ADVANCED MICROPROCESSORS & INTERFACING

ASSIGNMENT REPORT

Submitted by : (Roll no. wise)

- Hrishikesh Ugale
  BT17ECE026
- Hrithik Aditya
  BT17ECE027
- Pushpak Patil
  BT17ECE092

- **Problems Selected:**

1. Write an ALP to find out how many equal bytes are present between two data memory blocks [L1.6]
2. Write an ALP to store the following pattern, at the memory location starting from 40H..1H, 2H, 2H, 3H, 3H, 3H, 4H, 4H, 4H, 4H,.....upto 09H [L2.11]
3. Design a pre-settable alarm system using 8253/54 timer. Use thumbwheel switches to accept 4 digit value in seconds. Alarm should last for 10 seconds [L3.13]

- **Tools Used:**

1. Microsoft Macro Assembler (MASM) with DOSBOX
2. EMU8086
3. Proteus

Problem 1:

Write an ALP to find out how many equal bytes are present between two data memory blocks .

Program:

```
ASSUME CS:CODE, DS:DATA

DATA SEGMENT
BLOCK1 DB 12H, 15H, 34H, 47H, 0CAH, 35H, 92H, 21H, 26H, 32H
BLOCK2 DB 13H, 17H, 34H, 26H, 92H, 35H, 99H, 0A1H, 16H, 39H
BLOCKSIZE1 DB 0AH
BLOCKSIZE2 DB 0AH
RESULT DB 01H DUP(?)
DATA ENDS

CODE SEGMENT
START:
MOV AX, DATA
MOV DS, AX
MOV ES, AX
MOV SI, OFFSET BLOCK1
MOV DI, OFFSET BLOCK2
CLD
MOV BL, 00H
MOV DL, BLOCKSIZE1
L2:
MOV CH, 00H
MOV CL, BLOCKSIZE2
L1:
MOV AL, [SI]
SCASB
JNZ SKIP
INC BL
SKIP:
LOOP L1
INC SI
MOV DI, OFFSET BLOCK2
DEC DL
JNZ L2
MOV DI, OFFSET RESULT
```

Initialize Data Segment and Extra Segment Registers

Initialize Source Index and Destination Index Registers

Clear Direction Flag

Initialize result and counter 1 value

Initialize counter 2 value

Load byte of block1 pointed by SI into accumulator and compare it with other bytes of block 2. If found equal incremenr the result.

Continue the process for all bytes of block 1 by reloading the offset of block 2 in DI and incrementing SI.

