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

```python
[

    ("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:

```
┌───────────────────────────────────────────────────────────┐
│                       REGISTER FILE                        │
├───────────────────────────────────────────────────────────┤
│  x0  │  x1  │  x2  │  x3  │  x4  │  x5  │  x6  │  x7  │  x8  │  x9  │
│ zero│   ra│   sp│   gp│   tp│   t0│   t1│   t2│   s0│   s1│
│ 0000│ 1234│ 8000│ 0000│ 0000│ 0000│ 0000│ 0000│ 0000│ 0000│
│    0│ 4660│32768│    0│    0│    0│    0│    0│    0│    0│
├─────┼─────┼─────┼─────┼─────┼─────┼─────┼─────┼─────┼─────┤
│ x10  │ x11  │ x12  │ x13  │ x14  │ x15  │      │      │      │      │
│   a0│   a1│   a2│   a3│   a4│   a7│     │     │     │     │
│ 002A│ 0064│ 0000│ 0000│ 0000│ 0000│     │     │     │     │
│   42│  100│    0│    0│    0│    0│     │     │     │     │
└─────┴─────┴─────┴─────┴─────┴─────┴─────┴─────┴─────┴─────┘
```

`display_summary()`

Purpose: Shows non-zero registers and statistics

Example Output:

text

Copy

Download

```
📊 REGISTER FILE SUMMARY
============================================================


📍 Non-zero registers (4):
   x1(ra): 0x1234 (4660)
   x2(sp): 0x8000 (32768)
   x10(a0): 0x002A (42)

   x11(a1): 0x0064 (100)



📊 Total registers: 16

📊 Active registers: 4
============================================================
```

Key Concepts

Register Types:

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

```python
# 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:

```python
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.