Συστήματα Μικροϋπολογιστών
1η ομάδα ασκήσεων

Νικόλαος Παγώνας, el18175
Αναστάσιος Παπαζαφειρόπουλος, el18079

1η άσκηση

Για την μετατροπή σε assembly χρησιμοποιήθηκε ο πίνακας 2 του παραρτήματος 2 των σημειώσεων. Πρέπει να προσέξουμε ότι στις διευθύνσεις δίνεται πρώτα το LSByte και ύστερα το MSByte. Για παράδειγμα, όταν σε γλώσσα μηχανής διαβάζουμε **00 30**, αυτό αναφέρεται στην διεύθυνση **3000H**.

Μετατροπή σε assembly:

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

Με συμβολικές διευθύνσεις:

```
START:
        MVI C,08H ; χρήση του C ως μετρητή με αρχική τιμή 8
        LDA 2000H ; φόρτωσε το περιεχόμενο των dip switches στον A
FIRST:
        RAL       ; αριστερή ολίσθηση μέσω κρατουμένου, το MSB του A αποθηκεύεται στο CY
        JC SECOND ; αν το CY είναι 1 τότε πήγαινε να ανάψεις τα κατάλληλα LED
        DCR C     ; αν όχι μείωσε τον μετρητή C κατά 1
        JNZ FIRST ; επανάλαβε την διαδικασία μέχρι να βρεις άσσο
SECOND:
        MOV A,C   ; αποθήκευσε την τιμή του μετρητή στον A
        CMA       ; συμπλήρωμα ως προς 1 του A λόγω αρνητικής λογικής των LED
        STA 3000H ; εμφάνισε το κατάλληλο αποτέλεσμα στα LED
        RST 1
END
```

Για επαναλαμβανόμενη εκτέλεση στον προσομοιωτή

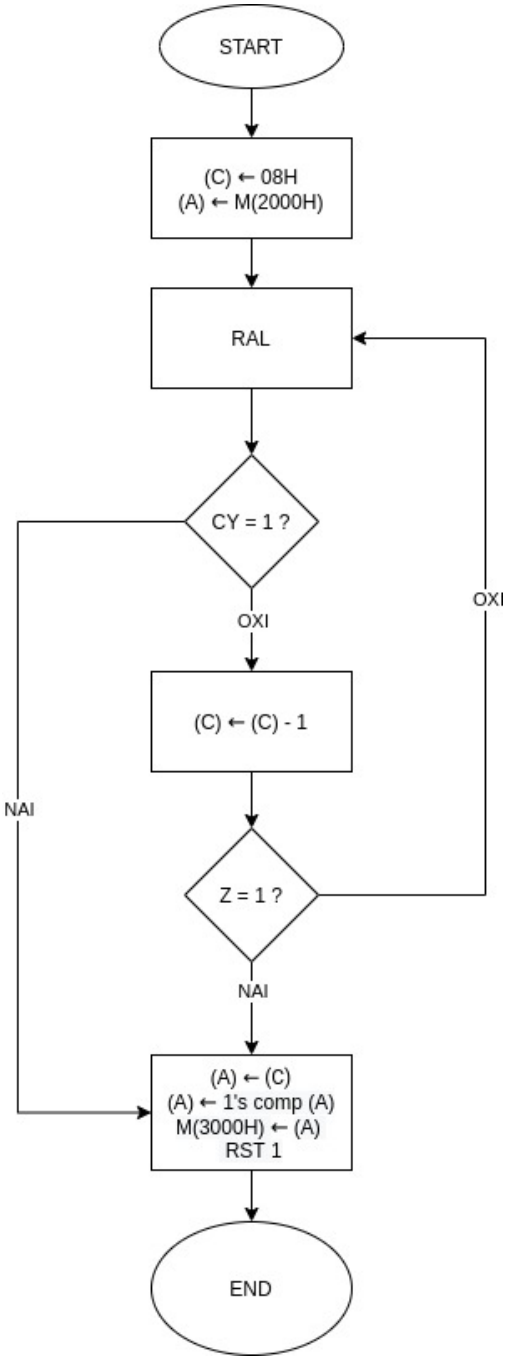
```
START:
        MVI C,08H
        LDA 2000H
FIRST:
        RAL
        JC SECOND
        DCR C
        JNZ FIRST
SECOND:
        MOV A,C
        CMA
        STA 3000H
        JMP START
END
```

