Συστήματα Μικρουπολογιστών

1η Ομάδα Ασκήσεων

Μοίρας Αλέξανδρος Α.Μ.: el18081

Παπαδημητρίου Ευθύμιος Α.Μ.: el18129

Ασκήσεις στην Γλώσσα Περιγραφής Υλικού Verilog

1η Άσκηση:

Το πρόγραμμα assembly:

```
MVI C,08H
LDA 2000H
F1: RAL
JC F2
DCR C
JNZ F1
F2: MOV A,C
CMA
STA 3000H
RST 1

END
```

Το πρόγραμμα βρίσκεται και στο αρχείο Exercise1_assembly.8085

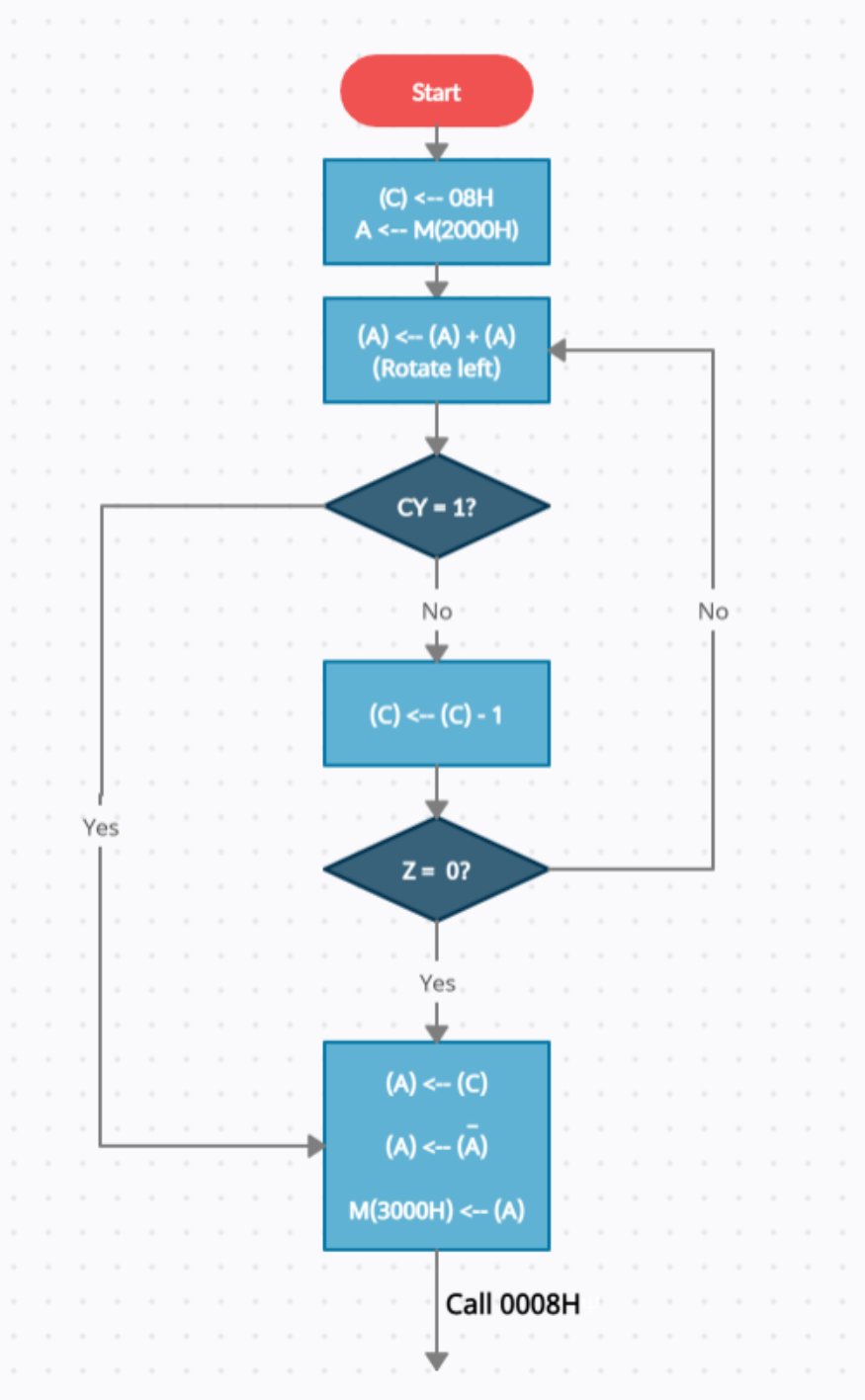
Σημείωση: Στο πρόγραμμα χρησιμοποιήθηκαν τα flags F1, F2 επειδή ο simulator δεν επέτρεπε jump κατευθείαν στη διεύθυνση μνήμης των αντίστοιχων εντολών.

