CS321 Mid Semester

Name: P. V. Sriram

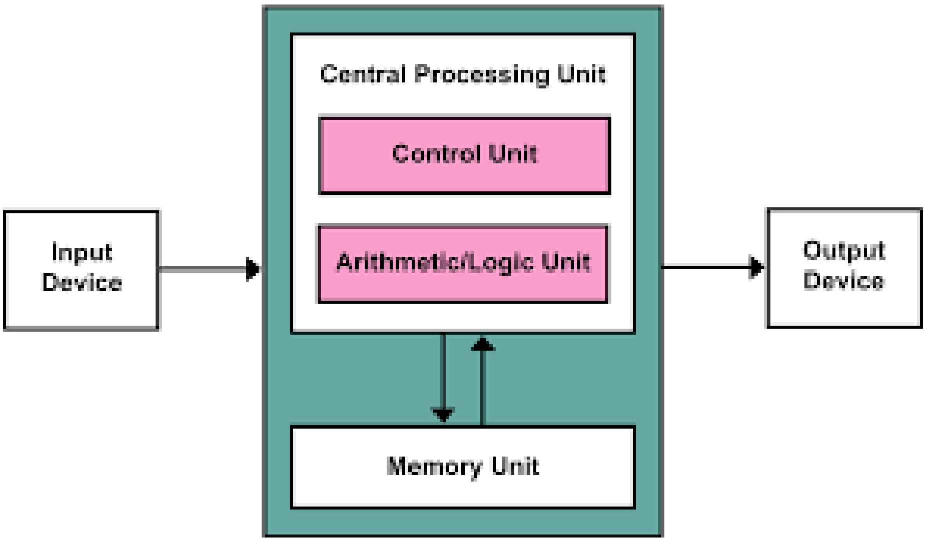
Roll No.: 1801CS37

A1)

Computer architecture is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. Computer architecture involves instruction set architecture design, microarchitecture design, logic design, and implementation and is concerned with balancing the performance, efficiency, cost, and reliability of a computer system

Von Neuman Architecture

The von Neumann architecture also known as the Stored program computer or Princeton architecture is a computer architecture based on a 1945 description by John von Neumann.



The design architecture here consists of:

- Central Processing Unit (CPU)
- Immediate Access Store (IAS)
- Input / Output (I/O)

Central Processing Unit

The CPU governs the computer and handles the data. It has four main parts as follows:

- **Arithmetic Logic Unit (ALU)**
  - This part handles all the arithmetic and logic operations such as calculations and comparisons.
- **Control Unit**
  - This part handles the movement of instructions to and from the memory and the execution of instructions one at a time.
- **Registers**
  - There are different types of registers depending on the information being stored.
    - Program Counter – stores location of next instruction
    - Current Instruction Register – stores current instruction being implemented
    - Accumulators – store results of calculations and comparisons
    - Status Register – stores data about the last operation
    - Interrupt Register – stores information regarding an interruption that occurred
- **Clock**
  - Instructions are executed to the beat of the clock. The faster the clock, the faster the computer is.

Immediate Access Store
This is commonly referred to as the Random-Access Memory (RAM) or Main Memory. It stores both instructions and data. Buses which allow the movement of instructions and data between different parts of the computer is called a data bus. Buses that detect locations in memory is called an address bus.

Input / Output

I/O refers to the accessories for inputting and outputting of data. A computer needs to read in data and send out data through I/O ports. An I/O controller is an interface that allows a user to attach any I/O device to the computer and send data in or out of the computer.

Dataflow Architecture

In data flow architecture, the whole software system is seen as a series of transformations on consecutive pieces or set of input data, where data and operations are independent of each other. In this approach, the data enters into the system and then flows through the modules one at a time until they are assigned to some final destination (output or a data store).

There are three types of execution sequences between modules–

- Batch sequential
- Pipe and filter or non-sequential pipeline mode
- Process control

Batch Sequential

In Batch sequential, separate programs are executed in order and the data is passed as an aggregate from one program to the next.

It provides simpler divisions on subsystems and each subsystem can be an independent program working on input data and produces output data.

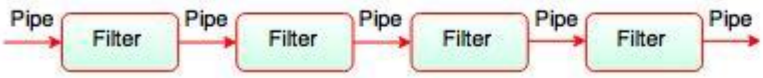


Fig. Batch Sequential

Pipe and Filter

Pipe is a connector which passes the data from one filter to the next. They are a directional stream of data implemented by a data buffer to store all data, until the next filter has time to process it.

Filter reads the data from its input pipes and performs its function on this data and places the result on all output pipes. If there is insufficient data in the input pipes, the filter simply waits. All filters are the processes that run at the same time, it means that they can run as different threads, coroutines or be located on different machines entirely.

