ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ

1Η ΟΜΑΔΑ ΑΣΚΗΣΕΩΝ

ΑΡΓΥΡΟΠΟΥΛΟΥ ΖΩΗ 03118904

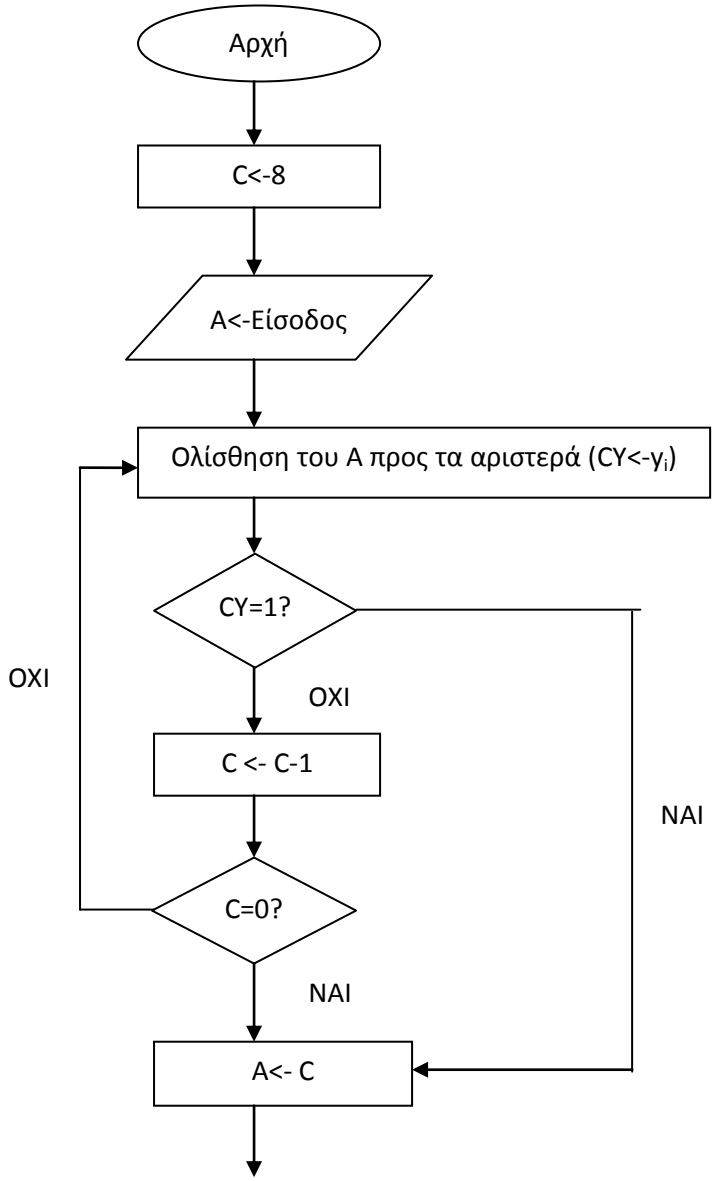
ΑΝΘΟΠΟΥΛΟΥ ΦΑΙΔΡΑ ΑΝΑΣΤΑΣΙΑ 0318818

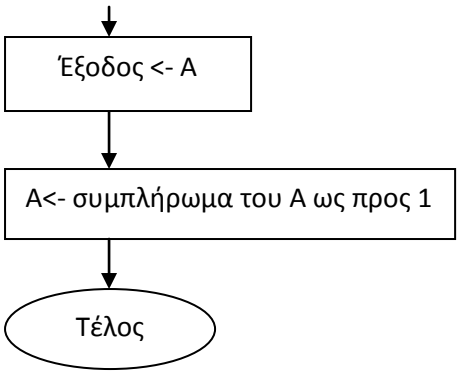
1η ΑΣΚΗΣΗ:

Ο κώδικας για αυτήν την άσκηση βρίσκεται στο αρχείο zip που παραδίδουμε, με το όνομα ask1.

Το πρόγραμμα αυτό παίρνει ως είσοδο έναν αριθμό και αναζητά την θέση (μετρώντας από τα δεξιά προς τα αριστερά) που βρίσκεται το most significant bit του. Επιστρέφει ως έξοδο το συμπλήρωμα ως προς το 1 του αριθμού που αντιπροσωπεύει την θέση αυτή.

Ακολουθεί το διάγραμμα ροής του:





Για να επαναλαμβάνεται για πάντα το πρόγραμμα ένας τρόπος είναι το να αφαιρέσουμε την εντολή DCR C και να αλλάξουμε την JC adr 2 σεJC adr 1.