Παρατηρώντας προσεκτικά τον κώδικα, αλλά και επιβεβαιώνοντας με τη χρήση του mLAB, συμπεραίνουμε ότι το πρόγραμμα

- παίρνει είσοδο από τα dip switches (2000H)
- βρίσκει τη θέση του πρώτου bit από τα αριστερά που είναι ίσο με 1
- εμφανίζει τη θέση αυτή σε δυαδική μορφή, στα LED εξόδου (3000H)

Για να εκτελείται συνεχώς, χρειάζεται μία εντολή jump η οποία επιστρέφει στην αρχή του προγράμματος.

Το διάγραμμα ροής που αντιστοιχεί στο παραπάνω πρόγραμμα:



2η και 3η άσκηση

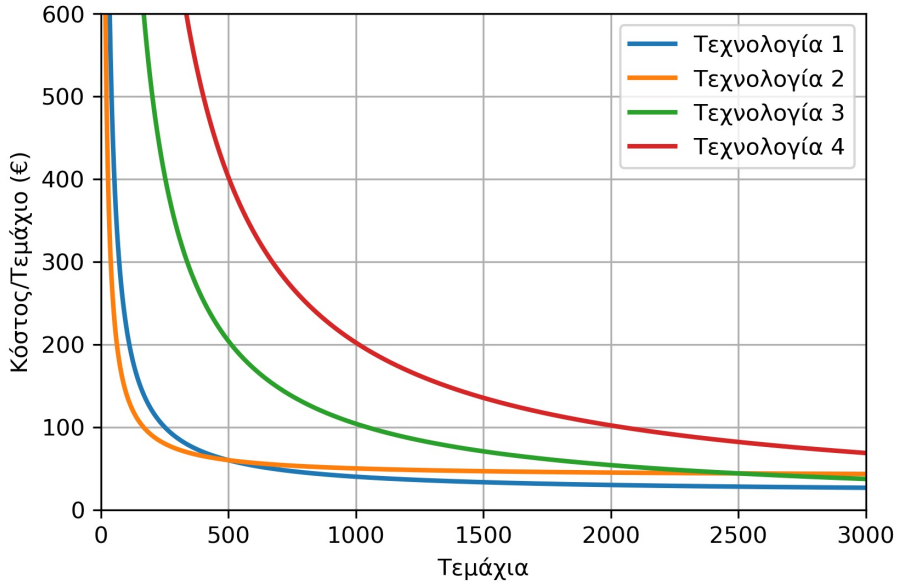
Ο κώδικας και των δύο ασκήσεων βρίσκεται στα αρχεία `ex2.8085` και `ex3.8085` αντίστοιχα, μαζί με τα κατάλληλα σχόλια για την κατανόηση της λειτουργίας των προγραμμάτων.

