# Register File Module Documentation

### Introduction

The Register File module implements the complete register system for the RISC-V 16-bit processor, including all 16 registers (x0-x15) with their standard RISC-V ABI (Application Binary Interface) names and functionality. It provides secure access to registers with proper bounds checking and read-only protection for the zero register (x0).

#### Key features:

- Complete 16-register implementation (x0-x15)
- Support for standard RISC-V ABI names
- Read-only protection for x0 (zero register)
- Detailed register visualization and debugging
- Multiple display formats (basic, rich, summary)
- Register access statistics and tracking

## Class Structure

Register

Class representing a single 16-bit register with name, ABI name, and access control.

RegisterFile

Main class implementing the complete register file with all 16 registers.

## **Detailed Method Documentation**

Register Class

```
init (name, abi name, purpose, initial value=0, read only=False)
```

Purpose: Creates a new register instance

#### Parameters:

- name: Register name (e.g., "x0")
- abi name: ABI name (e.g., "zero")
- purpose: Description of register purpose
- initial value: Starting value (default 0)
- read only: Whether register is writable (default False)

#### Features:

- Values are masked to 16 bits
- Read-only flag enforced on writes

#### Key Methods:

- read(): Returns current 16-bit value
- write (value): Updates register value (fails if read-only)
- reset (): Clears register (except read-only registers)

RegisterFile Class

```
__init__()
```

Purpose: Initializes all 16 registers with RISC-V ABI mappings

Registers initialized:

```
python
Copy

Download
[
    ("x0", "zero", "Hard-wired zero", True),
```

```
("x1", "ra", "Return address", False),

("x2", "sp", "Stack pointer", False),

# ... all 16 registers ...
```

#### Core Access Methods

```
read(reg num)
```

Purpose: Reads value from register

Parameters:

• reg num: Register number (0-15)

#### Returns:

• Register value (0 for invalid register numbers)

```
write(reg_num, value)
```

Purpose: Writes value to register

Parameters:

- reg\_num: Register number (0-15)
- value: 16-bit value to write

#### Returns:

• True if write succeeded, False otherwise

### Safety:

- Blocks writes to x0
- Ignores invalid register numbers
- Masks values to 16 bits

```
reset all()
```

Purpose: Resets all registers to 0 (except x0)

**Utility Methods** 

```
get_register_info(reg_num)
```

Returns: Tuple of (name, abi\_name, purpose) for register

```
get_register_by_name(name)
```

Purpose: Looks up register number by name

Accepts:

- Numbered names ("x0", "x15")
- ABI names ("zero", "sp", "a0")

Returns:

• Register number or -1 if not found

Display Methods

```
display_registers_rich()
```

Purpose: Beautiful terminal display using Rich library

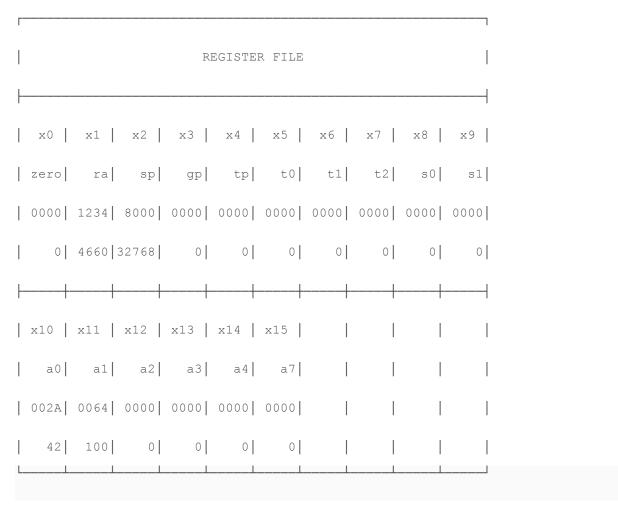
Features:

- Color-coded by register type
- 4x4 grid layout
- Shows both names and values

```
display_registers()
```

Purpose: ASCII table display of all registers

Format:



display\_summary()

Purpose: Shows non-zero registers and statistics

Example Output:

text

Сору

Download

REGISTER FILE SUMMARY

\_\_\_\_\_

↑ Non-zero registers (4):

x1(ra): 0x1234 (4660)

x2(sp): 0x8000 (32768)

## **Key Concepts**

### Register Types:

- 1. Zero Register (x0):
  - o Always returns 0
  - Writes are ignored
  - o Hardwired for constant zero value
- 2. Special Registers:
  - o ra (x1): Return address
  - o sp (x2): Stack pointer
  - o gp (x3): Global pointer
  - o tp (x4): Thread pointer
- 3. Temporary Registers (t0-t2):
  - o x5-x7: Caller-saved temporaries
- 4. Saved Registers (s0-s1):
  - o x8-x9: Callee-saved
- 5. Argument/Return Registers (a0-a7):
  - o x10-x15: Function arguments and return values

#### **Access Protection:**

• All writes are masked to 16 bits

- Invalid register numbers return safe values
- x0 is permanently read-only

# Example Usage

```
# Create register file
rf = RegisterFile()
# Write values
rf.write(1, 0x1234) # ra
rf.write(2, 0x8000) # sp
rf.write(10, 42) # a0
# Read back
value = rf.read(10) # Returns 42
# Display
rf.display_registers()
# Get register info
name, abi, purpose = rf.get register info(2) # Returns ("x2", "sp", "Stack
pointer")
# Lookup by name
reg_num = rf.get_register_by_name("a1") # Returns 11
```

# Testing

The module includes a test function (main()) that demonstrates:

- 1. Register file initialization
- 2. Writing to registers
- 3. Attempting to write to x0
- 4. Multiple display formats

#### Run tests with:

```
if __name__ == "__main__":
    main()
```

This register file implementation provides a complete and secure register system for the RISC-V processor, with extensive visualization capabilities for debugging and educational purposes. The strict access controls and clear register purposes follow standard RISC-V conventions while providing helpful debugging information.