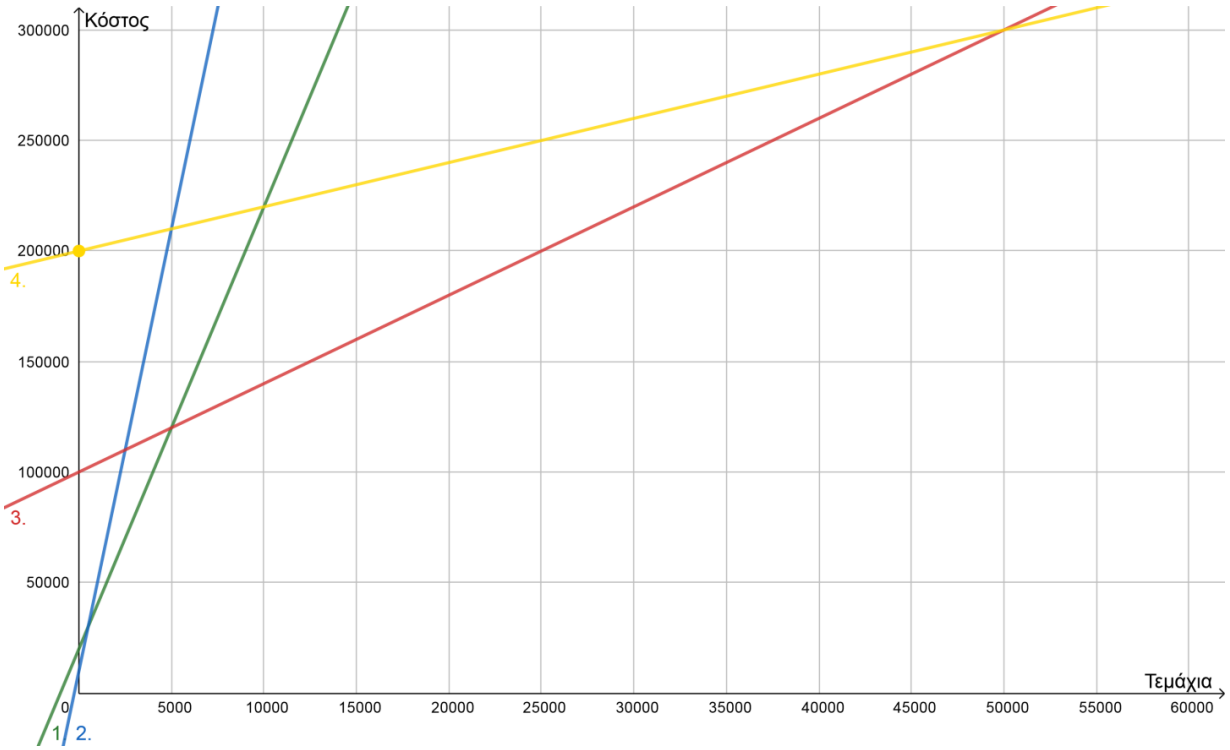
2η 3η ΑΣΚΗΣΗ

Στον προσομοιωτή του εκπαιδευτικού προγράμματος μLab αναπτύχθηκαν τα προγράμματα ask2.8085 και ask3.8085 σε assembly, τέτοια ώστε να πληρούν τα ζητούμενα των αντίστοιχων εκφωνήσεων. Οι κώδικες μαζί με διευκρινιστικά σχόλια βρίσκονται στο αρχείο που παραδώθηκε.

4η ΑΣΚΗΣΗ:

1. Η καμπύλη κόστους ανά τεμάχιο δίνεται από την εξίσωση y=20.000 + 20x€.
2. Η καμπύλη κόστους ανά τεμάχιο δίνεται από την εξίσωση y=10.000 + 40x€.
3. Η καμπύλη κόστους ανά τεμάχιο δίνεται από την εξίσωση y=100.000 + 4x €.
4. Η καμπύλη κόστους ανά τεμάχιο δίνεται από την εξίσωση y=200.000 + 2x€.

