The x87 Floating-Point Unit

Lecture 18

Intel Manual, vol. 1, Chapter 8

Robb T. Koether

Hampden-Sydney College

Wed, Mar 25, 2009

Outline

Floating-Point Data Types

- The x87 Floating Point Unit (FPU) recognizes three floating-point types.
  - **`float`** - single precision, 32 bits.
  - **`double`** - double precision, 64 bits.
  - **`long double`** - double extended precision, 80 bits.
- We will use the type **`double`** in our compiler.
- However, in the FPU, all calculations will be as **`long doubles`**.

The FPU Status Register

- The 16-bit status register contains a number of bit fields that are set by floating-point instructions, including a 3-bit field TOP that points to the top of the FPU stack.
- The size of the stack is 8; TOP holds a value from 0 to 7.
- We will have use later for the bit fields `C0`, `C1`, `C2`, and `C3`, which are condition codes containing information about the most recent floating-point operation.

The FPU Control Word

- The 16-bit control word contains a number of bit fields, including
  - A 2-bit field PC that controls precision.
  - A 2-bit field RC that controls rounding.

The PC Field

- The PC settings
  - `00` = single precision.
  - `10` = double precision.
  - `11` = double extended-precision.
- The default is double extended-precision.

The RC Field

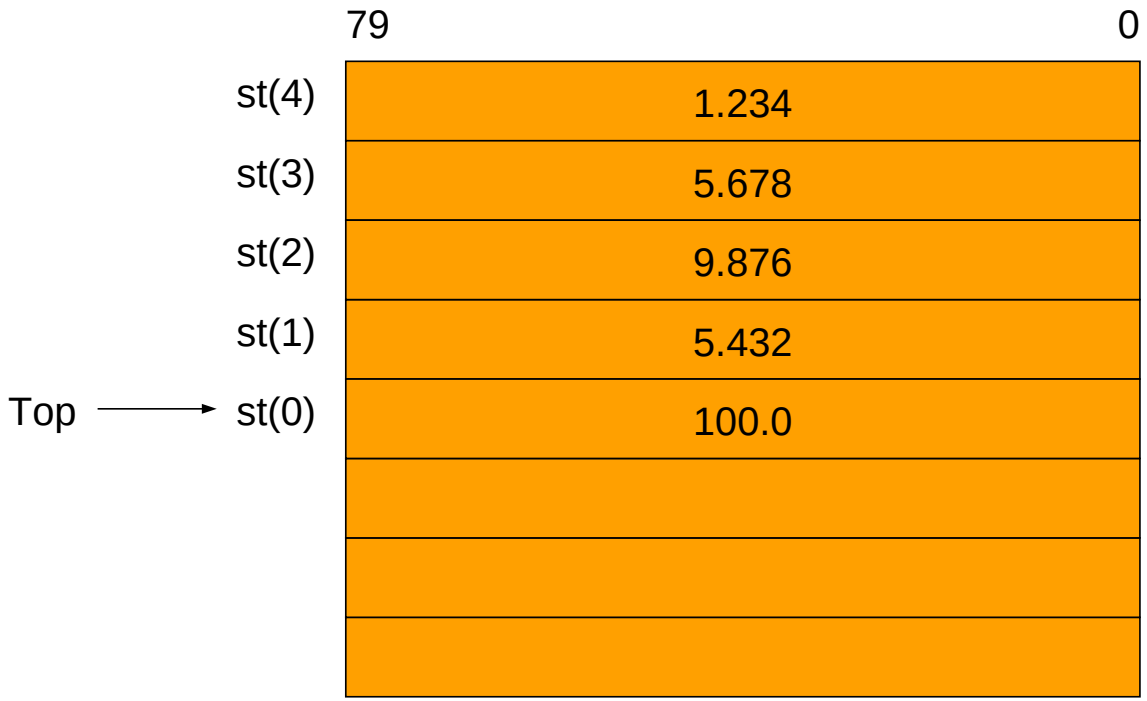
- The RC settings
  - `00` = Round to nearest.
  - `01` = Round down (towards $-\infty$).
  - `10` = Round up (towards $+\infty$).
  - `11` = Round towards zero.
- The default is round to nearest.
- Therefore, when we convert a **`double`** to an **`int`**, the value will be rounded, not truncated.