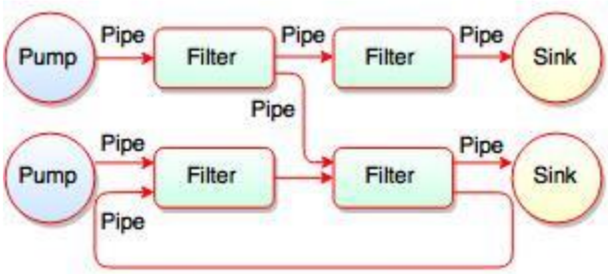


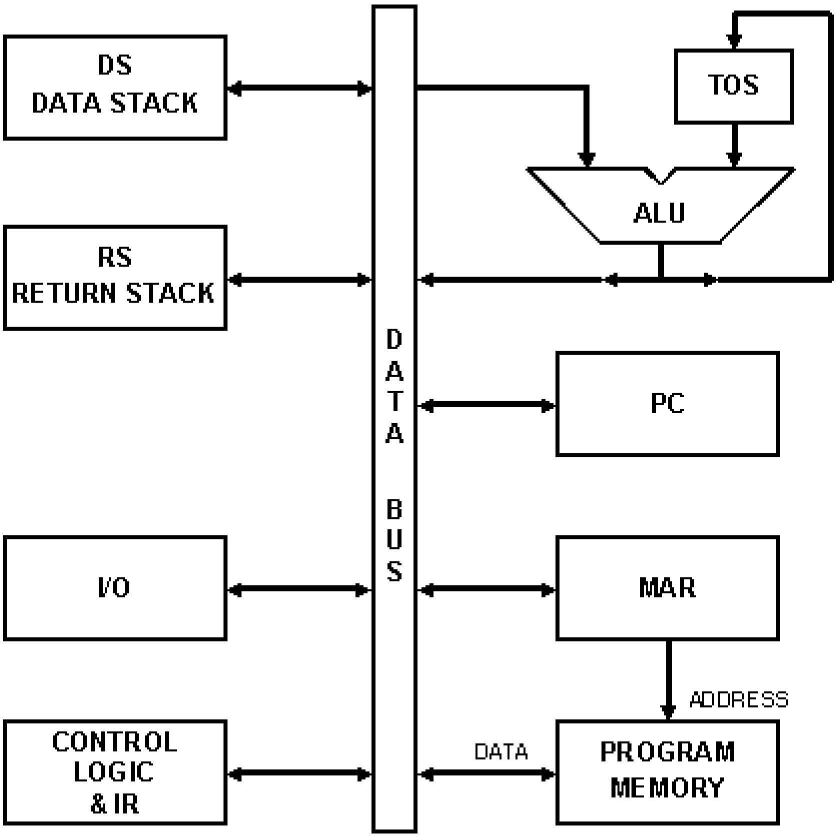
Fig. Pipes and Filters

Process Control

Process Control Architecture is a type of Data Flow Architecture, where data is neither batch sequential nor pipe stream. In process control architecture, the flow of data comes from a set of variables which controls the execution of process.

Stack Machine

Stack machine is a mode of computation where executive control is maintained wholly through append (push), readoff and truncation (pop), of a first-in-last-out (FILO, also last-in-first-out or LIFO) memory buffer, known as a stack, requiring very few processor registers.



In the stack machine, data is available at the top of the stack by default. The stack acts as a source and destination, push and pop instructions are used to access instructions and data from the stack.

In a stack machine, the operands used in the instructions are always at a known offset (set in the stack pointer), from a fixed location (the bottom of the stack, which in a hardware design might always be at memory location zero), saving precious in-cache or in-CPU storage from being used to store quite so many memory addresses or index numbers.

A2)

Leap Year or not

I have formulated an 8085-assembly language program to identify is an input year is a leap year or not. The rule is the year should either be divisible with 400 or 4.

The inputs are given into 2050(lower byte) and 2051(higher byte). Firstly, we check if lower byte is 0. If yes, we should check if the higher byte is divisible by 4. Together, these two rules verify the divisibility with 400. If the lower byte is not 0 we should check if it is divisible with 4 which verifier divisibility with 4.

Code

```
# ORG 2000

        LXI H,2050

        MOV A,M     // B<-M

        CPI 00

        JNZ LB

        MVI C,00    // C<-00H

        INX H

        MOV A,M     // A<-M


ITER:   CPI 04

        JC STORE    // check for carry

        SUI 04      // A<-A-B

        INR C // C<-C+1

        JMP ITER


STORE:  STA 3050    // 3050<-A

        MOV A,C     // A<-C

        STA 3051    // 3051<-A

        HLT   // terminate the program


LB:     CPI 04

        JC STORE    // check for carry

        SUI 04      // A<-A-B
```

