ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ
1η ΟΜΑΔΑ ΑΣΚΗΣΕΩΝ

ΦΟΙΤΗΤΕΣ
Αθανασίου Ιωάννης / Α.Μ.:03117041 /6ο Εξάμηνο
Καραβαγγέλης Αθανάσιος / Α.Μ.:03117022 / 6ο Εξάμηνο

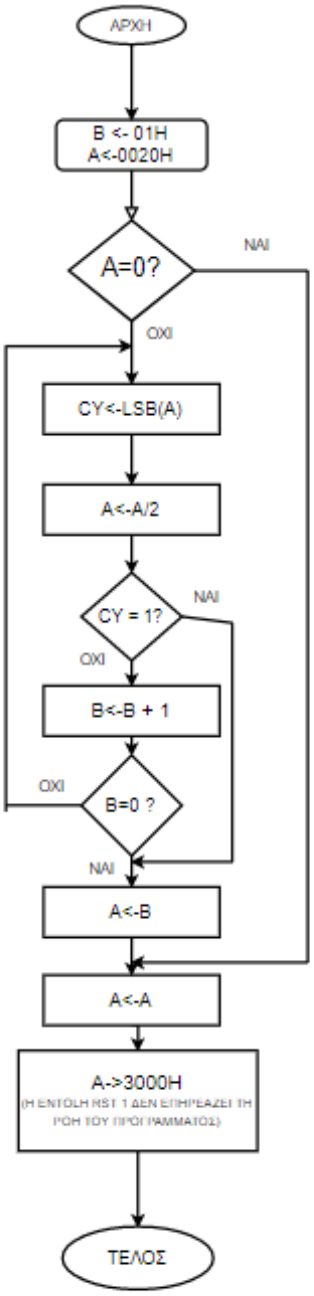
ΑΣΚΗΣΗ 1η

- Η assembly του προγράμματος που δίνεται σε γλώσσα μηχανής είναι:

```
      MVI B,01H
      LDA 2000H
      CPI 00H
      JZ ADR2
ADR3:
      RAR
      JC ADR1
      INR B
      JNZ ADR3
ADR1:
      MOV A,B
ADR2:
      CMA
      STA 3000H
      RST 1
      END
```

Προς επαλήθευση της παραπάνω assembly επισυνάπτεται στο φάκελο το αρχείο “askisi1arxiko.8085” όπου στη λειτουργία «Μνήμη και εντολές→Πρόγραμμα» φαίνεται η γλώσσα μηχανής και οι αντίστοιχες διευθύνσεις για τον παραπάνω κώδικα.

- Το διάγραμμα ροής παρατίθεται στην εικόνα στο πλάι της σελίδας.



- Για να επαναλαμβάνεται το πρόγραμμα χωρίς τέλος, αρκεί να προσθέσουμε μία ετικέτα "ARXH:" στην αρχή του κώδικα ,να αφαιρέσουμε την εντολή RST 1 και στο τέλος να εκτελούμε άλμα χωρίς συνθήκη σε αυτήν την ετικέτα (εντολή: JMP ARXH). Ο νέος κώδικας αποτελεί το αρχείο “askisi1jump.8085” του φακέλου της αναφοράς και δίνεται και παρακάτω.

```
ARXH:
      MVI B,01H
      LDA 2000H
      CPI 00H
      JZ ADR2
ADR3:
      RAR
      JC ADR1
      INR B
      JNZ ADR3
ADR1:
      MOV A,B
ADR2:
      CMA
      STA 3000H
      JMP ARXH
      ;RST 1
END
```

