




Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

6^ο Εξάμηνο

Συστήματα Μικροϋπολογιστών

1^η ΟΜΑΔΑ ΑΣΚΗΣΕΩΝ


Χατζή Ήβη

Ασκήσεις προσομοίωσης

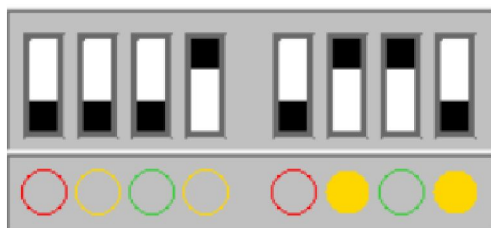
1^η ΑΣΚΗΣΗ:

Το πρόγραμμα που δίνεται σε assembly:

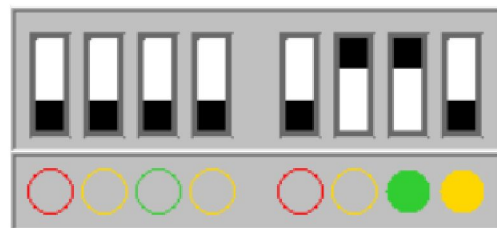
```
MVI C, 08H
LDA 2000H
RAL
JC 080DH
DCR C
JNZ 0805H
MOV A, C
CMA
STA 3000H
RST 1
```

Το πρόγραμμα διαβάζει τους διακόπτες εισόδου και εμφανίζει στα LED εξόδου τη θέση του σημαντικότερου 1 που υπάρχει στην είσοδο. Η έξοδος φαίνεται σε δυαδική μορφή:

Σημαντικότερο 1 στη θέση 5



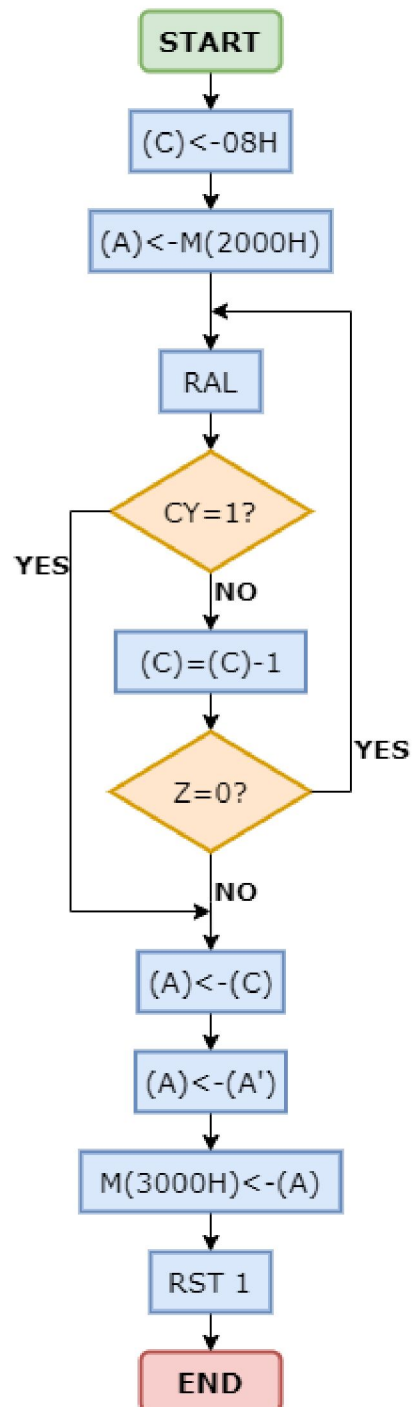
Σημαντικότερο 1 στη θέση 3



Για να επαναλαμβάνεται συνεχώς το πρόγραμμα αρκεί να προσθέσουμε μια εντολή `JMP` στο τέλος που θα πηδάει στην αρχή του προγράμματος (`C3 00 08` σε γλώσσα μηχανής).

```
START:
    MVI C, 08H
    LDA 2000H
JUMP1:
    RAL
    JC JUMP2
    DCR C
    JNZ JUMP1
JUMP2:
    MOV A, C
    CMA
    STA 3000H
    JMP START
END
```

