MIPS BASED IMAGE PROCESSING UNIT(GRAYSCALING UNIT)

1. Introduction

This report provides a comprehensive analysis of a MIPS-based Image Processing Unit (IPU) implementation. The project combines a MIPS processor core with specialized hardware for RGB to grayscale conversion, creating an integrated system capable of processing image data. The implementation uses Verilog HDL and demonstrates a practical application of processor-accelerator co-design.

The system consists of several key components:

1. A MIPS processor core with a 5-stage pipeline
2. A custom RGB to grayscale conversion IPU
3. A bus interface for communication between processor and IPU
4. Memory subsystems for instructions and data
5. Specialized register handling for image data

INSTRUCTION STRUCTURE

This processor is mainly made for I type instruction having Opcode, rs, rt, Immidate value.

I type Instructions set

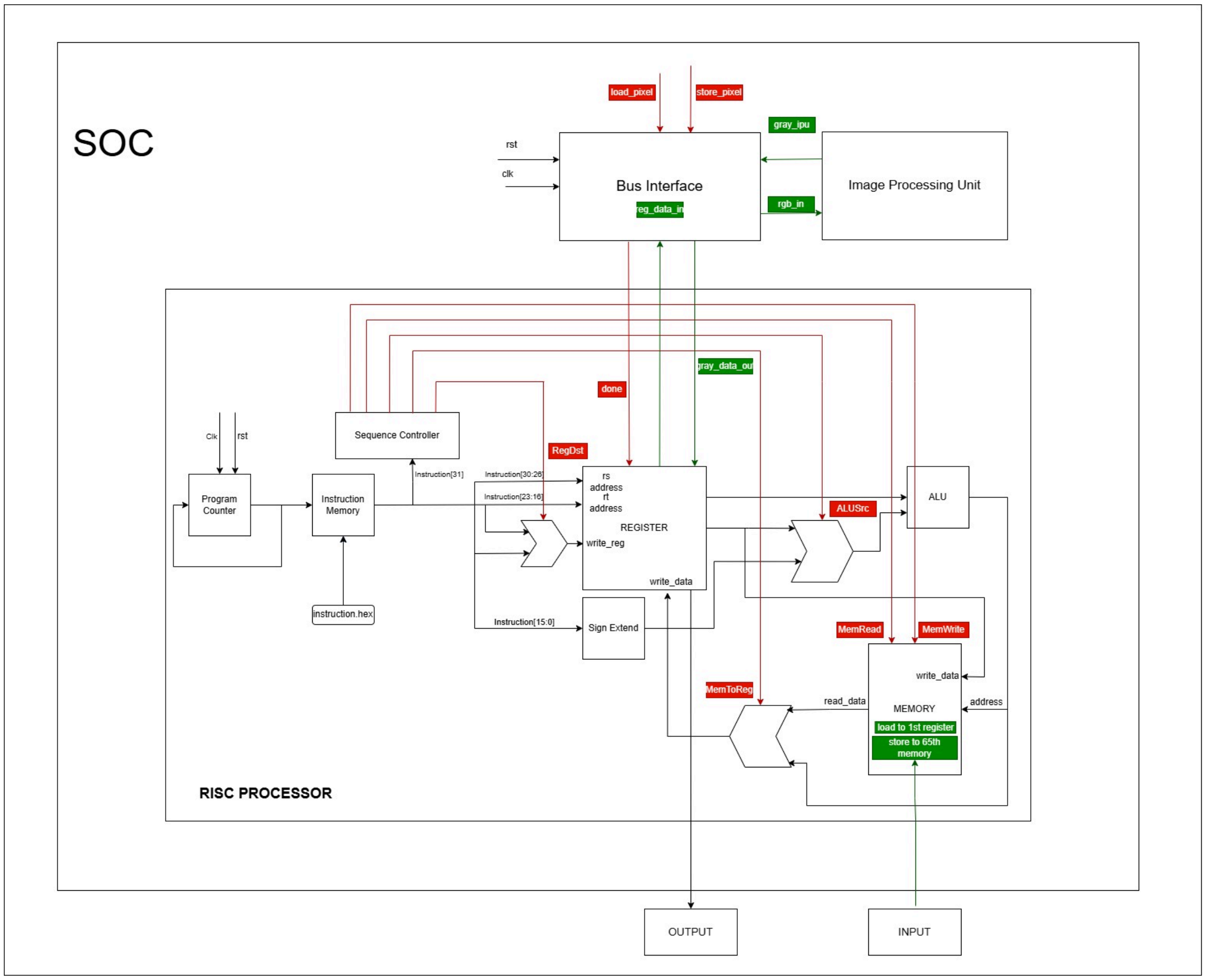
Opcode	rs	rt	Immidiate
1 bit	5 bit	8 bit	16 bit