The x87 FPU Architecture

- The FPU has 8 general purpose 80-bit (double extended-precision) registers.
- They are labeled `st(0)`, `st(1)`, ..., `st(7)`.
- They are organized as a stack, with `st(0)` on top.
- Typically, floating-point operations pop values off the stack and push results onto the stack.
- However, many instructions allow us to access any position in the stack.

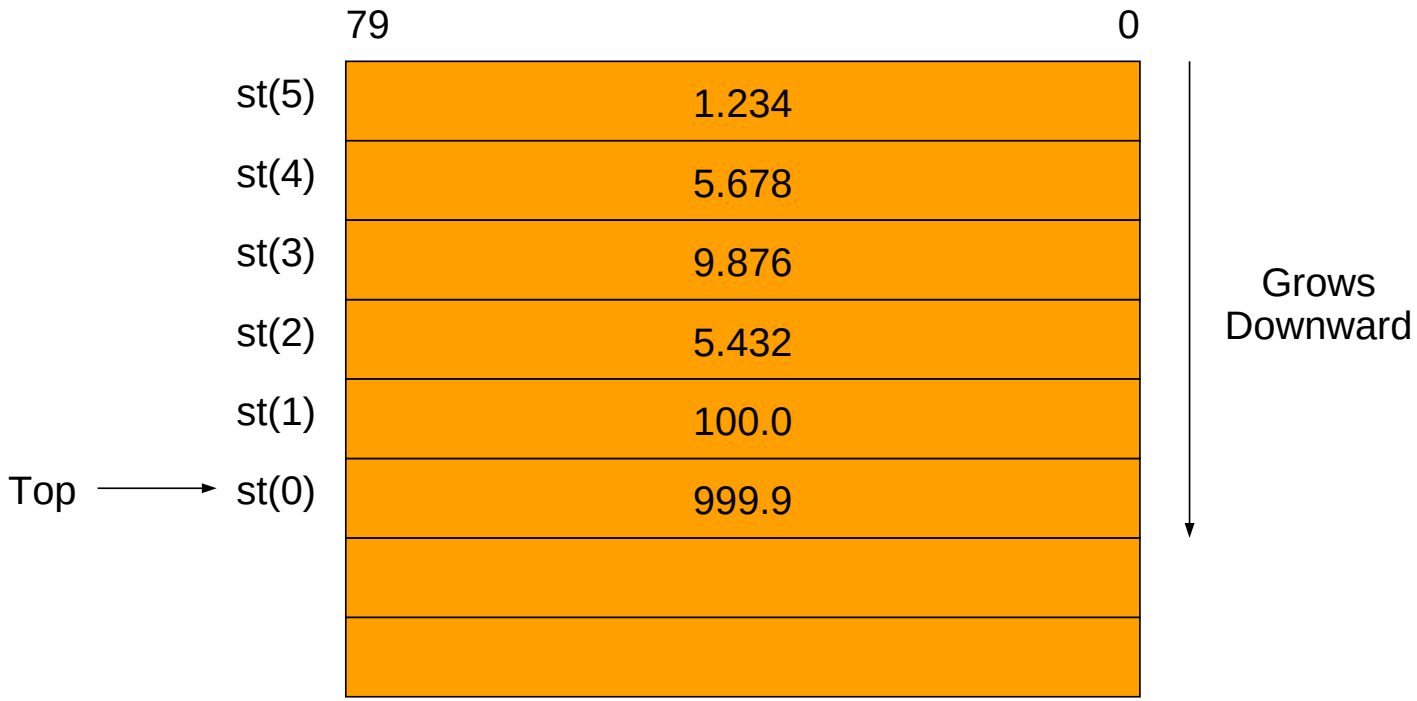
The FPU Stack

- Register `st(0)` is always on top of the stack.



