Μικροϋπολογιστές

1η Ομάδα Ασκήσεων

Φιλιππόπουλος Ορφέας el18082 orfeasfil2000@gmail.com
Παπαρρηγόπουλος Θοδωρής el18040 paparrigopoulosthodoris@gmail.com

1η Άσκηση

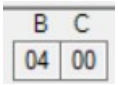
α)

Χρησιμοποιούμε έναν counter (καταχωρητής Α) που ξεκινάει από το 0 και φτάνει έως το 255 και τον αποθηκεύουμε σε διαδοχικές θέσεις μνήμης (ξεκινώντας από τη θέση μνήμης 0900H). Τα αποτελέσματα φαίνονται παρακάτω:

08FA	00	08FB	00	08FC	00	08FD	00	08FE	00	08FF	00	0900	00	0901	01	0902	02	0903	03
0904	04	0905	05	0906	06	0907	07	0908	08	0909	09	090A	0A	090B	0B	090C	0C	090D	0D
090E	0E	090F	0F	0910	10	0911	11	0912	12	0913	13	0914	14	0915	15	0916	16	0917	17
0918	18	0919	19	091A	1A	091B	1B	091C	1C	091D	1D	091E	1E	091F	1F	0920	20	0921	21
0922	22	0923	23	0924	24	0925	25	0926	26	0927	27	0928	28	0929	29	092A	2A	092B	2B
092C	2C	092D	2D	092E	2E	092F	2F	0930	30	0931	31	0932	32	0933	33	0934	34	0935	35
0936	36	0937	37	0938	38	0939	39	093A	3A	093B	3B	093C	3C	093D	3D	093E	3E	093F	3F
0940	40	0941	41	0942	42	0943	43	0944	44	0945	45	0946	46	0947	47	0948	48	0949	49
094A	4A	094B	4B	094C	4C	094D	4D	094E	4E	094F	4F	0950	50	0951	51	0952	52	0953	53
0954	54	0955	55	0956	56	0957	57	0958	58	0959	59	095A	5A	095B	5B	095C	5C	095D	5D
095E	5E	095F	5F	0960	60	0961	61	0962	62	0963	63	0964	64	0965	65	0966	66	0967	67
0968	68	0969	69	096A	6A	096B	6B	096C	6C	096D	6D	096E	6E	096F	6F	0970	70	0971	71
0972	72	0973	73	0974	74	0975	75	0976	76	0977	77	0978	78	0979	79	097A	7A	097B	7B
097C	7C	097D	7D	097E	7E	097F	7F	0980	80	0981	81	0982	82	0983	83	0984	84	0985	85
0986	86	0987	87	0988	88	0989	89	098A	8A	098B	8B	098C	8C	098D	8D	098E	8E	098F	8F
0990	90	0991	91	0992	92	0993	93	0994	94	0995	95	0996	96	0997	97	0998	98	0999	99
099A	9A	099B	9B	099C	9C	099D	9D	099E	9E	099F	9F	09A0	A0	09A1	A1	09A2	A2	09A3	A3
09A4	A4	09A5	A5	09A6	A6	09A7	A7	09A8	A8	09A9	A9	09AA	AA	09AB	AB	09AC	AC	09AD	AD
09AE	AE	09AF	AF	09B0	B0	09B1	B1	09B2	B2	09B3	B3	09B4	B4	09B5	B5	09B6	B6	09B7	B7
09B8	B8	09B9	B9	09BA	BA	09BB	BB	09BC	BC	09BD	BD	09BE	BE	09BF	BF	09C0	C0	09C1	C1
09C2	C2	09C3	C3	09C4	C4	09C5	C5	09C6	C6	09C7	C7	09C8	C8	09C9	C9	09CA	CA	09CB	CB
09CC	CC	09CD	CD	09CE	CE	09CF	CF	09D0	D0	09D1	D1	09D2	D2	09D3	D3	09D4	D4	09D5	D5
09D6	D6	09D7	D7	09D8	D8	09D9	D9	09DA	DA	09DB	DB	09DC	DC	09DD	DD	09DE	DE	09DF	DF
09E0	E0	09E1	E1	09E2	E2	09E3	E3	09E4	E4	09E5	E5	09E6	E6	09E7	E7	09E8	E8	09E9	E9
09EA	EA	09EB	EB	09EC	EC	09ED	ED	09EE	EE	09EF	EF	09F0	F0	09F1	F1	09F2	F2	09F3	F3
09F4	F4	09F5	F5	09F6	F6	09F7	F7	09F8	F8	09F9	F9	09FA	FA	09FB	FB	09FC	FC	09FD	FD
09FE	FE	09FF	FF	0A00	00	0A01	00	0A02	00	0A03	00	0A04	00	0A05	00	0A06	00	0A07	00

β)

Για να βρούμε τον συνολικό αριθμό άσσων στη παραπάνω ακολουθία αριθμών έχουμε έναν counter που θα τον αυξάνουμε κατά 1 κάθε φορά που θα βρίσκουμε έναν άσσο σε κάθε αριθμό. Για την ακρίβεια για κάθε αριθμό κάνουμε 8 RRC και όποτε το CY γίνεται 1 αυξάνουμε τον counter κατά 1. Το κάνουμε για όλους τους αριθμούς και βρίσκουμε τους συνολικούς άσσους. Η απάντηση είναι:



Το αποτέλεσμα είναι το αναμενόμενο καθώς ο συνολικός αριθμός των άσσων είναι:

2048/2 = 1024 = 0400H

γ)