Instruction_set [32] = Opcode [1] + rs [5] + rt [8] + Immidiate [16] + Dummy bit [2]

- Opcode = Instruction[31]
- rs = Instruction[30:26]
- rt = Instruction[23:16]
- Dummy Bit = Instruction[25:24] // These bits are not used for any purpose

2. System Architecture

The overall architecture integrates a **custom image processing unit** with a simple **RISC processor** and a **Bus Interface** to manage communication between components. Here's a high-level **overview and explanation** of the architecture:



2.1 Top-Level Structure

The system is organized into two main modules:

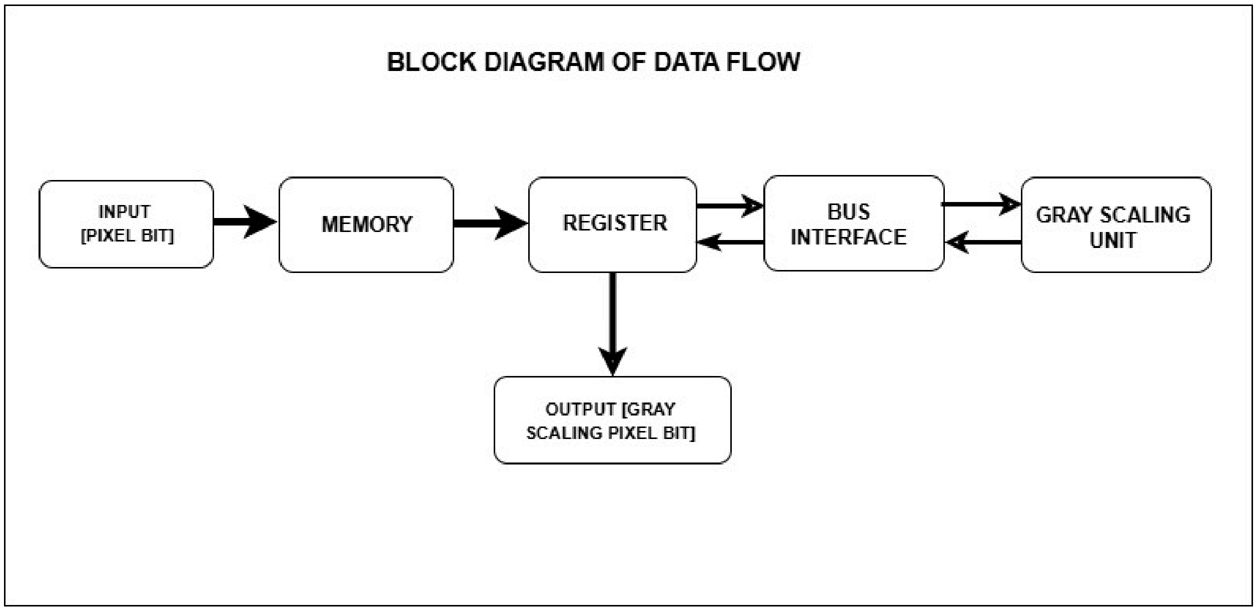
- `system_top`: The complete system integrating MIPS core and IPU
- `system_top_tb`: The testbench for verification

The design follows a Harvard architecture with separate instruction and data memories. The MIPS core communicates with the IPU through a dedicated bus interface while maintaining access to traditional memory and register files.

