Assignment - 1

Q1. Explain macro with example.

Ans. A macro is a group of repetitive instructions in a program which are modified only once and can be used as many times as necessary.

- The assembler generates machine codes for the group of instructions each time the macro is called

→ Syntax:-

```
macroname MACRO
instruction 1
instruction 2
:
ENDM
```

→ eg:

```
sum MACRO X,Y
    MOV AX,X
    MOV BX,Y
    ADD AY,BX
ENDM
```

Q2. Explain procedure with example

Ans. • The procedure, or subroutine, is an important part of any computer systems architecture.

- A procedure is a group of instructions that ~~does~~ usually performs one task, it is a reusable section of the software that is stored in memory once, but used as often as necessary.

→ Syntax:-

```
name PROC
; The procedure code
```

```
;goes here....
RET
name ENDP
```

→ Eg:

```
myProc PROC
    mov DX, offset msg
    mov AH, 09 H
    INT 21H
    RET
myProc ENDP
```

Q3. Write difference between procedure and macro:

Ans.

	MACRO	PROCEDURE
1.	Sequence of instructions that is written within the macro definition to support modular programming.	1. Set of instructions which can be called repetitively that performs a specific task.
2.	Requires more memory	2. Requires less memory
3.	Does not require CALL and RET instructions	3. Requires CALL and RET instructions
4.	Machine code is generated each time MACRO is called.	4. Machine code generates only once.
5.	Parameters are passed as a part of statement which calls the macro.	5. Parameters are passed in registers and memory locations of stack.
6.	Macro executes faster than a procedure.	6. Procedure executes slower than a macro.
7.	Used for Less than 10 instructions.	7. Used for more than 10 instructions.

Assignment-2

Q1. Explain string instruction of 80386.

Ans.
- Each string instruction may require a source operand, a destination operand, or both.
- For 32-bit segments, string instructions use ESI and EDI registers to point to the source and destination operands, respectively.
- There are five basic instructions for processing strings. They are:

1) MOVS:
- This instruction moves 1 byte, word or Double word of data from memory location to another.

2) LODS:
- This instruction ~~stores data from register~~ loads from memory. If the operand is of one byte, it is loaded into the AL register, if the operand is one ~~word~~ byte, it is loaded into the AL register, if the operand is one word, it is loaded into the AX register and a double word is loaded into the EAX register.

3) STOS:
- This instruction stores data from register CAL, AX, or EAX) to memory.

4) CMPS:
- This instruction compares two data items in memory. Data could be of a byte size, word or double word.

5) SCAS:
- This instruction compares the contents of a register CAL, AX or EAX) with the contents of an item in memory.

Q2. Explain direction flag.

Ans.
- The direction flag is a flag that controls the left-to-right or right-to-left direction of string processing, stored on the FLAGS register on all 80386 processors.
- When it is set to 0 (using the cld direction flag instruction CLD), it means that instructions that auto increment the source index and destination index (like movs) will increase both of them.
- In case it is set to 1 (using the set-direction flag instruction STD), the instruction will decrease them.
- This flag is used to determine the direction ('forward' or 'backward') in which several bytes of data will be copied from one place in the memory, to another.
- The direction is important mainly when the original data position in memory and the target data position overlap.

Assignment-4

Q1. Explain macro with example.

Ans. A macro is a group of repetitive instructions in a program which are modified only once and can be used as many times as necessary.

- The assembler generates machine codes for the group of instructions each time the macro is called.

→ Syntax:

```
macroname MACRO
instruction 1
instruction 2
;
ENDM
```

→ eg:

```
SUM MACRO X, Y
    mov AX, X
    mov BX, Y
    ADD AY, BX
ENDM
```

Q2. Explain procedure with example.

Ans. The procedure, or, subroutine, is an important part of any computer systems architecture.

- A procedure is a group of instructions that usually performs one task, it is a reusable section of the software that is stored in memory once, but used as often as necessary.

→ Syntax:-

```
name PROC
; The procedure code
; goes here.....
```

RET
name ENDP

→ Eg:

```
myProc PROC
        mov DX, offset msg
        mov AH, 09H
        INT 21H
        RET
myProc ENDP
```

Q3. Write difference between procedure and macro.

