Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Συστήματα Μικροϋπολογιστών

Σειρά Ασκήσεων 2	Θανάσουλας Γρηγόριος | ΑΜ: 03114131 Μόνου Σταματίνα | ΑΜ: 03114077

Άσκηση 1

Ο κώδικας μαζί με σχόλια για την ερμηνεία του περιλαμβάνεται στο αρχείο askisi1seira2.8085.

```
;Ερώτημα 1
            IN 10H
            LXI H,0900H   ; Φόρτωση αρχικής διεύθυνσης
            MVI A,00H     ; A=0
LP_1:       MOV M,A       ; [(H)(L)] = A
            CPI FFH       ; Συνθήκη A - 255
            JZ END_LP1    ; Αν A = 255
            INX H         ; M++
            INR A         ; A++
            JMP LP_1
END_LP1:
;Ερώτημα 2
            LXI H,0900H   ; Φόρτωση αρχικής διεύθυνσης
            MVI E,FFH     ; E = FFH = 255
            MVI C,00H     ; C = 0
            MVI B,00H     ; B = 0
LOOP_3:                   ; Από 255 μέχρι και 0
            MOV A,M       ; A = [M]
            MVI D,08H     ; Loop eight times, για κάθε ψηφίο του 8bit αριθμού
LOOP_2:     RAL           ; CY = MSB του τρέχοντος A
            JNC NEXT      ; Αν CY == 0
            INX B         ; [(B)(C)]++
NEXT:       DCR D               ; D--
            JNZ LOOP_2    ; Αν D != 0 (Z=0), Ξανά loop
END_LP2:    INX H         ; Επόμενη διεύθυνση μνήμης
            DCR E         ; E--
            MOV A,E       ; A = E
            CPI FFH       ; Αν κάνει 0 - 1 γίνεται FF
            JZ END_LP3    ;
            JMP LOOP_3
END_LP3:
;Ερώτημα 3
```

```
                LXI H,0900H   ; Φόρτωση αρχικής διεύθυνσης
                MVI E,FFH     ; E = FFH = 255
                MVI D,00H     ; D = 0
LOOP_4:         MOV A,M       ; Από 255 μέχρι και 0

                CPI 10H       ; A - 10H, CY=?
                JC SKIP       ; Αν CY = 1, δλδ. A < 10H
                CPI 60H       ; A - 60H, CY=?, Z=0
                JZ COUNT      ; Αν A == 60H
                JNC SKIP      ; Αν A > 60H
COUNT:          INR D         ; D++
SKIP:           INX H         ; Επόμενη διεύθυνση μνήμης
                DCR E         ; E--
                MOV A,E       ; A = E
                CPI FFH       ; Αν κάνει 0 - 1 γίνεται FF
                JZ END_LP4    ;
                JMP LOOP_4
END_LP4:
; Ερώτημα 4
READ_AGAIN:LDA 2000H
                MOV E,A       ; Δημιουργώ αντίγραφο του A
                ANI 07H       ; Μάσκα τρίτων τελευταίων ψηφίων
                CPI 00H       ; Αν κανένα από τα 3 τελευατία δεν είναι ON, σβήσε τα LEDS
                JZ DISPLAY_OFF

                MOV A,E
                ANI 04H               ; Μάσκα 3ου σημαντικού ψηφίου
                CPI 04H               ; Σύγκτιση με το 00000100
                JZ DISPLAY_D          ; Εμφάνιση του D

                MOV A,E
                ANI 02H               ; Μάσκα 2ου σημαντικού ψηφίου
                CPI 02H               ; Σύγκτιση με το 00000010
                JZ DISPLAY_C          ; Εμφάνιση του D

                MOV A,E
                ANI 01H               ; Μάσκα 1ου σημαντικού ψηφίου
                CPI 01H               ; Σύγκτιση με το 00000001
                JZ DISPLAY_B          ; Εμφάνιση του B
DISPLAY_OFF:    MVI A,FFH             ; Απενεργοποίηση όλων των LED
                STA 3000H
                JMP READ_AGAIN
DISPLAY_B:      MOV A,B
                CMA           ; Αντιστροφή για αρνητική λογική LEDS
                STA 3000H
                JMP READ_AGAIN
```

```
DISPLAY_C:  MOV A,C
            CMA          ; Αντιστροφή για αρνητική λογική LEDS
            STA 3000H
            JMP READ_AGAIN
DISPLAY_D:  MOV A,D
            CMA          ; Αντιστροφή για αρνητική λογική LEDS
            STA 3000H
            JMP READ_AGAIN
END
```

