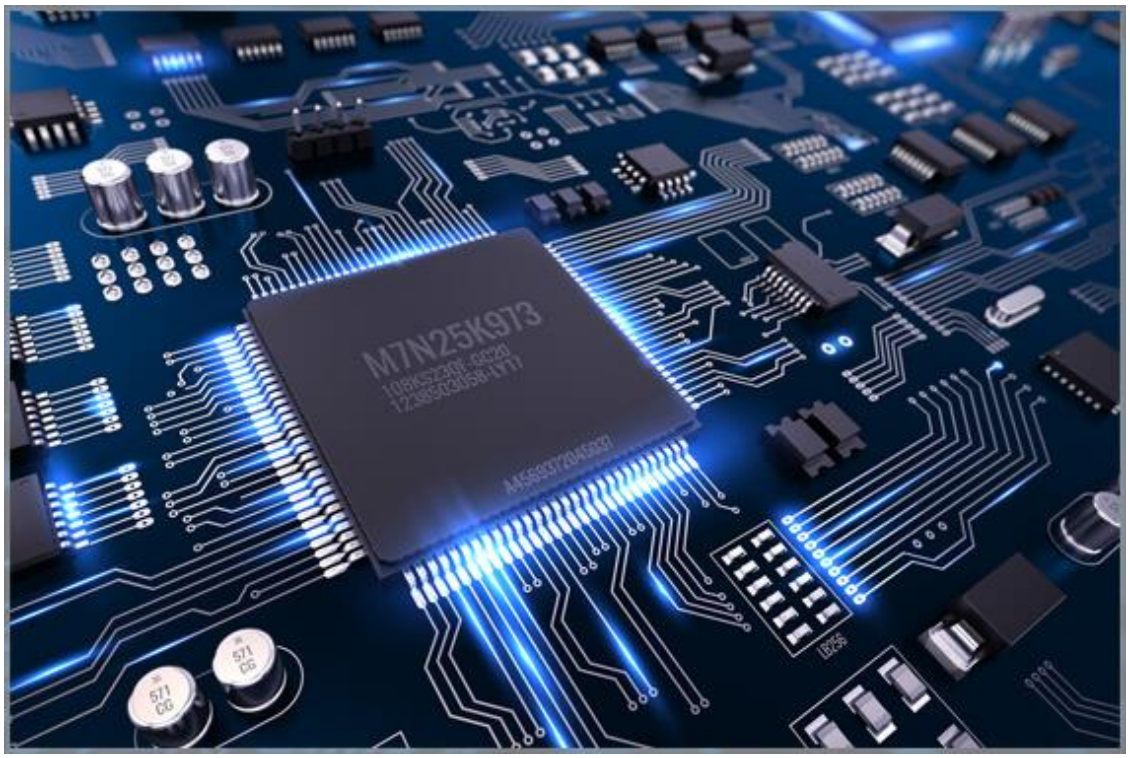
ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ

1Η ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ



APRIL 14, 2021

ΘΟΔΩΡΗΣ ΑΡΑΠΗΣ – EL18028
ΚΡΙΣ ΚΟΥΤΣΗ – EL18905

Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2020-2021

ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ

ΑΣΚΗΣΕΙΣ ΠΡΟΣΟΜΟΙΩΣΗΣ

Άσκηση 1

Μεταφράζοντας το πρόγραμμα της άσκησης από γλώσσα μηχανής σε γλώσσα Assembly λαμβάνουμε το παρακάτω αριστερά:

```
MVI C,08H
LDA 2000H
RAL
JC 080DH
DCR C
JNZ 0805H
MOV A,C
CMA
STA 3000H
RST 1

END
```