2.2 Data Flow

Image processing follows this sequence:

1. RGB pixel data is loaded from memory into registers
2. The processor initiates IPU operations through control signals
3. RGB data is transferred to the IPU via the bus interface
4. The IPU performs grayscale conversion
5. Results are stored back in registers/memory
6. From register/memory results[output gray scale pixel data] goes to next device



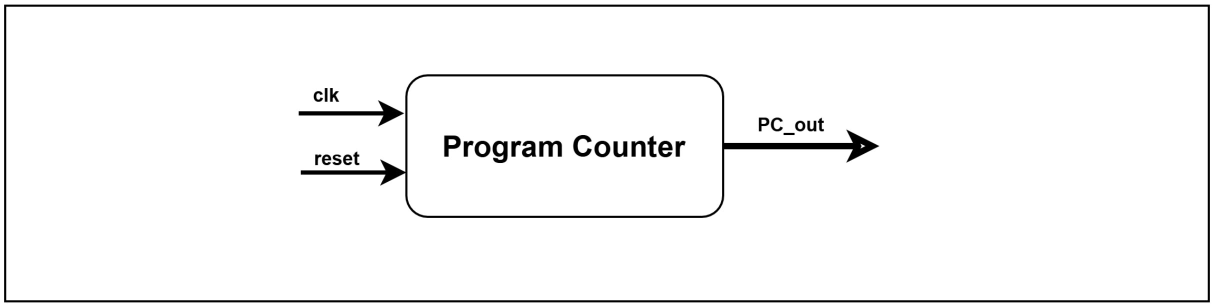
3. Detailed Component Analysis

3.1 MIPS Processor Core (`mips_top1`)

The MIPS implementation includes all fundamental components of a RISC processor:

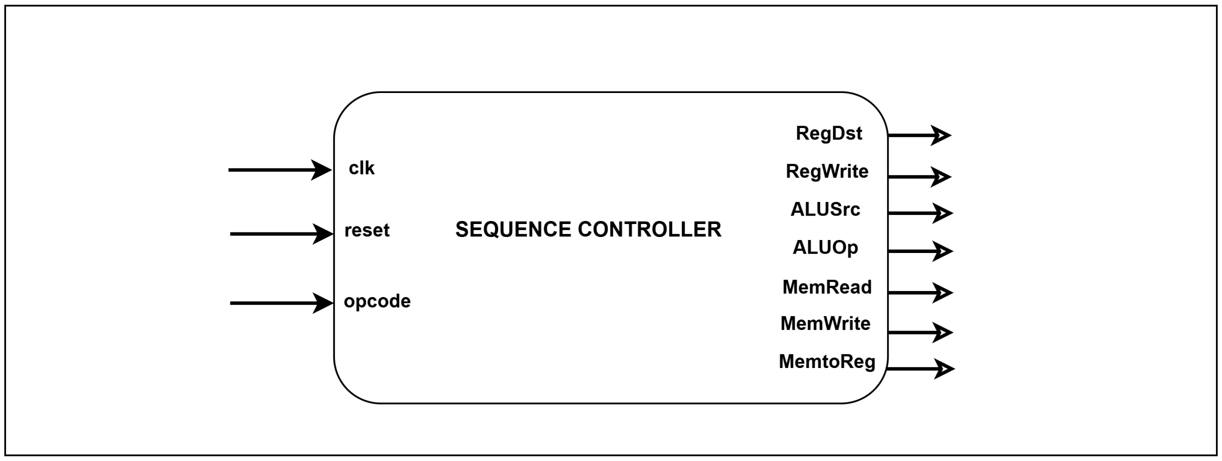
3.1.1 Program Counter

The PC is simplistic, incrementing by 1 each cycle. In a more robust implementation, it would handle branches and jumps.



3.1.2 Control Unit

The control unit implements a finite state machine (FSM) with 6 states (FETCH, DECODE, EXECUTE, MEM_READ, MEM_WRITE, WB). It generates all necessary control signals for the datapath.



