1. Explain the execution steps of 32 bit and 64 bit.

* The difference between 32-bit and 64-bit systems lies in the way they process data. 32-bit systems utilize data in 32-bit pieces, while 64-bit systems utilize data in 64-bit pieces. In general, the more data that can be processed at once, the faster the system can operate. [There are several other advantages to a 64-bit system, most practically the ability to use significantly greater amounts of physical memory (more than the 4 GB allowed by a 32-bit machine)](https://www.lifewire.com/32-bit-64-bit-2624554)

1. Give the explanation of section .data, section .text, section .bss

* In assembly language, a program can be divided into three sections: The data section, The bss section, and The text section. The data section is used for declaring initialized data or constants. This data does not change at runtime. You can declare various constant values, file names, or buffer size, etc., in this section. The bss section is used for declaring variables. [The text section is used for keeping the actual code](https://www.tutorialspoint.com/assembly_programming/assembly_basic_syntax.htm)

1. Why we define “2”? in macro disp\_msg 2

Self answer

1. Why we define %1 and %2 in macro?

* In a macro, %1 and %2 are used to refer to the first and second arguments passed to the macro respectively. [For example, if you have a macro #define SUM(X,Y) (X+Y), when you call SUM(3,4), X is replaced with 3 and Y is replaced with](https://stackoverflow.com/questions/47615345/passing-macro-arguments-to-macro-function)

1. Explain the type of directives?

* In NASM, directives are commands that give the assembler instructions on how to assemble the source code. They come in two types: user-level directives and primitive directives. User-level directives are implemented as macros that call the primitive forms of the directives. Primitive directives are the basic building blocks of the assembler’s functionality.

For example, the BITS directive is a user-level directive that sets the current target processor mode. It is implemented as a macro that calls the primitive directive \_\_BITS, which actually sets the target processor mode.

Another example is the SECTION directive, which is used to specify the section in which subsequent code or data should be placed. It is implemented as a macro that calls the primitive directive \_\_SECTION, which actually sets the current section.

In general, it is recommended to use the user-level forms of directives because they provide a higher level of abstraction and are easier to use. However, in some cases, it may be necessary to use primitive directives to achieve more fine-grained control over the assembly process.

1. What is difference bet ax,eax, rax register?

* The difference between ax, eax, and rax registers lies in their size and the architecture they belong to. rax is a 64-bit register introduced with the x86-64 extension of the x86 architecture. eax[is a 32-bit register that belongs to the x86 architectureax is a 16-bit register that also belongs to the x86 architecture](https://stackoverflow.com/questions/25455447/x86-64-registers-rax-eax-ax-al-overwriting-full-register-contents" \t "_blank)[**2**](https://stackoverflow.com/questions/25455447/x86-64-registers-rax-eax-ax-al-overwriting-full-register-contents). [The 32-bit and 64-bit registers overlap: for example, the bottom 32 bits of the 64-bit rax register are the 32-bit eax register](https://stackoverflow.com/questions/44972293/how-is-rax-different-from-eax)

1. What is Full Form of NASM?

* NASM stands for Netwide Assembler. It is an assembler and disassembler for the Intel x86 architecture.

1. Explain the string Operations.

* String operations are a set of instructions that operate on strings of characters or bytes. These instructions include MOVS (move string), LODS (load string), STOS (store string), CMPS (compare string), and SCAS (scan string). These instructions can be used with the REP, REPE/REPZ, and REPNE/REPNZ prefixes to repeat the operation for a specified number of times or until a condition is met.

1. What is meaning of resb , resw?

* RESB and RESW are NASM directives used to declare uninitialized data. RESB is used to reserve a specified number of bytes, while RESW is used to reserve a specified number of words (where a word is 2 bytes). For example, var1 resb 4 reserves 4 bytes of uninitialized data, while var2 resw 2 reserves 4 bytes (2 words) of uninitialized data.

1. Why we use global\_start?

* The global \_start directive is used in NASM assembly language to mark the \_start symbol as global so that its name is added to the object code. [This allows the linker (ld) to read the symbol in the object code and its value so it knows where to mark the entry point in the output executable](https://stackoverflow.com/questions/17898989/what-is-global-start-in-assembly-language)

1. Why we use macro?

* A macro is a sequence of instructions that can be invoked by name as a single instruction. Macros are used to simplify repetitive tasks by allowing the programmer to define a sequence of instructions once and then reuse it multiple times.

