Συστήματα Μικροϋπολογιστών

2η Σειρά Ασκήσεων

Αλέξανδρος Λιαροκάπης (03114860)
Απόστολος Μάνος (03114855)

Άσκηση 1

Σε ένα μ-Σ 8085 να γραφεί σε assembly ένα πρόγραμμα που να επιτελεί τις παρακάτω λειτουργίες:

(α) Να αποθηκευθούν οι αριθμοί 0-255 με αύξουσα σειρά στις διαδοχικές θέσεις της μνήμης με αρχή τη διεύθυνση 0900H. Οι αριθμοί να αποθηκευτούν όχι χειροκίνητα αλλα μέσω προγράμματος και να ελέγξετε αν έγινε η ζητούμενη λειτουργία σωστά.

(β) Υπολογίστε τον αριθμό των μονάδων των παραπάνω δεδομένων. Το αποτέλεσμα να αποθηκευτεί στον διπλό καταχωρητή BC.

(γ) Υπολογίστε το πλήθος από τους παραπάνω αριθμούς (0-255) που είναι μεταξύ των αριθμών 10H και 60H περιλαμβανομένων ($10H \leq x_n \leq 60H$) και φυλάξτε το αποτέλεσμα στον καταχωρητή D.

(δ) Όταν γίνεται ON το LSB της θύρας εισόδου dip switch να εμφανίζεται στη θύρα εξόδου των LED η τιμή του καταχωρητή BC, αν γίνει ON το επόμενο από το LSB των dip switches ο καταχωρητής C και με τον αμέσως επόμενο διακόπτη, ο καταχωρητής D. Στον έλεγχο των διακοπτών, προτεραιότητα να έχει κάθε φορά το υψηλότερης αξίας bit.

Λύση

(α)
```
;~~~~~~~~~~~~~~~~~~~~~~~~~~

EXA:      LXI B,0900H
          MVI A,00H

EXA_LOOP: STAX B
          INR A
          INR C
          JNZ EXA_LOOP
          RET
```

(β)
```
;~~~~~~~~~~~~~~~~~~~~~~~~~~

EXB:        PUSH D
            LXI B,0000H
            LXI H,0000H
            LXI D,0900H
            MVI A,00H

EXB_LOOP:   LDAX D
            CALL COUNT_BITS
            DAD B
            INR E
            MOV A,E
            CPI 00H
            JNZ EXB_LOOP
            MOV C,L
            MOV B,H
            POP D
            RET

;~~~~~~~~~~~~~~~~~~~~~~~~~~

COUNT_BITS: PUSH D
            MVI C,00H
```

```
CB_LOOP:    CPI 00H
            JZ CB_END
            MOV D,A
            ANI 01H
            JZ CB_IF_END
            INR C

CB_IF_END:  MOV A,D
            STC
            CMC
            RAR
            JMP CB_LOOP

CB_END:     POP D
            RET
```

(γ)

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
EXC:      MVI D,00H
          LXI B,0900H

EXC_LOOP: LDAX B
          CPI 10H
          JM EXC_SKIP
          CPI 61H
          JP EXC_SKIP
          INR D

EXC_SKIP: INR C
          MOV A,C
          CPI 255
          JNZ EXC_LOOP
          RET
```

(δ)

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
EXD:    LDA 2000H
        MOV L,A
        ANI 04H
        JZ EXD_ND
        MOV A,D
        CMA
        STA 3000H
        JMP EXD

EXD_ND: MOV A,L
        ANI 02H
        JZ EXD_NC
        MOV A,C
        CMA
        STA 3000H
        JMP EXD

EXD_NC: MOV A,L
```

```
        ANI 01H
        JZ EXD
        MOV A,B
        CMA
        STA 3000H
        JMP EXD

;~~~~~~~~~~~~~~~~~~~~~~~~~
```

Άσκηση 2

Δίνεται ένα μΥ-Σ 8085 που ελέγχει τα LED της πόρτας εξόδου (3000H) εξομοιώνοντας με αυτά τα φώτα ενός χώρου. Να γραφτεί πρόγραμμα Assembly, που όταν το MSB της θύρας εισόδου dip switch (θέση μνήμης 2000H) από OFF γίνει ON και ξανά OFF τότε να ανάβει όλα τα LED της πόρτας εξόδου. Αυτό να παραμένει ανοιχτό για περίπου 30 sec και μετά να σβήνει. Αν όμως ενδιάμεσα ξαναενεργοποιηθεί το push-button (OFF-ON-OFF το MSB των dip switch) να ανανεώνεται ο χρόνος των 30 sec. Να γίνει χρήση των ρουτινών χρονοκαθυστέρησης του εκπαιδευτικού συστήματος μLAB. Θεωρήστε ότι το σύστημα παρακολουθεί με διακριτική ικανότητα όχι μικρότερη του 1/2 sec.

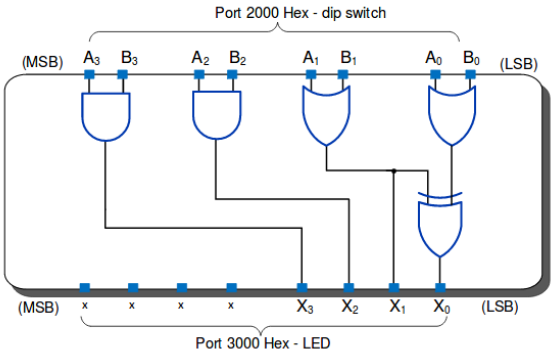
Λύση

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

CHECK_INPUT:      MOV A,L
                  CPI 02H
                  JZ CHECK_INPUT_S2

CHECK_INPUT_S1:   LXI D,0102H
                  CALL CHANGE_STATE
                  RET

CHECK_INPUT_S2:   LXI D,0302H
                  CALL CHANGE_STATE
                  RET

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

CHANGE_STATE:     LDA 2000H
                  ANI 01H
                  JNZ CHANGE_STATE_SET
                  MOV L,D
                  RET