Διάγραμμα ροής του προγράμματος:



2^η ΑΣΚΗΣΗ:

```
IN 10H
LXI B,01F4H      ;delay 01F4H=500ms=1/2sec
MVI E,01H        ;start at LSB LED, store it in E
MOV A,E
CMA
STA 3000H
CMA
```

```
PAUSE:
  LDA 2000H
  RRC          ;isolate LSB switch
  CALL DELB    ;delay
  JNC PAUSE    ;if LSB is 0 pause
  RLC
  RLC          ;isolate MSB switch
  JC RIGHT    ;if MSB is 1 shift right
```

```
LEFT:
  MOV A,E      ;get previous LED from E
  RLC          ;shift left
  CMA
  STA 3000H
  CMA
  MOV E,A      ;store new LED in E
  JMP PAUSE
```

```
RIGHT:
  MOV A,E      ;get previous LED from E
  RRC          ;shift right
  CMA
  STA 3000H
  CMA
  MOV E,A      ;store new LED in E
  JMP PAUSE
```

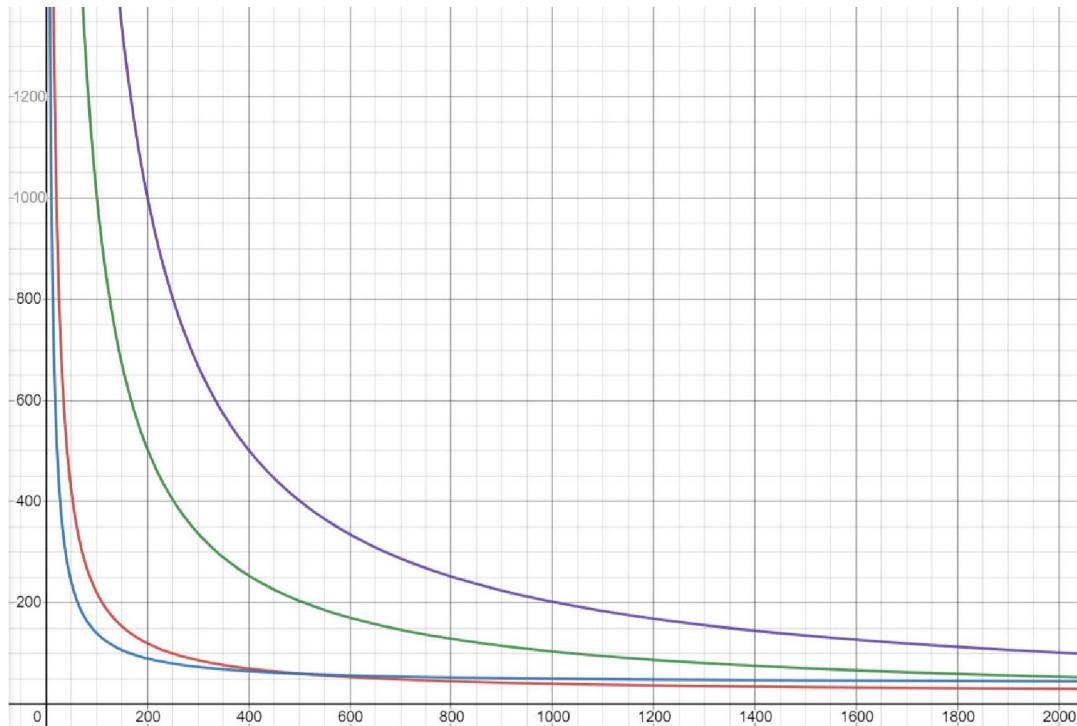
END

3^η ΑΣΚΗΣΗ:

```
IN 10H
START:   LXI B,01F4H      ;Delay 500ms
        LDA 2000H ;input
        CPI C7H   ;compare with 199 and jump if bigger
        JNC BLONK
        CPI 63H   ;compare with 99 and jump if bigger
        JNC BLINK
        MVI D,00H ;D holds the 10s digit, start with 0
DEC:     CPI 0AH   ;compare with 10
        JC  DONE  ;if smaller then it is the last digit
        SUI 0AH   ;else subtract 10
        INR D     ;increase 10s by one
        JMP DEC
DONE:    MOV E,A   ;last digit to E
        MOV A,D   ;10s digit to A
        RRC      ;rotate 4 times to put 10s digit in the 4 msb of A
        RRC
        RRC
        RRC
        ADD E
        CMA      ;output lamps have negative logic so invert A
        STA 3000H ;output
        JMP START
BLINK:   MVI A,F0H ;number between 100 and 199, 4 lsb lamps blink
        STA 3000H ;output 0000 1111 (inverse of A just set)
        CALL DELB ;delay
        MVI A,FFH
        STA 3000H ;output 0000 0000 (inverse of A just set)