INR C // C<-C+1

JMP LB

ORG 2050

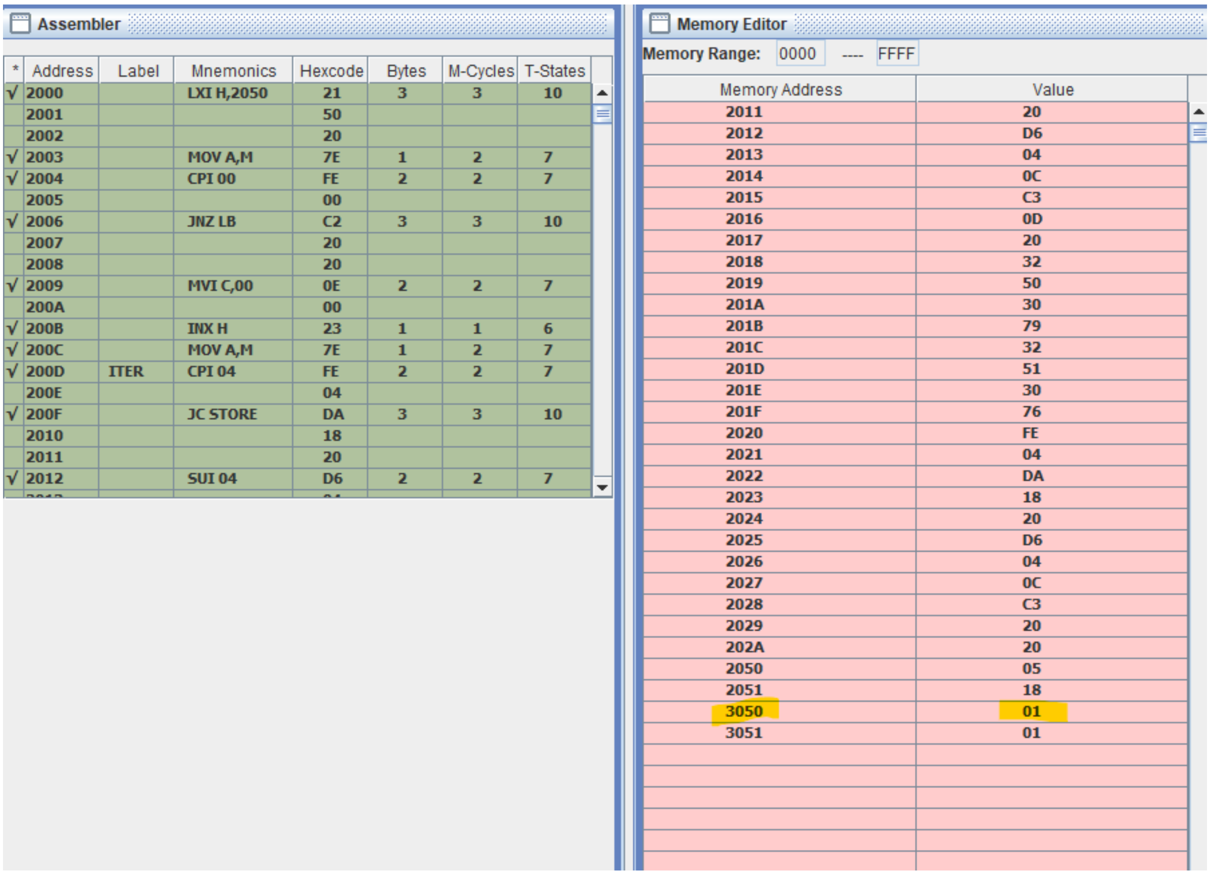
DB 05, 24D

Input

2405 D (Address pair - 2051H, 2050H)

Output

1 (3050H) (0 at 3050H indicates leap year, A non-zero number at 3050H indicates a non-leap year)



Assembler

*	Address	Label	Mnemonics	Hexcode	Bytes	M-Cycles	T-States
√	2000		LXI H,2050	21	3	3	10
	2001			50			
	2002			20			
√	2003		MOV A,M	7E	1	2	7
√	2004		CPI 00	FE	2	2	7
	2005			00			
√	2006		JNZ LB	C2	3	3	10
	2007			20			
	2008			20			
√	2009		MVI C,00	0E	2	2	7
	200A			00			
√	200B		INX H	23	1	1	6
√	200C		MOV A,M	7E	1	2	7
√	200D	ITER	CPI 04	FE	2	2	7
	200E			04			
√	200F		JC STORE	DA	3	3	10
	2010			18			
	2011			20			
√	2012		SUI 04	D6	2	2	7

Memory Editor

Memory Range: 0000 ---- FFFF

Memory Address	Value
2011	20
2012	D6
2013	04
2014	0C
2015	C3
2016	0D
2017	20
2018	32
2019	50
201A	30
201B	79
201C	32
201D	51
201E	30
201F	76
2020	FE
2021	04
2022	DA
2023	18
2024	20
2025	D6
2026	04
2027	0C
2028	C3
2029	20
202A	20
2050	05
2051	18
3050	01
3051	01

A3)

Date Difference function

I have devised an 8086-assembly language to find the number of days between two dates from the same year. The algorithm is to first calculate the number of days between the two dates and a reference date (1^{st} Jan) and subtract the resultant numbers.

Code

```
.model small

.stack 64

.data

date1 dw 31D, 01D

date2 dw 28D, 04D

.code

	date:

		MOV AX, @data		;Load Data in temp register

		MOV DS, AX		;Load address into data segment

		LEA SI, date1	;Load first date pointer

		LEA DI, date2	;Load second date pointer

		MOV AX, [SI]	;Load first date

		MOV BX, [DI]	;Load second date

		SUB BX, AX		;BX <- BX - AX

		MOV DX, BX		;DX <- BX

	month:

		INC SI		;Increment pointer

		INC SI			;Increment pointer

		MOV AX, [SI]	;Load first month

		MOV CX, AX		;CX <- AX

		DEC CX				;Decrement CX

		JZ next_month		;If zero, Jump to next_month

		CMP AX, 03D	;Compare first month to 3

		JC bef_feb_1	;If carry, Jump to bef_feb_1

		SUB CX, 02		;CX <- CX - 2
```

```
        SUB DX, 59D    ;DX <- DX - 59
bef_feb_1:
        SUB DX, 31D    ;DX <- DX - 31
        MOV AX, CX                ;AX <- CX
        MOV BL, 02D   ;BL <- 02D
        DIV BL                 ;AL <- AH / BL remainder
        CMP AH, 00                ;Compare AH with 00
        JNZ next_month            ;If not zero jump to next month
        INC DX                                ;Increment DX by 1
        LOOP bef_feb_1            ;Loop until CX = 0
next_month:
        INC DI              ;Increment pointer
        INC DI                          ;Increment pointer
        MOV AX, [DI]    ;Load second month
        MOV CX, AX      ;CX <- AX
        DEC CX          ;Decrement pointer
        JZ exit         ;If zero, jump to exit
        CMP AX, 03D     ;Compare second month to 3
        JC bef_feb_2    ;Jump if carry to bef_feb_2
        SUB CX, 02      ;CX <- CX - 2
        ADD DX, 59D   ;DX <- DX + 59
bef_feb_2:
        ADD DX, 31D     ;DX <- DX + 31
        MOV AX, CX        ;AX <- CX
        MOV BL, 02D   ;BL <- 02
        DIV BL          ;AL <- AH / BL remainder
        CMP AH, 00      ;Compare AH with 00
        JNZ exit        ;If not zero jump to next month
        DEC DX          ;Increment DX by 1
```

```
        LOOP bef_feb_2  ;Loop until CX = 0
exit:
        HLT         ;Terminate
        end date
.end
```

Input

Date 1: 31D, 01D – 31st Jan

Date 2: 28D, 04D – 28th April

Output

57H – 87 Days (DX Register)

```
AX=0100  BX=FF02  CX=0001  DX=0057  SP=0040  BP=0000  SI=0002  DI=0006
DS=0770  ES=075A  SS=0771  CS=076A  IP=005F   NV UP EI PL NZ NA PO NC
076A:005F F4            HLT
-t
```

A4)

Addressing Modes

Register Mode

Register mode is where the source of operands are registers themselves. We specify the name of the register which holds the data to be operated in the instruction

E.g. MOV BX, AX To move content of AX register into BX

```
AX=0005  BX=0000  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=000C   NV UP EI PL NZ NA PO NC
076A:000C 8BD8          MOV     BX,AX
-t

AX=0005  BX=0005  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=000E   NV UP EI PL NZ NA PO NC
076A:000E B80C03        MOV     AX,030C
```

Immediate Mode

Immediate mode is where the source operand is an 8 / 16-bit number and the destination operand are registers.

