Microprocessor

What is a directive? List and explain any four directives.

In microprocessor programming, a directive is a type of command that provides instructions to the assembler to perform certain tasks. These tasks can include defining constants, reserving memory space, specifying program start addresses, and more.

Here are four common directives in microprocessor programming:

1. ORG (origin): The ORG directive is used to specify the memory address at which the assembler should begin assembling the program. For example, if you use ORG 100h, the assembler will begin assembling the program at address 100h.
2. EQU (equal): The EQU directive is used to define a constant value that can be used throughout the program. For example, if you use MYCONST EQU 5, the assembler will replace every occurrence of MYCONST with the value 5.
3. DB (define byte): The DB directive is used to define a byte (8 bits) of memory. For example, if you use MYBYTE DB 1, the assembler will reserve one byte of memory and initialize it to the value 1.
4. DW (define word): The DW directive is used to define a word (16 bits) of memory. For example, if you use MYWORD DW 1234h, the assembler will reserve two bytes of memory and initialize them to the value 1234h (which is 00001234 in binary).

These are just a few examples of directives in microprocessor programming. There are many more directives available depending on the specific microprocessor and assembler being used.

Explain the programmer’s model of Intel 80386.

The Intel 80386 is a 32-bit microprocessor and its programmer's model is a representation of the processor's architecture that helps the programmer understand how to write code that can interact with the processor. The programmer's model of the 80386 consists of several components that are organized in a hierarchical manner.

Here are the major components of the programmer's model of the Intel 80386:

1. Registers: The 80386 has eight 32-bit general-purpose registers that are used for arithmetic and logic operations. These registers are EAX, EBX, ECX, EDX, EBP, ESI, EDI, and ESP. EAX and EBX are used as accumulator and base registers respectively, while ECX and EDX are used for counter and data operations. EBP is used as the base pointer, ESI and EDI are used as source and destination pointers, and ESP is used as the stack pointer.
2. Flags: The 80386 has a set of 32-bit flags that are used to indicate the results of arithmetic and logic operations. These flags include the carry flag, the zero flag, the sign flag, the overflow flag, and others.
3. Memory Management Unit (MMU): The MMU is responsible for translating virtual addresses to physical addresses. The 80386 uses a hierarchical paging scheme that allows for up to 4GB of virtual memory.
4. Control Registers: The 80386 has several control registers that are used to control various aspects of the processor's operation. These include the CR0 register, which controls the processor's operating mode (real mode or protected mode), and the CR3 register, which points to the base of the page directory.
5. Segment Registers: The 80386 has six segment registers that are used to specify the base address of various memory segments. These registers are CS (code segment), DS (data segment), SS (stack segment), ES (extra segment), FS (further segment), and GS (general segment).

Overall, the programmer's model of the Intel 80386 provides a comprehensive view of the processor's architecture, which is essential for writing code that interacts with the processor effectively.

> ## List and explain the operating modes of Intel 80386. Write at least three differentiating points.

The Intel 80386 has two operating modes: real mode and protected mode.

1. Real mode: In real mode, the processor operates as a 16-bit processor and is backward compatible with earlier x86 processors. Real mode allows direct access to the processor's physical memory, which is limited to 1 MB. Real mode

uses segment registers to specify the base address of different memory segments.

2. Protected mode: In protected mode, the processor operates as a 32-bit processor and provides memory protection and multitasking capabilities. Protected mode allows access to up to 4 GB of physical memory and uses a hierarchical paging scheme to translate virtual addresses to physical addresses. Protected mode also provides several levels of privilege (ring 0-3) to control access to system resources.

Here are three differentiating points between real mode and protected mode:

1. Addressing: In real mode, memory is addressed using 16-bit segment:offset addressing. The maximum addressable memory is limited to 1 MB. In protected mode, memory is addressed using 32-bit virtual addresses that are translated to physical addresses using a hierarchical paging scheme. The maximum addressable memory is 4 GB.
2. Privilege Levels: In real mode, there are no privilege levels. In protected mode, there are four privilege levels (rings) ranging from ring 0 (most privileged) to ring 3 (least privileged). This allows for fine-grained control of access to system resources.
3. Memory Protection: In real mode, there is no memory protection, which means that any program can access any part of memory. In protected mode, memory protection is enforced, which means that programs can only access memory that they have been granted access to. This improves system stability and security.

