14/4/21

1η Ομάδα Ασκήσεων

Συστήματα Μικροϋπολογιστών

Βαρδάκης Χριστόφορος el18883
Λυμπεράκης Γεώργιος el18881

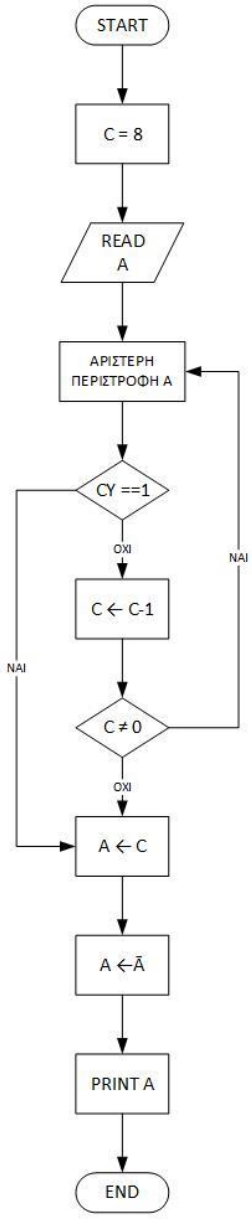
1η ΑΣΚΗΣΗ

Στην άσκηση αυτή καλούμαστε να ανακτήσουμε τις εντολές assembly του δοθέντος κομματιού που είναι σε γλώσσα μηχανής κα για να το πετύχουμε αυτό θα ακολουθήσουμε τις υποδείξεις που δίνονται .Οπότε διαβάζουμε το Opcode και πηγαίνοντας στον πίνακα του παρατήματος αντιστοιχούμε εντολή με Opcode. Θεωρούμε ότι $PC = 0800H$.

Επομένως , ο κώδικας που προκύπτει είναι ο παρακάτω.

```
START:

      MVI C,08H    ; C <- 8
      LDA 2000H    ; Read Switches

LABEL1:

      RAL          ; Make a left rotation
      JC LABEL2    ; If CY == 1 jump to LABEL 2 else continue
      DCR C        ; C <- C - 1
      JNZ LABEL1   ; If Z != 0 jump to LABEL 1 else continue

LABEL2:

      MOV A,C      ; A <- C
      CMA          ; A <- A'
      STA 3000H    ; Move A Data in output port
      RST 1        ; Restart (Interrupt)
```

Οπότε , το διάγραμμα ροής που αντιστοιχεί στον παραπάνω κώδικα είναι το εξής.



Το δοθέν πρόγραμμα εκτελεί την εξής λειτουργία : Διαβάζει τον δυαδικό αριθμό που αναπαρίσταται από τους διακόπτες εισόδου και εκτυπώνει στις λυχνίες εξόδου σε δυαδική μορφή τη θέση του πρώτου (ξεκινώντας από τα δεξιά) διακόπτη εισόδου που είναι ΟΝ.

Για να μπορέσουμε να έχουμε έναν ατέρμων βρόχο στο τέλος προσθέτουμε την εντολή άλματος ***JMP START***.

2η ΑΣΚΗΣΗ

Ο τρόπος λειτουργίας είναι ο εξής:

- Το πρόγραμμα δέχετε την έισοδο των dip switches και την αποθηκέυει.
- Κάνει δεξί shift στην είσοδο για να αφήσει ως κρατούμενο το LSB.
- Αν το LSB είναι 1 τότε το πρόγραμμα ξαναξεκινά δημιουργόντας endless loop.
- Αν όχι το πρόγραμμα κάνει αριστερό shift στην αποθηκευμένη είσοδο αφήνοντας κρατούμενο το MSB
- Αν το MSB είναι 1 μεταφέρει τον PB στον κώδικα της δεξιάς περιστροφής. Αλλιώς Συνεχίζει με την αριστερή.
- Στην αριστερή κίνηση:
    - Αρχικά διαβάζει την τιμή του τρέχοντος LED
    - ‘Επειτα την αντιστρέφει και την εμφανίζει.
    - Αφού την επαναφέρει στην αρχική της κατάσταση της εφαρμόζει αριστερό shift για να πάρει την επόμενη τιμή.
    - Τέλος επαναλαμβάνει το πρόγραμμα από την αρχή.
- Για την δεξιά κίνηση:
    - Εφαρμόζει τα ίδια βήματα με την αριστερή αλλάζοντας το αριστερό shift με δεξί.

Η υλοποίηση του προγράμματος φαίνεται στον παρακάτω κώδικα.

```
      IN 10H
      LXI B,01F4H       ;Delay 500ms = 0x1F4
      MVI E,01H         ;Set Initial LED to LSB

START:

      LDA 2000H         ;load inout to accumulator
      MOV D,A           ;store input to D
      RRC               ;Shift Right (to carry LSB)
      JC START          ;while LSB switch is on endless loop
      CALL DELB         ;Delay 0,5s
      MOV A,D           ;load D to accumulator
      RLC               ;Shift Left (to carry MSB)
      JC GORIGHT        ;if MSB switch is on use right circle

GOLEFT:           ;Left Circle

      MOV A,E           ;load LED to accumulator
      CMA               ;reverse logic
      STA 3000H         ;move acumulator to output
      CMA               ;reverse logic
      RLC               ;shift left
      MOV E,A           ;store next LED to E
      JMP START

GORIGHT:          ;Right Circle {Similar to Left}

      MOV A,E
      CMA
      STA 3000H
      CMA
      RRC
      MOV E,A
      JMP START

      END
```

3η ΑΣΚΗΣΗ

Στην τρέχουσα άσκηση καλούμαστε να επεκτείνουμε ένα ήδη υπάρχων πρόγραμμα, το οποίο δοθέντος ενός δυαδικός 8Bit αριθμός (μικρότερο του 100) να εμφανίζει στα αριστερά 4 LED εξόδου το πλήθος των δεκάδων ενώ στα υπόλοιπα 4 το πλήθος των μονάδων με τις οποίες αναπαρίσταστε σε δεκαδική μορφή ο αριθμός.