4η άσκηση

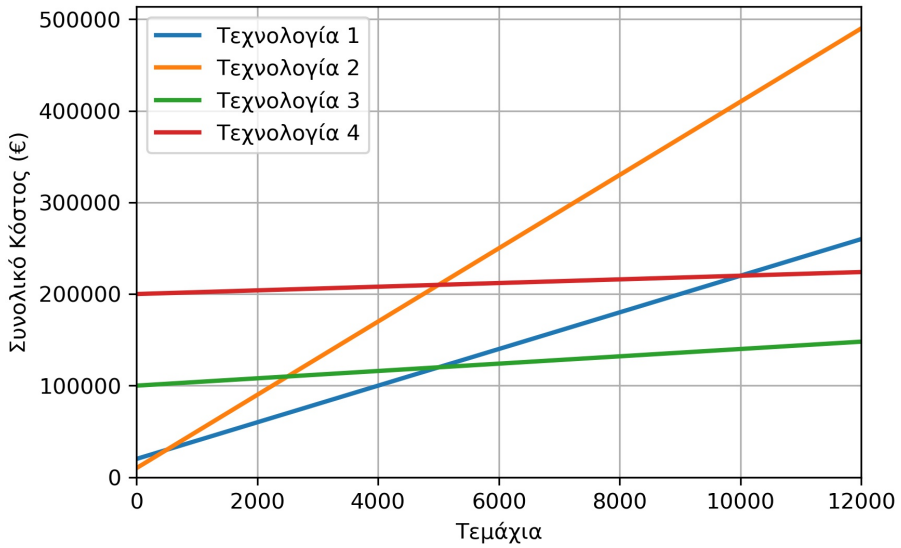
Αν συμβολίσουμε με x τον αριθμό των τεμαχίων:

Τεχνολογία	Συνολικό Κόστος (€)	Κόστος/τεμάχιο (€)
1η	$20000 + 20x$	$\frac{20000}{x} + 20$
2η	$10000 + 40x$	$\frac{10000}{x} + 40$
3η	$100000 + 4x$	$\frac{100000}{x} + 4$
4η	$200000 + 2x$	$\frac{200000}{x} + 2$

Απεικονίζουμε τις καμπύλες κόστους ανά τεμάχιο:



Και τις καμπύλες συνολικού κόστους:



Επίσης, χρησιμοποιώντας τις εξισώσεις των ευθειών που δίνουν το συνολικό κόστος για κάθε περίπτωση, βρίσκουμε τα σημεία τομής τους, και άρα ποιο κόστος είναι ελάχιστο στο εκάστοτε διάστημα. Προκύπτουν τα εξής:

- $0 < x < 500$: Ευνοϊκότερη η 2η τεχνολογία.
- $500 < x < 5000$: Ευνοϊκότερη η 1η τεχνολογία.
- $5000 < x < 100000$: Ευνοϊκότερη η 3η τεχνολογία.
- $x > 100000$: Ευνοϊκότερη η 4η τεχνολογία.

Για να εξαφανιστεί η επιλογή της πρώτης τεχνολογίας, πρέπει η 2η τεχνολογία να είναι ευνοϊκότερη ακόμα και στην περιοχή 500 με 5000. Συμβολίζοντας το υπό διερεύνηση κόστος των I.C. της 2ης τεχνολογίας με k:

$$20000 + 20x > 10000 + (k + 10)x$$

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

Επειδή το παραπάνω πρέπει να ισχύει για $x \in [500, 5000]$, έχουμε:

- Για $x = 500, k < 30$
- Για $x = 5000, k < 12$

Επομένως συνολικά, πρέπει το κόστος I.C. ανά τεμάχιο να είναι μικρότερο από 12 ευρώ προκειμένου να εξαφανιστεί η 1η τεχνολογία.

