**ASSEMBLY LANGUAGE PROGRAMMING -8086**

**ALP for addition of two 8-bit numbers** ALP **for Subtraction of two 8-bit**

DATA SEGMENT                                    **numbers**

VAR1 DB 85H                                         DATA SEGMENT

VAR2 DB 32H                                         VAR1 DB 53H

RES DB?                                                  VAR2 DB 2AH

DATA ENDS                                            RES DB?

ASSUME CS:CODE, DS:DATA              DATA ENDS

CODE SEGMENT                                    ASSUME CS:CODE,DS:DATA

START: MOV AX, DATA                       CODE SEGMENT

MOV DS, AX                                           START: MOV AX,DATA

MOV AL, VAR1                                       MOV DS,AX

MOV BL, VAR2                                       MOV AL,VAR1

ADD AL, BL                                             MOV BL,VAR2

MOV RES, AL                                          SUB AL,BL

MOV AH, 4CH                                         MOV RES,AL

INT 21H                                                    MOV AH,4CH

CODE ENDS                                            INT 21H

END START                                            CODE ENDS

END START

**ALP for Multiplication of two 8-bit numbers**

DATA SEGMENT

VAR1 DB 0EDH

VAR2 DB 99H

RES DW?

DATA ENDS

ASSUME CS: CODE, DS:DATA

CODE SEGMENT

START: MOV AX, DATA

MOV DS, AX

MOV AL, VAR1

MOV BL, VAR2

MUL BL

MOV RES, AX

MOV AH, 4CH

INT 21H

CODE ENDS

END START

**ALP for division of 16-bit number with 8-bit number**

DATA SEGMENT

VAR1 DW 6827H

VAR2 DB 0FEH

QUO DB?

REM DB?

DATA ENDS

ASSUME CS:CODE,DS:DATA

CODE SEGMENT

START: MOV AX, DATA

MOV DS, AX

MOV AX, VAR1

DIV VAR2

MOV QUO, AL

MOV REM, AH

MOV AH, 4CH

INT 21H

CODE ENDS

END START

**ALP for Subtraction of two 16-bit numbers**

DATA SEGMENT

VAR1 DW 8560H

VAR2 DW 3297H

RES DW?

DATA ENDS

ASSUME CS: CODE,DS:DATA

CODE SEGMENT

START: MOV AX, DATA

MOV DS, AX

MOV AX, VAR1

CLC

SUB AX, VAR2

MOV RES, AX

 MOV AH, 4CH

INT 21H

CODE ENDS

END START

**MODULAR PROGRAMMING**

**1) ALP for Multiplication of two 32-bit numbers**

**2) ALP to Sort a set of unsigned integer numbers in ascending/ descending order using Bubble sort algorithm.**

**ALP for Multiplication of two 32-bit numbers**

DATA SEGMENT

MULD DW 0FFFFH, 0FFFFH

MULR DW 0FFFFH, 0FFFFH

RES DW 6 DUP (0)

DATA ENDS

ASSUME CS: CODE,DS: DATA

CODE SEGMENT

START: MOV AX, DATA

MOV DS, AX

MOV AX, MULD

MUL MULR

MOV RES, AX

MOV RES+2, DX

MOV AX, MULD+2

MUL MULR

ADD RES+2, AX

ADC RES+4, DX

MOV AX, MULD

MUL MULR+2

ADD RES+2, AX

ADC RES+4, DX

JNC K

INC RES+6

K: MOV AX, MULD+2

MUL MULR+2

ADD RES+4, AX

ADC RES+6, DX

MOV AH, 4CH

INT 21H

CODE ENDS

END START

**ALP to Sort a set of unsigned integer numbers in ascending/ descending order using Bubble sort algorithm.**

DATA SEGMENT

A DW 0005H, 0ABCDH, 5678H, 1234H, 0EFCDH, 45EFH

DATA ENDS

ASSUME CS: CODE, DS: DATA

CODE SEGMENT

START: MOV AX, DATA

 MOV DS, AX MOV SI, 0000H MOV BX, A[SI]

DEC BX

 X2: MOV CX, BX MOV SI, 02H

 X1: MOV AX, A[SI]

INC SI

INC SI

CMP AX, A[SI]

XCHG AX, A[SI]

MOV A[SI-2], AX

X3: LOOP X1

DEC BX

JNZ X2

MOV AH, 4CH

INT 21H

CODE ENDS

END START

**Linking And Relocation**

The DOS linking program links the different object modules of a source program and function library routines to generate an integrated executable code of the source program.