Overall, the two operating modes of the Intel 80386 provide different trade-offs between backward compatibility, memory addressing, memory protection, and multitasking capabilities.

> ## Illustrate the following instructions. 1. DIV 2. MUL 3. CMP 4. CALL 5. PUSHA 6. CBW

1. DIV: The DIV instruction is used to perform unsigned division of the 16-bit or 32-bit operand by the contents of the AX register. Here's an example of how the DIV instruction can be used:

```
MOV AX, 0124h    ; Move the value 0124h into the AX register
MOV BX, 0005h    ; Move the value 0005h into the BX register
```

```
DIV BX           ; Divide the value in AX by BX
```

After the DIV instruction is executed, the quotient will be stored in the AX register and the remainder will be stored in the DX register.

1. MUL: The MUL instruction is used to perform unsigned multiplication of the 16-bit or 32-bit operand by the contents of the AX register. Here's an example of how the MUL instruction can be used:

```
MOV AX, 0124h    ; Move the value 0124h into the AX register
MOV BX, 0005h    ; Move the value 0005h into the BX register
MUL BX           ; Multiply the value in AX by BX
```

After the MUL instruction is executed, the product will be stored in the AX register.

1. CMP: The CMP instruction is used to compare two operands and set the appropriate flags based on the result of the comparison. Here's an example of how the CMP instruction can be used:

```
MOV AX, 0124h    ; Move the value 0124h into the AX register
MOV BX, 0125h    ; Move the value 0125h into the BX register
CMP AX, BX       ; Compare the value in AX with the value in BX
```

After the CMP instruction is executed, the flags will be set based on whether the value in AX is less than, equal to, or greater than the value in BX.

1. CALL: The CALL instruction is used to call a subroutine. Here's an example of how the CALL instruction can be used:

```
CALL MySubroutine ; Call the subroutine named MySubroutine
```

After the CALL instruction is executed, the address of the next instruction is pushed onto the stack and the program jumps to the first instruction of the subroutine.

1. PUSHA: The PUSHA instruction is used to push all of the 16-bit or 32-bit general-purpose registers onto the stack. Here's an example of how the PUSHA instruction can be used:

```
PUSHA            ; Push all of the general-purpose registers onto the stack
```

After the PUSHA instruction is executed, the values of all of the general-purpose registers are pushed onto the stack in the order AX, CX, DX, BX, SP, BP, SI, and DI.

1. CBW: The CBW instruction is used to sign-extend the value in the AL register to the AX register. Here's an example of how the CBW instruction can be used:

```
MOV AL, 80h      ; Move the value 80h into the AL register
CBW              ; Sign-extend the value in AL to the AX register
```

After the CBW instruction is executed, the value in AL will be sign-extended to the value in AX.

List and explain the addressing modes of Intel 80386.

Intel 80386 microprocessor supports several addressing modes that enable efficient use of memory and simplify the process of programming. The following are the addressing modes supported by the Intel 80386:

1. Register Addressing Mode: In this mode, the operand is located in one of the general-purpose registers, such as AX, BX, CX, DX, SI, DI, or BP. For example, MOV AX, BX instruction moves the contents of the BX register into the AX register.
2. Immediate Addressing Mode: In this mode, the operand is a constant value that is part of the instruction itself. For example, MOV AX, 1234h instruction moves the value 1234h into the AX register.
3. Direct Addressing Mode: In this mode, the operand is the memory address of the data. For example, MOV AX, \[1234h\] instruction moves the contents of the memory location whose address is 1234h into the AX register.
4. Indirect Addressing Mode: In this mode, the operand is the memory address that points to the memory location containing the data. For example, MOV AX, \[BX\] instruction moves the contents of the memory location pointed to by the BX register into the AX register.
5. Indexed Addressing Mode: In this mode, the operand is the sum of the memory address and the index register, which can be any of the following: SI, DI, BP, or BX. For example, MOV AX, \[BX+SI\] instruction moves the contents of the

memory location whose address is the sum of the BX and SI registers into the AX register.

