NAME: AMMAAR AHMAD ROLL No. - 1801CS08

1. ~~Vann~~ Von Neumann Architecture

It is based ~~on~~ on

(I) Uses of a single processor

(II) Uses one memory for both instructions and data. It cannot distinguish between data and instruction in a memory location. It 'knows' only because the location of a particular bit pattern in RAM

(III) Executes program by doing one instruction after the next in the serial manner using a fetch - decode - execute cycle.

It's CPU have 4 main parts

(I) The ALU, or Arithmetic Logic Unit ⇒ Used to perform Arithmetic and logical operations. Each CPU has its own word size. This is number of bits that can be operated on in one go. The bigger the CPU word's size, the more bits it can work on in one clock cycle and faster the work can be completed

(II) The Control Unit - This unit is in charge of 'fetching' each instruction that needs to be executed in a program by issuing control signals to the hardware. It then decodes the instruction and finally ~~decodes~~ issue ~~the~~ more control signals to the hardware to actually execute it

(III) Registers are fast memory circuits. They hold various information like address of next instruction (PC), the current instruction being executed (Current Instruction Register), the data being worked on the results of arithmetic and logical operators (Accumulator), information about the last operation information about the last operation (Status Register) and whether an ~~interrupt~~ interrupt has happened (~~Interrupt~~ Interrupt Register). Registers are covered ~~in a lot more detail~~ &

(IV) Clock: Instruction are carried out to the beat of a clock. Some ~~instruction~~ Instruction can be of more than one beat. Speed of clock beat determines the efficiency of computer

IAS – Immediate Access Store also known as RAM. – Random Access Memory
It can store a bit pattern. There is no difference between data and instruction stored in it. It is differentiated indirectly because of where the bit pattern is stored in RAM.
Operating System manages the use of memory to track RAM addresses as well as data
There are address buses in ~~data~~ computers which helps in storing and retrieving data from particular ~~to~~ location of memory. The data itself is moved between devices on a data bus. Control Bus manages the entire process

Input/Output
Computer needs peripherals from which it can read and send data. It's done through I/O port. Each port need to managed. An I/O controller is used to manage data in and out of I/O ports. Though I/O Ports can be directly connected to CPU, this is not done because on adding or removing a device, CPU has to be redesigned. Also all devices don't work on same voltage. In order to ~~avoid~~ avoid redesigning everytime I/O Controller acts a interface

Data Flow Architecture

In data flow architecture, information is pulled into a system which then flows through several modules and goes through transformation until destination is achieved. In data flow architecture, transformation can be used to reused and modified

Modules and Components
(i) Batch Sequential
(ii) Pipe and Filter
(iii) Process Control

Batch Sequential - The task is divided into several subtasks in batches. Batches perform subtasks and provide result to the next batch. Next batch starts only when previous batch is through

Pipe and Filter - This emphasise the incremental transformation of data to complete the task, it gives the possibility to process the data concurrently independent of others and later combined to draw useful output
It provides flexibility of decomposing the whole system into pipes, filters and data sinks. The pipes are interconnected and follow FIFO method to process the information. There is a flexibility to use both sequentially as well as parallel. It is different from batch sequential as there is concurrent processing

Process Control Architecture - Data is processed based on variables passed to it. The stream of data is processed by composing the whole system into several modules and is connected to process it further. In process control unit there are 2 main unit one is a processing unit and the other is the controller unit. The process unit does the job of changing the variable and the Controlling unit takes into account the changes that have been made

Conclusion :- Data flow Architecture depicts the workflow followed to create a software system. The workflow consist of a series of transformation on the input information, where information and operations are independent of each other.

