Memory Module Documentation

Introduction

The Memory module implements the complete memory system for the RISC-V 16-bit processor, following Harvard architecture with separate instruction and data memory spaces. It provides protected memory access with bounds checking and detailed memory operation tracking.

Key features:

- Separate instruction and data memory spaces
- Memory-mapped I/O with protected address ranges
- Detailed access logging and statistics
- Program loading from both binary files and direct instruction lists
- Memory visualization and debugging tools

Class Structure

`InstructionMemory`

Read-only memory for storing program instructions.

`DataMemory`

Read-write memory for data storage and manipulation.

Detailed Method Documentation

`InstructionMemory` Methods

`__init__(size=1024)`

Purpose: Initializes instruction memory

Parameters:

- `size`: Memory size in 16-bit words (default 1024)

Features:

- Initializes all memory locations to 0
- Tracks loaded program size
- Address range: 0x0000-0x03FF (for default size)

Program Loading Methods

`load_program(instructions, start_address=0)`

Purpose: Loads program from instruction list

Parameters:

- `instructions`: List of 16-bit instructions
- `start_address`: Loading address (default 0)

Validation:

- Checks address bounds
- Verifies program fits in memory

`load_from_binary_file(filename)`

Purpose: Loads program from binary file

Process:

- Reads file as little-endian 16-bit words
- Calls `load_program()` with converted instructions

Access Methods

`read_instruction(address)`

Purpose: Reads instruction from memory

Safety:

- Returns 0 (NOP) for invalid addresses
- Logs invalid access attempts

Utility Methods

`get_program_size()`

Returns: Size of loaded program in instructions

`display_memory(start=0, count=16)`

Purpose: Shows memory contents

Output: Address, hex value, binary, and disassembly

`_disassemble(instruction)`

Internal: Simple instruction disassembler for display

`DataMemory` Methods

`__init__(size=1024, base_address=0x1000)`

Purpose: Initializes data memory

Parameters:

- `size`: Memory size in 16-bit words (default 1024)
- `base_address`: Starting address (default 0x1000)

Features:

- Address range: 0x1000-0x13FF (for default size)
- Tracks all read/write operations

Access Methods

`read_word(address)`

Purpose: Reads 16-bit word from memory

Safety:

- Returns 0 for invalid addresses
- Logs all read operations

`write_word(address, value)`

Purpose: Writes 16-bit word to memory

Safety:

- Ignores invalid addresses
- Logs all write operations
- Masks value to 16 bits

Utility Methods

`clear_memory()`

Purpose: Resets all memory locations to 0

`get_statistics()`

Returns: Memory access statistics

Includes:

- Total accesses
- Read/write counts
- Memory configuration

`display_memory(start_offset=0, count=16)`

Purpose: Shows non-zero memory contents

Output: Address, hex value, and decimal value

`find_non_zero()`

Purpose: Finds all non-zero memory locations

Returns: List of (address, value) tuples

Key Concepts

Memory Architecture:

- Harvard architecture: Strict separation of instruction and data memory
- Instruction Memory:
    - Read-only after loading
    - Base address: 0x0000
    - Contains executable code
- Data Memory:
    - Read/write enabled
    - Base address: 0x1000
    - Used for variables and data storage

Address Mapping:

- All addresses are logical (processor-visible)
- Physical storage uses 0-based indexing
- Conversion handled internally:

```
physical_address = logical_address - base_address
```

Safety Features:

- Bounds checking on all accesses

- Invalid reads return 0 (acts as NOP for instructions)
- Invalid writes are ignored
- Detailed access logging

Example Usage

```
# Create memory system

imem = InstructionMemory(size=256)

dmem = DataMemory(size=256)



# Load program

program = [0x510A, 0x5205, 0x0312, 0xF000]  # ADDI, ADDI, ADD, HALT

imem.load_program(program)



# Work with data memory

dmem.write_word(0x1000, 0x1234)  # Store value

value = dmem.read_word(0x1000)   # Read back



# Inspect memory

imem.display_memory()

dmem.display_memory()



# Get statistics

stats = dmem.get_statistics()
print(f"Memory writes: {stats['writes']}")
```

Testing

The module includes a test function (`demo_memory_system()`) that demonstrates:

1. Memory initialization
2. Program loading
3. Data memory operations
4. Memory inspection
5. Statistics collection

Run tests with:

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

This memory system provides a robust implementation of RISC-V memory requirements with comprehensive debugging and visualization capabilities, essential for both processor simulation and educational purposes.