Το πρόγραμμα διαβάζει από την είσοδο ένα byte και εκτυπώνει στην έξοδο σε ποιο από τα 8 ψηφία του (από το 1ο-LSB- έως το 8ο-MSB-) βρέθηκε ο most significant άσσος ή εκτυπώνει 0 αν δεν υπάρχει άσσος.

Για να το πετύχει αυτό χρησιμοποιεί τον καταχωρητή C ως μετρητή ξεκινώντας τον από την τιμή 8 και στην συνέχεια μεταφέρει την είσοδο στον accumulator.

Έπειτα για να ελέγξει αν το MSB της εισόδου, που βρίσκεται στον accumulator, είναι 1, εκτελεί αριστερή ολίσθηση του accumulator τοποθετώντας το MSB στο κρατούμενο. Αν το κρατούμενο δεν είναι 1 τότε ο most significant άσσος δεν βρίσκεται στο 8ο Bit άρα μειώνεται το C κατά 1 αφού τώρα πρέπει να ελεγχθεί αν το 7ο Bit της αρχικής εισόδου είναι 1 επαναλαμβάνοντας (με χρήση jump στο F1) τη διαδικασία της αριστερής περιστροφής που εξηγήθηκε παραπάνω. Συνεχίζουμε έτσι έως ότου βρούμε τον most significant άσσο, οπότε γίνεται Jump σε παρακάτω μέρος του προγράμματος (F2), ή το περιεχόμενο του καταχωρητή C γίνει 0, που σημαίνει ότι ελέγχθηκαν και τα 8 Bit και ήταν όλα μηδενικά. Άρα μετά την εκτέλεση των παραπάνω ο C θα περιέχει τη θέση του most significant άσσου ή το 0 αν δεν υπήρχαν άσσοι. Έπειτα μετακινούμε το περιεχόμενο του C στον accumulator και αντιστρέφουμε όλα τα bits του αφού θέλουμε να το εκτυπώσουμε στην έξοδο που γνωρίζουμε ότι λειτουργεί με αντίστροφη λογική. Για να το εκτυπώσουμε το αποθηκεύουμε στη θέση 3000H η οποία στο σύστημα μας είναι mapped στην έξοδο. Άρα μετά την εκτέλεση ολόκληρου του προγράμματος στα LED της εξόδου θα εμφανίζεται σε δυαδική μορφή η θέση του Most significant άσσου της εισόδου. Ύστερα η RST 1 στο σύστημα μας καλεί μία ρουτίνα που βρίσκεται στη θέση 0008H και σώζει τα περιεχόμενα των καταχωρητών στη RAM και επιστρέφει στο Monitor πρόγραμμα.

Το διάγραμμα ροής του προγράμματος:



Για να εκτελείται αενάως το πρόγραμμα θα βάλουμε ένα Flag START στην αρχή του και την εντολή JMP START αμέσως μετά την αποθήκευση του accumulator στη θέση μνήμης 3000H που είναι mapped στην έξοδο. Η εντολή RST 1 θα παραλειφθεί. Ο κώδικας αυτού του νέου προγράμματος βρίσκεται στο αρχείο Exercise1_forever_assembly.8085. Και σε screenshot:

```
START: MVI C,08H
LDA 2000H
F1: RAL
JC F2
DCR C
JNZ F1
F2: MOV A,C
CMA
STA 3000H
JMP START
END
```

2η Άσκηση:

Η άσκηση βρίσκεται στο αρχείο Exercise2_assembly.8085 αλλά και σε screenshot παρακάτω:

```
LXI B,01F4H ; Kathorismos xronou kathysterhshs. 500 dekadiko sto
            ; zeugos BC wste kathysterhsh=500*1ms=0.5sec

LDA 2000H ; Diavasma arxikhs eisodou gia na anapsei swsta prwta to
          ; LSB LED se kathe periptwsh
RAL       ; Metafora tou MSB sto CY
JNC STARTLEFT ; Elegxos poia einai h arxikh fora peristrofhs
STARTRIGHT: MVI D,FDH ;An einai dexia D<--FD wste meta apo mia
JMP STARTSTOP ;dexia peristrofh, na anapsei arxika to 1o LED
STARTLEFT: MVI D,7FH ;An einai aristerh D<--7F entelws antistoixa
   ;O D sto programma tha periexei thn trexousa katastash twn LED

STARTSTOP: LDA 2000H ;Anagnwsh ths eisodou
MOV E,A ; Proswrinh apothhkeush eisodou gia metepeita elegxo
RAR     ; Metafora tou LSB sto CY
JNC STARTSTOP ; Elegxos an LSB=0 gia paush ths peristrofhs
MOV A,E ; Epanafora eisodou ston A
RAL     ; Metafora tou MSB sto CY
JNC ARISTERA ; An MSB=0 metavash sth routina aristerhs peristrofhs

DEXIA: MOV A,D ; Metafora trexousas katastashs ston A
RRC ; Dexia peristrofh tou anamenou LED
MOV D,A ; Apothhkeush trexousas katastashs
STA 3000H ; Ektypwsh ths katastashs sta LED exodou
CALL DELB ; Klhsh ths routinas kathysterhshs
JMP STARTSTOP ; LOOP

ARISTERA: MOV A,D ;plhrhs antistoixia me routina dexias peristrofh:
RLC ;Aristerh peristrofh tou anamenou LED
MOV D,A
STA 3000H
CALL DELB
JMP STARTSTOP

END
```

3η Άσκηση:

Η άσκηση βρίσκεται στο αρχείο Exercise3_assembly.8085 αλλά και σε screenshots παρακάτω:

```
LXI B,07D0H ; Fortwma tou 2000 sto zeugos BC gia thn DELB
START: LDA 2000H ; Fortwsh eisodou
CPI C8H ; Elegxos an eisodos megalyterh toy 199
JNC GTHAN199 ; An nai metavash sto GTHAN199
CPI 64H ; Elegxos an eisodos > 99 alla mikroterh tou 199
JNC GTHAN99 ; An nai metavash sto GTHAN99
MVI D,FFH ; Arxikopoihsh metrhth dekadwn sto FF wste otan
          ; auxh8ei kata 1 na xekina apo to 0

DECA: INR D ; Auxhsh dekadwn
SUI 0AH ; Afairoume synexws to 10
JNC DECA ; ews otou to apotelesma<0 wste na metrhsoume tis dekades
ADI 0AH ; Prosthetoume 10 ston arnhtiko pou proekypse gia na
        ; paroume tis monades
MOV E,A ; Metakinhsh monadwn ston kataxwrhth E
MOV A,D ; Metakinhsh dekadwn ston accumulator
RAL     ; wste na tis feroume sta 4 MSB pshfia me ta diadoxika RAL
RAL
RAL
RAL
ANI F0H ; Maska gia na vgaloume tyxon kratoumeno pou topo8eth8hke
        ; sto bit 3 kata tis peristrofes
ORA E   ; Topo8ethsh monadwn sta 4 LSB tou accumulator
CMA     ; Symplhrwma ws pros 1 logw arnhtikhs logikhs exodou
STA 3000H ; Eggrafh sthn exodo
JMP START ; Epistrofh sthn arxh gia aenah leitourgia
```

```
GTHAN199: MVI A,0FH ; Logw arnhtikhs logikhs 0F gia na
                    ; anavoun ta 4 MSB LEDs
STA 3000H ; Anama sthn exodo
CALL DELB ; Kathysterhsh
MVI A,FFH ; FF gia na svhsoun ola ta LED
STA 3000H ; Svhsimo olwn twn led sthn exodo
CALL DELB ; Kathysterhsh
JMP START ; Epistrofh sthn arxh gia aenah leitourgia

GTHAN99: MVI A,F0H ; Entelws antistoixa me to parapanw kommati
                   ; alla F0 wste na anapsoun ta 4 LSB
STA 3000H
CALL DELB
MVI A,FFH
STA 3000H
CALL DELB
JMP START

END
```

4η Άσκηση:

Η 1η τεχνολογία έχει σταθερό κόστος 20000€ και μεταβλητό (10+10)€ ανά τεμάχιο άρα συνολικό κόστος $20000 + 20 \cdot n$ όπου n ο αριθμός των τεμαχίων.

Η 2η τεχνολογία έχει σταθερό κόστος 10000€ και μεταβλητό (30+10)€ ανά τεμάχιο άρα συνολικό κόστος $10000 + 40 \cdot n$ όπου n ο αριθμός των τεμαχίων.

Η 3η τεχνολογία έχει σταθερό κόστος 100000€ και μεταβλητό (2+2)€ ανά τεμάχιο άρα συνολικό κόστος $100000 + 4 \cdot n$ όπου n ο αριθμός των τεμαχίων.

Η 4η τεχνολογία έχει σταθερό κόστος 200000€ και μεταβλητό (1+1)€ ανά τεμάχιο άρα συνολικό κόστος $200000 + 2 \cdot n$ όπου n ο αριθμός των τεμαχίων.

