ΣΗΜΜΥ ΕΜΠ
Συστήματα Μικροϋπολογιστών / 1η ομάδα ασκήσεων
Νικόλαος Πηγαδάς / el18445
2020-2021

Άσκηση 1η

μΥ-Σ 8085

0E 08 **3A** 00 20 **17 DA** 0D 08 **0D C2** 05 08 **79 2F 32** 00 30 **CF**

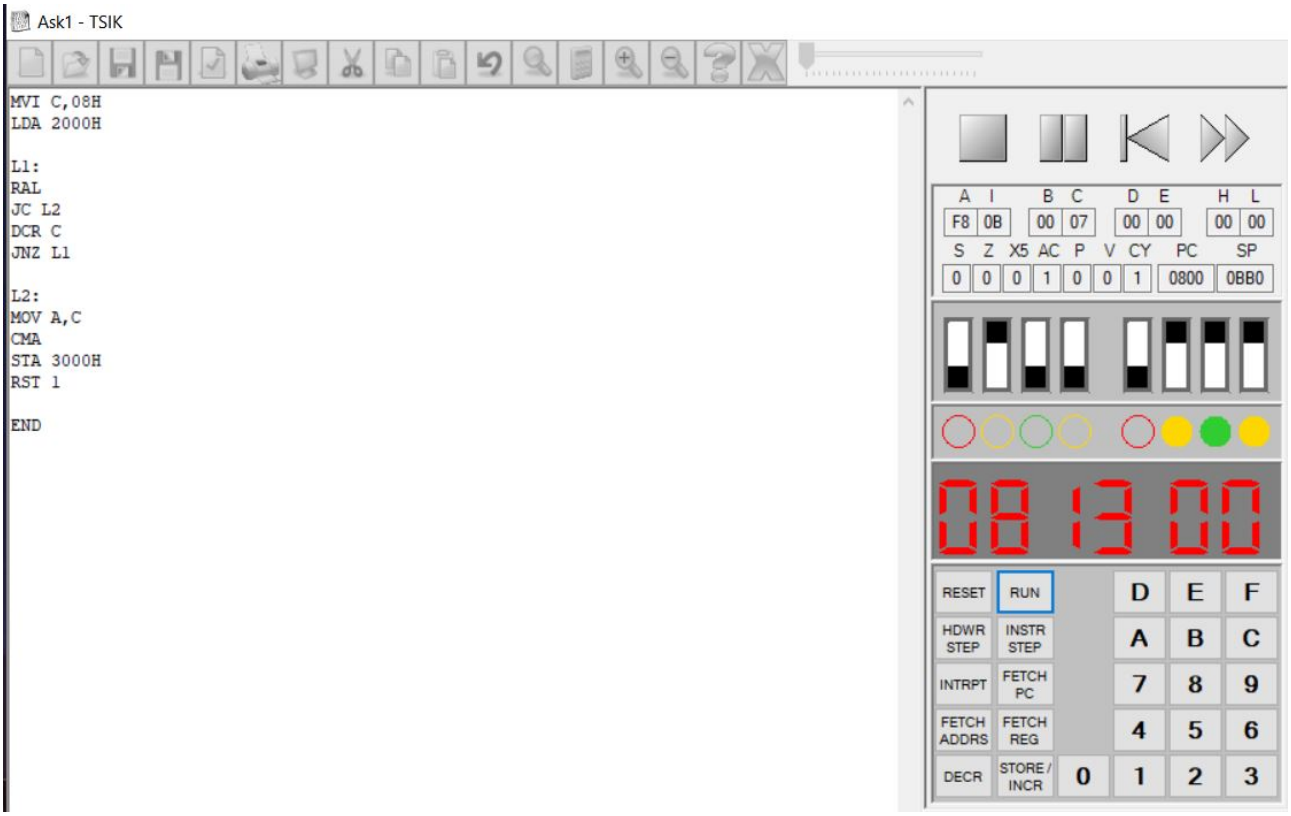
Υπόθεση: Το πρόγραμμα υποθέτουμε ότι είναι φορτωμένο στη μνήμη με αρχή τη διεύθυνση 0800 και οι bold κωδικοί είναι εντολές.

Disassembly

```
MVI    C, 08H      ; (C) = (8)
LDA    2000H       ; (A) = 2000H

L1:                ; FIRST LABEL
RAL                ; ROTATE + : A LEFT
JC L2              ; IF CY=1 THEN JUMP TO L2
DRC C              ; (C) <— (C) — 1
JNZ L1             ; IF Z != 0 JUMP TO L1

L2:                ; SECOND LABEL
MOV A, C           ; (A) <— (C)
CMA                ; (A) <— (A)'
STA 3000H          ; (A) —> 3000H
RST 1

END
```

Τρέχοντας την προσομοίωση, παρατηρώ ότι το πρόγραμμα δέχεται έναν 8-bit αριθμό από τους διακόπτες και εμφανίζει στα LEDs ένα δυαδικό αριθμό. Αυτός ο αριθμός, εκφράζει την θέση του άσσου που βρίσκεται αριστερότερα σε σχέση με όλους τους άλλους.