Άσκηση 2

Έστω η καθυστέρηση της DELB σε χ ms, τότε θα πρέπει να επαναλαμβάνεται μία διαδικασία καθυστέρησης και ανάγνωσης για 30 sec / x ms φορές. Συγκεκριμένα λοιπόν, στην περίπτωση που η DELB καθυστερεί ⅕ sec, χρειαζόμαστε επανάληψη 150 φορές.

Ο κώδικας μαζί με σχόλια για την ερμηνεία του περιλαμβάνεται στο αρχείο askisi2seira2.8085.

```
                IN 10H          ; Ορισμός Χρονοκαθυστέρησης στο ζεύγος BC
                                ; 1/5 sec = 200ms = 00000000 11001000b
                LXI B,00C8H

WAIT_FOR_ON:    LDA 2000H
                ANI 80H         ; Μάσκα για το MSB
                CPI 80H         ; Σύγκριση
                JNZ WAIT_FOR_ON
WAIT_FOR_OFF:   LDA 2000H       ; Περίμενε για να γίνει OFF
                ANI 80H
                CPI 00H
                JNZ WAIT_FOR_OFF; Αν δεν έχει γίνει OFF

LEDS_ON:        MVI A,00H       ; Άναψε τα LEDS
                STA 3000H

LOOP_RESET:     MVI E,00H       ; E=0, Μετρητής
COUNT_TO_ON:    CALL DELB       ; Μετράμε περιμένοντας να δούμε ένα ON
                INR E           ; E++
                LDA 2000H
                ANI 80H         ; Μάσκα για το MSB
                CPI 80H         ; Σύγκριση
                JZ COUNT_TO_OFF ; Αν έγινε ON συνέχισε τη μέτρηση περιμένοντας για OFF
                INR E           ; D++
                MOV A,E         ; A=E
                CPI 96H         ; A == 150?
                JZ LEDS_OFF             ; Σβήσε τα φώτα, πέρασε ο χρόνος
                JMP COUNT_TO_ON; Συνέχισε τη μέτρηση
```

```
COUNT_TO_OFF:   CALL DELB
                INR E                   ; E++
                LDA 2000H
                ANI 80H                 ; Μάσκα για MSB
                CPI 00H                 ; Έγινε OFF;
                JZ LOOP_RESET           ; Αν έγινε OFF Reset τον μετρητή
                MOV A,E                 ; A=E
                CPI 96H                 ; A == 150?
                JZ LEDS_OFF             ; Σβήσε τα φώτα, πέρασε ο χρόνος
                JMP COUNT_TO_OFF        ; Συνέχισε τη μέτρηση

LEDS_OFF:       MVI A,FFH
                STA 3000H
                JMP WAIT_FOR_ON
END
```

