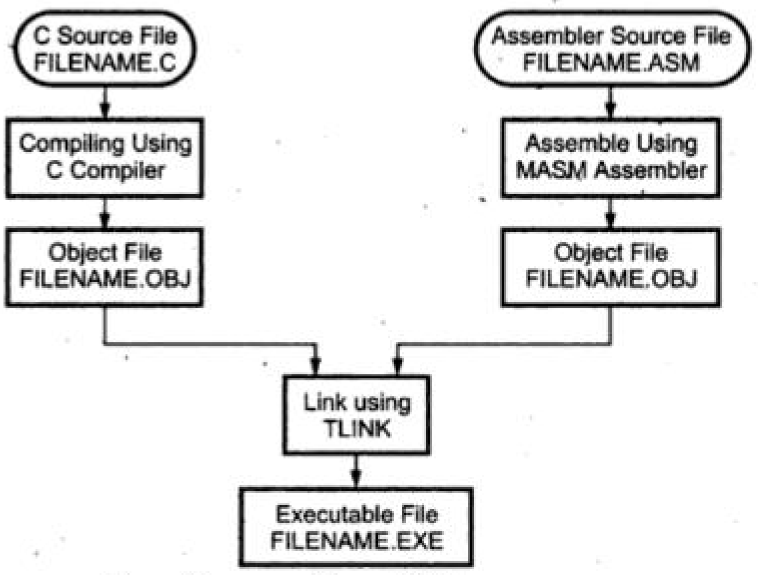
Microprocessor Dec 18

1a. Write a short note on mixed language programming.

1. There are times when programs need to call programs written in other languages referred as mixed language programming. For example, when a particular subprogram is available in a language other than language you are using, or when algorithms are described more naturally in a different language, you need to use more than one language.
2. Mixed-language programming always involves a call to a function, procedure, or subroutine. Mixed-language calls involve calling functions in separate modules. Instead of compiling all source programs with same compiler, different compilers or assemblers are used as per the language used in the programs.
3. Microsoft C supports this mixed language programming. So it can combine assembly code routines in C as a separate language.
4. C program calls assembly language routines that are separately assembled by-MASM (MASM Assembler). These assembled modules are linked with the compiled C modules to get executable file. Fig shows the compile, assemble and link processes using C compiler, MASM assembler, and TUNIC.


C Source File FILENAME.C
Compiling Using C Compiler
Object File FILENAME.OBJ
Assembler Source File FILENAME.ASM
Assemble Using MASM Assembler
Object File FILENAME.OBJ
Link using TLINK
Executable File FILENAME.EXE
Compiler, assembly and link processes

1b. Explain flag register of 80386 microprocessor

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the 80386 is in Protected Mode, the 80386 will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction.

NT (Nested Task, bit 14)

This flag applies to Protected Mode. NT is set to indicate that the execution of this task is nested within another task. If set, it indicates that the current nested task's Task State Segment (TSS) has a valid back link to the previous task's TSS. This bit is set or reset by control transfers to other tasks.

IOPL (Input/Output Privilege Level, bits 12-13)

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAG register.

OF (Overflow Flag, bit 11)

OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow into the sign bit (high-order bit) of the result but did not result in a carry/borrow out of the high order bit, or vice-versa. For 8/16/32 bit operations, OF is set according to overflow at bit 7/15/31, respectively.

DF (Direction Flag, bit 10)

DF defines whether ESI and/or EDI registers post decrement or post increment during the string instructions. Post increment occurs if DF is reset. Post decrement occurs if DF is set.

IF (INTR Enable Flag, bit 9)

The IF flag, when set, allows recognition of external interrupts signaled on the INTR pin.
When IF is reset, external interrupts signaled on the INTR are not recognized. IOPL indicates the maximum CPL value allowing alteration of the IF bit when new values are popped into EFLAGS or FLAGS.

TF (Trap Enable Flag, bit 8)

TF controls the generation of exception 1trap when single-stepping through code. When TF is set, the 80386 generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debug registers DR0±DR3.

SF (Sign Flag, bit 7)

SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.

ZF (Zero Flag, bit 6)

ZF is set if all bits of the result are 0. Otherwise it is reset.