E.g. MOV AX, 05H To move value 05H into AX register

```
AX=076C  BX=0000  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=0009   NV UP EI PL NZ NA PO NC
076A:0009 B80500        MOV     AX,0005
-t

AX=0005  BX=0000  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=000C   NV UP EI PL NZ NA PO NC
076A:000C 8BD8          MOV     BX,AX
```

Direct Mode

Direct addressing mode is where the effective address of the memory location (16-bit number) at which data operand is stored is directly given in the instruction.

E.g. MOV AX, ds:[0300H] To move the content from effective address 0300H to AX

```
AX=0005  BX=0005  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=000E   NV UP EI PL NZ NA PO NC
076A:000E A10003        MOV     AX,[0300]                        DS:0300=0001
-t

AX=0001  BX=0005  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=0011   NV UP EI PL NZ NA PO NC
076A:0011 BB0C03        MOV     BX,030C
```

Indirect Mode

Indirect addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI and SI. We use these registers to hold the effective address to be specified in the instruction

E.g. MOV AX, ds:[BX] To move the contents from Address (EA) stored in BX to AX

```
AX=0001  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=0014   NV UP EI PL NZ NA PO NC
076A:0014 8B07          MOV     AX,[BX]                          DS:0300=0001
-t

AX=0001  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=0016   NV UP EI PL NZ NA PO NC
076A:0016 8B4704        MOV     AX,[BX+04]                       DS:0304=0003
```

Based Mode

Based index mode is where the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement. When BX holds the base value of EA, 20-bit physical address is calculated from BX, DS and if BP holds the base value, BP and SS are used.

E.g. MOV AX, ds:[BX + 04H] Here, Effective address is BX + 04H

```
AX=0001  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=0016   NV UP EI PL NZ NA PO NC
076A:0016 8B4704        MOV     AX,[BX+04]                      DS:0304=0003
-t

AX=0003  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=0019   NV UP EI PL NZ NA PO NC
076A:0019 8B4404        MOV     AX,[SI+04]                      DS:0304=0003
```

Indexed Mode

Indexed mode is where the offset address of the operand is given by the sum of contents of SI / DI registers and b-bit/16-bit displacements. SI / DI register is used to hold an index value for memory data and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

E.g. MOV AX, ds:[SI + 04H] Here EA is SI + 04H

```
AX=0003  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=0019   NV UP EI PL NZ NA PO NC
076A:0019 8B4404        MOV     AX,[SI+04]                      DS:0304=0003
-t

AX=0003  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=001C   NV UP EI PL NZ NA PO NC
076A:001C 8B4001        MOV     AX,[BX+SI+01]                   DS:0601=8A26
```

Based-Index Mode

Based Index mode is where the offset address of the operand is given by the sum of contents of BX / BP (base register) and SI / DI (index register). SI / DI register is used to hold an index value for memory data and BX / BP store the base value of EA.

```
AX=8A26  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=001F   NV UP EI PL NZ NA PO NC
076A:001F 0000          ADD     [BX+SI],AL                      DS:0600=F8
-t

AX=8A26  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=0021   NV UP EI PL NZ NA PE CY
076A:0021 0000          ADD     [BX+SI],AL                      DS:0600=1E
```

Based Index Displacement Mode

Based Index Displacement mode is where the offset address of the operand is given by the sum of contents of BX / BP (base register) and SI / DI (index register) and an 8-bit / 16-bit displacement. SI / DI register is used to hold an index value for memory data and BX / BP store the base value of EA.

E.g. MOV AX, ds:[BX + SI + 01H]

```
AX=0003  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=001C   NV UP EI PL NZ NA PO NC
076A:001C 8B4001        MOV     AX,[BX+SI+01]                      DS:0601=8A26
-t

AX=8A26  BX=0300  CX=032A  DX=0000  SP=0040  BP=0000  SI=0300  DI=0000
DS=076C  ES=075A  SS=079D  CS=076A  IP=001F   NV UP EI PL NZ NA PO NC
076A:001F 0000          ADD     [BX+SI],AL                         DS:0600=F8
```

String Mode

String mode is used to operate on string data. In this the value of SI and DI are auto incremented and decremented depending upon the value of directional flag. The effective address of the source data is stored in SI register and the EA of destination is stored in DI register; Base address is the DS.

E.g. REP MOVSB

```
AX=0000  BX=0000  CX=0005  DX=0000  SP=0040  BP=0000  SI=0300  DI=0600
DS=076E  ES=0000  SS=079F  CS=076A  IP=0018   NV UP EI PL NZ NA PO NC
076A:0018 F3            REPZ
076A:0019 A4            MOVSB
-t

AX=0000  BX=0000  CX=0004  DX=0000  SP=0040  BP=0000  SI=0301  DI=0601
DS=076E  ES=0000  SS=079F  CS=076A  IP=0018   NV UP EI PL NZ NA PO NC
076A:0018 F3            REPZ
076A:0019 A4            MOVSB
```

Input/Output Mode

Input/Output Mode is used to access data from the standard I/O mapped devices or ports. In Direct I/O addressing, 8-bit address is used to specify the periphery location. In Indirect I/O addressing, we use the DX register to store the I/O address.

