Experiment Number : (8)

The Stack

Object:

To study the stack and stack pointer and to get to know the stack pointer.

Theory:

Stack is an area of memory for keeping temporary data. Stack also is used by CALL instruction to keep return address for procedure, RET instruction gets this value from the stack and returns to that offset. Quite the same thinks happens when INT instruction calls an interrupt, it stores in stack flag register, code segment and offset, RET instruction is used to return from interrupt call.

We can also use the stack to keep any other data, there are two instructions work with the stack:

- **Push instruction: To stores 16 bit value in the stack**

  PUSH REG
  PUSH SREG
  PUSH MEMORY
  PUSH Immediate
  REG: AX, BX, CX, DX, SI, DI, BP, SP.
  SREG: DS,CS, ES, SS.
  MEMORY: [BX], [BX]+SI+7, etc…
  Immediate: 5, -24, 3Fh, 10001101b, etc…

- **Pop instruction: To get 16 bit value from the stack**

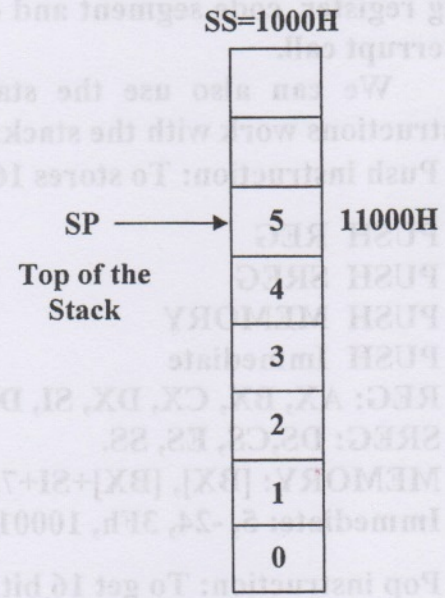
  POP REG
  POP SREG
  POP MEMORY
  REG: AX, BX, CX, DX, SI, DI, SP, SP.
  SREG: DS, ES, SS, (Except CS).
  MEMORY: [BX], [BX]+SI+7, etc…

Notes:

- **PUSH and POP work with 16 bit values only.**
- **PUSH IMMEDIATE works only on 80186 CPU and later!**
- **The stack uses FILO (First In Last Out) algorithm.**
- **PUSH AX , store the register AX to the stack, at first stores the Low (AL) and then stored the High (AH).**
- **POP AX, get the last value stored in the stack to the register AX, at first the last value is stored in the High (AH) and then other value is stored in the Low (AL).**

Example:

You have the values (1, 2, 3, 4 ,5)H in the stack, and the value of the stack segment is 1000H and the physical of the last values which stored in the stack (5) is 11000H. Show how can you pop the number from the stack? And what is the stack pointer in each case?



1. **The last SP = Physical address – SS*10H**
   =11000-(1000*10) = 1000H
   POP AX
   AX = 0504
2. **Other SP = 1000 + 2 = 1002H**
   POP BX
   BX = 0302H
3. **Other SP = 1002 + 2 =1004H**
   POP CX
   CX = 0100H
4. **Final SP = 1004 + 2 =1006H**

It is very important to do equal number of PUSHs and POPs, otherwise the stack maybe corrupted and it will be impossible to return to the operating system. As you already know we use RET instruction to return to operating system, so when the program starts there is a return address in the stack (generally it's 0000H).

PUSH and POP instruction are especially useful because we don't have too many registers to operate with, so here is a trick:
Store the original value of the register in the stack (Using PUSH).
Use the register for any purpose.
Restore the original value of the register from the stack (Using POP).

The stack memory area is set by SS (Stack Segment) register, and SP (Stack Pointer) register. Generally operating system sets values of these registers on program start.