 The DOS linking program links the different object modules of a source program and function library routines to generate an integrated executable code of the source program. The main input to the linker is the .OBJ file that contains the object modules of the source programs. Other supporting information may be obtained from the files generated by the MASM. The linker program is invoked using the following options.

 C> LINK or

 C>LINK MS.OBJ

 The .OBJ extension is a must for a file to be accepted by the LINK as a valid object file. The first object may generate a display asking for the object file, list file and libraries as inputs and an expected name of the .EXE file to be generated. The output of the link program is an executable file with the entered filename and .EXE extension. This executable filename can further be entered at the DOS prompt to execute the file.

In the advanced version of the MASM, the complete procedure of assembling and linking is combined under a single menu invokable compile function. The recent versions of MASM have much more sophisticated and user-friendly facilities and options. A linker links the machine codes with the other required assembled codes. Linking is necessary because of the number of codes to be linked for the final binary file. The linked file in binary for ***run*** on a computer is commonly known as executable file or simply ‘.exe.’ file. After linking, there has to be re-allocation of the sequences of

placing the codes before actually placement of the codes in the memory.

 The loader program performs the task of reallocating the codes after finding the physical RAM addresses available at a given instant. The DOS linking program links the different object modules of a source program and function library routines to generate an integrated executable code of the source program. The main input to the linker is the .OBJ file that contains the object modules of the source programs. Other supporting information may be obtained from the files generated by the MASM. The linked file in binary for ***run*** on a computer is commonly known as executable file or simply ‘.exe.’ file. After linking, there has to be re-allocation of the sequences of placing the codes before actually placement of the codes in the memory.The loader program performs the task of reallocating the codes after finding the physical RAM addresses available at a given instant. The ***loader*** is a part of the operating system and places codes into the memory after reading the ‘.exe’ file. This step is necessary

 because the available memory addresses may not start from 0x0000, and binary codes have to be loaded at the different addresses during the run. The loader finds the appropriate start address. In a computer, the loader is used and it loads into a section of RAM the program that is ready to run. A program called*locator* reallocates the linked file and creates a file for permanent location of codes in a standard format.

**Segment combination**

In addition to the linker commands, the assembler provides a means of regulating the way segments in

 different object modules are organized by the linker.

Segments with same name are joined together by using the modifiers attached to the SEGMENT directives. SEGMENT directive may have the form

  Segment name SEGMENT Combination-type

 where the combine-type indicates how the segment is to be located within the load module. Segments that have different names cannot be combined and segments with the same name but no combine-type will cause a linker error. The possible combine-types are:

 ü **PUBLIC –**If the segments in different modules have the same name and combine-type PUBLIC, then they are concatenated into a single element in the load module. The ordering in the concatenation is specified by the linker command.

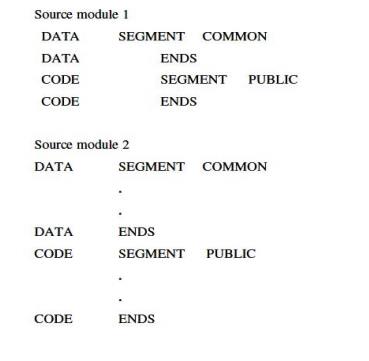
 ü **COMMON –**If the segments in different object modules have the same name andthe combine-type is COMMON, then they are overlaid so that they have the same starting address. The length of the common segment is that of the longest segment being overlaid.

