Electronics and Computer Science

Faculty of Physical Sciences and Engineering

University of Southampton

Peter Alexander

11/04/2021

ELEC6234: Assignment 1 Report

Module Leader: Tomasz Kazmierski

A Report submitted for the Award of

Electronic Engineering with Computer Systems

# Introduction

# Technical Work

## Development Strategy

The code for this processor was written with the same module structure as the example processor given in the notes. These modules were developed in the order seen in Section 2.5.

The example modules given in the notes served as a basis for the modules in this processor. The basic development cycle was as follows:

1. Strip back the example module to the most basic features (e.g. ALU functions RA, RB, ADD).
2. Implement a self-checking module testbench to verify those features.
3. Implement new features to the module and testbench incrementally until the module is fully featured and verified.

The testbenches used assert statements and tasks to perform repeated tests with different parameters.

## Instruction Set and ALU Functions

Below are tables containing the instruction set implemented and the ALU functions. The only instruction added on top of those required by the spec is MOV. This was added to facilitate IO operations (see Section 2.3).

|  |  |  |
| --- | --- | --- |
| **Instruction** | **ALU Function** | **Description** |
| NOP | N/A | Do nothing |
| ADDI Rd Rs 9 | RADD | Rd = Rs + Imm |
| ADD Rd Rs | RADD | Rd = Rs + Rd |
| MULL Rd Rs | RMULL | Rd = lower half of Rs \* Rd (signed) |
| BEQ Rd Rs Imm | RSUB | PC = PC + Imm if Rd == Rs, else PC = PC + 1 |
| BNE Rd Rs Imm | RSUB | PC = PC + Imm if Rd != Rs, else PC = PC + 1 |
| MOV Rd Rs | RB | Rd = Rs |

Table 1: A table showing the instructions implemented

|  |  |
| --- | --- |
| **ALU Function** | **Description** |
| RA | alu\_result = a\_in |
| RB | alu\_result = b\_in |
| RADD | alu\_result = a\_in + b\_in |
| RSUB | alu\_result = a\_in – b\_in |
| RMULL | alu\_result = lower half of a\_in \* b\_in (signed) |

Table 2: A table showing the ALU functions implemented

## I/O

Each ALU input has been multiplexed with the register module output, SW[7:0], and {8{SW[8]}}. When Rd or Rs is set to 2 or 3, CPU-level logic sets the appropriate ALU input to SW[7:0] or {8{SW[8]}} respectively. The switch values can be accessed in two ways.

1. As Rd or Rs in a BEQ/BNE evaluation. This allows branching based on the value of SW[8] to be performed in a single cycle:

BEQ %0 %3 0 # If SW[8] == 0 Don’t increment PC

1. The MOV instruction is used to place the value of SW[7:0] or SW[8] into any other register.

MOV %4 %2 # Move SW[7:0] into register 4

The output of the ALU is connected to the LEDs so both the program result and the workings of the CPU can be observed.

## Final Block Diagram

Figure 1 shows the full system block diagram. The only block not contained within one of the 5 core modules is the Input Selector. This is implemented as two always\_comb blocks in the top-level module.

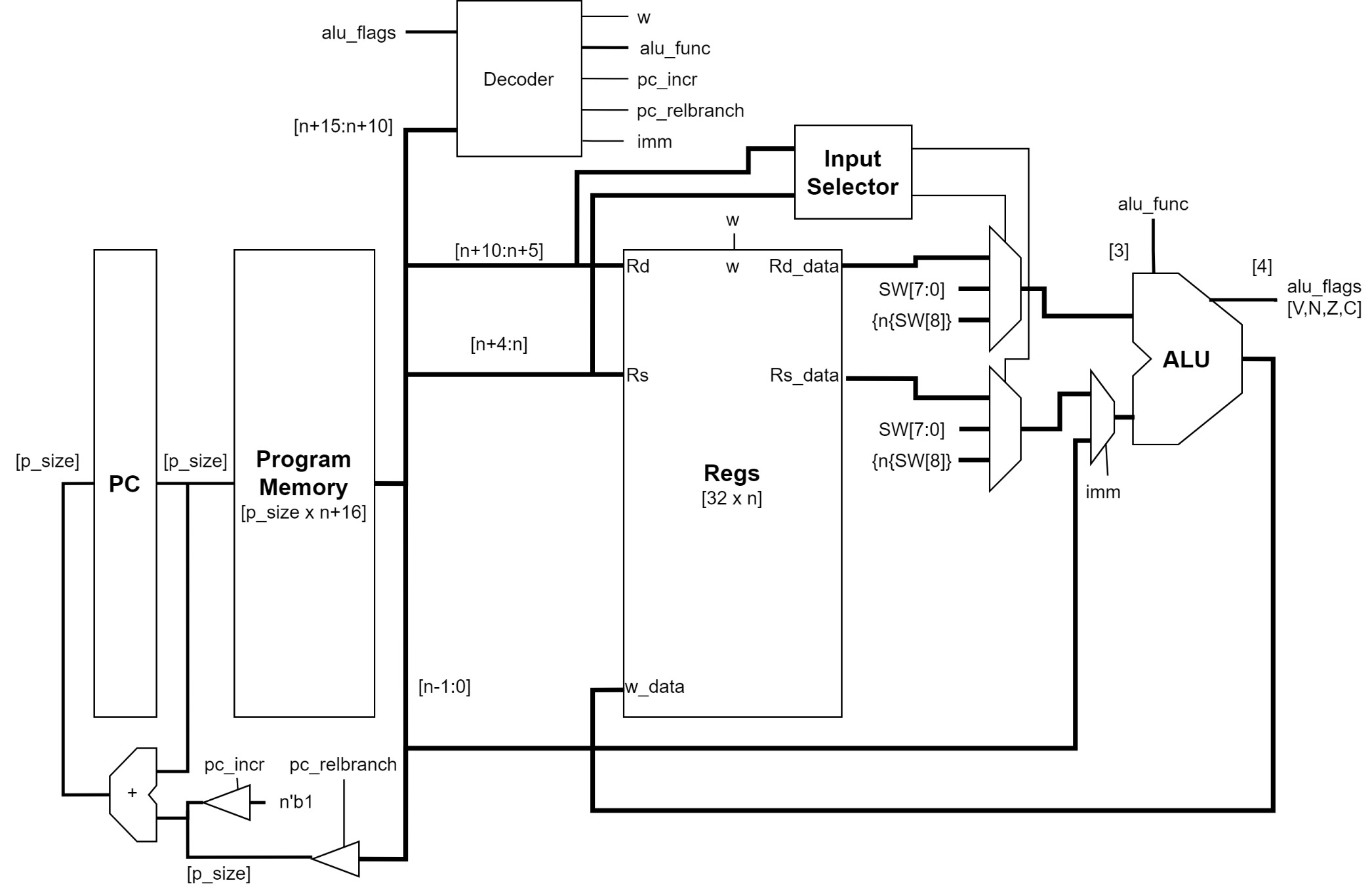


Figure 1: A figure showing the completed system block diagram

## Hardware Blocks

### ALU

### Registers

### PC

### Program Memory

### Decoder and CPU

## Software

### Assembler

### Application Program

## Results

# Conclusion