In the context of a more modular and organized `mips_ipu.v` design, the sequence controller may be implemented as a dedicated module (e.g., `seq_control`) instantiated within the top-level MIPS module. This `seq_control` module acts as the central FSM (Finite State Machine), managing transitions between the stages of instruction execution and generating appropriate control signals for datapath coordination.

Key Functional Overview of `seq_control` Module:

This controller operates as a Moore state machine with the following states:

- **FETCH**: Begins instruction cycle by setting up a memory read.
- **DECODE**: Prepares control logic and evaluates opcode.
- **EXECUTE / ADDRESS CALCULATION**: Processes ALU operations or address calculation.
- **MEM_READ / MEM_WRITE**: Handles memory interaction.
- **WB (WRITE BACK)**: Writes result back to register.

Inputs:

- `clk`, `reset`: Timing and reset signals.
- `opcode`: Extracted from instruction; determines flow.

Outputs:

- Control lines such as `RegDst`, `ALUSrc`, `ALUOp`, `MemRead`, `MemWrite`, `MemtoReg`, and `RegWrite`.

Behavior Inside `mips_ipu.v`:

- This module is instantiated and connected to the opcode lines from instruction memory.
- All internal control lines that drive MUXes, ALU, and memory modules are sourced from this FSM.
- Transitions and control signals are updated each cycle based on the current state and opcode.

Illustrative Behavior:
For a
`lw` instruction:

1. In `FETCH`, instruction is loaded.
2. In `DECODE`, controller identifies `lw` opcode.
3. In `MEM_READ`, sets `MemRead = 1`, `ALUSrc = 1`, `RegWrite = 1`, `MemtoReg = 1`.
4. In `WB`, write enable signal is pulsed to store result.

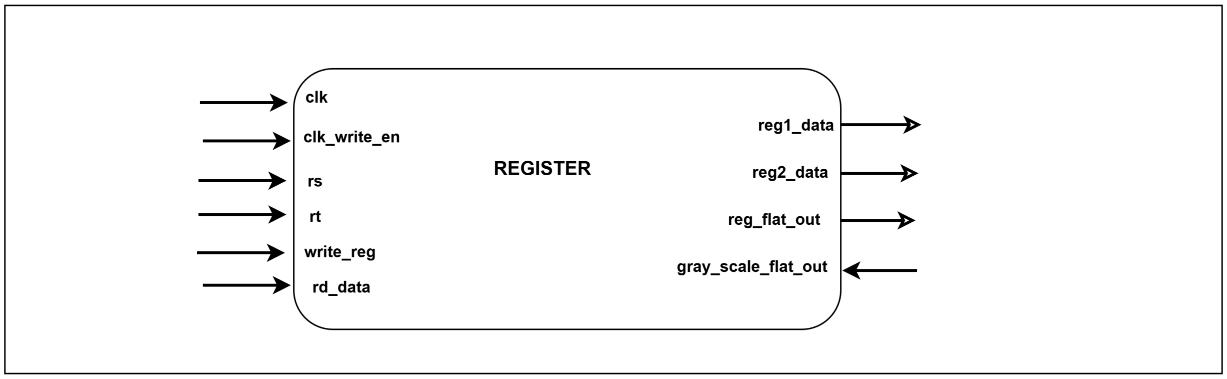
Thus, `seq_control` synchronizes the control signals to the processor pipeline and ensures precise sequencing of operations.

Within the `mips_ipu.v` top-level module, the sequence controller is embedded as combinational and sequential logic blocks that drive the core control signals for the datapath. It is tightly coupled with the instruction decoding logic and plays a vital role in transitioning the system through each instruction stage.