AF (Auxiliary Carry Flag, bit 4)

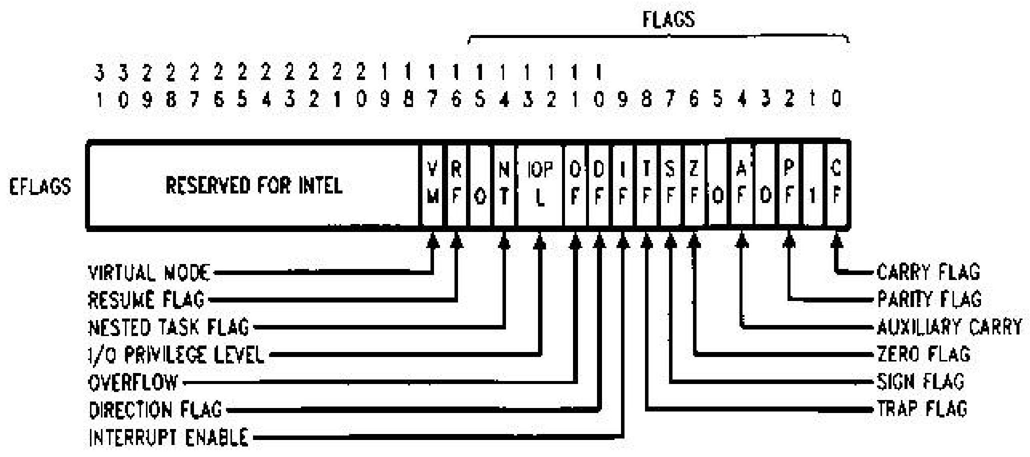
The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise AF is reset. AF is affected by carry out of, or borrow into bit 3 only, regardless of overall operand length: 8, 16 or 32 bits.

PF (Parity Flags, bit 2)

PF is set if the low-order eight bits of the operation contains an even number of ``1's'' (even parity). PF is reset if the low-order eight bits have odd parity. PF is a function of only the low-order eight bits, regardless of operand size.

CF (Carry Flag, bit 0)

CF is set if the operation resulted in a carry out of (addition), or a borrow into (subtraction) the high-order bit. Otherwise CF is reset. For 8-, 16- or 32-bit operations, CF is set according to carry/borrow at bit 7, 15 or 31, respectively.



1d. Give formats of initialisation command words(ICW's) of 8259 PIC.

The Programming 8259 requires two types of command words. Initialization Command Words (ICWs) and Operational Command Words (OCWs). The Programming 8259 can be initialized with four ICWs; the first two are compulsory, and the other two are optional based on the modes being used. These words must be issued in a given sequence. After initialization, the 8259A can be set up to operate in various modes by using three different OCWs; however, they are not necessary to be issued in a specific sequence.

Initialization Command Word 1 (ICW1):

A write command issued to the 8259 with $A_0 = 0$ and $D_4 = 1$ is interpreted as ICW1, which starts the initialization sequence. It specifies,

Initialization Command Word 2 (ICW2)

A write command following ICW1, with $A_0 = 1$ is interpreted as ICW2. This is used to load the high order byte of the interrupt vector address of all the interrupts.

Initialization Command Word 3 (ICW3):

ICW3 is required only if there is more than one 8259 in the system and if they are cascaded. An ICW3 operation loads a slave register in the Programming 8259. The format of the byte to be loaded as an ICW3 for a master 8259 or a slave is shown in the Fig. 14.78. For master, each bit in ICW3 is used to specify whether it has a slave 8259 attached to it on its corresponding IR (Interrupt Request) input. For slave, bits $D_0 – D_2$ of ICW3 are used to assign a slave identification code (slave ID) to the Programming 8259.

Initialization Command Word 4 (ICW4):

It is loaded only, if the D_0 bit of ICW1 is seta The format of ICW4 is shown in Fig. 14.79.

It specifies,

- **Whether to use special fully nested mode or non special fully nested mode.**
- **Whether to use buffered mode or non buffered mode.**
- **Whether to use Automatic EOI or Normal EOI.**
- **CPU used, 8086/8088 or 8085.**