Επομένως ,εμείς καλούμαστε πέρα από τη βασική λειτουργία του προγράμματος να προσθέσουμε τις παρακάτω δύο επεκτάσεις (έστω N ο αριθμός εισόδου) .

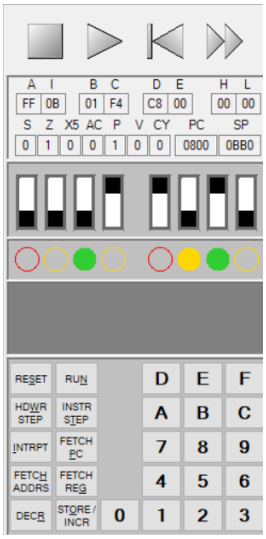
I. $Αν\ \mathbf{99 < N < 200} \rightarrow Συνεχώμενο\ ανοιγόκλειμα\ των\ 4\ LSB\ των\ LED$.
II. $Αν\ \mathbf{N > 199} \rightarrow Συνεχώμενο\ ανοιγόκλειμα\ των\ 4\ MSB\ των\ LED$.

Οπότε, ο κώδικας assembly που χρησιμοποιήθηκε είναι ο παρακάτω και συνοδεύεται με συνοπτικά επεξηγηματικά σχόλια .

```
READ_INPUT:

      LDA 2000H     ;load input in A
      MOV D,A      ;D <- A ,make a copy of input

      CPI 63H      ;Compare the number with 99
      JNC BIGGER_99 ;If it's bigger then it is a special case

      MVI B,FFH    ;FFH has Complement of 2 -> -1

DECA:

      INR B        ;B <- B + 1
      SUI 0AH      ;A <- A - 10
      JNC DECA     ;Do it again if the result is positive

      ADI 0AH      ;Correct the negative result by adding 10
      MOV C,A      ;Store for now the result in reg C

      MOV A,B      ;Bring in Accumulator the B

      RLC          ;Rotate left accumulator
      RLC          ;for
      RLC          ;four
      RLC          ;times
```

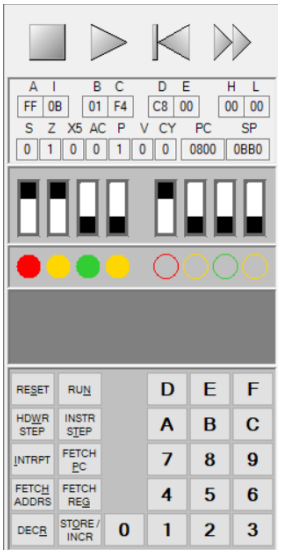
```
      ADD C             ;A <- A + C
      CMA               ;A <- A' (complementary)
      LXI B,01F4H       ;put delay for 0.5 second

      STA 3000H         ;save in output port the result
      LXI B,01F4H       ;put delay for 0.5 second

      JMP READ_INPUT    ;Return to read the next input


BIGGER_99:

      CPI C8H           ;Check if input > 200
      JNC BIGGER_199    ;If it's true then go to print it properly

      LXI B,01F4H       ;Put delay for 0.5 second

      JMP LOWER_LED     ;Go to print the result

BIGGER_199:

      LXI B,01F4H       ;put delay for 0.5 second
      JMP UPPER_LED     ;Go to print the result



UPPER_LED:

      MVI A,0FH         ;Put the mask in accumulator
      STA 3000H         ;Light the MSB LEDS
      LXI B,01F4H       ;Put delay for 0.5 second
      MVI A,FFH         ;Put a new mask in A to close the LEDS
      STA 3000H         ;Close the MSB LEDS
      LXI B,01F4H       ;Put delay for 0.5 second
      LDA 2000H         ;Read the Input port again
      CMP D             ;Check if it has changed
      JNZ READ_INPUT    ;If it is true the return
      JMP UPPER_LED     ;Else continue to open-close the LEDS


LOWER_LED:

      MVI A,F0H         ;Put the mask in accumulator
      STA 3000H         ;Light the LSB LEDS
      LXI B,01F4H       ;Put delay for 0.5 second
      MVI A,FFH         ;Put a new mask in A to close the LEDS
      STA 3000H         ;Close the MSB LEDS
      LXI B,01F4H       ;Put delay for 0.5 second
      LDA 2000H         ;Read the Input port again
      CMP D             ;Check if it has changed
      JNZ READ_INPUT    ;If it is true the return
      JMP LOWER_LED     ;Else continue to open-close the LEDS

END
```

Παρακάτω ακολουθούν 3 ενδεικτικά παραδείγματα (testcases) για τον έλεγχο του κώδικα .