ΑΣΚΗΣΗ 2η

```
IN 10H ;
MVI B,10H ;
MVI D,01H ; for the step
MVI E,00H ; flag remembering direction when 1st_LSB=ON

MVI A,01H ;
CMA ;
STA 3000H ; initialization
CALL DELB ;

MASTER:
      LDA 2000H
      ANI 02H ;      apomonwsh tou 2nd_LSB
      CPI 00H
      JNZ STABLE ;   if 2nd_LSB!=0->ON go to STABLE
      LDA 2000H
      ANI 01H ;      apomonwsh tou 1st_lSB
      CPI 00H ;      if 1st_LSB!=0->ON go to POREIA_2
      JNZ POREIA_2 ;
      JMP POREIA_1 ; else if 1st_LSB->OFF go to POREIA_1

STABLE:
      MVI A,FEH ;    A<- (1)'
      STA 3000H ;    opposite logic, show
      CALL DELB ;
      JMP MASTER ;   all over again

POREIA_1:
        MVI E,00H ;
        MOV A,D ;
        RLC ;        step
        MOV D,A ;
        CMA ;
        STA 3000H ;
        CALL DELB ;
        JMP MASTER ;

POREIA_2:
     LOOP1:
           MOV A,E ;
           CPI 00H ;
           JNZ LOOP2  ;  if a!=0 go to loop2
     LOOP3:
           MVI E,00H ;
           MOV A,D ;
           CPI 80H ;
           JNC LOOP2 ; if A > 255 go to loop2
           RLC ;      else step <- <-
           MOV D,A
           CMA
           STA 3000H ; opposite logic, show
           CALL DELB ;
           JMP MASTER ;
     LOOP2:
           MOV A,D ;
           MVI E,01H ;
           CPI 01H ;
           JZ LOOP3 ; if A==1 go to loop3
           RRC ;     else step -> ->
           MOV D,A
           CMA
           STA 3000H ;
           CALL DELB ;
           JMP MASTER ;
END
```

Για την 2η άσκηση επισυνάπτεται στο αρχείο zip και το αρχείο προσομοίωσης “askisi2.8085”.

ΑΣΚΗΣΗ 3η

```
START: LDA 2000H                    ; READ INPUT
     MVI D,FFH                    ; INITIALIZE B
     CPI 63H                      ; COMPARE WITH 99 (= 63 HEX)
     JNC GREATER_THAN_100         ; GO TO WHAT HAPPENS IF INPUT>100

GREATER_THAN_100:                   ; LOOP THAT PERFORMS (INPUT MOD 100)
     JNC GREATER_THAN_100
     ADI 64H                      ; WHEN NEGATIVE, ADD 100
     JMP CALCULATE                ; NOW OUR NUMBER IS < 100


CALCULATE:
     INR D                        ; COUNT TENS
     SUI 0AH                      ; SUBSTRACT 10 UNTIL NEGATIVE
     JNC CALCULATE
     ADI 0AH                      ; ADD 10 (A NOW CONTAINS THE UNITS)
     MOV C,A                      ; STORE A IN REGISTER C
     SUB A                        ; A = 0
     ADD D                        ; A = "HOW MANY TENS WE HAVE"
     RLC                          ; MAKE THE 4 MSBs SO THAT THEY CONTAIN THE TENS
     RLC
     RLC
     RLC
     ADD C                        ; FILL WITH THE UNITS IN THE 4 LSBs
     CMA                          ; PRINT RESULT
     STA 3000H
     JMP START

     END
```

Για την 3η άσκηση επισυνάπτεται και το αρχείο προσομοίωσης “askisi2.8085”.