Notes:

- **"PUSH *source*" instruction dose the following:**
  Subtract 2 from SP register.
  Write the value of source to the address SS:SP.
- **"POP *destination*" instruction dose the following:**
  Write the value at address SS:SP to destination.
  Add 2 to SP register.
- **The current address pointed by SS:SP is called The TOP of The Stack.**

Notes:

For COM files stack segment is generally the code segment, and stack pointer is set to value of (0FFFEH). At the address SS:0FFFE stored a return address for RET instruction that executed in the end of the program.
You can visually see that stack operation by clicking on (Stack) button on emulator window. The TOP of the stack is marked with "<" sign.

Procedure:

1. **Write a 8086 program to do the following:**
   - **Store the value 1234H to the register AX.**
   - **Store the value 5678H to the register BX.**
   - **Store the register AX to the stack.**
   - **Copy the value of the register BX to AX.**
   - **Restore the value of the register AX from the stack.**
2. **Execute above program and write the result of the registers which used.**
3. **Repeat above program except restore the value of the register AX from the stack.**
4. **Execute the new program and find the result of the stack.**
5. **What are the value of the SS and SP and what physical address of starting of the stack.**

Home Work:

1. **You have AX=1212h and BX=3434h write a 8086 program to exchange between above two register by using the stack.**

2. **Write a 8086 program to calculate the expression C=A*B, where A is the value stored in the memory at the physical address 12345h and 12346h, and B is stored also in memory at the physical address 10000h and 10001h, store the result in the stack.**

3. **Find the value of the SS and SP after run above program and then calculate the physical address of the stack.**

4. **What are mean of the FILO?**

Experiment Number : (9)

Applications

Q1) You have DT_1 = 1,2,3,4,5,6,7,8,9,10 in decimal mode. Write 8086 program to calculate the area of the square which the length represented by DT_1. Store the results into DT_2.

Q2) You have DT_1 = 1,2,3,4,5,6,7,8,9,10 in decimal mode. Write 8086 program to calculate the area of the rectangle which the length and width represented by DT_1 respectively. Store the results into DT_2.

Q3) Write 8086 program to convert the string of the DT_1 to the lower case. Store the results into DT_2.
DT1 = COMPUTER

Q4) You have 5 marks stored in DT_1, find the average and store the results in DT_2.
DT_1 = 55,45,10,64,96 in decimal mode.

Q5) You have DT_1 = 1,1,1,0,1,0,1,0,1,1,1,0,0,0,0 in binary mode. Write 8086 program to find the sum and carry of full adder which the input (A, B, C_i) represented by DT_1 respectively. Store the results into DT_2 (SUM & CARRY).

Q6) You have DT_1 = 1,2,3,4,5,6,7,8,9,10 in decimal mode. Write 8086 program to calculate the expression $Y = X^2 - 10 + X$; where X is represented by DT_1 store the results in DT_2.

Q7) You have DT_1 = 1000,2000,3000,4000,5000,6000,7000,8000,9000,9999 in decimal mode. Write 8086 program to copy DT_1 to the stack.

Q8) Write 8086 program to store the even number from 1000 to 2000 in DT_1.

Q9) You have DT_1 = 1,2,3,4,5,6,7,8,9,10 in decimal mode. Write 8086 program to find the sum of the DT_1. store the results into DT_2, and then find the average. Store it into DT_3.

Experiment Number : (10)

Unsigned Jump Instructions

Object:

To understand the unsigned jump instructions and how can use it properly

Theory

Controlling the program flow is a very important thing, this is where your program can make decisions according to certain conditions. Jump instruction is the important instruction which that can control of the programs flow.

There are two main types of this instruction, and there are

1. **Unconditional JUMP instruction: this instruction is execute without any condition.**
2. **Conditional JUMP instruction: these instruction not execute until the condition is true**

Unconditional Jump Instructions:

The basic instruction that transfers control to another point in the program is unconditional jump instruction, and represented by:

JMP Label

To declare a Label in your program, just type the name and add":" to the end, label can any character combination but cannot start with a number.

For example here 3 legal label definitions:

Label1:
Label2:
A:

Label can be declared on a separate line or before any other instruction, for example:

X1:
MOV AX, 1 **or**
X2: MOV AX, 2.

