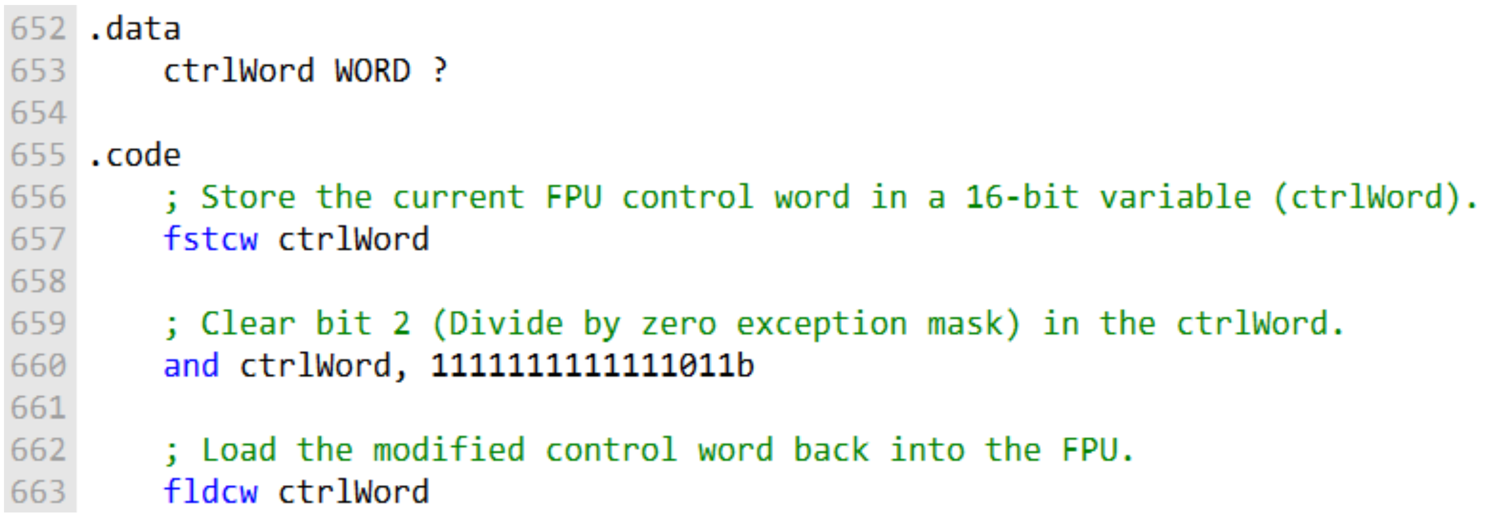
UnMasking Floating Point Exceptions

The passage below describes how to mask and unmask floating-point exceptions in assembly language.

Floating-point exceptions are masked by default, which means that when a floating-point exception is generated, the processor assigns a default value to the result and continues executing code.

To unmask a floating-point exception, you must clear the appropriate bit in the FPU control word. The following code unmasks the divide by zero exception:



```
.data
    ctrlWord WORD ?

.code
    ; Store the current FPU control word in a 16-bit variable (ctrlWord).
    fstcw ctrlWord

    ; Clear bit 2 (Divide by zero exception mask) in the ctrlWord.
    and ctrlWord, 1111111111111011b

    ; Load the modified control word back into the FPU.
    fldcw ctrlWord
```

This code stores the current FPU control word in ctrlWord, then clears the appropriate bit (bit 2) to unmask the "Divide by zero" exception.

Finally, it loads the modified control word back into the FPU, ensuring that the exception is unmasked for subsequent operations.

Once the divide by zero exception is unmasked, the processor will try to execute an appropriate exception handler.

If no exception handler is installed, the processor will display an error message and terminate the program.

Here is an example of code that divides by zero and generates an unmasked exception:

```
; Code that intentionally divides by zero, generating an unmasked exception
fild val1        ; Load a value into ST(0)
fdiv val2       ; Attempt to divide by zero (unmasked exception)
fst val2        ; Store the result (won't be reached due to exception)
```