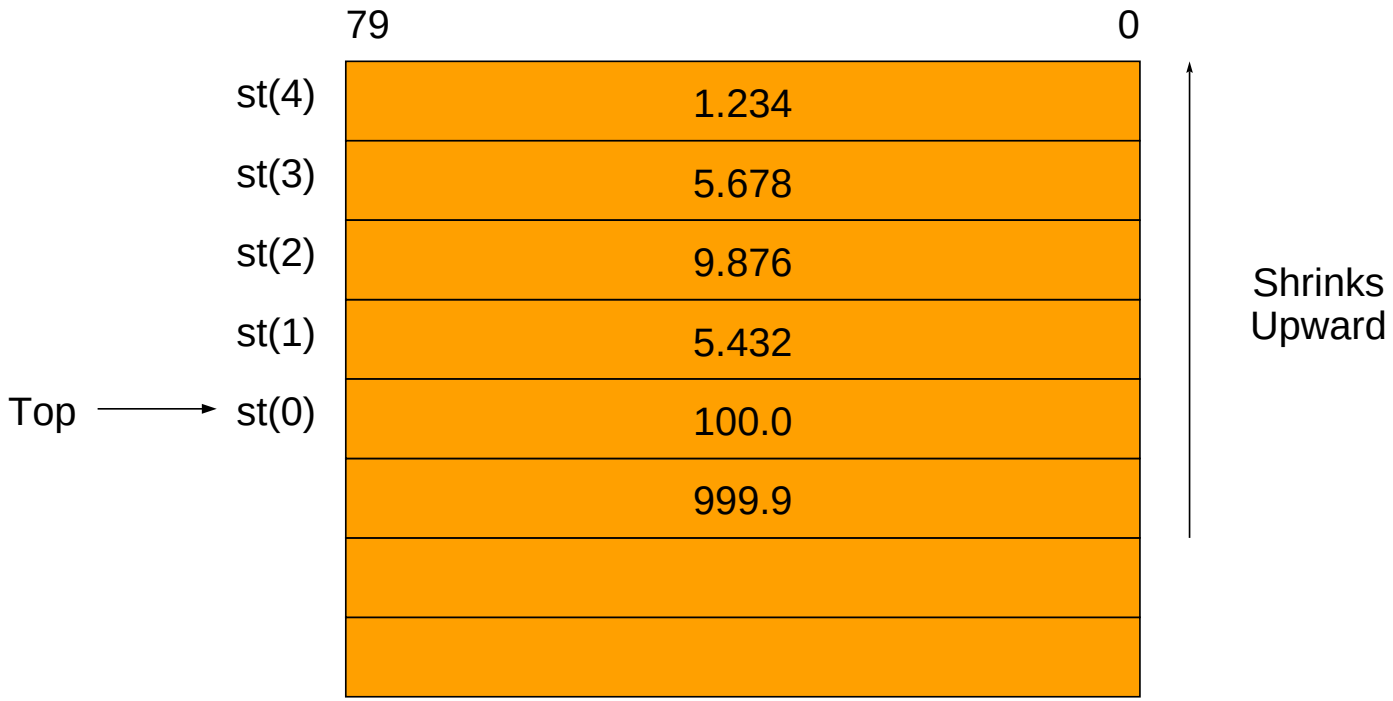
The FPU Stack

- When we push, `st(0)` moves to the next register.



The FPU Stack

- When we pop, `st(0)` moves to the previous register.



The x87 Instruction Set

- We will be interested in three categories of instruction.
  - Data transfer.
  - Basic arithmetic.
  - Comparisons.
- Other categories are
  - Transcendental instructions (trig, exponential, and logarithmic functions).
  - Loading constants (0, 1, π, $\log_2 10$, etc.)

Data Transfer - Load

The `fld` Load Instruction

```
fld src
```

- Pushes the floating-poing value at *src* onto the FPU stack.
- The operand *src* may be a memory address or an FPU register `st(i)`. It cannot be a non-FPU register.
- Examples
  - `fld avg`
  - `fld (%esp)`

Data Transfer - Load

The `fild` Integer Load Instruction

```
fild src
```

- Converts the integer at *src* to double extended-precision and pushes it onto the FPU stack. The operand *src* is a memory address.
- Examples
  - `fild count`
  - `fild (%esp)`

The x87 Instruction Set

- Many FPU instructions come in two versions.
  - The basic instruction: `Fxxx`.
  - The same instruction, followed by popping the FPU stack: `FxxxP`.