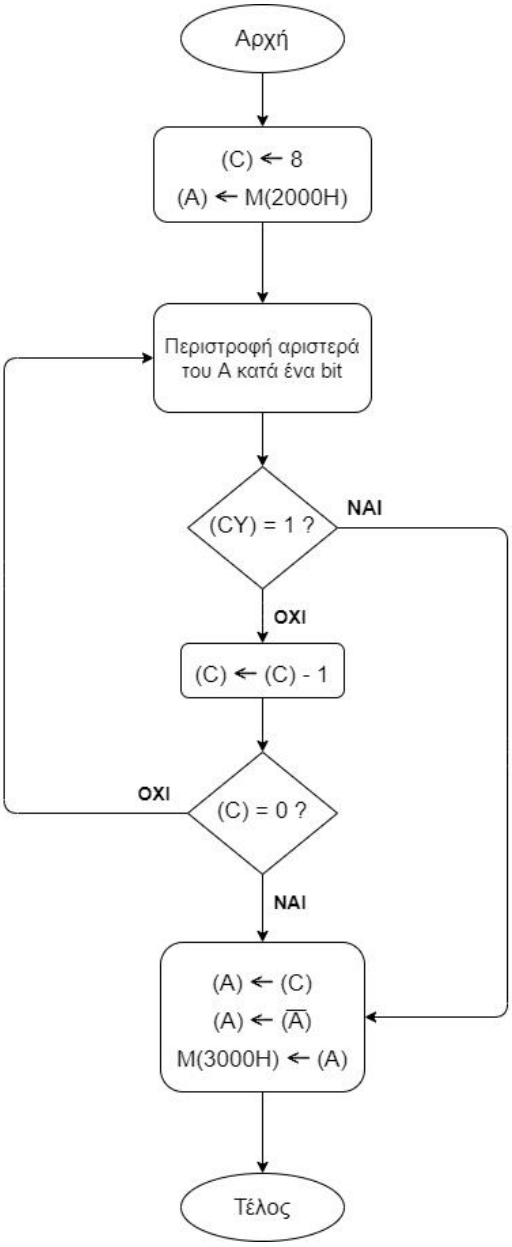
```
0800 0E   MVI C,08H
0801 08
0802 3A   LDA 2000H
0803 00
0804 20
0805 17   RAL
0806 DA   JC RESET
0807 00
0808 00
0809 0D   DCR C
080A C2   JNZ RESET
080B 00
080C 00
080D 79   MOV A,C
080E 2F   CMA
080F 32   STA 3000H
0810 00
0811 30
0812 CF   RST 1
```

Περιέχεται στο αρχείο *Translated_in_Assembly.8085*. Προκειμένου, όμως, να μεταφραστεί και να εκτελεστεί θα πρέπει να αντικαταστήσουμε τις διευθύνσεις των συνθηκών άλματος με ετικέτες, όπως φαίνεται στο τελικό πρόγραμμα, γιατί αλλιώς θα αποτύχει η μετάφραση.

Η λειτουργία του προγράμματος αυτού είναι να εμφανίζει σε δυαδική αναπαράσταση στα φωτάκια LED (αριθμημένα από δεξιά προς τα αριστερά κατά 1, 2,… ,8) τον αριθμό της θέσης του πιο αριστερού dip switch που είναι ON ή, από άλλη οπτική, επιστρέφει το συμπλήρωμα ως προς 1 (αριθμημένα από αριστερά προς τα δεξιά κατά 0, 1,… ,7) του αριθμού της θέσης του πιο αριστερού dip switch που είναι ON. Η τιμή αυτή αναπαριστάται δυαδικά από τα αναμένα LEDs (Αντίστροφη λογική, $0 \rightarrow ON, 1 \rightarrow OFF$.)

Δεξιά παρατίθεται το αντίστοιχο πρόγραμμα με τις συμβολικές διευθύνσεις.

Το διάγραμμα ροής του προγράμματος είναι το ακόλουθο:



Προκειμένου να αποκτήσει συνεχή λειτουργία το πρόγραμμα, αρχικά μπορούμε βγάλουμε την εντολή RST 1 (η οποία καταλαμβάνει και ένα byte μνήμης και η οποία θέτει το $(PC) = 8 * 001 = 0008H$) και στην θέση της να βάλουμε μία συνθήκη άλματος JMP START, όπου START είναι η ετικέτα της διεύθυνσης 0800, δηλαδη της αρχής του προγράμματος. Ως εκτούτου, θα έχουμε το εξής πρόγραμμα που περιέχεται στο αρχείο *Modified_to_run_forever.8085*:

```
START:
     MVI C,08H
     LDA 2000H
REPEAT:
     RAL
     JC STORE
     DCR C
     JNZ REPEAT
STORE:
     MOV A,C
     CMA
     STA 3000H
     JMP START

     END
```

