ASSIGNMENT 4

1. With a neatly labelled diagram explain 80386DX interfacing with 80387DX. Explain how 80387 manage to report error causing op-code and operand to 80386DX efficiently.

Ans: The 80386DX is a 32-bit microprocessor that was widely used in personal computers during the late 1980s and early 1990s. The 80387DX, on the other hand, is a numeric coprocessor specifically designed to handle floating-point arithmetic operations.

Interfacing between the 80386DX and the 80387DX involves connecting the two devices and enabling communication between them. Here's a general overview of the process:

1. Physical connection: The 80386DX and the 80387DX are typically connected using a dedicated bus known as the coprocessor bus. This bus allows data transfer and control signals between the microprocessor and the math coprocessor.

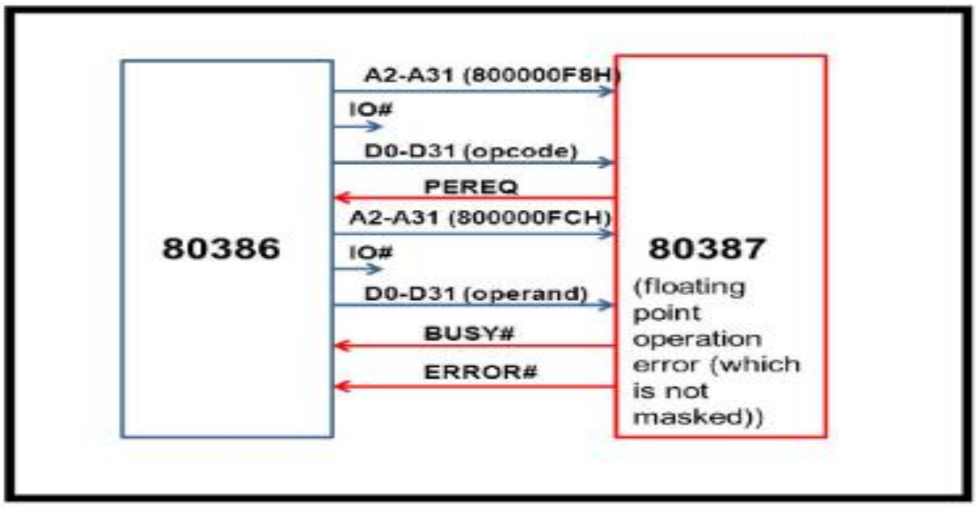
2. Initialization: The 80386DX needs to initialize the 80387DX before it can start using its capabilities. This is done by sending specific initialization commands and control signals to the math coprocessor.

3. Data transfer: Once initialized, data can be transferred between the microprocessor and the math coprocessor. The 80386DX sends instructions and data to the 80387DX, which performs the required arithmetic operations and returns the results back to the microprocessor.

4. Synchronization: To ensure proper synchronization between the two devices, synchronization signals are exchanged. These signals coordinate the timing and execution of instructions, ensuring that the microprocessor and math coprocessor work together seamlessly.

5. Error handling: The 80387DX also provides error handling capabilities. It can detect and handle various types of floating-point errors, such as overflow or underflow conditions.

.



The 80387 is a math coprocessor designed to work in conjunction with the 80386DX microprocessor, enhancing its computational capabilities. While the 80386DX handles general-purpose instructions, the 80387 focuses on executing floating-point arithmetic operations.

To efficiently report error-causing op-codes and operands to the 80386DX, the 80387 employs a mechanism called exception handling. Exceptions are events that occur during the execution of a program that disrupt the normal flow of instructions. When an error occurs in the 80387, it raises an exception and communicates the relevant information to the 80386DX.

The 80387 has a dedicated set of exception handling instructions that allow it to report errors to the 80386DX efficiently. These instructions include FERR (Floating-Point Error) and FISTTP (Floating-Point Store Integer with Truncation and Pop) among others.

When an error occurs during a floating-point operation, the 80387 sets the FERR flag in its control register and stores information about the error, such as the type of error and the offending op-code or operand. The 80387 then generates an exception, which interrupts the normal execution of the 80386DX.