Ans

MACRO	PROCEDURE
1. Sequence of instructions that is written within the macro definition to support modular programming.	1. Set of instructions which can be called repetitively that performs a specific task.
2. Requires more memory.	2. Requires less memory.
3. Does not require CALL and RET instructions.	3. Requires CALL and RET instructions
4. Machine code is generated each time MACRO is called.	4. Machine code generates only once.
5. Parameters are passed as a part of statement which calls the macro.	5. Parameters are passed in registers and memory locations of stack.
6. Macro executes faster than a procedure.	6. Procedure executes slower than a macro.
7. Used for less than 10 instructions.	7. Used for more than 10 instructions.

Assignment - 5

Q1. Explain BT, JS, Loop instruction with example.

Ans. 1) BT:

- BT saves the value of the bit indicated by the base (first operand) and the bit offset (second operand) into the carry flag.
- Flags affected : CF
- eg:-

```
BT BX, CX
BT EAX, ECX
```

2) JS

- Jump if sign flag=1 or jump if negative
- Conditions for jump:- SF=1
- eg:-

```
TEST eax,eax
JS  0xABCD0000
```

3) LOOP

- Loop decrements the count register without changing any of the flags.
- Conditions are then checked for the form of Loop being used.
- If the conditions are met, a short jump is made to the label given by the operand to Loop.
- Flags affected: None
- eg:-

```
MOV CX, 0004H
MOV BX, 7526H
LABEL1 MOV AX, CODE
OR BX, AX
LOOP LABEL1
```

Q2. Explain Paging on 80386.

Ans. • Paging is required if you want to run multiple 8086 tasks on single 80386.

• Paging is beneficial on multi-user system, in an open architecture, than structured system.

• When paging is disabled, the 4 GB physical address space is organized into segments that can be of any size from 1 byte to 4 GB.

→ Paging is useful or necessary for any of the following reasons:-

• To create multiple V86 tasks. Each task must map the lower megabyte of linear addresses to different physical locations.

• To emulate the megabyte wrap. Wrapping truncates the higher-bit to give addresses only up to 20 bits long.

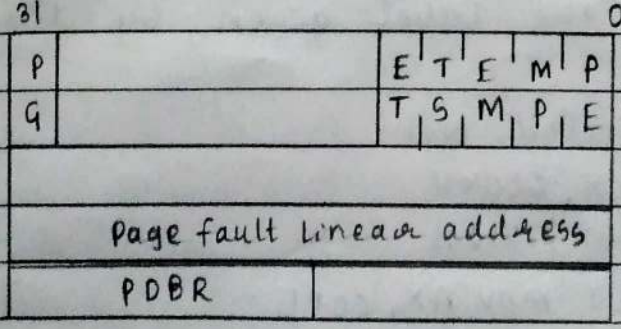
• To create a virtual address space larger than the physical address space.

• To redirect or trap references to memory mapped I/O devices.

• To share 8086 OS code or ROM code that is common to several 8086 programs that are continuously executing.

Q3. Draw control registers of 80386.

Ans.



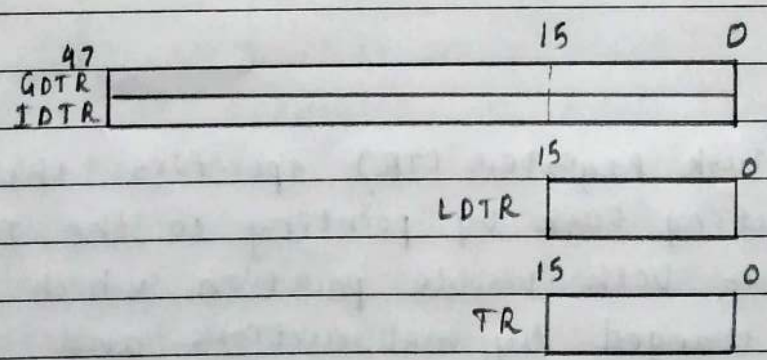
Assignment - 6

Q1. Explain system addresses registers.

Ans.: There are four system address registers: TR (Task register), IDTR (Interrupt Descriptor Table Register), GDTR (Global Descriptor Table Register), LDTR (Local ~~Desc~~ Descriptor Table Register)

→ These registers hold the addresses for the four special descriptor table segments.

- The TR (Task register) points to the Table state segment.
- The IDTR (Interrupt Descriptor Table Register) points to the Interrupt Descriptor (IDT)
- The GDTR (Global Descriptor Table Register) points to the Global Descriptor Table (GDT).
- The LDTR (Local Descriptor Table Register) points to the Local Descriptor Table (LDT).