Short Conditional Jump Instructions:

Unlike JMP instruction that dose an unconditional jump, there are instructions that do a conditional jumps (jump only where some condition in act).

These instructions (Conditional jump instructions) contain also two main types and there are:

1. **Unsigned conditional Jump instructions.**
2. **Signed conditional Jump instructions.**

Unsigned Conditional Jump Instructions:

This instructions represented by:

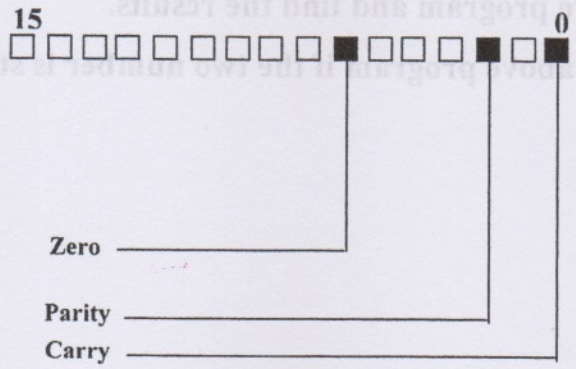
1.	JC Label	Jump if carry	CF = 1
2.	JPE Label	Jump if parity even	PF = 1
3.	JPO Label	Jump if parity odd	PF = 0
4.	JZ Label	Jump if zero	ZF = 1
5.	JE Label	Jump if equal	ZF = 1
6.	JA Label	Jump if above	CF = 0 & ZF = 0
7.	JB Label	Jump if below	CF = 1
8.	JAE Label	Jump if above or equal	CF = 0
9.	JBE Label	Jump if below or equal	CF = 1 OR ZF = 1

Some of the above instructions can be negative as seen below:

1.	JNC Label	Jump if not-carry	CF = 0
2.	JNZ Label	Jump if not-zero	ZF = 0
3.	JNE Label	Jump if not-equal	ZF = 0
4.	JNA Label	Jump if not-above	CF = 1 OR ZF = 1
5.	JNB Label	Jump if not-below	CF = 0

All these instructions the sign number is neglect (I.E the negative number can't represented by above instructions)

All of the above instructions test the some of the flag and its:



Example:

Write 8086 program to find the largest number of DT_1. store the result into DT_2. DT_1 = 7,1,5,4,8,6,3,2,9

```
.DATA
DT1 DB 7, 1, 5, 4, 8, 6, 3, 2, 9
DT2 DB 0
.CODE
MOV AX, @DATA
MOV DS, AX
MOV SI, OFFSET DT1
MOV DI, OFFSET DT2
MOV BX, 0
MOV CX, 0008
MOV AL, [SI+BX]
L: INC BX
CMP AL, [SI+BX]
JA M
MOV AL, [SI+BX]
M:LOOP L
MOV [DI], AL
RET
```

Procedure

1. **Write 8086 program to compare between two numbers (X_1 & X_2) each one is 1 byte.**
2. **If X_1 is grater than X_2 store X_1 into DT_1.**
3. **If X_1 is equal X_2 store X_1 into DT_2.**
4. **if X_1 is smallest than X_2 store X_2 into DT_3.**
5. **Let X_1 = 0A & X_2 = 0C in hexadecimal mode**
6. **Execute the above program and find the results.**
7. **How can modify above program if the two number is stored into DT_4.**

Home Work:

1. **What are the results of the program below, explain the program flow**

```
.CODE
MOV AX, 5
MOV BX, 3
JMP CALC
BACK: JMP STOP
CALC:
ADD AX, BX
JMP BACK
STOP:
RET
```

2. **You have DT_1 = 1,5,7,2,10,4,8,4, write 8086 program to find the largest and smallest number of the DT_1 store the results (Largest) into DT_2, (Smallest) into DT_3.**
3. **Write 8086 program to compare between two number (X, Y) that stored in the stack, if the X is grater than Y store 1 in DT_1, if X is equal Y store 2 in DT_1 otherwise store 3 in DT_3.**
4. **Write a short note on:**

   JZ label, JE label
5. **Write a program to convert any small character to capital character stored in the location DT_1. Store the results into DT_2**

   DT1 = AaBbCcDd

Experiment Number : (11)

Signed Conditional Jump Instructions

Object:

To recognize the different between the signed and unsigned conditional jump instructions, and also where are used.