The 80386DX, upon receiving the exception, switches its execution to an exception handling routine defined by the programmer. This routine can inspect the FERR flag and retrieve the error information stored by the 80387. By accessing this information, the 80386DX can determine the specific error that occurred, including the op-code or operand responsible for the error.

This mechanism allows the 80387 to efficiently communicate error information to the 80386DX without requiring the coprocessor to halt its operation entirely. The 80386DX can handle the exception and take appropriate action, such as displaying an error message, terminating the program, or attempting to recover from the error.

In summary, the 80387 efficiently reports error-causing op-codes and operands to the 80386DX by utilizing exception handling mechanisms. The coprocessor raises exceptions, sets flags, and stores error information, which the 80386DX can access and process to handle errors appropriately.

2. Write an 80387DX ALP to compute x=a*(b+c)/d.

Ans: ; Assuming a, b, c, and d are stored in memory

```
; a is at address A
; b is at address B
; c is at address C
; d is at address D
; Result x will be stored at address X

section .data
    A dd 2.5 ; Example value for a
    B dd 1.8 ; Example value for b
    C dd 3.2 ; Example value for c
    D dd 2.0 ; Example value for d
    X dd 0.0 ; Result variable x

section .text
    global _start
_start:
    fld dword [A] ; Load a into FPU stack
    fld dword [B] ; Load b into FPU stack
    fld dword [C] ; Load c into FPU stack
    fadd ; Add b and c
    fmulp ; Multiply a by the sum (b + c)

    fld dword [D] ; Load d into FPU stack
    fdivp ; Divide the product (a * (b + c)) by d

    fstp dword [X] ; Store the result in variable x

    ; Exit the program
    mov eax, 1 ; System call number (exit)
    xor ebx, ebx ; Exit code 0
    int 0x80 ; Call the kernel```
```

In this program, the values of a, b, c, d, and the result x are stored in memory. The program loads these values into the FPU stack using the `fld` instruction. It then performs the necessary calculations using FPU instructions such as `fadd`, `fmulp`, and `fdivp`. Finally, the result is stored back in memory at the address X using the `fstp` instruction.

3. Compare and contrast

a. Memory Mapped I/O and I/O Mapped I/O

b. String and Non-string based operations

Ans: a. Memory Mapped I/O and I/O Mapped I/O:

Memory Mapped I/O and I/O Mapped I/O are two different approaches used for interfacing Input/Output (I/O) devices with a computer system. Here's a comparison and contrast between the two:

Memory Mapped I/O:
- In memory mapped I/O, the I/O devices are treated as if they are memory locations.
- The same address bus and data bus used for accessing memory are also used for accessing I/O devices.
- The I/O devices are assigned specific memory addresses, and reading from or writing to those addresses accesses the corresponding I/O device.
- No specific instructions are required to interact with I/O devices, as they are accessed like regular memory locations.
- Memory-mapped I/O simplifies the programming interface but may limit the address space available for memory.

I/O Mapped I/O:

- In I/O mapped I/O, a separate I/O address space is used for accessing I/O devices.
- A separate address bus and set of instructions are used for I/O operations.
- The I/O devices are assigned unique I/O addresses, and specific instructions are used to communicate with them.
- I/O mapped I/O provides a clear distinction between memory and I/O operations but requires separate instructions and address decoding logic.
- I/O mapped I/O allows for a larger address space for memory, as it does not consume memory address space for I/O devices.