6. Based Addressing Mode: In this mode, the operand is the sum of the memory address and the base register, which can be any of the following: BX, BP, SI, or DI. For example, MOV AX, \[BX+1234h\] instruction moves the contents of the memory location whose address is the sum of the BX register and 1234h into the AX register.
7. Relative Addressing Mode: In this mode, the operand is the memory address that is obtained by adding the displacement to the instruction pointer (IP) register. For example, JMP Label instruction jumps to the memory location whose address is obtained by adding the displacement to the IP register.

These addressing modes offer a great deal of flexibility to the programmer and allow the efficient use of memory by the processor. By using these addressing modes, the programmer can reduce the amount of code required for performing various tasks and optimize the performance of the microprocessor.

> ## What are Macro and Procedure? Differentiate both with respect to control transfer, parameters, syntax and example.

Macro and Procedure are two important concepts in programming that are used to simplify code development, reduce code redundancy and improve code maintainability. Though both Macro and Procedure perform a similar function, they differ in their implementation and usage.

1. Macro: A macro is a sequence of instructions that is defined once and can be used multiple times in a program. It is a powerful tool used in assembly language programming for performing repetitive tasks. Macros are expanded inline, which means that they are copied directly into the code at the point where they are called, thus saving execution time.

Control transfer: In macro, control transfer is done using the JMP (jump) instruction. Parameters: Macros use symbolic constants or variables as parameters. Syntax: Macros are defined using the .MACRO and .ENDM directives. Example:

```
.MACRO ADDITION X,Y
    MOV AX, X
    ADD AX, Y
.ENDM
```

In the above example, the macro ADDITION adds two numbers X and Y and stores the result in the AX register.

1. Procedure: A procedure is a set of instructions that are defined once and can be called multiple times in a program. It is a way to modularize the code, making it easier to understand and maintain. Procedures are stored in memory as a separate entity and are executed whenever they are called.

Control transfer: In procedure, control transfer is done using the CALL and RET instructions. Parameters: Procedures use parameters passed through the stack or registers. Syntax: Procedures are defined using the PROC and ENDP directives. Example:

```
MyProc PROC x:WORD, y:WORD
    MOV AX, x
    ADD AX, y
    RET
MyProc ENDP
```

In the above example, the procedure MyProc adds two numbers x and y and returns the result.

Difference between Macro and Procedure:

Control transfer: Macros use the JMP (jump) instruction for control transfer, whereas procedures use CALL and RET instructions.

Parameters: Macros use symbolic constants or variables as parameters, whereas procedures use parameters passed through the stack or registers.

Syntax: Macros are defined using the .MACRO and .ENDM directives, whereas procedures are defined using the PROC and ENDP directives.

Inline expansion: Macros are expanded inline, which means they are copied directly into the code at the point where they are called, whereas procedures are stored in memory as a separate entity and are executed whenever they are called.

In summary, both macros and procedures are important constructs in programming that help to reduce code redundancy and improve code maintainability. While macros are more suitable for performing repetitive tasks, procedures are more suitable for modularizing the code and making it more organized.

> Explain the following bits of EFLAG registers of Intel 80386. 1. CF 2. ZF 3. DF 4. SF 5. IF 6. VM

EFLAGS (Extended Flags) is a 32-bit register in Intel 80386 processor that contains information about the current state of the processor. It consists of various individual bit flags that are used to store the results of arithmetic and logical operations performed by the processor. Here are the explanations of some of the important bits of the EFLAGS register:

1. CF (Carry Flag): CF flag is set when an arithmetic or logical operation produces a carry or borrow out of the most significant bit of the result. It is used to indicate overflow or underflow in unsigned arithmetic operations. For example, when we add two 8-bit numbers 255 and 1, the result is 0 with the carry flag set to 1.
2. ZF (Zero Flag): ZF flag is set when the result of an arithmetic or logical operation is zero. It is used to indicate equality or inequality in comparisons. For example, when we subtract two equal numbers, the result is 0 with the zero flag set to 1.
3. DF (Direction Flag): DF flag is used to control the direction of string operations such as MOVS (move string) and CMPS (compare string). When DF flag is set to 0, string operations are performed from the lower to higher memory address, and when DF flag is set to 1, string operations are performed from higher to lower memory address.
4. SF (Sign Flag): SF flag is set when the result of an arithmetic or logical operation has the most significant bit set to 1. It is used to indicate whether the result of the operation is positive or negative. For example, when we subtract a larger number from a smaller number, the result is negative with the sign flag set to 1.
5. IF (Interrupt Flag): IF flag is used to enable or disable hardware interrupts. When IF flag is set to 1, interrupts are enabled and when IF flag is set to 0, interrupts are disabled.
6. VM (Virtual Mode): VM flag is used to indicate whether the processor is in virtual 8086 mode or protected mode. When VM flag is set to 1, the processor is in virtual 8086 mode and when VM flag is set to 0, the processor is in protected mode.

In summary, the EFLAGS register contains various bit flags that are used to store the results of arithmetic and logical operations performed by the processor. These flags are used to indicate various conditions such as carry, zero, direction, sign, interrupt, and virtual mode.

> Explain the memory organization of the Intel 80386 processor with the appropriate diagram.

The Intel 80386 is a 32-bit processor, which means it can address up to 2^32 (4GB) of memory. The memory organization of the 80386 is divided into two main categories: physical memory and virtual memory.

1. Physical Memory: Physical memory is the actual memory present in the system, which is divided into several regions, as shown in the diagram below:

a. Conventional Memory: This is the first 1MB of memory in the system, which is divided into several regions such as BIOS, video memory, and system memory.

b. Extended Memory: This is the memory beyond the first 1MB of physical memory, which is used for storing applications and data.

c. Reserved Memory: This is the memory reserved for system use, which is not accessible to applications.

1. Virtual Memory: Virtual memory is a technique used by the 80386 to extend the available memory beyond the physical memory limit of the system. The virtual memory organization is shown in the diagram below:

a. Logical Address Space: This is the address space seen by applications, which is divided into several segments such as code, data, and stack.

b. Page Table: This is a data structure used by the 80386 to map logical addresses to physical addresses.

c. Physical Address Space: This is the actual physical memory in the system, which is accessed using the page table.

d. Disk Storage: This is the secondary storage used by the 80386 to store pages of memory that are not currently in use.

The virtual memory organization allows applications to access more memory than the physical memory limit of the system. When an application requests memory that is not currently present in physical memory, the 80386 swaps out some of the least recently used pages to disk and loads the requested page from disk into physical memory. This technique allows applications to use more memory than the physical memory limit of the system.

Define the following. 1. Logical Address 2. Linear address 3. Physical Address

1. Logical Address: A logical address is an address generated by a program that is used to access a specific memory location. It is also known as a virtual address because it is not a physical address in the memory. The logical address is translated to a physical address by the memory management unit (MMU) before it is used to access the memory.
2. Linear Address: A linear address is an address that is generated by combining the logical address with a base address. The base address is provided by the operating system and is used to map the logical address to a physical address. The linear address is also known as a virtual address, but it is different from the logical address because it is a direct representation of the memory location.
3. Physical Address: A physical address is the actual location of data in the memory. It is a binary number that uniquely identifies the location of data in the memory. The physical address is generated by the memory management unit (MMU) by translating the linear address to a physical address. The physical address is used by the memory controller to access the data stored in the memory.

In summary, a logical address is an address generated by a program, a linear address is a virtual address that is mapped to a physical address, and a physical address is the actual location of data in the memory.

List and explain any four-flag control instructions.