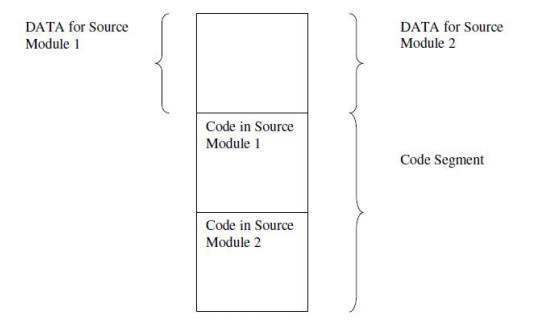
ü **STACK –**If segments in different object modules have the same name and thecombine type

 STACK, then they become one segment whose length is the sum of the lengths of the individually specified segments. In effect, they are combined to form one large stack

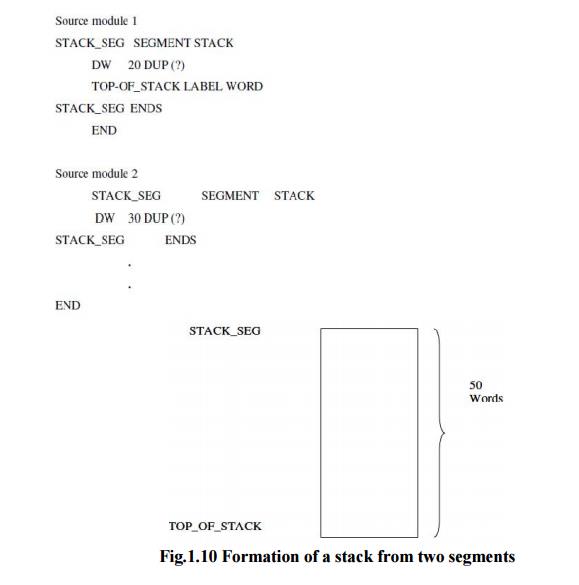
ü **AT** **–** The AT combine-type is followed by an expression that evaluates to a constant which is to be the segment address. It allows the user to specify the exact location of the segment in memory.

**MEMORY**–This combine-type causes the segment to be placed at the last of the loadmodule. If more than one segment with the MEMORY combine-type is being linked, only the first one will be treated as having the MEMORY combine type; the others will be overlaid as if they had COMMON combine-type.



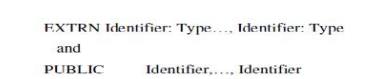


**Fig. 1.9 Segment combinations resulting from the PUBLIC and Common Combination types**



**Access to External Identifiers**

If an identifier is defined in an object module, then it is said to be a*local* (or *internal) identifier*relative to the module. If it is not defined in the module but is defined inone of the other modules being linked, then it is referred to as an *external* (or *global*)*identifier*relative to the module. In order to permit other object modules to reference someof the identifiers in a given module, the given module must include a list of the identifiers to which it will allow access. Therefore, each module in multi-module programs may contain two lists, one containing the external identifiers that can be referred to by other modules. Two lists are implemented by the EXTRN and PUBLIC directives, which have the forms:



where the identifiers are the variables and labels being declared or as being available to other modules.The assembler must know the type of all external identifiers before it can generate the proper machine code, a type specifier must be associated with each identifier in an EXTRN statement. For a variable the type may be BYTE, WORD, or DWORD and for a label it may be NEAR or FAR.

ü One of the primary tasks of the linker is to verify that every identifier appearing in an EXTRN statement is matched by one in a PUBLIC statement. If this is not the case, then there will be an undefined reference and a linker error will occur. The offsets for the local identifier will be inserted by the assembler, but the offsets for the external identifiers and all segment addresses must be inserted by the linking process. The offsets associated with all external references can be assigned once all of the object modules have been found and their external symbol tables have been examined. The assignment of the segment addresses is called*relocation* and is done after the linking process has determined exactly where each segment is to be put in memory.

Memory

**Stacks in 8086 Microprocessor**

The stack is a block of memory that may be used for temporarily storing the contents of the registers inside the CPU. It is a top-down data structure whose elements are accessed using the stack pointer (SP) which gets decremented by two as we store a data word into the stack and gets incremented by two as we retrieve a data word from the stack back to the CPU register.

**Stacks**

 The stack is a block of memory that may be used for temporarily storing the contents of the registers inside the CPU. It is a top-down data structure whose elements are accessed using the stack pointer (SP) which gets decremented by two as we store a data word into the stack and gets incremented by two as we retrieve a data word from the stack back to the CPU register.