E.g. IN AL, 80H

```
AX=FFFF  BX=0064  CX=000B  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=076B  CS=076A  IP=0000   NV UP EI NG NZ NA PO NC
076A:0000 E480          IN      AL,80
```

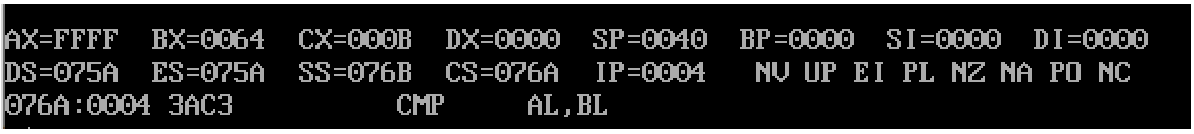
Relative Mode

Relative addressing mode is where the EA of a program instruction is specified relative to Instruction pointer (IP) by an 8-bit signed displacement.

Implied Mode

Implied mode is where data operand is a part of the instruction itself. There are no operands here as the instructions themselves specify the data to be operated.

E.g. CMP AL, BL



```
AX=FFFF  BX=0064  CX=000B  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=076B  CS=076A  IP=0004   NV UP EI PL NZ NA PO NC
076A:0004 3AC3          CMP     AL,BL
```

Code

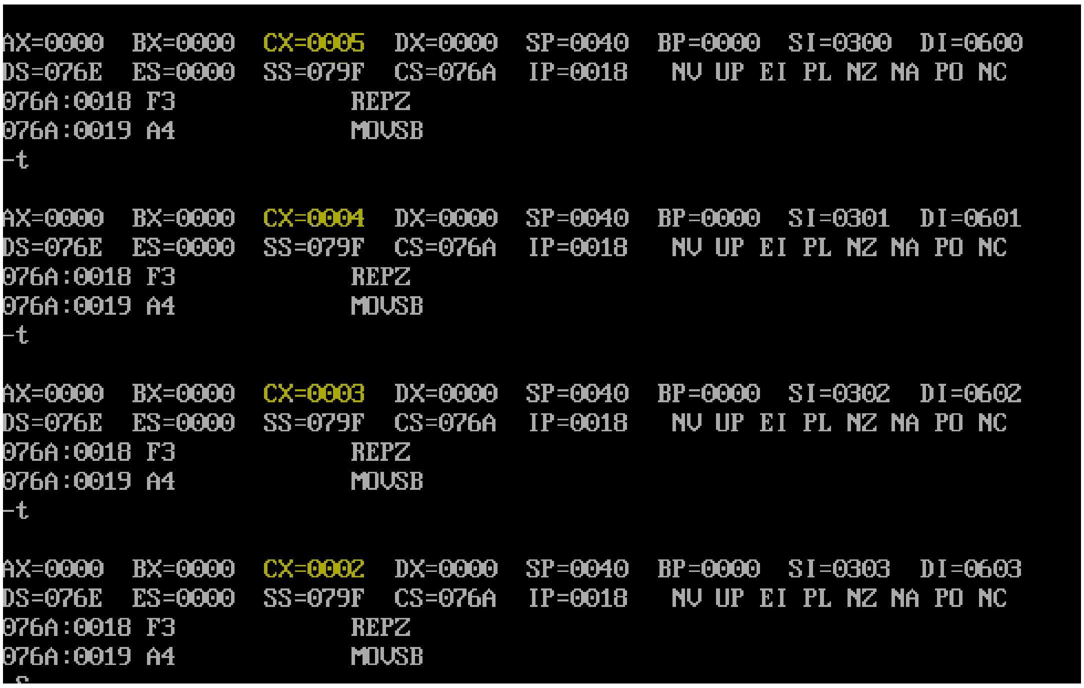
```
.model small

.stack 64

.data

org 300H

arr dw 0001, 0002, 0003, 0004, 0005

.code

        start:

                MOV AX, @DATA       ;Load Data in temp register

                MOV DS, AX       ;Load data into data into Data Segment

                LEA SI, arr    ;Load data into SI

                MOV AX, 05H  ; Immediate Addressing, moving 5 into AX

                MOV BX, AX    ; Register Addressing, moving AX content into BX

                MOV AX, ds:[0300H];Direct Addressing, moving [BX] content into AX

                MOV BX, 0300H; Immediate Addressing, moving 300 into BX

                MOV AX, ds:[BX];Indirect Addressing, moving [BX] content into AX

                MOV AX, ds:[BX + 04H];Base Addressing, moving [BX + 04] content into AX

                MOV AX, ds:[SI + 04H];Index Addressing, moving[SI + 04] content into AX

                MOV AX, ds:[BX + SI + 01H];Base Index addressing

                end start

        .end
```

A5)

