 0 : 1;
    1 1 1 1 : 1;
endprimitive
```

F3 = ABC + (A + B)CD + (B + CD)E

```
module ask5_3(A,B,C,D,E,F3);
output F3;
input A,B,C,D,E;
wire w1,w2,w3,w4,w5,w6

and
G1(w1,A,B,C),
G2(w2,C,D);

or
G3(w3,A,B),
G4(w4,w2,B);

and
G5(w5,w3,w2),
G6(w6,w4,E);

or(F3,w1,w5,w6);

endmodule
```

F4 = A(BC + D + E) + CDE

```
module ask5_4(A,B,C,D,E,F4);
output F4;
input A,B,C,D,E;
wire w1,w2,w3,w4;

and
G1(w1,B,C),
G2(w2,C,D,E);

or G3(w3,w1,D,E);

and G4(w4,w3,A);

or G5(F4,w4,w2);
endmodule
```

ii)

F1 = A(CD + B) + BC'D'

```
module ask5_1(A,B,C,D,F1);
output F1;
input A,B,C,D;

assign F1 = ( (A & ((C & D)| B) ) | (B & ~C & ~D) )

endmodule
```

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

```
module ask5_2(A,B,C,D,F2);
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));

endmodule
```

F3 = ABC + (A + B)CD + (B + CD)E

```
module ask5_3(A,B,C,D,E,F3);
output F3;
input A,B,C,D,E;

assign F3 = ( (A&B&C) | ((A | B)&C&D) | ((B | (C*D))&E) )

endmodule
```

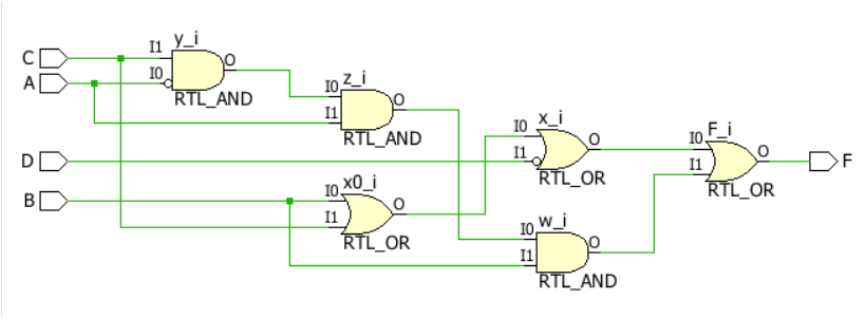
F4 = A(BC + D + E) + CDE

```
module ask5_4(A,B,C,D,E,F4);
output F4;
input A,B,C,D,E;

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

endmodule
```

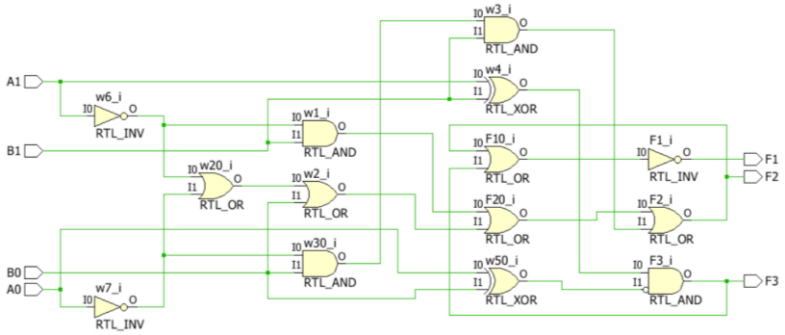
6η ΑΣΚΗΣΗ

i)

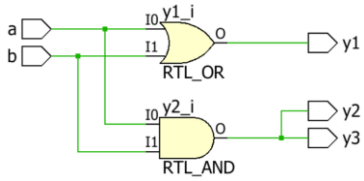
a)



b)



c)



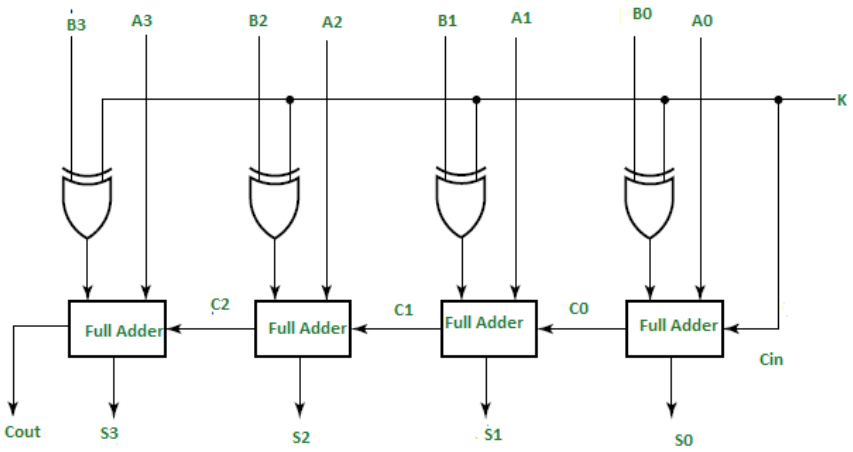
ii)