1. Difference between macros and procedure

* The main difference between a macro and a procedure is that a macro is expanded inline at each point where it is invoked, while a procedure is a separate block of code that is called by other code. This means that when a macro is invoked, the assembler generates code for the entire sequence of instructions defined by the macro, while when a procedure is called, the assembler generates a single call instruction that transfers control to the procedure’s code.

1. Explain Divide Instruction.

* The DIV instruction is used to perform unsigned integer division in x86 assembly language. It divides the contents of the accumulator register (AX, EAX, or RAX, depending on the operand size) by the specified operand and stores the quotient in the AL, AX, or EAX register and the remainder in the AH, DX, or EDX register.

1. Write DIV instruction syntax.

* The syntax for the DIV instruction is: DIV src. The src operand can be a register or a memory location. For example: DIV BX divides the contents of the accumulator register by the contents of the BX register.

1. Explain ADD instruction.

* The ADD instruction is used to perform integer addition in x86 assembly language. It adds the value of the second operand to the value of the first operand and stores the result in the first operand. The syntax for the ADD instruction is: ADD dest, src. Both operands can be registers or memory locations, but at least one operand must be a register. For example: ADD AX, BX adds the contents of the BX register to the contents of the AX register and stores the result in the AX register.

1. Explain 80386 Flag Register.

* The 80386 flag register is a 32-bit register that contains a collection of flags that indicate the status of the processor and the results of arithmetic and logical operations. Some of the flags in the 80386 flag register include the carry flag (CF), which indicates if an operation generated a carry or borrow; the parity flag (PF), which indicates if the least significant byte of the result contains an even number of 1 bits; the zero flag (ZF), which indicates if the result of an operation is zero; and the sign flag (SF), which indicates if the result of an operation is negative.

1. Explain SUB Instruction. With syntax.

* The SUB instruction is used to perform integer subtraction in x86 assembly language. It subtracts the value of the second operand from the value of the first operand and stores the result in the first operand. The syntax for the SUB instruction is: SUB dest, src. Both operands can be registers or memory locations, but at least one operand must be a register. For example: SUB AX, BX subtracts the contents of the BX register from the contents of the AX register and stores the result in the AX register.

1. What is DEC instruction?

* The DEC instruction is used to decrement an integer value by 1 in x86 assembly language. It subtracts 1 from the value of its operand and stores the result in the same operand. The syntax for the DEC instruction is: DEC dest. The dest operand can be a register or a memory location. For example: DEC AX decrements the contents of the AX register by 1.

1. Explain CMP instruction.

* The CMP instruction is used to perform a comparison between two operands in x86 assembly language. It subtracts the value of the second operand from the value of the first operand and sets the flags in the flag register based on the result, but does not store the result of the subtraction. The syntax for the CMP instruction is: CMP dest, src. Both operands can be registers or memory locations, but at least one operand must be a register. For example: CMP AX, BX compares the contents of the AX and BX registers by subtracting the contents of the BX register from the contents of the AX register and setting the flags accordingly.

1. Explain OR instruction.

* The OR instruction is used to perform a bitwise inclusive OR operation between two operands in x86 assembly language. It performs a bitwise OR operation on each pair of corresponding bits in the two operands and stores the result in the first operand. The syntax for the OR instruction is: OR dest, src. Both operands can be registers or memory locations, but at least one operand must be a register. For example: OR AX, BX performs a bitwise OR operation on the contents of the AX and BX registers and stores the result in the AX register.

1. Explain XOR instruction.

* The XOR instruction is used to perform a bitwise exclusive OR operation between two operands in x86 assembly language. It performs a bitwise XOR operation on each pair of corresponding bits in the two operands and stores the result in the first operand. The syntax for the XOR instruction is: XOR dest, src. Both operands can be registers or memory locations, but at least one operand must be a register. For example: XOR AX, BX performs a bitwise XOR operation on the contents of the AX and BX registers and stores the result in the AX register.