CHANGE_STATE_SET: MOV L,E
                  RET

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

EX2:              LXI H,0001H
                  MVI A,0FFH
                  STA 3000H
                  LXI B,0C8H

EX2_LOOP:         CALL DELB
                  CALL CHECK_INPUT
                  MOV A,L
                  INR H
                  CPI 03H
                  JNZ EX2_NOT_DETECTED
                  MVI H,00H
                  MVI A,00H
                  STA 3000H
```

```
EX2_NOT_DETECTED: MOV A,H
                  CPI 0C8H
                  JNZ EX2_LOOP
                  MVI H,00HH
                  MVI A,0FFH
                  STA 3000H
                  JMP EX2_LOOP

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Ασκηση 3

Να εξομοιωθεί η λειτουργία ενός υποθετικού I.C. που περιλαμβανει 5 πύλες όπως φαίνεται στο σχήμα. Τα bits εισόδου πρέπει να αντιστοιχούν ακριβώς όπως φαίνονται στο σχήμα με τα dip switches της πόρτας εισόδου 2000 Hex, και οι έξοδοι με τα LEDs που πρέπει να είναι τα τέσσερα LSB της πόρτας εξόδου 3000 Hex. Οι πύλες, όπως φαίνονται στο σχήμα, είναι 2 AND, 2 OR και 1 XOR. Η αντιστοιχία των led με τις λογικές στάθμες έχει ως εξής: αναμμένο led ⇒ '1' και σβηστό led ⇒ '0'. Οι αδιάφορες θέσεις της εξόδου να έχουν μόνιμα σβηστά led.



Λύση

```
;~~~~~~~~~~~~~~~~~~

SET_X3: MOV A,B
        AND C
        RRC
        RRC
        RRC
        ANI 8
        OR D
        MOV D,A
        RET

;~~~~~~~~~~~~~~~~~~

SET_X2: MOV A,B
        AND C
        RRC
        RRC
        ANI 4
        OR D
        MOV D,A
        RET

;~~~~~~~~~~~~~~~~~~

SET_X1: MOV A,B
        OR C
        RRC
        ANI 2
        OR D
```

```
        MOV D,A
        RET

;~~~~~~~~~~~~~~~~~~

SET_X0: MOV A,B
        OR C
        MOV E,A
        RRC
        RRC
        XRA E
        ANI 1
        OR D
        MOV D,A
        RET

;~~~~~~~~~~~~~~~~~~

EX3:    MVI D,0
        LDA 2000H
        MOV B,A
        RRC
        MOV C,A
        CALL SET_X3
        CALL SET_X2
        CALL SET_X1
        CALL SET_X0
        MOV A,D
        STA 2000H
        END

;~~~~~~~~~~~~~~~~~~
```

Άσκηση 6

Να λυθούν τα προβλήματα 4.37, 4.40, 4.47

Λύση

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

    assign S = x ^ y;
    assign C = x & y;
endmodule

module full_adder(x,y,z,S,C);
    input x, y, z;
    output S, C;
    wire w1, w2, w3;

    half_adder h1(.x(x), .y(y), .S(w1), .C(w2));
    half_adder h2(.x(w1), .y(z), .S(S), .C(w3));
    assign C = w2 | w3;
endmodule

module adder_4bit(c, A, B, S, C);
    input wire c;
    input wire [3:0] A, B;
    output wire [3:0] S;
    output wire C;
    wire [3:0] Cs;

    full_adder fa(A[0], B[0], c, S[0], Cs[0]);

    genvar i;
    generate
    for (i = 1; i < 4; i=i+1) begin
        full_adder fa(A[i], B[i], Cs[i-1], S[i], Cs[i]);
    end
    endgenerate

    assign C = Cs[3];
endmodule

module addsuber_4bit(M, A, B, S, C);
    input wire M;
    input wire [3:0] A, B;
    output wire [3:0] S;
    output wire C;
    wire [3:0] Ws;

    genvar i;
    generate
    for (i = 0; i < 4; i = i + 1) begin
```

```
        xor(Ws[i], B[i], M);
    end
    endgenerate

    adder_4bit adder(M, A, Ws, S, C);
endmodule

module ex447 (A,B,C,D, F1, F2);
    input A, B, C, D;
    output F1, F2;

    assign F1 = (~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)

    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)
endmodule
```

Άσκηση 7

Να λυθούν τα προβλήματα 6.38, 6.39 και 8.10.

Λύση

```
module counter (out, load, up, down, in, reset, clock);
    output reg [3:0] out;
    input load, up, down, carry_in, reset, clock;
    input [3:0] in;

    always @ (posedge clock, negedge reset) begin
        case (1'b1)
            ~reset: out <= 4'b0000;
            load: out <= in;
            up: out <= out + 1'b1;
            down: out <= out - 1'b1;
        endcase
    end

endmodule

module ex638 (out, in, select, reset, clock);
    output wire [3:0] out;
    input wire [3:0] in;
    input wire [1:0] select;
    input wire reset, clock;

    reg [2:0] counter_inputs;

    counter ctr(out, counter_inputs[2], counter_inputs[1], counter_inputs[0], in, reset, clock);

    always @ (select)
        case (select)
            2'b00: counter_inputs <= 3'b000;
            2'b01: counter_inputs <= 3'b001;
            2'b10: counter_inputs <= 3'b010;
            2'b11: counter_inputs <= 3'b100;
        endcase

endmodule
```