Store the result in memory.

```
MOV [DI], BL
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

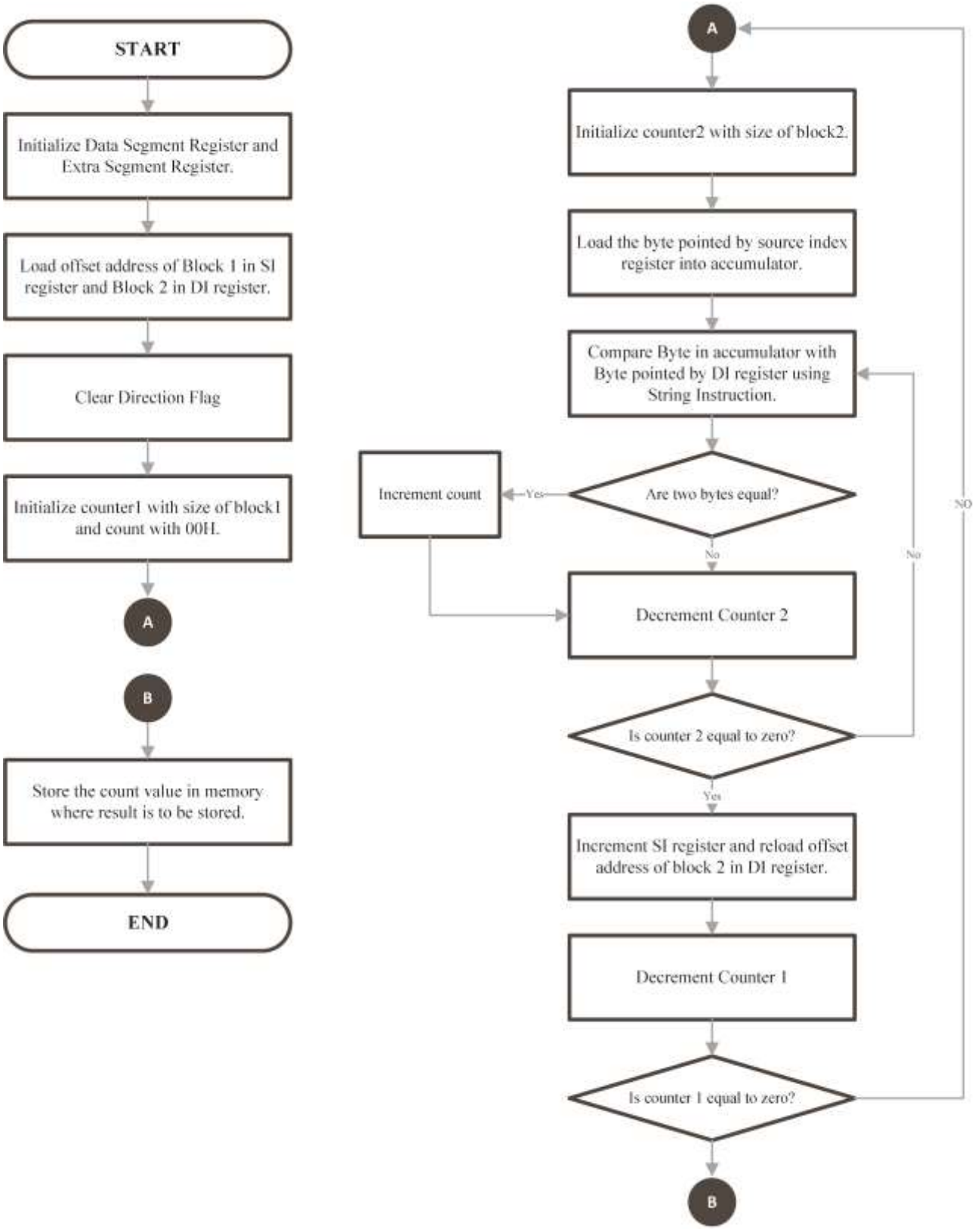
End the program

Algorithm:

Above program makes use of string instructions to find out the number of equal bytes present between two memory blocks. In order to use string instructions, we initialise one block in the data segment and another block in the extra segment. Offset address of block in Data segment is loaded in SI register and offset address of block in Extra segment is loaded in DI register. Direction Flag is cleared so that string instruction proceeds in incremental manner. SCASB instruction is used to compare two bytes - one stored in AL register and other pointed by DI register in Extra Segment. If two bytes are equal, the carry flag is set.

Initially offset addresses of block1 and block2 are loaded in SI and DI registers respectively. First byte from block1 is loaded into the AL register and then it is compared with every byte of block 2 in a loop. Whenever two bytes are found to be the same, count is incremented. This process is repeated in a loop for every byte of block1. At the end of the final iteration, we obtain a total number of equal bytes present in two memory blocks.

Flow Chart:


START
Initialize Data Segment Register and Extra Segment Register.
Load offset address of Block 1 in SI register and Block 2 in DI register.
Clear Direction Flag
Initialize counter1 with size of block1 and count with 00H.
A
B
Store the count value in memory where result is to be stored.
END
A
Initialize counter2 with size of block2.
Load the byte pointed by source index register into accumulator.
Compare Byte in accumulator with Byte pointed by DI register using String Instruction.
Are two bytes equal?
Yes
Increment count
No
Decrement Counter 2
Is counter 2 equal to zero?
No
Yes
Increment SI register and reload offset address of block 2 in DI register.
Decrement Counter 1
Is counter 1 equal to zero?
NO
B

Results:

Block1:

12H, 15H, **34H**, 47H, 0CAH, **35H**, **92H**, 21H, **26H**, 32H

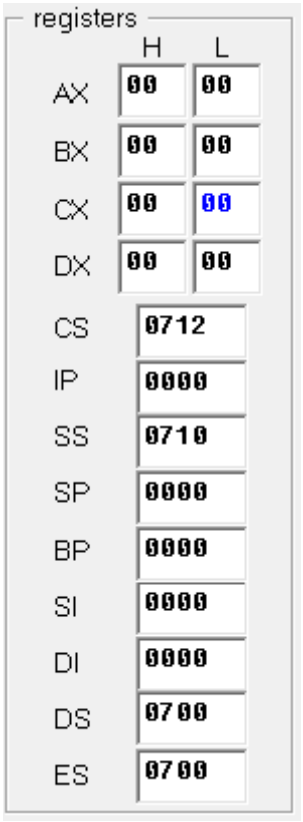
Block 2:

13H, 17H, **34H**, **26H**, **92H**, **35H**, 99H, 0A1H, 16H, 39H

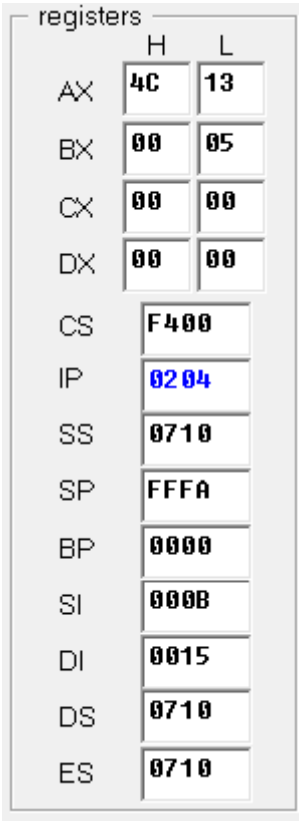
Clearly we can see that two memory blocks have 4 equal bytes.

BL Register stores the count of equal number of bytes.

Figure below shows the contents of registers of 8086 before and after simulation of above program in 8086 simulator - EMU8086



Before Execution



After Execution

Problem 2:

Write an ALP to store the following pattern, at the memory location starting from 40H..1H, 2H, 2H, 3H, 3H, 3H, 4H, 4H, 4H, 4H,.....upto 09H

Program:

```
ASSUME CS:CODE

CODE SEGMENT
START:
MOV AX, 3000H
MOV DS, AX
MOV SI, 40H
MOV CX, 0009H
L1:
MOV BL, CL
MOV AL, 0AH
SUB AL, BL
MOV BL, AL
MOV BH, AL
L2:
MOV DS:[SI], BH
INC SI
DEC BL
JNZ L2
LOOP L1
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Initialize Data Segment and Source Index Registers

Initialize primary count

Obtain secondary count and data by subtracting primary count from 0AH

Store data at location pointed by SI in data segment and increment SI. Repeat process in loop until secondary count bceomes zero

Loop until primary count becomes 0

End the program

Algorithm:

Above program stores the desired pattern in memory. Algorithm is to nest two loops one inside the other such that the count value of the outer loop decides the number of iterations in the inner loop. Initial step of the program is to initialize the data segment. CX register is loaded with primary count i.e count of distinct numbers to be stored in memory. This count value when subtracted from 0AH gives the value that is to be stored and number of times it is to be stored.

For example, the initial primary count is 09H.

0AH - 09H = 01H

This means that in the current iteration of the primary loop, 01H is to be stored 01 times. In the next iteration, primary count becomes 08H.

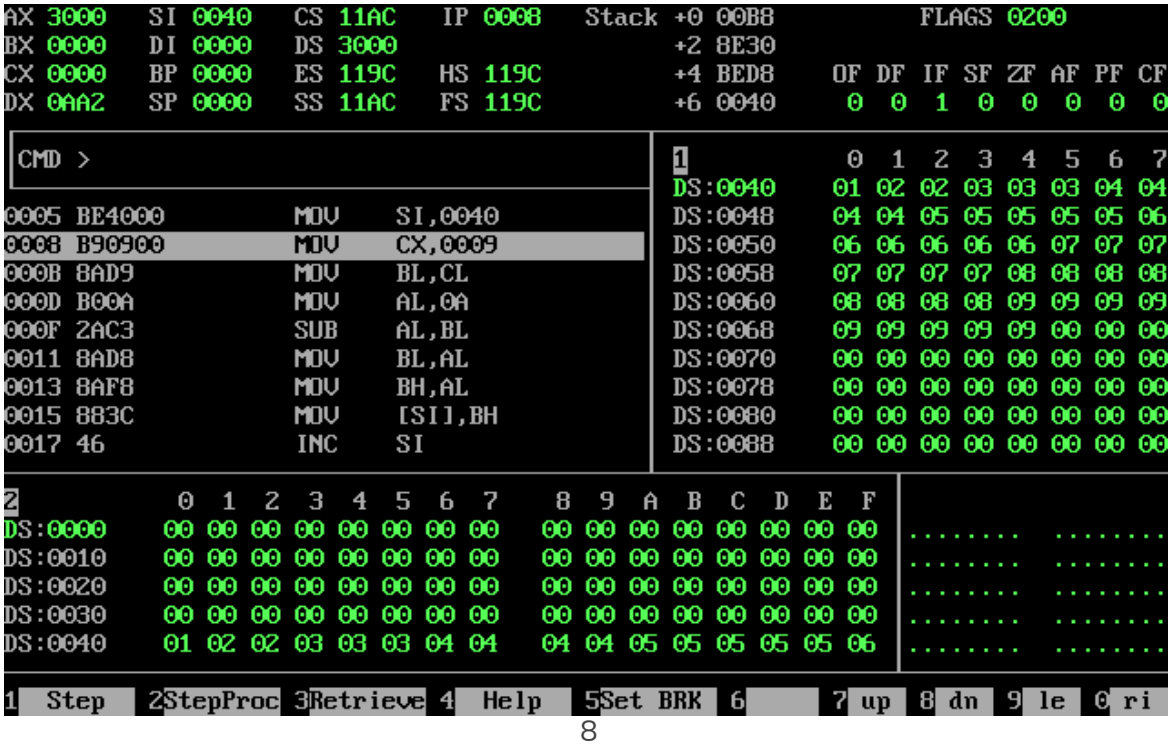
0AH - 09H = 02H

This means that in the current iteration of the primary loop, 02H is to be stored 02 times and this process continues. Thus secondary count is obtained by subtracting primary count from 0AH. Secondary loop then stores this value at a location pointed by SI in the data segment of memory. After storing the value of SI is incremented so that the next byte is stored at consecutive locations and not overwritten.