ΑΣΚΗΣΗ 4η

- Γενικά ισχύει:

Κόστος= Αρχικό + (Κόστος-ICs + Κόστος-κατασκευής)*Πλήθος

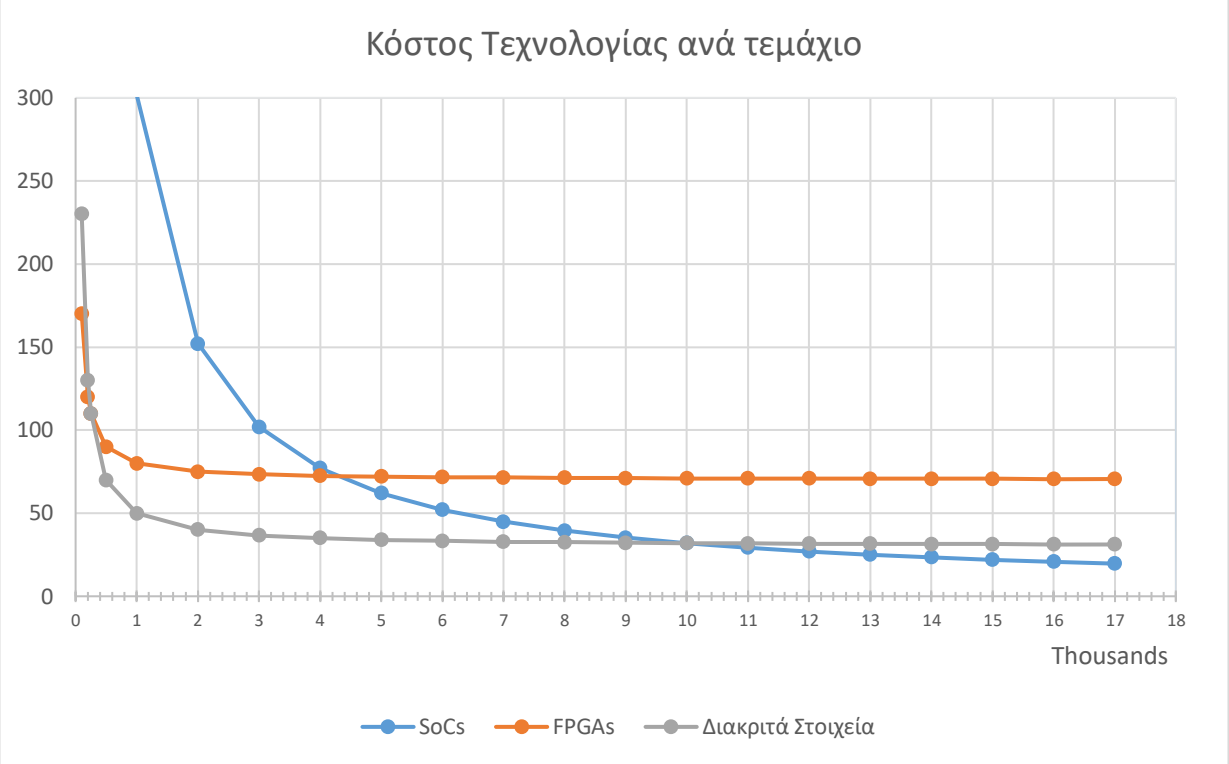
Επομένως βρίσκουμε ότι :

- $K_{SoCs}(x)= 300000 + 2*x$
- $K_{FPGAs}(x) = 10000 + 70*x$
- $K_{Διακριτά}(x)=20000 + 30*x$

Έτσι λοιπόν για το κόστος ανά τεμάχιο έχουμε:

- $\mathrm{KSoCs(x)} = \frac{300000}{x} + 2$
- $\mathrm{KFPGAs(x)} = \frac{10000}{x} + 70$
- $\mathrm{KΔιακριτά(x)} = \frac{20000}{x} + 30$

Οι συναρτήσεις Κόστους Τεχνολογίας ανά τεμάχιο αναπαρίστανται σε κοινό διάγραμμα παρακάτω:



- Ουσιαστικά παρατηρούμε ότι για έναν αρκετά μικρό αριθμό τεμαχίων από **0-110 τεμάχια** επικρατούν τα FPGAs. Το πλήθος τεμαχίων αυτό είναι μικρό εξαιτίας της υψηλής τιμή των ICs ανά τεμάχιο για τα FPGAs. Έπειτα από **500-10000** το χαμηλότερο κόστος το έχουμε με την παραγωγή διακριτών στοιχείων. Από τα **10000 τεμάχια κ έπειτα** παρατηρούμε ότι το χαμηλότερο κόστος το έχουμε για την παραγωγή SoCs. Μάλιστα, από το σημείο εκείνο και έπειτα είναι συνεχώς αυξανόμενη η διαφορά ανάμεσα στο κόστος ανά τεμάχιο SoC και το κόστος ανά τεμάχιο για διακριτά στοιχεία. Γενικά παρατηρούμε σταδιακή σύγκλιση προς το σταθερό όρο των συναρτήσεων κόστους ανά τεμάχιο δηλαδή το άθροισμα (Κόστος ICs+Κόστος Κατασκευής) που είναι 2 , 70 και 30 ευρώ για SoCs,FPGAs,Διακριτά Στοιχεία αντίστοιχα .

- Υποθέτοντας ότι δε γνωρίζουμε το κόστος των ICs για τα FPGAs η συνάρτηση κόστους τότε είναι:

$$K_{FPGAs}(x) = 10000 + (a + 10)*x$$

Ουσιαστικά ψάχνουμε την τιμή αυτή για τα ICs με την οποία το κόστος των FPGAs θα είναι χαμηλότερο από εκείνο των Διακριτών στοιχείων για κάθε τιμή του x,δηλαδή:

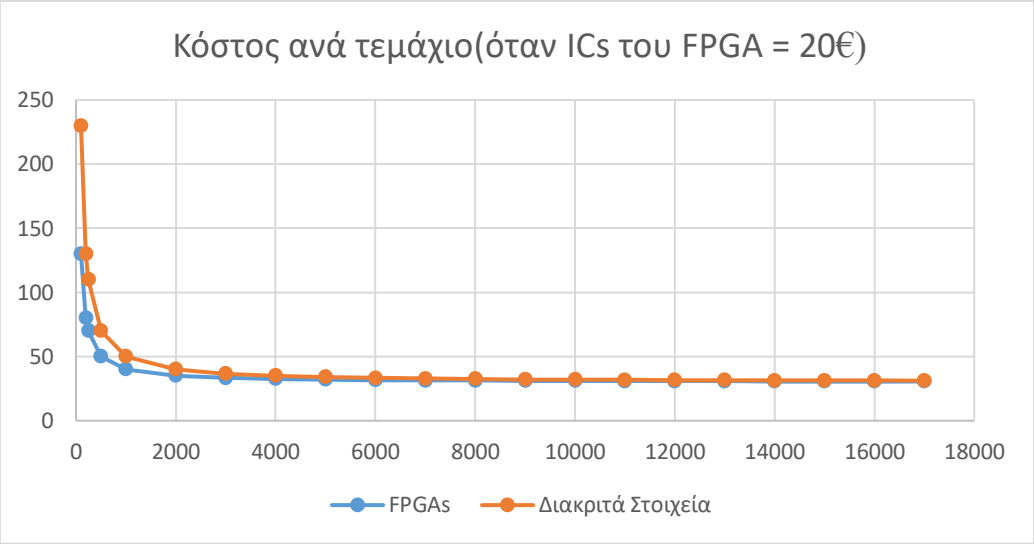
$$K_{Διακριτά}(x) = 20000 + 30*x > K_{FPGAs}(x) = 10000 + (a + 10)*x$$

Έτσι έπειτα από πράξεις καταλήγουμε στην έκφραση :