Το αποτέλεσμα πρέπει να είναι το 51H = 81. Το αποτέλεσμα αυτό βγαίνει και από το κώδικά μας (εμφανίζεται στον καταχωρητή D):


```
;a)
IN 10H
MVI A,00H ;counter 0-255
LXI H,0900H ;starting memory
MOV M,A ;store A in 0900

FOR:
        INX H ;HL<-HL+1 (next memory location)
        INR A ;increase counter
        MOV M,A ;store A in memory
        CPI FFH ;if equals to 255 stop
        JNZ FOR ;if not continue

;b)
LXI B,0000H
LXI H,0900H
MOV E,M ;number counter

TOTAL_ONES:
        MVI D,00H ;for 8 bits counter

        COUNT_ONES:
```

```
                MOV A,E
                RRC
                JNC SKIP_ME ;if CY != 1 skip increasing counter
                INX B
                SKIP_ME:
                        INR D ;increase 8 bit counter
                        MOV E,A ; shifted for next bit
                        MOV A,D
                        CPI 08H ; if 8th bit -> next number
                        JNZ COUNT_ONES
                        MOV A,E
                        CPI FFH ;end if 255
                        JZ TASK_C
                        INR H ;NEXT NUMBER
                        MOV E,M ;number counter
                        JMP TOTAL_ONES

;c)

TASK_C:

MVI D,00H
LXI H,0900H
MVI B,00H ;INDEX, BCS ARRAY MAY NOT BE IN ascending order

FIND:
        MOV A,M ;number counter
        CPI 10H ; PRATKIKA ELEGXW EAN TO INDEX EINAI MESA STO RANGE 00H-FFH stis
                ; parakatw grammes kwdika
        JC SKIP
        CPI 61H
        JNC SKIP
        INR D
        SKIP:
                MOV C,A
                MOV A,B
                CPI FFH ;if 255 stop
                JZ TASK_END
                MOV A,C
                INR B
        INX H
        JMP FIND


TASK_END:
RST 1
END
```

2η Άσκηση

Ελέγχουμε κάθε 0.1s (με χρήση της CALL DELB) την κατάσταση του MSB των dip switches. Αρχικά προφανώς τα dip switches είναι αρχικοποιημένα στη τιμή OFF. Εάν το MSB γίνει κάποια στιγμή 1 τότε περιμένουμε για να μας ξαναέρθει OFF προκειμένου να ανάψουμε τα LED αλλιώς περιμένουμε 0.1s (αυξάνοντας τον καταχωρητή D (καταχωρητής για να ελέγχουμε την ισότητα με το 200 (200*0.1 = 20s)) κατά 1) και ξαναελέγχουμε το MSB των dip switches. Όταν γίνει αυτό (δηλαδή μας έρθει πάλι OFF) αρχικά μηδενίζουμε τον μετρητή για τα 20s(D) και στη συνέχεια πάμε σε μία κατάσταση όπου περιμένουμε για ON (δηλαδή να γίνει 1 το MSB των dip switches), αλλιώς περιμένουμε 0.1s(αυξάνοντας τον καταχωρητή D (καταχωρητής για να ελέγχουμε την ισότητα με το 200 (200*0.1 = 20s)) κατά 1) και ξαναελέγχουμε το MSB των dip switches. Και στις δύο καταστάσεις (WAIT_FOR_OFF και WAIT_FOR_ON) αυξάνουμε τον καταχωρητή για τον χρόνο(D) κάθε 0.1s και όποτε γίνεται ίσος με 200 (200*0.1s = 20s) σβήνουμε τα LEDs και συνεχίζουμε τον κώδικα από εκεί που είχαμε μείνει.

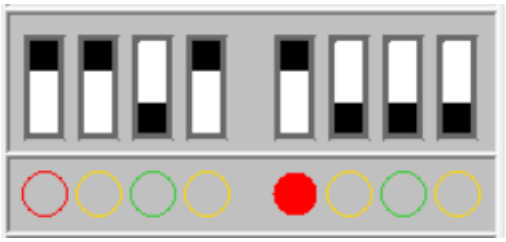
```
LXI B,0064H ;DELAY 0.1S
MVI D,00H ;TIMER
MVI E,00H ; ALL LEDS ON


WAIT_FOR_ON:
CALL DELB ;WAIT 0.1s
MOV A,E
CMA
STA 3000H ;LOAD CURRENT STATE TO LEDS
INR D ;INCREASE TIMER
MOV A,D
CPI C8H ;IF 20SEC
JZ TURN_OFF_ON ;THEN INITIALIZE CLOSE LEDS (ON DIDNT COME)
CONTINUE_ON:
LDA 2000H ;READ INPUT
ANI 80H
CPI 80H ;CHECK IF ON CAME (MSB = 1)
JZ WAIT_FOR_OFF ;IF YES WAIT FOR OFF NOW
JMP WAIT_FOR_ON ;ELSE WAIT AGAIN FOR OFF


TURN_OFF_ON: ;JUST TURN OFF LEDS BCS 20s PASSED
MVI E,00H
MOV A,E
CMA
STA 3000H
JMP CONTINUE_ON


WAIT_FOR_OFF:
CALL DELB
MOV A,E
CMA
```

```
STA 3000H ;LOAD CURRENT STATE TO LEDS
INR D ;INCREASE TIMER
MOV A,D
CPI C8H ;IF 20SEC
JZ TURN_OFF_OFF ; THEN INITIALIZE CLOSE LEDS (OFF DIDNT COME)
CONTINUE_OFF:
LDA 2000H ;READ INPUT
ANI 80H
CPI 00H ;CHECK IF OFF CAME (MSB = 0)
JNZ WAIT_FOR_OFF ;IF NOT, REPEAT WAIT FOR OFF
MVI D,00H ;ELSE INITIALIZE AGAIN TIMER TO 0
MVI E,FFH ;CURRENT STATE OF LEDS NOW IS: ON
JMP WAIT_FOR_ON  ;WAIT FOR ON NOW


TURN_OFF_OFF: ;JUST TURN OFF LEDS BCS 20s PASSED
MVI E,00H
MOV A,E
CMA
STA 3000H
JMP CONTINUE_OFF

RST 1
END
```

