ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ

1η Εργαστηριακή Άσκηση

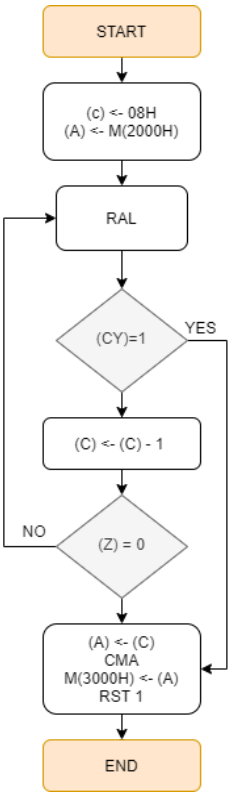
Ακαδημαϊκό έτος 2020-2021

Χριστίνα Προεστάκη | ΑΜ : 03118877

Νικόλαος Μπέλλος | ΑΜ : 03118183

1η Άσκηση

Κώδικας ASSEMBLY	Κώδικας στον προσομοιωτή
MVI C,08H LDA 2000H RAL **JC** 080DH DCR C **JNZ** 0805H MOV A,C CMA STA 3000H RST 1	START: MVI C,08H LDA 2000H GOTO_LOOP: RAL JC GOTO_DISPLAY DCR C JNZ GOTO_LOOP GOTO_DISPLAY: MOV A,C CMA STA 3000H JMP START END



Για να επιτευχθεί η συνεχής λειτουργία του παραπάνω προγράμματος αρκεί να προσθέσουμε την εντολή JMP START πριν το τέλος του προγράμματος (END), όπου η ετικέτα START θα βρίσκεται στην αρχή. Έτσι, θα δημιουργήσουμε ένα συνεχές loop.

2η Άσκηση

```
      IN  10H          ;for memory protection
      LXI B,01F4H      ;insert 500ms delay to B
      MVI E,01H        ;value of initial led pattern
START:
      LDA 2000H        ;load switches to A
      MOV D,A          ;load A to D
      RRC              ;get value of LSB
      JC START         ;if LSB in ON start over
      CALL DELB        ;500ms delay
      MOV A,D          ;load D to A
      RLC              ;get value of MSB
      JC GORIGHT
GOLEFT:
      MOV A,E          ;load E to A
      CMA              ;reverse A for LEDs
      STA 3000H        ;load value A to LEDs
      CMA              ;reverse A back
      RLC              ;rotate left LEDs
      MOV E,A          ;store state of LEDs to E
      JMP START        ;start over
GORIGHT:
      MOV A,E          ;load state of LEDs to A
      CMA              ;reverse A for LEDs
      STA 3000H
      CMA              ;reverse A back
      RRC              ;rotate right LEDs
      MOV E,A          ;store state of LEDs to E
      JMP START        ;start over

      END
```

3η Άσκηση

```
      LXI B,01F4H      ;insert 500ms delay to B
START:
      LDA 2000H        ;load switches to A
      CPI C8H          ;compare A to 200
      JNC TOOBIG       ;if A > 199 goto TOOBIG
      CPI 64H          ;compare A to 100
      JNC BIG          ;if A > 99 goto BIG
      MVI D,FFH        ;initial value of counter
DECA:
      INR D            ;D = D + 1
      SUI 0AH          ;A = A - 10
      JNC DECA         ;if A > 0 continue
      ADI 0AH          ;if A < 0 correct negative
      MOV E,A          ;load units (A) to E
      MOV A,D          ;load decimal units (D) to A
      RLC              ;move decimal units 4 bits to left
      RLC
      RLC
      RLC
      ADD E            ;add units to decimal
      CMA              ;reverse A for LEDs
      STA 3000H        ;load value A to LEDs
      JMP START
BIG:
      MVI A,F0H        ;load value of 4 LSB LEDs to A
      STA 3000H        ;load A to LEDs
      CALL DELB        ;delay 500ms
      MVI A,FFH        ;load value of blank LEDs to A
      STA 3000H        ;load A to LEDs
      CALL DELB        ;delay 500ms
      JMP START        ;start over
TOOBIG:
      MVI A,0FH        ;load value of 4 MSB LEDs to A
      STA 3000H        ;load A to LEDs
      CALL DELB        ;delay 500ms
      MVI A,FFH        ;load value of blank LEDs to A
      STA 3000H        ;load A to LEDs
      CALL DELB        ;delay 500ms
      JMP START        ;start over

      END
```

Θεωρητικές Ασκήσεις

4η Άσκηση

Αρχικά υπολογίζουμε τη συνάρτηση κόστους της κάθε τεχνολογίας ανά τεμάχιο.