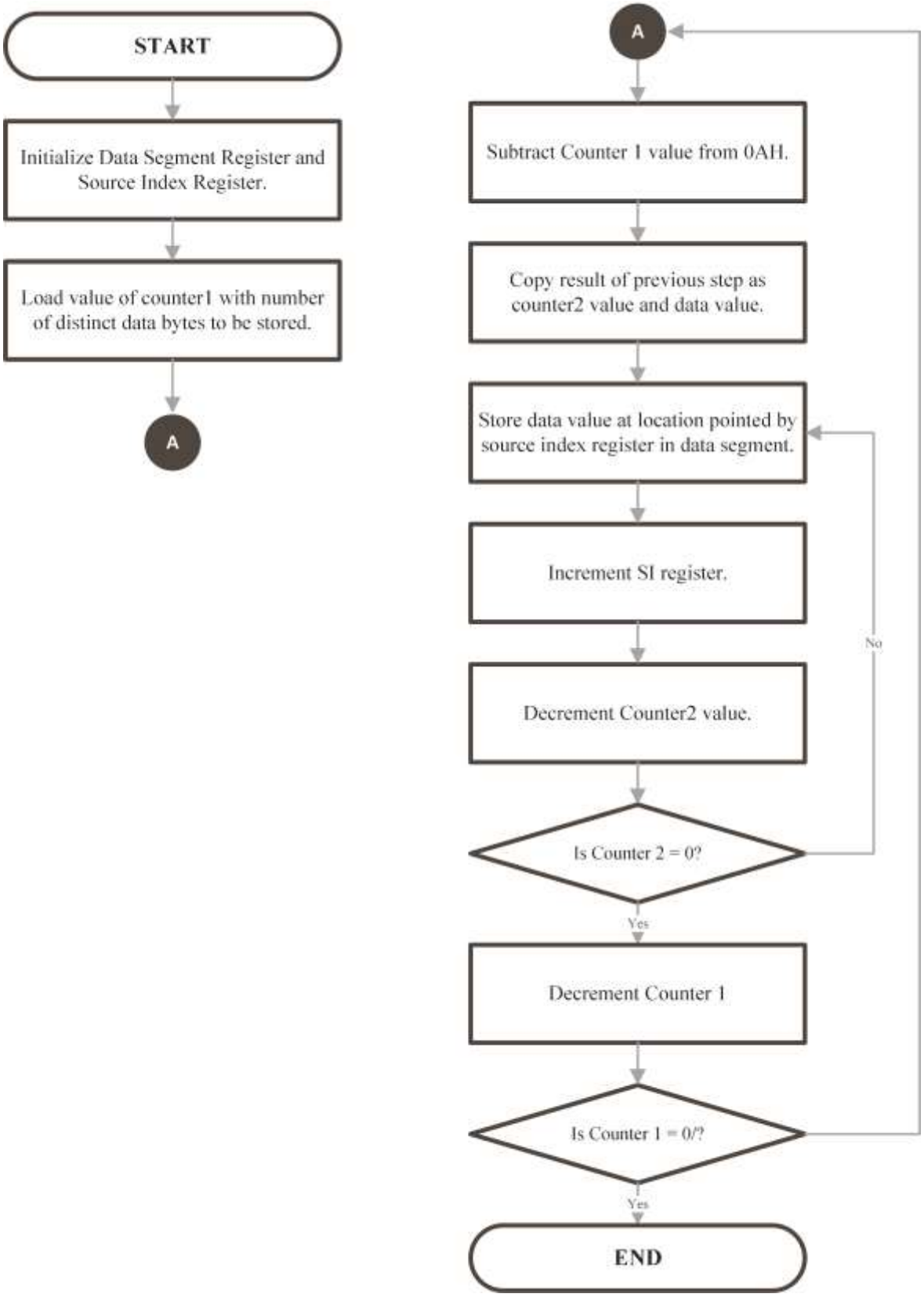
This process is continued till primary count becomes 00H and then the program is halted.

Results:

Memory map of MASM after execution of the above program after location 0040H is shown in the figure below.



Flow Chart:


START
Initialize Data Segment Register and Source Index Register.
Load value of counter1 with number of distinct data bytes to be stored.
A
A
Subtract Counter 1 value from 0AH.
Copy result of previous step as counter2 value and data value.
Store data value at location pointed by source index register in data segment.
Increment SI register.
Decrement Counter2 value.
Is Counter 2 = 0?
No
Yes
Decrement Counter 1
Is Counter 1 = 0/?
Yes
END

Problem 3

Design a pre-settable alarm system using 8253/54 timer. Use thumbwheel switches to accept 4 digit values in seconds. Alarm should last for 10 seconds

Program:

```
ASSUME CS:CODE, DS:DATA

DATA SEGMENT
      PORTA EQU 00F8H
      PORTB EQU 00FAH
      PORTC EQU 00FCH
      PORT_CWR EQU 00FEH
      CNTR0 EQU 0078H
      CNTR1 EQU 007AH
      CNTR2 EQU 007CH
      CNTR_CWR EQU 007EH
DATA ENDS

CODE SEGMENT
START:
MOV AX, DATA
MOV DS,AX
```