Stack Machine Architecture

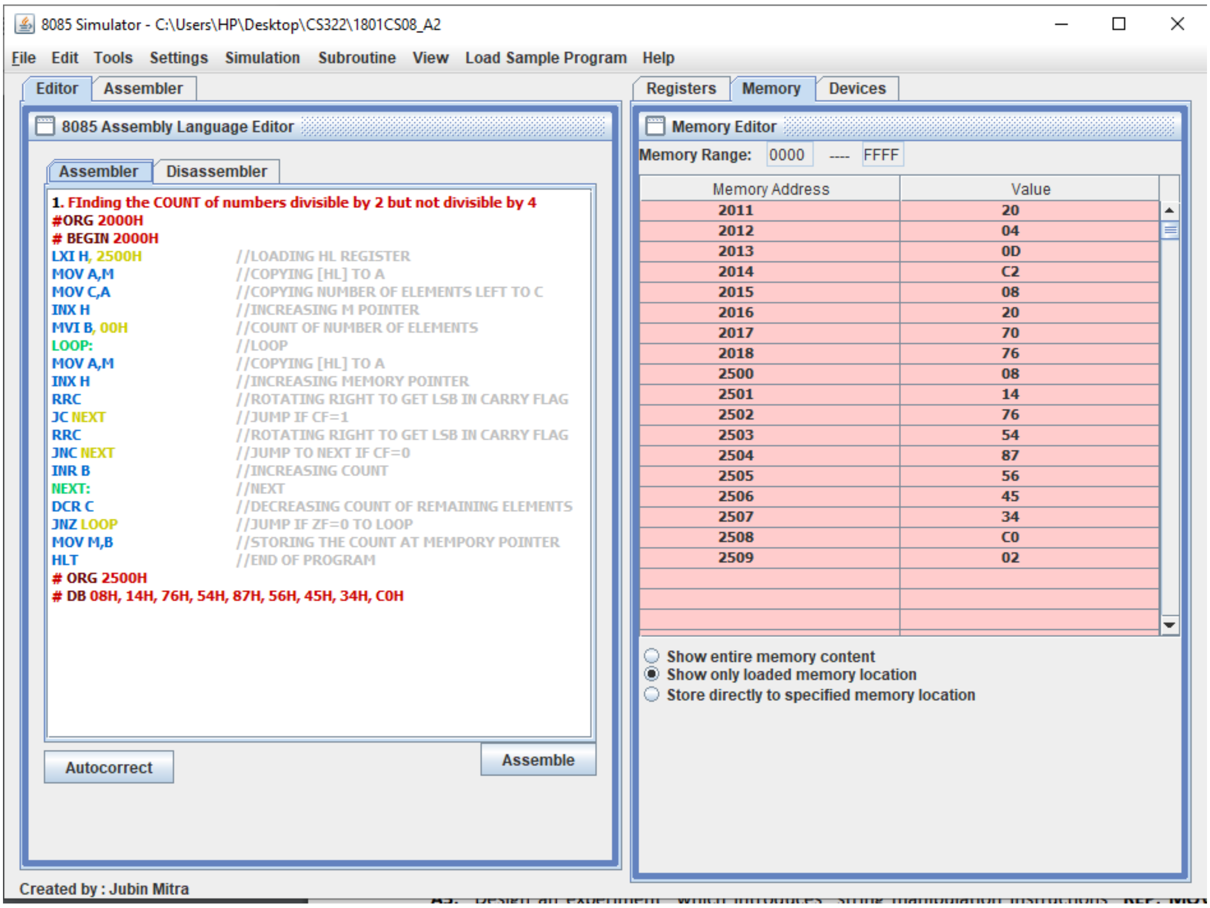
It is a computational model like Turing Machine, Belt Machine and many others. The central and most important component of a Stack machine is a Stack which is used in place of register to hold temporary variables. It support push and pop operation
It uses last in first out (LIFO) to hold short lived temporary values. Most of its instruction assume that operands will be from ~~the~~ stack and results placed in the stack.

For speed, a stack machine often implements some part of its stack with registers. To operate ~~quick~~ quickly, operands of the ALU may be the top 2 registers of the stack and result from ALU is stored in the top register of the stack. Some stack machine have a stack of limited size implemented as register file

Stack machine may have their expression stack and their call return stack separated as one integrated structure. If they are separated, the instructions of the stack machine can be pipelined with fewer interactions and less design complexity. Usually it can run faster.

2. FInding the COUNT of numbers divisible by 2 but not divisible by 4

```
#ORG 2000H

# BEGIN 2000H

LXI H, 2500H            //LOADING HL REGISTER

MOV A,M                 //COPYING [HL] TO A

MOV C,A                 //COPYING NUMBER OF ELEMENTS LEFT TO C

INX H           //INCREASING M POINTER

MVI B, 00H              //COUNT OF NUMBER OF ELEMENTS

LOOP:           //LOOP

MOV A,M                 //COPYING [HL] TO A

INX H           //INCREASING MEMORY POINTER

RRC             //ROTATING RIGHT TO GET LSB IN CARRY FLAG

JC NEXT         //JUMP IF CF=1

RRC             //ROTATING RIGHT TO GET LSB IN CARRY FLAG

JNC NEXT                //JUMP TO NEXT IF CF=0

INR B           //INCREASING COUNT

NEXT:           //NEXT

DCR C           //DECREASING COUNT OF REMAINING ELEMENTS

JNZ LOOP                //JUMP IF ZF=0 TO LOOP

MOV M,B                 //STORING THE COUNT AT MEMPORY POINTER

HLT             //END OF PROGRAM

# ORG 2500H

# DB 08H, 14H, 76H, 54H, 87H, 56H, 45H, 34H, C0H
```