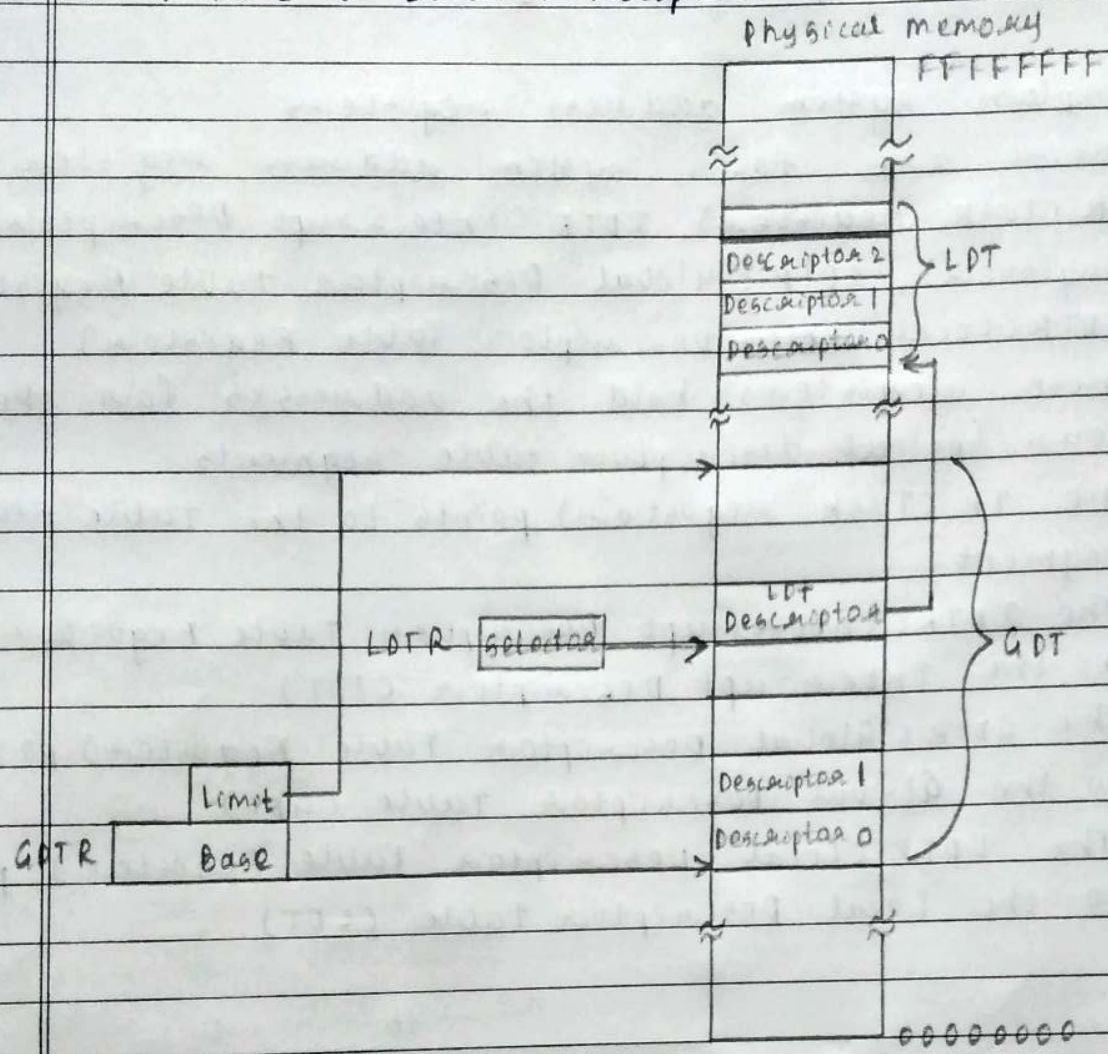


Q2. Explain Segment selectors ~~for~~ LDTR and TR

Ans.: 1). LDTR is a 16-bit register.