1. Explain Addressing Modes.

* Addressing modes are the ways in which operands can be specified in x86 assembly language. Some common addressing modes include:
* Immediate addressing: The operand is a constant value specified in the instruction itself. For example: MOV AX, 1 moves the immediate value 1 into the AX register.
* Register addressing: The operand is a value stored in a register. For example: MOV AX, BX moves the value stored in the BX register into the AX register.
* Direct addressing: The operand is a value stored in memory at a specific address. The address is specified in the instruction itself. For example: MOV AX, [0x1234] moves the value stored at memory address 0x1234 into the AX register.
* Register indirect addressing: The operand is a value stored in memory at an address specified by the contents of a register. For example: MOV AX, [BX] moves the value stored at the memory address specified by the contents of the BX register into the AX register.
* Base plus index addressing: The operand is a value stored in memory at an address calculated by adding a base address specified by the contents of one register to an offset specified by the contents of another register. For example: MOV AX, [BX+SI] moves the value stored at the memory address calculated by adding the contents of the BX and SI registers into the AX register.

1. Write Syntax of Addressing Modes.

* The syntax for specifying operands using different addressing modes varies depending on the mode being used. Here are some examples:
* Immediate addressing: operand
* Register addressing: register
* Direct addressing: [address]
* Register indirect addressing: [register]
* Base plus index addressing: [base\_register + index\_register]

1. Give Example of addressing Modes.

* Here are some examples of instructions that use different addressing modes:
* Immediate addressing: MOV AX, 1
* Register addressing: MOV AX, BX
* Direct addressing: MOV AX, [0x1234]
* Register indirect addressing: MOV AX, [BX]
* Base plus index addressing: MOV AX, [BX+SI]

1. What is meaning of “[ ]” bracket? (mov eax,[1234H])

* In x86 assembly language, square brackets [ ] are used to indicate that the value inside the brackets is a memory address and that the operand is the value stored at that address. For example, in the instruction MOV EAX, [1234H], the value inside the square brackets is 1234H, which is a memory address. The instruction moves the value stored at that memory address into the EAX register.

1. Which are conditional instruction?

* Conditional instructions are instructions that are executed only if a certain condition is met. In x86 assembly language, conditional instructions include conditional jump instructions such as JE (jump if equal), JNE (jump if not equal), JG (jump if greater), and JL (jump if less), as well as conditional loop instructions such as LOOPZ (loop while zero) and LOOPNZ (loop while not zero).

1. Which are Unconditional Instruction?

* Unconditional instructions are instructions that are always executed, regardless of any conditions. In x86 assembly language, unconditional instructions include arithmetic instructions such as ADD, SUB, and MUL, data transfer instructions such as MOV and PUSH, and control flow instructions such as JMP (unconditional jump) and CALL (call procedure).

1. Give example of Conditional JUMP instruction.

* CMP AX, BX ; compare the contents of the AX and BX registers

JE equal ; jump to the 'equal' label if the contents of AX and BX are equal

; code here is executed if AX and BX are not equal

JMP end ; jump to the 'end' label

equal:

; code here is executed if AX and BX are equal

end:

Copy

In this example, the JE instruction is a conditional jump instruction that jumps to the equal label if the CMP instruction sets the zero flag, indicating that the contents of the AX and BX registers are equal.

1. Give example of Unconditional JUMP instruction.

* Here is an example of an unconditional jump instruction in x86 assembly language:

JMP start ; jump to the 'start' label

; code here is skipped

start:

; code here is executed

Copy

In this example, the JMP instruction is an unconditional jump instruction that always jumps to the start label, regardless of any conditions.

1. What is command line argument?

* A command line argument is a value passed to a program when it is invoked from the command line. Command line arguments allow users to specify options or input data for the program. For example, if a program is invoked with the command myprogram input.txt output.txt, then input.txt and output.txt are command line arguments that specify the input and output files for the program.

1. Explain Rotate instruction.

* The rotate instructions in x86 assembly language are used to perform bitwise rotation on the bits of an operand. There are four rotate instructions: ROL (rotate left), ROR (rotate right), RCL (rotate left through carry), and RCR (rotate right through carry). These instructions shift the bits of the operand to the left or right, with the bits that are shifted off one end being reintroduced at the other end.

1. Give example of all Rotate instructions.

* Here are some examples of rotate instructions in x86 assembly language:
* ROL AX, 1: rotates the bits of the AX register to the left by 1 bit.
* ROR BX, 2: rotates the bits of the BX register to the right by 2 bits.
* RCL CX, 3: rotates the bits of the CX register to the left by 3 bits, through the carry flag.
* RCR DX, 4: rotates the bits of the DX register to the right by 4 bits, through the carry flag

1. Explain GDTR,LDTR,TR,MSW registers

* The GDTR (Global Descriptor Table Register) and LDTR (Local Descriptor Table Register) are special registers in x86 processors that hold information about the Global Descriptor Table (GDT) and Local Descriptor Table (LDT) respectively. [The GDTR contains the base address and limit of the GDT, while the LDTR contains a segment selector that points to an LDT descriptor in the GDT**1**](https://stackoverflow.com/questions/66144924/what-are-ldtr-and-gdtr).