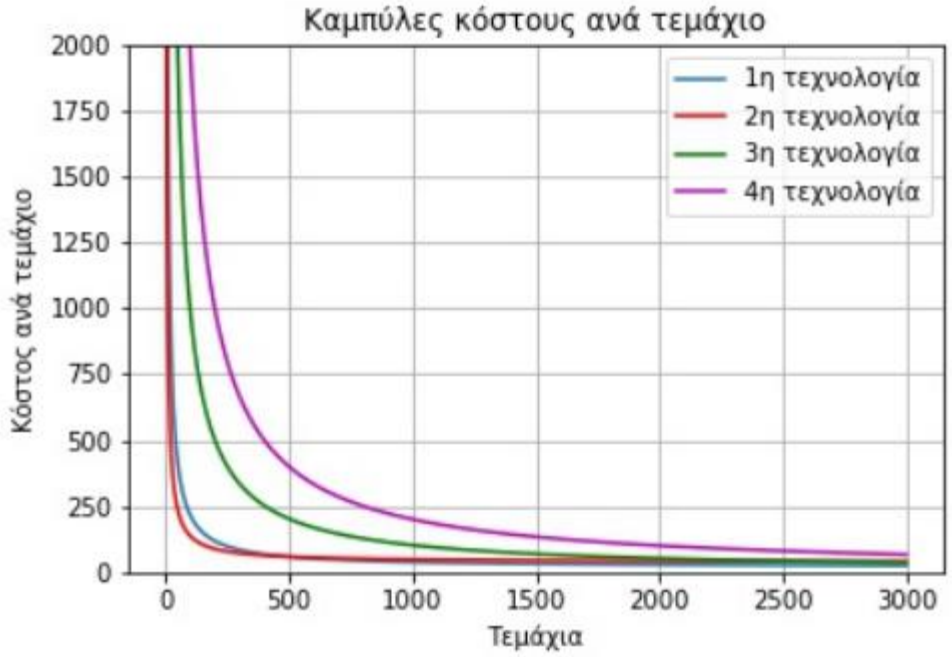
Το κόστος ανά τεμάχιο της κάθε τεχνολογίας θα είναι το συνολικό της κόστος διαιρεμένο με τον αριθμό των τεμαχίων n.

Για τον σχεδιασμό των καμπυλών κόστους ανά τεμάχιο χρησιμοποιήθηκε ο παρακάτω κώδικας σε python:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
n = np.arange(1, 3000, 1)
x1 = (20000 + (10 + 10)*n)/n
x2 = (10000 + (30 + 10)*n)/n
x3 = (100000 + (2 + 2)*n)/n
x4 = (200000 + (1 + 1)*n)/n
plt.figure()
plt.plot(x1)
plt.plot(x2, color = 'r')
plt.plot(x3, color = 'g')
plt.plot(x4, color = 'm')
plt.grid()
plt.ylim(0,2000)
plt.xlabel('Τεμάχια')
plt.ylabel('Κόστος ανά τεμάχιο')
plt.legend(['1η τεχνολογία', '2η τεχνολογία', '3η τεχνολογία', '4η τεχνολογία'])
plt.title('Καμπύλες κόστους ανά τεμάχιο')
plt.savefig("Καμπύλες κόστους.jpeg")
```

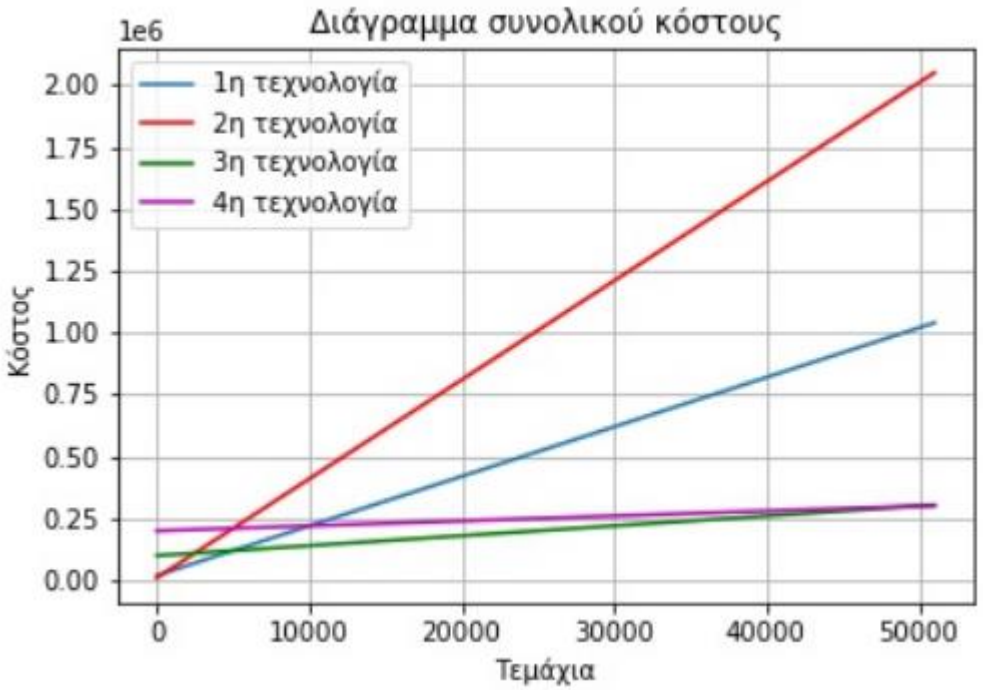
και προέκυψε το παρακάτω διάγραμμα:



Επειδή από το παραπάνω διάγραμμα δεν γίνεται εύκολα κατανοητό σε ποια περιοχή συμφέρει η κάθε τεχνολογία, σχεδιάζουμε, με εντελώς αντίστοιχο τρόπο, και το διάγραμμα συνολικού κόστους της κάθε τεχνολογίας για εξαγωγή ασφαλέστερων συμπερασμάτων (αφού συγκρίνοντας το κόστος ανά τεμάχιο, μεταξύ δύο τεχνολογιών, για συγκεκριμένο αριθμό τεμαχίων, ο αριθμός των τεμαχίων θα απαλειφθεί από τον παρονομαστή και τελικά θα συγκρίνουμε το συνολικό κόστος των τεχνολογιών).