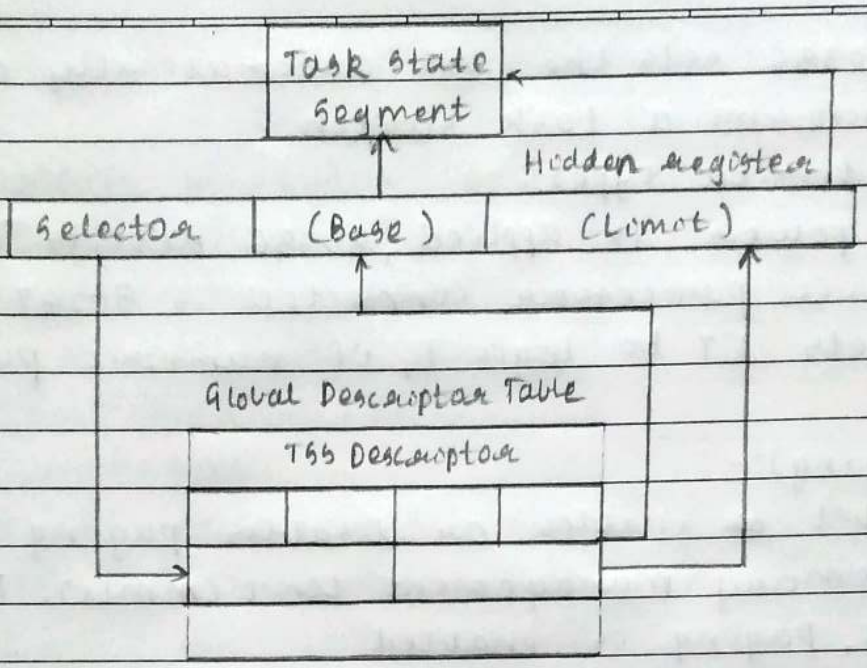
- It does not specify any limit or base address for the segment but it specifies the address of the LDT descriptor stored in the Global Descriptor Table (GDT).
- It shows how contents of LDTR are used indirectly

to define a Local Descriptor Table.



2) . The Task Register (TR) specifies the currently executing task by pointing to the TSS.

- It has both visible portion which can be read and changed by instructions and invisible portion (maintained by the 80386 to correspond to the visible portion which can not be read by any instruction).
- The selector on the visible portion is used to specify a TSS descriptor on the GDT and invisible portion is used to cache the base and limit values from the TSS descriptor.



Q3. Explain CR0 register.

Ans. The CR0 holds the msw (machine status word).

- It contains six status bits:-

i) PE (protection Enable):-

- This bit is similar to the VM bit in EFLAGS in that it controls the 80386's mode of operation. When PE is set, it is in protection mode otherwise it operates in Real mode.

ii) MP (math present):-

- When this bit is set, the 80386 assumes that real floating point hardware (80287 or 80387) is present in the system.
- When this bit is clear, the 80386 assumes that no such coprocessor exists, and will not attempt to use real floating point hardware.

iii) EM (Emulate Coprocessor):-

- When this bit is set, the 80386, will generate an exception 11 (device not available) whenever it attempts to execute a floating point instruction.

iv) TS (Task switched):-

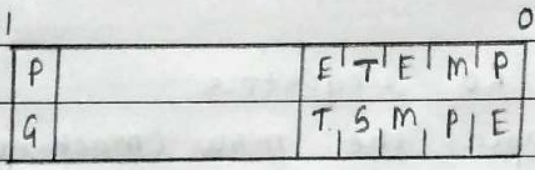
- The 80386 sets the bit automatically every time it performs a task switch.

v) ET (extension type):-

- When power is applied, 80386 detects whether numeric processor connected is 80287 or 80387 and sets ET to logic 1, if numeric processor is 80387.

vi) PG (paging):-

- This bit enables or disables paging mechanism on memory management Unit (MMU). If bit is set, paging is enabled.



Assignment -7

Q1. Explain Assembler Directives.

Ans. An assembler supports directives to define data, to organise segments to control procedure, to define ~~data,~~ macros.

→ Assembler directives:-

i) DB directive

- The DB directive is used to declare a ~~BYE~~ BYTE-2-BYTE variable.
- eg:

```
Byte1 DB 10H
```

ii) DW directive

- The DW directive is used to declare a Word type variable.
- eg:

```
Word DW 1234H
```

iii) DD directive

- The DD directive is used to declare a double word.
- eg:

```
Dword1 DW 12345678H
```

iv) STRUCT and ENDS directive:-

~~STRUCT~~

- The STRUCT directive tells the assembler that ~~the~~ a user unonitialized data structure follows.
- A structure ends by using ENDS directive.
- Syntax:-

```
structure_element_name element_data_type ?
....
...
....
ENDS
```

v) The EQU directive:-

- The EQU directive is used to give name to some symbol or value

→ The following operators can also be used to declare an Equate:

i) THIS BYTE

eg: A-BYTE EQU THIS BYTE
DB 10

2) THIS WORD

eg: A_word EQU THIS WORD
DW 1000

3) THIS DWORD

eg: A_dword EQU THIS DWORD
DD 4294967295

vi) EXTERN directive:-

- It is used to tell the assembler that the name or Label following the directive are some other assembly module
- eg: PROCEDURE-HERE SEGMENT
EXTERN SMART-DIVIDE: FAR
PROCEDURE-HERE ENDS