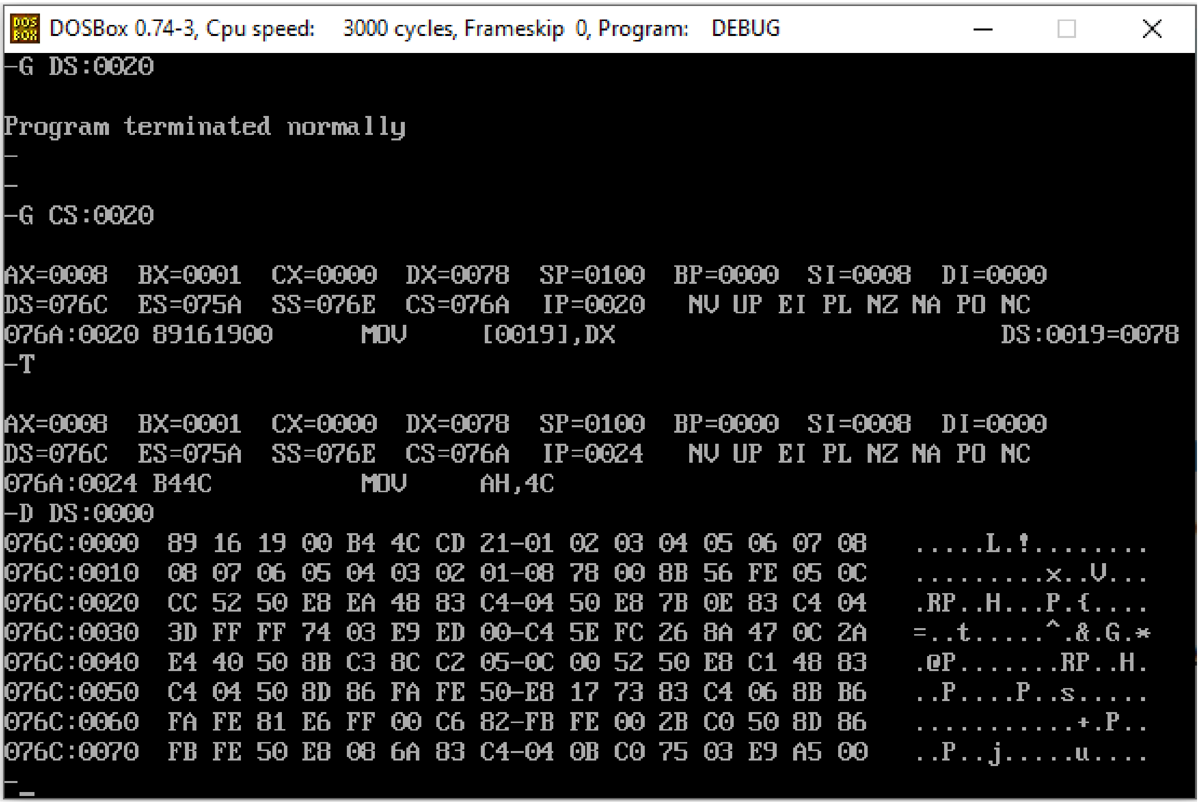


3. GIVEN 2 ARRAY OF 8 BIT NUMBERS FIND SUM OF ARR[I]*PRI[I]. PRIORITY IS UNIQUE AND IN BETWEEN 1 TO NUM INCLUSIVE

```
.MODEL SMALL

.STACK 100H

.DATA

        ARR DB 1,2,3,4,5,6,7,8   ;ARRAY ARR

        PRI DB 8,7,6,5,4,3,2,1   ;ARRAY PRI

        NUM DB 8                 ;NUMBER OF ELEMENTS

        SUM DW ?                 ;SUM OF ARR[I]*PRI[I]

.CODE

START:  MOV AX, @DATA                 ;AX = DATA SEGMENT ADDRESS

        MOV DS, AX               ;LOADING DS TO AX

        MOV SI, 00H              ;OFFSET
```

```
MOV DX, 00H             ;SUM=0
MOV CL, NUM             ;LOOP COUNT VARIABLE
MOV CH, 00H             ;COUNT VARIABLE EXTENSION
REPEAT:     MOV AL, ARR[SI]         ;LOADING NUMBER FROM ARR
MOV BL, PRI[SI]         ;LOADING NUMBER FROM PRI
MUL BL                  ;AL*BL
ADD DX, AX              ;ADDING RESULT TO SUM
INC SI                  ;INCREMENT OFFSET
LOOP REPEAT             ;CX=CX-1 IF CX!=0 LOOP AGAIN
MOV SUM, DX             ;STORING SUM
MOV AH, 4CH             ;RETURNING CONTROL TO OS
INT 21H
END START
.END
```

OUTPUT IN DS:0019H=78H=120



4. Types of addressing modes:

1) Register mode – In this type of addressing mode both the operands are registers.

Example- MOV AX, BX

2) Immediate mode – In this type of addressing mode the source operand is a 8 bit or 16 bit data. Destination operand can never be immediate data.

Example-MOV AX, 2000

3) Displacement or direct mode – In this type of addressing mode the effective address is directly given in the instruction as displacement.

Example-MOV AX, [0500]

4) Register indirect mode – In this addressing mode the effective address is in SI, DI or BX.

Example-MOV AX,[DI]

5) Based indexed mode – In this the effective address is sum of base register (BX,BP) and index register(SI,DI).

Example- MOV AX, [BX+DI]

6) Indexed mode – In this type of addressing mode the effective address is sum of index register and displacement.

Example- MOV AX,[SI+2000]

7) Based mode – In this the effective address is the sum of base register and displacement.