$K1(x) = 20000 + 20/x$

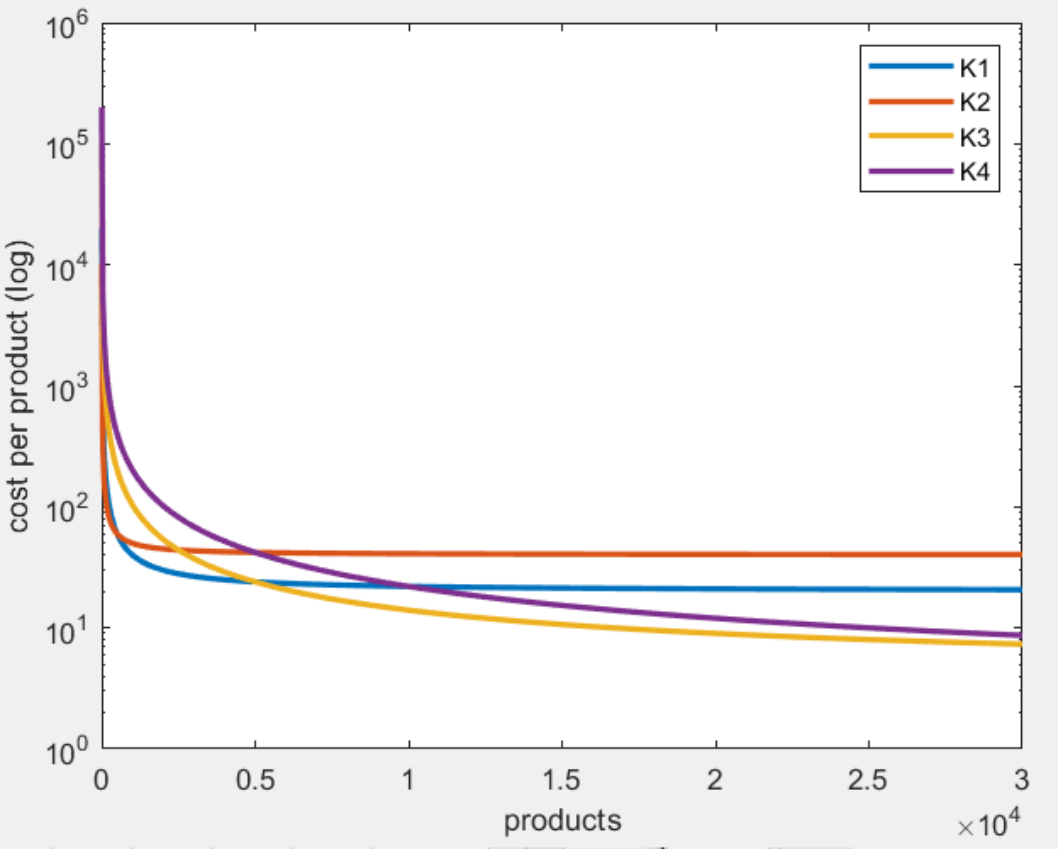
$K2(x) = 10000 + 40/x$

$K3(x) = 100000 + 4/x$

$K4(x) = 200000 + 2/x$

,όπου x ο αριθμός των τεμαχίων.

Η συνάρτηση κόστους ανά τεμάχιο αναπαρίσταται στο παρακάτω διάγραμμα.



Για να συγκρίνουμε τις καμπύλες και συνεπώς τις τεχνολογίες, βρίσκουμε τα σημεία τομής των καμπυλών.

Ζεύγος Καμπυλών	**Σημείο Τομής (x)**
K1-K2	500
K2-K3	2500

K1-K3	5000
K2-K4	5000
K1-K4	10000
K3-K4	50000

Παρατηρώντας το διάγραμμα, βρίσκουμε ποια τεχνολογία έχει το χαμηλότερο κόστος στα εύρη τιμών που ορίζουν τα σημεία τομής.

x	Συμφέρουσα Τεχνολογία
(0, 500)	2
(500, 5000)	1
(5000, 10000)	3
(10000, 50000)	4

Έστω ότι για τιμή κόστους κ η 2η τεχνολογία είναι πιο συμφέρουσα από την 1η τεχνολογία.
K2 < K1 => 10000 + (κ+10)x/x < 20000 + 20/x => κ < 10000/x + 10 \forall x > 0
Επομένως πρέπει κ-10 < 0 => κ < 10.