3η Άσκηση

i)

Αρχικά εάν ο αριθμός είναι ίσος με 0 τότε απλά σβήνει όλα τα LEDS, αλλιώς ολισθαίνουμε συνεχώς προς τα δεξιά τον A μέχρι να βρεθεί ο πρώτος άσσος του και ανάλογα με το πόσες επαναλήψεις αυξάνουμε τον B. Στη συνέχεια περνάμε τον B στον A, αρχικοποιούμε τον C με το 00000001, και μέχρι να γίνει ο A ίσος με τον 01H (τότε θα έχει ο C το position του LED που θέλουμε να ανάψουμε) (τον μειώνουμε κατά ένα σε κάθε λούπα) ολισθαίνουμε τον C προς τα αριστερά.
Ένα παράδειγμα, αξιοποιώντας το μLab είναι:



```
START:
LDA 2000H
CPI 00H ;if number is zero read next input
JZ ZERO
MVI B,00H ;final position of 1 (int)



FOR:
INR B
RRC   ;if CY = 0 then loop
JNC FOR

MVI C,01H ;LED OUTPUT (by the time is not zero
          ;it is initialized to 00000001)
MOV A,B   ;position of first 1

FOR_:
CPI 01H ;if A = 00000001 then end
JZ LOAD
MOV D,A ;save position
MOV A,C ;by the time A is not 01 move position 1 pos
        ;left
RLC
MOV C,A
MOV A,D
DCR A   ;decrease A until 01
JMP FOR_


LOAD:
MOV A,C ;save LED state to A
```

```
ZERO:
CMA
STA 3000H
RST 1
JMP START
END
```

ii)

Αξιοποιούμε την ρουτίνα KIND προκειμένου να διαβάσουμε από το πληκτρολόγιο. Στη συνέχεια αρχικοποιούμε τον B που θα έχει το τελικό επιθυμητό αποτέλεσμα ενώ αρχικοποιούμε τον C στο 01H (του οποίου θα του κάνουμε κάθε φορά RLC μέσω του A). Μετά μπαίνουμε σε μία λούπα όπου μέχρι ο A να μηδενιστεί κάνουμε OR μεταξύ του B και του C (μέσω του A προφανώς) και μετατοπίζουμε το C μία θέση αριστερά. Κατά αυτό το τρόπο ο B θα έχει το επιθυμητό αποτέλεσμα (ήδη σε ανάστροφη λογική για το συγκεκριμένο αποτέλεσμα).
Για παράδειγμα εάν διαβάσω από το πληκτρολόγιο 3 τότε:
B → 00

C → 01
B → B OR C = 01
A → 2

C → 10
B → B OR C = 11
A → 1

C → 100
B → B OR C = 111
A → 0
stop

Ένα παράδειγμα εκτέλεσης για keyboard input = 4 θα έχει ως αποτέλεσμα (στο µLab):

```
START:
CALL KIND ;read input from keyboard

MVI C,01H ;01->10->100 ETC
MVI B,00H ;result

FOR:
DCR A
CPI 00H ;while A is not 0
JZ STOP
MOV D,A ;save A
```

```
MOV A,B ;result to A
ORA C   ;and with C (next 1 added) (i.e. 0000011-> 0000111)
MOV B,A ;current result to B
MOV A,C ;next C
RLC
MOV C,A ;Save next C to C
MOV A,D ;get A
JMP FOR ;repeat


STOP:
MOV A,B
STA 3000H ;ALERT: NO NEED FOR CMA, LED STATE IS ALREADY FIXED
          ;ALREADY IN INVERSE LOGIC
RST 1
JMP START
END
```