Example-MOV AL, [BP+ 0100]

8) Based indexed displacement or relative mode – In this type of addressing mode the effective address is the sum of index register, base register and displacement.

Example- MOV AL, [SI+BP+2000]

9) String mode – This addressing mode is related to string instructions. In this the value of SI and DI are auto incremented and decremented depending upon the value of directional flag.

Example- MOVSB, LODSW

10) Implied mode - In this addressing mode the operand is implied in the opcode

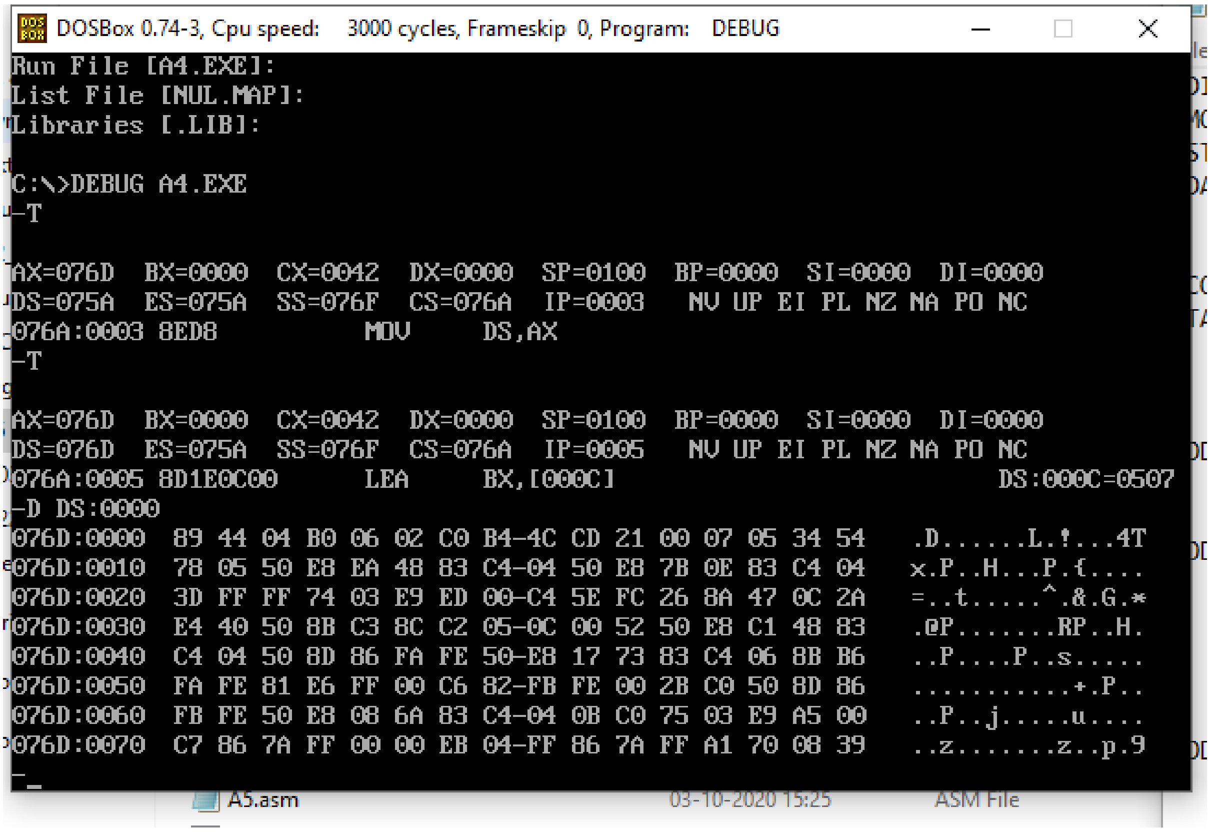
Example- STC, CTC, DAA

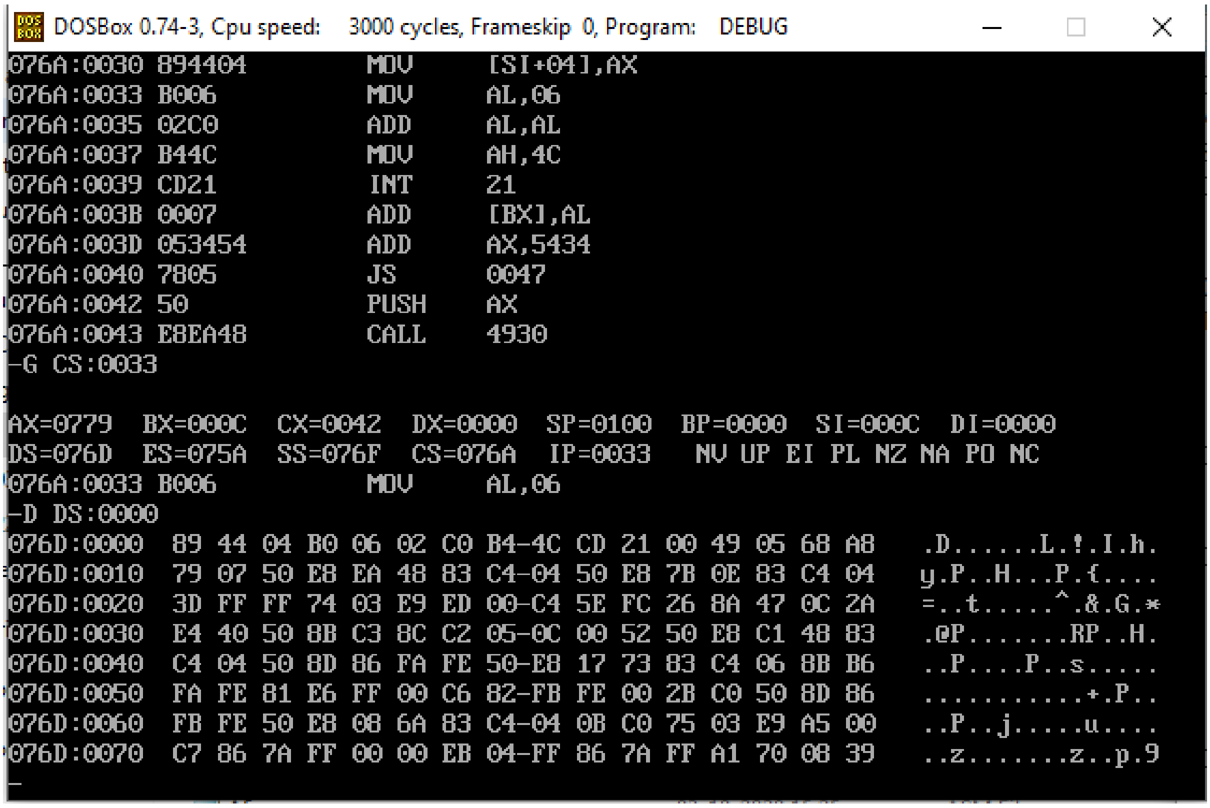
SOURCE CODE:

```
;DIFFERENT ADDRESSING MODES
.MODEL SMALL
.STACK 100H
.DATA
        ARR DB 07H, 05H, 34H, 54H, 78H
        LEN DB 05H
.CODE
START:  MOV AX, @DATA                  ;AX = DATA SEGMENT ADDRESS
        MOV DS, AX              ;LOADING DS TO AX
        LEA BX, ARR             ;LOADING BASE ADDRESS OF ARR
        MOV AL, [BX]            ;REGISTER INDIRECT ADDRESSING MODE
        ADD AL, CL              ;REGISTER ADDRESSING MODE