Theory:

Sign Number:

There is no way to say for sure whether the hexadecimal byte (0FFh) is positive or negative, it can represent both decimal value "255" & "-1".
8 bit can be used to create 256 combinations (include zero), so we simply presume the first 128 combinations (0..127) will represent positive number and next 128 combinations (128..255) will represent negative numbers.

In order to get "-5" we should subtract 5 from the number of combination (256), so will get 256 – 5 = 251.
Using this complex way to represent negative numbers has some meaning in math when you add "-5" to "5" you should get "zero". This is what happens when processor add tow bytes 5 and 251, the result gets over 255, because of the overflow processor gets zero.

Example:

+	11111011	-5	Or	251
	00000101	5		
	100000000	0		

This bit goes off because it dose not fit into byte

When combinations 128..255 are used the high bit is always 1, so this maybe used to determine the sign of the number.

The same principle is used for words (16 bit values), 16 bits create 65536 combinations, first 32768 combinations (0..32767) are used to represent positive numbers, and next 32768 combinations (32768..65535) represent negative numbers.

Overflow:

Any changing of the number from positive +ve to negative –ve or changing from negative –ve to positive +ve is represent the over flow.

Example:

Let AL = 127 in decimal
BL = 1
AL = AL + BL = 128
The OF flag will set as one because 127 represent +ve and 128 represent –ve value (changing from positive to negative).

AL = 129 in decimal
BL = 2
AL = AL – BL = 127
The OF flag will set as one because 129 represent -ve and 127 represent +ve value (changing from negative to positive).

AX = 32767 in decimal
BX = 1
AX = AX + BX = 32768
The OF flag will set as one because 32767 represent +ve and 32768 represent – ve value (changing from positive to negative).

AX = 32770 in decimal
BX = 4
AX = AX – BX = 32766
The OF flag will set as one because 32770 represent -ve and 32766 represent +ve value (changing from negative to positive).

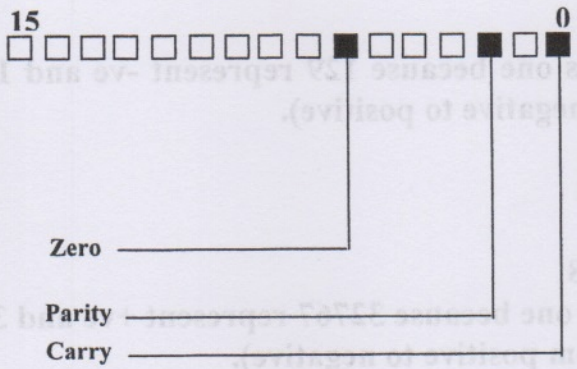
Jump instructions for signed number:

1.	JE Label	Jump if equal	ZF = 1
2.	JZ Label	Jump if zero	ZF = 1
3.	JG Label	Jump if grater	ZF = 0 & SF=OF
4.	JL Label	Jump if less	SF ≠ OF
5.	JGE Label	Jump if grater or equal	SF=OF
6.	JLE Label	Jump if less or equal	ZF = 1 OR SF≠OF

Some of the above instructions can be negative as seen below:

1.	JNE Label	Jump if not-equal	ZF = 0
2.	JNZ Label	Jump if not-zero	ZF = 0
3.	JNG Label	Jump if not-grater	ZF = 1 OR SF≠OF
4.	JNL Label	Jump if not-less	SF = OF

All of the above instructions test the some of the flag and its:



Example:

Write 8086 program to find the min value of DT_1, store the result into DT_2.
DT_1 = -5, 3, 1, -6, 2, 1, 0, -11, 10, 9

Solution:

```
.DATA
DT1 DB -5,3,1,-6,2,1,0,-11,10,10,9
DT2 DB 0
.CODE
MOV AX@ DATA
MOV DS, AX
MOV SI, OFFSET DT1
MOV DI, OFFSET DT2
MOV CX, 0009
MOV BX, 0
MOV AL, [SI+BX]
L: INC BX
CMP AL, [SI+BX]
JLE M
MOV AL. [SI+BX]
M: LOOP L
MOV [DI], AL
RET
```

Procedure:

You have 5 numbers (-5, 4, -7, 4, 0) write 8086 program to do:

1. **Let above program is DT_1.**
2. **Let DS=1000H , SI=0000 , DI=0050H**
3. **Find the no. of the number that is negative.**
4. **Store the results into DT_2.**
5. **Execute the above program and find the results.**

Home Work:

1. **Write 8086 program to find the max and min values of the DT_1, store the results into DT_2**

   DT_1 = -5, 3, 1, -6, 2, 1, 0, -11, 10, 9

2. **Write 8086 program to store the numbers (-5) to (5) into DT_1 without using the counter CX.**

3. **Write 8086 program to store the no. of the even numbers of the DT_1 store the results into DT_2.**

   DT_1 = -10, 10, 21, -1, 0, -11

4. **Write 8086 program to store the number (1) to DT_3 if the summation of DT_1 and DT_2 caused zero results otherwise store (0) in DT_3**

   DT_1 = 1, 5, 10, -12, 0

   DT_2 = 4, -5, -10, -12, 0

Experiment Number : (12)

Applications

Q1) You have 10 marks of students stored in the location DT_1, write 8086 program to find the number of students that pass, store the results into DT_2

DT_1 = 50, 30, 60, 100, 40, 45, 30, 20, 70, 90.

Q2) Write 8086 program to convert any lower case of string store in the location DT_1 to the upper case. Store the results into DT_2

DT_1 = University OF Technology

Q3) Write 8086 program to find the no. of the equal number of DT_1 & DT_2. Store the results into DT_3.

DT_1 = 10, 20, 30, 40, 50

DT_2 = 10, 15, 30, 35, 50

Q4) Write 8086 program to subtract DT_1 from DT_2 and then store (1) in DT_3 if the numbers is positive otherwise store (0).

DT_1 = 10, -10, 20, 30, 40

DT_2 = 5, -1, 40, 30, 4

Q5) Write 8086 program to find the number of character that is capital letter of the string stored in DT_1 , store the result into DT_2.
DT_1 = COmPutEr

Experiment Number : (13)

Interrupt Service Routine

Object:

To learn how can the user print the any string on the screen.

Theory

The interrupt which is providing the printing the string on the user-screen is INT21.

The method to use the INT21 is:

1. **Define the string which you want to print in (.DATA) section.**
2. **Use the number (10) to move the cursor to the next line of the screen.**
3. **Use the number (13) to move the cursor to the beginning of the line of the screen.**
4. **Use ('$') always at the end of your string.**
5. **Loaded the register DX by offset of your string.**
6. **Loaded the register AH by (09).**
7. **Write the command (INT 21h) when you want to print your message.**

Example:

Write 8086 program to print (Welcome) 10 time in your screen.

Solution

```
.DATA
MESSAGE DB 'Welcome',13,10,'$'
.CODE
MOV AX@DATA
MOV DS,AX
MOV DX, OFFSET MESSAGE
MOV CX, 0010
MOV AH, 09
L: INT 21H
LOOP L
RET
```

Problems:

1. **Write 8086 program to compare between two numbers A & B and then print on the screen**
   - **Largest if A>B.**
   - **Equal if A=B.**
   - **Smallest if A<B.**

   Let A = 100 & B = 200
2. **Write 8086 program to print the shape below on the user screen**

```
*
**
***
****
*****
```