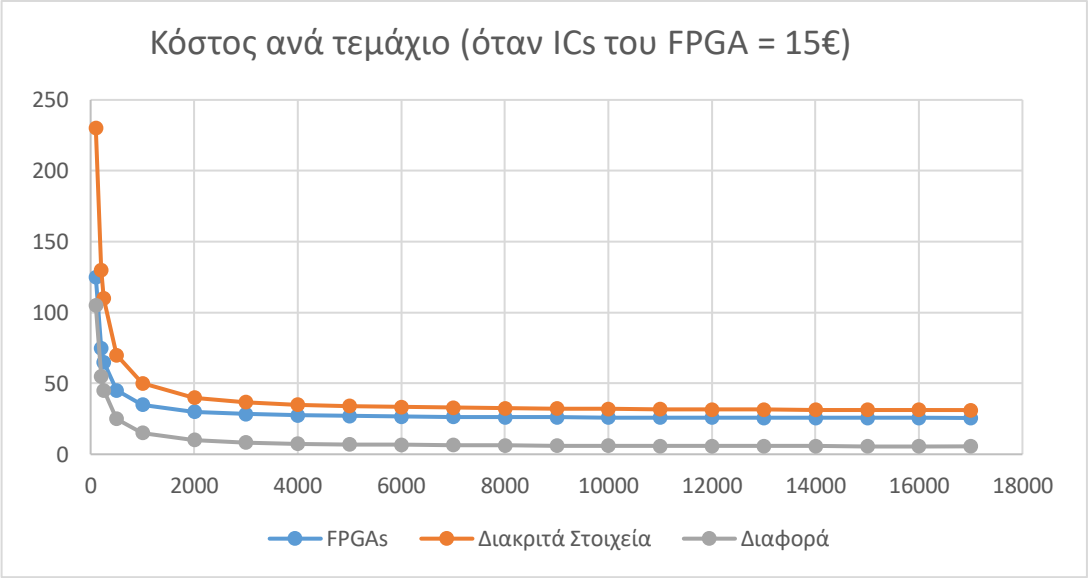
$$\mathbf{a < \frac{10000}{x} + 20}$$

Καθώς ζητάμε αυτή η συνθήκη να ισχύει για κάθε πιθανό x πρέπει να ερευνήσουμε τι συμβαίνει όταν το x πλησιάζει το +∞. Όταν το x->+∞ η τιμή του a πρέπει να είναι a ≤ 20€ .

Ωστόσο με τον τρόπο αυτό τα συνολικά κόστη παραγωγής ανά τεμάχιο για τα FPGAs και τα Διακριτά Στοιχεία θα έχουν σχεδόν μηδενική διαφορά! Επομένως , η τεχνολογία των FPGAs ουσιαστικά δε θα εξαφανίσει τα Διακριτά Στοιχεία. Η μικρή αυτή διαφορά φαίνεται και στο παρακάτω διάγραμμα και βλέπουμε ότι ασυμπτωτικά γίνεται μηδενική.



Έτσι συμπεραίνουμε ότι με μία ακόμη χαμηλότερη τιμή ,αλλά ταυτόχρονα όχι πολύ χαμηλή, για τα ICs της τάξης των 15€ θα έχουμε μία σημαντική διαφορά η οποία ασυμπτωτικά θα γίνει 5€ ανά τεμάχιο όταν μιλάμε για μεγάλες ποσότητες τεμαχίων (x->+∞).Τα παραπάνω φαίνονται στο κάτωθι διάγραμμα:



ΑΣΚΗΣΗ 5η

i) Περιγραφή σε επίπεδο πυλών

α) F1= A(CD+B) + BC'D'

```
module circuit_with_F1(F, A, B, C, D);
output F;
input A, B, C, D;
wire C_not, D_not, w1, w2, w3, w4;
not
        G1(D_not, D),
        G2(C_not, C);
and
        G3(w1, C, D),
        G5(w4, B, C_not, D_not),
        G6 (w3, A, w2);
or
        G4(w2, B, w1),
        G7(F, w3, w4);
endmodule
```

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

```
primitive F2(x, A, B, C, D);
output x;
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;
endtable
endprimitive

module circuit_with_F2 (x, A, B, C, D);
output x;
input A, B, C, D;
F2(x, A, B, C, D);
endmodule
```