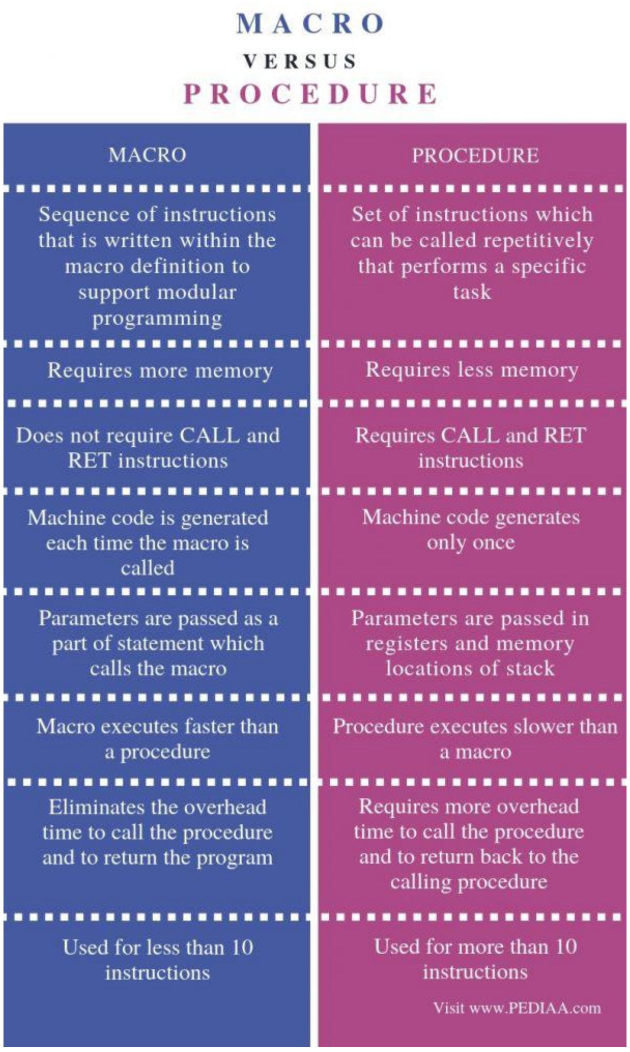
5a. Differentiate procedure and macro .Write a program to find the factorial of a number using procedure.
(First write difference between from the next image and then write program.Write only text from the image)

Program:

```
DATA SEGMENT
A DB 5
DATA ENDS
CODE SEGMENT
        ASSUME DS:DATA,CS:CODE
START:
      MOV AX,DATA
      MOV DS,AX
      MOV AH,00
      MOV AL,A
 L1:  DEC A
      MUL A
      MOV CL,A
      CMP CL,01
      JNZ L1
      MOV AH,4CH
      INT 21H
CODE ENDS
```

```
END START

;OUTPUT:->
;-G CS: 001B
;
;AX=0078  BX=0000  CX=0001  DX=0000  SP=0000  BP=0000  SI=0000
 DI=0000
;DS=0BA8  ES=0B98  SS=0BA8  CS=0BA9  IP=001B   NV UP EI PL ZR N
A PE NC
;0BA9:001B B44C          MOV     AH,4C
```



MACRO
VERSUS
PROCEDURE

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

Visit www.PEDIAA.com

5b. Explain the interrupt structure of 8086 microprocessor.

1. An interrupt is a special condition that arises during the working of a microprocessor. The microprocessor services it by executing a subroutine called Interrupt Service Routine (ISR).
2. There are three sources of interrupts for 8086.

Hardware interrupt-

These interrupts occur as signals on the external pins of the microprocessor. 8086 has two pins to accept hardware interrupts, NMI and INTR.

Software interrupt-

These interrupts are caused by writing the software interrupt instruction INT n where ‘n’ can be any value from 0 to 255 (00H to FFH). Hence all 256 interrupts can be invoked by software.

Error conditions (Exception or types)-

8086 is interrupted when some special conditions occur while executing certain instructions in the program. Example: An error in division automatically causes the INT 0 interrupt.

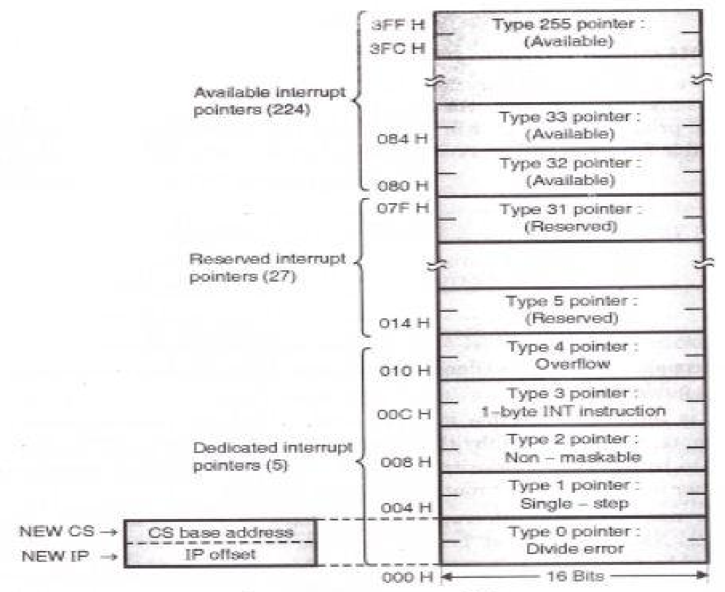
Interrupt Vector Table (IVT):