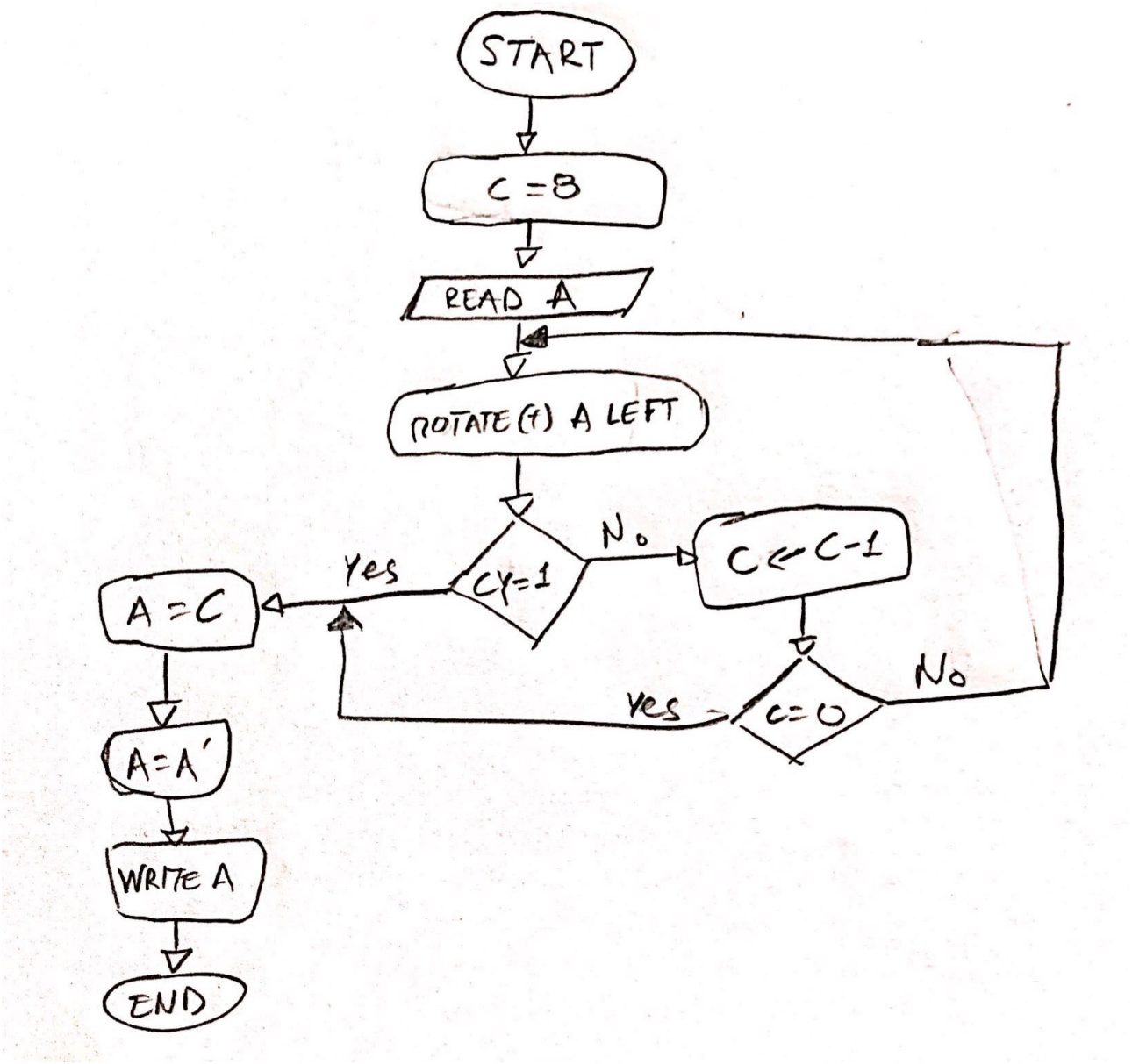
Για παράδειγμα τρέχοντας το παρακάτω παράδειγμα:



Από τους διακόπτες διάλεξα τυχαία τον αριθμό 01000111. Έκανα fetch address 0800 και κατόπιν run. Άναψαν τα 3 πρώτα LEDs, δείχνοντας την θέση του αριστερότερου άσσου: 00000111 ή αλλιώς 7 στο δεκαδικό.

Σημείωση: Αν δεν έχουμε άσσο, τα LEDs παραμένουν σβηστά, υποδεικνύοντας ότι δεν υπάρχει.