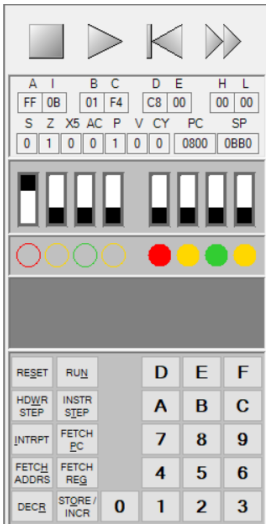
Είσοδος : 0001 1010 σε δεκαδική μορφή ο αριθμός 26 .

Έξοδος : 0010 0110 άρα 2 δεκάδες & 6 μονάδες οπότε ο αριθμός 26.



Είσοδος : 1100 1000 σε δεκαδική μορφή ο αριθμός 200 (μεγαλύτερος 199) .

Έξοδος : 1111 0000 αναγνωρίζεται επιτυχώς.



Είσοδος : 1000 0000 σε δεκαδική μορφή ο αριθμός 128 (μεγαλύτερος 99 και μικρότερος του 200) .

Έξοδος : 0000 1111 αναγνωρίζεται επιτυχώς.

Σχόλιο: Στο επίπεδο προσομοίωσης τα LED αναβόσβηναν (όλα ανοιχτά όλα κλειστά και ξανά) συνεχόμενα έως ότου διαβαστεί διαφορετική είσοδος . Στη φωτογραφία φαίνεται η στιγμή που είναι όλα αναμμένα.

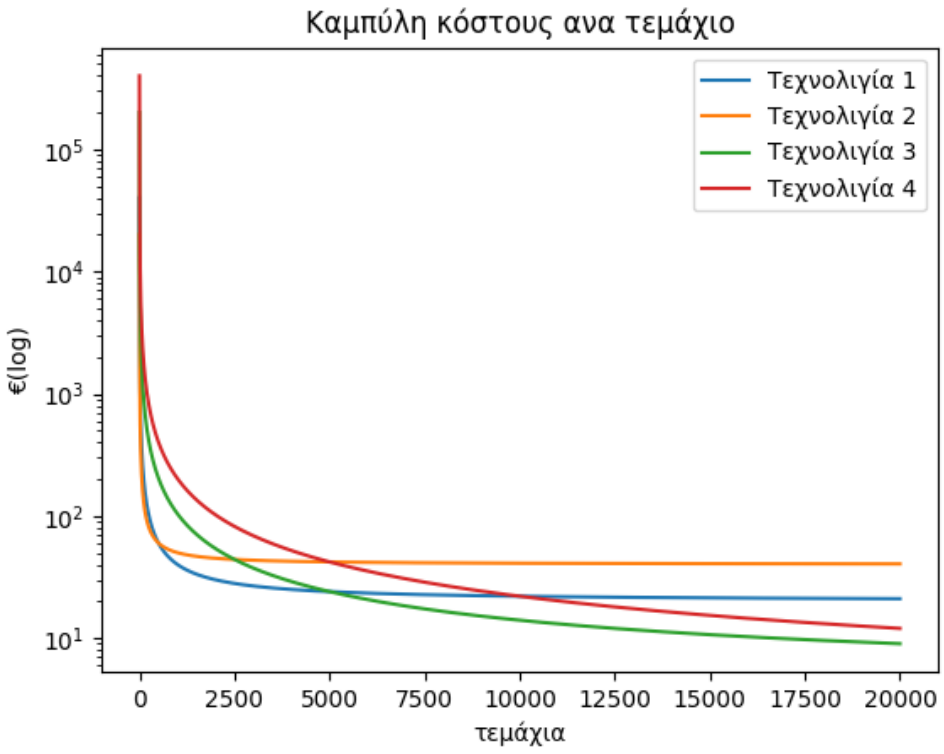
4η ΑΣΚΗΣΗ

Αρχικά υπολογίζουμε τις εξισώσεις κόστους και κόστους ανά τεμάχιο ως προς τον αριθμό των τεμαχίων.
Γενικά:

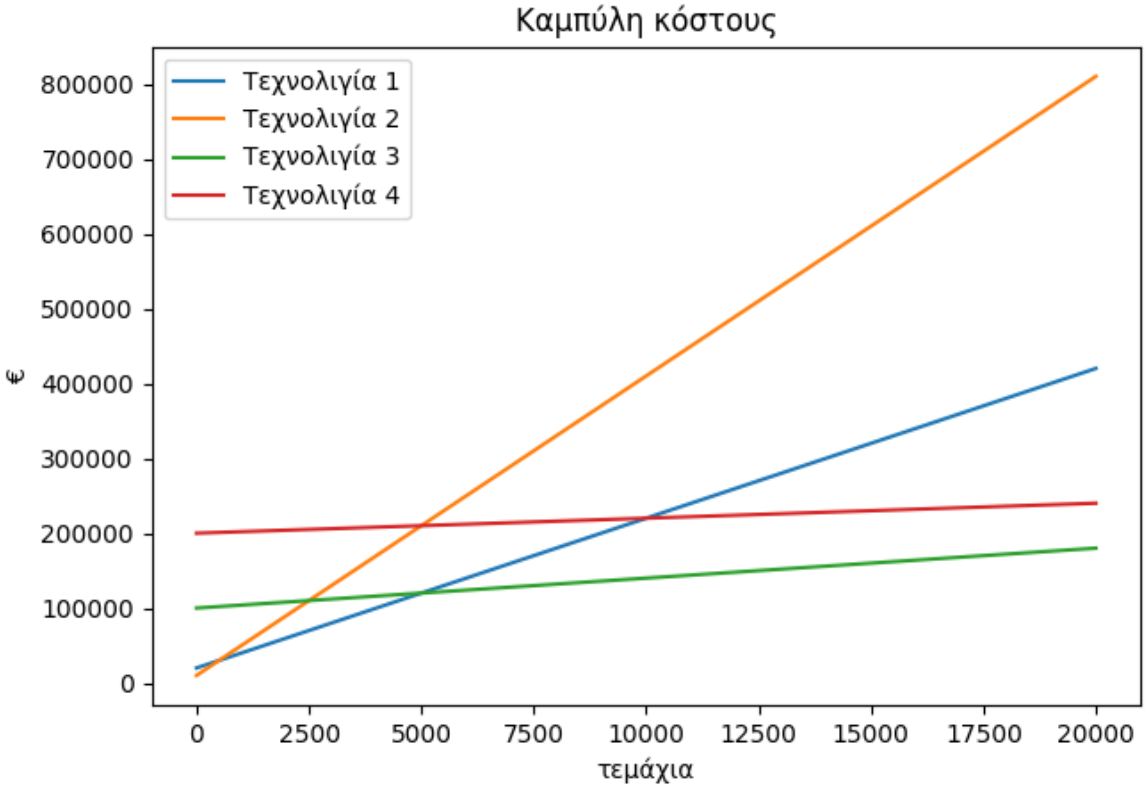
Για αρχικό κόστος σχεδίασης = I_0, κόστος I.C./ πλακέτα = i, κόστος συναρμολόγησης = c
Καμπύλη κόστους: $K_{(x)} = I_0 + (i+c)x$
Καμπύλη κόστους / τεμάχιο: $f_{(x)} = I_0/x + (i+c)$

Τεχνολογία 1:
Καμπύλη κόστους: 20x + 20.000
Καμπύλη κόστους / τεμάχιο: 20.000/x + 20
Τεχνολογία 2:
Καμπύλη κόστους: 40x + 10.000
Καμπύλη κόστους / τεμάχιο: 10.000/x +40
Τεχνολογία 3:
Καμπύλη κόστους: 4x + 100.000
Καμπύλη κόστους / τεμάχιο: 100.000/x + 4
Τεχνολογία 4:
Καμπύλη κόστους: 2x + 200.000
Καμπύλη κόστους / τεμάχιο: 200.000/x + 2

‘Ετσι έχουμε τις καμπύλες κόστους/τεμάχιο σε κοινό διάγραμμα (λογαριθμικής κλίμακας) :



Από τις καμπύλες κόστους, και συγκεκριμένα τα σημεία τομής τους, μπορούμε να υπολογίσουμε τις περιοχές όπου η κάθε τεχνολογία είναι συμφερότερη



Περιοχή 1: Αρχικά παρατηρούμε ότι η καμπύλη που βρισκετε χαμηλότερα είναι αυτή της τεχνολογίας 2 , και συγκεκριμένα από τα 0 τεμάχια μέχρι το σημείο όπου

$$K_1(x) = K_2(x) \rightarrow 20x + 20000 = 40x + 10000 \rightarrow x = 500$$

Περιοχή 2: Τώρα χαμηλότερα βρίσκετε η τεχνολογία 1, στη περιοχή από 500 τεμάχια έως το σημείο x όπου

$$K_1(x) = K_3(x) \rightarrow 20x + 20000 = 4x + 100000 \rightarrow x = 5000$$

Περιοχή 3: Τώρα χαμηλότερα βρίσκετε η τεχνολογία 3, στη περιοχή από 5.000 τεμάχια έως το σημείο x όπου

$$K_3(x) = K_4(x) \rightarrow 4x + 100000 = 2x + 200000 \rightarrow x = 50000$$

Περιοχή 3: Για οποιαδήποτε ποσότητα μεγαλύτερη των 50.000 τεμαχίων η συμφερότερη τεχνολογία είναι η τεχνολογία 4

Για να εξαφανιστεί πρέπει για κάθε αριθμό τεμαχίων να ισχύει

$$K'_2(x) < K_1(x) \rightarrow (10 + i')x + 10000 < 20x + 20000 \rightarrow (i' - 10)x < 10000$$

Έπειδή το x εκφράζει αριθμό τεμαχίων είναι γνήσια θετικό, άρα για να ισχύει για κάθε x πρέπει

$$i' - 10 < 0 \rightarrow i' < 10$$

Άρα για κόστος μικρότερο των 10€ η τεχνολογία 2 εξαφανίζει την επιλογή της τεχνολογίας 1.

5η ΑΣΚΗΣΗ

i. Η πρώτη λογική συνάρτηση είναι η παρακάτω

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

Η περιγραφή της σε επίπεδο πυλών με τη χρήση της Verilog (HDL)

```
module Ex_5a_F1 (A,B,C,D,F1);
      output F1;
      input A,B,C,D;

      wire Bnot,Cnot,w1,w2,w3,w4;

      not (Bnot, B);
      not (Cnot, C);

      and  (w1, B,C);
      or   (w2,w1,B);
      and  (w3,w2,A);

      and  (w4,Bnot,Cnot,D);

      or   (F1, w4, w5);

end module
```

Η επόμενη συνάρτηση έχει τύπο

$$F2 = \sum(0,2,3,5,7,9,10,11,13,14)$$

Αποτελεί κύκλωμα ορισμένο από τον χρήστη (primitive), μιας και η υλοποίηση της χρειάζεται πίνακα αληθείας.