Τις παρουσιάζουμε όλες σε ένα κοινό διάγραμμα (το x θα έπρεπε να είναι αυστηρά μεγαλύτερο ή ίσο του 0, καθώς αντιπροσωπεύει τα τεμάχια) αλλά δεν μπορούσαμε να το ορίσουμε στο πρόγραμμα):



Τα σημεία τομής είναι:**1-2**: (500, 30.000), **1-3**: (5.000, 120.000), **1-4**: (10.000, 220.000), **2-3**: (2.500, 110.000), **2-4**: (5.000, 210.000), **3-4**: (50.000, 300.000)

Επομένως η περίπτωση **1.** είναι η πιο συμφέρουσα για 500-5.000 τεμάχια, η **2.**για 0-500 τεμάχια, η **3.** για 5.000-50.000 τεμάχια και η **4.** για περισσότερα από 50.000 τεμάχια.

Για να ερευνήσουμε για ποια τιμή κόστους ανά τεμάχιο (έστω α) της περίπτωσης **2.** θα μπορεί να εξαφανιστεί η **1.** αρκεί να βρούμε το μέγιστο α για το οποίο ισχύει η ανισότητα $10.000 + (10+\alpha)x \leq 20.000 + 20x$ για κάθε x. Λύνοντας βρίσκουμε ότι θα πρέπει $(\alpha-10)x \leq 10.000$ για κάθε x. Αυτό ισχύει για $\alpha \leq 10$. Άρα για τιμή 10€ ανά τεμάχιο ή λιγότερο στην περίπτωση **2**, η **1** μπορεί να εξαφανιστεί.

Συμπεραίνουμε πως για μεγαλύτερο αριθμό στοιχείων σε μια πλακέτα συμφέρει να χρησιμοποιήσουμε κάποιο SoC, το οποίο το περιμέναμε. Σε αυτήν την περίπτωση συμφέρει γενικά η χρήση SoC, αφού έχουν μικρότερο μέγεθος, που είναι επιθυμητό και ίσως αναγκαίο από την στιγμή που κατασκευάζουμε φορητή συσκευή (συνήθως αρκετά μικρές σε μέγεθος).

5η ΑΣΚΗΣΗ

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

Περιγραφή σε επίπεδο πυλών

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

not G1(Bnot,B),
G3(Cnot,C);

and G2(w1,B,C),
G5(w3,Bnot,Cnot,D),
G6(w4,A,w2);

or G4(w2,D,w1),
G7(F1,w4,w3);

endmodule
```

Μοντελοποίηση ροής δεδομένων

```
modulecircuit_F1 (F1,A,B,C,D);
outputF1;
inputA,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)**

Περιγραφή σε επίπεδο πυλών

```
primitive UDP_F2 (F2, A, B, C, D);

    output F2;
    input A, B, C, D;
    table // πίνακας αληθείας για τηνF2

// A  B C  D:    F2 // column header

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
```

Μοντελοποίηση ροής δεδομένων

```
primitiveUDP_F2 (F2, A, B, C, D);

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**

Περιγραφή σε επίπεδο πυλών

```
module circuit_F3(F3,A,B,C,D)
output F3;
input  A,B,C,D,E;
wire w1,w2,w3,w4,w5,w6;

and G1(w1,w6,D),
G4(w4,A,B,C),
G5(w5,w2,D,E)
G3(w3,B,C);

or G2(w2,B,C),
G6(w6,w3,A),
G7(F3,w4,w1,w5);
endmodule
```

Μοντελοποίηση Ροής Δεδομένων

```
module circuit_F3 (F3, A, B , C, D, E);
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+(C D) + E) +B CDE**

