Data Types and Instruction Set of 8087

Internally, all data operands are converted to the 80-bit temporary real format. We have 3 types.

- Integer data type
- Packed BCD data type
- Real data type

**Coprocessor data types**

**Integer Data Type**

**Packed BCD**

**Real data type**

Example

Converting a decimal number into a Floating-point number.

1) Converting the decimal number into binary form.
2) Normalize the binary number
3) Calculate the biased exponent.
4) Store the number in the floating-point format.

Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>100.25</td>
</tr>
<tr>
<td>2)</td>
<td>1100100.01 = 1.10010001 * 26</td>
</tr>
<tr>
<td>3)</td>
<td>110+01111111=10000101</td>
</tr>
<tr>
<td>4)</td>
<td>Sign = 0</td>
</tr>
<tr>
<td></td>
<td>Exponent = 10000101</td>
</tr>
<tr>
<td></td>
<td>Significand = 10010001000000000000000000</td>
</tr>
</tbody>
</table>

In step 3 the biased exponent is the exponent a 26 or 110, plus a bias of 01111111 (7FH), single precision no use 7F and double precision no use 3FFFH.

In step 4 the information found in prior step is combined to form the floating point no.

**INSTRUCTION SET**

The 8087 instruction mnemonics begins with the letter F which stands for Floating point and distinguishes from 8086.

- These are grouped into Four functional groups.
- The 8087 detects an error condition usually called an exception when it executing an instruction it will set the bit in its Status register.

**Types**

I. DATA TRANSFER INSTRUCTIONS.
II. ARITHMETIC INSTRUCTIONS.
III. COMPARE INSTRUCTIONS.
IV. TRANSCENDENTAL INSTRUCTIONS.
(Trigonometric and Exponential)
I Data Transfers Instructions:

- **REAL TRANSFER**
  - **FLD** Load real
  - **FST** Store real
  - **FSTP** Store real and pop
  - **FXCH** Exchange registers

- **INTEGER TRANSFER**
  - **FILD** Load integer
  - **FIST** Store integer
  - **FISTP** Store integer and pop

- **PACKED DECIMAL TRANSFER (BCD)**
  - **FBLD** Load BCD
  - **FBSTP** Store BCD and pop

**Example**

- **FLD Source** - Decrement the stack pointer by one and copies a real number from a stack element or memory location to the new ST.
  - `FLD ST(3)`; Copies ST(3) to ST.
  - `FLD LONG_REAL[BX]`; Number from memory; copied to ST.

- **FLD Destination** - Copies ST to a specified stack position or to a specified memory location.
  - `FST ST(2)`; Copies ST to ST(2), and increment stack pointer.
  - `FST SHORT_REAL[BX]`; Copy ST to a memory at a SHORT_REAL[BX]

- **FXCH Destination** – Exchange the contents of ST with the contents of a specified stack element.
  - `FXCH ST(5)`; Swap ST and ST(5)

- **FILD Source** – Integer load. Convert integer number from memory to temporary-real format and push on 8087 stack.
  - `FILD DWORD PTR[BX]`; Short integer from memory at [BX].

- **FIST Destination** - Integer store. Convert number from ST to integer and copy to memory.
  - `FIST LONG_INT`; ST to memory locations named LONG_INT.

- **FISTP Destination** - Integer store and pop. Identical to FIST except that stack pointer is incremented after copy.

- **FBLD Source** - Convert BCD number from memory to temporary-real format and push on top of 8087 stack.

II Arithmetic Instructions:

- Four basic arithmetic functions:
  - Addition, Subtraction, Multiplication, and Division.
  - **Addition**
**FADD** Add real
**FADDP** Add real and pop
**FIADD** Add integer

**Subtraction**
**FSUB** Subtract real
**FSUBP** Subtract real and pop
**FISUB** Subtract integer
**FSUBR** Subtract real reversed
**FSUBRP** Subtract real and pop
**FISUBR** Subtract integer reversed

**Multiplication**
**FMUL** Multiply real
**FMULP** Multiply real and pop
**FIMUL** Multiply integer

**Advanced**
**FABS** Absolute value
**FCHS** Change sign
**FPREM** Partial remainder
**FPRNDINT** Round to integer
**FSCALE** Scale
**FSQRT** Square root
**FXTRACT** Extract exponent and mantissa.

**Example**

- **FADD** – Add real from specified source to specified destination. Source can be a stack or memory location. Destination must be a stack element. If no source or destination is specified, then ST is added to ST(1) and stack pointer is incremented so that the result of addition is at ST.
  - FADD ST(3), ST ;Add ST to ST(3), result in ST(3)
  - FADD ST,ST(4) ;Add ST(4) to ST, result in ST.
  - FADD ;ST + ST(1), pop stack result at ST
  - FADDP ST(1) ;Add ST(1) to ST. Increment stack ;pointer so ST(1) become ST.

- **FSUB** – Subtract the real number at the specified source from the real number at the specified destination and put the result in the specified destination.
  - FSUB ST(2), ST ;ST(2) = ST(2) – ST.
  - FSUB Rate ;ST = ST – real no from memory.
  - FSUB ;ST = (ST(1) – ST)

- **FSUBP** - Subtract ST from specified stack element and put result in specified stack element. Then increment the pointer by one.
  - FSUBP ST(1) ;ST(1)-ST. ST(1) becomes new ST

- **FISUB** – Integer from memory subtracted from ST, result in ST.
  - FISUB Cars_Sold ;ST becomes ST – integer from memory
III Compare Instructions:

- **Comparison**
  - FCOM Compare real
  - FCOMP Compare real and pop
  - FCOMPP Compare real and pop twice
  - FICOM Compare integer
  - FICOMP Compare integer and pop
  - FTST Test ST against +0.0
  - FXAM Examine ST

III Transcendental Instruction:

- **Transcendental FPTAN**
  - Partial tangent FPATAN
  - Partial arctangent F2XM1
  - 2x - 1
  - FYL2X Y log2X
  - FYL2XP1 Y log2(X+1)

  **Example**
  - **FPTAN** – Compute the values for a ratio of Y/X for an angle in ST. The angle must be in radians, and the angle must be in the range of 0 < angle < π/4.
  - **F2XM1** – Compute Y=2x-1 for an X value in ST. The result Y replaces X in ST. X must be in the range 0≤X≤0.5.
  - **FYL2X** - Calculate Y(LOG2X). X must be in the range of 0 < X < ∞ any Y must be in the range -∞<Y<+∞.
  - **FYL2XP1** – Compute the function Y(LOG2(X+1)). This instruction is almost identical to FYL2X except that it gives more accurate results when compute log of a number very close to one.

**Constant Instructions.**

- **Load Constant Instruction**
  - FLDZ Load +0.0
  - FLDI Load +1.0
  - FLDPI Load π
  - FLDL2T Load log210
  - FLDL2E Load log2e
  - FLDLG2 Load log102
  - FLDELN2 Load loge2

**ALGORITHM**

To calculate x to the power of y
- Load base, power.
- Compute (y)*( log2 x)
- Separate integer(i) ,fraction(f) of a real number
- Divide fraction (f) by 2
- Compute (2 f/2) * ( 2f/2)
- xy = (2x) * (2y )