Άσκηση 3

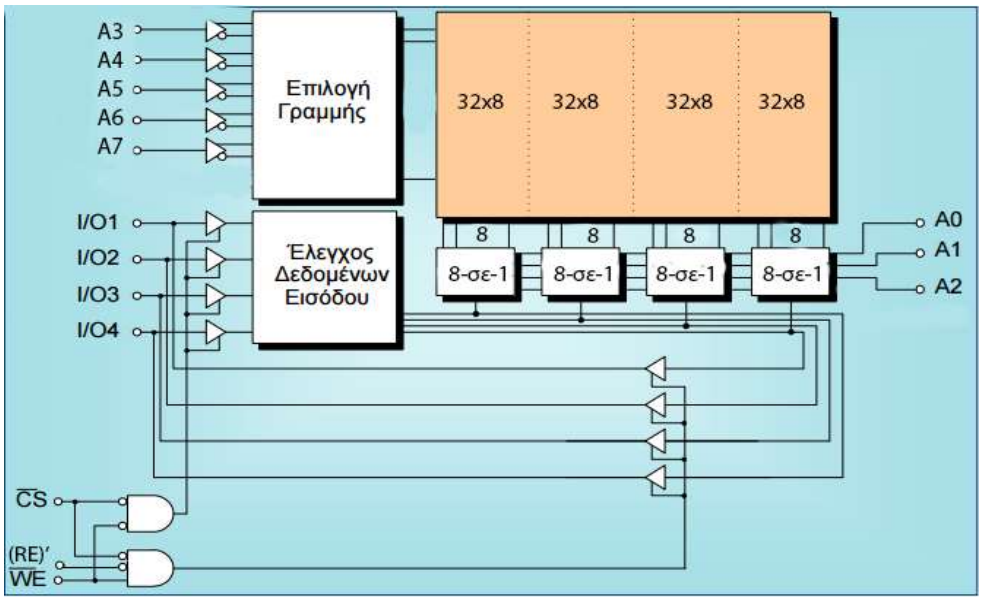
Ο κώδικας μαζί με σχόλια για την ερμηνεία του περιλαμβάνεται στο αρχείο askisi3seira2.8085.

```
                IN 10H
START:          MVI D,00H           ; D = 0, register αποτελέσματος

                LDA 2000H           ; Απομόνωση A3
                ANI 80H             ; Μάσκα 1000 0000
                MOV B,A             ; B = A3

                LDA 2000H           ; Απομόνωση B3
                ANI 40H             ; Μάσκα 0100 0000
                RLC                 ; Μεταφορά bit μία θέση αριστερά

                ANA B               ; Υπολογισμός X3
                RRC                 ; Μεταφορά στη σωστή θέση
                RRC
                RRC
                RRC
                ORA D               ; Τοποθέτηση στο αποτέλεσμα
                MOV D,A

                LDA 2000H           ; Απομόνωση A2
                ANI 20H             ; Μάσκα 0010 0000
                MOV B,A             ; B = A2

                LDA 2000H           ; Απομόνωση B2
```

```
                ANI 10H         ; Μάσκα 0001 0000
                RLC             ; Μεταφορά bit μία θέση αριστερά

                ANA B           ; Υπολογισμός X2
                RRC             ; Μεταφορά στη σωστή θέση
                RRC
                RRC
                ORA D           ; Τοποθέτηση στο αποτέλεσμα
                MOV D,A


                LDA 2000H       ; Απομόνωση A1
                ANI 08H         ; Μάσκα 0000 1000
                MOV B,A         ; B = A1

                LDA 2000H       ; Απομόνωση B1
                ANI 04H         ; Μάσκα 0000 0100
                RLC             ; Μεταφορά bit μία θέση αριστερά

                ORA B           ; Υπολογισμός X1
                RRC             ; Μεταφορά στη σωστή θέση
                RRC
                MOV C,A         ; Αποθήκευση C=X1
                ORA D           ; Τοποθέτηση στο αποτέλεσμα
                MOV D,A

                LDA 2000H       ; Απομόνωση A0
                ANI 02H         ; Μάσκα 0000 0010
                MOV B,A         ; B = A0

                LDA 2000H       ; Απομόνωση B0
                ANI 01H         ; Μάσκα 0000 0001
                RLC             ; Μεταφορά bit μία θέση αριστερά

                ORA B           ; Υπολογισμός A0 OR B0
                XRA C           ; (A0 OR B0) XOR X1
                RRC             ; Μεταφορά στη σωστή θέση

                ORA D           ; Τοποθέτηση στο αποτέλεσμα
                CMA             ; Αντιστροφή για αρνητική λογική LEDS
                STA 3000H
                JMP START
END
```