Intialize Data Segment register

```
MOV DX, PORT_CWR
MOV AL, 9AH
OUT DX, AL
```

Initialize 8255 in I/O mode with port A, and Cu as i/p port and Cl as o/p port

```
BACK:
MOV DX, PORTC
MOV AL, 0FFH
OUT DX, AL
```

Turn off the LED

```
CHECK:
IN AL, DX
TEST AL, 80H
JZ CHECK
```

Wait for button press

```
MOV AL, 00H
OUT DX, AL
```

Turn ON the LED

```
MOV DX, PORTB
IN AL, DX
MOV CL, AL
MOV DX, PORTA
IN AL, DX
MOV CH, AL
```

Read timer duration value from Port A and Port B of 8255 and store 16 bit count value (4 digit BCD count)

```
MOV DX, CNTR_CWR
MOV AL, 31H
OUT DX, AL
```

Initialize Counter 0 in Mode 0.

```
MOV DX, CNTR0
MOV AL, CL
OUT DX, AL
```

Load LSB of count

```
MOV AL, CH
OUT DX, AL
```

Load MSB of count

```
MOV DX, CNTR_CWR
MOV AL, 51H
OUT DX, AL
MOV AL, 0AH
MOV DX, CNTR1
OUT DX, AL
```

Initialize counter 1 in mode 0 and load count of 10D in counter.

```
MOV DX, PORTC
L1:
IN AL, DX
TEST AL, 40H
JZ  L1
```

Wait for buzzer to go high.

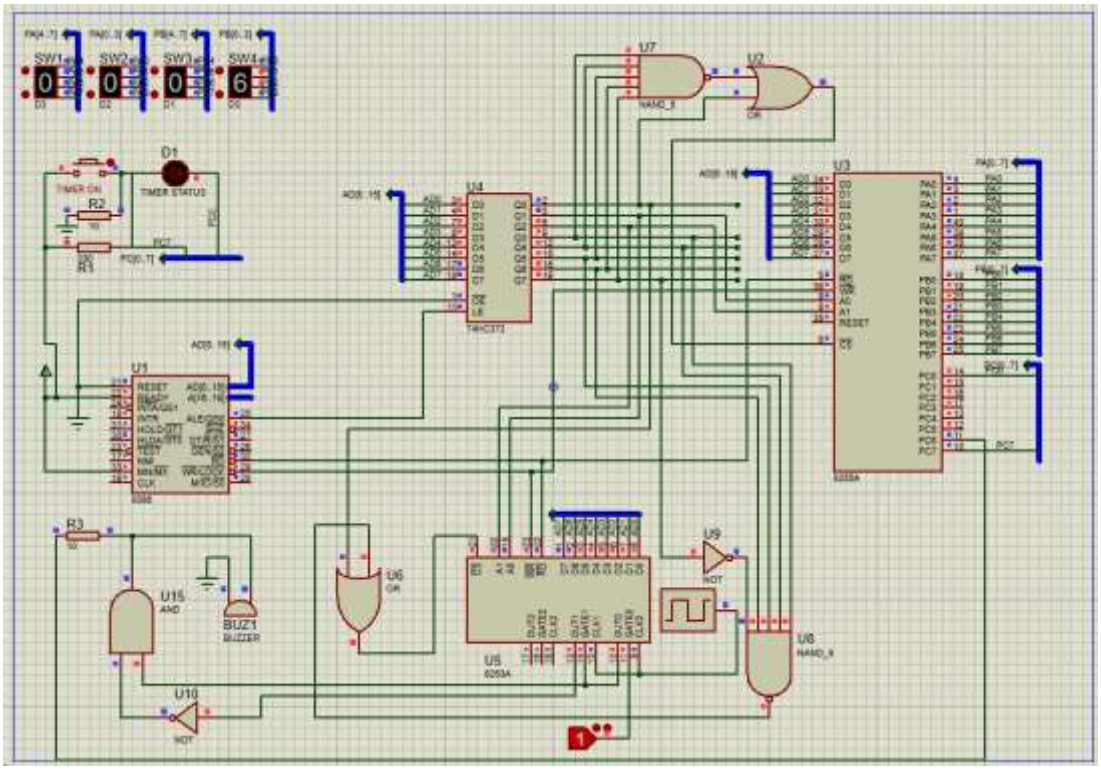
```
L2:
IN AL, DX
TEST AL, 40H
JNZ L2
JMP BACK
```

Now wait for buzzer to go low. When buzzer goes low, go back and wait for another key press to restart the timer.

```
CODE ENDS
END START
```

```
END
```

Diagram:



Algorithm:

1. 8255 is configured in I/O mode with port A, Port B and Port C upper initialized as input port and Port C lower initialized as output port all operated in mode 0 by loading appropriate control word.
2. A loop reads port C upper continuously and checks if the push button is pressed.
3. When the button press is detected, the program jumps out of the loop and reads Port A and Port B to get the count input by the user using the thumbwheel switch.
4. This count is loaded into counter 0 in BCD format initialized in mode 0. Counter 1 is initialized in mode 0 and count of 0AH (10 sec) is loaded into it.
5. Two loops to that read port C lower continuously to detect transition in buzzer state from low to high to low. When transition is detected, jump back to step 2.