```
import numpy as np
import matplotlib.pyplot as plt
```

```
n = np.arange(0, 51000, 1)
x1 = 20000 + (10 + 10)*n
x2 = 10000 + (30 + 10)*n
x3 = 100000 + (2 + 2)*n
x4 = 200000 + (1 + 1)*n
plt.figure()
plt.plot(x1)
plt.plot(x2, color = 'r')
plt.plot(x3, color = 'g')
plt.plot(x4, color = 'm')
plt.grid()
plt.xlabel('Τεμάχια')
plt.ylabel('Κόστος')
plt.legend(['1η τεχνολογία', '2η τεχνολογία', '3η τεχνολογία', '4η τεχνολογία'])
plt.title('Διάγραμμα συνολικού κόστους')
plt.savefig("Διάγραμμα κόστους.jpeg")
```



Από το τελευταίο διάγραμμα φαίνεται ότι για πολύ μικρό αριθμό τεμαχίων συμφέρει η 2η τεχνολογία (που είναι και λογικό αφού έχει και το μικρότερο σταθερό κόστος), έπειτα συμφέρει περισσότερο η 1η, ύστερα η 3η και για πολύ μεγάλο αριθμό τεμαχίων η 4η.

Με βάση την παραπάνω παρατήρηση επιλύουμε τις κατάλληλες ανισώσεις για να βρούμε τις περιοχές:

Η 2η τεχνολογία συμφέρει όσο: $Κόστος\ 2ης < Κόστος\ 1ης \Rightarrow$

$10000 + 40n < 20000 + 20n \Rightarrow n < 500$ τεμάχια

Η 1η τεχνολογία συμφέρει έναντι της 3ης όσο: $Κόστος\ 1ης < Κόστος\ 3ης \Rightarrow$

$20000 + 20n < 100000 + 4n \Rightarrow n < 5000$ τεμάχια

Η 3η τεχνολογία συμφέρει έναντι της 4ης όσο: $Κόστος\ 3ης < Κόστος\ 4ης \Rightarrow$

$100000 + 4n < 200000 + 2n \Rightarrow n < 50000$ τεμάχια

Άρα η 1η τεχνολογία συμφέρει για 500-5000 τεμάχια, η 2η για 0-500 τεμάχια, η 3η για 5000-50000 τεμάχια και η 4η για 50000 τεμάχια και άνω.

Για να εξαφανιστεί η επιλογή της 1ης τεχνολογίας, αλλάζοντας την τιμή κόστους ανά τεμάχιο I.C. της 2ης τεχνολογίας των FPGAs, θέλουμε η 2η να είναι πιο συμφέρουσα από την 1η τουλάχιστον έως το σημείο όπου η 3η γίνεται πιο συμφέρουσα της 1ης δηλαδή τα 5000 τεμάχια. Άρα αναζητούμε την τιμή του κόστους ανά τεμάχιο I.C. στη δεύτερη τεχνολογία για την οποία η 2η τεχνολογία συμφέρει έναντι της 1ης μέχρι και τα 5000 τεμάχια. Απαιτώντας το κόστος της 1ης τεχνολογίας να είναι ίδιο με το κόστος της 2ης για 5000 τεμάχια βρίσκουμε τη μέγιστη τιμή x ανά τεμάχιο I.C. στην τεχνολογία των FPGA που ικανοποιεί την υπόθεση του ερωτήματος.

$20000 + (10 + 10) \cdot 5000 = 10000 + (x + 10) \cdot 5000 \Rightarrow$

$5000x = 60000 \Rightarrow x = 12€$ ανά τεμάχιο I.C.

5η Άσκηση:

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