        MOV [BX], AL            ;REGISTER INDIRECT ADDRESSING MODE
        MOV AL, [BX+01H]        ;BASED ADDRESSING MODE
        MOV [BX+01H], AL        ;BASED ADDRESSING MODE

        MOV SI, 02H             ;IMMEDIATE ADDRESSING MODE
        MOV AL, [BX+SI]                ;BASED INDEXED ADDRESSING MODE
        ADD AL, AL              ;REGISTER ADDRESSING MODE
        MOV [BX+SI], AL                ;BASED INDEXED ADDRESSING MODE

        MOV AL, [BX+SI+01H]     ;BASED INDEXED RELATIVE ADDRESSING MODE
        ADD AL, AL              ;REGISTER ADDRESSING MODE
        MOV [BX+SI+01H], AL     ;BASED INDEXED RELATIVE ADDRESSING MODE

        LEA SI, ARR             ;LOADING ADDRESS OF ARR
```

```
        MOV AL, [SI+04H]        ;INDEXED ADDRESSING MODE

        STC                     ;IMPLIED ADDRESSING MODE

        ADC AL, 00H             ;IMMEDIATE ADDRESSING MODE

        MOV [SI+04H], AX        ;INDEXED ADDRESSING MODE


        MOV AL, [0006H]                 ;DIRECT ADDRESSING MODE

        ADD AL, AL              ;REGISTER ADDRESSING MODE


        MOV AH, 4CH

        INT 21H

        END START

.END
```

