Floating-Point Unit

The **FPU** is a specialized hardware unit that performs floating-point calculations. It has its own set of registers called a **register stack.**

The FPU loads values from memory into the register stack, performs calculations, and stores stack values into memory.

The FPU evaluates mathematical expressions in **postfix format.**

A **postfix expression** is an expression where the operands appear before the operators. For example, the postfix expression for the **in-fix expression (5 * 6) - 4** is **5 6 * 4 -.**

Left to Right	Stack		Action
5	5	ST (0)	push 5
5 6	5 6	ST (1) ST (0)	push 6
5 6 *	30	ST (0)	Multiply ST(1) by ST(0) and pop ST(0) off the stack.
5 6 * 4	30 4	ST (1) ST (0)	push 4
5 6 * 4 -	26	ST (0)	Subtract ST(0) from ST(1) and pop ST(0) off the stack.

The FPU uses an expression stack to hold intermediate values during the evaluation of postfix expressions. The expression stack is a **last-in-first-out (LIFO) stack.**

This means that the last value pushed onto the stack is the first value popped off the stack.

To evaluate a postfix expression, the FPU pushes the operands onto the expression stack.

Then, it pops the top two operands off the stack and performs the operation specified by the next operator in the expression. The result of the operation is pushed onto the stack.

This process continues until all of the operators in the expression have been processed.

The following table shows some examples of equivalent infix and postfix expressions:

Infix expression	Postfix expression
(5 * 6) - 4	5 6 * 4 -
(A + B) * C	A B + C *
(A - B) / C	A B - C /
A + B	A B +
(A - B) / D	A B - D /
(A + B) * (C + D)	A B + C D + *
((A + B) / C) * (E - F)	A B + C / E F - *

Note that the parentheses in the infix expressions are not necessary in the postfix expressions, because the order of operations is implied by the order of the operands.

For example, in the expression (A + B) * (C + D), the multiplication operation is performed before the addition operation, even though there are no parentheses. This is because multiplication has a higher precedence than addition.

Here is an example of how the FPU would evaluate the postfix expression A B + using its expression stack:

- Expression stack: empty
- Next operand: A
- Push A onto the expression stack
- Expression stack: A
- Next operand: B
- Push B onto the expression stack
- Expression stack: A, B
- Next operator: +
- Pop the top two operands off the expression stack and perform the addition operation
- Push the result of the addition operation onto the expression stack
- Expression stack: A + B

The FPU would then stop evaluating the expression, because there are no more operands or operators. The result of the expression is A + B.

The FPU evaluates all postfix expressions in the same way.

It pushes the operands onto the expression stack, pops the top two operands off the stack and performs the operation specified by the next operator, and pushes the result of the operation onto the stack.

This process continues until all of the operators in the expression have been processed. The final value on the expression stack is the result of the expression.