        CALL DELB ;delay
        JMP START
BLONK:   MVI A,0FH  ;number over 199, 4 msb lamps blink
        STA 3000H ;output 1111 0000 (inverse of A just set)
        CALL DELB ;delay
        MVI A,FFH
        STA 3000H ;output 0000 0000
        CALL DELB ;delay
        JMP START
END
```

Θεωρητικές Ασκήσεις: 4^η ΑΣΚΗΣΗ

Γραφική παράσταση κόστους ανά τεμάχιο για τις 4 τεχνολογίες:



1  $y_1 = (20000 + 20 \cdot x)/x$

2  $y_2 = (10000 + 40 \cdot x)/x$

3  $y_3 = (100000 + 4 \cdot x)/x$

4  $y_4 = (200000 + 2 \cdot x)/x$

Για την 1^η τεχνολογία συμφέρει η περιοχή από 500 μέχρι 5000 τεμάχια.

Για τη 2^η τεχνολογία συμφέρει η περιοχή από 1 μέχρι 500 τεμάχια.

Για την 3^η τεχνολογία συμφέρει η περιοχή από 5000 μέχρι 50000 τεμάχια.

Για την 4^η τεχνολογία συμφέρει η περιοχή από 50000 τεμάχια και πάνω

Για να εξαφανιστεί η επιλογή της 1^{ης} τεχνολογίας θέλουμε η τιμή της 2^{ης} να είναι χαμηλότερη μέχρι τα 5000 τεμάχια, όπου συμφέρει πλέον η 3^η τεχνολογία.

Για $x = 5000$ τεμάχια το κόστος είναι $y_1 = 120000\text{€}$

Άρα για αρχικό κόστος 10000 για τη 2^η τεχνολογία παίρνουμε

$$120000 = 10000 + 5000a \Rightarrow a = 22\text{€}$$

Άρα η νέα εξίσωση της 2^{ης} τεχνολογίας είναι $y_2' = 10000 + 22x$ οπότε το κόστος ανά τεμάχιο FPGA πρέπει να είναι 12€.

Ασκήσεις στη Γλώσσα Περιγραφής Υλικού Verilog

5^η ΑΣΚΗΣΗ:

(i) Περιγραφή σε επίπεδο πυλών (δομική) των λογικών συναρτήσεων:

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

```
module ex5i_1(A, B, C, D, F1);  
    output F1;  
    input A, B, C, D;  
    wire Bnot, Cnot, w1, w2, w3, w4;  
    and (w1, B, C);  
    or (w2, w1, D);  
    and (w3, A, w2);  
    and (w4, Bnot, Cnot, D);  
    or (F, w3, w4);  
    not (Bnot, B);  
    not (Cnot, C);  
endmodule
```