In summary, memory mapped I/O treats I/O devices as memory locations, using the same address and data buses, while I/O mapped I/O uses a separate address space and instructions for accessing I/O devices.

b. String and Non-string based operations:

String operations and non-string operations refer to different ways of manipulating data, particularly in programming or computer science contexts. Here's a comparison and contrast between the two:

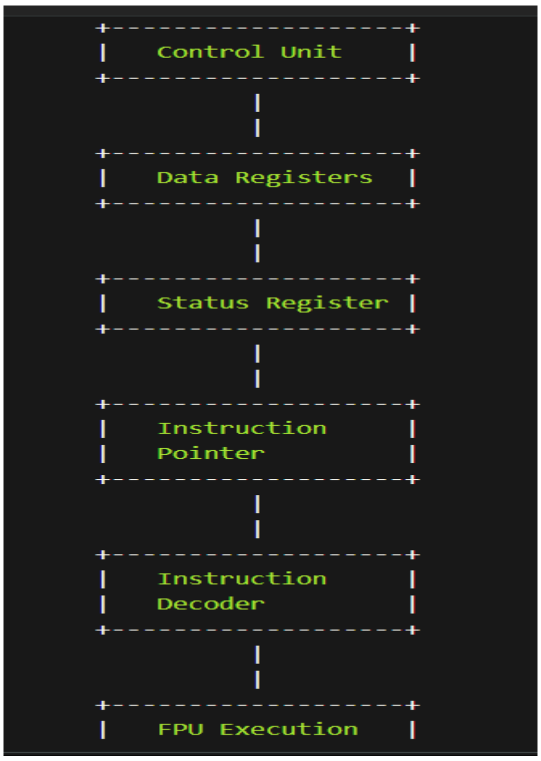
String-based operations:
- String-based operations involve manipulating or working with sequences of characters (strings).
- Examples of string-based operations include string concatenation, string comparison, searching for substrings, and string manipulation functions like length calculation, case conversion, and character extraction.
- String-based operations typically operate on entire strings or portions of strings, treating them as a single unit.
- String-based operations are commonly used in text processing, parsing, and handling textual data.

Non-string based operations:
- Non-string based operations involve working with data that is not organized as strings.
- These operations can include arithmetic calculations, logical operations, bitwise operations, array manipulations, and algorithmic computations.
- Non-string based operations typically work on individual data elements, such as numbers, Boolean values, or binary digits.
- Non-string based operations are more general-purpose and can be used in various domains such as numerical analysis, data processing, graphics, and scientific computing.

In summary, string-based operations focus on manipulating and processing sequences of characters, while non-string based operations encompass a broader range of data types and operations, typically involving numerical or logical computations.

4. With appropriate block diagram explain 80387DX math co processor register set.

Ans: The 80387DX math coprocessor, also known as the Intel 387, was a floating-point coprocessor designed to work with Intel's 80386 microprocessor. It provided enhanced mathematical capabilities to the 80386-based systems. The math coprocessor had its own set of registers and operated independently of the main processor. Here is a block diagram explaining the register set of the 80387DX math coprocessor:



```
+--------------------+
|   Control Unit     |
+--------------------+
          |
          |
+--------------------+
|   Data Registers   |
+--------------------+
          |
          |
+--------------------+
|   Status Register  |
+--------------------+
          |
          |
+--------------------+
|   Instruction      |
|   Pointer          |
+--------------------+
          |
          |
+--------------------+
|   Instruction      |
|   Decoder          |
+--------------------+
          |
          |
+--------------------+
|   FPU Execution    |
```

1. Control Unit: The control unit manages the overall operation of the math coprocessor. It receives instructions from the main processor and controls the execution of these instructions.

2. Data Registers: The data registers are used for storing operands and intermediate results during mathematical operations. The 80387DX had eight 80-bit data registers, labeled ST(0) through ST(7). These registers were capable of storing single-precision floating-point values.

3. Status Register: The status register holds various status flags that provide information about the result of arithmetic operations. These flags include the condition code flags, which indicate whether the result is zero, negative, or positive, and other control flags that affect the behavior of the coprocessor.

4. Instruction Pointer: The instruction pointer is responsible for keeping track of the currently executing instruction. It points to the next instruction to be fetched and executed.

5. Instruction Decoder: The instruction decoder decodes the instructions received from the control unit and generates control signals for the execution unit.

6. FPU Execution Unit: The FPU execution unit performs the actual mathematical operations, such as addition, subtraction, multiplication, and division. It operates on the data stored in the data registers and produces results that are stored back in the data registers.