Ασκήσεις στη Γλώσσα Περιγραφής Υλικού Verilog

5η Άσκηση

(i) Δομική Περιγραφή:

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

```
module F1(A, B, C, D);
    output F1;
    input A, B, C, D;
    wire w1, w2, w3, w4, Bnot, Cnot;

    and G1(w1, B, C);
    or G2(w2, w1, D);
    and G3(w3, w2, A);
    not G4(Bnot, B);
    not G5(Cnot, C);
    and G6(w4, Bnot, Cnot, D);
    or G7(F1, w4, w3);
endmodule
```

- F2(A, B, C, D)=Σ (0, 2, 3, 5, 7, 9, 10, 11, 13, 14)

```
primitive UDP_02467(A, B, C, D, F2);
    output F2;
    input A, B, C, D;
    table
//  A B C D : x
    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=ABC + (A + BC)D+(B+ C)DE

```
module F3(A, B, C, D, E, F3);
    output F3;
    input A, B, C, D, E;
    wire w1, w2, w3, w4, w5;

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

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

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

```
module 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, w2, A);
    and G4(w4, B, C, D, E);
    or G4(F4, w4, w3);
endmodule
```

(ii) Μοντελοποίηση Ροής Δεδομένων:

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

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

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

endmodule;
```

- F2(A, B, C, D) = Σ (0, 2, 3, 5, 7, 9, 10, 11, 13, 14)

```
primitive UDP_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);
endprimitive
```

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

```
module 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 = A(B + CD + E) + BCDE

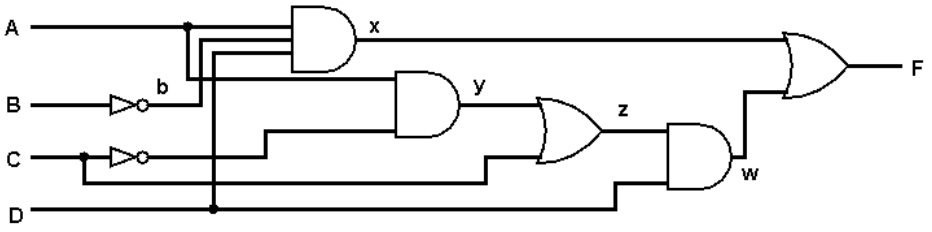
```
module F4(A, B, C, D, E, F4);
    ouput F4;
    input A, B, C, D, E;

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

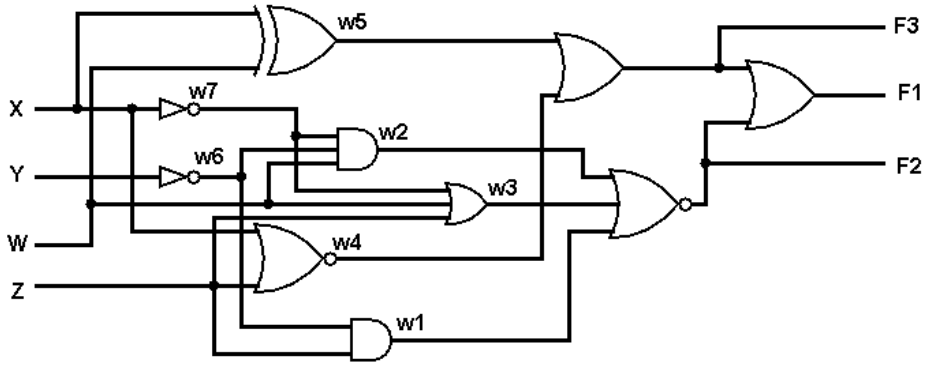
6η Άσκηση

(i)

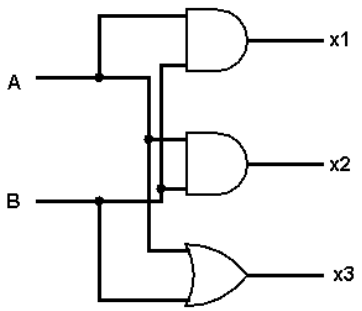
(a)



(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);
  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 M);
  wire C1, C2, C3;
  wire w0, w1, w2, w3;
  xor G1(w0, B[0], M);
  xor G2(w1, B[1], M);
  xor G3(w2, B[2], M);
  xor G4(w3, B[3], M);
  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)

```
module _4_bit_add_sub(output [3:0] S, output C, V, input [3:0] A,
B, input M);
  assign {C, S} = M ? A - B : A + B;
endmodule
```

Άσκηση 7η

(i)

```
module Mealy_Machine (
      output reg y_out,
      input x_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)
            if (reset == 0) state <= a;
            else state <= next_state;
      always @ (state, x_in)
            case (state)
                  a: if (x_in) next_state = a; else next_state = d;
                  b: if (x_in) next_state = a; else next_state = c;
                  c: if (x_in) next_state = b; else next_state = d;
                  d: if (x_in) next_state = d; else next_state = c;
            endcase
      always @ (state, x_in)
            case (state)
                  a, b, c: y_out = ~x_in;
                  d: y_out = x_in;
      endcase
endmodule
```

(ii)

```
module Moore_Machine (
      output y_out,
      input x_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 (x_in) state <= a; else state <= d;
                b: if (x_in) state <= a; else state <= c;
                c: if (x_in) state <= d; else state <= b;
                d: if (x_in) state <= d; else state <= c;
          endcase
     always @ (state)
          case (state)
                a, d: y_out <= 1'b0;
                b, c: y_out <= 1'b1;
          endcase
endmodule
```