The Intel 80386 processor provides several instructions to control the value of the flags in the EFLAGS register. Here are four-flag control instructions:

1. CLC: This instruction clears the carry flag (CF) in the EFLAGS register. The carry flag is set when the result of an operation generates a carry or borrow. The CLC instruction is used to reset the carry flag to zero.
2. STC: This instruction sets the carry flag (CF) in the EFLAGS register. The STC instruction is used to set the carry flag to one.
3. CLD: This instruction clears the direction flag (DF) in the EFLAGS register. The direction flag determines the direction in which string operations are performed.

When the DF is set to one, the string operations are performed from high to low memory locations. The CLD instruction is used to reset the direction flag to zero.

4. STD: This instruction sets the direction flag (DF) in the EFLAGS register. The STD instruction is used to set the direction flag to one, which causes the string operations to be performed from high to low memory locations.

These instructions are used to control the value of flags in the EFLAGS register, which affect the behavior of the processor during operations. By controlling the value of these flags, the programmer can modify the behavior of the processor to suit the specific requirements of the program.

> ## What would be the content of AL register after execution of the following instructions? MOV AL, 23H MOV CL, 03H ROL AL, 02H

The content of the AL register after the execution of the following instructions:

1. MOV AL, 23H: This instruction moves the value 23H (35 decimal) into the AL register. So, the content of the AL register is 23H.
2. MOV CL, 03H: This instruction moves the value 03H (3 decimal) into the CL register. So, the content of the CL register is 03H.
3. ROL AL, 02H: This instruction rotates the bits in the AL register to the left by 2 positions. The carry flag (CF) is rotated into the least significant bit (LSB) of the AL register, and the most significant bit (MSB) is rotated into the carry flag. The value of CL is used as the rotation count.

The initial value of the AL register is 23H, which can be represented in binary as 0010 0011. After rotating the bits to the left by 2 positions, the LSB is shifted into the CF, and the value of AL becomes 80H (128 decimal), which can be represented in binary as 1000 0000.

Therefore, after the execution of these instructions, the content of the AL register is 80H (128 decimal).

> ## List the type of jump instructions. Explain any one instruction of each type.

The Intel 80386 processor provides several types of jump instructions, which are used to transfer control to different parts of a program. Here are some types of jump instructions:

1. Unconditional jump: This type of jump instruction unconditionally transfers control to a specified address. The JMP (jump) instruction is an example of an unconditional jump instruction. For example, the instruction "JMP Label" transfers control to the instruction at the memory location specified by the Label.
2. Conditional jump: This type of jump instruction transfers control to a specified address if a certain condition is met. There are several conditional jump instructions, such as JZ (jump if zero), JNZ (jump if not zero), JC (jump if carry), JNC (jump if not carry), etc. For example, the instruction "JZ Label" transfers control to the instruction at the memory location specified by the Label if the zero flag (ZF) in the EFLAGS register is set.
3. Near jump: This type of jump instruction transfers control to a memory location within the same code segment. The short jump instruction (JMP SHORT) is an example of a near jump instruction. For example, the instruction "JMP SHORT Label" transfers control to the instruction at the memory location specified by the Label, which must be within -128 to +127 bytes of the current instruction.
4. Far jump: This type of jump instruction transfers control to a memory location in a different code segment. The far jump instruction (JMP FAR) is an example of a far jump instruction. For example, the instruction "JMP FAR Pointer" transfers control to the instruction pointed to by the Pointer, which contains both a segment selector and an offset.

One example of a jump instruction is JZ (jump if zero), which transfers control to a specified address if the zero flag (ZF) in the EFLAGS register is set. Here's an example:

```
MOV AX, 0     ; Move the value 0 into the AX register
CMP AX, 0     ; Compare the value in AX with 0
JZ Label      ; Jump to Label if AX is zero
; Continue with the rest of the code here
Label:
; Code to execute if AX is zero
```

In this example, the CMP instruction compares the value in the AX register with 0, and sets the zero flag (ZF) in the EFLAGS register if they are equal. The JZ instruction then transfers control to the instruction at the memory location specified by the Label if the zero flag is set, which means that the value in the AX register is zero.