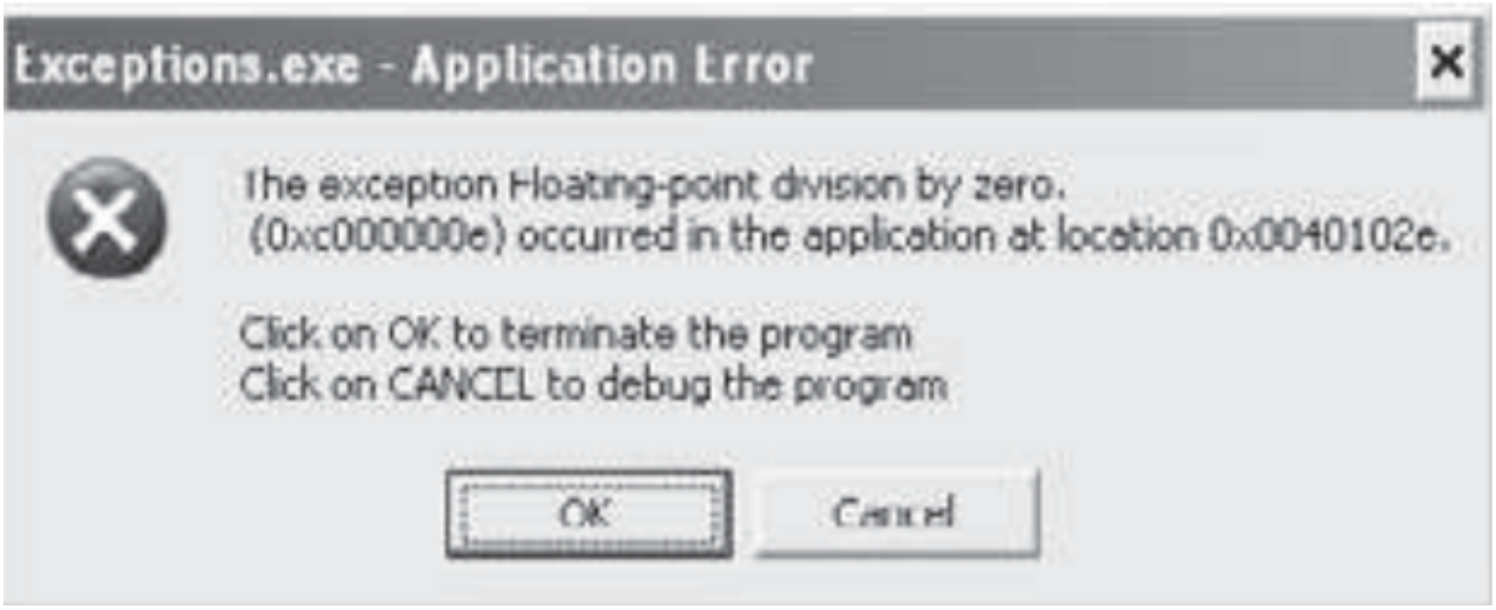
This code snippet intentionally divides by zero, which will generate an unmasked exception. Here's how it works step by step:

fild val1: Loads a value from the memory location val1 into the FPU's top stack register ST(0).

fdiv val2: Attempts to divide the value in ST(0) by the value stored in val2. Since the divisor is zero, this operation triggers a "Divide by zero" exception, which is unmasked as per the previous instructions.

fst val2: This instruction is intended to store the result back into the memory location val2. However, it won't be reached due to the unmasked exception generated in the previous step.

The program will trigger a system dialog indicating "Floating-Point Division by Zero," giving the user the option to retry or terminate the program.


Exceptions.exe - Application Error

The exception Floating-point division by zero.
(0xc000000e) occurred in the application at location 0x0040102e.

Click on OK to terminate the program
Click on CANCEL to debug the program

Q: Write an instruction that loads a duplicate of ST(0) onto the FPU stack.
A: You can load a duplicate of ST(0) onto the FPU stack with the fild ST(0) instruction.

Q: If ST(0) is positioned at absolute register R6 in the register stack, what is the position of ST(2)?
A: When ST(0) is at absolute register R6, ST(2) is positioned at absolute register R4 in the register stack.

Q: Name at least three FPU special-purpose registers.
A: Three FPU special-purpose registers are FSW (Floating-Point Status Word), FSTP (Floating-Point Control), and FTW (Floating-Point Tag Word).

Q: When the second letter of a floating-point instruction is B, what type of operand is indicated?
A: When the second letter of a floating-point instruction is B, it indicates that the instruction operates on BCD (Binary Coded Decimal) operands.

Q: Which floating-point instructions accept immediate operands?
A: Some floating-point instructions that accept immediate operands (constants) include fld1 (load +1.0), fldz (load +0.0), fldpi (load pi), fldl2t (load the base-2 logarithm of 10), and fldl2e (load the base-2 logarithm of e).