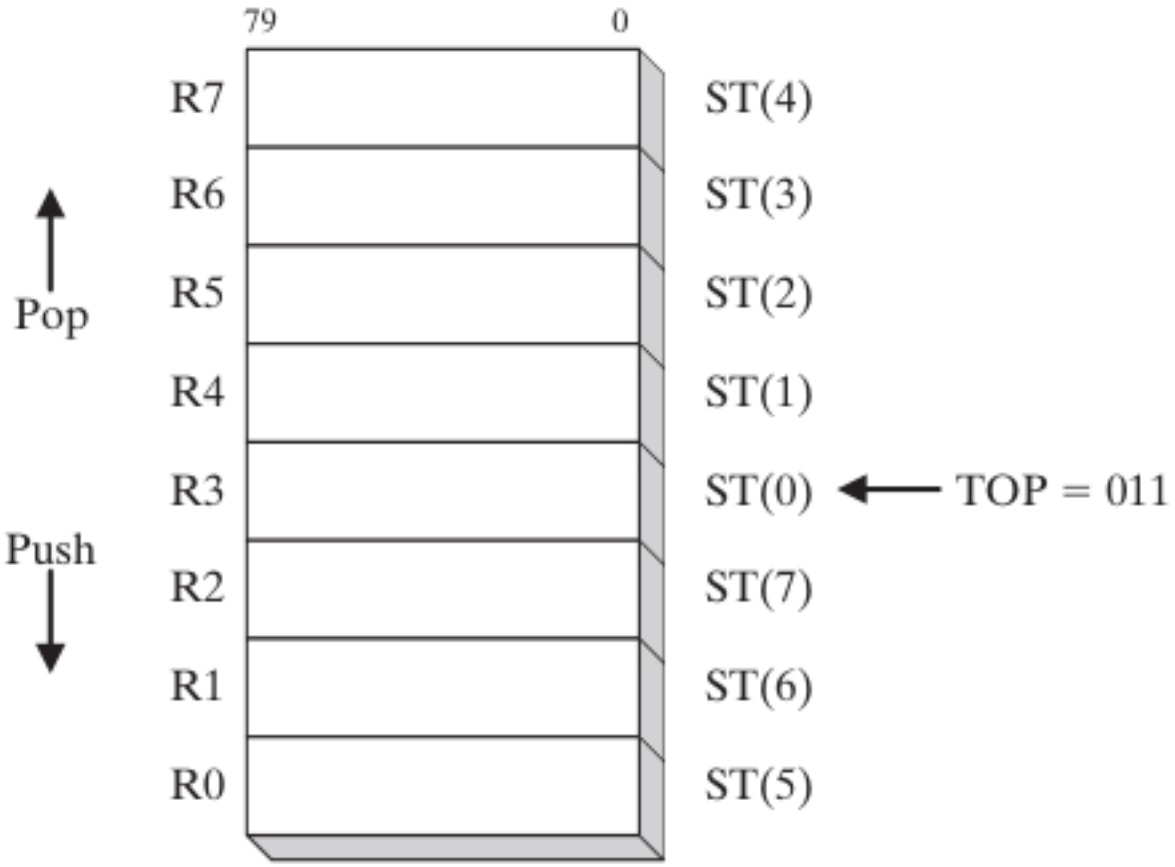
FPU DATA REGISTERS

The FPU has eight individually addressable 80-bit data registers named R0 through R7. Together, they are called a register stack.

The FPU stack works in a **last-in-first-out (LIFO)** manner. This means that the last value pushed onto the stack is the first value popped off the stack.

The top of the FPU stack is indicated by a three-bit field named TOP in the FPU status word. The register at the top of the stack is also known as ST(0).

The rest of the registers are known as ST(1), ST(2), ..., ST(7), in order.