Διάγραμμα ροής



Για να έχουμε συνεχή λειτουργία του παραπάνω προγράμματος μπουρούμε να προσθέσουμε μια ετικέτα START στην αρχή του προγράμματος και στο τέλος του την εντολή JMP START. Μέσω της RST 1 γίνεται διακοπή του προγράμματος, μέχρι την εισαγωγή νέας εισόδου και την εκτέλεσή της με την εντολή RUN.

Άσκηση 2η

```
IN 10H                  ;Επιτρέπει πρόσβαση για αποθήκευση μεταβλητών και δεδομένων
                        ;οπουδήποτε στην διαθέσιμη μνήμη RAM.
LXI B,01F4H             ;Ο διπλός καταχωρητής BC παίρνει την τιμή 500, για να χρησιμοποιηθεί
                        ;μετά στο delay
MVI A,01H               ;Αρχική θέση LED
CMA                     ;Συμπλήρωμα ως προς 1 του A (αντίστροφη λογική)
STA 3000H               ;Θύρα εξόδου των LED
CMA                     ;Ξαναπαίρνουμε την αρχική τιμή του A
MOV D,A                 ;Φυλάμε το περιεχόνεο του A στον D

INFINITE_LOOP:          ;Αρχή του ατέρμονου βρόχου
  CALL DELB             ;Εισάγει το ζητούμενο delay των 0.5 sec
  LDA 2000H             ;Διάβασμα εισόδου  (διακόπτες) και αποθήκευση στον A
  ANI 01H               ;Απομόνωση του LSB του A, που είναι ίσο με το LSB της εισόδου (των
                        ;διακοπτών)
  CPI 01H               ;Σύγκριση του LSB με το 1
  JZ NEXT               ;Αν Z=1, πήγαινε στην ετικέτα NEXT (δηλαδή αν ο LSB διακόπτης είναι
                        ;ON)
 JMP INFINITE_LOOP              ;Αν ο LSB διακόπτης είναι OFF,
                                ;μην ανάψεις LED και πήγαινε πάλι πάνω

  NEXT:                 ;Ανάβει νέο LED
       LDA 2000H        ;Βάζω στο A τα δεδομένα της εισόδου
       ANI 80H          ;Απομονώνω το MSB του A που είναι και αυτό των διακοπτών
       CPI 80H          ;Σύγκριση MSB με την μονάδα
       JZ RIGHT         ;Αν Z=1, πήγαινε στην ετικέτα LEFT  (το MSB των διακοπτών είναι ON)
       JMP LEFT         ;Αλλιώς, πήγαινε στην ετικέτα RIGHT (το MSB των διακοπτών είναι OFF)

  LEFT:                 ;Αριστερή ολίσθηση των LEDs
        MOV A,D         ;Το A παίρνει την τιμή που διάβασε αρχικά από την είσοδο
         RLC            ;Ολίσθηση προς τα αριστερά
         MOV D,A        ;Το D ξαναφυλάει την τιμή του A
         CMA            ;Συμπλήρωμα ως προς 1 του A (αντίστροφη λογική)
         STA 3000H      ;Ανάβει το LED που υποδεικύει το A
         CMA            ;Το A ανακτά το αρχικό περιεχόμενό του
         JMP INFINITE_LOOP      ;Νέα επανάληψη

 RIGHT:                 ;Δεξιά ολόσθηση των LEDs
         MOV A,D        ;Το A παίρνει την τιμή που διάβασε αρχικά από την είσοδο
         RRC            ;Ολίσθηση προς τα δεξιά
         MOV D,A        ;Το D ξαναφυλάει την τιμή του A
         CMA            ;Συμπλήρωμα ως προς 1 του A (αντίστροφη λογική)
         STA 3000H      ;Ανάβει το LED που υποδεικύει το A
         CMA            ;Το A ανακτά το αρχικό περιεχόμενό του
         JMP INFINITE_LOOP      ;Νέα επανάληψη
END
```

Αρχικά γίνεται επανάληψη μέχρι να λάβει την τιμή 1 ο διακόπτης LSB (δεξιότερος). Μόλις την λάβει, ελέγχει αν το MSB έχει και αυτό την τιμή 1. Αν ναι, ολισθαίνουν ανάβοντας προς τα δεξιά τα LEDs. Αν όχι, προς τα αριστερά. Εδώ αξίζει να σημειωθεί και ότι ισχύει η αντίστροφη λογική απεικόνισης. Τα αναμμένα LEDs ερμηνεύονται ως OFF, ενώ τα σβησμένα ως ON. Έτσι, πρέπει να αντιστραφεί το περιεχόμενο του A μέσω του συμπληρώματος ως προς 1 (CMA A) πριν την έξοδο (STA 3000H). Αμέσως μετά, χρειάζεται και δεύτερη αντιστροφή για να γίνει η ροή του προγράμματος με την σωστή τιμή της εισόδου.