String operations

REP

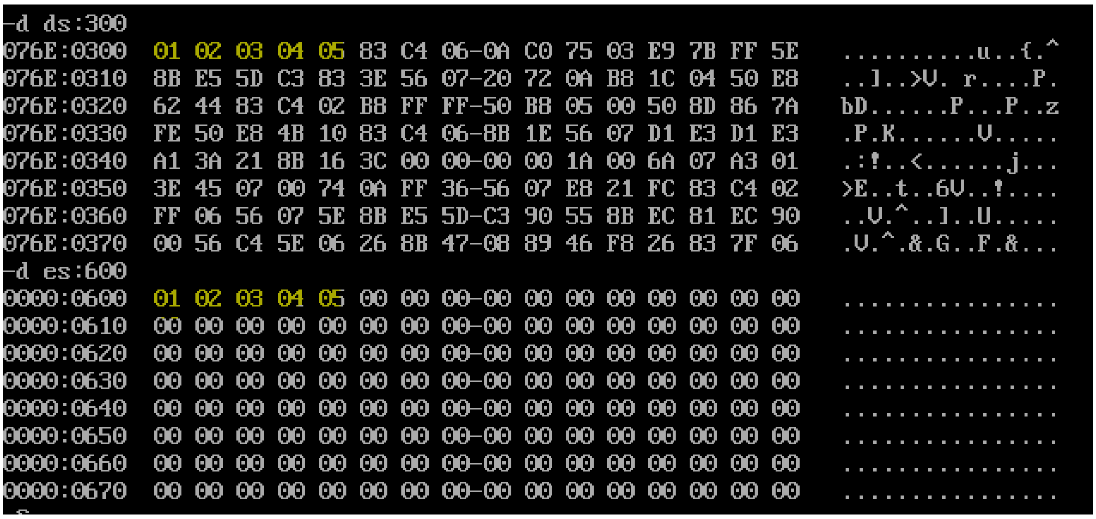
Repeat a given instruction till CX=0



```
AX=0000  BX=0000  CX=0005  DX=0000  SP=0040  BP=0000  SI=0300  DI=0600
DS=076E  ES=0000  SS=079F  CS=076A  IP=0018   NV UP EI PL NZ NA PO NC
076A:0018 F3            REPZ
076A:0019 A4            MOVSB
-t

AX=0000  BX=0000  CX=0004  DX=0000  SP=0040  BP=0000  SI=0301  DI=0601
DS=076E  ES=0000  SS=079F  CS=076A  IP=0018   NV UP EI PL NZ NA PO NC
076A:0018 F3            REPZ
076A:0019 A4            MOVSB
-t

AX=0000  BX=0000  CX=0003  DX=0000  SP=0040  BP=0000  SI=0302  DI=0602
DS=076E  ES=0000  SS=079F  CS=076A  IP=0018   NV UP EI PL NZ NA PO NC
076A:0018 F3            REPZ
076A:0019 A4            MOVSB
-t

AX=0000  BX=0000  CX=0002  DX=0000  SP=0040  BP=0000  SI=0303  DI=0603
DS=076E  ES=0000  SS=079F  CS=076A  IP=0018   NV UP EI PL NZ NA PO NC
076A:0018 F3            REPZ
076A:0019 A4            MOVSB
```

MOVS

Moves the contents of byte given by DS:SI into ES:DI



```
-d ds:300
076E:0300  01 02 03 04 05 83 C4 06-0A C0 75 03 E9 7B FF 5E   ..........u..{.^
076E:0310  8B E5 5D C3 83 3E 56 07-20 72 0A B8 1C 04 50 E8   ..]..>V. r....P.
076E:0320  62 44 83 C4 02 B8 FF FF-50 B8 05 00 50 8D 86 7A   bD......P...P..z
076E:0330  FE 50 E8 4B 10 83 C4 06-8B 1E 56 07 D1 E3 D1 E3   .P.K......V.....
076E:0340  A1 3A 21 8B 16 3C 00 00-00 00 1A 00 6A 07 A3 01   .:!..<......j...
076E:0350  3E 45 07 00 74 0A FF 36-56 07 E8 21 FC 83 C4 02   >E..t..6V..!....
076E:0360  FF 06 56 07 5E 8B E5 5D-C3 90 55 8B EC 81 EC 90   ..V.^..]..U.....
076E:0370  00 56 C4 5E 06 26 8B 47-08 89 46 F8 26 83 7F 06   .V.^.&.G..F.&...
-d es:600
0000:0600  01 02 03 04 05 00 00 00-00 00 00 00 00 00 00 00   ................
0000:0610  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0000:0620  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0000:0630  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0000:0640  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0000:0650  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0000:0660  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0000:0670  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

CMPS

Compares byte at ES:DI with word at DS:SI and sets flags

```
AX=0000  BX=0000  CX=0005  DX=0000  SP=0040  BP=0000  SI=0300  DI=0600
DS=076E  ES=0000  SS=079F  CS=076A  IP=0025   NV UP EI PL NZ NA PO NC
076A:0025 F3            REPZ
076A:0026 A6            CMPSB
-t

AX=0000  BX=0000  CX=0000  DX=0000  SP=0040  BP=0000  SI=0305  DI=0605
DS=076E  ES=0000  SS=079F  CS=076A  IP=0027   NV UP EI PL ZR NA PE NC
076A:0027 8D360003      LEA     SI,[0300]                          DS:0300=0201
```

SCAS

Compares byte at ES:DI with AL and sets flags according to result

```
AX=0001  BX=0000  CX=0005  DX=0000  SP=0040  BP=0000  SI=0300  DI=0600
DS=076E  ES=0000  SS=079F  CS=076A  IP=0034   NV UP EI PL ZR NA PE NC
076A:0034 AE            SCASB
-t

AX=0001  BX=0000  CX=0005  DX=0000  SP=0040  BP=0000  SI=0300  DI=0601
DS=076E  ES=0000  SS=079F  CS=076A  IP=0035   NV UP EI PL ZR NA PE NC
076A:0035 7303          JNB     003A
```

LODS

Moves the byte at address DS:SI into AL; SI is incr/decr by 1

```
AX=0001  BX=0000  CX=0005  DX=0000  SP=0040  BP=0000  SI=0301  DI=0601
DS=076E  ES=0000  SS=079F  CS=076A  IP=003B   NV UP EI PL NZ NA PO NC
076A:003B AC            LODSB
-t

AX=0002  BX=0000  CX=0005  DX=0000  SP=0040  BP=0000  SI=0302  DI=0601
DS=076E  ES=0000  SS=079F  CS=076A  IP=003C   NV UP EI PL NZ NA PO NC
076A:003C 47            INC     DI
```

STOS

Moves contents of AL to byte address given by ES:DI; DI is incr/dec by 1

```
AX=0002  BX=0000  CX=0005  DX=0000  SP=0040  BP=0000  SI=0302  DI=0602
DS=076E  ES=0000  SS=079F  CS=076A  IP=003D   NV UP EI PL NZ NA PO NC
076A:003D AA            STOSB
-t

