# Lab 5 Final Lab-Description

**Jiahao Chen tobychen@bu.edu**

**Xinran Zhang zhangxr@bu.edu**

In this final lab, we modified the components we created in Lab5 milestone and managed to complete the full single cycle datapath. It has the ability to complete add, and, or, cbz, sub, movk, stur, ldur and b(unconditional branch) operation.

## Program Counter: We changed the offset and PC mux we used in first milestone.

## We create the instruction memory module and ALUmux, the register mux and control unit based on the requirement. Also we modified the program counter to make it fit the requirement of the data path. In order to make the input of the register file much more obvious, we designed a special register mux that owns the control of RReg1, RReg2, WReg.

## Because of our ALU function, the functions that we’re managed separately to achieve are MOVK, ADD, AND, ORR, SUB, STUR, LDUR, BRANCH and CBZ. The instructions are provided below:

memory\_file[0] = 32'b11110010100\_0000000000000001\_00010; // movk x2, 1   
        memory\_file[1] = 32'b11110010100\_0000000000000111\_00001; // moxk x1, 7   
        memory\_file[2] = 32'b10001011000\_00001\_000000\_00010\_01010; // add x10, x2, x1           
        memory\_file[3] = 32'b10001010000\_01010\_000000\_00001\_00011; // and x3, x1, x10   
        memory\_file[4] = 32'b10001011000\_00010\_000000\_00001\_00100; // add x4, x1, x2   
        memory\_file[5] = 32'b10101010000\_00010\_000000\_00001\_00101; // orr x5, x1, x2   
        memory\_file[6] = 32'b11001011000\_00010\_000000\_00001\_00110; // sub x6, x1, x2   
        memory\_file[7] = 32'b11111000000\_00000000011\_00010\_00001; // stur x1, [x2, 3]   
        memory\_file[8] = 32'b11111000010\_00000000011\_00010\_00111; // ldur x7, [x2, 3]   
        memory\_file[9] = 32'b00010100000\_000000000000000000010; // b 2   
        memory\_file[11] = 32'b11110010100\_0000000000000000\_01000; // movk x8, 0   
        memory\_file[12] = 32'b10110100000\_000000000000011\_00111; // cbz x7, 3   
        memory\_file[13] = 32'b10110100000\_000000000000010\_01000; // cbz x8, 2

In order to achieve MOVK, we designed a special module to implement the MOVK. The module code is provided as follows:

module movk(instruction, RFRD2, WriteReg, MovK, MemMov, rst);

input [31:0] instruction;

input [63:0] RFRD2, MemMov;

input rst;

input MovK;

output reg [63:0] WriteReg;

always@(posedge rst, MovK)

begin

if (MovK==1)

WriteReg<={RFRD2[63:16], instruction[20:5]};

else if (MovK==0)

WriteReg<=MemMov;

end

endmodule

Since the whole datapath have no output, we set parameter tests to examine whether we got the right value of each component. Such as PCtest, RFWDtest, RFRD1test, etc. we can check the value from the waveform.

The code of our top module is listed below,

The waveform is provided as follows