3.1.3 Register

The register file has several notable features:

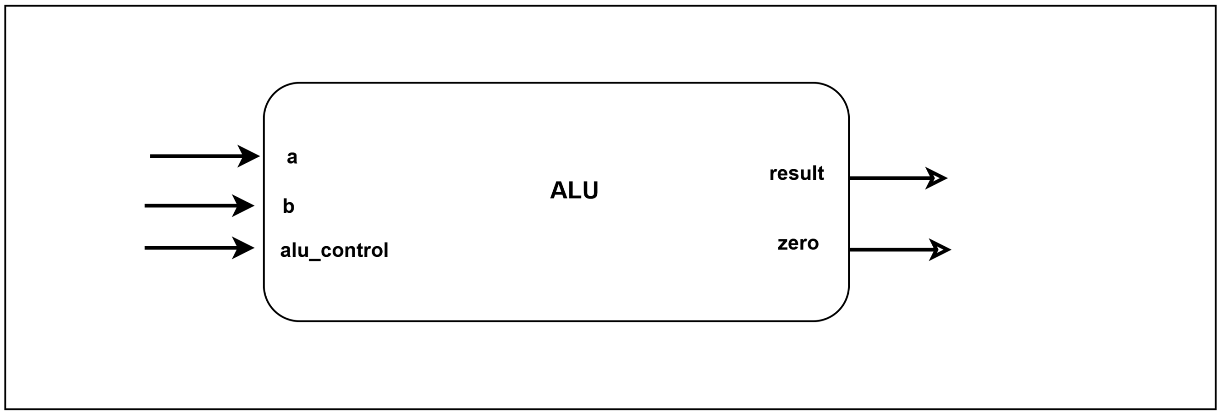
- 256 32-bit registers
- Special handling for grayscale data through **`grayscale_flat_out`**
- Dedicated outputs for monitoring specific registers
- Initialization from "registers.hex" file



3.1.4 ALU And Supported components

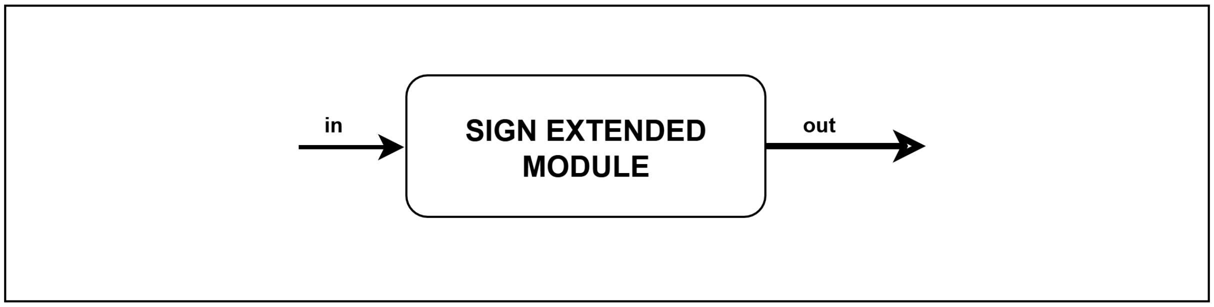
The ALU supports basic operations:

The ALU implements basic arithmetic and logical operations, though the comments don't match the actual operations (e.g., 3'b000 is addition but commented as AND).



3.1.5 Sign Extended Module

Takes the 16-bit immediate from I-type instructions and converts it to a 32-bit signed value. It ensures proper operand size for ALU operations.



3.1.6 Mux

The design includes three key multiplexers:

A. REGISTER WRITE MUX(reg_write_mux)

Purpose:

Selects the destination register (`rd` or `rt`) for R-type vs. I-type instructions.

B. ALU SOURCE MUX (alu_sourse_mux)

Purpose:

Chooses between a register value and a sign-extended immediate for the ALU's second operand.

C. MEM TO REG MUX(memtoreg)

Purpose:

Selects the data to write back to the register file (ALU result or memory load).