Η περιγραφή της σε επίπεδο πυλών με τη χρήση της Verilog (HDL)

```
primitive Ex_5a_F2 (F2, A, B, C, D);
    output F2;
    input A, B, C, D;

    table
    A B C D: F2;
    0 0 0 0: 1;  //0
    0 0 0 1: 0;  //1
    0 0 1 0: 1;  //2
    0 0 1 1: 1;  //3
    0 1 0 0: 0;  //4
    0 1 0 1: 1;  //5
    0 1 1 0: 0;  //6
    0 1 1 1: 1;  //7
    1 0 0 0: 0;  //8
    1 0 0 1: 1;  //9
    1 0 1 0: 1;  //10
    1 0 1 1: 1;  //11
    1 1 0 0: 0;  //12
    1 1 0 1: 1;  //13
    1 1 1 0: 1;  //14
    1 1 1 1: 0;  //15
    endtable
endprimitive
```

Η τρίτη συνάρτηση έχει τον παρακάτω κλειστό τύπο

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

Οπότε , έχουμε το παρακάτω κομμάτι κώδικα

```
module Ex_5a_F3 (A,B,C,D,E,F3);
      output F3;
      input A,B,C,D,E;

      wire w1,w2,w3,w4;

      and (w1, B, C);
      or  (w2, A, w1);
      and (w3, w2 ,D);

      and (w4, A,B,C);

      or  (F3, w3 ,w4);

end module
```

Τέλος η συνάρτηση με κλειστό τύπο

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

Έχει την παρακάτω υλοποίηση:

```
module Ex_5a_F4(F4, A, B, C, D, E);
    output F4;
    input A, B, C, D;
    wire w1 , w2, w3;

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

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

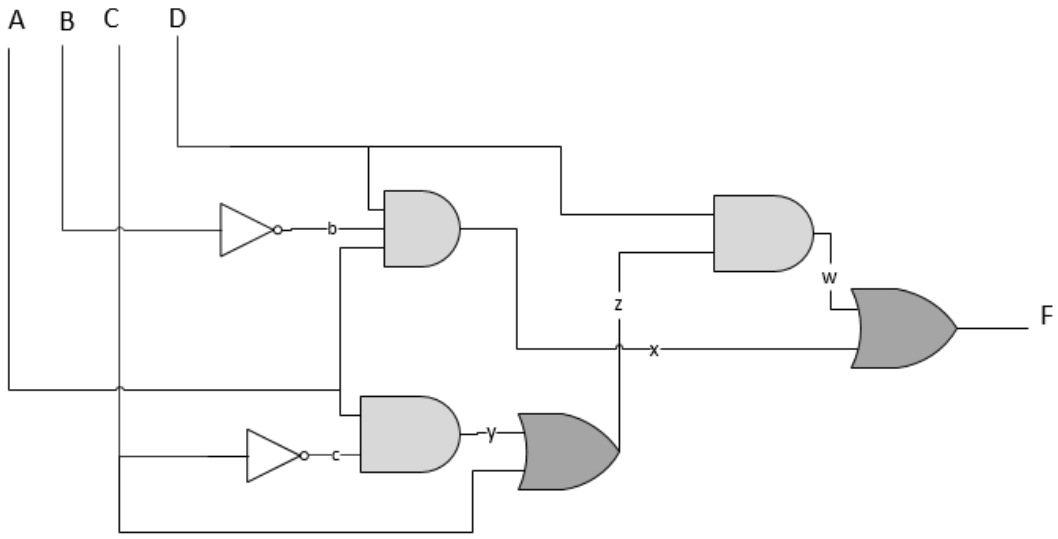
ii. Για τις ίδιες συναρτήσεις ο κώδικας Verilog που απαιτήθηκε για να υλοποιηθούν με μοντελοποίηση ροής δεδομένων είναι ο παρακάτω:

```
//F1
module Ex_5b_F1 (A,B,C,D,F1);

      output F1;
      input A,B,C,D;

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

end module

//F2
primitive Ex_5b_F2(F2, A, B, C, D);
    output F2;
    input A, B, C, D;
    F2 = (~A & ~B & ~C & ~D)|(~A & ~B & C & ~D)|    // 0 or 2
         (~A & ~B & C & D)|(~A & B & ~C & D)|      // 3 or 5
         (~A & B & C & D)|(A & ~B & ~C & D)|       // 7 or 9
         (A & ~B & C & ~D)|(A & ~B & C & D)|       // 10 or 11
         (A & B & ~C & D)|(A & B & C & ~D)         // 13 or 14

endprimitive

///F3
module Ex_5b_F3 (A,B,C,D,E,F3);

      output F3;
      input A,B,C,D,E;

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

end module

//F4
module Ex_5b_F4(F4, A, B, C, D, E);
    output F4;
    input A, B, C, D;