 The process of storing the data in the stack is called ‘pushing into’ the stack and the

reverse process of transferring the data back from the stack to the CPU register is known as

‘popping off’ the stack. The stack is essentially *Last-In-First-Out*(LIFO) data segment. This means that the data which is pushed into the stack last will be on top of stack and will be popped off the stack first.The stack pointer is a 16-bit register that contains the offset address of the memory location in the stack segment. The stack segment, like any other segment, may have a memory block of a maximum of 64 Kbytes locations, and thus may overlap with any other segments. Stack Segment register (SS) contains the base address of the stack segment in the memory.

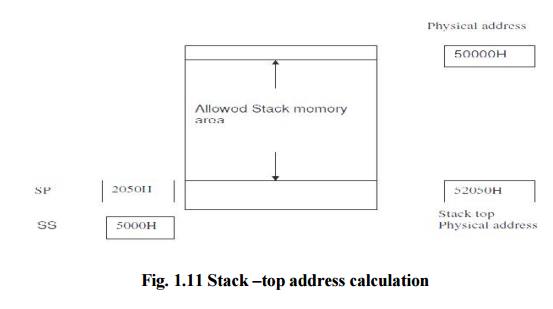
The Stack Segment register (SS) and Stack pointer register (SP) together address the stack-top as explained below:

https://www.brainkart.com/media/extra/mursxxd.jpg

If the stack top points to a memory location 52050H, it means that the location 52050H is already occupied with the previously pushed data. The next 16 bit push operation will decrement the stack pointer by two, so that it will point to the new stack-top 5204EH and the decremented contents of SP will be 204EH. This location will now be occupied by the recently pushed data.

 Thus for a selected value of SS, the maximum value of SP=FFFFH and the segment can have maximum of 64K locations. If the SP starts with an initial value of FFFFH, it will be decremented by two whenever a 16-bit data is pushed onto the stack. After successive push operations, when the stack pointer contains 0000H, any attempt to further push the data to the stack will result in stack overflow.

After a procedure is called using the CALL instruction, the IP is incremented to the next instruction. Then the contents of IP, CS and flag register are pushed automatically to the stack. The control is then transferred to the specified address in the CALL instruction i.e. starting address of the procedure. Then the procedure is executed.



**8086 Microprocessor code Procedures**

1 Calls, Returns, and Procedure Definitions 2 Saving and Restoring Registers 3 Procedure Communication 4 Recursive Procedures

**Procedures**

A procedure is a set of code that can be branched to and returned from in such a way that the code is as if it were inserted at the point from which it is branched to. The branch to procedure is referred to as the *call,* and the corresponding branch back is known as the *return*. The return is always made to the instruction immediately following the callregardless of where the call is located.

**1 Calls, Returns, and Procedure Definitions**

 The CALL instruction not only branches to the indicated address, but also pushes the return address onto the stack. The RET instruction simply pops the return address from the stack. The registers used by the procedure need to be stored before their contents are changed, and then restored just before their contents are changed, and then restored just before the procedure is excited.

 A CALL may be direct or indirect and intrasegment or intersegment. If the CALL is intersegment, the return must be intersegment. Intersegment call must push both (IP) and (CS) onto the stack. The return must correspondingly pop two words from the stack. In the case of intrasegment call, only the contents of IP will be saved and retrieved when call and return instructions are used.

 Procedures are used in the source code by placing a statement of the form at the beginning of the procedure

  Procedure name PROC Attribute and by terminating the procedure with a statement

Procedure name ENDP

The attribute that can be used will be either NEAR or FAR. If the attribute is NEAR, the RET instruction will only pop a word into the IP register, but if it is FAR, it will also pop a word into the CS register.

A procedure may be in:

1. The same code segment as the statement that calls it.

2. A code segment that is different from the one containing the statement that calls it, but in the same source module as the calling statement.

  3. A different source module and segment from the calling statement.

 In the first case, the attribute could be NEAR provided that all calls are in the same code segment as the procedure. For the latter two cases the attribute must be FAR. If the procedure is given a FAR attribute, then all calls to it must be intersegment calls even if the call is from the same code segment. For the third case, the procedure name must be declared in EXTRN and PUBLIC statements.