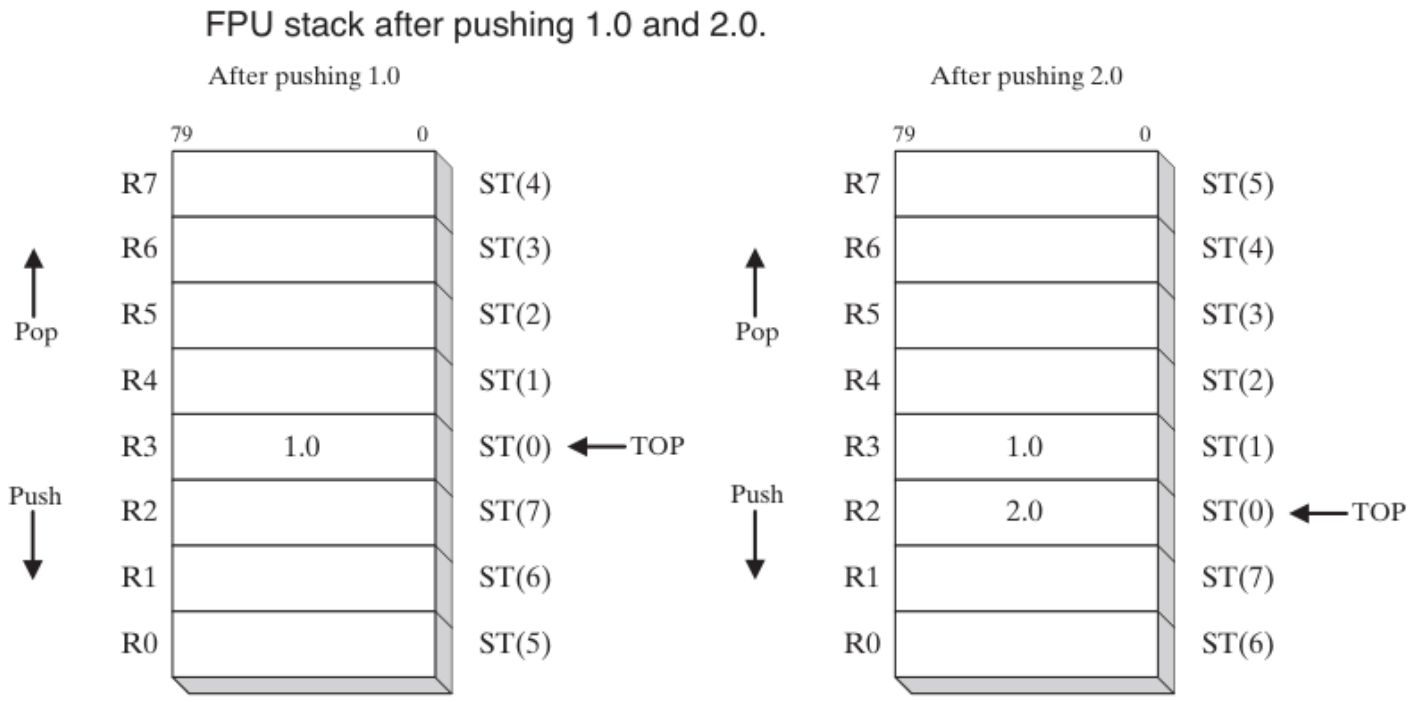


To push a value onto the FPU stack, the FPU decrements TOP by 1 and copies the value into the register identified as ST(0). If TOP equals 0 before a push, TOP wraps around to register R7.

To pop a value off the FPU stack, the FPU copies the data at ST(0) into an operand, then adds 1 to TOP. If TOP equals 7 before the pop, it wraps around to register R0.

If loading a value into the stack would result in overwriting existing data in the register stack, a floating-point exception is generated.

The following diagram shows the FPU stack after 1.0 and 2.0 have been pushed onto the stack:



To perform a floating-point operation, the FPU pops the required operands off the stack, performs the operation, and pushes the result back onto the stack.

For example, to perform the addition operation 1.0 + 2.0, the FPU would pop the operands 1.0 and 2.0 off the stack, add them together, and push the result, 3.0, back onto the stack.

The FPU stack is a powerful tool for performing floating-point calculations. It allows the FPU to efficiently perform complex operations without having to store intermediate results in memory.

The functionality of the Floating-Point Unit (FPU) in a processor, which is responsible for handling floating-point arithmetic operations. Let's break down some of the key points:

ST(n) Notation:

The FPU uses a notation like ST(0), ST(1), and so on to refer to stack registers. ST(0) always represents the top of the stack. This notation is used to address data within the FPU's stack-based architecture.

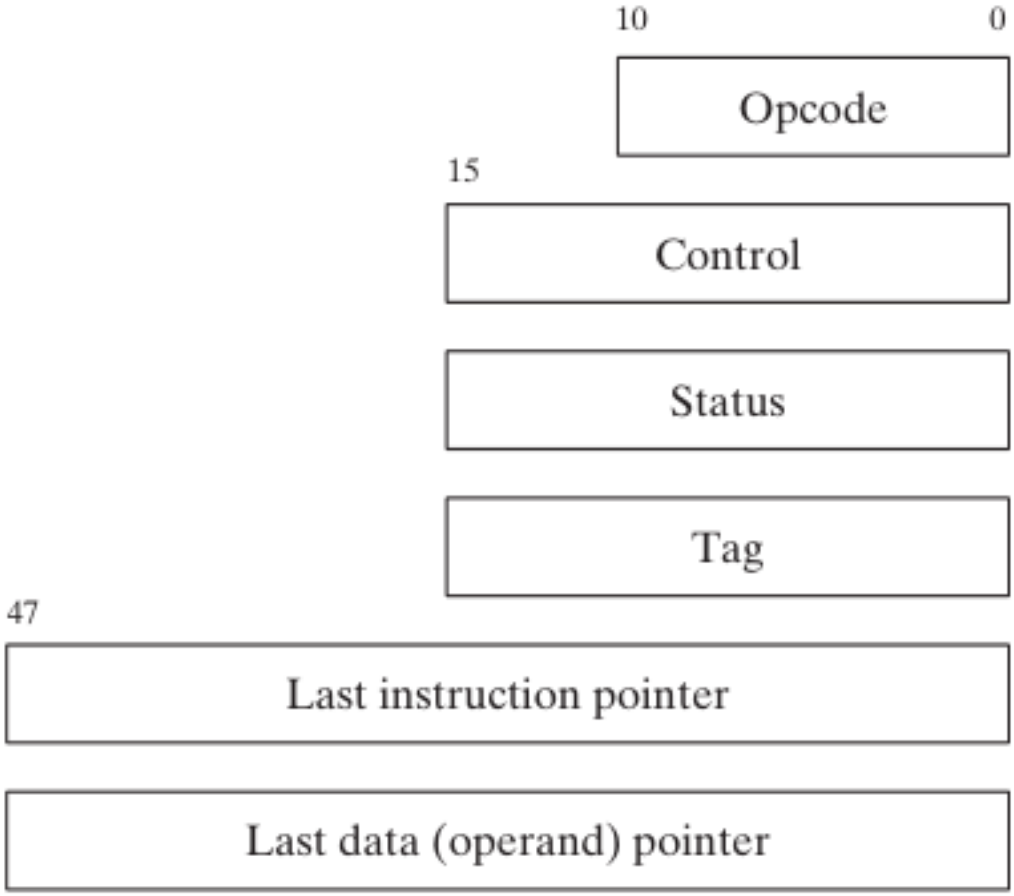
Floating-Point Value Format:

The floating-point values within FPU registers are stored in the IEEE 10-byte extended real format. This format is sometimes referred to as temporary real.

temporary real.

When the FPU stores the result of an operation in memory, it may translate it into various formats, including integers, long integers, single precision, double precision, or packed binary-coded decimal (BCD).

FPU special-purpose registers.



Special-Purpose Registers: The FPU has several special-purpose registers:

Opcode Register: Stores the opcode of the last non-control instruction executed.

Control Register: Controls the precision and rounding method used by the FPU during calculations. It can also be used to mask individual floating-point exceptions.

Status Register: Contains the top-of-stack pointer, condition codes, and information about exceptions or warnings.

Tag Register: Indicates the content of each register in the FPU data-register stack. It uses two bits per register to specify whether the register holds a valid number, zero, or a special value (NaN, infinity, denormal, unsupported format) or is empty.

Last Instruction Pointer Register: Stores a pointer to the last non-control instruction executed.

Last Data (Operand) Pointer Register: Stores a pointer to a data operand, if any, used by the last executed instruction.

ROUNDING IN FPU

The concept of rounding in floating-point calculations and how the Floating-Point Unit (FPU) handles it. I'll break down the key points for better understanding:

Purpose of Special-Purpose Registers: The special-purpose registers mentioned are used by operating systems to preserve the state of the FPU when switching between tasks. This state preservation is crucial for multitasking, where the CPU switches between different tasks.

Rounding in Floating-Point Calculations: The FPU aims to produce precise results from floating-point calculations, but sometimes the destination operand (the format in which the result is stored) can't represent the exact calculated result. Rounding is used to adjust the result to fit within the destination format.

Example of Rounding: Let's take an example where the precise result is **1.0111**, but the destination format can only represent three fractional bits.

Rounding can occur in two ways:

(a) Round up to the next higher value by adding 0.0001: 1.0111 → 1.100
(b) Round down to the closest value by subtracting 0.0001: 1.0111 → 1.011

Similar rounding can be applied for negative values:

(a) -1.0111 → -1.100
(b) -1.0111 → -1.011

Rounding Methods: The FPU provides four rounding methods:

- **Round to Nearest Even:** The result is rounded to the nearest value. If two values are equally close, the result is an even value (least significant bit = 0).

- **Round Down Toward -∞:** The result is rounded to a value less than or equal to the precise result.

- **Round Up Toward +∞:** The result is rounded to a value greater than or equal to the precise result.

- **Round Toward Zero (Truncation):** The absolute value of the rounded result is less than or equal to the precise result.

FPU Control Word: The FPU control word contains the RC (Rounding Control) field, which specifies the rounding method to use. The values are binary-encoded as follows:

- **00 binary:** Round to nearest even (default).
- **01 binary:** Round down toward negative infinity.
- **10 binary:** Round up toward positive infinity.
- **11 binary:** Round toward zero (truncate).

The default method is **"Round to Nearest Even"**, which is generally considered the most accurate and appropriate for most application programs.

The tables below would show how these rounding methods would be applied to the binary values 1.0111 and -1.0111, respectively.

Example: Rounding +1.0111.

Method	Precise Result	Rounded
Round to nearest even	1.0111	1.100
Round down toward $-\infty$	1.0111	1.011
Round toward $+\infty$	1.0111	1.100
Round toward zero	1.0111	1.011

Example: Rounding –1.0111.

Method	Precise Result	Rounded
Round to nearest (even)	-1.0111	-1.100
Round toward $-\infty$	-1.0111	-1.100
Round toward $+\infty$	-1.0111	-1.011
Round toward zero	-1.0111	-1.011

Rounding in floating-point arithmetic is essential to ensure that calculated results fit within the chosen format while minimizing error. The method chosen depends on the specific requirements of the application.