Data Transfer - Store

The `fild` Integer Load Instruction

```
fst  dst
fstp dst
```

- `fst` transfers the value at `st(0)` to *dst*. The operand *dst* may be a memory address or an FPU register `st(`*i*`)`.
- `fstp` is the same, except that it also pops the value off the FPU stack.
- Examples
  - `fst avg`
  - `fstp avg`

Data Transfer - Store

The `fist` Integer Store Instruction

```
fist  dst
fistp dst
```

- `fist` transfers the value at `st(0)` to *dst* and converts it to an integer. The operand *dst* is a memory address.
- fistp is the same, except that it also pops the value off the FPU stack.
- Examples
  - `fist (%esp)`
  - `fistp (%esp)`

Arithmetic - Add

The `faddp` Add Instruction

`faddp`

- Adds `st(0)` to `st(1)` and stores the result in `st(1)`.

$$\texttt{st(1)} \leftarrow \texttt{st(1)} + \texttt{st(0)}$$

- Pops the FPU stack, thereby removing `st(0)` and bringing `st(1)` to the top of the stack (`st(0)`).
- There are several other versions of `fadd` - see the manual.

Arithmetic - Multiply

The `fmulp` Multiply Instruction

```
fmulp
```

- Multiplies `st(1)` by `st(0)` and stores the result in `st(1)`.

$$\texttt{st(1)} \leftarrow \texttt{st(1)} \times \texttt{st(0)}$$

- Pops the FPU stack.
- There are several other versions of `fmul` - see the manual.

Arithmetic - Subtract

The `fsubrp` Subtract Instruction

`fsubrp`

- Subtracts `st(0)` from `st(1)` and stores the result in `st(1)`.

$$\texttt{st(1)} \leftarrow \texttt{st(1)} - \texttt{st(0)}$$

- Pops the FPU stack.
- There are several other versions of `fsub` - see the manual.

Arithmetic - Subtract

- The Intel manual describes `fsubp` and `fsubrp` as follows.
- `fsubp`
  - `st(1)` ← `st(1)` − `st(0)`; Pop stack.
  - Machine code `DEE9`.
- `fsubrp`
  - `st(1)` ← `st(0)` − `st(1)`; Pop stack.
  - Machine code `DEE1`.
- However, the gnu assembler will reverse their meanings.

Arithmetic - Divide

The `fdivrp` Divide Instruction

`fdivrp`

- Divides `st(1)` by `st(0)` and stores the result in `st(1)`.

$$\texttt{st(1)} \leftarrow \texttt{st(1)}/\texttt{st(0)}$$

- Pops the FPU stack.
- There are several other versions of `fdiv` - see the manual.

Arithmetic - Divide

- The Intel manual describes `fdivp` and fdivr as
- `fdivp`
  - `st(1)` ← `st(1)/st(0)`; Pop stack.
  - Machine code `DEF9`.
- `fdivrp`
  - `st(1)` ← `st(0)/st(1)`; Pop stack.
  - Machine code `DEF1`.
- However, the gnu assembler will reverse their meanings.

Arithmetic - Square Root

- There is a square-root instruction `fsqrt`.
- If we wanted to, we could create a special square-root operator, say `#`.
- Then the source code

  ```
  a = #b;
  ```

  would be interpreted as "assign to `a` the square root of `b`."
- No function call would be required.

Transcendental Functions

- The same is true of the following transcendental functions.
  - `fsin`: `st(0)` $\leftarrow \sin(\texttt{st(0)})$.
  - `fcos`: `st(0)` $\leftarrow \cos(\texttt{st(0)})$.
  - `fptan`: `st(0)` $\leftarrow 1.0$, `st(1)` $\leftarrow \tan(\texttt{st(0)})$.
  - `fpatan`: `st(0)` $\leftarrow \arctan(\texttt{st(1)}/\texttt{st(0)})$.
  - `fsincos`:
    `st(0)` $\leftarrow \cos(\texttt{st(0)})$, `st(1)` $\leftarrow \sin(\texttt{st(0)})$.

Assignment

Homework

- Read about the floating-point operations in the Intel Manual, Vol. 2A, Chapter 3, pages 215 through 338.