**2 Saving and Restoring Registers**

 When both the calling program and procedure share the same set of registers, it is necessary to save the registers when entering a procedure, and restore them before returning to the calling program.

MSK PROC NEAR

PUSH AX

PUSH BX

PUSH CX

POP CX

POP BX

POP AX

RET

MSK ENDP

**3 Procedure Communication**

 There are two general types of procedures, those operate on the same set of data and those that may process a different set of data each time they are called. If a procedure is in the same source module as the calling program, then the procedure can refer to the variables directly.

 When the procedure is in a separate source module it can still refer to the source module directly provided that the calling program contains the directive

PUBLIC ARY, COUNT, SUM

EXTRN ARY: WORD, COUNT: WORD, SUM: WORD

**4 Recursive Procedures**

 When a procedure is called within another procedure it called recursive procedure. To make sure that the procedure does not modify itself, each call must store its set of parameters, registers, and all temporary results in a different place in memory

**Macros - 8086 Microprocessor**

Macros is needed for providing the programming ease of a procedure while avoiding the linkage.

**Macros**

**Disadvantages of Procedure**

1. Linkage associated with them.

 2.  It sometimes requires more code to program the linkage than is needed to perform the task. If this is the case, a procedure may not save memory and execution time is considerably increased.

3. **Macros**is needed for providing the programming ease of a procedure while avoiding thelinkage. Macro is a segment of code that needs to be written only once but whose basic structure can be caused to be repeated several times within a source module by placing a single statement at the point of each reference.

 A macro is unlike a procedure in that the machine instructions are repeated each time the macro is referenced. Therefore, no memory is saved, but programming time is conserved (no linkage is required) and some degree of modularity is achieved. The code that is to be repeated is called the prototype code. The prototype code along with the statements for referencing and terminating is called the macro definition.

 Once a macro is defined, it can be inserted at various points in the program by using macro calls. When a macro call is encountered by the assembler, the assembler replaces the

call with the macro code. Insertion of the macro code by the assembler for a macro call is referred to as a macro expansion. In order to allow the prototype code to be used in a variety of situations, macro definition and the prototype code can use dummy parameters which can be replaced by the actual parameters when the macro is expanded. During a macro expansion, the first actual parameter replaces the first dummy parameter in the prototype code, the second actual parameter replaces the second dummy parameter, and so on.

A macro call has the form

%Macro name (Actual parameter list) with the actual parameters being separated by commas.

%MULTIPLY (CX, VAR, XYZ[BX]

**Local Labels**

 Consider a macro called ABSOL which makes use of labels. This macro is used to replace the operand by its absolute value.

%\*DEFINE (ABSOL(OPER)) ( CMP %OPER, 0

JGE NEXT

NEG %OPER

%NEXT: NOP)

 When the macro ABSOL is called for the first time, the label NEXT will appear in the program and, therefore it becomes defined. Any subsequent call will cause NEXT to be redefined. This will result in an error during assembly process because NEXT has been associated with more than one location. One solution to this problem would be to have NEXT replaced by a dummy parameter for the label. This would require the programmer to keep track of dummy parameters used.

 One solution to this problem is the use of **Local Labels.**Local labels are special labels that will have suffixes that get incremented each time the macros are called. These suffixes are two digit numbers that gets incremented by one starting from zero. Labels can be declared as local label by attaching a prefix **Local. Local List of Local labels** at the end of first statement in the macro definition.

**THE 8086 MICROPROCESSOR**

 1. What is microprocessor?

 A microprocessor is a multipurpose, programmable, clock-driven , register-based electronic device that reads binary information from a storage device called memory, accepts binary data as input and processes data according to those instructions, and provides result as output.

 2. What is Accumulator?

 The Accumulator is an 8-bit register that is part of the arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

 3. What is stack?

 The stack is a group of memory locations in the R/W memory that is used for temporary storage of binary information during the execution of a program

 4. What is a subroutine program?

 A subroutine is a group of instructions written separately from the main program to perform a function that occurs repeatedly in the main program. Thus subroutines avoid the repetition of same set of instructions in the main program.

 5. Define addressing mode.

 Addressing mode is used to specify the way in which the address of the operand is specified within the instruction.