iii) Το πρόγραμμα ακολουθεί την διαδικασία των σημειώσεων. Το πρόγραμμα διαβάζει μία προς μία γραμμή του πληκτρολογίου (2800H) και ελέγχει την θέση μνήμης 1800H. Όταν πατηθεί ένα κουμπί κάνει jump στην αντίστοιχη υπο-ρουτίνα στην οποία απλά θα τυπώσει στην οθόνη (στις 2 αριστερότερες θέσεις 0B04 και 0B05) αυτό που αντιστοιχεί στο πλήκτρο που πατήθηκε με χρήση των STDM και DCD.

```
IN 10H

START:
MVI A,10H
LXI D,0B00H
STA 0B00H
STA 0B01H
STA 0B02H
STA 0B03H    ;4 rightmost 7-segment displays need to be "empty" (empty has code 10)

MVI A,FEH ;A = 1111 1110  -- first line
STA 2800H
LDA 1800H
ANI 03H       ;keep only the 2 LSBs because we only care about instr step and fetch pc buttons
CPI 02H
JZ CODE_INSTR_STEP ;INSTR_STEP is pressed
CPI 01H
JZ CODE_FETCH_PC         ;FETCH PC is pressed, etc...

MVI A,FDH ;A = 1111 1101 -- 2nd line
STA 2800H
LDA 1800H
ANI 07H ;keep only the 3 LSBs
CPI 06H
```

```
JZ CODE_RUN
CPI 05H
JZ CODE_FETCH_REG
CPI 03H
JZ CODE_FETCH_ADRS

MVI A,FBH ;A = 1111 1011 – 3rd line
STA 2800H
LDA 1800H
ANI 07H ;keep only the 3 LSBs
CPI 06H
JZ CODE_0
CPI 05H
JZ CODE_STORE/INCR
CPI 03H
JZ CODE_DECR

MVI A,F7H ;A = 1111 0111 -- 4th line
STA 2800H
LDA 1800H
ANI 07H ;keep only the 3 LSBs
CPI 06H
JZ CODE_1
CPI 05H
JZ CODE_2
CPI 03H
JZ CODE_3

MVI A,EFH ;A = 1110 1111  -- 5th line
STA 2800H
LDA 1800H
ANI 07H ;keep only the 3 LSBs
CPI 06H
JZ CODE_4
CPI 05H
JZ CODE_5
CPI 03H
JZ CODE_6

MVI A,DFH ;A = 1101 1111 -- 6th line
STA 2800H
LDA 1800H
ANI 07H ;keep only the 3 LSBs
CPI 06H
JZ CODE_7
CPI 05H
JZ CODE_8
CPI 03H
JZ CODE_9
```

```
MVI A,BFH ; A = 1011 1111 -- 7th line
STA 2800H
LDA 1800H
ANI 07H ; keep only the 3 LSBs
CPI 06H
JZ CODE_A
CPI 05H
JZ CODE_B
CPI 03H
JZ CODE_C

MVI A,7FH ;A = 0111 1111 -- 8th line
STA 2800H
LDA 1800H
ANI 07H ;keep only the 3 LSBs
CPI 06H
JZ CODE_D
CPI 05H
JZ CODE_E
CPI 03H
JZ CODE_F
JMP START

CODE_INSTR_STEP:
MVI A,06H    ; this happens if INSTR_STEP is pressed
STA 0B04H
MVI A,08H
STA 0B05H
JMP FINISH

CODE_FETCH_PC:
MVI A,05H ;same for FETCH_PC, etc...
STA 0B04H
MVI A,08H
STA 0B05H
JMP FINISH

CODE_RUN:
MVI A,04H
STA 0B04H
MVI A,08H
STA 0B05H
JMP FINISH

CODE_FETCH_REG:
MVI A,00H
STA 0B04H
MVI A,08H
```

```
STA 0B05H
JMP FINISH

CODE_FETCH_ADRS:
MVI A,02H
STA 0B04H
MVI A,08H
STA 0B05H
JMP FINISH

CODE_0:
MVI A,00H
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_STORE/INCR:
MVI A,03H
STA 0B04H
MVI A,08H
STA 0B05H
JMP FINISH

CODE_DECR:
MVI A,01H
STA 0B04H
MVI A,08H
STA 0B05H
JMP FINISH

CODE_1:
MVI A,01H
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_2:
MVI A,02H
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_3:
MVI A,03H
STA 0B04H
MVI A,00H
```

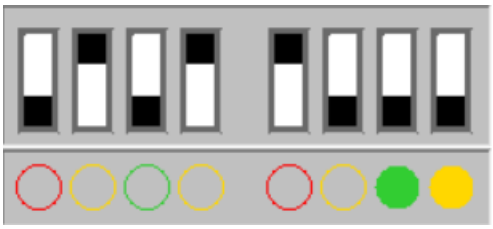
```
STA 0B05H
JMP FINISH

CODE_4:
MVI A,04H
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_5:
MVI A,05H
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_6:
MVI A,06H
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_7:
MVI A,07H
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_8:
MVI A,08H
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_9:
MVI A,09H
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_A:
MVI A,0AH
STA 0B04H
MVI A,00H
```

```
STA 0B05H
JMP FINISH

CODE_B:
MVI A,0BH
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_C:
MVI A,0CH
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_D:
MVI A,0DH
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_E:
MVI A,0EH
STA 0B04H
MVI A,00H
STA 0B05H
JMP FINISH

CODE_F:
MVI A,0FH
STA 0B04H
MVI A,00H
STA 0B05H

FINISH:
CALL STDM
CALL DCD
JMP START

END
```

4η Άσκηση

Αρχικά αποθηκεύω τον A στον B, μετατοπίζω τον A (ας τον πω initial για ευκολία) και τον αποθηκεύω στον D. Κάνω AND τον A με τον B και απομονώνω το 4 bit το οποίο και θα είναι το X3. Στη συνέχεια επαναφέρω τον initial στον A και τον κάνω AND με τον B και απομονώνω το 4 bit το οποίο τώρα θα είναι το X2. Με την παραπάνω λογική και με κατάλληλες μετατοπίσεις υλοποιείται και το υπόλοιπο σύστημα. Η ακριβής περιγραφή είναι σε σχόλια μέσα στον κώδικα που θα παραδοθεί στο zip αρχείο. Για επαλήθευση της ορθότητας του προγράμματος μας παρακάτω είναι ένα παράδειγμα:

για input = 01011000 η έξοδος θα έπρεπε να ήταν 00000011 και στο μLab βγάζουμε:

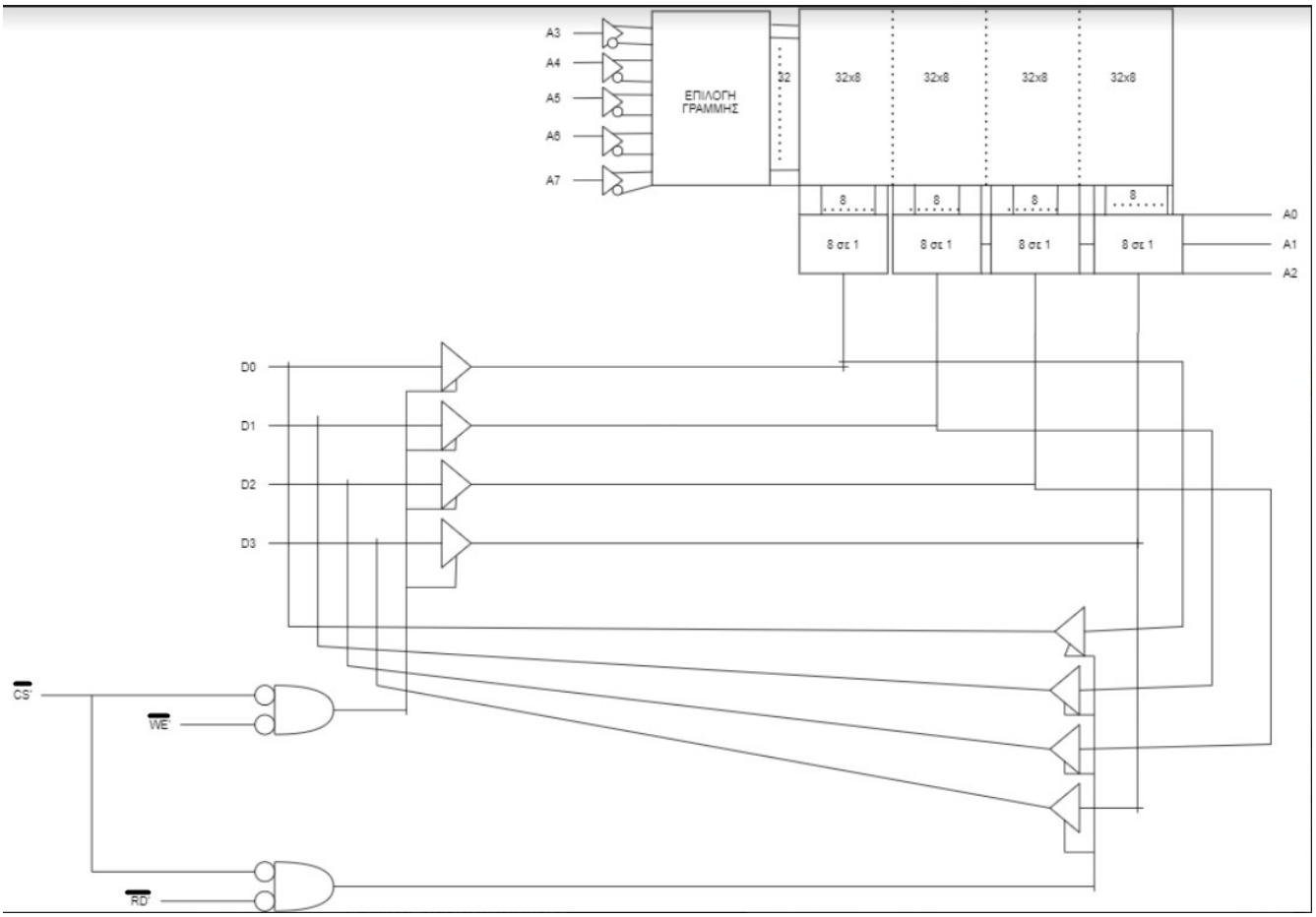


```
START:
LDA 2000H ;Load dip switch
MOV B,A   ;Intial number
RRC
MOV D,A      ; Right shift that A1 and B1 are in the same position

ANA B              ;Store AND A3-B3 at A
RRC
RRC
RRC          ;Result is sent to X3
ANI 08H      ;Keeps only X3
MOV E,A      ;E is the current result
MOV A,D      ; Load the shifted value

ANA B              ;Store AND A2-B2 at A
RRC          ;Result is sent to X3
ANI 08H      ;apomonwnetai
ORA E        ;OR between A2 AND B2 and A3 AND B3 to X3
RRC          ;Move to X2
ADD E
MOV E,A      ;Store A to E
MOV A,D      ;Load the shifted value

XRA B              ;XOR A1 and B1
RRC          ; Move to X1
ANI 02H      ; Keep only this bit
MOV C,A      ;Store it to use later
ADD E
MOV E,A      ; Store A to E
```

```
MOV A,C     ;Store previous result A1 XOR B1 to A
RRC         ;Move to X0
MOV C,A     ;Store A to C
MOV A,D     ;Load again the shifted value
XRA B               ; A0 XOR B0
ANI 01H     ;Keep only this bit
XRA C               ;XOR of A1 XOR B1 and A0 XOR B0
ADD E               ;Now A has the final output
CMA
STA 3000H

JMP START
END
```

5η Άσκηση

Σπάσαμε την μνήμη των 256x4 σε 32 γραμμές και σε 32 στήλες (ως 8 4τραδες bit, αφού έχουμε 32 γραμμές). Κατά αυτό το τρόπο όταν θέλουμε να διασχίσουμε συνεχόμενα δεδομένα (πχ έναν πίνακα που είναι αποθηκευμένος σε διαδοχικές θέσεις μνήμης) θα χρειάζεται να αλλάξουμε μόνο τα 3 τελευταία bits της διευθύνσεις ενώ τα υπόλοιπα 5 θα παραμένουν σταθερά! Ποιο συγκεκριμένα η εσωτερική οργάνωση αυτής της μνήμης SRAM είναι:



Για να γράψουμε κάτι στη μνήμη θα πρέπει να ενεργοποιήσουμε αρχικά τα κατάλληλα σήματα και λόγο ανάστροφης λογικής θα πρέπει CS’ = 0 και WE’ = 0. Τότε θα ενεργοποιηθεί η αριστερή “ομάδα” των buffers που καταλήγουν στους 8 σε 1 πολυπλέκτες. Κατά αυτό το τρόπο τα bits στα D0, D1 ,D2 ,D3, μέσω τον πολυπλεκτών και μίας αντίστοιχης διεύθυνσής(A0-A7, A0-A2 για τους 8-1 MUX για επιλογή τεσσάρων στηλών και A3-A7 για την επιλογή γραμμής) (προκειμένου να ενεργοποιήσουμε και τις θέσεις μνήμης στις οποίες θέλουμε να γράψουμε), θα γραφούν (ταυτόχρονα) μέσα στην SRAM.

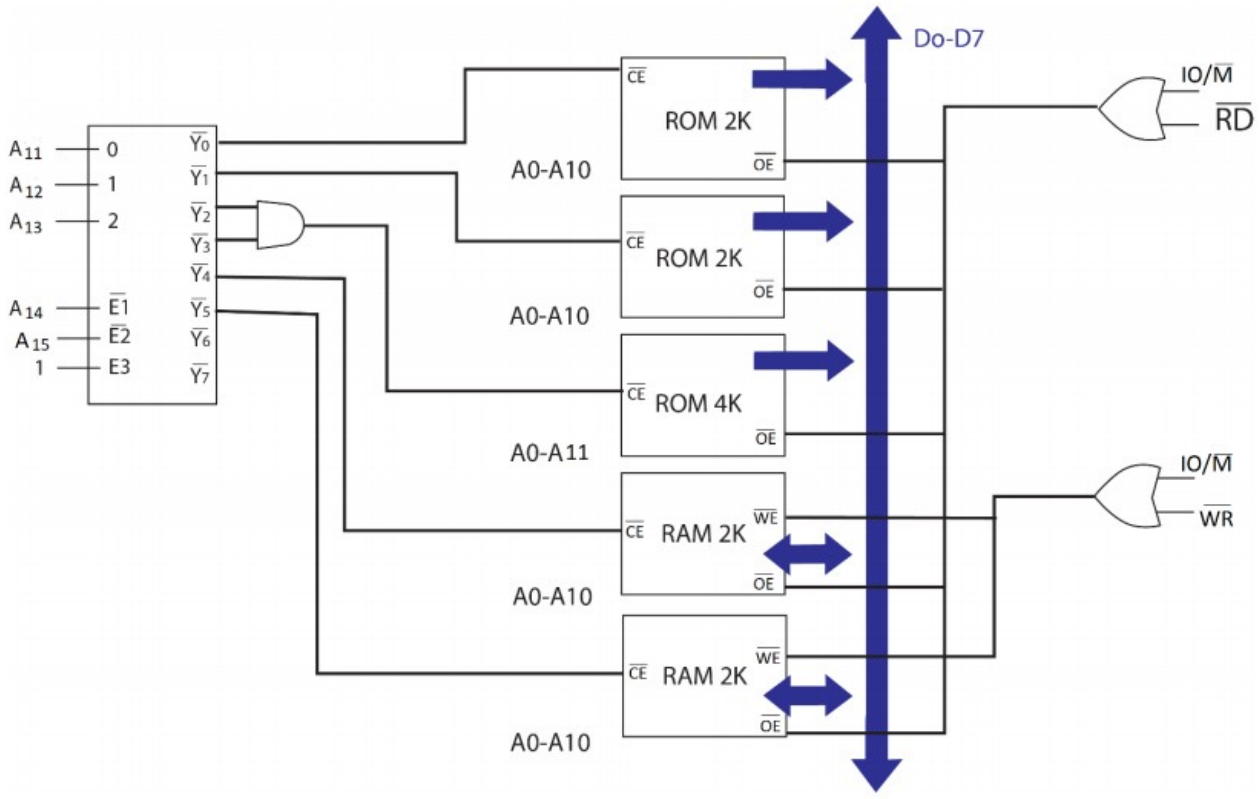
Όσον αφορά την ανάγνωση από τη SRAM θα πρέπει να ενεργοποιήσουμε αρχικά τα κατάλληλα σήματα και λόγο ανάστροφης λογικής θα πρέπει CS’ = 0 και RD’ = 0. Τότε θα ενεργοποιηθεί η δεξιά “ομάδα” των buffers που ξεκινάνε από τους 8 σε 1 πολυπλέκτες. Κατά αυτό το τρόπο θα διαβαστούν τα bits από τις εξόδους των πολυπλεκτών, μέσω μίας αντίστοιχης διεύθυνσής(A0-A7, A0-A2 για τους 8-1 MUX για επιλογή τεσσάρων στηλών και A3-A7 για την επιλογή γραμμής) (προκειμένου να ενεργοποιήσουμε και τις θέσεις μνήμης στις οποίες θέλουμε να διαβάσουμε), προκειμένου να γραφούν (ταυτόχρονα) τα δεδομένα από τη μνήμη στα D0, D1, D2, D3.