AX=0002  BX=0000  CX=0005  DX=0000  SP=0040  BP=0000  SI=0302  DI=0603
DS=076E  ES=0000  SS=079F  CS=076A  IP=003E   NV UP EI PL NZ NA PO NC
076A:003E F4            HLT
```

Code

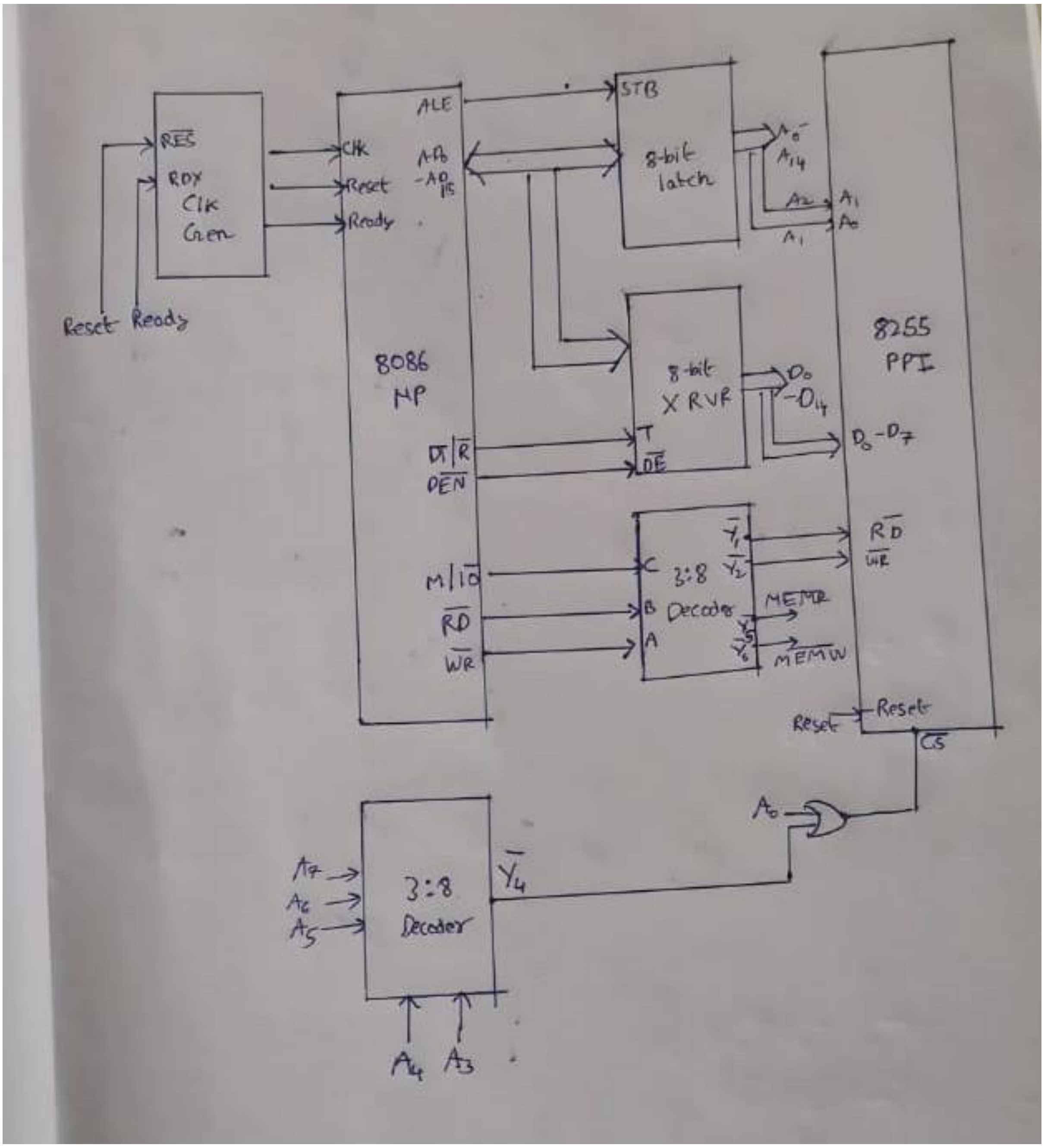
;REP, MOVS,CMPS, SCAS, LODS, and STOS

```
.model small
.stack 64
.data
org 300H
arr db 01H, 02H, 03H, 04H, 05H
.code
    same:
        MOV DX, 01H

    start:
        MOV AX, @DATA       ;Load Data in temp register
        MOV DS, AX       ;Load data into data into Data Segment
    movs_:
        LEA SI, arr    ;Load data into SI
        MOV DI, 600H
        MOV AX, 00H
        MOV ES, AX
        MOV CX, 05H
        CLD
        REP MOVSB
    cmps_:
        LEA SI, arr    ;Load data into SI
        MOV DI, 600H
        MOV CX, 05H
        CLD
        REPE CMPSB
    scas_:
        LEA SI, arr    ;Load data into SI
```

```
        MOV DI, 600H
        MOV CX, 05H
        MOV AL, 01H
        CLD
        SCASB
        JNC lods_
        MOV DX, 01H
lods_:
        INC SI
        LODSB
stod_:
        INC DI
        STOSB
        HLT
        end start
.end
```

A6)

Interfacing diagram of 8086, 8255 PPI



Code

.model small

.stack 64

.data

ORG 3000H

tem db 64

.code

```
start:
        MOV DX, 3000H
        IN AL, DX
        CMP AL, 64H
        JNZ START
        INC DX
        OUT DX, AL
skip:
        HLT
        end start
.end
```

A7)

Interrupt for printing Data (INT 21H)

We experiment here with the interrupt which helps in printing the data onto the console

```
AX=096B  BX=0000  CX=0030  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=076A  IP=000D   NV UP EI PL NZ NA PO NC
076A:000D CD21          INT     21
-t

AX=096B  BX=0000  CX=0030  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A0   NV UP DI PL NZ NA PO NC
F000:14A0 FB            STI
-t