Q2. Explain E-Flag register

Ans. - EFLAGS registers contains 13 flags which can be categorized in 3 different groups:-

1. Status flags:

- These flags reflect the state of a particular program.

i) CF (carry flag):

- This bit is set by arithmetic instructions that generate either a carry or a borrow

ii) PF (parity flag):

- The parity bit is set by most instructions if the least significant 8-bit of the result contains even number of ones.

iii) AF (Auxiliary carry flag):
- This bit is set when there is a carry or borrow after a nibble addition or subtraction, respectively.

iv) ZF (Zero flag):
- Zero flag is set to 1, if the result of an operation is zero.

v) SF (Sign flag):
- Sign flag is set to 1, if the result of an operation is negative (MSB = 1).

vi) OF (Overflow flag):
- This flag is set if the result of a signed operation is too large to fit in the number of bits available (7-bits for 8-bit numbers) to represent it.

2. Control flags:
- These flags directly affect the operation of few instructions.

i) DF (direction flag)
- When the DF flag is cleared string operations process strings from low memory up towards high memory. SI and DI are incremented by 1.
- If the D flag is set, then SI and DI are decremented by 1 after each operation to process strings from high to low memory.

3. System flags:
- These flags reflect the current status of the machine.

i) VM (Virtual memory) flag:-
- This flag indicates operating mode of 80386.

- When this flag is set, 80386 switches from protected mode to virtual 8086 mode.

ii) R (Resume) flag / Restart flag:

- This flag, when set allows selective masking of some exceptions at the time of debugging.

iii) NT (nested flag):

- This flag is set when one system task invokes another task.

iv) IOPL (I/O Priviledge level):

- The two bits on the IOPL are used by the processor and the operating system to determine your applications access.

v) IF (interrupt flag):

- When IF is set, the 80386 recognizes and handles external hardware interrupts on its INTR pin.
- If the onterrupt flag is cleared, 80386 ignores any inputs on this pin.

vi) TF (trap flag):

- Trap flag allows to single-step through programs.

Assignment-8

Q1. Explain string specific instructions.

Ans. There are five basic instructions for processing strings. They are:

i) MOVS:-

- This instruction moves 1 Byte, Word or Doubleword of data from memory location to another.

ii) LODS:-

- This instruction loads from memory
- If the operand is of one byte, it is loaded into the AL register, if the operand is on word, it is loaded onto the AX register and a doubleword is loaded into the EAX register.

iii) STOS:-

- This instruction stores data from register (AL, AX or EAX) to memory.

iv) CMPS:-

- This instruction compares two data ~~items~~ items in memory.
- Data could be of byte size, word or doubleword

v) SCAS:-

- This instruction compares the contents of a register (AL, AX or EAX) with the contents of an item in memory.

Q2. Explain stack specific instructions.

Ans. The stack manipulation instructions include:-

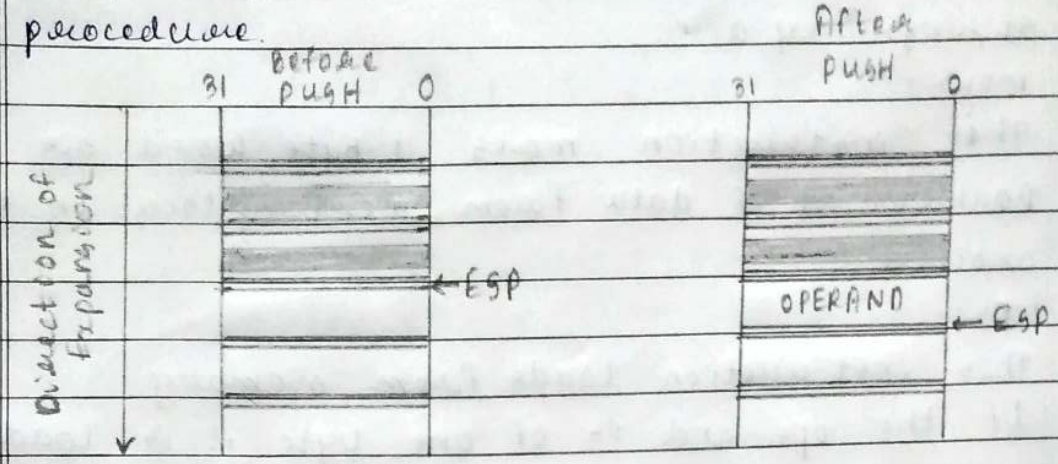
1. ~~Push~~ Push

→ Function:

- To decrement the stack pointer (ESP)
- Then to transfer the source operand to the top of stack indicated by ESP.