Together, these components form the register set of the 80387DX math coprocessor, enabling it to perform complex floating-point arithmetic operations efficiently.

5. Answer the following:

a. Form an instruction that adds the contents of register 3 to the top of the stack.

b. Choose an instruction that subtracts the contents of register 2 from the top of the stack and store the result in register 2.

c. What does the FSAVE instruction save?

Ans: a. To add the contents of register 3 to the top of the stack, you would typically use an instruction like `PUSH R3`, which pushes the contents of register 3 onto the stack.

b. An instruction that subtracts the contents of register 2 from the top of the stack and stores the result in register 2 could be `POP R2`, followed by a subtraction operation between the popped value and the contents of register 2, and then storing the result back in register 2.

c. The `FSAVE` instruction typically saves the state of the floating-point unit (FPU), including its registers and control flags, to memory for later restoration or context switching purposes.

6. Explain the following 80387 instructions

a. FINIT b. F2XM1 c. FSQRT

Ans: a. FINIT:

The `FINIT` instruction is used to initialize the Floating-Point Unit (FPU) of the 80387 math coprocessor. It stands for "FPU Initialize." When executed, `FINIT` resets the FPU to a known state and prepares it for subsequent floating-point operations. It performs the following actions:
- Clears any pending exceptions in the FPU status register.
- Resets the control word to its default value, which enables all exceptions and sets the rounding mode to round to the nearest representable value.
- Clears the FPU tag word, indicating that all FPU registers are empty and available for use.

b. F2XM1:
The `F2XM1` instruction computes the value of 2 raised to the power of the value in the ST(0) register minus 1. It stands for "FPU 2 to the X minus 1." This instruction is used to perform exponentiation operations in the FPU. It follows these steps:
- Reads the value from the ST(0) register, which represents the exponent.
- Computes 2 raised to the power of that exponent, subtracts 1 from the result, and stores the final value back in the ST(0) register.

c. FSQRT:
The `FSQRT` instruction calculates the square root of the value in the ST(0) register. It stands for "FPU Square Root." When executed, it performs the following actions:
- Reads the value from the ST(0) register.
- Calculates the square root of that value and stores the result back in the ST(0) register.
- Handles special cases such as negative values, zero, or infinity, as specified by the IEEE 754 floating-point standard.

These instructions are part of the instruction set provided by the 80387 math coprocessor and are used to perform specific mathematical operations efficiently in floating-point arithmetic.

7. What is the difference between the FTST instruction and FXAM?

Ans: The FTST (Floating-Point Test) and FXAM (Floating-Point Examine) instructions in the 80387 math coprocessor are used for different purposes.

1. FTST (Floating-Point Test):
The FTST instruction is used to test the value in the ST(0) register of the FPU. It performs a comparison of the value against zero and sets the appropriate condition code flags in the FPU status register. Here's the correct description of FTST:
- FTST compares the value in the ST(0) register with zero.
- It sets the condition code flags in the FPU status register based on the result of the comparison. These flags include the zero (Z), sign (S), parity (P), and overflow (O) flags.
- FTST is primarily used for conditional branching or decision-making based on the value in the ST(0) register. For example, it can be used to check if the value is positive, negative, or zero.

2. FXAM (Floating-Point Examine):
The FXAM instruction is used to examine and classify the value in the ST(0) register of the FPU. It provides information about the value's characteristics, such as its sign, exponent, and whether it is a special value like NaN (Not-a-Number) or infinity. Here's the correct description of FXAM:
- FXAM examines the value in the ST(0) register and sets the appropriate condition code flags in the FPU status register.
- The flags set by FXAM include the sign (S), zero (Z), overflow (O), underflow (U), precision (P), and exception summary (ES) flags.
- FXAM helps classify the value in the ST(0) register into different categories, such as zero, denormalized numbers, normalized numbers, special values like NaN or infinity, and other exceptional conditions.
- The classification information provided by FXAM is useful for decision-making, error handling, or determining the appropriate arithmetic operations to perform based on the characteristics of the value.