```
module circuit_F1 (A, B, C, D, F1);
    output F1;
    input A, B, C, D;
    wire w1, w2, w3, w4, w5, w6;

    and G1 (w1, B, C);
    or G2 (w2, w1, D);              //A(BC + D)
    and G3 (w3, w2, A);

    not G4 (w4, B);
    not G5 (w5, C);                 //B'C'D
    and G6 (w6, w4, w5, D);

    or G7 (F1, w3, w6);
```

```
endmodule
```

//$F2(A,B,C,D) = \Sigma(0,2,3,5,7,9,10,11,13,14)$:

```
primitive truth_table (x, A, B, C, D);
output x;
input A, B, C, D;
    table
    0000:1;
    0001:0;
    0010:1;
    0011:1;
    0100:0;
    0101:1;
    0110:0;
    0111:1;
    1000:0;
    1001:1;
    1010:1;
    1011:1;
    1100:0;
    1101:1;
    1110:1;
    1111:0;
    endtable
endprimitive

module F2_circuit (A, B, C, D, F2)
    output F2;
    input A, B, C, D;
    truth_table (F2, A, B, C, D);
endmodule
```

```
//F3 = ABC + (A + BC)D + (B + C)DE
module circuit_F3 (A, B, C, D, E, F3);
    output F3;
    input A, B, C, D, E;
    wire w1, w2, w3, w4, w5, w6, w7;

    and G1(w1, A, B, C);          //ABC
    and G2 (w2, B, C);
    or G3 (w3, A, w2);
    and G4 (w4, w3, D);           //(A+BC)D

    or G5 (w5, B, C);
    and G6 (w6, w5, D, E);        //(B+C)DE

    or G7 (F3, w1, w4, w6);       //F3
endmodule

//F4 = A(B + CD + E) + BCDE:
module circuit_F4 (A, B, C, D, E, F4);
    output F4;
    input A, B, C, D, E;
    wire w1, w2, w3, w4;

    and G1 (w1, C, D);
    or G2 (w2, B, w1, E);
    and G3 (w3, A, w2);           //A(B+CD+E)