 6. Define instruction cycle.

It is defined as the time required to complete the execution of an instruction.

 7 . Write a program to add a data byte located at offset 0500H in 2000H segment to another data byte available at 0600H in the same segment and store the result at 0700H in the same segment.

MOV AX, 2000H; initialize DS with value MOVDS, AX; 2000H

  MOV AX, [500H]; Get first data byte from 0500H offset ADD AX, [600H]; Add this to the second byte from 0600H MOV [700H],AX; store AX in 0700H

HLT; Stop.

 8. What are the different types of addressing modes of 8086 instruction set? The different addressing modes are:

 i. Immediate

 ii. Direct

 iii. Register

 iv.Register indirect

 v. Indexed

 vi.Register relative

 vii.Based indexed

 viii.         Relative based indexed

  9. What are the different types of instructions in 8086 microprocessor? The different types of instructions in 8086 microprocessor are:

i. Data copy / transfer instructions

  ii. Arithmetic and logical instructions

  iii. Branch instructions

  iv.Loop instruction

  v. Machine control instruction

  vi.Flag manipulation instruction

  vii.Shift and rotate instruction

  viii.         String instruction

 10.            What is assembly level programming?

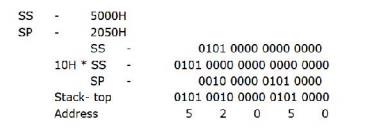
 A program called assembler is used to convert the mnemonics of instruction and data into their equivalent object code modules. The object code modules are further converted into executable code using linker and loader programs. This type of programming is called assembly level programming.

 11. What is a stack?

 Stack is a top-down data structure, whose elements are accessed using a pointer that is implemented using the SS and SP registers. It is a LIFO data segment.

 12. How is the stack top address calculated?

The stack top address is calculated using the contents of the SS and SP register. The contents of stack segment (SS) register is shifted left by four bit positions (multiplied by (0h)) and the resulted 20-bit content is added with the 16-bit offset value of the stack pointer (SP) register.



13. What are macros?

 Macros are small routines that are used to replace strings in the program. They can have parameters passed to them, which enhances the functionality of the micro itself‟

 14.how are constants declared?

 Constants are declared in the same way as variables, using the format: Const – Label EQU 012h

 When the constants label is encountered, the constant numeric value is exchanged for the string.

 15. Write an assembly language program for a 16-bit increment and will not affect the

contents of the accumulator.

 MACRO inc16 variable; Increment two bytes starting at “variable” Local INC16 End

INC variable; Increment the low 8 bits

PUSH ACC

MOV A variable; Are the incremented low 8 bits = 0? JNZ INC 16 End

INC variable + 1

Inc16 End; Yes – increment the upper 8 bits

 POP ACC

END MAC

 16. What will happen if a label within a macro is not declared local?

 If a label within a macro is not declared local, then at assembly time, there will be two types of errors:

I. The first will state that there are multiple labels in the source.

II. The second will indicate that jump instructions don’t know which one to use.

 17. Write an assembly language program to load the accumulator with a constant value. MACRO invert value

if (value==0) MOV A, #1

else

clr A end if

END MAC.

18. What is the difference between the microprocessor and microcontroller?

Microprocessor does not contain RAM, ROM and I/O ports on the chip. But a microcontroller contains RAM, ROM and I/O ports and a timer all on a single chip.

19. What is assembler?

The assembler translates the assembly language program text which is given as input to the assembler to their binary equivalents known as object code. The time required to translate the assembly code to object code is called access time. The assembler checks for syntax errors & displays them before giving the object code.

20. What is loader?

The loader copies the program into the computer’s main memory at load time and begins the program execution at execution time.

21. What is linker?

A linker is a program used to join together several object files into one large object file. For large programs it is more efficient to divide the large program modules into smaller modules. Each module is individually written, tested & debugged. When all the modules work they are linked together to form a large functioning program.

22 .Explain ALIGN & ASSUME.

The ALIGN directive forces the assembler to align the next segment at an address divisible by specified divisor. The format is ALIGN number where number can be 2, 4, 8 or 16. Example ALIGN 8.