Άσκηση 2

```
     IN 10H
     LXI B,01F4H  ;This will be the delay (500ms = 01F4Hms)
     MVI E,FEH    ;Initialise E to store FEH (11111110) so that the LSB of LEDs is switched on
START:
     LDA 2000H    ;Load INPUT from dip switches to A
     CALL DELB    ;delay of 500ms
     RRC          ;Rotate right to check if CY = 1
     JNC START    ;If CY = 0 then the LSB dip switch is off so start again
     RLC          ;Getting back to the initial state
     RLC          ;Rotate left to check if CY = 1
     JC RIGHT     ;If CY = 1 then jump to RIGHT
LEFT:
     MOV A,E      ;Give A the previous state of LEDs
     STA 3000H    ;OUTPUT to the LEDs
     RLC          ;Move the LED light one position to the left
     MOV E,A      ;Store new state in E (E will always store the position
                  ;of the LED light which is ON)
     JMP START
RIGHT:            ;Similar pattern as above
     MOV A,E
     STA 3000H
     RRC
     MOV E,A
     JMP START

     END
```

Το πρόγραμμα βρίσκεται στο αρχείο *Άσκηση_2.8085*.

Άσκηση 3

```
START:
     LDA 2000H
     CPI 64H      ;Is it greater than 99? (A-99)
     JNC CASE1    ;If yes then jump to CASE1 (A>99)
     MVI D,FFH    ;Else A<=99
DECA:
     INR D
     SUI 0AH      ;While A>10 do A-10
     JNC DECA     ;If A>0 continue
     ADI 0AH      ;Correct negative remainder
     MOV E,A      ;Save units in E
     MOV A,D      ;save tens in A
     RLC
     RLC
     RLC
     RLC          ;Move the value of tens to the 4 MSB of A
     ADD E        ;Add units to A so that we have: tens->MSB, units->LSB
     CMA          ;Get the supplement of A beacause LED lights switch on 0 value
     STA 3000H    ;Output A (LED lights turn on indicating the value of A as implied above)
     JMP START
CASE1:
     CPI C8H      ;Is it greater than 199 (A-99)
     JNC CASE2    ;If yes then jump to CASE2 (A>199)
     MVI A,F0H    ;A = 1111 0000, 4 most right LEDs switch on
     STA 3000H
     MVI A,FFH    ;A = 1111 1111, 4 most right LEDs switch off
     STA 3000H
     JMP START    ;Jump to start and repeat until we get a new input
CASE2:
     MVI A,0FH    ;A = 0000 1111, 4 most left LEDs switch on
     STA 3000H
     MVI A,FFH    ;A = 1111 1111, 4 most left LEDs switch off
     STA 3000H
     JMP START    ;Jump to start and repeat until we get a new input

     END
```

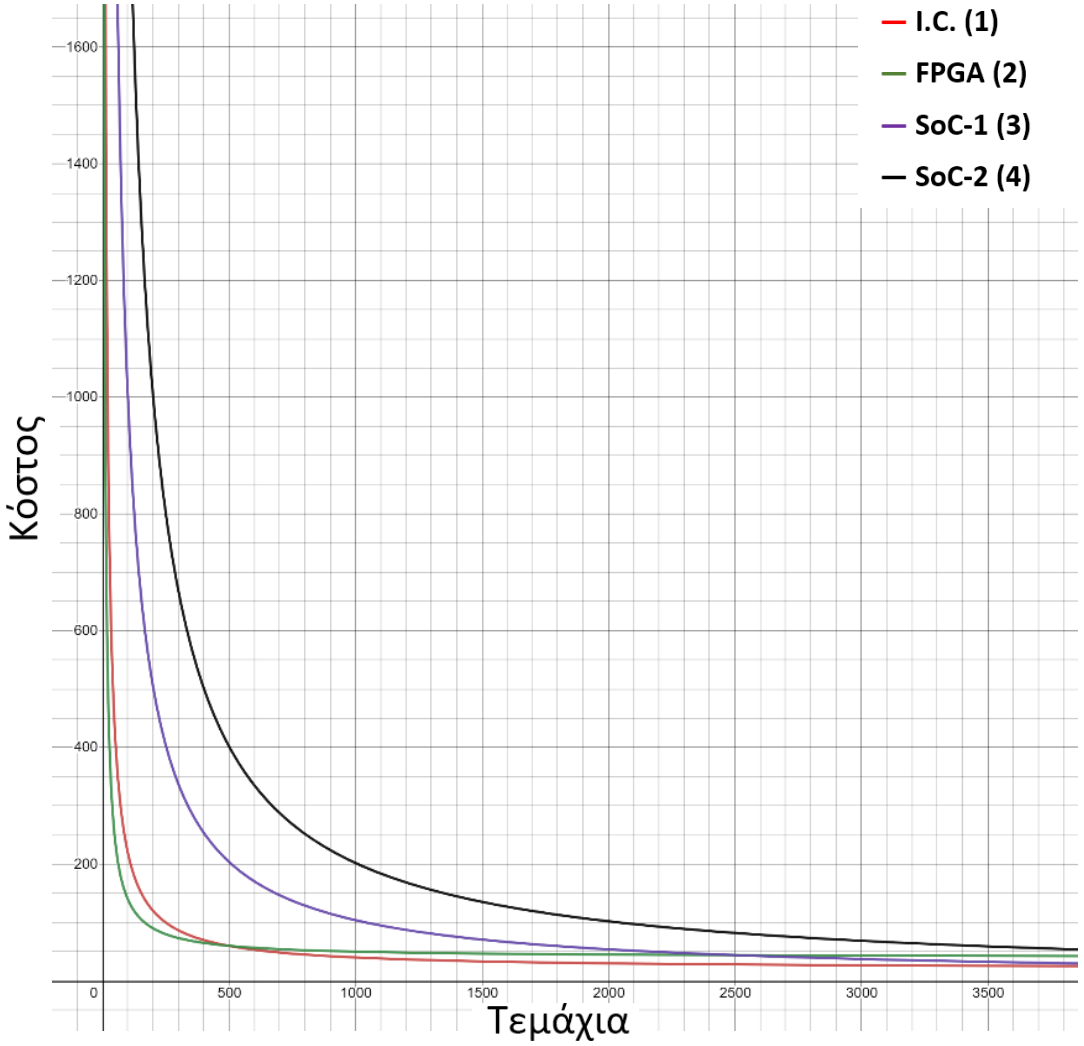
Το πρόγραμμα βρίσκεται στο αρχείο *Άσκηση_3.8085*.