    and G4 (w4, B, C, D, E);      //BCDE

    or G5 (F4, w3, w4);           //F4
```

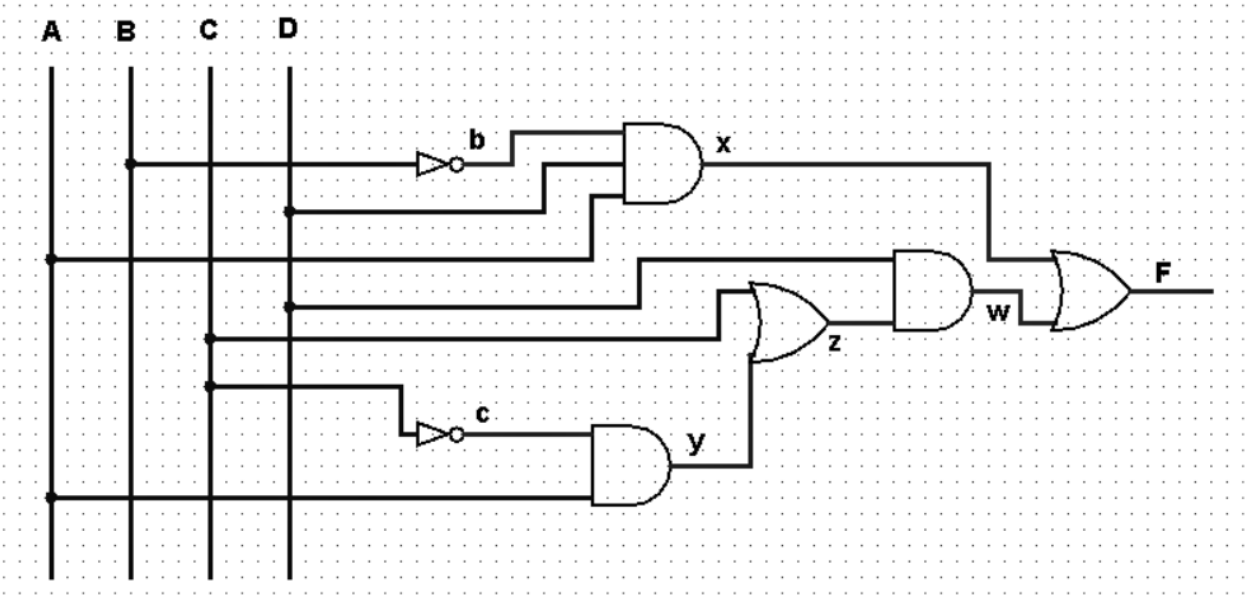
```
endmodule
```

ii)

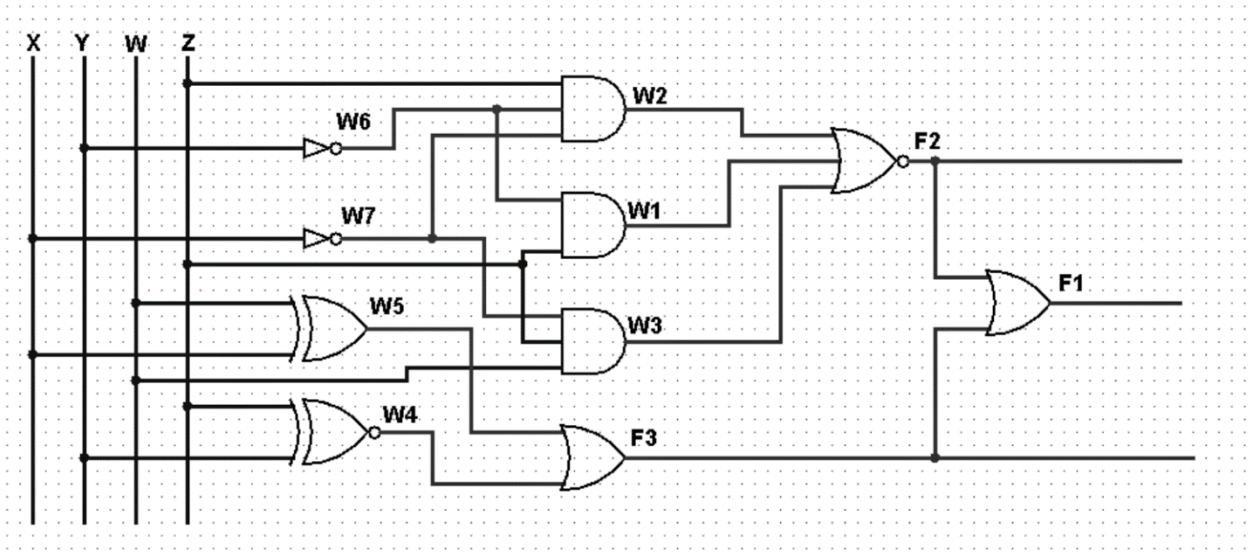
```
//F1:
module circuit_F1 (A, B, C, D, F1);
    output F1;
    input A, B, C, D;
    assign F1 = A&((B&C)|D) | (~B&~C&D);
endmodule