γ) F3(A, B, C, D) = ABC + (A+B)CD + (B+CD)E

```
module circuit_F3 (F, A, B, C, D, E);
output F;
input A, B, C, D, E;
wire w1, w2, w3, w4, w5, w6;
and
        G1(w1, A, B, C),
        G3(w3, w2, C, D),
         G4(w4, C, D),
        G6(w6, w5, E);
or
        G2(w2, A, B),
        G5(w5, w4, B),
        G7(F, w1, w3, w6);
endmodule
```

δ) F4 = A(BC+D+E)+CDE

```
module circuit_with_F4 (F, A, B, C, D, E);
output F;
input A, B, C, D, E;
wire w1, w2, w3, w4;
and
        G1(w1, B, C),
        G4(w4, C, D, E),
        G3(w3, w2, A);
or
        G2(w2, w1, D, E,
        G5(F, w3, w4);
endmodule
```

ii) Περιγραφή σε επίπεδο ροής δεδομένων

α) F1=A(CD+B) + BC'D'

```
module circuit_with_F1(F, A, B, C, D);
output F;
input A, B, C, D;
assign F=A&(C&D|B)|(B&~C&~D);
endmodule
```

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

Με χρήση του πίνακα Karnaugh βρίσκουμε την έκφραση της F2 σε άθροισμα ελαχιστόρων

F2(A, B, C, D)= B'D' + CD + A'BD + AC

'Εχουμε επομένως:

```
module circuit_with_F2 (F, A, B, C, D);
output F;
inupt A, B, C, D;
assign F=(~B&~D)|(C&D)|(~A&B&D)|(A&C);
endmodule
```

γ) F3(A, B, C, D)= ABC +(A+B)CD + (B+CD)E

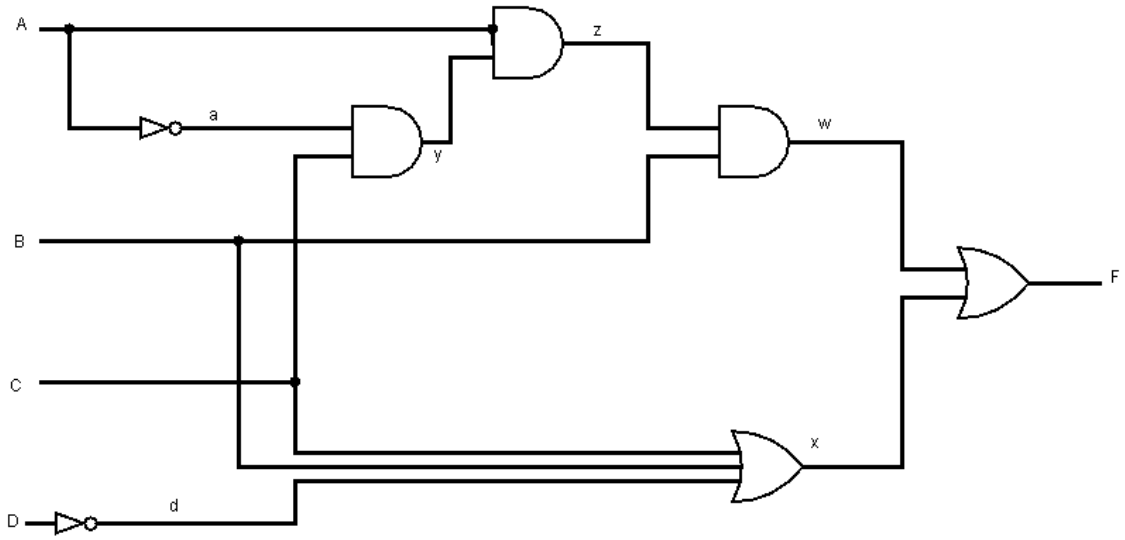
```
module circuit_with_F3(F, A, B, C, D, E);
output F;
input A, B, C, D, E;
assign F=(A&B&C)|((A|B)&C&D)|((B|C&D)&E);
endmodule
```