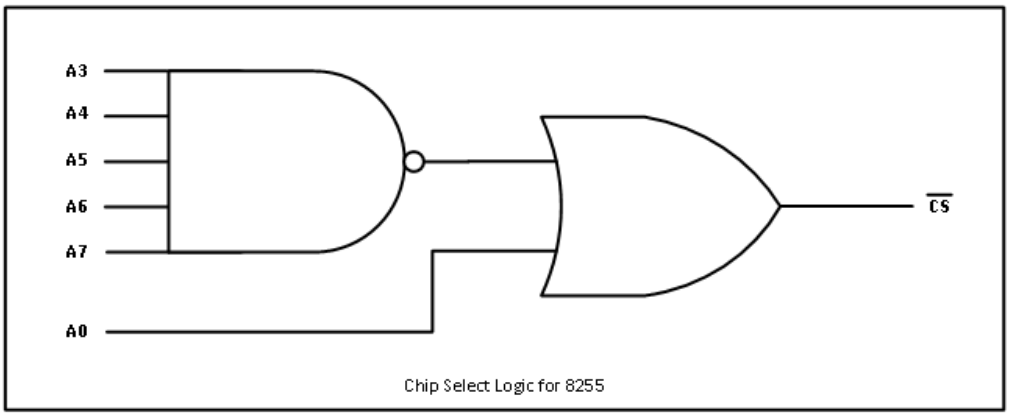
Design Description:

Hardware Description:

Proposed design of alarm consists of 8086 processor, 8055 Programmable Peripheral Interface, Thumbwheel buttons and 8254 Programmable Timer as shown in the figure.

8086 is set up in minimum mode. Demultiplexing of the address bus is carried out using 8 bit Latch 74HC373. Active low OE is grounded. Latch Enable LE is connected to ALE pin of 8086. Since only 8 bit addresses are used throughout the design, only one latch is being used to demultiplex Address lines A0 to A7.

8255 is interfaced with 8086 by partial decoding logic so that it has 8 bit address F8H for port A. Figure below shows address decoding logic for 8255.

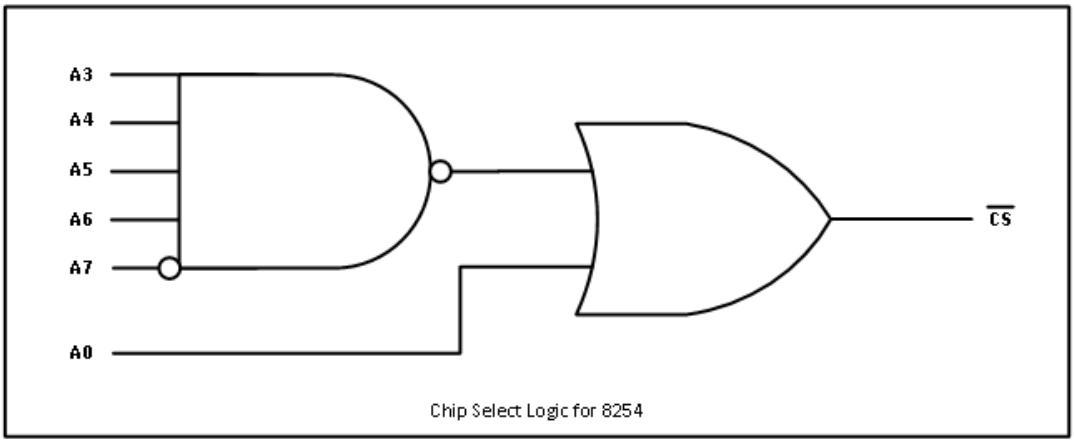


Chip Select Logic for 8255

A7	*A6*	*A5*	*A4*	*A3*	*A2*	*A1*	*A0*	*Address*	*PORT*
1	1	1	1	1	0	0	0	F8	PORT A
1	1	1	1	1	0	1	0	FA	PORT B
1	1	1	1	1	1	0	0	FC	PORTC
1	1	1	1	1	1	1	0	FE	CWR

Thumbwheel switches are interfaced with port A and port B of 8255 as shown in the system diagram. Each thumbwheel switch has 4 output lines which give a binary equivalent of the BCD number selected on the thumbwheel switch. 4 thumbwheel switches are interfaced so that 4 digit BCD count can be given as input by the user.