Άσκηση 4

Αρχικά, με βάση τα δεδομένα, εξάγουμε τις σχέσεις κόστους ανά τεμάχιο και έχουμε τις παρακάτω συναρτήσεις κόστους $(f_i(x))$ ανά τεμάχιο x:

1. $f_1(x) = \frac{20000+20x}{x}$
2. $f_2(x) = \frac{10000+40x}{x}$
3. $f_3(x) = \frac{100000+4x}{x}$
4. $f_4(x) = \frac{200000+2x}{x}$

Οι αντίστοιχες καμπύλες φαίνονται παρακάτω:



Στη συνέχεια εξισώνουμε κάθε συνάρτηση κόστους σε ζεύγη και λαμβάνουμε τα εξής σημεία τομής των καμπυλών:

1-2: $f_1(x) = f_2(x) \Longrightarrow x = 500$

1-3: $f_1(x) = f_3(x) \Longrightarrow x = 5000$

1-4: $f_1(x) = f_4(x) \Longrightarrow x = 10000$

2-3: $f_2(x) = f_3(x) \Longrightarrow x = 2500$

2-4: $f_2(x) = f_4(x) \Longrightarrow x = 5000$

3-4: $f_3(x) = f_4(x) \Longrightarrow x = 50000$

Σύμφωνα με τα παραπάνω και σε συνδυασμό με το διάγραμμα των γραφικών παραστάσεων, συμπεραίνουμε πως οι συμφερότερες περιοχές τεμαχίων για κάθε τεχνολογία είναι:

(1) I.C.: $500 < x < 5000$

(2) FPGA: $0 < x < 500$

(3) SoC-1: $5000 < x < 50000$

(4) SoC-2.: $50000 < x$

Εύκολα παρατηρούμε πως οι τεχνολογίες με το μεγαλύτερο κόστος σχεδίασης (SoC-1, SoC-2) είναι πιο κερδοφόρες για υψηλότερους αριθμούς τεμαχίων.

Προκειμένουν να εξαφανιστεί η επιλογή της πρώτης τεχνολογίας θα πρέπει, για το νέο κόστος (έστω κ) ανά I.C. της τεχνολογίας των FPGAs, να ισχύει:

$$f_2{}'(x) = \frac{10000 + (10 + \kappa)x}{x} < \frac{20000 + 20x}{x} = f_1(x) \Longrightarrow$$

$$\kappa < \frac{10000}{x} + 10$$

Για $x \to \infty$ θα πρέπει να ισχύει $\kappa < 10$, αλλά επειδή προφανώς δεν μπορούν να σχεδιαστούν άπειρα τεμάχια αρκεί να ισχύει $\kappa \leq 10$. Άρα, οποιαδήποτε τιμή από 0 εώς 10€ για το νέο κόστος κ αρκεί για να αποκλειστεί η πρώτη τεχνολογία.

