The x87 Floating-Point Unit
Lecture 27
Intel Manual, Vol. 1, Chapter 8
Intel Manual, Vol. 2

Robb T. Koether
Hampden-Sydney College
Fri, Mar 27, 2015
1. The x87 FPU Architecture
2. The FPU Registers and Stack
3. The x87 Instruction Set
4. The FPU Status Register and Rounding
5. Assignment
Outline

1. The x87 FPU Architecture
2. The FPU Registers and Stack
3. The x87 Instruction Set
4. The FPU Status Register and Rounding
5. Assignment
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`. 
Outline

1. The x87 FPU Architecture
2. The FPU Registers and Stack
3. The x87 Instruction Set
4. The FPU Status Register and Rounding
5. Assignment
The FPU has 8 general purpose 80-bit (double extended-precision) registers.

They are labeled \( st(0), st(1), \ldots, st(7) \).

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.
Register $\text{st}(0)$ is always on top of the stack.
When we push, \( st(0) \) moves to the next register.

When we pop, \( st(0) \) moves to the previous register.
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>st(4)</td>
<td>1.234</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>st(3)</td>
<td>5.678</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>st(2)</td>
<td>9.876</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>st(1)</td>
<td>5.432</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>st(0)</td>
<td>100.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The FPU Stack

The x87 Floating-Point Unit
Outline

1. The x87 FPU Architecture
2. The FPU Registers and Stack
3. The x87 Instruction Set
4. The FPU Status Register and Rounding
5. Assignment
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, $\pi$, $\log_2 10$, etc.)
Data Transfer - Load

The `fld` Load Instruction

```
fld src
```

- Pushes the floating-point 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 \texttt{fild} Integer Load Instruction

\begin{verbatim}
fild \ src
\end{verbatim}

- Converts the integer at \textit{src} to double extended-precision and pushes it onto the FPU stack. The operand \textit{src} is a memory address.

- Examples
  - \texttt{fild count}
  - \texttt{fild (%esp)}
Many FPU instructions come in two versions.

- The basic instruction: $F_{xxx}$.
- The same instruction, followed by popping the FPU stack: $F_{xxx}P$. 
The **`fld`** 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`
The **fist** Integer Store Instruction

```markdown
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)`
The \texttt{faddp} Add Instruction

\begin{itemize}
  \item Adds $\text{st}(0)$ to $\text{st}(1)$ and stores the result in $\text{st}(1)$.
    \begin{equation*}
      \text{st}(1) \leftarrow \text{st}(1) + \text{st}(0)
    \end{equation*}
  \item Pops the FPU stack, thereby removing $\text{st}(0)$ and bringing $\text{st}(1)$ to the top of the stack ($\text{st}(0)$).
  \item There are several other versions of \texttt{fadd} - see the manual.
\end{itemize}
Arithmetic - Multiply

The \texttt{fmulp} Multiply Instruction

\texttt{fmulp}

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

\[ st(1) \leftarrow st(1) \times st(0) \]

- Pops the FPU stack.
- There are several other versions of \texttt{fmul} - see the manual.
The \texttt{fsubrp} Subtract Instruction

- \textbf{Subtracts $st(0)$ from $st(1)$ and stores the result in $st(1)$}.

\[
st(1) \leftarrow st(1) - st(0)
\]

- Pops the FPU stack.
- There are several other versions of \texttt{fsub} - see the manual.
The Intel manual describes \texttt{fsubp} and \texttt{fsubrp} as follows.

\texttt{fsubp}
- \texttt{st(1) \leftarrow st(1) - st(0)}; Pop stack.
- Machine code \texttt{DEE9}.

\texttt{fsubrp}
- \texttt{st(1) \leftarrow st(0) - st(1)}; Pop stack.
- Machine code \texttt{DEE1}.

However, the gnu assembler will reverse their meanings.
The \texttt{fdivrp} Divide Instruction

\texttt{fdivrp}

- Divides $\text{st}(1)$ by $\text{st}(0)$ and stores the result in $\text{st}(1)$.
  
  $$\text{st}(1) \leftarrow \text{st}(1)/\text{st}(0)$$

- Pops the FPU stack.

- There are several other versions of \texttt{fdiv} - see the manual.
The Intel manual describes **fdivp** and **fdivr** as

**fdivp**
- \( \text{st}(1) \leftarrow \text{st}(1)/\text{st}(0) \); Pop stack.

**fdivrp**
- \( \text{st}(1) \leftarrow \text{st}(0)/\text{st}(1) \); Pop stack.
- Machine code DEF1.

However, the gnu assembler will reverse their meanings.
There is a square-root instruction \texttt{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.
The same is true of the following transcendental functions.

- \textit{fsin}: \texttt{st(0) \leftarrow \sin(st(0)).}
- \textit{fcos}: \texttt{st(0) \leftarrow \cos(st(0)).}
- \textit{fptan}: \texttt{st(0) \leftarrow 1.0, st(1) \leftarrow \tan(st(0)).}
- \textit{fpatan}: \texttt{st(0) \leftarrow \arctan(st(1)/st(0)).}
- \textit{fsincos}: \texttt{st(0) \leftarrow \cos(st(0)), st(1) \leftarrow \sin(st(0)).}
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 $c_0$, $c_1$, $c_2$, and $c_3$, 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.
Outline

1. The x87 FPU Architecture
2. The FPU Registers and Stack
3. The x87 Instruction Set
4. The FPU Status Register and Rounding
5. Assignment
Assignment

- Read about the architecture of the FPU in the Intel Manual, Vol. 1, Chapter 8.
- Read about the floating-point operations in the Intel Manual, Vol. 2A, Chapter 3.