3η άσκηση

```
IN 10H          ; Επιτρέπει πρόσβαση για αποθήκευση μεταβλητών και δεδομένων
                ; οπουδήποτε στην διαθέσιμη μνήμη RAM.
MVI E,00H       ; Αρχικοποίηση μεταβλητής που φυλάει την είσοδο
LXI B,01F4H     ; Ο διπλός καταχωρητής BC παίρνει την τιμή 500 για να χρησιμοποιηθεί
                ; μετά στο delay

INPUT: LDA 2000H        ; Διαβάζεται η είσοδος
       CMP E            ; Αν είναι ίδια/ δεν έχω…
       JZ INPUT         ; …ξαναδιαβάζω συνεχώς
       MOV E,A          ; Αφού έχει αλλάξει η είσοδος ενημέρωσέ την
       MVI H,FFH        ; Αρχικοποίηση μετρητή δεκάδων με 0
       CPI 64H          ; Σύγκριση με το 100
       JNC CHECK        ; Μεγαλύτερο; Πήγαινε στην ετικέτα CHECK
       JMP SHOW         ; Μικρότερο; Εμφάνισέ το
       JMP INPUT        ; Αλλιώς επανάληψη

CHECK: CPI C8H          ; Σύγκριση με 200
       CALL DELB        ; Καθυστέρηση 0.5 sec
       JNC MSB          ; Μεγαλύτερο του 200 => άναψε τα MSB
       JMP LSB          ; Μικρότερο του 200 => άναψε τα LSB

MSB: MVI L,0FH          ; (L) <— 00001111 (αντίστροφη λογική στην έξοδο)
   JMP FLASH            ; Δίνει τα κατάλληλα SB στην FLASH

LSB: MVI L,F0H          ; (L) <— 11110000 (αντίστροφη λογική στην έξοδο)
   JMP FLASH            ; Δίνει τα κατάλληλα SB στην FLASH

FLASH:
       MOV A,L          ; Δέχεται 00001111 ή 11110000
       STA 3000H        ; Τυπώνει στην έξοδο
       CALL DELB        ; Καθυστερεί
       MVI A,FFH        ; Για να αναβοσβήνουν τα LEDs
       STA 3000H        ; >>      >>      >>      >>
       LDA 2000H        ; Διάβασμα εισόδου
       CMP E            ; Έλεγχος αν άλλαξε η είσοδος, μέσω σύγκρισης
                        ; με το E που φυλάει την προηγούμενη είσοδο
       JZ CHECK         ; CHECK και όχι FLASH για να αναβοσβήνουν τα LEDs
       CALL DELB        ; Καθυστέρηση
       MOV A,E          ; Εφόσον έφτασες εδώ, η είσοδος είναι διαφορετική
                        ; φύλαξέ την
       JMP INPUT        ; Πήγαινε να ελέγξεις την είσοδο
```

```
SHOW:
    INR H               ; Μετράει τις δεκάδες
    SUI 0AH             ; Αφαιρεί από τον A 10 μέχρι…
    JNC SHOW            ; …να γίνει αρνητικός
    ADI 0AH             ; Προσθέτω 10 για να γίνουν θετικές οι μονάδες
    MOV C,A             ; Αποθήκευση μονάδων
    MOV A,H             ; Αποθήκευση δεκάδων στον A
    RRC                 ; Πολ/σμός επί 2, 4 φορές, για να απεικονίζονται
    RRC                 ; οι δεκάδες από το 5ο LED και μετά
    RRC
    RRC
    ADD C               ; Πρόσθεση μονάδων
    CMA                 ; Λόγω αντίστροφης λογικής, συμπλήρωμα ως προς 1 του A
    STA 3000H           ; Άναψε τα κατάλληλα LEDs για αριθμούς <100
    JMP INPUT           ; Ξαναδιάβασε input στην ετικέτα INPUT
END
```

4η Άσκηση

Τεχνικο-οικονομική προσέγγιση κατασκευής μιας φορητής ηλεκτρονικής συσκευής με τη χρήση 3 διαφορετικών τεχνολογιών:

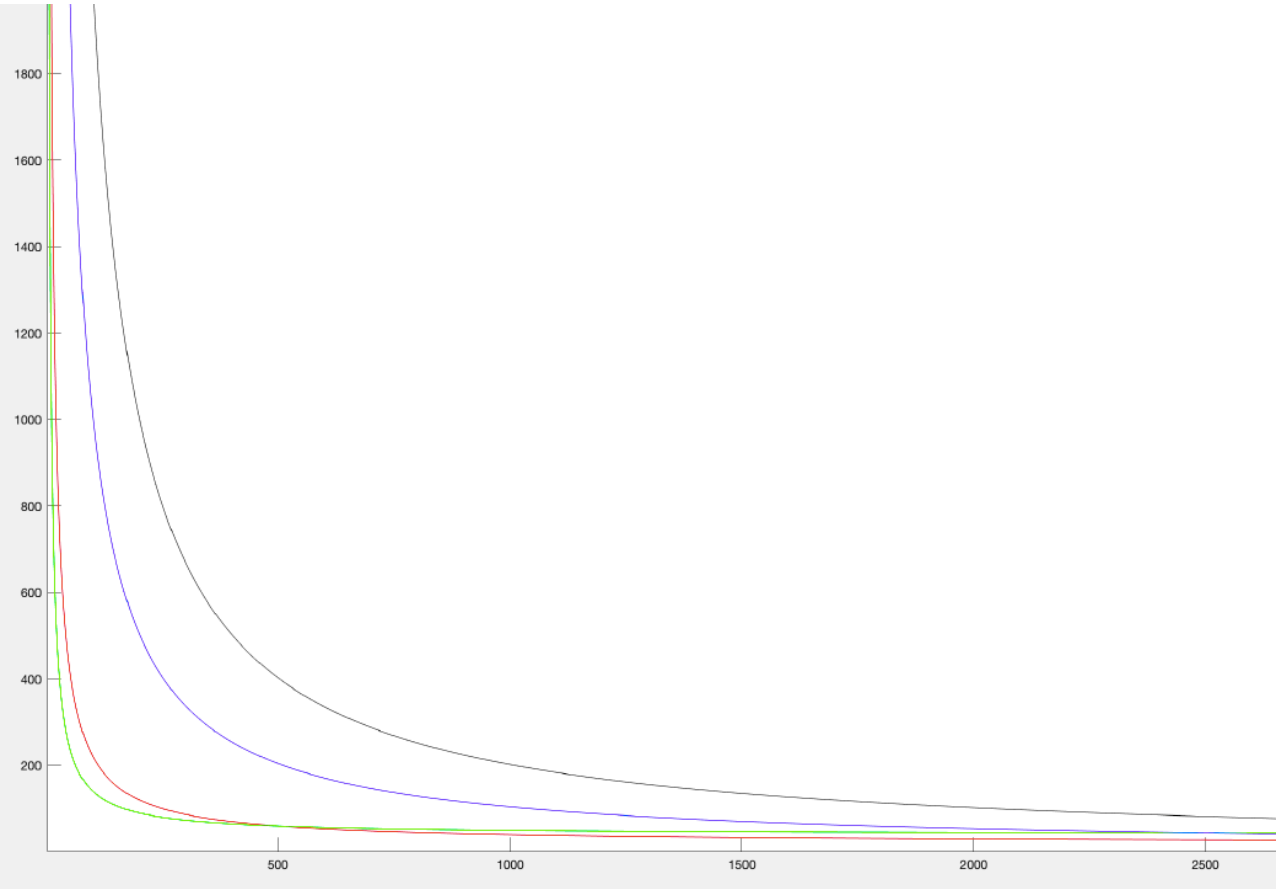
Τεχνολογία	Κόστος	Κόστος ανά τεμάχιο
Χρήση διακριτών στοιχείων και I.C. όπως μικροελεγκτών, περιφερειακών, μνημών κλπ. Τοποθετημένα σε μια σε μια σχετικά μεγάλη πλακέτα	(20000 + 20x) ευρώ	(20000 + 20x)/x ευρώ
Χρήση FPGAs και μικρού αριθμού περιφερειακών τοποθετημένα σε μια σε μια πλακέτα.	(10000 + 40x) ευρώ	(10000 + 40x)/x ευρώ
Σχεδίαση ειδικού SoC-1 με μια μικρή πλακέτα.	(100000 + 4x) ευρώ	(100000 + 4x)/x ευρώ
Σχεδίαση ειδικού SoC-2 με μια πολύ μικρή πλακέτα.	(200000 + 2x) ευρώ	(200000 + 2x)/x ευρώ

Καμπύλες κόστους από MATLAB:

```
axis([1 4000 1 4000])
x=1:1:4000;

t1 = (20000 + 20*x)./x;
plot(x, t1, 'r');
hold on;

t2= (10000 + 40*x)./x;
plot(x, t2, 'g');
hold on;

t3 = (100000 + 4*x)./x;
plot(x, t3, ’b');
hold on;

t4 = (200000 + 2*x)./x;
plot(x, t4, ’k');
```

Περικομμένες γραφικές καμπύλων κόστους για καλύτερη ανάγνωση:

κόστος
ανά τεμάχιο



αριθμός τεμαχίων

Τεχνολογία 1 == Τ1 == κόκκινο
Τεχνολογία 2 == Τ2 == Πράσινο
Τεχνολογία 3 == Τ3 == μπλε
Τεχνολογία 4 == Τ4 == μαύρο

Μετά από υπολογισμούς προκύπτει ότι οι 4 περιοχές που είναι συμφερότερες (χαμηλότερου κόστους) για την κάθε μία τεχνολογία είναι:

$$T2: \quad 1 < x < 500$$
$$T1: \quad 500 < x < 5000$$
$$T3: \quad 5000 < x < 50000$$
$$T4: \quad x > 50000$$

Παρατηρώ ότι με την αύξηση τεμαχίων που παράγονται, γίνεται πιο συμφέρουσα η τεχνολογία με ακριβότερο κόστος σχεδίασης.