The ASSUME directive assigns a logical segment to a physical segment at any given time. It tells the assembler what address will be in the segment registers at execution time. Example ASSUME CS: code, DS: data, SS: stack

23. Explain PTR & GROUP

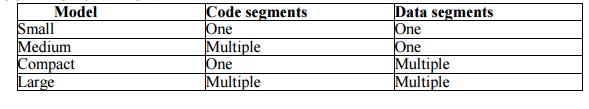
A program may contain several segments of the same type. The GROUP directive collects them under a single name so they can reside in a single segment, usually a data segment. The format is Name GROUP Seg-name,…..Seg-name

PTR is used to assign a specific type to a variable or a label. It is also used to override the declared type of a variable.

24. Explain about MODEL

This directive provides short cuts in defining segments. It initializes memory model before defining any segment. The memory model can be SMALL, MEDIUM,

COMPACT or LARGE.



25 Explain PROC & ENDP

PROC directive defines the procedures in the program. The procedure name must be unique. After PROC the term NEAR or FAR are used to specify the type of procedure. Example FACT PROC FAR. ENDP is used along with PROC and defines the end of the procedure.

26. Explain SEGMENT & ENDS

An assembly program in .EXE format consists of one or more segments. The starts of these segments are defined by SEGMENT and the end of the segment is indicated by ENDS directive. Format Name SEGMENT

27. Explain TITLE & TYPE

The TITLE directive helps to control the format of a listing of an assembled program. It causes a title for the program to print on line 2 of each page of the program listing. Maximum 60 characters are allowed. Format TITLE text. TYPE operator tells the assembler to determine the type of specified variable in bytes. For bytes the assembler gives a value 1, for word 2 & double word 4.

 28 Define SOP

The segment override prefix allows the programmer to deviate from the default

segment

Eg     :         MOV CS: [BX] , AL

 29 Define variable.

A variable is an identifier that is associated with the first byte of data item. In assembly language statement: COUNT DB 20H, COUNT is the variable.

 30. What are procedures?

Procedures are a group of instructions stored as a separate program in memory and itis called from the main program whenever required. The type of procedure depends on where the procedures are stored in memory. If it is in the same code segment as that of the main program then it is a near procedure otherwise it is a far procedure.

 31. Explain the linking process.

A linker is a program used to join together several object files into one large object file. The linker produces a link file which contains the binary codes for all the combined modules. It also produces a link map which contains the address information about the link files. The linker does not assign

Absolute addresses but only relative address starting from zero, so the programs are relocatable & can be put anywhere in memory to be run.

 32, Define variable

A variable is an identifier that is associated with the first byte of data item. In assembly language statement: COUNT DB 20H, COUNT is the variable.

 33. What are procedures?

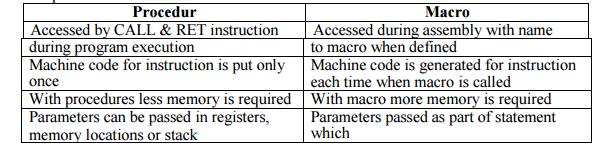
Procedures are a group of instructions stored as a separate program in memory and it is called from the main program whenever required. The type of procedure depends on where the procedures are stored in memory. If it is in the same code segment as that of the main program then it is a near procedure otherwise it is a far procedure.

 34. Explain the linking process.

A linker is a program used to join together several object files into one large object

file. The linker produces a link file which contains the binary codes for all the combined modules. It also produces a link map which contains the address information about the link files. The linker does not assign absolute addresses but only relative address starting from zero, so the programs are relocatable & can be put anywhere in memory to be run.

35. Compare Procedure & Macro.



**Procedur**

Accessed by CALL & RET instruction

during program execution

Machine code for instruction is put only once

With procedures less memory is required

Parameters can be passed in registers, memory locations or stack

**Macro**

Accessed during assembly with name to macro when defined

Machine code is generated for instruction each time when macro is called

With macro more memory is required

Parameters passed as part of statement which

36.            What is the maximum memory size that can be addressed by 8086?

In 8086, an memory location is addressed by 20 bit address and the address bus is 20 bit address and the address bus is 20 bits. So it can address up to one mega byte (2^20) of memory space.