80x86 Floating Point Operations

Introduction

- Unlike integers, floating point numbers cannot be represented in memory easily.
- We need to have some conversion to have a unique numeric representation for all floating point numbers.
- For that we use the standard introduced by IEEE(Institute of Electrical and Electronics Engineers) called IEEE-754 Standard.
- In IEEE-754 standard a floating point number can be represented either in Single Precision(4 Bytes) or Double Precision(8 Bytes).
- There is also another representation called Extended Precision which uses 10 Bytes to represent a number.
- If we convert a double precision floating point number to a single precision number, the result won't go wrong but it will be less precise.

IEEE 754 : Briefly

- IEEE-754 Standard can be summarized as follows.

Single Precision	1 bit	8 bits	23 bits
Double Precision	1 bit	11 bits	52 bits
	S	Exponent	Fraction

- S is the Sign Bit. If the number is -ve then S = 1 else S = 0.
- The number $f = (-1) \times S \times (1 + 0.Fraction) \times 2^{(Exponent - Bias)}$
- Bias : 127 for Single Precision and 1023 for Double Precision

Floating Point Hardware

- In the early intel microprocessors there were no built in floating point instructions.
- If we had to do some floating point operations then we had to do that using some software emulators(which will be about 100 times slower than direct hardware) or adding an extra chip to the PC's mother board called math coprocessor or floating point coprocessor.
- For 8086 and 8088 the math coprocessor was 8087. For 80286 it was 80287 and for 80386 it was 80387.
- From 80486DX onwards intel started integrating the math co-processor into the main processor.
- But still it exists like a separate unit inside the main processor with a separate set of instructions, separate stack and status flags.

Floating Point Hardware

- The hardware for floating point operations are made for even doing operations in extended precision.
- But if we are using single/double precision numbers from memory it will be automatically converted while storing or loading.
- There are eight floating point coprocessor registers called ST0, ST1, ST2, ... ST7.
- They are organized in the form of a stack with ST0 in the top.
- When we push or load a value to the stack, value of each registers is pushed downward by one register.
  item Using these floating point registers we implement floating point operations.
- There is also a 1 word Status Flag for the floating point operations, which is analogous to the CPU Flags.
- It contains the status of the floating point operations.

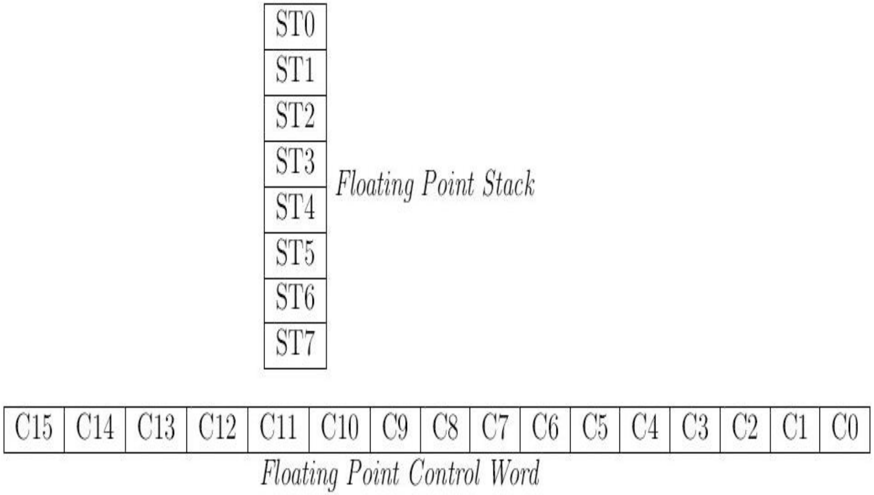

Figure: Floating Point Hardware

Floating Point Instructions : Load / Store Instructions

- FLD src : Loads the value of src on to the top of Floating Point stack(ie. ST0). src should be either a floating point register or a single precision/double precision/extended precision floating point number in memory.
  ST0 = src
- FILD src : Loads an integer from memory to ST0. src should be a word/double word/ quad word in memory.
  ST0 = (float) src
- FLD1 : Stores 1 to the top of Floating Point Stack.
- FLDZ : Stores 0 to the top of Floating Point Stack.