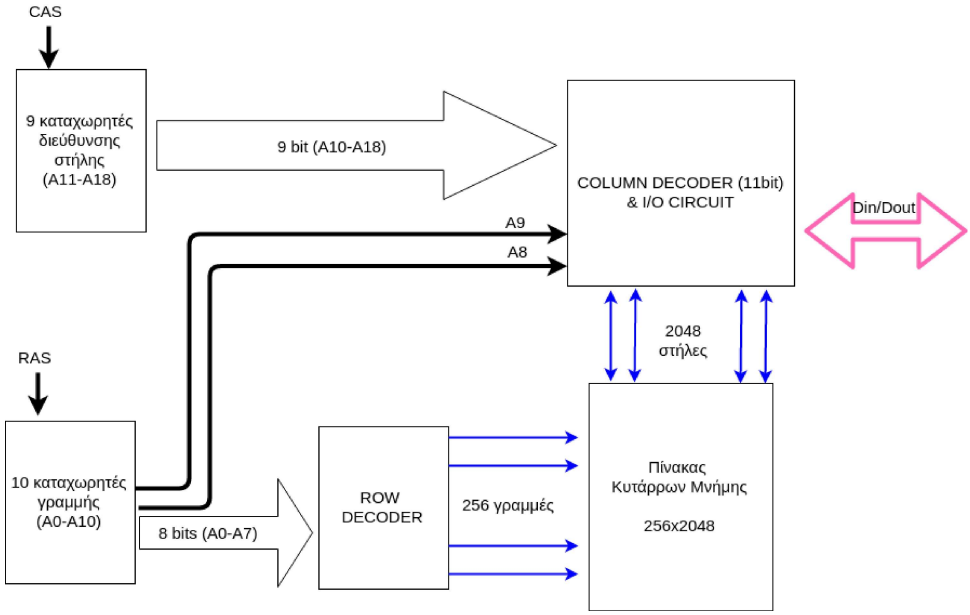
Άσκηση 4

Για τα 256x4 = 2^8 x 4 bit SRAM ένα δυνατό εσωτερικό διάγραμμα είναι το παρακάτω:



Η είσοδος (CS)' όταν είναι 1 απομονώνει και την είσοδο και την έξοδο. Όταν το (CS)' = 0, αν η το σήμα (WE)' = 0 γίνεται εγγραφή, ενώ αν το (RE)' = 0, γίνεται ανάγνωση. **Σημειώνεται δε, ότι αν είναι ενεργά ταυτόχρονα τα σήματα εγγραφής και ανάγνωσης, δηλαδή WR' = 0 και RD'=0, αποκλείεται η ανάγνωση και δίνεται προτεραιότητα στη διαδικασία της εγγραφής.**

Για την DRAM 512Kx1, ένα πιθανό διάγραμμα εσωτερικής οργάνωσης είναι το παρακάτω. Για τον προσδιορισμό της διεύθυνσης γίνεται πολυπλεξία του σήματος. Αρχικά, αποστέλλονται τα bits A0-A10 της διεύθυνσης γραμμής με τον παλμό RAS να είναι ενεργός, ενώ στη συνέχεια γίνεται αποστολή των bits A11-A18 με τον παλμό CAS να είναι ενεργός. Η εγγραφή της δυναμική μνημης γίνεται στο αρνητικό μέτωπο του παλμού (W)' ενώ το σήμα (G)' είναι 1. Η ανάγνωση γίνεται στο μέτωπο του παλμού (G)' ενώ η είσοδος (W)' πρέπει να είναι 1.Επιπλέον απαραίτητη είναι η τακτική ανανέωση του περιεχομένου της μνήμης στην περίπτωση που δεν έχουμε συχνή εγγραφή ή ανάγνωση απο αυτή. Η ανανέωση γίνεται ανά γραμμές της μνήμης, συγκεκριμένα ενεργοποιώντας την είσοδο (RAS)' και προσδιορίζοντας την γραμμη της μνήμης που θα ανανεωθεί. Η είσοδος (CAS)' παραμένει ανενεργή.



Άσκηση 5

Τα chips που δίνονται είναι τα παρακάτω:

- 10 Kbytes ROM
  - 4Kx8bit
  - 4Kx8bit
  - 2Kx8bit
- 6 Kbytes RAM
  - 2Kx8bit
  - 4Kx8bit

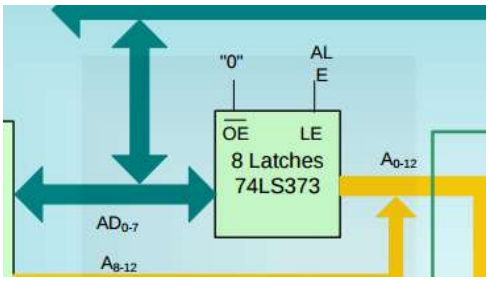
Αν υποθέσουμε ότι οι διευθύνσεις αποδίδονται στα τσιπ της μνήμης με τη σειρά που καταγράφονται στην προηγούμενη λίστα, οι διευθύνσεις που θα αντιστοιχούν στο κάθε τσιπ μνήμης φαίνονται παρακάτω:

Μνήμη	Chips	Πρώτη Διεύθυνση	Τελευταία Διεύθνση
10 Kbytes ROM	4Kx8bit	0000H	0FFFH
	4Kx8bit	1000H	1FFFH
	2Kx8bit	2000H	27FFH
6 Kbytes RAM	2Kx8bit	2800H	2FFFH
	4Kx8bit	3000H	3FFFH

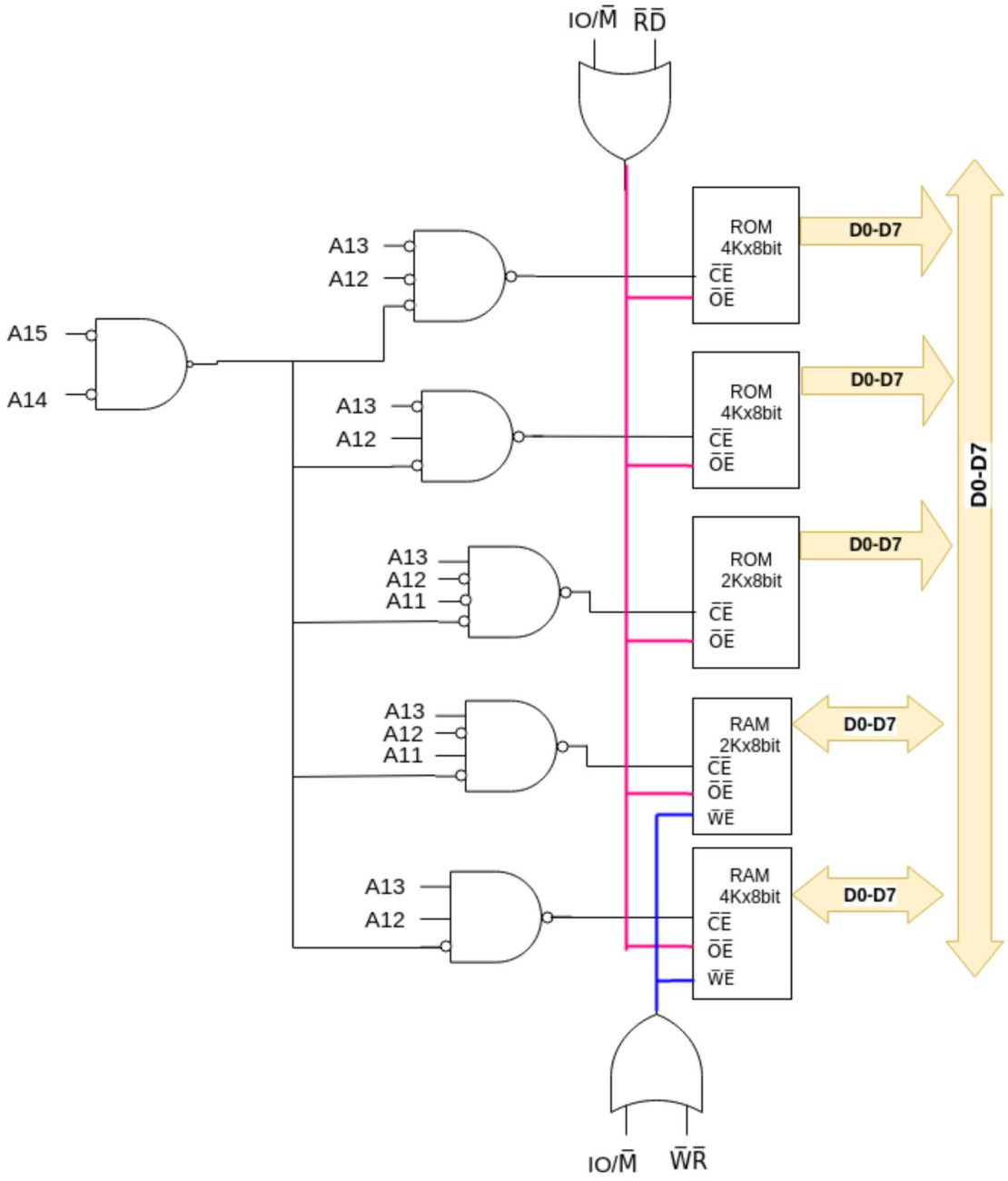
Αντίστοιχα, ο χάρτης μνήμης που προκύπτει είναι:

Διεύθυνση	Δυαδική αναπαράσταση	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0000H	0000 0000 0000 0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0FFFH	0000 1111 1111 1111	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
1000H	0001 0000 0000 0000	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1FFFH	0001 1111 1111 1111	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
2000H	0010 0000 0000 0000	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
27FFH	0010 0111 1111 1111	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1
2800H	0010 1000 0000 0000	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
2FFFH	0010 1111 1111 1111	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
3000H	0011 0000 0000 0000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
3FFFH	0011 1111 1111 1111	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Παρατηρούμε από τον χάρτη μνήμης πως για τον προσδιορισμό των chip στα οποία αντιστοιχεί κάθε μνήμη αρκεί να ελέγξουμε τα bits A11 έως A15



Σημειώνεται ότι γίνεται πολυπλεξία των A0-A7 με τα D0-D7 και για αυτό το λόγο χρειάζονται 7 latches, ώστε να λάβουμε τα A0-A7.



Άσκηση 6

```
//Πρόβλημα 4.37
module half_adder (output P, G, input A, B);
    xor (P,A,B);
    and (G,A,B);
endmodule

module full_adder (output S, Cout, input A, B, C);
    wire w1, w2, w3;
```

```
    half_adder HA1 (w1, w2, A, B);
    half_adder HA2 (S, w3, w1, C);
    or (Cout, w2, w3);
endmodule

module 4bit_adder_subtracter (output [3: 0] S, output C,
    input [3: 0] A, B, input M);
    wire [0: 3] w;
    wire [1: 3] c;
    xor X0 (w[0], B[0], M),
        X1 (w[1], B[1], M),
        X2 (w[2], B[2], M),
        X3 (w[3], B[3], M);
    full_adder FA0 (S[0], c[1], A[0], w[0], M),
        FA1 (S[1], c[2], A[1], w[1], c[]),
        FA2 (S[2], c[3], A[2], w[2], c[2]),
        FA3 (S[3], C, A[3], w[3], c[3]);
endmodule

//Πρόβλημα 4.40
module problem_4_40_adder_sub (output [3: 0] S, output C,
    input [3: 0] A, B, input M);
    assign {C, S} = M? A - B: A + B;
endmodule

//Πρόβλημα 4.47
module problem_4_47 (output F1, F2, input A, B, C, D);
    //Ορίζουμε την F1 με βάση τους ελαχιστόρους της συνάρτησης
    assign F1 = (~A & ~B & ~C & D) |//m1
        (~A & ~B & C & D) |         //m3
        (~A & B & ~C & ~D) |        //m4
        (A & ~B & C & D) |          //m11
        (A & B & ~C & ~D) |         //m12
        (A & B & ~C & D) |          //m13
        (A & B & C & ~D) |          //m14
        (A & B & C & D) |           //m15

    assign F2 = (~A & ~B & ~C & D) |//m1
        (~A & ~B & C & ~D) |        //m2
        (~A & B & ~C & D) |         //m5
        (~A & B & C & D) |          //m7
        (A & ~B & ~C & ~D) |        //m8
        (A & ~B & C & ~D) |         //m10
        (A & ~B & C & D) |          //m11
        (A & B & ~C & D) |          //m13
        (A & B & C & D) |           //m15
endmodule
```