→ Use:-

• To place parameters on the stack before calling a procedure.



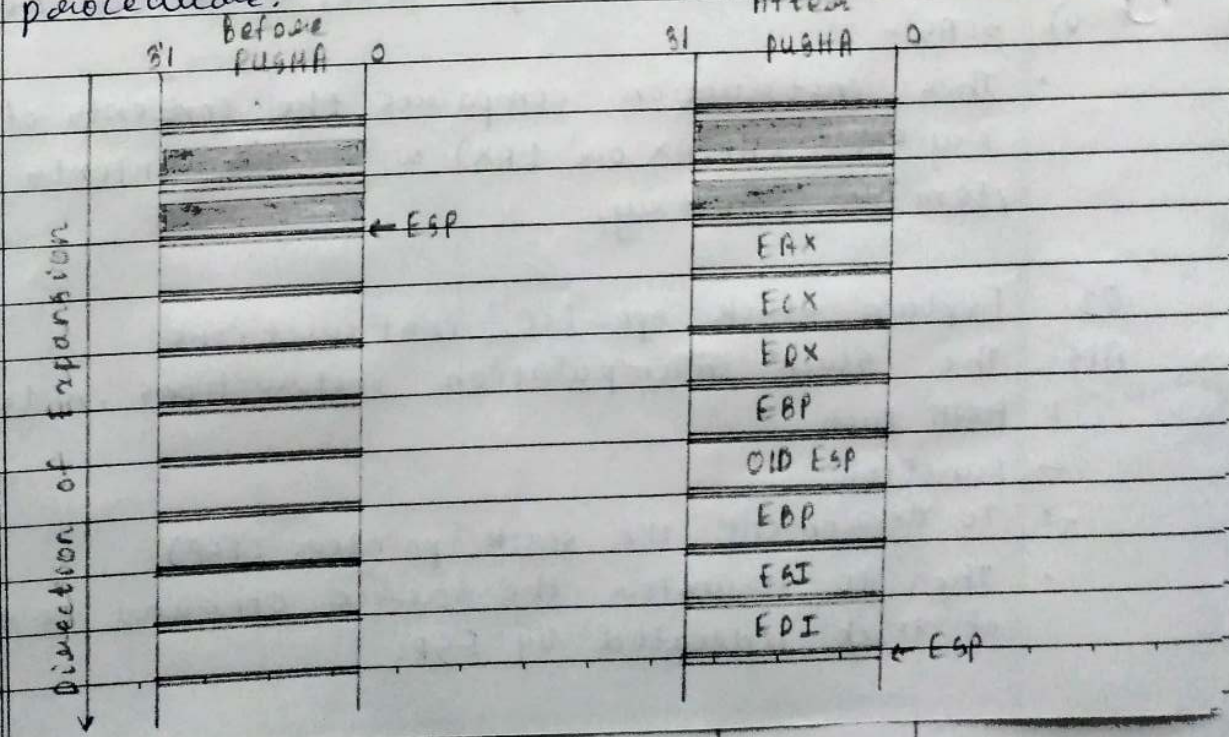
2. ~~PUSH~~ PUSHA

→ Function:-

• To save the contents of 8 general registers on the stack.

→ Use:-

• To simplify procedure calls by reducing the number of instructions required to restore the contents of the general registers for use in a procedure.