ELEMENTS OF ARR IS FROM DS:000C TO DS:0010



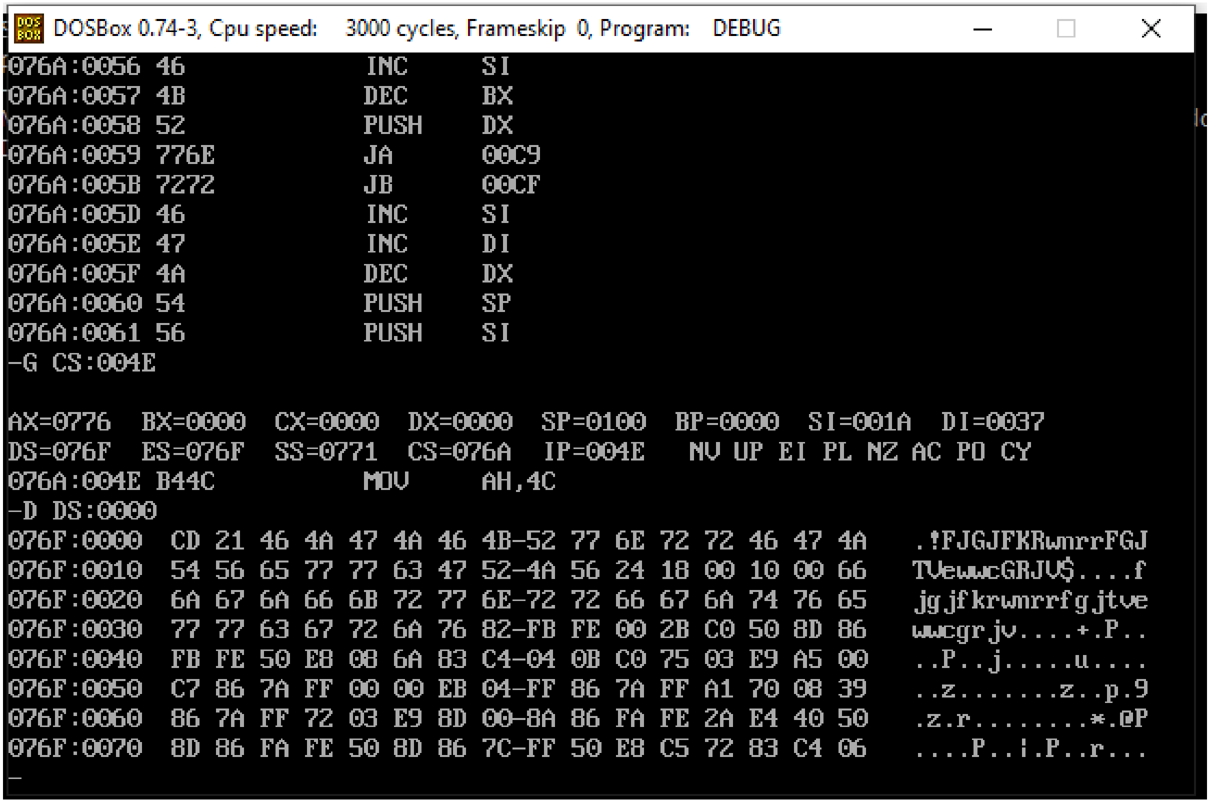


5. FINDING THE NUMBER OF UPPERCASE CHARACTERS IN A STRING

```
.MODEL SMALL

.STACK 100H

.DATA

        STRING1 DB 'FJGJFKRwnrrFGJTVewwcGRJV$'

        LEN     DW      0

        NUM     DW      0

        STRING2 DB      ?


.CODE

START:  MOV AX, @DATA                   ;AX = DATA SEGMENT ADDRESS

        MOV DS, AX              ;LOADING DS TO AX

        MOV ES, AX              ;LOADING ES TO AX

        LEA DI, STRING1                 ;LOADING STARTING ADDRESS OF STRING1 TO DI
```

```
        MOV AL, '$'         ;CHARACTER TO CHECK FOR END OF STRING
NEXT:   SCASB               ;COMPARING AL WITH [DI]
        JE DONE                     ;IF EQUAL THEN JUMP TO DONE
        INC LEN             ;INCREMENT LENGTH OF STRING
        JMP NEXT            ;JUMP BACK TO NEXT
DONE:   LEA SI, STRING1     ;SOURCE POINTER
        LEA DI, STRING2             ;DESTINATION POINTER
        MOV CX, LEN         ;COPYING LENGTH FOR LOOP
        REP MOVSB           ;COPYING ENTIRE BLOCK FROM SI TO DI
        MOV CX, LEN         ;COPYING LENGTH FOR LOOP
        LEA SI, STRING2     ;SOURCE POINTER
        LEA DI, STRING2             ;DESTINATION POINTER
REPEAT:         LODSB               ;AL=[SI]
        OR AL, 20H          ;CHANGING TO LOWERCASE
        STOSB               ;[DI]=AL
        LOOP REPEAT         ;REPEAT WHILE CX!=0
        LEA SI, STRING1     ;SOURCE POINTER
        LEA DI, STRING2             ;DESTINATION POINTER
        MOV CX, LEN         ;COPYING LENGTH FOR LOOP
FOR:    CMPSB               ;COMPARING [SI] AND [DI]
        JE SKIP             ;IF EQUAL JUMP TO SKIP
        INC NUM                     ;IF NOT EQUAL INCREMENT COUNT
SKIP:   LOOP FOR            ;CX=CX-1 LOOP WHILE CX!=0
        MOV AH, 4CH         ;RETURNING CONTROL TO OS
        INT 21H
        END START
.END
```