Άσκηση 7

```
//Πρόβλημα 6.38
module problem_6_38_a (output [3: 0] A, input [3: 0] I,
```

```
    input up, down, load, clk)
    reg [3: 0] out;

    always @ (posedge clk)
    if (load) A <= I;
    else if (up) A <= A + 4'b0001;      //A <= A + 1
    else if (down) A <= A - 4'b0001;    //A <= A - 1
    else A <= A;                        //Διατήρηση Εξόδου
endmodule


module problem_6_38_b (output [3: 0] A, input [3: 0] I,
    input s1, s0, clk)

    case ({s1, s0})                     //Έλεγχος με βάση τα selectors
        2'b00: A <= A + 4'b0001;        //Up
        2'b01: A <= A - 4'b0001;        //Down
        2'b10: A <= I;                  //Φόρτωση εισόδου
        2'b11: A <= A;                  //Καμία αλλαγή
    endcase
endmodule

//Πρόβλημα 6.39

//Behavioral decription
module problem_6_39 (output [2: 0] count, input clk, reset);
    always @ (posedge clk, negedge reset)
        if (reset == 0) count <= 0; //Επαναφορά στο 0
        else case (count)
            0: count <= 1;
            1: count <= 2;
            2: count <= 4;
            4: count <= 5;
            5: count <= 6;
            6: count <= 0;
        endcase
endmodule

//Structural description


//Πρόβλημα 8.10
module problem_8_10 (input x, y, clk, reset);   //Προσθέσαμε και reset για την επαναφορά
    reg [1:0] next_state, current_state;
    parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;   //Ορισμός σταθερών
    always @ (posedge clk, negedge reset)
        if (reset == 0) current_state <= s0;             //Επαναφορά στην κατάσταση 00
        else current_state <= next_state;               //Μετάβαση στην επόμενη κατάσταση

    always @ (state, x, y) begin   //Διαρκής υπολογισμός νέας κατάστασης
        next_state = s0;
        case(current_state)
            s0: if (x==0) next_state = s0; else next_state = s1;
            s1: if (y==0) next_state = s2; else next_state = s3;
```

```
            s2: if (x==0) next_state = s0; else if (y == 0) next_state = s2; else next_state = s3;
            s3: if (x==0) next_state = s0; else if (y == 0) next_state = s2; else next_state = s3;
        endcase
    end
endmodule
```

Άσκηση 7 / Πρόβλημα 6.39

Ειδικά για το ερώτημα αυτό και την μετατροπή ώστε να γίνεται χρήση D Flip-Flop, ακολουθούμε τα παρακάτω βήματα:

Σχηματίζουμε τον πίνακα καταστάσεων:

A[n]	B[n]	C[n]	A[n+1]	B[n+1]	C[n+1]	D_A	D_B	D_C
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	0	0	0	0	0	0

Για το D_A

Σχηματίζουμε τον πίνακα Karnaugh

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

από όπου προκύπτει ότι D_A = A'B + AB'=A XOR B

Για το D_B

Σχηματίζουμε τον πίνακα Karnaugh

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

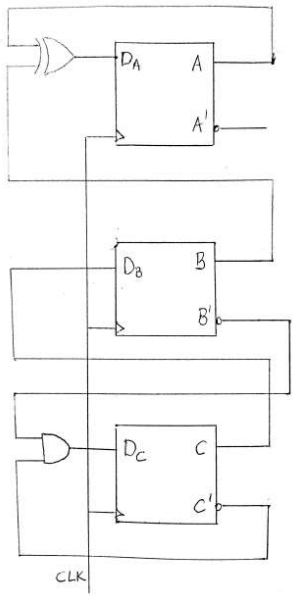
από όπου προκύπτει ότι Db= C

Για το D_C

Σχηματίζουμε τον πίνακα Karnaugh

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

από όπου προκύπτει ότι Dc = B'C'= (NOT B) AND (NOT C)




```
module D_FF (Q, Q_inv, D, Clk);
	output Q, Q_inv;
	input D, Clk;
	reg Q, Q_inv;
	always @ (posedge Clk)
		Q <= D;
		Q_inv <=  ~Q   ;
endmodule

module prob_6_39 ( output A, B , C, input Clk);
	wire Da, Dc, B_inv, C_inv;
	xor (Da, A, B);
	and (Dc, B_inv, C_inv);
	D_FF    (A, A_inv, Da, Clk );
	D_FF (B, B_inv, C, Clk );
	D_FF (C, C_inv, Dc, Clk);
endmodule
```