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

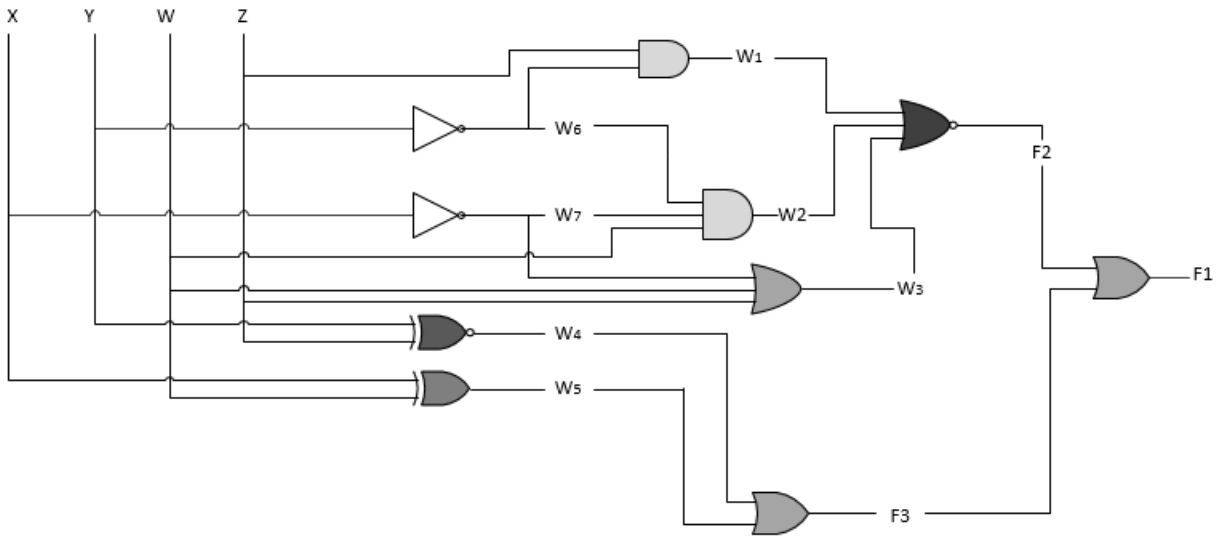
6η ΑΣΚΗΣΗ

i.

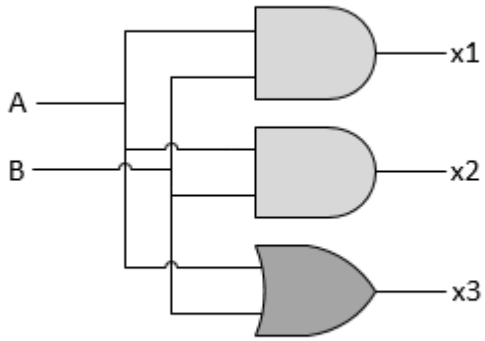
a. Με βάσει την περιγραφή σε Verilog το λογικό διάγραμμα που προκύπτει είναι το κάτωθι:



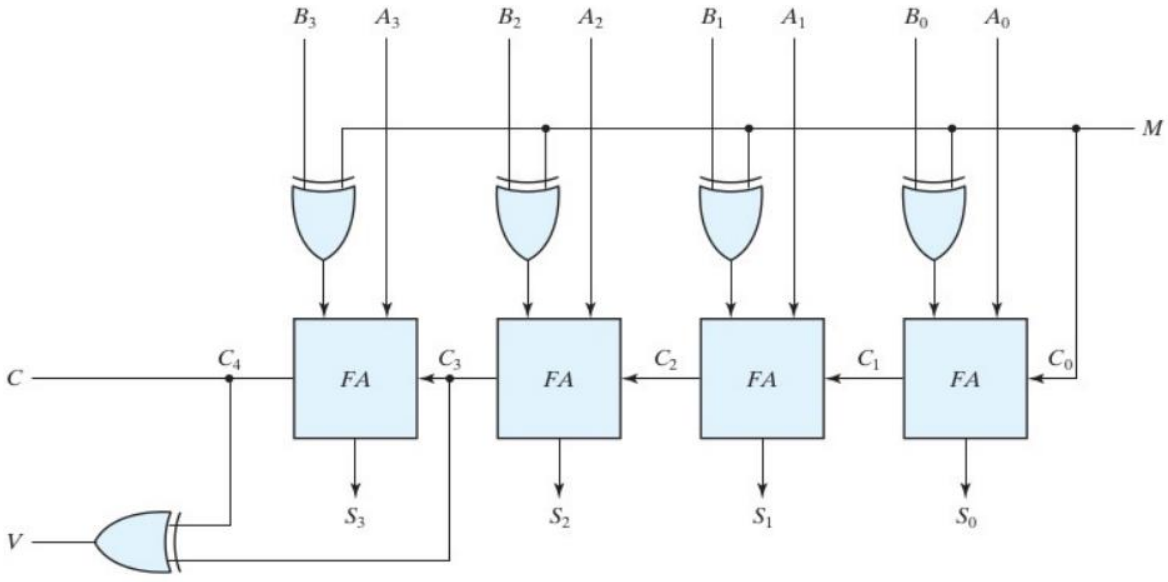
b. Αντίστοιχα, για την επόμενη περιγραφή προκύπτει το παρακάτω



c. Όμοια και για την τελευταία :



ii. Ο αθροιστής – αφαιρέτης υλοποίειτε όπως και ο απλός αθροιστής με τη χρήση 4 full adders, οι οιποίοι όμως έχουν την δευτερη έισοδο και την είσοδο ελέγχου συνδεδεμένες σε μια πύλη xor, όπως φαίνετε στο σχήμα