Άσκηση 5

(i)

```
module Circuit_F1_gates (A, B, C, D, F);
    input A, B, C, D;
    output F;

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

    and G1 (w1, B, C);
    or G2 (w2, w1, D);
    and G3 (w3, w2, A);
    not GnB (Bnot, B), GnC (Cnot, C);
    and G4 (w4, Bnot, Cnot, D);
    or G5 (F, w4, w3);
endmodule
```

```
primitive UDP_F2_gates(F2, A, B, C, D);
    output F2;
    input A, B, C, D;
    table    //Πίνακας αληθείας για την F2
    //A    B    C    D:    F2 // Column header comment
    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:    0;
    1      0    0    1:    1;
    1      0    1    0:    1;
    1      0    1    1:    1;
    1      1    0    0:    0;
    1      1    0    1:    1;
    1      1    1    0:    1;
    1      1    1    1:    0;
    end table
end primitive
```

(ii)

```
module Circuit_F1_dataflow (A, B, C, D, E, F);
    input A, B, C, D;
    output E, F;

    assign F = (((B & C) | D) & A) | (~B & ~C & D);
endmodule
```

```
primitive UDP_F2_dataflow(F2, A, B, C, D);
    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);
endprimative
```

```
module Circuit_F3_gates(A, B, C, D, E, F3);
    input A, B, C, D, E;
    output F3;

    wire w1, w2, w3, w4, w5, w6, w7;

    and G1(w5, A, B, C);
    and G2(w1, B, C);
    or  G3(w2, A, w1);
    and G4(w6, w2, D);
    or  G5(w3, B, C);
    and G6(w4, D, E);
    and G7(w7, w3, w4);
    or  G8(F3, w5, w6, w7);
endmodule
```

```
module Circuit_F3_dataflow (A, B, C, D, E, F3)
    input A, B, C, D, E;
    output F3;
    assign F3 = (A & B & C) | ((A | (B & C)) & D) | ((B | C) & (D & E));
endmodule
```

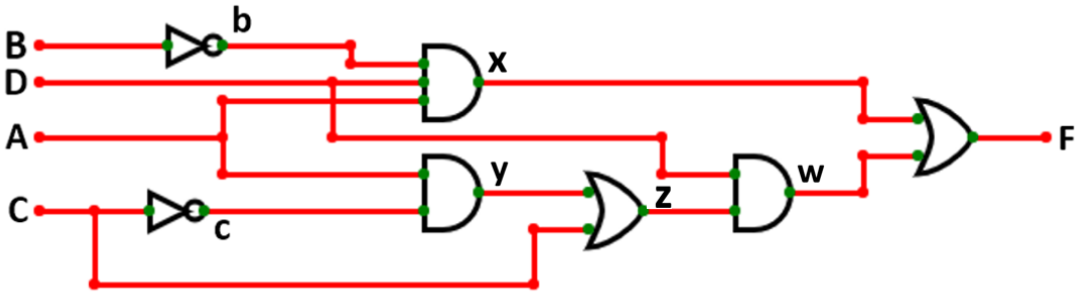
```
module Circuit_F4_gates (A, B, C, D, E, F);
    input A, B, C, D, E;
    output F;

    wire w1, w2, w3, w4;

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