6η Άσκηση

Ο ζητούμενος χάρτης μνήμης είναι:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	address	memory
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	ROM 2K_1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	07FFH	
0	0	0	0	1	0	0		0	0	0	0	0	0	0	0	0800H	ROM 2K_2
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFFH	
0		0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H	ROM 4K_3
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H	RAM 2K_1
0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	27FFH	
0		1	0	1	0	0	0	0	0	0	0	0	0	0	0	2800H	RAM 2K_2
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	2FFFH	

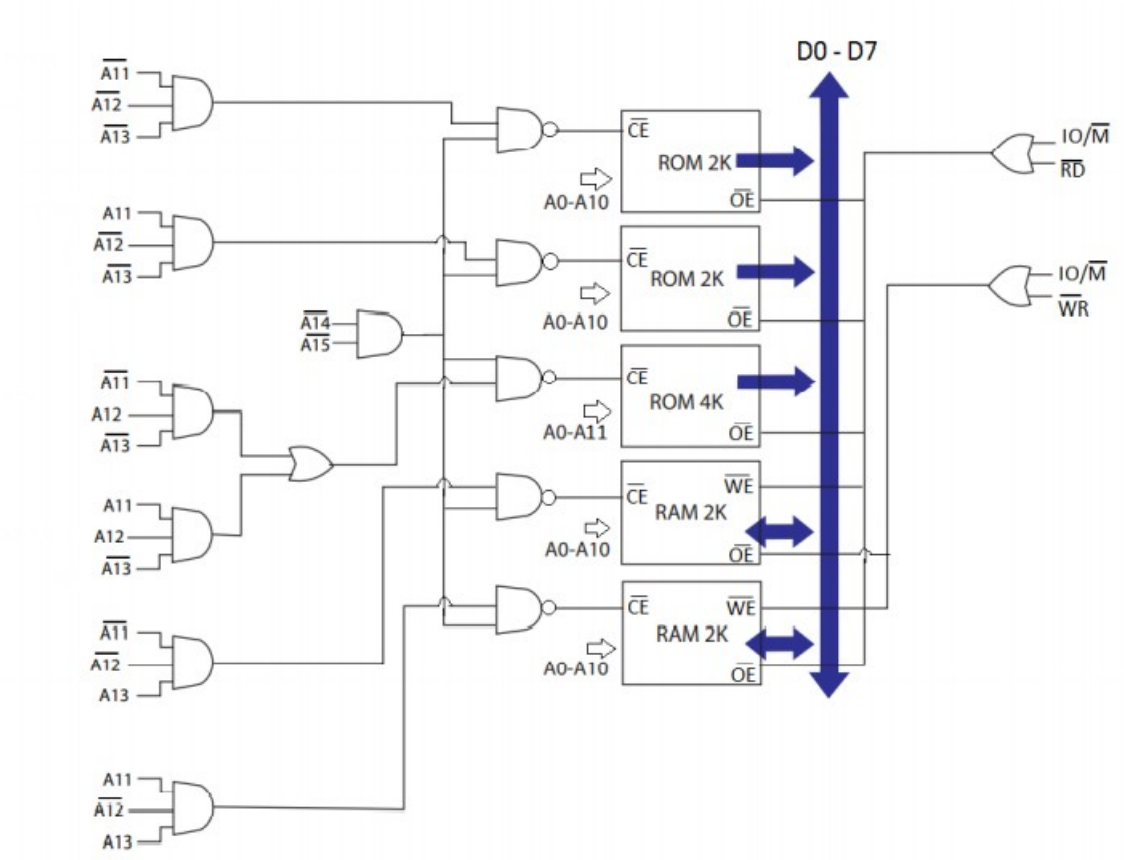
α) Το ζητούμενο κύκλωμα με αποκωδικοποιητή 3:8 και πύλες:



Επιλέξαμε τα bits A11, A12, A13 προκειμένου να αποφανθούμε με ποια μνήμη θέλουμε να “επικοινωνήσουμε”. Αξιοσημείωτο είναι το γεγονός ότι χρησιμοποιήσαμε μία AND πύλη (λόγο ανάστροφης λογικής, αλλιώς θα ήταν OR) προκειμένου να “ενεργοποιήσουμε” την RAM των 4K καθώς είτε έχουμε Y2 είτε Y3 τότε αναφερόμαστε σε αυτήν (στο χάρτη μνήμης είτε A11 A12 A13 = 010 είτε A11 A12 A13 = 011 να έχουμε τότε πάλι αναφερόμαστε στην RAM των 4K).

β) Παρακάτω φαίνεται το ζητούμενο κύκλωμα χρησιμοποιώντας μόνο λογικές πύλες:

β) Με λογικές πύλες:



Παρατηρούμε ότι και με χρήση πυλών το κύκλωμα είναι εξίσου απλό!

7η Άσκηση

Ο ζητούμενος χάρτης μνήμης είναι:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	address	memory
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	ROM 16K_1
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	2FFFH	
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3000H	RAM 4K_1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000H	RAM 4K_2
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	4FFFH	
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	5000H	RAM 4K_3
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFFH	
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000H	ROM 16K_1
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	6FFFH	
0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	7000H	Θύρα εξόδου

Επιλέγουμε τα A12, A13, A14 προκειμένου να μας προσδιορίσουν σε ποια μνήμη αναφερόμαστε. Για να καταλάβουμε πότε αναφερόμαστε στη θύρα εισόδου (70H) αξιοποιούμε διευθύνσεις που έχουν το εξής φορμάτ:

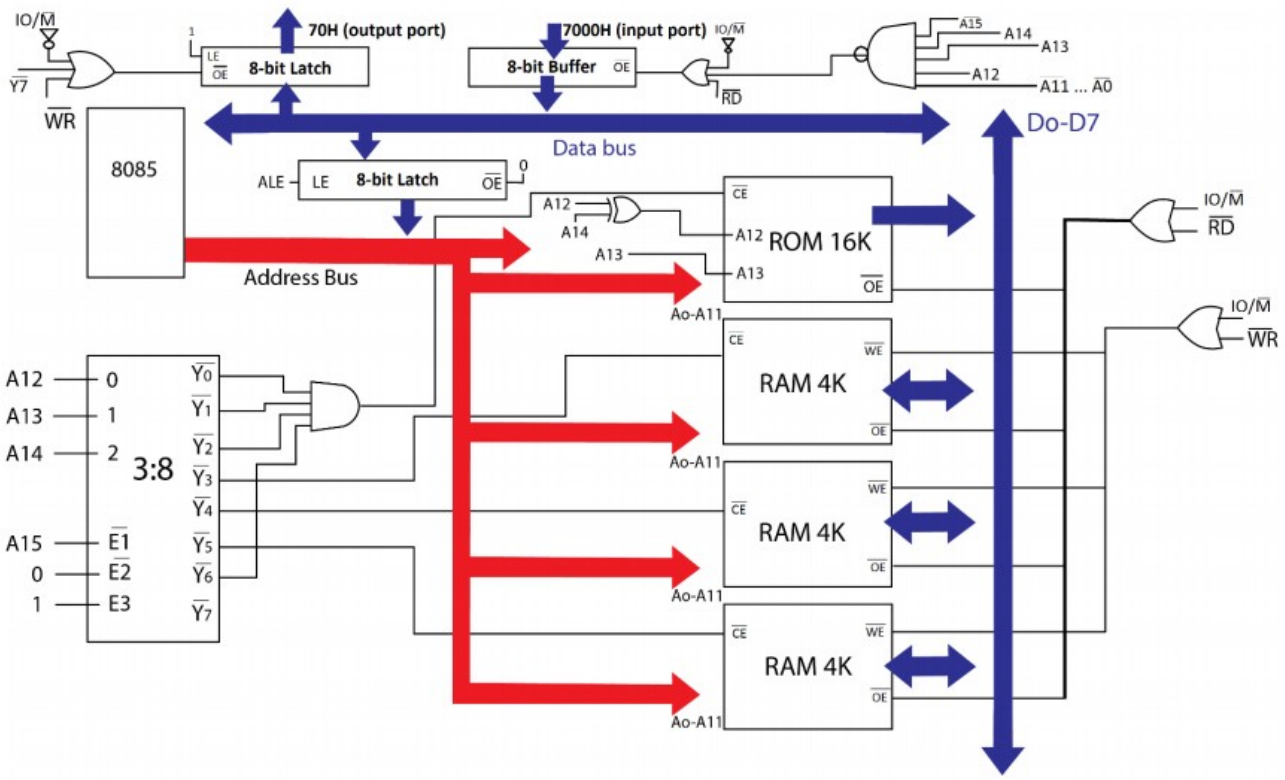
0111XXXXXXXXXXX

μαζί προφανώς και με κατάλληλα σήματα (πχ. IO/M’)

Για τη θύρα εξόδου (7000H) ελέγχουμε για τη συγκεκριμένη διεύθυνση:

011100000000

Το κύκλωμα του μΥ-Σ 8085 με το παραπάνω χάρτη έχει το παρακάτω κύκλωμα:



Προκειμένου να χωρίσουμε τη ROM σε δύο μέρη (σε μία 14K και σε μία 4K και σε μη διαδοχικές θέσεις μνήμης (14K → 0000H:2FFFH και 4K → 6000H:6FFFH)) πρέπει να προσθέσουμε ένα εξτρά κύκλωμα προκειμένου να “σειριοποιήσουμε” τις θέσεις μνήμης μέσα στη ROM, καθώς η 14K και η 4K εν τέλη μέσα στη 16K ROM είναι σε διαδοχικές θέσεις μνήμης (επιπροσθέτως χρειαζόμαστε 14bits καθώς είναι μεγέθους 16K, άρα θα κρατήσουμε τα A0-A13). Δηλαδή προσθέτουμε ένα XOR με εισόδους το A12 και A14, το οποίο καταλήγει στο A12 της ROM, έτσι ώστε όταν είμαστε στα πρώτα 12K της ROM A14 = 0 και άρα η XOR θα μας δώσει A12 ενώ τα A13 και A12 παίρνουν τιμές 00, 01, 10 (αυξάνονται δηλαδή σειριακά). Στη συνέχεια για τα υπόλοιπα 4K θα πρέπει να είναι τα A13 και A12 (που μπαίνουν μέσα στη ROM και όχι τα αρχικά) 11 (και μόνο 11 γιατί τα A13 A12 = 10 πάντα σε αυτή τη περιοχή διευθύνσεων) προκειμένου να επιτευχθεί αυτή η “σειριοποίηση” των διευθύνσεων μέσα στη ROM όπως αναφέραμε παραπάνω. Με την XOR αυτή επιτυγχάνεται αυτό καθώς στο πεδίο των 4K για τη ROM παρατηρούμε ότι A14 = 1 και A12 = 0 πάντα και άρα η XOR θα μου δώσει αποτέλεσμα 1 και άρα παίρνουμε στη θέση του A12 1 πάντα με αποτέλεσμα να έχουμε A13 A12 = 11 πάντα (με αποτέλεσμα να επιτυγχάνεται η “σειριοποίηση” καθώς προηγουμένως είχαμε σταματήσει στο 10).

Όσον αφορά τις υπόλοιπες μνήμες γίνονται enabled με βάση τα A14 A13 A12 και έχουν ως input τα A0-A11 (4K μνήμες άρα χρειάζονται 12 bits).