Έτσι προκύπτει η παρακάτω περιγραφή σε Verilog:

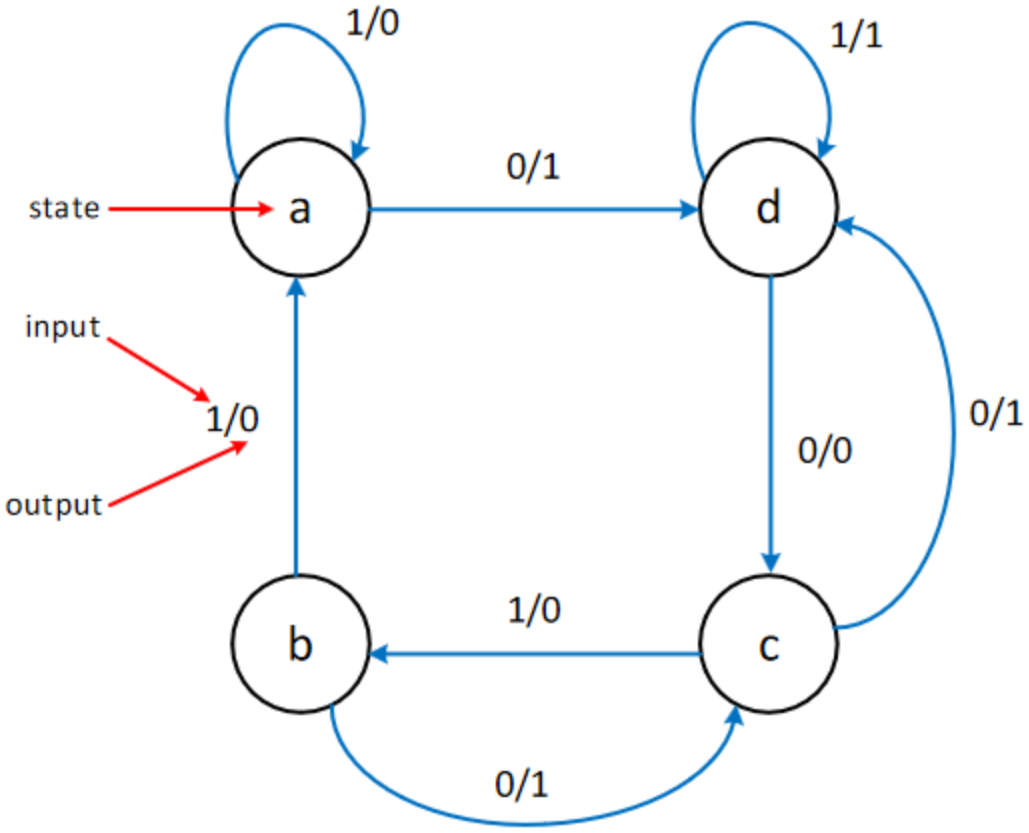
```
module half_adder (S, C, x, y);

	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, C1, z);
	or G1(C, C2, C1);

endmodule

module _4_bit_adder(S, C, V, A, B, M);

	output [3:0] S;
	output C, V;
	input [3:0] A, B;
	input M;
	wire C1, C2, C3;//Itermediate Carriers
	wire w0, w1, w2, w3;

//xor gates for subtraction
	xor G1(w0, B[0], M);
	xor G2(w1, B[1], M);
	xor G3(w2, B[2], M);
	xor G4(w3, B[3], M);
//chain of full adders
	full_adder FA0 (S[0], C1, w0, A[0], M);
	full_adder FA1 (S[1], C2, w1, A[1], C1);
	full_adder FA2 (S[2], C3, w2, A[2], C2);
	full_adder FA3 (S[3], C, w3, A[3], C3);
	xor G5(V, C, C3);

endmodule
```

iii. Με τη χρήση του τελεστή υπό συνθήκη (? :) η περιγραφή σε Verilog γίνετε:

```
module _4_bit_adder(S,C, V, A, B, M);

output [3:0] S;
output Cout;
input [3:0] A, B;
input M;
assign {Cout, S} = M ? A - B : A + B;

endmodule
```

7η ΑΣΚΗΣΗ

i. Στο τρέχων ερώτημα επικεντρωνόμαστε σε ένα Mealy Finite State Machine όπου με βάσει αυτό θα δημιουργήσουμε ένα μοντέλο Verilog του οποίου η έξοδος θα είναι συνδυασμός του σήματος εισόδου και της τρέχουσας κατάστασης που βρισκόμαστε.