OUTPUT AT DS:001D = 10H = 16

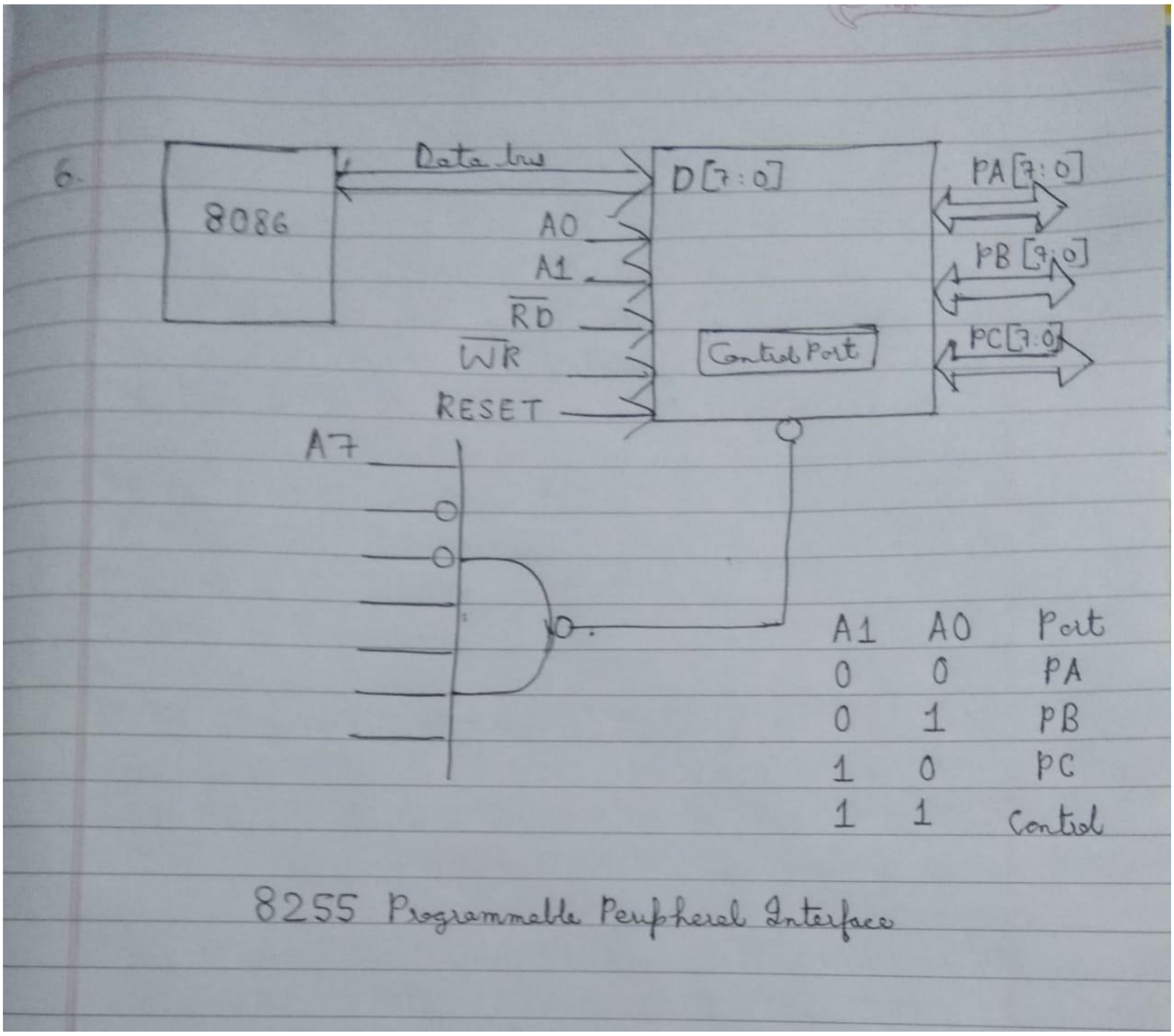


```
DOSBox 0.74-3, Cpu speed:    3000 cycles, Frameskip  0, Program:    DEBUG
076A:0056 46            INC     SI
076A:0057 4B            DEC     BX
076A:0058 52            PUSH    DX
076A:0059 776E          JA      00C9
076A:005B 7272          JB      00CF
076A:005D 46            INC     SI
076A:005E 47            INC     DI
076A:005F 4A            DEC     DX
076A:0060 54            PUSH    SP
076A:0061 56            PUSH    SI
-G CS:004E

AX=0776  BX=0000  CX=0000  DX=0000  SP=0100  BP=0000  SI=001A  DI=0037
DS=076F  ES=076F  SS=0771  CS=076A  IP=004E   NV UP EI PL NZ AC PO CY
076A:004E B44C          MOV     AH,4C
-D DS:0000
076F:0000  CD 21 46 4A 47 4A 46 4B-52 77 6E 72 72 46 47 4A   .!FJGJFKRwnrrFGJ
076F:0010  54 56 65 77 77 63 47 52-4A 56 24 18 00 10 00 66   TVewwcGRJV$....f
076F:0020  6A 67 6A 66 6B 72 77 6E-72 72 66 67 6A 74 76 65   jgjfkrwnrrfgjtve
076F:0030  77 77 63 67 72 6A 76 82-FB FE 00 2B C0 50 8D 86   wwcgrjv....+.P..
076F:0040  FB FE 50 E8 08 6A 83 C4-04 0B C0 75 03 E9 A5 00   ..P..j.....u....
076F:0050  C7 86 7A FF 00 00 EB 04-FF 86 7A FF A1 70 08 39   ..z.......z..p.9
076F:0060  86 7A FF 72 03 E9 8D 00-8A 86 FA FE 2A E4 40 50   .z.r........*.@P
076F:0070  8D 86 FA FE 50 8D 86 7C-FF 50 E8 C5 72 83 C4 06   ....P..|.P..r...
-
```