8254 is interfaced with 8086 by partial decoding logic so that it has 8 bit address 78H for Counter 0. Figure below shows address decoding logic for 8254.

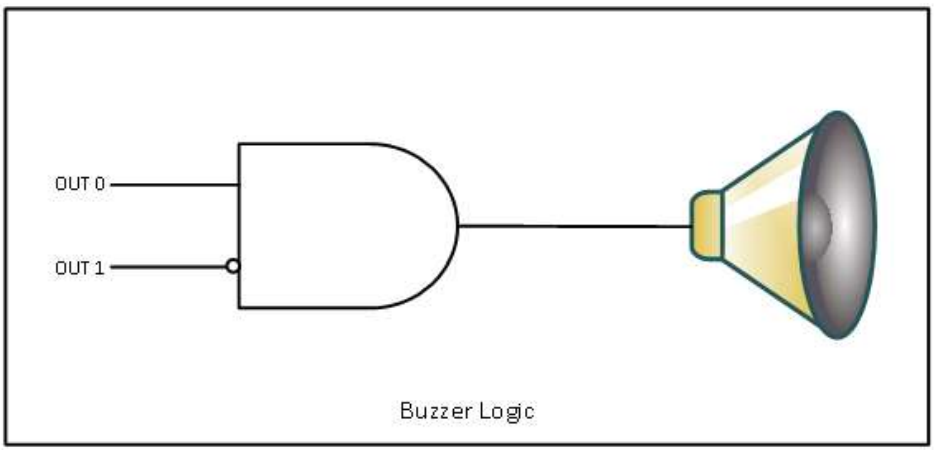


Chip Select Logic for 8254

A7	*A6*	*A5*	*A4*	*A3*	*A2*	*A1*	*A0*	*Address*	*Counter*
0	1	1	1	1	0	0	0	78	Counter 0
0	1	1	1	1	0	1	0	7A	Counter 1
0	1	1	1	1	1	0	0	7C	Counter 2
0	1	1	1	1	1	1	0	7E	CWR

LED and Push Button are interfaced with Port C of 8255 as shown in the system diagram. Port C upper is used as input port for Push button interfacing and Port C Lower is used as output port for LED interfacing.

Buzzer is interfaced with logic as shown in figure below.



Buzzer Logic

As per the logic, buzzer produces sound only when output of counter 0 is high and output of counter 1 is low. This arrangement is done so that the buzzer rings only for 10 seconds after the timer reaches terminal count.

Software Description:

In 8255, Port A, Port B and Port C upper are initialized as input ports and Port C lower is initialized as output port. All ports are being operated in mode 0. Control Word for initialization is illustrated below.

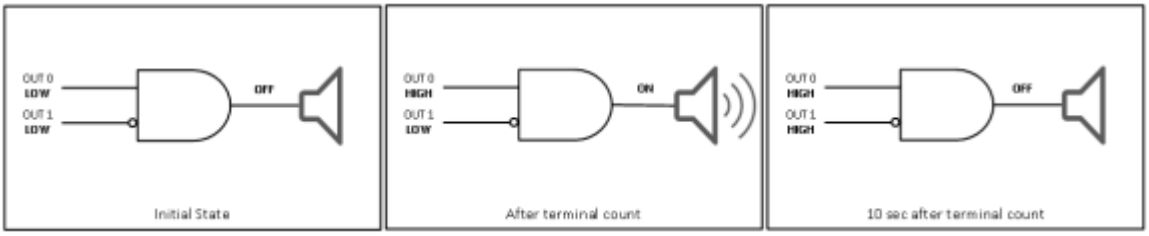
1	0	0	1	1	0	1	0	9AH

The Main Program continuously reads the input port C upper and checks if the push button is pressed. When button press is detected, port A and Port B are read to know the time duration input by the user. PA provides MSB (Most Significant Byte) and PB provides LSB (Least Significant Byte) in BCD format.

In 8254, counter 0 and counter 1 both are initialized in mode 0. 2 Byte count input by the user is loaded in counter 0. Gate of counter 0 is connected to logic High. Output of counter 0 is connected to Gate of counter 1. Counter 1 is loaded with count value 0AH (10 in decimal). Both counters are provided with a 1Hz pulse as input clock.