//F2:
module circuit_F2 (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:
module circuit_F3 (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:
module circuit_F4 (A, B, C, D, E, F4);
    output F4;
    input A, B, C, D, E;
    assign F4 = (A&(B | (C&D) | E) | (B&C&D&E);
endmodule
```

6η Άσκηση:

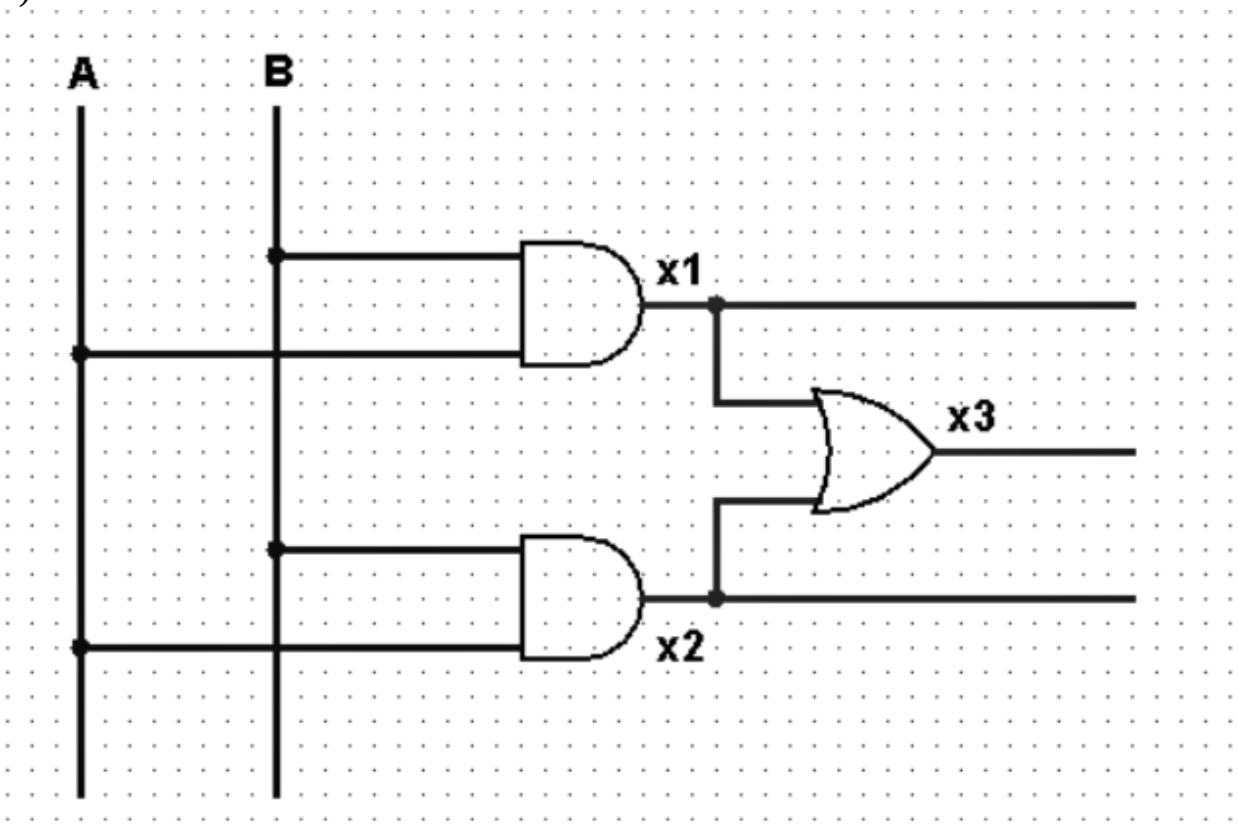
i) a)



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);
               HA2 (S, C2, S1, z);
    or G1 (C, C2, C1);
endmodule

module 4_bit_adder_subtractor (output [3:0] Sum, output C4,
            Overflow, input [3:0] A, B, input M);
    wire C1, C2, C3;
    wire [3:0] selected_B;
```

xor

(selected_B[0], B[0], M),

(selected_B[1], B[1], M),

(selected_B[2], B[2], M),

(selected_B[3], B[3], M);

full_adder

FA0 (Sum[0], C1, A[0], selected_B[0], M),

FA0 (Sum[1], C2, A[1], selected_B[1], C1),

FA0 (Sum[2], C3, A[2], selected_B[2], C2),

FA0 (Sum[3], C4, A[3], selected_B[3], C3);

xor(Overflow, C3, C4);

endmodule

iii) **module** 4_bit_adder_subtractor (**output** [3:0] Sum, **output** Cout, Overflow, **input**[3:0] A, B, **input** M);

assign {Cout, Sum} = (M)? (A-B) : (A+B);

assign Overflow = C3 ^ C4;

endmodule

Άσκηση 7η:

i)

module ask7i (Out, In, clock, reset);

output Out;

reg Out;

input In, clock, reset;

reg [1:0] state;

parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;

always @ (**posedge** clock, **negedge** reset)

```
        if (reset==0) begin state <= a; Out <= 1; end
        else case (state)
            a: if (In) begin state <= a; Out <= 0; end
               else begin state <=d; Out <= 1; end
            b: if (In) begin state <= a; Out <= 0; end
               else begin state <=c; Out <= 1; end
            c: if (In) begin state <= b; Out <= 0; end
               else begin state <=d; Out <= 1; end
            d: if (In) begin state <= d; Out <= 1; end
               else begin state <=c; Out <= 0; end
        endcase
    endmodule
```

ii)

```
    module ask7ii (Out, In, clock, reset);
        output Out;
        input In, clock, reset;
        reg [1:0] 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 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
        case (state)
                a: assign Out = 0;
                b: assign Out = 1;
                b: assign Out = 1;
                d: assign Out = 0;
        endcase
endmodule
```