Αρχικά , παρουσιάζομαι το πίνακα καταστάσεων που εξαντλούν όλες τις περιπτώσεις που μπορεί να βρεθεί η έξοδος και ποια θα είναι η επόμενη κατάσταση στο διάγραμμα

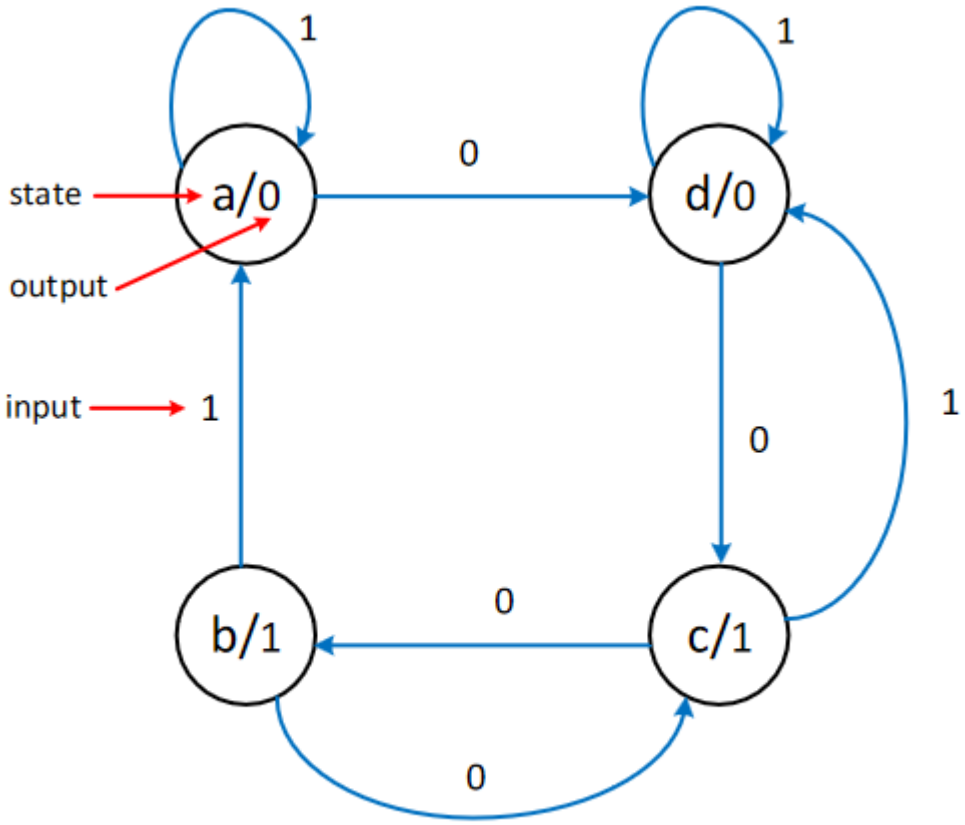
καταστάσεων συναρτήσει της τωρινής κατάστασης που βρίσκεται το αυτόματο και της εισόδου που θα λάβει ώστε να συνεχίσει.

α/α	Τωρινή Κατάσταση		Είσοδος	Μελλοντική Κατάσταση		Έξοδος
	A_n	B_n	X	A_{n+1}	B_{n+1}	Y
0	0	0	0	1	1	1
1	0	0	1	0	0	0
2	0	1	0	1	0	1
3	0	1	1	0	0	0
4	1	0	0	1	1	1
5	1	0	1	1	0	0
6	1	1	0	1	0	0
7	1	1	1	1	1	1

Επομένως, με βάσει όλα αυτά δημιουργήσαμε το παρακάτω μοντέλο Verilog.

```
module Mealy_FSM (x,y,Reset,Clock);

      output reg y;
      input x;

      reg[1:0] Current_State,Next_State;
      parameter a = 2'd0 , b = 2'd1 , c = 2'd2 , d = 2'd3 ;

      always @ ( posedge Clock, negedge Reset)

            if( Reset == 0) Next_State <= a;
            else Current_State <= Next_State;

       always @ (x , Current_State)
             case (Current_State)

             a : if (~x) begin Next_State <= d; y <= 1; end
                 else begin Next_State <= a; y<= 0;  end

             b : if (~x) begin Next_State <= c; y <= 1; end
                 else begin Next_State <= a; y<= 0;  end

             c : if (~x) begin Next_State <= d; y <= 1; end
                 else begin Next_State <= b; y<= 0; end

             d : if (~x) begin Next_State <= c; y <= 0; end
                 else begin Next_State <= d; y<= 1; end

            endcase

end module
```

ii. Στο ερώτημα αυτό έχουμε το παρακάτω Moore Finite State Machine .



Όμοια με το ερώτημα i. Έχουμε τον πίνακα καταστάσεων:

α/α	Τωρινή Κατάσταση		Είσοδος	Μελλοντική Κατάσταση		Έξοδος
	A_n	B_n	X	A_{n+1}	B_{n+1}	Y
0	0	0	0	0	1	0
1	0	0	1	0	0	0
2	0	1	0	1	0	1
3	0	1	1	0	0	1
4	1	0	0	0	1	1
5	1	0	1	1	1	1
6	1	1	0	1	0	0
7	1	1	1	1	1	0

Έτσι δημιουργούμε το παρακάτω μοντέλο Verilog:

```
module Moore_FSM(x,y,clock,reset);

      output [1: 0]y;
    input x,clock,reset;

      reg [1: 0]state;
    parameter a = 2'd0, b = 2'd1, c = 2'd2, d = 2'd3;

      always @ (posedge clock, negedge reset)
        if(reset == 0) state <= a;

            else case (state)

                  a: begin
                if(~x) state <= d; else state <= a;
                y <= 0;
                end

                  b: begin
                if(~x) state <= c; else state <= a;
                y <= 1;
                end

                  c: begin
                if(~x) state <= b; else state <= d;
                y <= 1;
                end

                  d: begin
                if(~x) state <= c; else state <= d;
                y <= 0;
                end

            endcase

endmodule
```