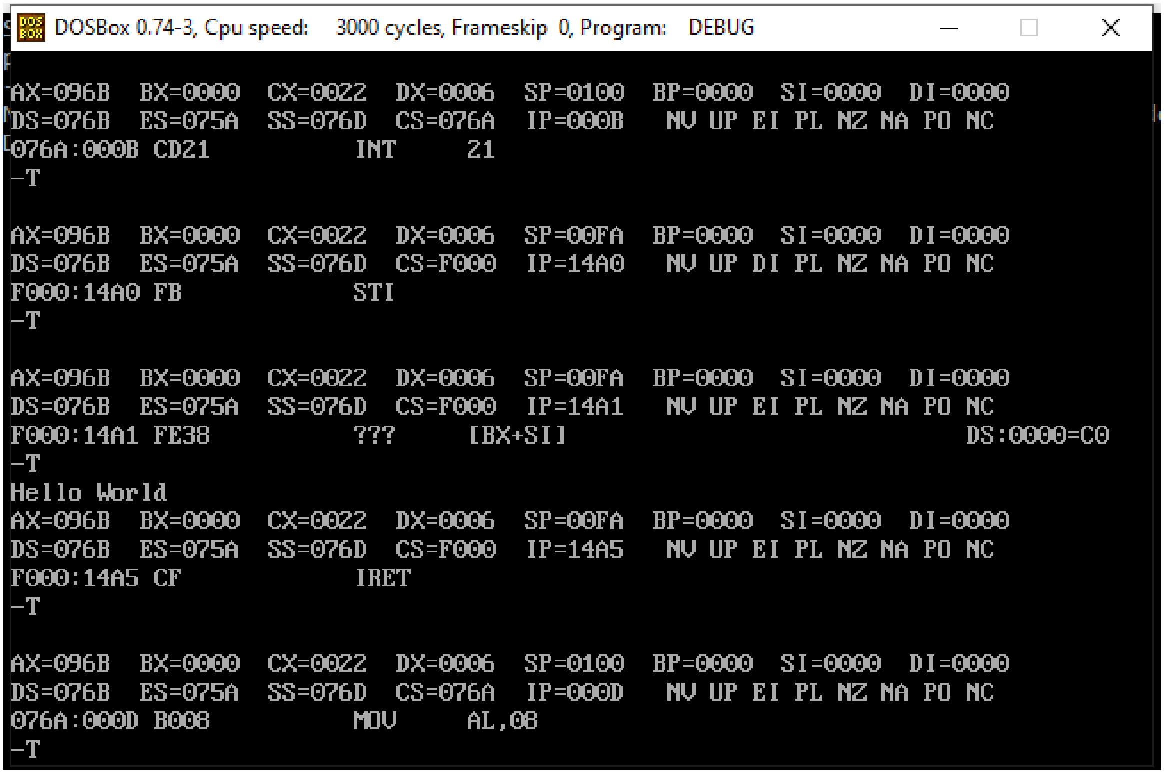
6. TEMPERATURE CONTROL USING PORT A AND PORT B OF 8255 INTERFACE

```
.MODEL SMALL
.STACK 64H
.CODE
START:  IN AL, 80H      ;TAKING INPUT FROM PORT A
        MOV BL,64H      ;BL=64H
        CMP AL, BL      ;COMPARING AL AND BL
        JL START        ;IF AL<BL JUMP TO START
        OUT 82H
        MOV AH,4CH
        INT 21H
END START
.END
```

INTERFACE DIAGRAM



7. INTERRUPT STEPS SCREENSHOT



INTERRUPT COMMAND TAKES MORE THAN ONE DEBUG CYCLE TO COMPLETE.

IN THE PROCESS STACK POINTER, INSTRUCTION POINTER, CODE SEGMENT, STACK SEGMENT ALL VALUES GET CHANGED AS CAN BE SEEN FROM THE FIGURE. IN THIS INT 21 COMMAND FOUR DEBUG CYCLE WAS CONSUMED BEFORE RETURNING TO NEXT LINE AFTER INT 21H COMMAND.

CURRENT CODE SEGMENT AND INSTRUCTION POINTER IS PUSHED INTO THE STACK IN ORDER TO RETURN BACK AFTER EXECUTION OF INT 21H