Περιγραφή σε επίπεδο πυλών

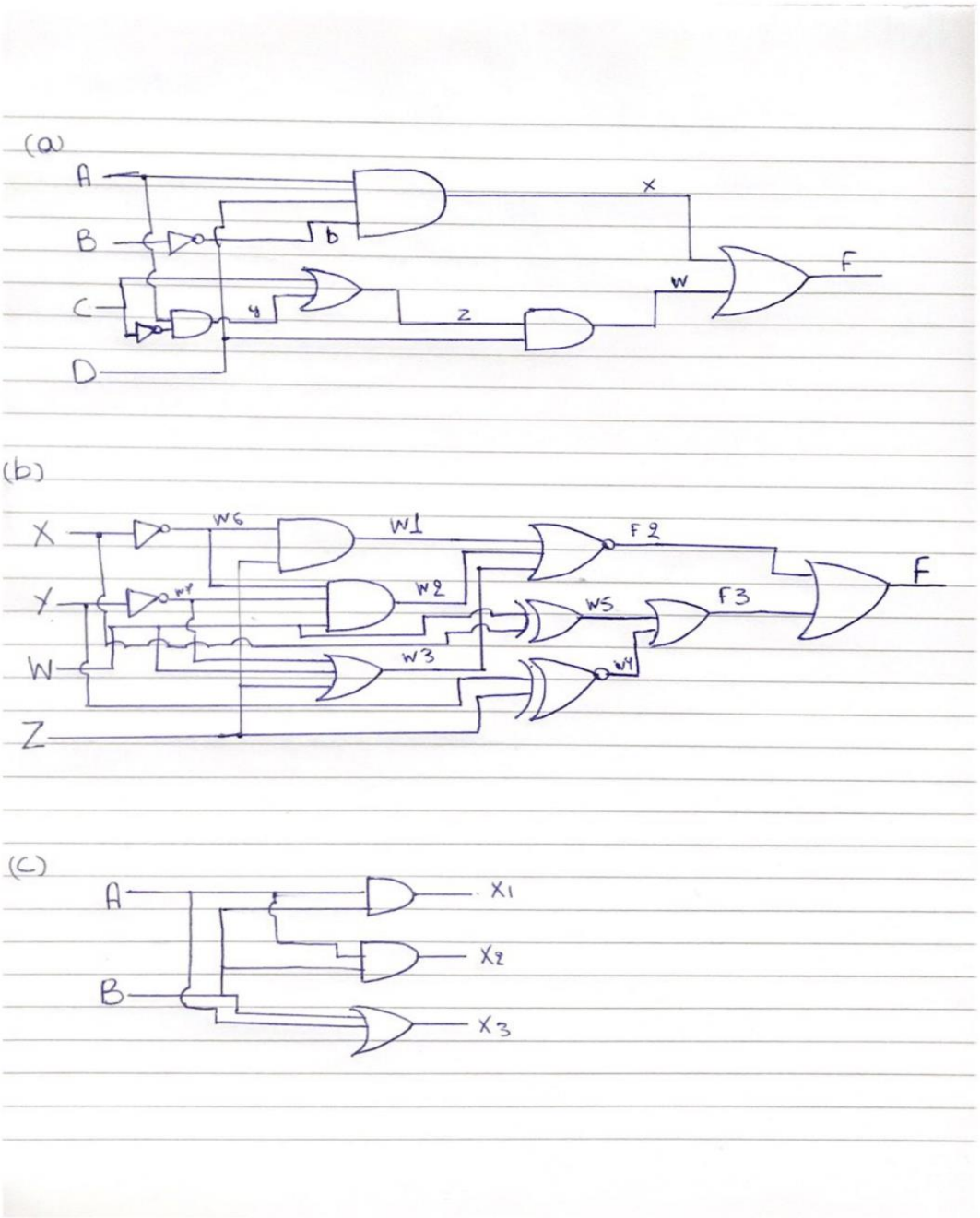
```
modulecircuit_F4 (F4, A, B, C, D, E);
output F4;
input A, B, C, D, E;
wire w1, w2, w3, w4;

andG1(w1,C,D),
G2(w2, B,C, D, E),
G4(w4, A, w3);

or G3(w3, w1, B, E),
G5(F4, w4, w2);
endmodule
```

Μοντελοποίηση Ροής Δεδομένων

```
module circuit_F4 (F4, A, B, C, D, E);
outputF4;
input A, B, C, D, E;
assign F4 = (A & (B |(C& D) | E)) | (B &C & D & E);
endmodule
```

6η ΑΣΚΗΣΗ

(i)



(i)**module**half_adder (output S, C, input x, y);

xor (S, x, y);

and(C, x, y);

endmodule

```
    modulefull_adder(output S, C, input x, y, z);
    wire S1, C1, C2;
    half_adderHA1(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
```

7η ΑΣΚΗΣΗ

(i)

```
module States(x,y);
output [0:1]y;
input x, change, reset;
reg [1:0]state;
parameter a, b, c, d;
always@(posedge change, negedge reset)
if (reset==0) state <= a;
else case(state)
a: if (~x)
begin
state<=d;
      y<=1;
end
else
begin
state<=a;
       y<=0;
end
b: if (x)
begin
state<=a;
y<=0;
end
else
begin
state<=c;
        y<=1;
end
c: if (x)
begin
state<=b;
      y<=0;
end
else
begin
state<=d;
       y<=1;
end
d: if (~x)
begin
```

```
state<=c;
      y<=0;
end
else
begin
state<=d;
       y<=1;
end
endcase
endmodule
```

(ii)

```
module Moore_Machine (
outputy_out,
inputx_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;

endcaseendmodule