The TR (Task Register) is a special register that holds information about the current task. It contains a segment selector that points to a Task State Segment (TSS) descriptor in the GDT.

The MSW (Machine Status Word) register is a 16-bit register that holds system flags and control information. It is part of the CR0 control register in x86 processors.

1. Explain features of 80386.

* The 80386 microprocessor, introduced by Intel in 1985, was the first 32-bit microprocessor in the x86 family. Some of its features include:
* [32-bit data bus and address bus, allowing it to address up to 4 GB of physical memory**1**](https://electronicsdesk.com/80386-microprocessor.html).
* [Support for virtual memory addressability of up to 64 TB**1**](https://electronicsdesk.com/80386-microprocessor.html).
* [Support for a variety of operating clock frequencies, including 16 MHz, 20 MHz, 25 MHz, and 33 MHz**1**](https://electronicsdesk.com/80386-microprocessor.html).
* [Three-stage pipeline architecture: fetch, decode and execute**1**](https://electronicsdesk.com/80386-microprocessor.html).
* Three operating modes: real mode, protected mode and virtual mode. The protected mode was extended from the 80286 to allow the 80386 to address up to 4 GB of memory. [With the addition of segmented addressing system, it can expand up to 64 terabytes of virtual memory**2**](https://en.wikipedia.org/wiki/I386).

1. Explain flag registers of 8086 and 80386.

* The flag registers in both the 8086 and 80386 microprocessors are used to indicate the status of the processor and the results of arithmetic and logical operations. The flag register in the 8086 is a 16-bit register called the FLAGS register, while the flag register in the 80386 is a 32-bit register called the EFLAGS register.

Both registers contain a collection of flags that serve similar purposes. Some common flags include the carry flag (CF), which indicates if an operation generated a carry or borrow; the parity flag (PF), which indicates if the least significant byte of the result contains an even number of 1 bits; the zero flag (ZF), which indicates if the result of an operation is zero; and the sign flag (SF), which indicates if the result of an operation is negative.

1. Explain memory size of 8086 and 80386.

* The memory size of a microprocessor refers to the amount of memory it can address. The 8086 microprocessor has a 20-bit address bus, allowing it to address up to 2^20 bytes (1 MB) of memory. [The 80386 microprocessor has a 32-bit address bus, allowing it to address up to 2^32 bytes (4 GB) of physical memory](https://www.eeeguide.com/features-of-80186-80286-80386-and-80486-microprocessor/)

1. What is Protected Mode?

* Protected mode is an operating mode of the 80386 microprocessor that allows system software to use features such as virtual memory, paging and safe multi-tasking. In protected mode, the 80386 can address up to 4 GB of memory and provides hardware support for memory protection and multitasking.

1. What is paging mechanism?

* Paging is a memory management mechanism supported by the 80386 microprocessor in protected mode. It allows the physical memory of a computer to be used more efficiently by dividing it into smaller units called pages. When a program needs to access data that is not currently in physical memory, the operating system can use paging to bring the required data into memory from secondary storage.

1. Explain control registers.

* The 80386 microprocessor has several control registers that are used to control and configure various aspects of its operation. Some common control registers include the CR0 register, which controls basic system functions such as paging and protected mode; the CR2 register, which contains the linear address that caused a page fault; and the CR3 register, which contains the base address of the page directory used for paging.

1. Explain descriptors, selector.

* In the 80386 microprocessor, descriptors and selectors are used to manage memory segmentation in protected mode. A descriptor is a data structure that defines the characteristics of a segment, such as its base address, size, and access rights. A selector is a 16-bit value that identifies a descriptor in a descriptor table. The 80386 uses selectors to locate descriptors in the Global Descriptor Table (GDT) or Local Descriptor Table (LDT) when performing memory accesses.

1. Explain data bus size and address bus size of 8086.

* The 8086 microprocessor has a 16-bit data bus and a 20-bit address bus. This means that it can transfer 16 bits of data at a time and can address up to 2^20 bytes (1 MB) of memory.