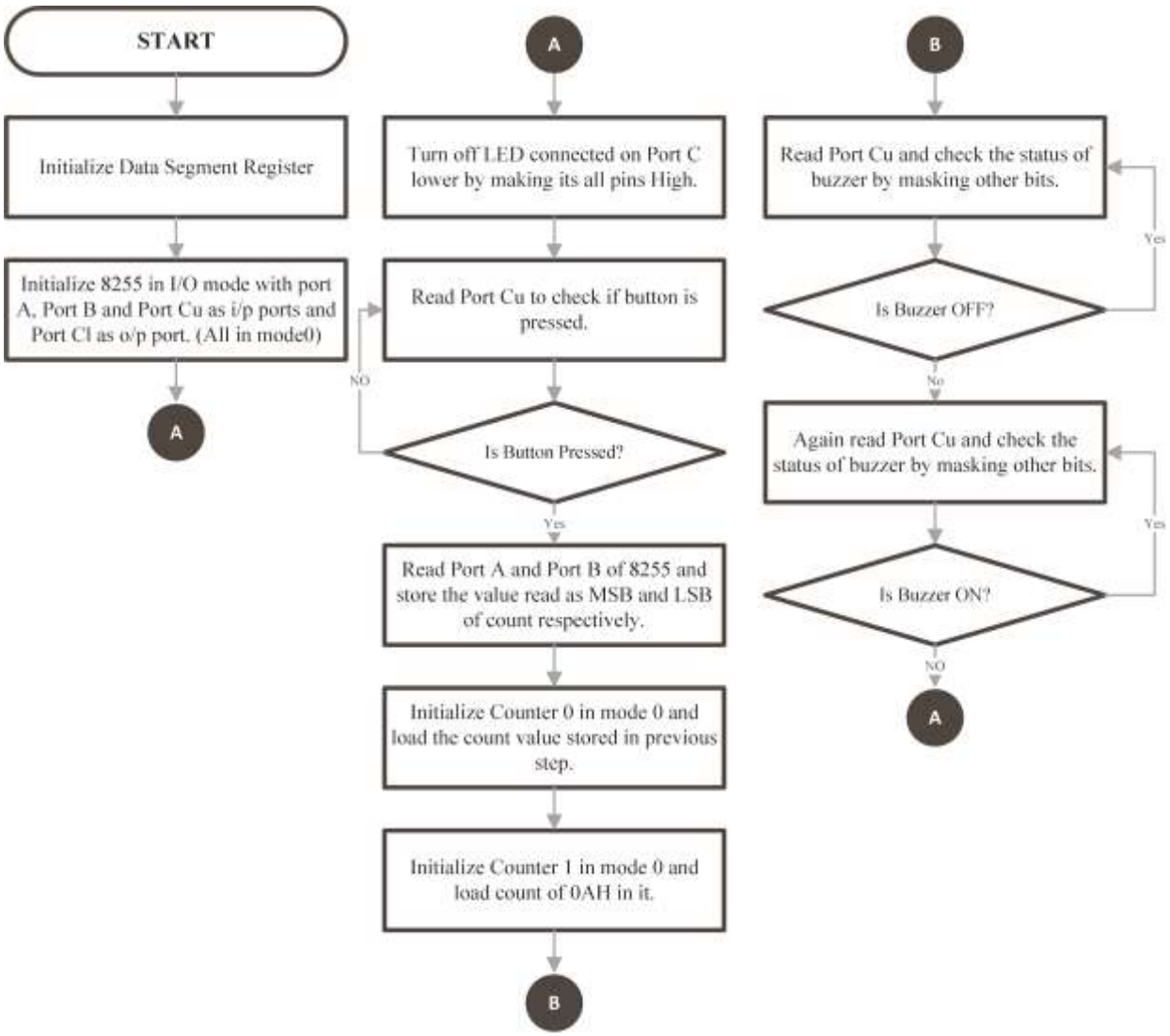
Initially when count is loaded in both the counters, Gate of counter 0 is high and Gate of counter 1 which is connected to output of counter 0 is low. So counter 0 starts counting but counter 1 has not yet started the count. After counter 0 reaches the terminal count, output of counter 0 goes high. This makes Gate of counter 1 high and hence counter 1 starts counting 10 seconds.

During this duration output of counter 0 is high and that of counter 1 is low. Using buzzer logic shown in the figure above, the buzzer produces sound. After 10 seconds when output of counter 1 also goes high, the buzzer stops producing sound. Figure below shows the working of the buzzer at different stages.



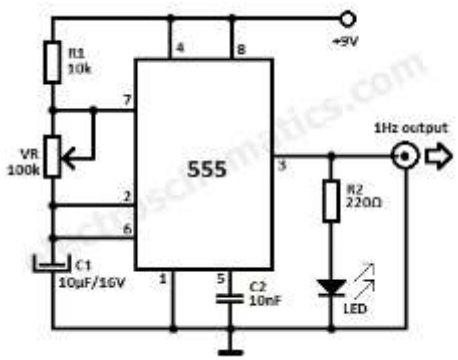
Program waits for transition in the buzzer state from low to high to low and when this transition is detected it returns back to the start of the program where it waits for the key press to start the timer.

Flow Chart:



Miscellaneous:

Proposed design uses 1 Hz clock as input to the counter. 1Hz clock can be designed using a 555 timer IC using circuit shown in the figure below.



References:

[1] “Advanced Microprocessors and Peripherals” by K M Bhurchandi

[2]” Microprocessors & Interfacing” by Douglas V Hall

[3]” www.electroschematics.com” for design of 1Hz clock using 555 timer.

[4] “8086 Microprocessor: Architecture & Interfacing” by Bharat Acharya