Νέα ανά μονάδα τιμή y των I.C. για τα οποία η Τ2 είναι πιο συμφέρουσα από την Τ1:

$$10000 + (y+10)x < 20000 + 20x \Rightarrow y < 10 + 10000/x$$

Επιλογικά, για $y \leq 10€$ εξαφανίζεται η επιλογή της πρώτης τεχνολογίας.

Άσκηση 5η

(I) Επίπεδο πυλών//Δομική περιγραφή των δοσμένων λογικών συναρτήσεων

```
module Circuit (input A, B, C, D, E,  output F1, F2, F3, F4);
wire A', B', C', D';
not (A', A);
not (B', B);
not (C', C);
not (D', D);

// F1 = A(BC +D) + B'C'D
wire w1, w2, w3, w4;
and (w1, B, C);
or   (w2, w1, D);
and (w3, A, w2);
and (w4, B', C', D);
or   (F1, w3, w4);
```

```
// F2 (A, B, C, D) = Σ (0, 2, 3, 5, 7, 9, 10, 11, 13, 14)
wire r0, r2, r3, r5, r7, r9, r10, r11, r13, r14;
and (r0, A', B', C', D');
and (r2, A', B', C, D');
and (r3, A', B', C, D);
and (r5, A', B, C', D);
and (r7, A', B, C, D);
and (r9, A, B', C', D);
and (r10, A, B', C, D');
and (r11, A, B', C, D);
and (r13, A, B, C', D);
and (r14, A, B, C, D');
or   (F2, r0, r2, r3, r5, r7, r9, r10, r11, r13, r14);

// F3 = ABC + (A + BC)D + (B + C)DE
wire w5, w6, w7, w8, w9, w10;
and (w5, A, B, C);
or   (w6, A, w1);
and (w7, w6, D);
and (w8, D, E);
or   (w9, B, C);
and (w10, w9, w8);
or   (F3, w5, w7, w10);

// F4 = A(B + CD + E) + BCDE
wire w11, w12, w13, w14;
and (w11, C, D);
or   (w12, B, w11, E);
and (w13, A, w12);
and (w14, B, C, D, E);
or   (F4, w13, w14);
endmodule
```

(II) Μοντελοποίηση ροής δεδομένων (Verilog)

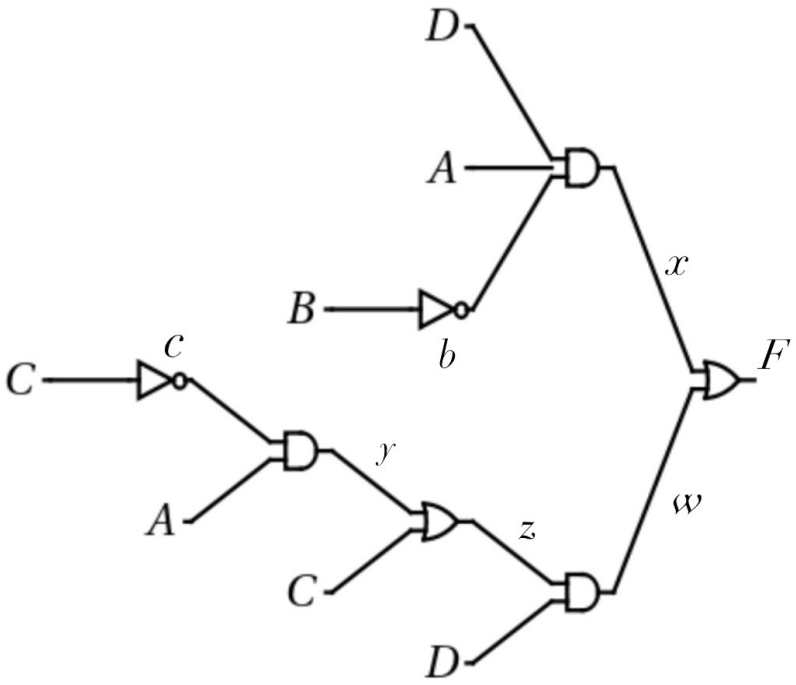
```
module Circuit (input A, B, C, D, E,  output F1, F2, F3, F4);
assign
// F1 = A(BC + D) + B'C'D
F1 = (A & ((B & C) | D)) | (~B, ~C, D),

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

// F3 = ABC + (A + BC)D + (B+ C)DE
F3 = (A & B & C) | ((A | (B & C)) & D) | ((B | C) & D & E),

// F4 = A(B + CD + E) + BCDE
F4 = (A & (B | (C & D) | E)) | (B & C & D & E);
endmodule
```

6η άσκηση

(i)