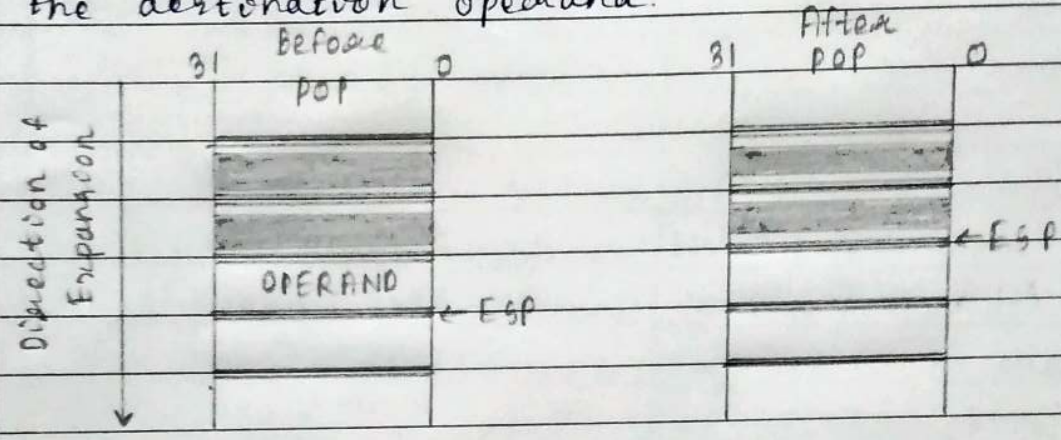
3. POP

→ Function:-

• To increament ESP to point to the new top of stack.

→ Use:-

• To transfer the word or double word at the current top of the stack (indicated by ESP) to the destination operand.

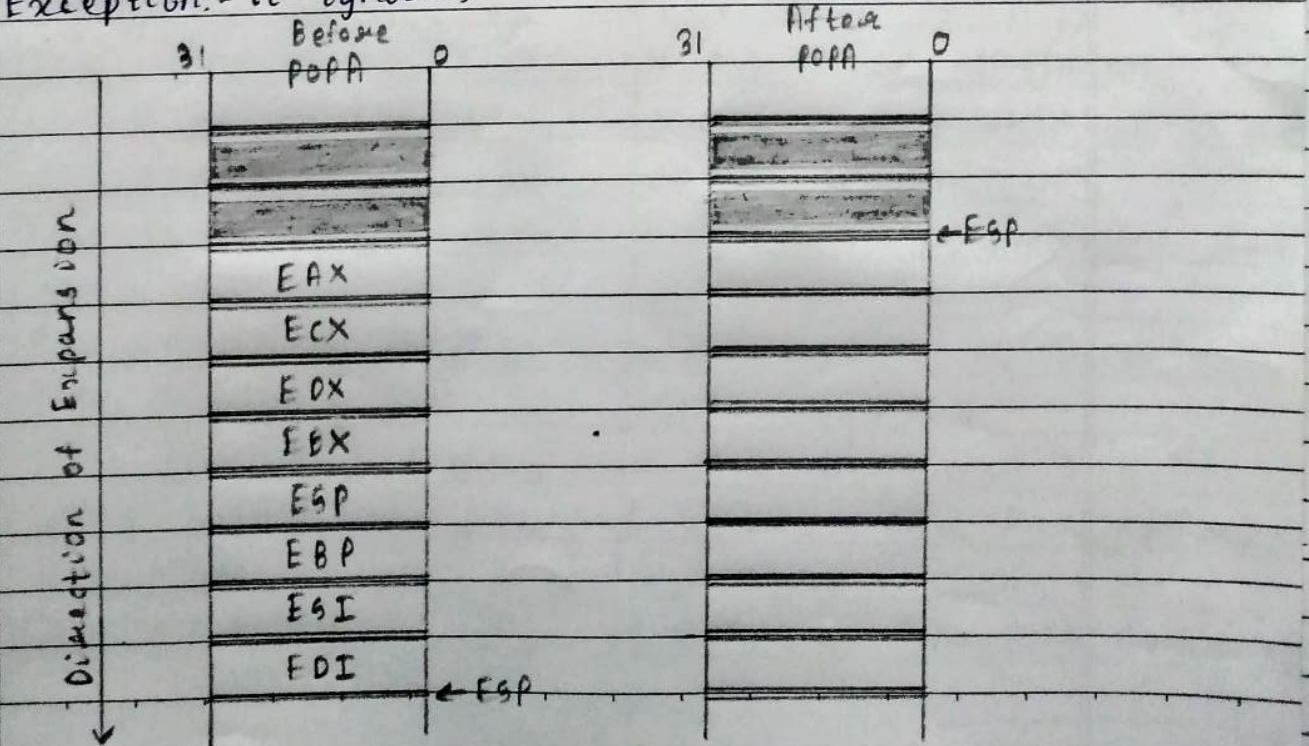


4. POPA

→ Function:-

• To restore the registers saved on the stack by PUSHA

→ Exception:- it ignores the saved value of ESP.



Assignment-9

Q1. What is Procedure? Compare NEAR and FAR procedure.

Ans. The procedure, or subroutine, is a group of instructions that usually performs one task. It is a reusable section of the software that is stored in memory once, but used as often as necessary.