1. Explain data bus size and address bus size of 80386.

* The 80386 microprocessor has a 32-bit data bus and a 32-bit address bus. This means that it can transfer 32 bits of data at a time and can address up to 2^32 bytes (4 GB) of memory.

1. Which are assembly language directives? (DB, DW, DD, DQ and DT)

* DB, DW, DD, DQ, and DT are assembly language directives used to define and allocate storage space for data in x86 assembly language. DB (Define Byte) is used to allocate storage space for one or more bytes of data. DW (Define Word) is used to allocate storage space for one or more words (2 bytes) of data. DD (Define Doubleword) is used to allocate storage space for one or more doublewords (4 bytes) of data. DQ (Define Quadword) is used to allocate storage space for one or more quadwords (8 bytes) of data. DT (Define Ten Bytes) is used to allocate storage space for one or more ten-byte values.

1. How to convert protected mode to real mode? Which bit is set?

* To switch from protected mode to real mode in the 80386 microprocessor, the PE (Protection Enable) bit in the CR0 control register must be cleared. This can be done by executing the following instruction: MOV CR0, EAX. Before executing this instruction, the EAX register must be loaded with the current value of the CR0 register, with the PE bit cleared.

1. Which control register is used for paging?

* The CR3 control register is used for paging in the 80386 microprocessor. It contains the base address of the page directory used for paging.

1. Explain size of GDTR and IDTR and LDT register.

* The GDTR (Global Descriptor Table Register) and IDTR (Interrupt Descriptor Table Register) are special registers in the 80386 microprocessor that hold information about the Global Descriptor Table (GDT) and Interrupt Descriptor Table (IDT) respectively. Both registers are 48 bits in size and consist of a 16-bit limit field and a 32-bit base address field. The limit field specifies the size of the table in bytes, while the base address field specifies the linear address of the first byte of the table.

The LDTR (Local Descriptor Table Register) is another special register in the 80386 microprocessor that holds information about the Local Descriptor Table (LDT). It is a 16-bit register that contains a segment selector that points to an LDT descriptor in the GDT.

1. Write “1 to 9” hexadecimal numbers into ASCII value.

* Here are the ASCII values for the hexadecimal numbers “1” to “9”:
* “1” = 0x31
* “2” = 0x32
* “3” = 0x33
* “4” = 0x34
* “5” = 0x35
* “6” = 0x36
* “7” = 0x37
* “8” = 0x38
* “9” = 0x39

1. Write “A to F” hexadecimal numbers into ASCII value.

* Here are the ASCII values for the hexadecimal numbers “A” to “F”:
* “A” = 0x41
* “B” = 0x42
* “C” = 0x43
* “D” = 0x44
* “E” = 0x45
* “F” = 0x46

1. Why we convert HEX to ASCII?

* Converting hexadecimal values to ASCII is useful when you want to represent hexadecimal data as human-readable text. For example, if you have a byte of data with the value 0xAB, you can convert it to ASCII to represent it as the string "AB".

1. Why we convert ASCII to HEX?

* Converting ASCII values to hexadecimal is useful when you want to represent text data as hexadecimal values for storage or transmission. For example, if you have a string "AB" that you want to store as a byte of data, you can convert it to hexadecimal to represent it as the value 0xAB.

1. Which are disadvantage of 8086? And advantage of 80386?

* The 8086 microprocessor has some disadvantages compared to the 80386 microprocessor. [One disadvantage of the 8086 is its segmented memory architecture, which can make it more difficult to program than the 80386, which supports a flat memory model in protected mode**1**](https://www.geeksforgeeks.org/architecture-of-8086/). Another disadvantage of the 8086 is its lack of support for virtual memory and paging, which are features supported by the 80386 in protected mode.

The 80386 microprocessor has several advantages over the 8086 microprocessor. One advantage is its larger data bus and address bus sizes, which allow it to transfer more data at a time and address more memory. Another advantage is its support for protected mode, which provides hardware support for memory protection and multitasking

1. Explain in detail Data Transfer Instructions

* Data transfer instructions are used to move data between registers, memory, and I/O devices in x86 assembly language. Some common data transfer instructions include MOV, which moves data from one location to another; PUSH, which pushes data onto the stack; POP, which pops data from the stack; IN, which reads data from an I/O port; and OUT, which writes data to an I/O port.