Παρακάτω παρατίθενται τα λογικά διαγράμματα των ψηφιακών κυκλωμάτων που ορίζονται στις δοσμένες περιγραφές Verilog:

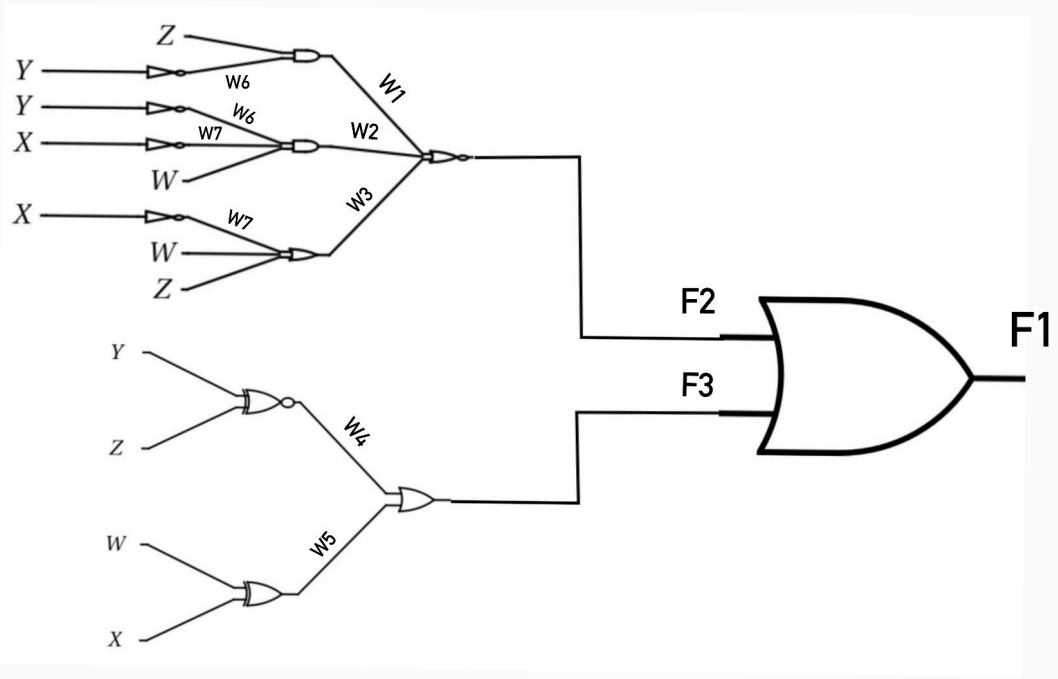
(a)



$$(D \wedge A \wedge \neg B) \vee (((\neg C \wedge A) \vee C) \wedge D)$$

(*D* AND *A* AND (NOT *B*)) OR ((((NOT *C*) AND *A*) OR *C*) AND *D*)

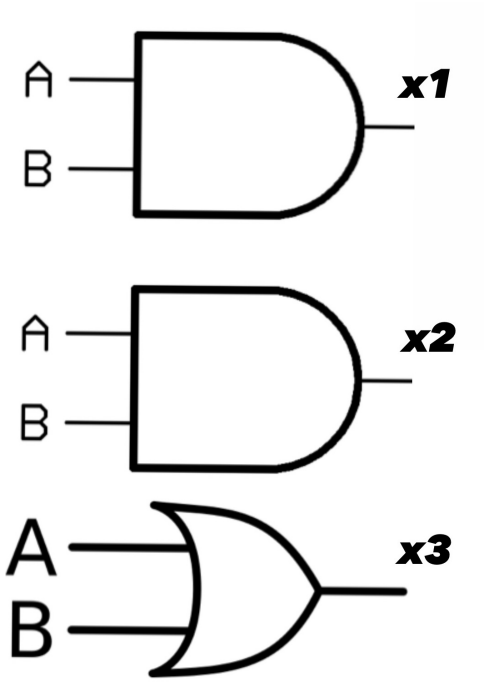
(b)



$$\neg\,((Z \wedge \neg\, Y) \vee (\neg\, Y \wedge \neg\, X \wedge W) \vee (\neg\, X \vee W \vee Z)) \vee ((Y \,\overline{\vee}\, Z) \vee (X \veebar W))$$

(NOT ((*Z* AND (NOT *Y*)) OR ((NOT *Y*) AND (NOT *X*) AND *W*) OR (NOT *X*) OR *W* OR *Z*)) OR (*Y* XNOR *Z*) OR (*W* XOR *X*)

(c)



(ii)