δ) F4=A(BC+D+E)+CDE

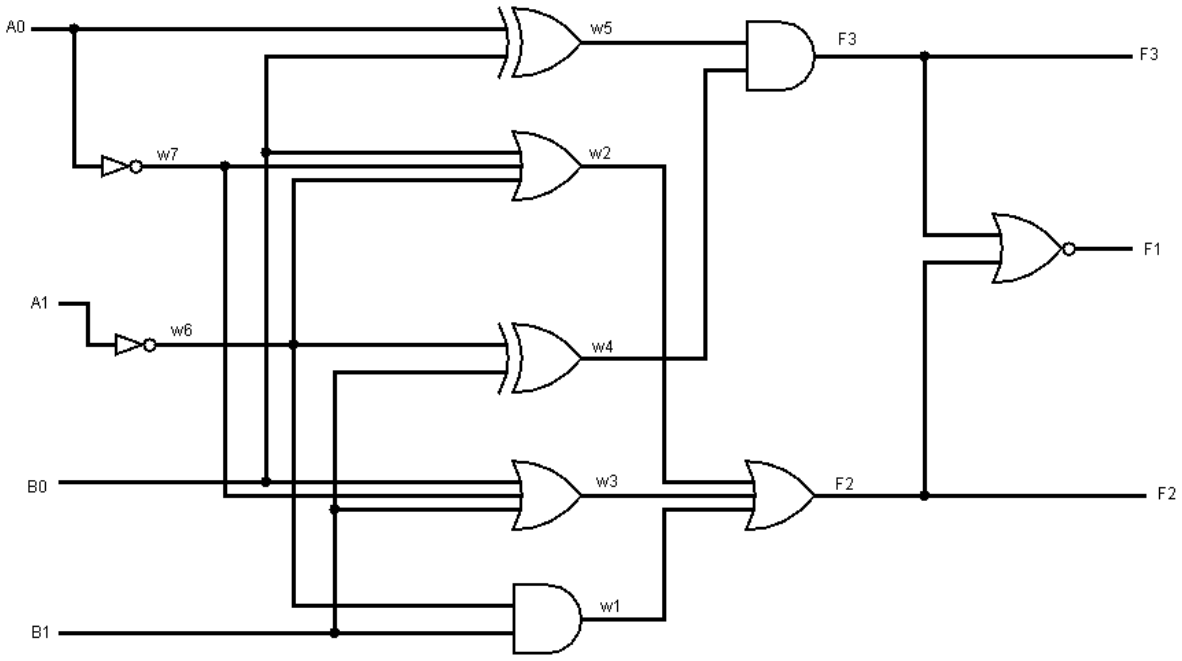
```
module circuit_with_F4(F,A , B, C, D, E);
output F;
input A, B, C, D, E;
assign F=A&((B&C)|D|E)|(C&D&E);
endmodule
```

ΑΣΚΗΣΗ 6η

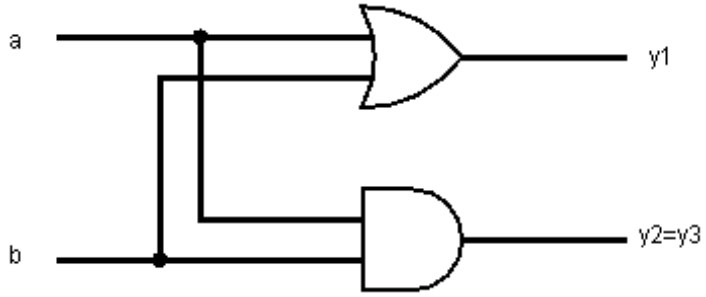
i) (a)



(b)