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

```
module ex5i_2(A, B, C, D, F2);  
    output F2;  
    input A, B, C, D;  
    wire w0, w2, w3, w5, w7, w9, w10, w11, w13, w14;  
    not  
        (Anot, A),  
        (Bnot, B),  
        (Cnot, C),  
        (Dnot, D);  
    and  
        (w0, Anot, Bnot, Cnot, Dnot),  
        (w2, Anot, Bnot, C, Dnot),  
        (w3, Anot, Bnot, C, D),  
        (w5, Anot, B, Cnot, D),  
        (w7, Anot, B, C, D),  
        (w9, A, Bnot, Cnot, D),  
        (w10, A, Bnot, C, Dnot),  
        (w11, A, Bnot, C, D),  
        (w13, A, B, Cnot, D),  
        (w14, A, B, C, Dnot);  
  
    or (F2, w0, w2, w3, w5, w7, w9, w11, w13, w14);  
endmodule
```

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

```

module ex5i_3(A, B, C, D, E, F3);
  output F3;
  input A, B, C, D, E;
  wire w1, w2, w3, w4, w5, w6, w7;
  and (w1, A, B, C);
  and (w2, B, C);
  or (w3, A, w2);
  and (w4, w3, D);
  and (w6, D, E);
  or (w5, B, C);
  and (w7, w5, w6);
  or (F3, w1, w4, w7);
endmodule

```

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

```

module ex5i_4(A, B, C, D, E, F4);
  output F4;
  input A, B, C, D, E;
  wire w1, w2, w3, w4;
  and (w1, D, C);
  or (w2, B, w1, E);
  and (w3, A, w2);
  and (w4, B, C, D, E);
  or (F4, w3, w4);
endmodule

```

(ii) Κώδικας Verilog για τις συναρτήσεις αυτές σε μοντελοποίηση ροής δεδομένων.

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

```

module ex5ii_1(A, B, C, D, F1);
  output F1;
  input A, B, C, D;
  assign F1 = (A & ((B & C) | D)) | (~B & ~D & D);
endmodule

```


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

```

module ex5ii_2(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

```

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

```

module ex5ii_3(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

```

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

```

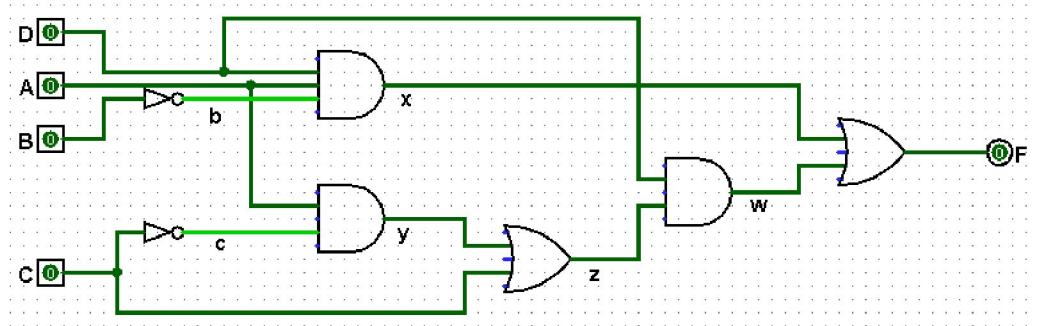
module ex5ii_4(A, B, C, D, E, F4);
  output F4;
  input A, B, C, D, E;
  assign F4=(A&(B|(C&D)|E))|(B&C&D&E);
endmodule

```

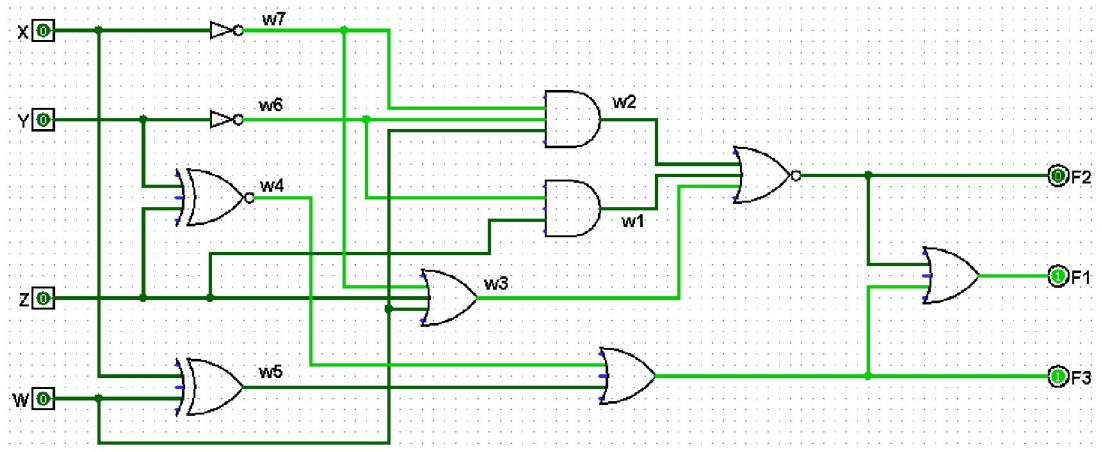
6^η ΑΣΚΗΣΗ:

(i) Λογικό διάγραμμα των ψηφιακών κυκλωμάτων που ορίζονται.

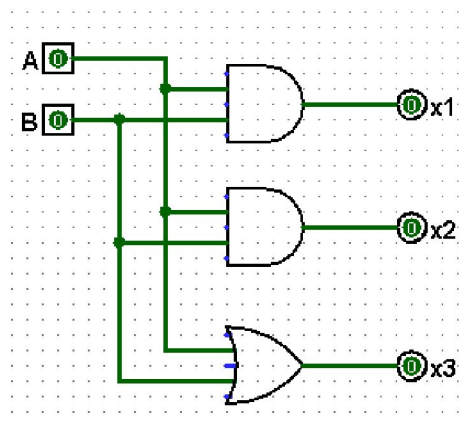
a)



b)



c)



(ii) Ιεραρχική περιγραφή HDL σε επίπεδο πύλης για έναν 4-bit αθροιστή-αφαιρέτη για μη προσημασμένους δυαδικούς αριθμούς.

```
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, input [3:0]A, B, input C0);  
    wire C1, C2, C3, [0:3]X //Intermediate Carriers and xor output.  
    xor  
        (X[0], B[0], C0),  
        (X[1], B[1], C0),  
        (X[2], B[2], C0),  
        (X[3], B[3], C0);  
    //instantiate chain of full adders.  
    full_adder FA0(Sum[0], C1, A[0], X[0], C0),  
               FA1(Sum[1], C2, A[1], X[1], C1),  
               FA2(Sum[2], C3, A[2], X[2], C2),  
               FA3(Sum[3], C4, A[3], X[3], C3);  
endmodule
```

(iii) Περιγραφή ροής δεδομένων HDL ενός 4-bit αθροιστή-αφαιρέτη μη προσημασμένων αριθμών, χρησιμοποιώντας τον τελεστή υπό συνθήκη (? :).

```
module 4_bit_adder_subtractor(output [3:0]Sum, output C4, input [3:0]A, B, input C0);  
    assign {C4, Sum}=C0 ? (A-B) : (A+B);  
endmodule
```

7^η ΑΣΚΗΣΗ:

(i) Mealy FSM

```
module Mealy (y, x, clock, reset);  
    output y;  
    input x, clock, reset;  
    reg [1:0] state;  
    reg y;  
    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) state <= d; else state <= a;  
            b: if (~x) state <= c; else state <= a;  
            c: if (~x) state <= d; else state <= b;  
            d: if (~x) state <= c; else state <= d;  
        endcase  
    always@ (state, x)  
        case(state)  
            a, b, c: y = ~x;  
            d: y = x;  
        endcase  
endmodule
```

(ii) Moore FSM

```
module Mealy (y, x, clock, reset);  
    output y;  
    input x, clock, reset;  
    reg [1:0] state;  
    reg y;  
    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) state <= d; else state <= a;  
            b: if (~x) state <= c; else state <= a;  
            c: if (~x) state <= b; else state <= d;  
            d: if (~x) state <= c; else state <= d;  
        endcase
```

```
always@ (state)
  case(state)
    a, d: y = 1'b0;
    b, c: y = 1'b1;
  endcase
endmodule
```