Ιεραρχική περιγραφή HDL σε επίπεδο πύλης για έναν αθροιστή-αφαιρέτη 4 bit για μη προσημασμένους δυαδικούς αριθμούς:

```
//Βοηθητικά modules
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); //Instantiate HAs
        half_adder HA2 (S, C2, S1, z);
        or G1 (C, C2, C1);
endmodule
```

```
//Module αθροιστή-αφαιρέτη 4bit
module adder_subtractor(output [3:0] S, output C4, V, input M, input [3:0] A, B);

        wire C1, C2, C3;
        wire b0, b1, b2, b3;

        xor G1 (b0, B[0], M);
        xor G2 (b1, B[1], M);
        xor G3 (b2, B[2], M);
        xor G4 (b3, B[3], M);

        full_adder FA_0 (S[0], C1, A[0], b0, M);
        full_adder FA_1 (S[1], C2, A[1], b1, C1);
        full_adder FA_2 (S[2], C3, A[2], b2, C2);
        full_adder FA_3 (S[3], C4, A[3], b3, C3);

        xor G5 (V, C4, C3);

end module
```

(iii)

```
module adder_subtractor(output [3:0] S, output C4, V, input M, input [3:0] A, B);
        assign {S, C4} = M ? A - B : A + B;
endmodule
```

Άσκηση 7η

(i)

```
module edge_dependant ( clk, rst, eisodos, eksodos );
                                                    //dhlwseis I/O
  input clk, rst, eisodos;
  output eksodos;
                                                    //dhlwseis kataxwrhtwn
  reg [1:0] state;
  reg eksodos;
                                                    //dhlwseis parametrwn
  parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2’b11;
                                                    //always block
  always @( posedge clk, posedge rst ) begin
                                    // Se periptwsh akmhs reset pame sto state a
```

```
if( rst ) begin
   state <= 2'b00;
   eksodos <= 0;
end
                              // Alliws ginetai elegxos tou FSM me vash tis eisodous
else  begin
   case( state )                          //elegxos twn pithanwn cases
   a: begin                                                  // State a
       if( eisodos ) begin                // eisodos = 1 => paramenei sto a
         state <= a;
         eksodos  <= 0;                   // output 0 (fainetai sthn akmh)
       end
       else begin                         // alliws, eisodos = 0
          state <= d;                     // pame sthn katastash d tou FSM
          eksodos <= 1;                   // output 1
       end                                // omoiws gia ta ypoloipa cases
   end

   b: begin
       if( eisodos ) begin
          state <= a;
          eksodos  <= 0;
       end
       else  begin
         state <= c;
         eksodos <= 1;
       end

   end

   c: begin
       if( eisodos ) begin
          state <= b;
          eksodos  <= 0;
       end
       else begin
         state <= d;
         eksodos <= 1;
       end

   end
  d: begin
       if( eisodos ) begin
          state <=d;
          eksodos  <= 1;
```

```
          end
          else begin
            state <= c;
            eksodos <= 0;
          end

      end
      default: begin                    // an den isxuei kati apo ta parapanw
          state <= 2'b00;               // h genikh/default periptwsh einai na
          eksodos <= 0;                 // ksekina me to a me output 0
      end
    endcase
  end
end

endmodule
```

(ii) // Moore FSM: output only depends on current state

```
module state_dependant (s_in, clk, rst, d_out);
                                                // dhlwseis I/O
input clk;
input rst;
input s_in;                             // input sequence, (binary)
output reg d_out;                       // detects output of sequence
                                        // dhlwsh parametrwn
parameter
 parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;

reg [2:0] current_state, next_state;

                                                //always block
always @ ( posedge clk, posedge rst )
begin
 if ( rst==1 )
 current_state <= a;                    // Se periptwsh akmhs reset pame sto state a
 else
 current_state <= next_state;                   // alliws, next state
end
                                                //always block
always @ ( current_state, s_in )
begin
 case ( current_state )                         //elegxos twn pithanwn cases
 a: begin                                       // opws kai prin, me vash to FSM
  if ( s_in==1 )
   next_state = a;
```

```
  else
   next_state = d;
 end
 b: begin
  if ( s_in==0 )
   next_state = c;
  else
   next_state = a;
 end
 c: begin
  if( s_in==0 )
   next_state = b;
  else
   next_state = d;
 end
 d: begin
  if ( s_in==0 )
   next_state = c;
  else
   next_state = d;
 end
 default: next_state = a;                    //omoiws kai edw, default state to a
 endcase
end

                                                    //always block
always @ ( current_state )
begin
 case ( current_state )                             //cases gia to twrino state
 a:   d_out = 0;                                    //ws state dependant kai oxi edge
 b:   d_out = 1;                                    //analoga me to state exoume kai
 c:   d_out = 1;                                    //to antistoixo output
 d:   d_out = 0;
 default:  d_out = 0;
 endcase
end
endmodule
```

Σημείωση πρώτη: οι κώδικες βρίσκονται στον αντίστοιχο φάκελο του zip.

Σημείωση δεύτερη: για την verilog χρησιμοποιήθηκε το site:
www.edaplayground.com

————————————————ΤΕΛΟΣ ΕΡΓΑΣΙΑΣ————————————————