```
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), // Instantiate HAs
HA2 (S, C2, S1, z);
or G1 (C, C2, C1);
endmodule


module AddSub_4bit (A,B,Cin,Sum,Cout);
input [3:0] A,B;
input Cin;
output [3:0] Sum;
output Cout;
wire w1,w2,w3,C1,C2,C3;

xor G1 (w1,Cin,B[0]);
full_adder FA1 (A[0],w1,Cin,Sum[0],C1);
xor G2 (w2,Cin,B[1]);
full_adder FA2 (A[1],w2,C1,Sum[1],C2);
xor G3 (w3,Cin,B[2]);
full_adder FA3 (A[2],w3,C2,Sum[2],C3);
xor G4 (w4,Cin,B[3]);
full_adder FA4 (A[3],w4,C3,Sum[3],Cout);

endmodule
```

iii)

```
module add_sub (A, B, Cin, Sum, Cout );
input [3: 0] A, B;
input Cin;
output [3: 0] Sum;
output Cout;
assign {Cout, Sum} = Cin ? (A + ~B + Cin) : (A + B + Cin);
endmodule
```

Όπου το κύκλωμα αθροιστή-αφαιρέτη τεσσάρων bits μη προσημασμένων αριθμών, είναι το παρακάτω:



7η ΑΣΚΗΣΗ

i)

```
module Moore_Model (y, x, clock);
output y;
input x, clock;
reg state;
always @ ( posedge clock)
begin
//apo state b kai c, to output = input se kathe peript.
assign y = ((state == b) | (state == c)) ? (x) : (~x)
case (state)
a: if (~x) state <= b; else state <= c;
b: if (~x) state <= c; else state <= d;
c: if (~x) state <= b; else state <= d;
d: if (~x) state <= c; else state <= a;
endcase
end
endmodule
```

ii)

```
module Moore_Model_b (y, x, clock);
output y;
input x, clock;
reg state;
always @ ( posedge clock)
begin
case (state)
a: if (~x) state <= b; else state <= c;
b: if (~x) state <= c; else state <= d;
c: if (~x) state <= b; else state <= d;
d: if (~x) state <= c; else state <= a;
endcase
end
//ftanontas se state b kai c, output = 1
assign y = ((state == b) | (state == c)) ? (x) : (~x)
endmodule
```

iii)

```
module updown_counter (
output reg [3: 0] A, // Counter data output
output C_out, // Output carry
input Up_down, CLK, Clear
);
assign C_out = Up_down & (A == 4'b1111);
always @ ( posedge CLK, negedge Clear)
//thewrisame to clear triggered sto '1'
if (Clear) A <= 4'b0000;
//metrhsh anw gia Up_down == 1
else if (Up_down) A <= A + 1'b1;
else A <= A - 1'b1;
endmodule
```