In summary, the FTST instruction is used to compare the value in the ST(0) register with zero and set condition code flags based on the result. On the other hand, the FXAM instruction examines and classifies the value in the ST(0) register, providing information about its characteristics and special values.

8. Discuss the register set of 80387.

Ans: The 80387 math coprocessor, also known as the FPU (Floating-Point Unit), is an extension to the Intel x86 architecture that provides hardware support for floating-point arithmetic operations. The register set of the 80387 consists of several registers that are used for storing operands, intermediate results, and control information. Here's a discussion of the register set:

1. Floating-Point Data Registers (ST(0) to ST(7)):
The 80387 has eight 80-bit floating-point data registers, labeled ST(0) to ST(7). These registers are used for storing floating-point operands, intermediate results, and final results of arithmetic operations. Each register can store single-precision (32-bit), double-precision (64-bit), or extended-precision (80-bit) floating-point values. The top of the stack is represented by ST(0), and the bottom by ST(7).

2. Floating-Point Control Register (Control Word):
The Control Word register (CW) is used to control various aspects of the FPU's behavior. It stores control settings such as precision control, rounding mode, and exception masking. The Control Word allows programmers to configure the FPU's behavior according to their specific requirements.

3. Floating-Point Status Register (Status Word):
The Status Word register (SW) is used to indicate the status of the FPU. It contains flags that provide information about the results of floating-point operations, such as overflow, underflow, precision loss, and various exceptional conditions. The Status Word flags are used for error checking and exception handling.

4. Floating-Point Tag Register (Tag Word):
The Tag Word register (TW) is used to track the state of the floating-point data registers. Each entry in the Tag Word corresponds to one of the eight floating-point data registers (ST(0) to ST(7)). The Tag Word indicates whether a register contains a valid floating-point value, an empty value, or a special value like NaN (Not-a-Number) or infinity.

5. Floating-Point Instruction Pointer (Instruction Pointer):
The Instruction Pointer register (IP) is used to store the address of the next instruction to be executed by the FPU. It is used for exception handling and debugging purposes.

6. Floating-Point Operand Pointer (Operand Pointer):
The Operand Pointer register (OP) is used to store the address of the current operand being processed by the FPU. It is used for exception handling and debugging purposes.

These registers collectively form the register set of the 80387 math coprocessor. They provide the necessary storage and control mechanisms for performing floating-point arithmetic operations and handling exceptions in a hardware-accelerated manner.

9. What are the data types supported by 80387?

Ans: The 80387 math coprocessor supports multiple data types for floating-point arithmetic operations. The supported data types are as follows:

1. Single-Precision Floating-Point:
The 80387 supports single-precision floating-point arithmetic operations, which use 32 bits to represent floating-point numbers. Single-precision values follow the IEEE 754 standard and have a sign bit, an 8-bit exponent, and a 23-bit fraction.

2. Double-Precision Floating-Point:
The 80387 also supports double-precision floating-point arithmetic operations, which use 64 bits to represent floating-point numbers. Double-precision values also follow the IEEE 754 standard and have a sign bit, an 11-bit exponent, and a 52-bit fraction.

3. Extended-Precision Floating-Point:
One of the notable features of the 80387 is its support for extended-precision floating-point arithmetic operations. Extended-precision values use 80 bits to represent floating-point numbers, providing higher precision compared to single-precision and double-precision formats. Extended-precision values have a sign bit, a 15-bit exponent, and a 64-bit fraction.

It's important to note that the 80387 can operate on all three data types (single, double, and extended precision) within its register set. The eight floating-point data registers (ST(0) to ST(7)) can store values of any of these data types, allowing the coprocessor to perform arithmetic operations on different precisions and handle intermediate results with higher accuracy.

The choice of data type depends on the desired precision and the trade-off between precision and performance. Single precision is typically used when a lower precision is sufficient, double precision provides higher accuracy, and extended precision is employed when maximum precision is required, but at the cost of increased memory and processing requirements.