1. Explain in detail Arithmetic Instructions

* Arithmetic instructions are used to perform arithmetic operations such as addition, subtraction, multiplication, and division in x86 assembly language. Some common arithmetic instructions include ADD, which adds two operands; SUB, which subtracts one operand from another; MUL, which multiplies two operands; DIV, which divides one operand by another; and INC and DEC, which increment and decrement an operand by 1, respectively.

1. Explain in detail Bit Manipulation Instructions

* Bit manipulation instructions are used to perform bitwise operations on the bits of an operand in x86 assembly language. Some common bit manipulation instructions include AND, which performs a bitwise AND operation on two operands; OR, which performs a bitwise OR operation on two operands; XOR, which performs a bitwise exclusive OR operation on two operands; NOT, which performs a bitwise NOT operation on an operand; and SHL and SHR, which shift the bits of an operand to the left or right.

1. Explain in detail Program Execution Transfer Instructions

* Program execution transfer instructions are used to control the flow of execution in x86 assembly language programs. Some common program execution transfer instructions include JMP, which performs an unconditional jump to a specified location; conditional jump instructions such as JE (jump if equal) and JNE (jump if not equal), which perform conditional jumps based on the state of the flags in the flag register; CALL, which calls a procedure; and RET, which returns from a procedure.

1. Explain in detail String Instructions

* String instructions are used to perform operations on strings of data in x86 assembly language. Some common string instructions include MOVS (move string), which moves a string of data from one location to another; LODS (load string), which loads a string of data from memory into a register; STOS (store string), which stores a string of data from a register into memory; CMPS (compare string), which compares two strings of data; and SCAS (scan string), which scans a string of data for a specified value.

1. Explain in detail Processor Control Instructions

* Processor control instructions are used to control the operation of the processor in x86 assembly language. Some common processor control instructions include HLT, which halts the processor until an interrupt occurs; CLI, which clears the interrupt flag, disabling interrupts; STI, which sets the interrupt flag, enabling interrupts; CLC, which clears the carry flag; and STC, which sets the carry flag.

1. Explain size of page, page table, page directory.

* In the 80386 microprocessor, a page is a fixed-size block of memory used for paging. The size of a page is 4 KB. A page table is a data structure that maps virtual memory addresses to physical memory addresses. Each entry in a page table corresponds to a page of virtual memory and contains the physical address of the page in memory. A page directory is another data structure that contains pointers to page tables. Each entry in a page directory corresponds to a range of virtual memory addresses and contains the physical address of the page table that maps those addresses.

1. What is difference between multitasking and multithreading?

* Multitasking and multithreading are two techniques used in operating systems to manage multiple processes and tasks. The main difference between multitasking and multithreading is that multitasking involves running multiple independent processes or tasks concurrently, while multithreading involves dividing a single process into multiple threads that can execute concurrently[**1**](https://www.geeksforgeeks.org/difference-between-multi-tasking-and-multi-threading/).

Multitasking allows the CPU to perform multiple tasks simultaneously by rapidly switching between them. This gives the illusion that all tasks are running at the same time. In contrast, multithreading allows the CPU to execute multiple threads of the same process simultaneously. Threads share the same memory space and resources of the parent process, allowing them to communicate and synchronize data easily.

1. Explain in detail Privilege Levels.

* Privilege levels are used in x86 processors to provide a mechanism for protecting system resources and data from unauthorized access. In the 80386 microprocessor, there are four privilege levels, numbered from 0 to 3. Level 0 is the most privileged level, while level 3 is the least privileged level.

Each segment descriptor in the Global Descriptor Table (GDT) or Local Descriptor Table (LDT) has a privilege level associated with it, called the Descriptor Privilege Level (DPL). When a program attempts to access a segment, the processor checks the privilege level of the program against the DPL of the segment descriptor. If the program’s privilege level is less than or equal to the DPL, access is allowed; otherwise, access is denied.

1. What is TLB Buffer?

* The Translation Lookaside Buffer (TLB) is a cache used by x86 processors to speed up virtual-to-physical address translation during paging. When a program accesses a virtual memory address, the processor checks the TLB to see if it contains a cached translation for that address. If it does, the processor uses the cached translation to quickly determine the physical memory address; otherwise, it must perform a slower page table lookup to determine the physical address.

**Steps for execution :-**

If disp.asm is our program file then steps for execution are as below

:-$ nasm –f elf64 -o disp.o disp.asm

This command creates an object file disp.o with elf32 file format.

$ ld –o disp disp.o

This command creates an executable with a name hello from disp.o object file

$ ./disp

This command executes disp program