5η άσκηση

```
//==========GATE LEVEL IMPLEMENTATION==========//


//----------------------F1----------------------//

module Circuit_F1 (A,B,C,D,F1);
  output F1;
  input A,B,C,D;

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

  not
    (Bnot, B),
    (Cnot, C);
  and
    (w1, B, C),
    (w3, Bnot, Cnot, D);
  or
    (w2, w1, D);
  and
    (w4, A, w2);
  or
    (F1, w4, w3);
endmodule

//----------------------F2----------------------//

module Circuit_F2 (A,B,C,D,F2);
  output F2;
  input A,B,C,D;

wire Anot, Bnot, Cnot, Dnot, m0, m2, m3, m5, m7, m9, m10, m11, m13,
     m14;

not
  (Anot, A),
  (Bnot, B),
  (Cnot, C),
  (Dnot, D);

and
  (m0,  Anot, Bnot, Cnot, Dnot),
  (m2,  Anot, Bnot, C, Dnot),
  (m3,  Anot, Bnot, C, D),
  (m5,  Anot, B, Cnot, D),
  (m7,  Anot, B, C, D),
  (m9,  A, Bnot, Cnot, D),
  (m10, A, Bnot, C, Dnot),
  (m11, A, Bnot, C, D),
  (m13, A, B, Cnot, D),
  (m14, A, B, C, Dnot);

or
  (F2, m0, m2, m3, m5, m7, m9, m10, m11, m13, m14);

endmodule

//----------------------F2 using truth table----------------------//

primitive table_F2 (A, B, C, D, F2);
```

```
  output F2;
  input A, B, C, D;
  table
  // A B C D : F2
     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;
  endtable
endprimitive

//----------------------F3----------------------//


module Circuit_F3 (A,B,C,D,E,F3);
  output F3;
  input A, B, C, D, E;

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

  and (w1, A, B, C);
  and (w2, B, C);
  or (w3, A, w2);
  and (w4, w3, D);
  or (w5, B, C);
  and (w6, w5, D, E);
  or (F3, w1, w4, w6);
endmodule

//----------------------F4----------------------//

module Circuit_F4 (A,B,C,D,E,F4);
  output F4;
  input A, B, C, D, E;

  wire

  and (w1, C, D);
  or (w2, B, w1, E);
  and (w3, A, w2);
  and (w4, B, C, D, E);
  or (F4, w3, w4);
endmodule

//==========DATA FLOW IMPLEMENTATION==========//

//----------------------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)

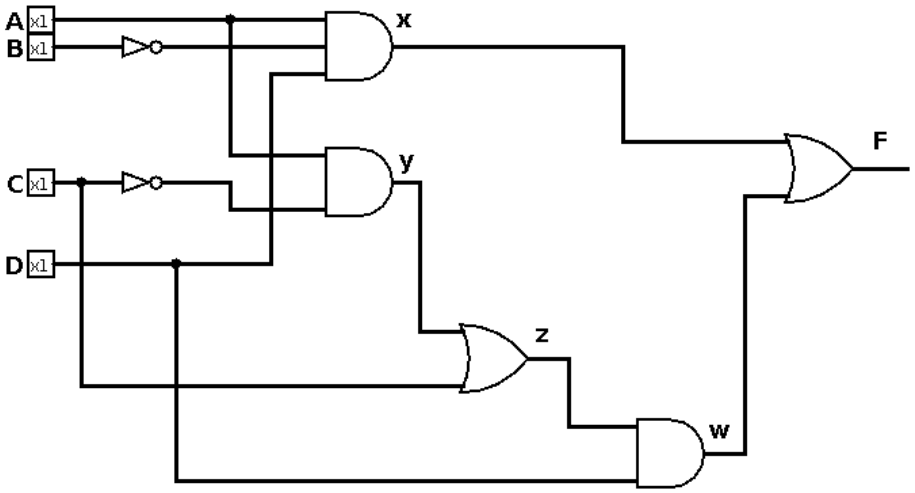


Figure 1: Circuit A

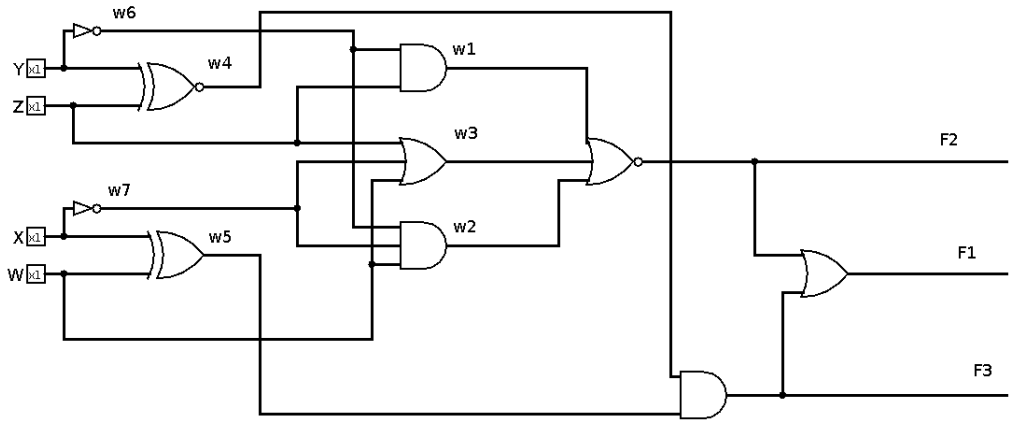


Figure 2: Circuit B

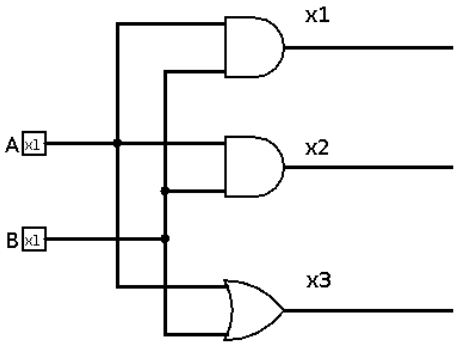


Figure 3: Circuit C

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, C2, 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, input
    [3:0] A, B, input M);
  wire C1, C2, C3;
  wire w0, w1, w2, w3;

  xor
    (w0, B[0], M),
    (w1, B[1], M),
    (w2, B[2], M),
    (w3, B[3], M);

  full_adder
    FA0(Sum[0], C1, A[0], B[0], C0),
    FA1(Sum[0], C1, A[0], B[0], C0),
    FA2(Sum[0], C1, A[0], B[0], C0),
    FA3(Sum[0], C1, A[0], B[0], C0);
endmodule
```