1. The interrupt vector (or interrupt pointer) table is the link between an interrupt type code and the procedure that has been designated to service interrupts associated with that code. 8086 supports total 256 types i.e. 00H to FFH.
2. For each type it has to reserve four bytes i.e. double word. This double word pointer contains the address of the procedure that is to service interrupts of that type.
3. The higher addressed word of the pointer contains the base address of the segment containing the procedure. This base address of the segment is normally referred as NEW CS.
4. The lower addressed word contains the procedure’s offset from the beginning of the segment. This offset is normally referred as NEW IP.
5. Thus NEW CS: NEW IP provides NEW physical address from where user ISR routine will start.
6. As for each type, four bytes (2 for NEW CS and 2 for NEW IP) are required; therefore interrupt pointer table occupies up to the first 1k bytes (i.e. 256 x 4 = 1024 bytes) of low memory.
7. The total interrupt vector table is divided into three groups namely,

A. Dedicated interrupts (INT 0…..INT 4)

B. Reserved interrupts (INT 5…..INT 31)

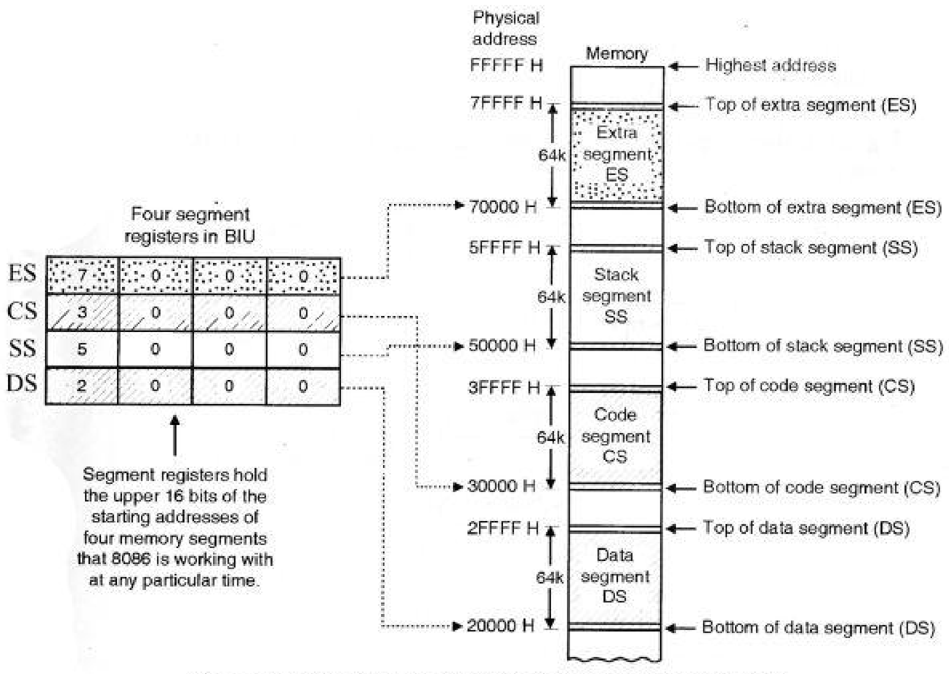
C. Available interrupts (INT 32…..INT 225)


3FF H
3FC H
Type 255 pointer : (Available)
Available interrupt pointers (224)
084 H
Type 33 pointer : (Available)
080 H
Type 32 pointer : (Available)
07F H
Type 31 pointer : (Reserved)
Reserved interrupt pointers (27)
014 H
Type 5 pointer : (Reserved)
010 H
Type 4 pointer : Overflow
00C H
Type 3 pointer : 1-byte INT instruction
Dedicated interrupt pointers (5)
008 H
Type 2 pointer : Non – maskable
004 H
Type 1 pointer : Single – step
NEW CS →
CS base address
NEW IP →
IP offset
Type 0 pointer : Divide error
000 H
16 Bits
Interrupt vector table

6a. Explain segmentation of 8086 microprocessor.Give its advantages.

he need of memory segmentation is explained below:

1. The BIU (Bus Interfacing Unit) contains four special purpose registers called as segment registers. These are Code Segment (CS) register, Stack Segment (SS) register, Extra Segment (ES) register and Data Segment (DS) register.
2. All these are 16 bit registers.
3. The number of address lines in 8086 is 20. So the 8086 BIU will send out a 20 bit address in order to access one of the 1,048,576 or 1MB memory locations.
4. But it is interesting to note that the 8086 does not work the whole 1MB memory at any given time. However it works with only four 64 KB segments within the whole 1 MB memory.
5. The four segment registers actually contain the upper 16 bits of the starting addresses of the four memory segments of 64 KB each with which the 8086 is working at that instant of time.
6. A segment is a logical unit of memory that may be up to 64 kilo bytes long.
7. Each segment is made up of memory contiguous memory locations. It is independent, separately addressable unit.
8. Starting addresses will always be changing. They are not fixed.
9. Figure shows one of the possible ways to position the four 64 KB segments within the 1 MB memory space of 8086.



Physical address
Memory
FFFFF H
Highest address
7FFFF H
Top of extra segment (ES)
64k
Extra segment ES
70000 H
Bottom of extra segment (ES)
5FFFF H
Top of stack segment (SS)
Stack segment SS
50000 H
Bottom of stack segment (SS)
3FFFF H
Top of code segment (CS)
Code segment CS
30000 H
Bottom of code segment (CS)
2FFFF H
Top of data segment (DS)
Data segment DS
20000 H
Bottom of data segment (DS)
Four segment registers in BIU
ES 7 0 0 0
CS 3 0 0 0
SS 5 0 0 0
DS 2 0 0 0
Segment registers hold the upper 16 bits of the starting addresses of four memory segments that 8086 is working with at any particular time.

One way of positioning four 64k byte segments within the 1M byte memory space of an 8086

Advantages of memory segmentation:

1. Segmentation provides a powerful memory management mechanism.
2. It allows programmers to partition their programs into modules that operate independently of one another.
3. Segments allow two processes to easily share data.
4. It allows to extend the address ability of a processor i.e. segmentation allows the use of 16 bit registers to give an addressing capability of 1 MB. Without segmentation, it would require 20 bit registers.
5. Segmentation makes it possible to separate the memory areas for stack, code and data.
6. It is possible to increase the memory size of code data or stack segments beyond 64 KB by allotting more than one segment for each area.

6b. Explain different addressing modes of 8086 microprocessor.

Addressing modes refer to the different methods of addressing the operands. Addressing modes of 8086 are as follows:

Immediate addressing mode-

In this mode, the operand is specified in the instruction itself. Instructions are longer but the operands are easily identified.

Example:

MOV CL, 12H

This instruction moves 12 immediately into CL register. CL ← 12H

Register addressing mode-

In this mode, operands are specified using registers. This addressing mode is normally preferred because the instructions are compact and fastest executing of all instruction forms.

Registers may be used as source operands, destination operands or both.

Example:

MOV AX, BX

This instruction copies the contents of BX register into AX register. AX ← BX

Direct memory addressing mode-

In this mode, address of the operand is directly specified in the instruction. Here only the offset address is specified, the segment being indicated by the instruction.

Example:

MOV CL, [4321H]

This instruction moves data from location 4321H in the data segment into CL.

The physical address is calculated as

DS * 10H + 4321

Assume DS = 5000H

∴PA = 50000 + 4321 = 54321H

∴CL ← [54321H]

Register based indirect addressing mode-

In this mode, the effective address of the memory may be taken directly from one of the base register or index register specified by instruction. If register is SI, DI and BX then DS is by default segment register.

If BP is used, then SS is by default segment register.

Example:

MOV CX, [BX]

This instruction moves a word from the address pointed by BX and BX + 1 in data segment into CL and CH respectively.

CL ← DS: [BX] and CH ← DS: [BX + 1]

Physical address can be calculated as DS * 10H + BX.

Register relative addressing mode-

In this mode, the operand address is calculated using one of the base registers and an 8 bit or a 16 bit displacement.

Example:

MOV CL, [BX + 04H]

This instruction moves a byte from the address pointed by BX + 4 in data segment to CL.

CL ← DS: [BX + 04H]

Physical address can be calculated as DS * 10H + BX + 4H.

Base indexed addressing mode-

Here, operand address is calculated as base register plus an index register.

Example:

MOV CL, [BX + SI]

This instruction moves a byte from the address pointed by BX + SI in data segment to CL.

CL ← DS: [BX + SI]

Physical address can be calculated as DS * 10H + BX + SI