(c)



ii)

```
module half_adder (output S,C ,input x,y) ; //Πρώτα θα ορίσουμε τον Half Adder (HA) xor(S
,x ,y );
and(C ,x ,y );
endmodule

module full_adder(output S,C , input x,y,z) ; //Έπειτα θα ορίσουμε το Full Adder (FA)
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_suber(output [3:0] Sum , output C4 , input [3:0]A , input [3:0] B,
                                                                input M); // Τώρα το σύνθετο
wire c1,c2,c3,B0,B1,B2,B3;
xor
        XOR0(B0,B[0],M),
        XOR1(B1,B[1],M),
        XOR2(B2,B[2],M),
        XOR3(B3,B[3],M);
full_adder
        FA0(Sum[0],c1,A[0],B0,M),
        FA1(Sum[1],c2,A[1],B[1],M),
        FA2(Sum[2],c2,A[2],B[2],M),
        FA3(Sum[3],c3,A[3],B[3],M);
endmodule
```

iii)

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

ΑΣΚΗΣΗ 7η

i)

Mealy FSM

```
module Mealy_Model(y_out,x_in,clock,reset);
output reg y_out;
input x_in, 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 <= next_state
always@(state, x_in)
case (state)
a: if (x_in) next_state=c else next_state=b;
b: if (x_in) next_state=d else next_state=c;
c: if (x_in) next_state=d else next_state=b;
d: if (x_in) next_state=a else next_state=c;
endcase
always@(state)
case (state)
    a,d: y_out=~x_in;
    b,c: assign y_out=x_in;
endcase
endmodule
```

ii)

Moore FSM

```
module Moore_Model(x_in,y_out,clock,reset);
output y_out;
input x_in, 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 <= next_state
always@(state, x_in)
case (state)
    a: if (x_in) next_state=c else next_state=b;
    b: if (x_in) next_state=d else next_state=c;
    c: if (x_in) next_state=d else next_state=b;
    d: if (x_in) next_state=a else next_state=c;
```

```
endcase
always@(state)
case (state)
      a,d: assign y_out=0;
      b,c: assign y_out=1;
endcase
endmodule
```

iii)

```
module 4_bit_up_down_counter (clock, clear, x_in ,y_out )

input clock, clear ,x_in ;
output [3:0] y_out ;
reg [3:0] state ;
parameter s0=4'b0000 ,s1=4'b0001 ,s2=4'b0010 ,s3=4'b0011 ,        //16 possible states
            s4=4'b0100, s5=4'b0101 ,s6=4'b0110 ,s7=4'b0111 ,
            s8=4'b1000 ,s9=4'b1001 ,s10=4b'1010 ,s11=4b'1011 ,
            s12=4'b1100 ,s13=4'b1101 ,s14=4b'1110 ,s15=4b'1111 ;
always@(posedge clock, negedge clear)
if (clear==0) state <= S0;
else
always@(state ,x_in)
      case (state)
            s0: if(x_in) state<=s1 else state <= s15;
            s1: if(x_in) state<=s2 else state <= s0;
            s2: if(x_in) state<= s3 else state <= s1;
            s3: if(x_in) state<= s4 else state <= s2;
            s4: if(x_in) state<=s5 else state <= s3;
            s5: if(x_in) state<=s6 else state <= s4;
            s6: if(x_in) state<=s7 else state <= s5;
            s7: if(x_in) state<=s8 else state <= s6;
            s8: if(x_in) state<=s9 else state <= s7;
            s9: if(x_in) state<=s10 else state <= s8;
            s10: if(x_in) state<=s11 else state <= s9;
            s11: if(x_in) state<=s12 else state <= s10;
            s12: if(x_in) state<=s13 else state <= s11;
            s13: if(x_in) state<=s14 else state <= s12;
            s14: if(x_in) state<=s15 else state <= s13;
            s15: if(x_in) state<=s0 else state <= s14;
      endcase
assign y_out = state ; //output ← state
endmodule
```