```
module Circuit_F4_dataflow (A, B, C, D, E, F);
    input A, B, C, D, E;
    output F;
    assign F = (A & (B | (C & D) | E)) | (B & C & D & E);
endmodule
```

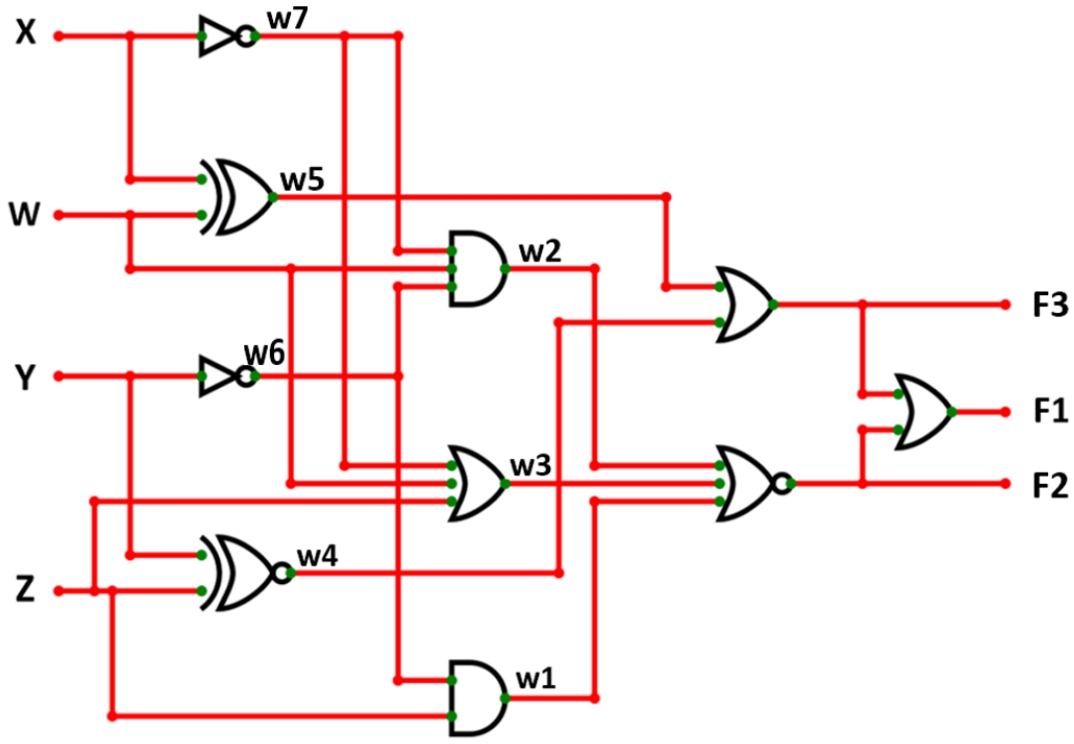
Άσκηση 6

(i)

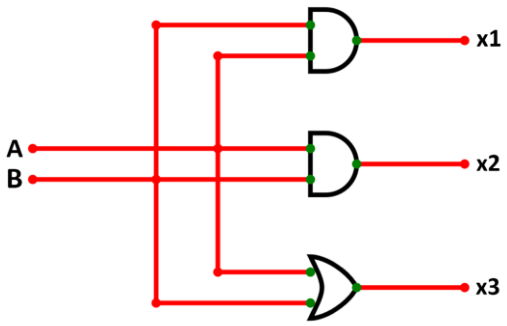
(a)



(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);
    half_adder HA2(S, C2, C1, z);
    or G1(C, C2, C1);
endmodule

module 4_bit_add_sub(output [3:0] S, output C, V, input [3:0] A, B, input K);
    wire C1, C2, C3;
    wire w0, w1, w2, w3;

    xor G1(w0, B[0], K);
    xor G2(w1, B[1], K);
    xor G3(w2, B[2], K);
    xor G4(w3, B[3], K);

    full_adder FA0 (S[0], C1, w0, A[0], K);
    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)

```
module 4_bit_add_sub(output [3:0] S, output C, V, input [3:0] A, B, input K);
    assign {C, S} = K ? A – B : A + B;
endmodule
```

Άσκηση 7

(i)

```
module Mealy_Machine (output reg out, input in, clock, reset);
    reg [1:0] state, next_state;
    parameter a = 2'b00,
              b = 2'b11,
              c = 2'b10,
              d = 2'b01;
    always @ (posedge clock, negedge reset) // state update or reset
        if (reset == 0) state <= a;
        else state <= next_state;
    always @ (state, in)
        case (state)
            a: if (in) next_state = a; else next_state = d;
            b: if (in) next_state = a; else next_state = c;
            c: if (in) next_state = b; else next_state = d;
            d: if (in) next_state = d; else next_state = c;
        endcase
    always @ (state, in) // output formation
        case (state)
            a, c: out = ~in;
            b, d: out = in;
        endcase
endmodule
```

(ii)

```
module Moore_FSM_ii (input in, clock, reset, output out);
    Reg [1: 0] state;
    Parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;
    always @ (postedge clock, negedge reset)
        if (reset == 0) state <= a;
        else case (state)
            a: if (in) state <= a; else state <= d;
            b: if (in) state <= a; else state <= c;
            c: if (in) state <= d; else state <= b;
            d: if (in) state <= d; else state <= c;
        endcase
    always @ (state)
        case (state)
            a, d: out <= 1'b0;
            b, c: out <= 1'b1;
        endcase
endmodule
```