Northeastern University

Department of Electrical and Computer Engineering

EECE2323: **Digital Systems Design Lab**

Lecturer: **Dr. Emad Aboelela**

TAs:
Ke Chen
Linbin Chen

Lab # 4 - 5: **Adding Register File to ALU, Adding Data Memory to the Datapath**

Group # **14**:
Jin Hyeong Kim,
Timothy VanSlyke

Semester: Spring 2018
Date: March 9, 2018
Lab Session: Tuesday, 1:00PM
Lab Location: 9 Hayden Hall, Northeastern University,
Boston, MA 02115

Contents

1 Introduction

In these experiments, we will investigate the problem of introducing stateful components to our existing computer system. Specifically, the system will gain persistent storage media in the form of registers and data memory (main memory/random-access memory). In order to support imperitive computation, it is necessary that a computer system provide methods of storing, accessing, and modifying persistent state. Modern CPU architectures typically provide a finite set of registers which may be used to store machine words across the execution of multiple instructions. Additionally, and while not typically considered to be part of the CPU itself, data memory is used to provide a conceptually infinite (but finite in practice) storage medium to the computer system.

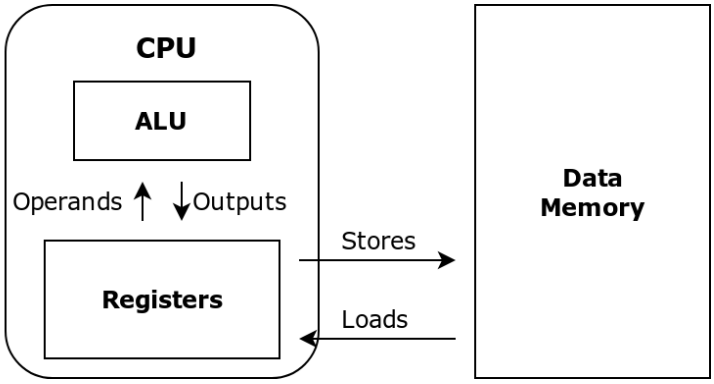

Figure 1: Simplified/minimalist diagram of a computer system with registers and data memory.

A minimalist memory ei

2 Design Approach

3 Results and Analysis

3.1 Design Simulation

3.2 Hardware Testing

4 Conclusions

Appendices

A reg_file.v

```
‘timescale 1ns / 1ps

module reg_file #( parameter WIDTH = 9, DEPTH = 4 )(
  input rst,
  input clk,
  input wr_en,
  input [1:0] rd0_addr,
  input [1:0] rd1_addr,
  input [1:0] wr_addr,
  input [8:0] wr_data,
  output wire [8:0] rd0_data,
  output wire [8:0] rd1_data
);
  reg [WIDTH-1:0] storage [0:DEPTH-1];

  always @(posedge clk) begin
    if(rst) begin
      storage[0] <= 0;
      storage[1] <= 0;
      storage[2] <= 0;
      storage[3] <= 0;
    end
    else if(wr_en)
      storage[wr_addr] <= wr_data;
  end
  assign rd0_data = storage[rd0_addr];
  assign rd1_data = storage[rd1_addr];
endmodule
```

Figure 2: `reg_file.v` - 4×9 register file implementation in verilog.

B eightbit_alu.v

```
‘timescale 1ns / 1ps

module eightbit_alu(
    input signed [7:0] a,
    input signed [7:0] b,
    input [2:0] s,
    output reg [7:0] f,
    output reg ovf,
    output reg take_branch
  );
  always @(a, b, s) begin
    // operations on ’f’
    case(s)
      0: f = a + b;
      1: f = ~b;
      2: f = a & b;
      3: f = a | b;
      4: f = a >>> 1;
      5: f = a << 1;
      6: f = 0;
      7: f = 0;
    endcase
    // overflow check for addition
    ovf = (s == 0) && (f[7] != a[7]) && (f[7] != b[7]);
    // ’take_branch’ special cases
    take_branch = ((s == 6) && (a == b)) || ((s == 7) && (a != b));
  end
endmodule
```

Figure 3: `eightbit_alu.v` - 8-bit ALU in implementation in verilog.