Floating Point Instructions : Load / Store Instructions

- FST dest : Stores ST0 to dest and will not pop off the value from the stack. dest should be a coprocessor register or a single precision/double precision F.P. number in memory.
  dest = ST0
- FSTP dest : Works similar to FST, it also pops off the value from ST0 after storing.
- FIST dest : Converts the number present in the top of F.P.. stack to an integer and stores that into dest. dest should be a coprocessor register or a single precision/double precision F.P.. number in memory. The way the number is being rounded depend on the value of coprocessor flags. But default it will be set in such a way that the number in ST0 is being rounded to the nearest integer.
  dest = (float)ST0

Floating Point Instructions : Load / Store Instructions

- FISTP dest : Works similar to FIST and it will also pop off the value from top of the stack.
- FXCH STn : The nth coprocessor register will be exchanged with ST0.
  $ST0 \longleftrightarrow STn$
- FFREE STn : Frees up the nth coprocessor register and marks it as empty.

Floating Point Instructions : Arithmetic Operations

- FADD src : ST0 = ST0 + src, src should be a coprocessor register or a single precision/double precision F.P. number in memory.
- FIADD src : ST0 = ST0 + (float)src. This is used to add an integer with ST0. src should be word or dword in memory.
- FSUB src : ST0 = ST0 - src, src should be a coprocessor register or a single precision/double precision F.P. number in memory.
- FSUBR src : ST0 = src - ST0, src should be a coprocessor register or a single precision /double precision F.P. number in memory.
- FISUB src : ST0 = ST0 - (float)src, this is used to subtract an integer from ST0. src should be word or dword in memory.
- FISUBR src : ST0 = (float)src - ST0.

Floating Point Instructions : Arithmetic Operations

- FMUL src : $ST0 = ST0 \times src$, src should be a coprocessor register or a single precision /double precision F.P. number in memory.
- FIMUL src : $ST0 = ST0 \times (float)src$, src should be a coprocessor register or a single precision/double precision F.P. number in memory.
- FDIV src : $ST0 = ST0 \div src$, src should be a coprocessor register or a single precision/double precision F.P. number in memory.
- FDIVR src : $ST0 = src \div ST0$, src should be a coprocessor register or a single precision /double precision F.P. number in memory.
- FIDIV src : $ST0 = ST0 \div (float)src$, src should be a word or dword in memory.
- FIDIVR src : $ST0 = (float)src \div ST0$, src should be a word or dword in memory.

Floating Point Instructions : Comparison Instructions

- The usual comparison instructions FCOM and FCOMP affects the coprocessors status word, but the processor cannot execute direct jump instruction by checking these values.
- So we need to copy the coprocessor flag values to the CPU flags in order to implement a jump based on result of comparison.
- FSTSW instruction can be used to copy the value of coprocessor status word to AX and then we can use SAHF to copy the value from AL to CPU flags.

Floating Point Instructions : Comparison Instructions

- FCOM src : Compares src with ST0, src should be a coprocessor register or a single precision/double precision F.P. number in memory.
- FCOMP src : Works similar to FCOM, it will also pop off the value from ST0 after comparison.
- FSTSW src : Stores co-processor status word to a dword in memory or to AX register.

Floating Point Instructions : Comparison Instructions

- SAHF : Stores AH to CPU Flags. Eg:

```
fld dword[var1]
fcomp dword[var2]
fstsw
sahf
ja greater
```

  In Pentium Pro and later processors (like Pentium II, III, IV etc) Intel added two other instructions FCOMI and FCOMIP which affects the CPU Flags directly after a floating point comparison.

- FCOMI src : Compares ST0 with src and affects the CPU Flags directly. src must be coprocessor register.
- FCOMIP : Compares ST0 with src and affects the CPU Flags directly. It will then pop off the value in ST0. src must be a coprocessor register.

Floating Point Instructions : Miscellaneous Instructions

- FCHS : $ST0 = -(ST0)$
- FABS : $ST0 = |ST0|$
- FSQRT : $ST0 = \sqrt{ST0}$
- FSIN : $ST0 = sin(ST0)$
- FCOS : $ST0 = cos(ST0)$
- FLDPI : Loads value of IA into ST0

THANK YOU