iii)

```
module 4_bit_adder_subtractor (output [3:0] Sum, output C4, input
    [3:0] A, B, input M);

  assign {C4, Sum} = M ? (A+(~B)+1) : (A+B);
endmodule
```

7η άσκηση

Κωδικοποιούμε τις καταστάσεις των αυτομάτων ως εξής:

$$a \to 00$$

$$b \to 01$$

$$c \to 10$$

$$d \to 11$$

Η είσοδος συμβολίζεται με x και η έξοδος με y. Το MSB και το LSB της κατάστασης με A και B αντίστοιχα. Έτσι, έξοδος y για το αυτόματο Mealy είναι η εξής:

A	B	x	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Η απλοποίηση με χάρτη Karnaugh δίνει: $y = ABx + A'x' + B'x'$

Αντίστοιχα, για την έξοδο του αυτομάτου Moore ισχύει:

A	B	y
0	0	0
0	1	1
1	0	1
1	1	0

Έχουμε $y = AB' + A'B = A$ XOR B

Έτσι, η υλοποίηση σε Verilog για τα δύο αυτόματα είναι η εξής:

```
//--------------------Mealy automaton------------------//

module Mealy (y, x, clock, reset);
  output [1:0] y;
  input x, 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; // Begin at a
    else case (state)
      a: if (x) state <= a; else state <= d;
      b: if (x) state <= a; else state <= c;
      c: if (x) state <= b; else state <= d;
      d: if (x) state <= d; else state <= c;
    endcase
  assign y = (state[1] & state[0] & x) | (~state[1] & ~x) | (~state
    [0] & ~x);
endmodule

//--------------------Moore automaton------------------//

module Moore (y, x, clock, reset);
  output [1:0] y;
  reg [1:0] state;
  parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;
```

```
25   always @ (posedge clock, negedge reset)
26     if (reset == 0) state <= a; // Begin at a
27     else case (state)
28       a: if (x) state <= a; else state <= d;
29       b: if (x) state <= a; else state <= c;
30       c: if (x) state <= d; else state <= b;
31       d: if (x) state <= d; else state <= c;
32     endcase
33   assign y = (state[1] & ~state[0]) | (~state[1] & state[0]);
34   // or we can use XOR
35 endmodule
```