	NEAR Procedure	FAR Procedure
1.	A near procedure refers to a procedure which is on the same code segment from that of the call instruction.	1. A far procedure refers to a procedure which is on the different code segment from that of the call instruction.
2.	It is also called intra-segment procedure	2. It is also called inter-segment procedure call.
3.	A near procedure replaces the old IP with new IP.	3. A far procedure call replaces the old CS:IP pairs with new CS:IP pairs.
4.	The value of old IP is pushed on to the stack. SP = SP-2	4. The value of old CS:IP pairs are pushed on to the stack SP = SP-2.
5.	Less stack locations are required.	5. More stack locations are required.
6.	Procedure is inside Code Segment	6. Procedure is outside Code Segment
7.	Uses keyword near	7. Use keyword FAR
8.	Syntax:- CALL near_procedure_name	8. Syntax:- CALL far_procedure_name
9.	Eg:- call Delay	9. Eg:- call FAR PTR Delay

Q2. Explain PUBLIC and EXTERN directives.

Ans. The directive EXTERN informs the assembler that the names, procedures and labels declared after the directive have already been defined in some other assembly language modules.

- while on other module, where the names and procedures and labels actually appear, they must be declared public, using the PUBLIC directive.

→ eg:

```
MODULE1    SEGMENT
PUBLIC     FACTORIAL FAR
MODULE1    ENDS
MODULE2    SEGMENT
EXTERN     FACTORIAL FAR
MODULE2    ENDS
```

Assignment - 10

Q1. List the different components of motherboard.

Ans. Following are the components of motherboard;

- Expansion slots (PCI Express, PCI, and AGP)
- 3-pin case fan connectors
- Back pane connectors
- Heat sink
- 4-pin (P4) power connector
- Inductor
- Capacitor
- CPU socket
- Nourth bridge
- Screw hole
- memory slot
- Super I/O
- ATA/IDE disk drive primary connection
- 24-pin ATX power supply connector
- coin cell battery (CMOS backup battery)
- RAID
- System panel connectors
- FWH
- Southbridge
- Serial port connector
- USB headers
- Jumpers
- Integrated circuit
- 1394 headers
- SPDIF
- CD-IN
- BIOS
- Bus

- Cache memory
- Chipset
- Diode
- Dip switches
- Electrolytic
- Floppy connection

Assignment – Addition

Q1. Explain ADD and SHIFT algorithm with example.

Ans 1) ADD:

- It adds integers
- It replaces the destination operand with the sum of the source and destination operands.
- Sets CF if overflow occurs.

→ eg:

- ADD AL, 0F0H ; Add immediate ~~ope~~ number 0F0H to contents of AL
- ADD CL, TOTAL [BX] ; Add byte from effective address
- ADD CX, TOTAL [BX] ; Add word from effective address
- ADD ECX, EAX ; Add contents of ECX to the contents of EAX and store result in EDX

2) SHIFT:

i) SAL

- Shifts the destination byte, word, or double word operand left by one or the number of bits specified in the count operand.
- The processor shifts zeroes in from right (low-order) side of the operand as bits exits from the left (high-order) side.
- eg: SAL CX, 1

ii) SAR

- Shifts the destination byte, word, or doubleword operand to the right by one or by the number of bits specified in the count operand.
- The processor preserves the sign of the operand by shifting in zeroes on the left (high-

order) side of the value is positive or by shifting by ones of the value is negative.

- eg: MOV CL, 05H
  SAL AX, CL

Q2. ~~What~~ Explain what is Interrupt.

Ans.
- Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor.
- The microprocessor responds to that interrupt with an ISR (Interrupt Service Routine).

→ There are 2 types of interrupts:-

i) Hardware interrupts:-

- Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

1. INTR:
- The INTR input of the 80386 allows external devices to interrupt 80386 program execution.
- This input is sampled at the beginning of each instruction cycle.

2. Non-maskable Interrupt (NMI):
- ~~NMI requests that~~ As name indicates this interrupt input is non-maskable.
- This input is edge-triggered.
- A valid interrupt on this pin causes 80386 to execute interrupt service routine.

3. Reset:
- Reset inputs forces 80386 to go into reset state.
- This is an active high signal.
- When this signal is high it resets system

resources, such as I/O ports, and the interrupt flag.

ii) Software interrupts

- Some instructions are inserted at the desired position into the program to create interrupts.

→ It includes:-

1) INT- Interrupt instruction with type number
- It is 2-byte instruction.
- First byte provides the opcode and the second byte provides the interrupt type number.