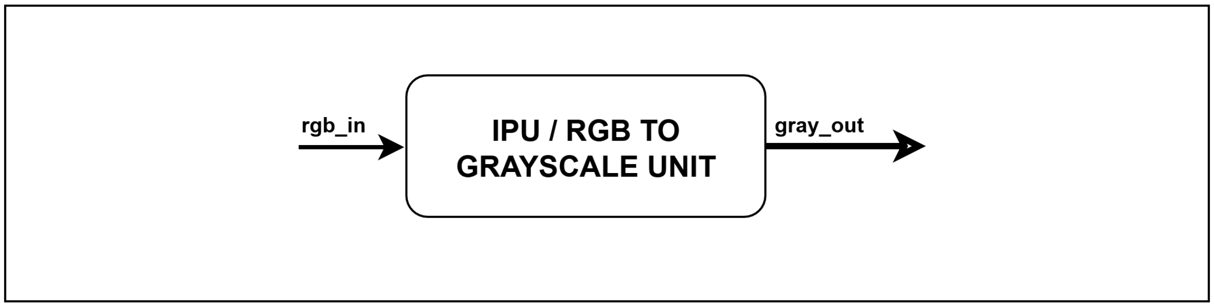
3.1.7 Image Processing Unit(IPU)

The IPU consists of two main components:

This implements the standard luminance formula:

Gray = (0.299R + 0.587G + 0.114B)

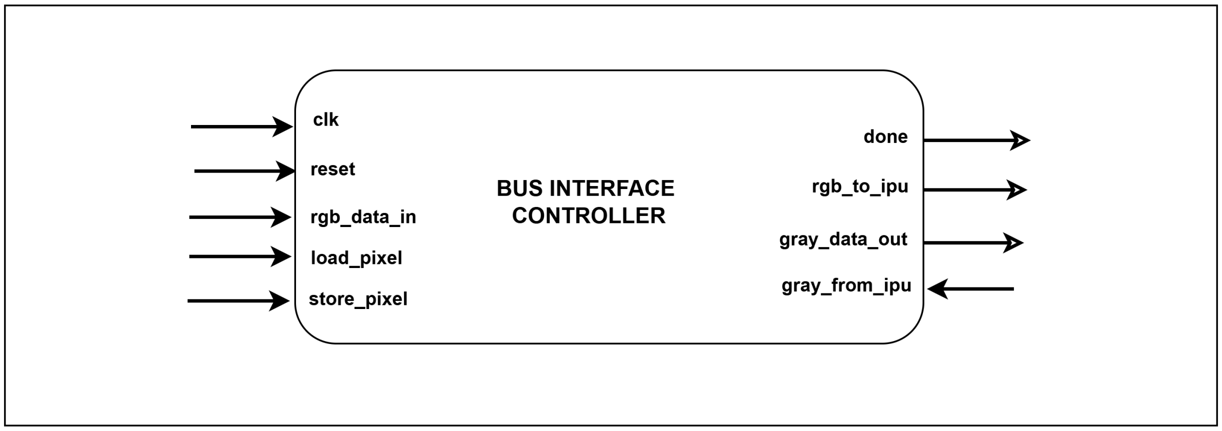
The fixed-point arithmetic uses integer multiplication followed by division to maintain precision.



3.1.8 Bus Interface

The bus interface manages communication between the MIPS core and IPU using a 4-state FSM (IDLE, LOAD, PROCESS, STORE). It handles:

- Data transfer synchronization
- Handshaking via done signal
- Pipeline control

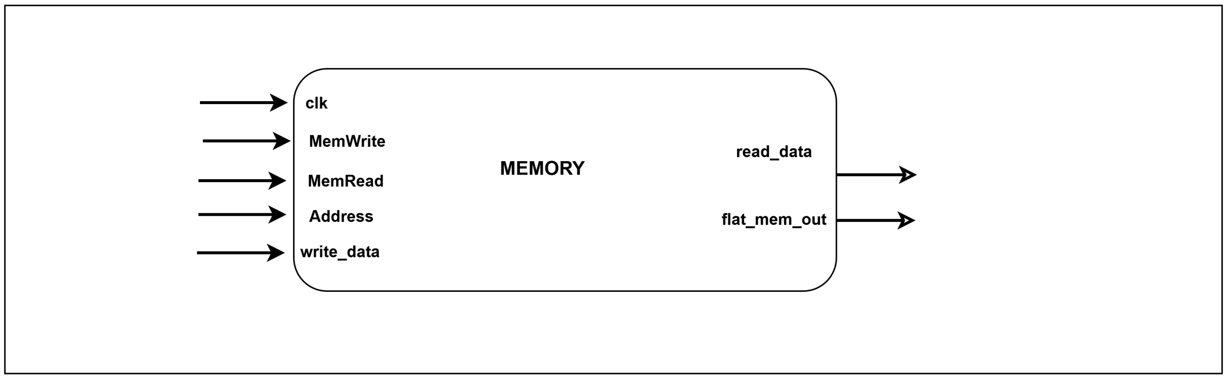


3.1.9 Memory

The data memory module provides both traditional memory access and specialized handling for image data:

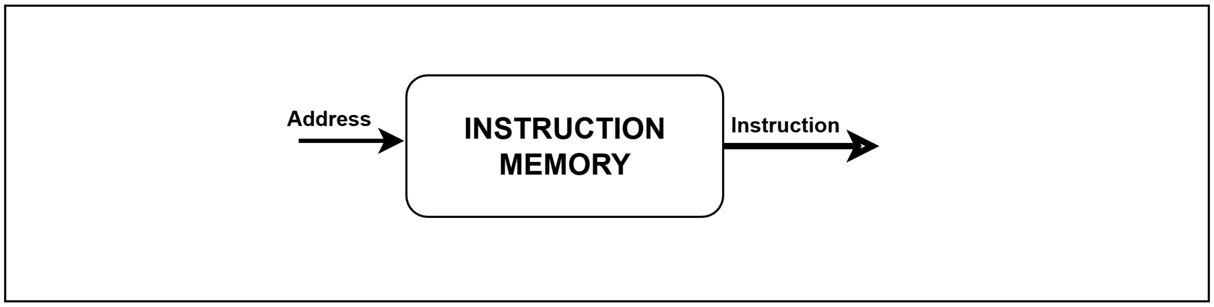
Key features:

- 2200-word memory space
- Initialization from "data_mem.hex"
- Special flattened output for efficient image data transfer



3.1.10 Instruction Memory

The instruction memory is initialized from a hex file, supporting up to 2200 32-bit instructions. The direct addressing without alignment handling suggests this is a simplified implementation.



3.2 System Integration (`system_top`)

The top module integrates all components and adds image processing control logic:

The image processing control uses a 4-cycle state machine to:

1. Initialize operations
2. Load pixel data
3. Process through IPU
4. Store results

Real-Life Applications of Grayscale Conversion in Image Processing

Grayscale conversion is one of the most fundamental yet powerful operations in image processing, with applications spanning nearly every field that deals with visual information. While color images contain rich data, converting them to grayscale often serves as a critical preprocessing step that enables more efficient analysis, reduces computational complexity, and highlights essential features. Here's an in-depth examination of real-world applications where grayscale conversion plays a vital role.

1. Medical Imaging and Diagnostics

- Grayscale imaging forms the backbone of medical radiography:
- Medical Imaging And Diagnostics
- MRI Interpretation

2. Industrial Machine Vision

- Automated Quality Control
- Barcode and OCR Systems

3. Surveillance and Security

- Facial Recognition System
- Motion Detection Algorithm

4. Document Processing and Archival

- Historical Document Preservation
- Receipt and Invoice Processing

5. Remote Sensing and Satellite Imagery

- Agricultural Monitoring
- Urban Planning

6. Automotive and Transportation

- Autonomous Vehicle Vision
- License Plate Recognition

7. Art Conservation and Restoration

- Painting Analysis
- Digital Restoration

8. Specialized Photography

- Astrophotography
- Forensic Photography

Future Directions

Emerging applications of grayscale processing include:

1. **Medical AI**: Combining grayscale radiomics with deep learning for early disease prediction
2. **Edge AI**: Ultra-low-power grayscale vision for IoT devices
3. **Quantum Imaging**: Grayscale pattern recognition for quantum radar systems
4. **Cultural Heritage**: Multi-spectral grayscale fusion for artifact analysis