AX=096B  BX=0000  CX=0030  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A1   NV UP EI PL NZ NA PO NC
F000:14A1 FE38          ???     [BX+SI]                         DS:0000=45
-t
Experimenting with Interupyts!
AX=096B  BX=0000  CX=0030  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A5   NV UP EI PL NZ NA PO NC
F000:14A5 CF            IRET
-t
```

Pre interrupt Phase:

```
AX=096B  BX=0000  CX=0030  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=076A  IP=000D   NV UP EI PL NZ NA PO NC
076A:000D CD21          INT     21
-d cs:0
076A:0000  B8 6B 07 8E D8 8D 36 00-00 8D 14 B4 09 CD 21 F4   .k....6.......!.
076A:0010  45 78 70 65 72 69 6D 65-6E 74 69 6E 67 20 77 69   Experimenting wi
076A:0020  74 68 20 49 6E 74 65 72-72 75 70 79 74 73 21 24   th Interrupyts!$
076A:0030  03 E9 11 01 B8 2F 00 50-8B 46 FC 8B 56 FE 05 0C   ...../.P.F..V...
076A:0040  00 52 50 E8 EA 48 83 C4-04 50 E8 7B 0E 83 C4 04   .RP..H...P.{....
076A:0050  3D FF FF 74 03 E9 ED 00-C4 5E FC 26 8A 47 0C 2A   =..t.....^.&.G.*
076A:0060  6B 09 00 00 A5 14 6B 09-00 00 0D 00 6A 07 A3 01   k.....k.....j...
076A:0070  C4 04 50 8D 86 FA FE 50-E8 17 73 83 C4 06 8B B6   ..P....P..s.....
```

As we can see there are initially few instructions present in the code segment before the interrupt call and the SP points to 0040

As we can see, interrupts are executed in 3 parts, namely

1) **STI (Set interrupt flag)**

   Here, the Interrupt Flag (IF) in the EFLAGS is set. After the IF flag is set, the processor begins responding to external, maskable interrupts after the next instruction is executed.

```
AX=096B  BX=0000  CX=0030  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=076A  IP=000D   NV UP EI PL NZ NA PO NC
076A:000D CD21          INT     21
-d cs:0
076A:0000  B8 6B 07 8E D8 8D 36 00-00 8D 14 B4 09 CD 21 F4   .k....6.......!.
076A:0010  45 78 70 65 72 69 6D 65-6E 74 69 6E 67 20 77 69   Experimenting wi
076A:0020  74 68 20 49 6E 74 65 72-72 75 70 79 74 73 21 24   th Interrupyts!$
076A:0030  03 E9 11 01 B8 2F 00 50-8B 46 FC 8B 56 FE 05 0C   ...../.P.F..V...
076A:0040  00 52 50 E8 EA 48 83 C4-04 50 E8 7B 0E 83 C4 04   .RP..H...P.{....
076A:0050  3D FF FF 74 03 E9 ED 00-C4 5E FC 26 8A 47 0C 2A   =..t.....^.&.G.*
076A:0060  6B 09 00 00 A5 14 6B 09-00 00 0D 00 6A 07 A3 01   k.....k.....j...
076A:0070  C4 04 50 8D 86 FA FE 50-E8 17 73 83 C4 06 8B B6   ..P....P..s.....
```

```
AX=096B  BX=0000  CX=0030  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A0   NV UP DI PL NZ NA PO NC
F000:14A0 FB            STI
-d cs:0
F000:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

   We can see that the Code segment has changed and SP now points to 003A. Due to this step, the processor registers and device code is saved.

2) **Execute Device code**

Here, the necessary peripheral device necessities are fulfilled and we can see the change in IP (Instruction pointer) to prove that. We can also see the line “Experimenting with interrupts” to say that the functions are executed

```
AX=096B  BX=0000  CX=0030  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A0   NV UP DI PL NZ NA PO NC
F000:14A0 FB            STI
-d cs:0
F000:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

```
Experimenting with Interrupyts!
AX=096B  BX=0000  CX=0030  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A5   NV UP EI PL NZ NA PO NC
F000:14A5 CF            IRET
-d cs:0
F000:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

3) IRET (Return to original program)

Here, we come back to the initial program and continue the usual execution. We can show that by the change in CS content, Change in IP and SP.

```
Experimenting with Interrupyts!
AX=096B  BX=0000  CX=0030  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A5   NV UP EI PL NZ NA PO NC
F000:14A5 CF            IRET
-d cs:0
F000:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
F000:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

```
AX=096B  BX=0000  CX=0030  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=076A  IP=000F   NV UP EI PL NZ NA PO NC
076A:000F F4            HLT
-d cs:0
076A:0000  B8 6B 07 8E D8 8D 36 00-00 8D 14 B4 09 CD 21 F4   .k....6.......!.
076A:0010  45 78 70 65 72 69 6D 65-6E 74 69 6E 67 20 77 69   Experimenting wi
076A:0020  74 68 20 49 6E 74 65 72-72 75 70 79 74 73 21 24   th Interrupyts!$
076A:0030  03 E9 11 01 B8 2F 00 50-8B 46 FC 8B 56 FE 05 0C   ...../.P.F..V...
076A:0040  00 52 50 E8 EA 48 83 C4-04 50 E8 7B 0E 83 C4 04   .RP..H...P.{....
076A:0050  3D FF FF 74 03 E9 ED 00-C4 5E FC 26 8A 47 0C 2A   =..t.....^.&.G.*
076A:0060  6B 09 00 00 A5 14 6B 09-00 00 0F 00 6A 07 A3 01   k.....k.....j...
076A:0070  C4 04 50 8D 86 FA FE 50-E8 17 73 83 C4 06 8B B6   ..P....P..s.....
```

Code

```
.model small

.stack 64

.data

        start db "Experimenting with Interrupyts!" , "$"

.code

        MOV AX, @data        ;Load Data in temp register

        MOV DS, AX           ;Load address into data segment

        LEA SI, start    ;Load first date pointer

        lea dx , [SI]

        mov ah , 09h

        int 21h

        HLT

        end

.end
```